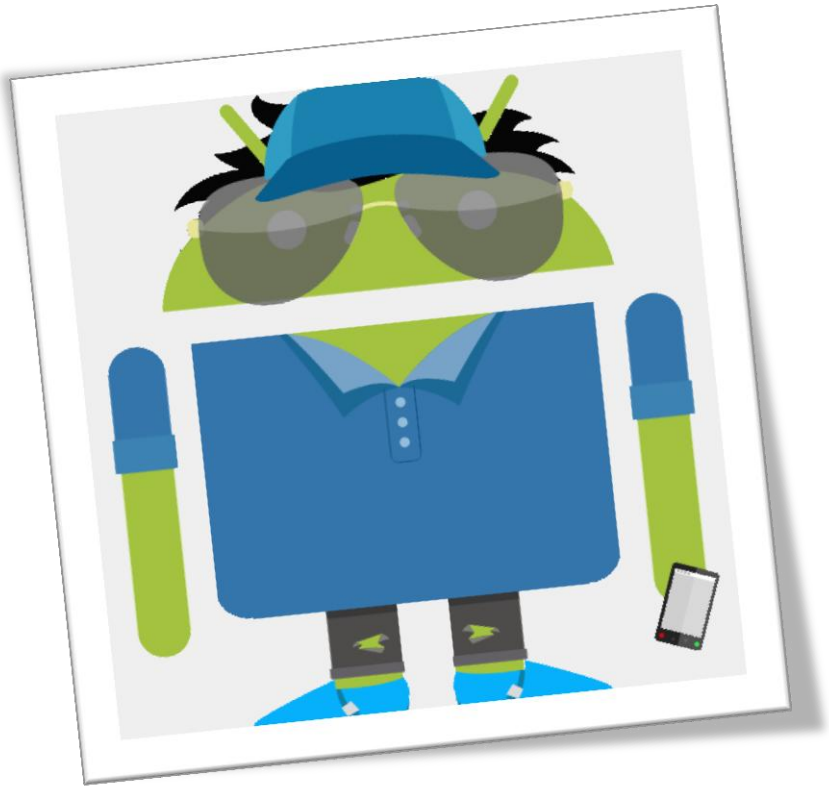
Сборник уроков. Часть I.



Google Android

… это несложно

О книге

Когда я собрался погрузиться в разработку Android, я не смог найти в сети толковых материалов для новичков на русском языке. Думаю, этот факт оттолкнул многих желающих начать изучение. Поэтому я решил этим желающим помочь.

Я создал сайт startandroid.ru и начал излагать там те темы, которые сам уже изучил. Материал на сайте представлен в виде небольших последовательных уроков. Каждый урок сопровожден примером, который раскрывает тему, и наглядными скриншотами.

От читателей периодически поступали просьбы сделать уроки доступными офлайн. Поэтому я скомпоновал первые 100 уроков в этот сборник.

О чем книга

Книга научит вас разрабатывать приложения для Android. Начнем с установки среды разработки и запуска первого приложения. Научимся компоновать экраны, обрабатывать нажатия и создавать свое меню. После этого вы уже способны сами написать несложное приложение.

Ну а далее пойдут базисные темы: несколько экранов, диалоги, вкладки, списки, настройки, работа с данными, фоновые задачи, уведомления.

Все темы рассмотрены в разрезе Android 2.X. Новшества третьей версии не вошли в этот сборник.

Для кого книга

Книга рассчитана на абсолютных новичков в Android, но знакомых с Java. Также понадобятся знания XML и SQL.

После прочтения вы вплотную приблизитесь к среднему уровню разработчика. С этим запасом знаний вы сами сможете свободно продолжать изучение с помощью других источников, и расти дальше.

От автора

Желаю успехов в изучении! Если возникнут затруднения или что-то будет непонятно - добро пожаловать на форум forum.startandroid.ru. Будем разбираться: что, как и почему не работает.

Дмитрий Виноградов

Урок 1. Введение.

Добрый день!

Это урок-введение. Здесь мы пока ничего кодить не будем, я распишу причины, которые побудили меня создать этот сайт.

Я начал свое знакомство с разработкой под Android с этих упражнений с официального сайта. Половину того, чего делал - не понимал. Но какие-то знания оттуда вынес и немного почитал теории на том же ресурсе. На этом мое знакомство с Android завершилось) Я не знал куда двигаться дальше. Про книги я почему-то не подумал, а зря ...

Позже я наткнулся на статью «Five of the best Android development books». Интерес снова появился, стало понятно куда двигаться, я начал читать эти книги. Понял не все, но уже гораздо больше чем с первых примеров. Но если читать такие книги полностью с нуля, то многое будет неочевидно и непонятно.

Поэтому я хочу попытаться создать уроки по Android для начинающих, чтобы у читателя за спиной не оставалось непонятных ему тем. Я постараюсь максимально подробно все объяснять и делать различные примеры для большей наглядности. С каждым новым уроком буду вводить и использовать новые понятия и объекты, и использовать объекты из прошлых уроков для закрепления.

Каждый урок я стараюсь делать максимально независимым и обособленным, чтобы можно было зайти, посмотреть нужную тему и не просматривать кучу лишнего. Примеры стараюсь подбирать наиболее четко отображающие тему урока.

В отличие от некоторых авторов я не собираюсь научить вас программированию за "24 часа" или за "30 уроков". Мы все понимаем, что это невозможно) Я не знаю, сколько у меня получится уроков. Думаю, что хватит около сотни, чтобы дать весь базис достаточно подробно. А потом еще сотня уйдет на различные продвинутые фишки. В общем тем, кто начинает изучать, скорее всего не придется много бегать по другим сайтам (кроме официального хелпа) за дополнительными знаниями. Здесь вы найдете много полезной, последовательной и изложенной простым языком информации.

Надо понимать, что мои уроки это не всегда руководство "как именно надо делать". Я могу чем-то пренебречь и что-то упустить, чтобы показать тему урока и не давать лишний материал. Поэтому прошу не считать все изложенное в уроках единственно правильным способом реализации.

Если у вас возникают проблемы с Android, то у сайта есть замечательный форум, который всегда рад помочь новичкам разобраться даже в самых простых вопросах. Правда, он защищен от спамеров контрольным вопросом и ответить на него, чтобы зарегиться, вы сможете только прочитав первые пять уроков. Это вынужденная защита. Зато, с момента введения этого вопроса, на форум не проник ни один спамер!

На данный момент уроки охватывают темы:

- создание экрана (в редакторе и программно)
- обработчики нажатия
- логи и всплывающие сообщение
- обычное меню, контекстное меню
- анимация View-компонентов
- создание и вызов Activity (+ возврат результата)
- Activity Lifecycle (состояния Activity)
- Intent, Intent Filter
- хранение данных (Preferences, SQLite)
- список и адаптеры
- диалоги
- Parcel, Parcelable
- Preferences при хранении настроек приложения
- работа с файлами
- Tab вкладки

- парсинг XML
- асинхронность (Handler, AsyncTask)
- сервисы

Я продолжаю читать книги и уроки будут появляться, пока я сам буду развиваться. В итоге, я думаю, мы придем к тому, что станем достаточно продвинутыми разработчиками, востребованными на рынке. В общем, как говорит один наш лохматый друг с ОРТ - "не переключайтесь") Будет интересно!

На следующем уроке мы установим и настроим среду разработки Android-приложений.

P.S.

Разработка ведется на Java. Также могут понадобиться знания SQL, XML и прочих смежных технологий. Считается, что вы знакомы с ними. Если нет, то что-нибудь базисное по Java надо будет прочесть.

Гугл периодически выпускает обновления Android и для среды разработки. Поэтому вполне возможно, что содержимое урока немного устарело и реальная картинка отличается от скринов. Если это отличие кардинально или примеры не работают, пишите об этом на форуме в ветке урока. Будем актуализировать. Если же отличие только в цвете фона приложения или размере шрифта, то это, конечно, не критично и на посыл урока не влияет.

Урок 2. Установка и настройка среды разработки Eclipse и SDK Tools

Для того, чтобы писать программы - нужна среда разработки. Google рекомендует использовать для этих целей Eclipse с плагином Android Development Tools (ADT). В этом уроке мы подробно рассмотрим, как установить и настроить эту среду разработки.

Я буду описывать установку применимо к операционной системе Windows 7.
Использую этот мануал по установке - http://developer.android.com/sdk/installing.html
Системные требования - http://developer.android.com/sdk/requirements.html

Перед прочтением рекомендую посмотреть:

SDK - http://ru.wikipedia.org/wiki/SDK

В этой статье много скриншотов и инструкций. Учитывайте, что версии ПО постоянно меняются и у вас все может выглядеть по-другому и версии могут быть другими.
О том, как обновить компоненты, есть отдельная статья.

1. Java SDK - JDK

Т.к. разработка приложений ведется на Java, нам нужно скачать и установить соответствующее SDK, называемое еще JDK (если, конечно, оно уже не установлено).

Скачать можно здесь. Недавно появилась седьмая версия. Ничего не могу сказать о ней, но есть мнение, что сырая, потому рекомендую скачать проверенную шестую версию В разделе **Java Platform, Standard Edition** жмете **JDK Download**, ставите галку, что принимаете лицензионное соглашение и скачиваете файл соответственно Вашей операционной системе. С установкой проблем возникнуть не должно. После этого желательно перезагрузиться.

2. Android SDK

Android SDK включает в себя инструменты, необходимые для разработки Android-приложений. Содержимое можно посмотреть здесь, а скачать здесь. Рекомендуется скачивать EXE-шник, но я предлагаю скачать ZIP-версию и самим распаковать в какой-нить удобный для вас каталог.

Учтите, что это должен быть каталог "на векА". И лучше его не перемещать никуда, иначе придется перенастраивать среду разработки. Предлагаю где-нибудь создать каталог Android. Крайне желательно, чтобы путь к нему был коротким. Идеально - <имя диска>:\android (у меня это будет f:\android). Для себя запомним этот каталог под псевдонимом **<Android>**. И в него распакуем наш архив SDK, получим **<Android>\android-sdk-windows**.

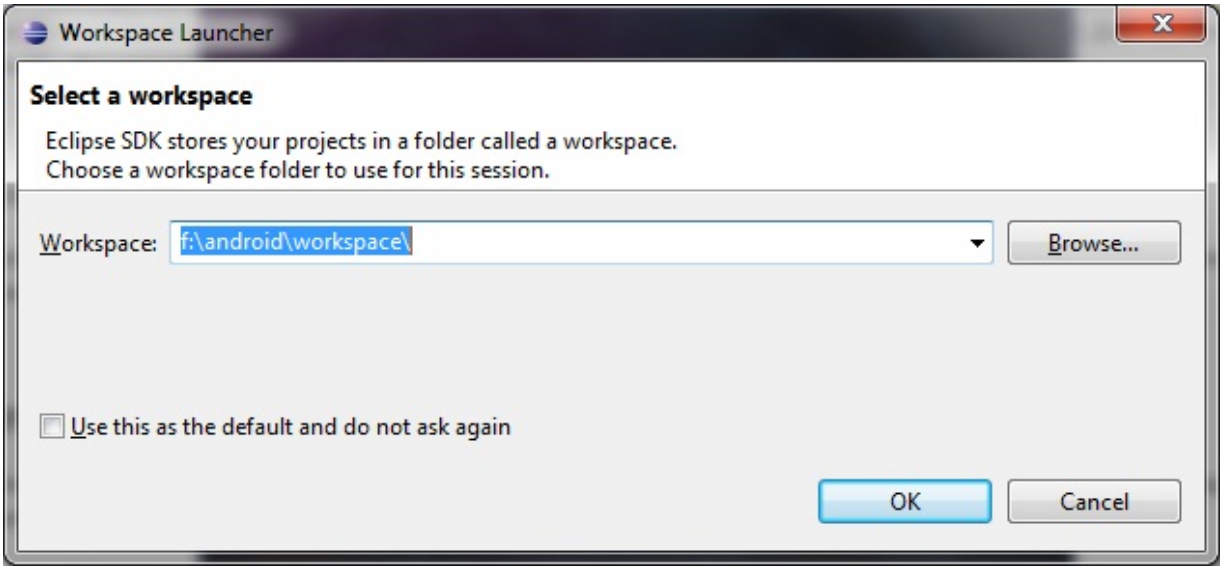
3. Eclipse

Симпатичная и удобная среда разработки, где мы и будем творить и созидать). Ее мы будем брать здесь. Гугл рекомендует нам версию **Eclipse Classic**. Согласимся с ним и скачаем именно эту версию. Распаковываем архив в <Android>, получаем **<Android>\eclipse**.

4.ADT

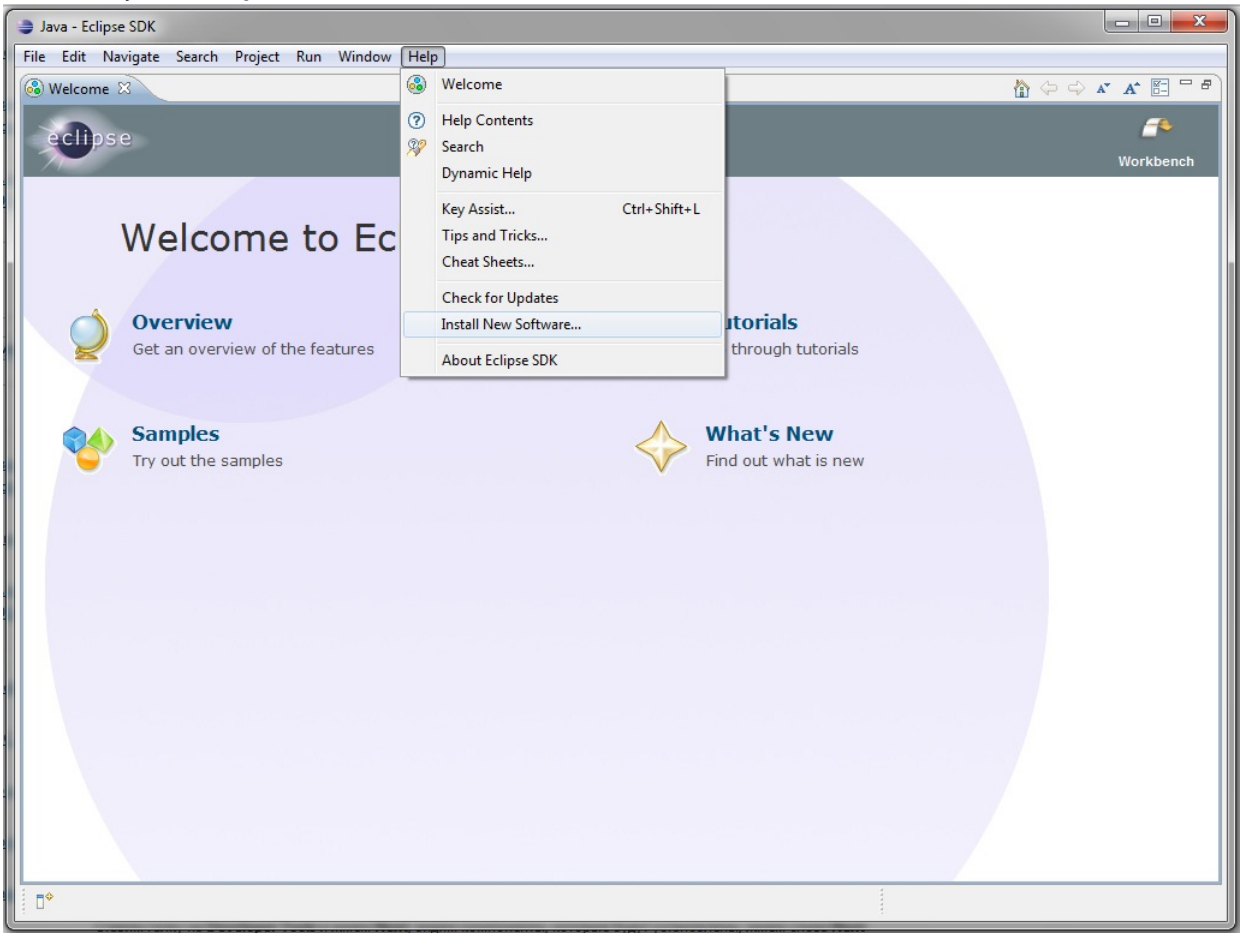
По умолчанию Eclipse не особо годится для разработки Android-приложений. ADT - плагин, который настраивает среду разработки для использования Android SDK и добавляет возможность удобной разработки.

Запускаем Eclipse (<Android>\eclipse\eclipse.exe). При первом запуске он попросит указать ему рабочий каталог, где он будет хранить информацию о проектах. Предлагаю опять же не ходить далеко и создать каталог **<Android>\workspace** и указать этот каталог.

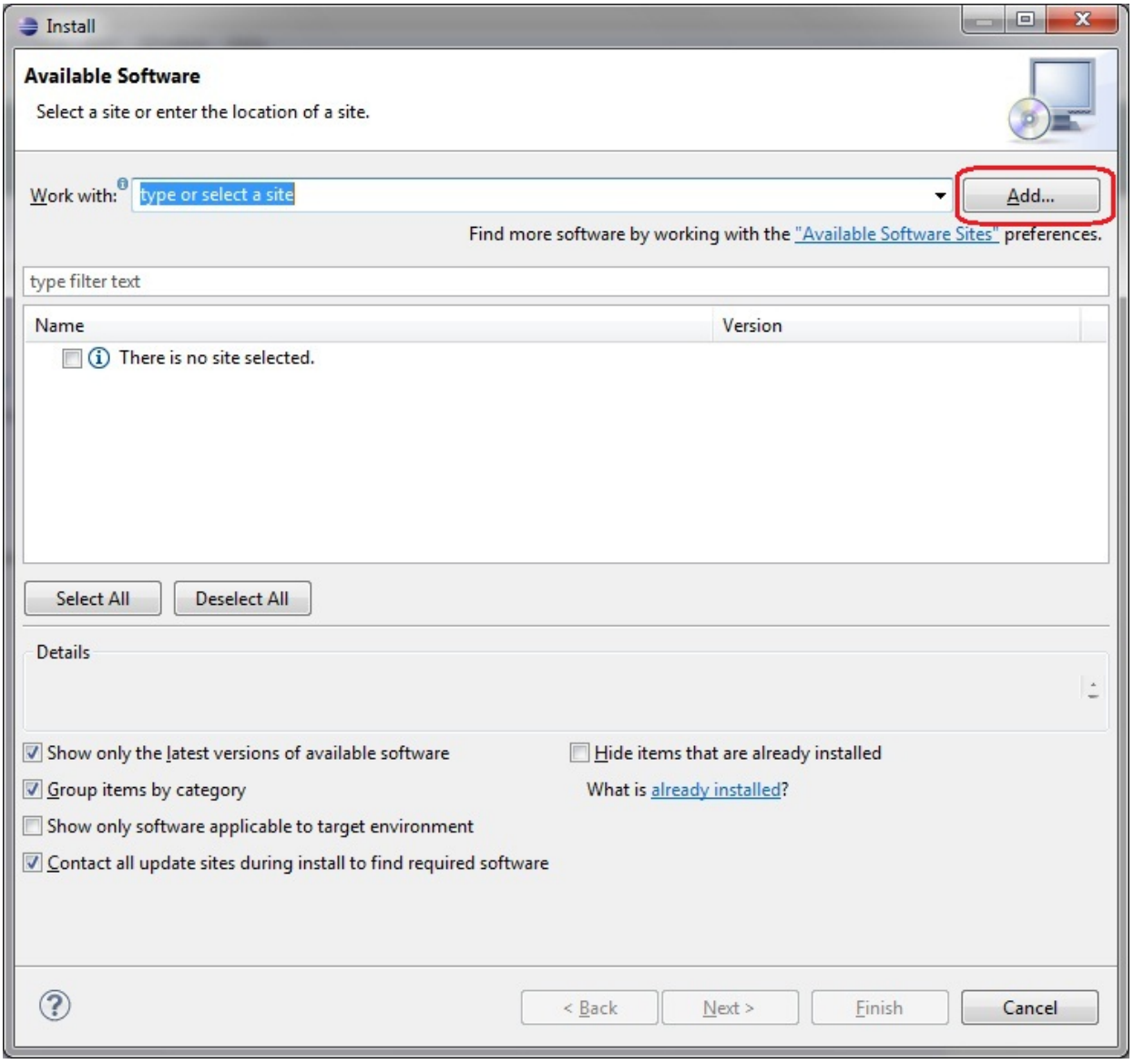


Итак Eclipse запущен. Скачаем ADT плагин.

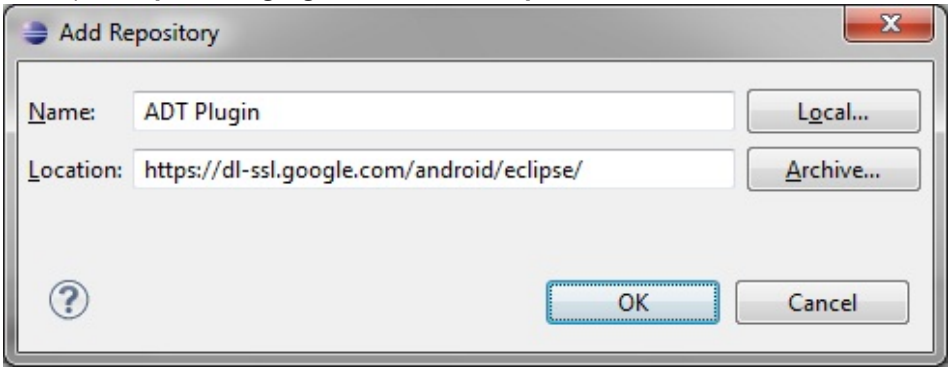
В меню запускаем **Help > Install New Software**...



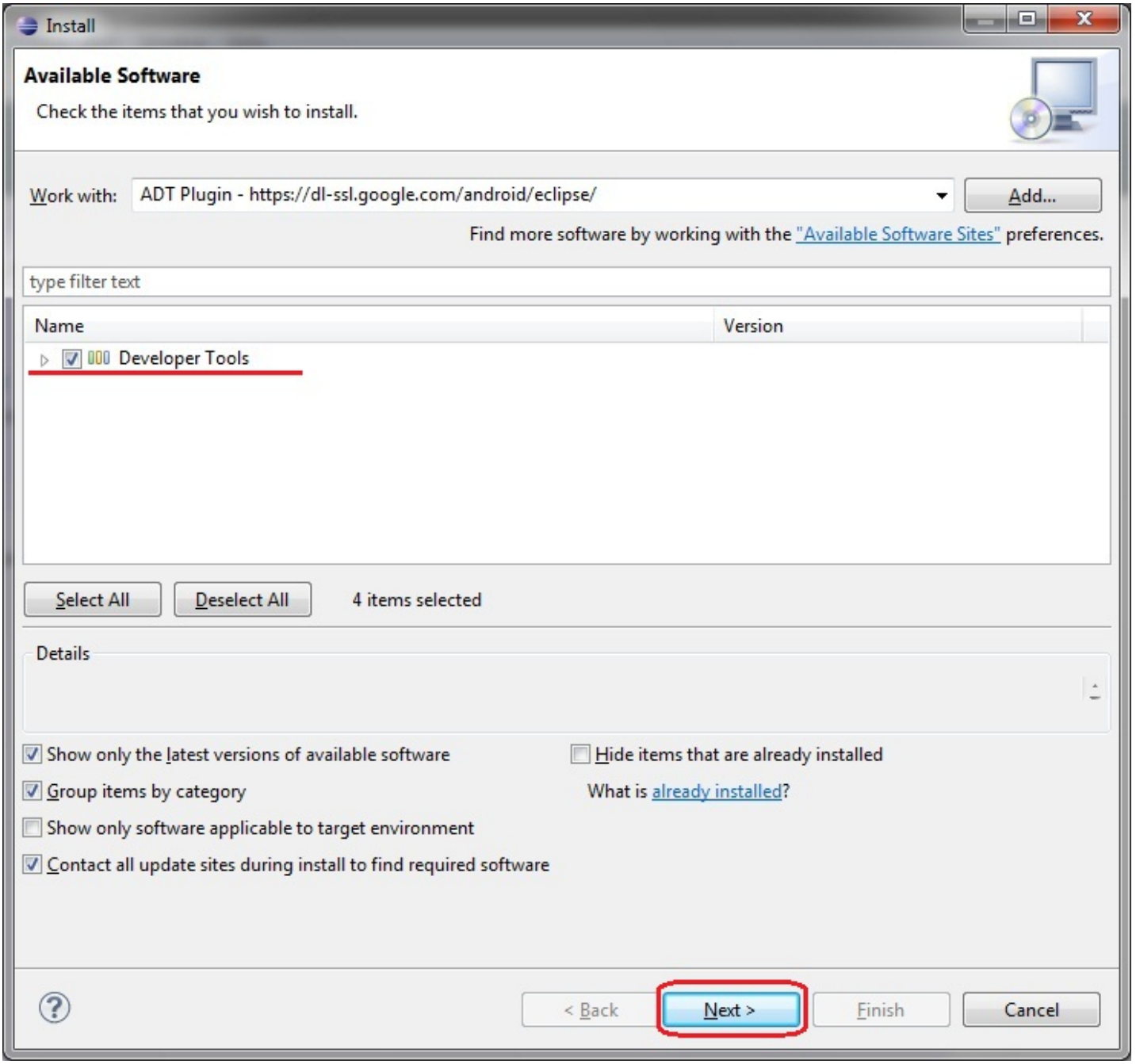
Жмем кнопку **Add** в правом верхнем углу



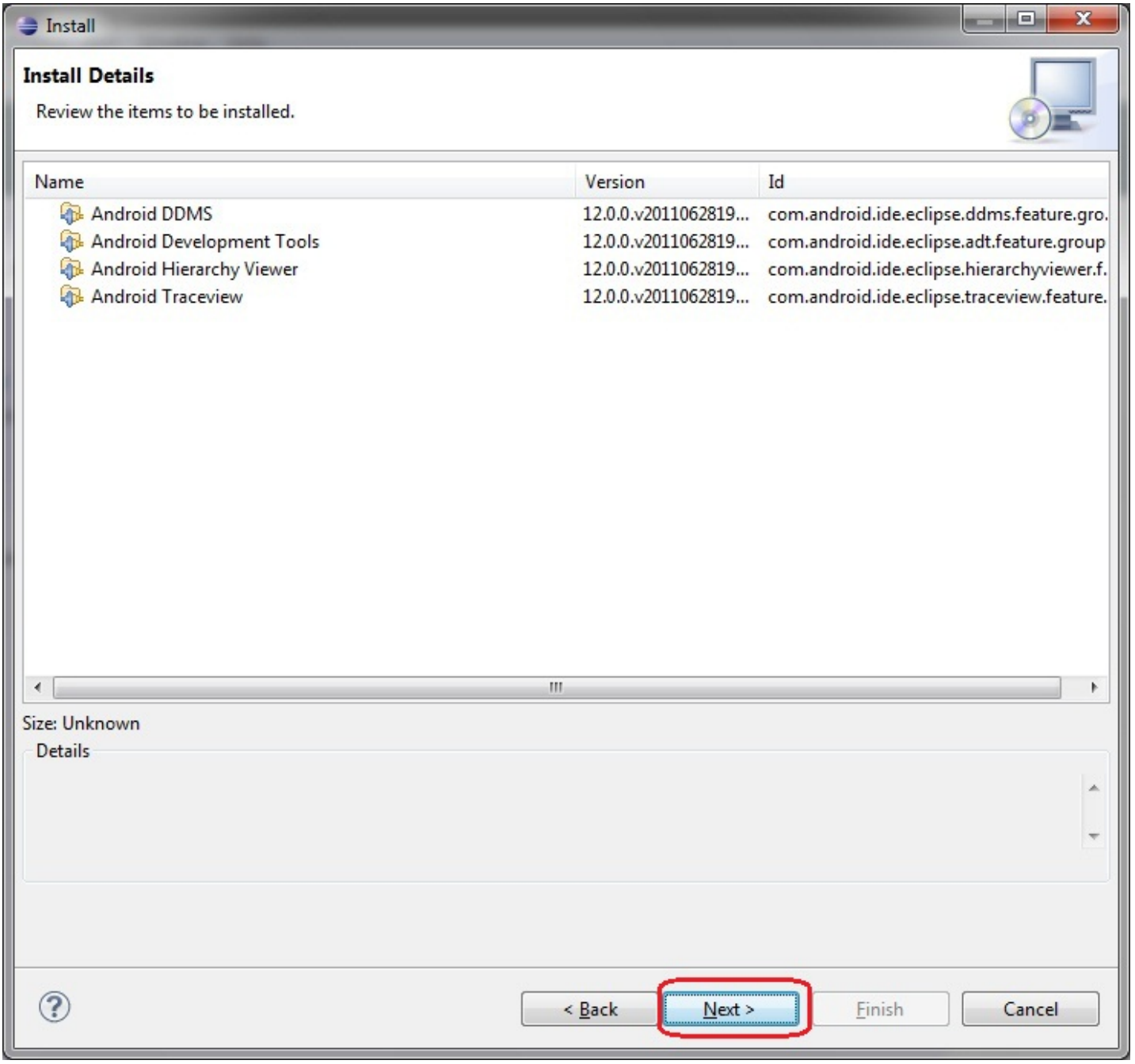
Вводим "**ADT Plugin**" в поле **Name**
URL адрес: **https://dl-ssl.google.com/android/eclipse/** - в поле **Location**



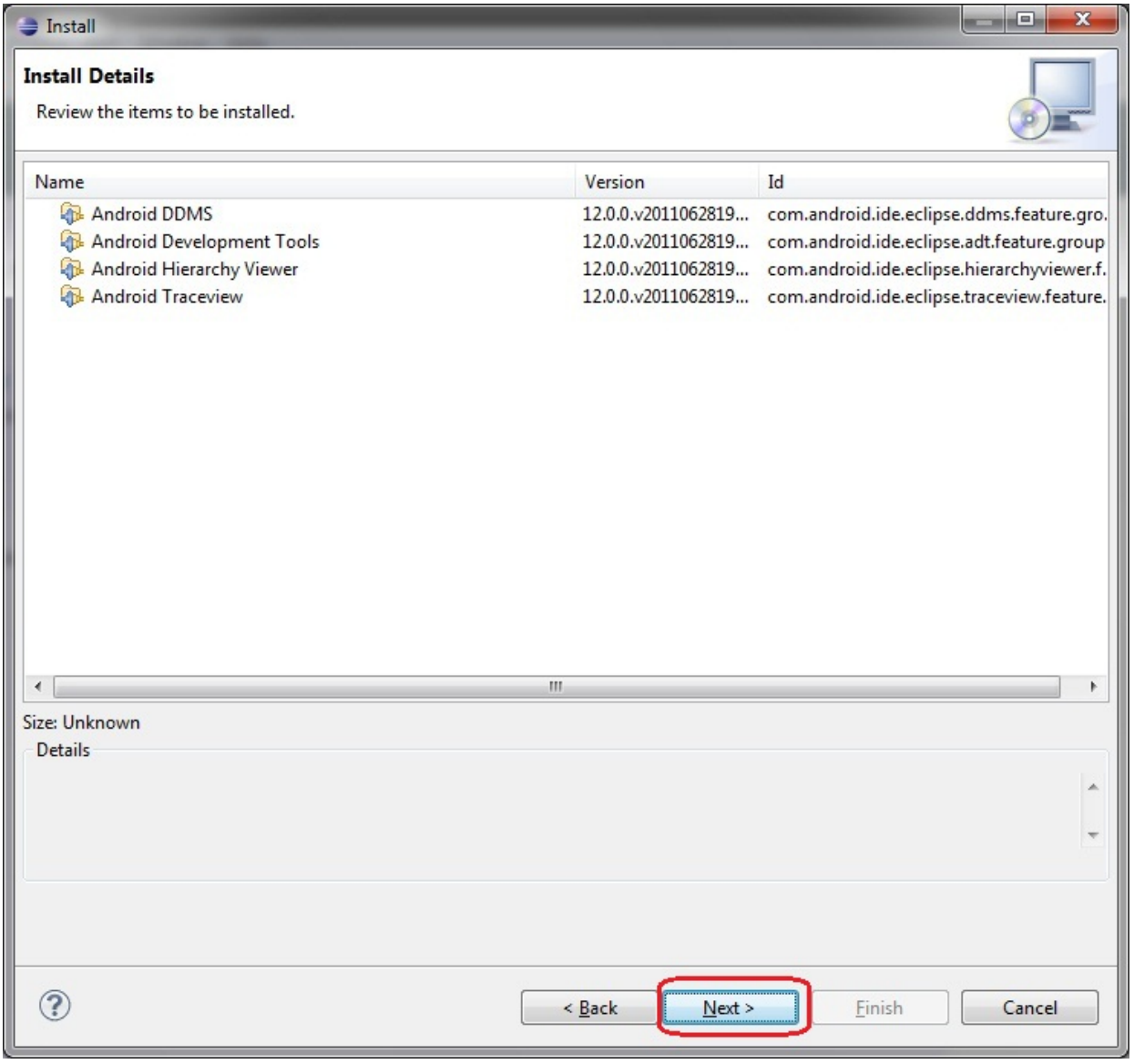
Жмем **OK** и ждем, пока появится **Developer Tools** (Если возникают проблемы, используйте http вместо https)
Ставим галку на **Developer Tools** и жмем **Next**



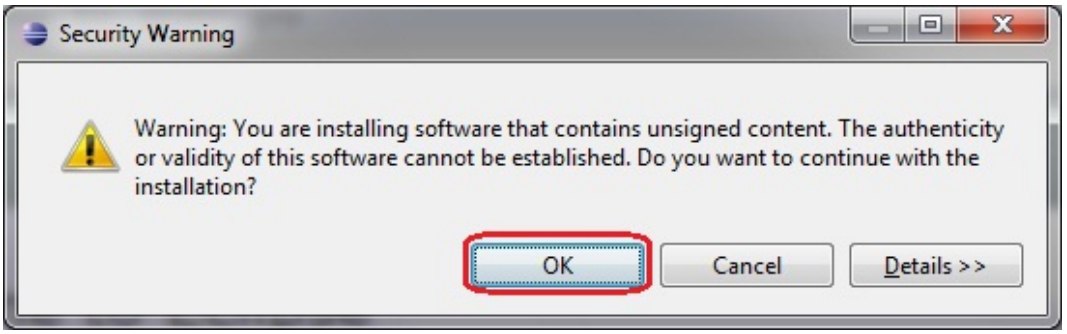
Видим компоненты, которые будут установлены, жмем снова **Next**



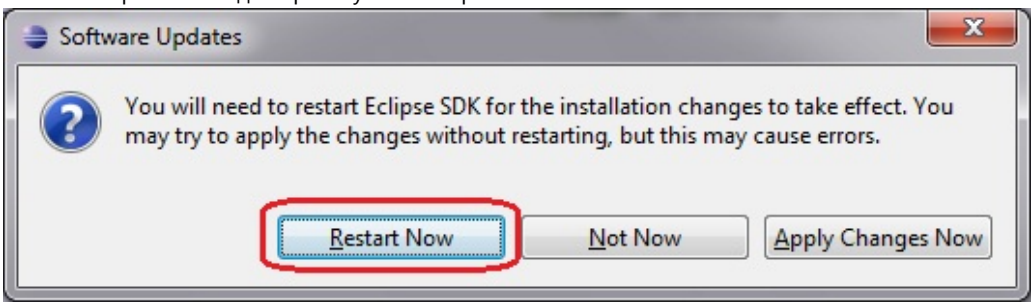
Читаем и принимаем лицензионное соглашение и жмем **Finish**



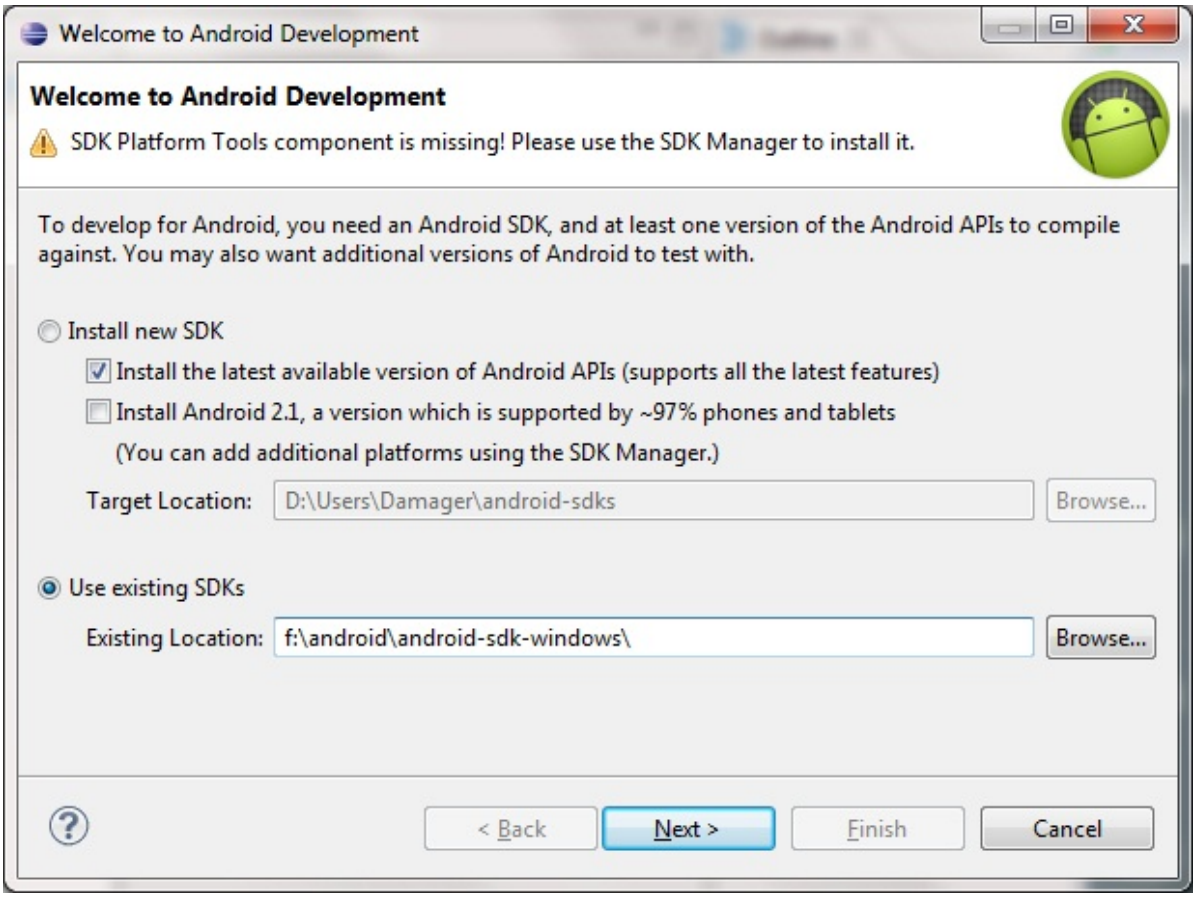
Начинается закачка компонентов. Если выскочит **Security warning** о том, что **the authenticity or validity of the software can't be established**, жмите **OK**.



У меня процесс занял около минуты.
После завершения надо перезапустить Eclipse - **Restart Now**



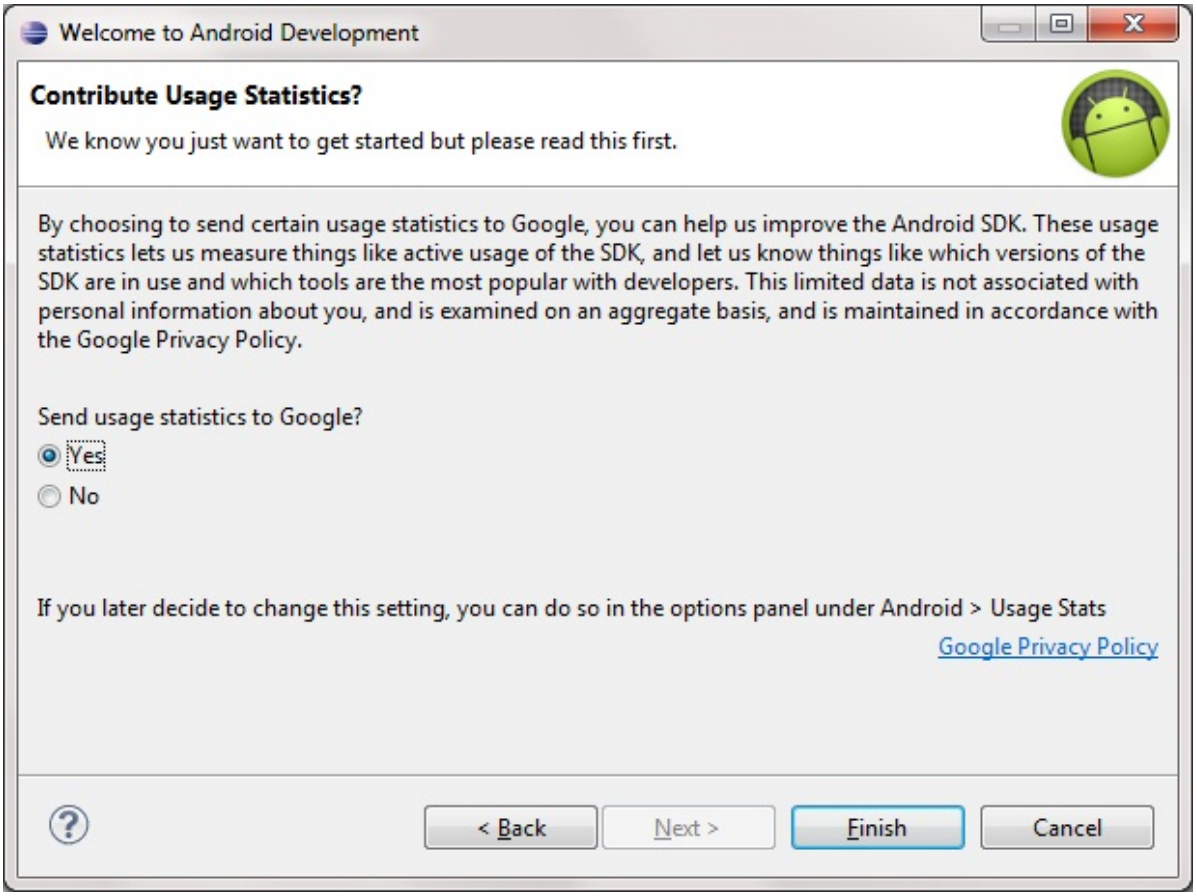
После перезапуска Eclipse выдаст такой диалог:



Первый пункт нужен для тех, кто по каким то причинам не скачал и не распаковал SDK на втором шаге данной инструкции. Eclipse сможет сделать это сам. И дополнительно сразу же скачает последнюю Android-платформу. Также он предлагает скачать платформу версии 2.1, как наиболее поддерживаемую кучей устройств.

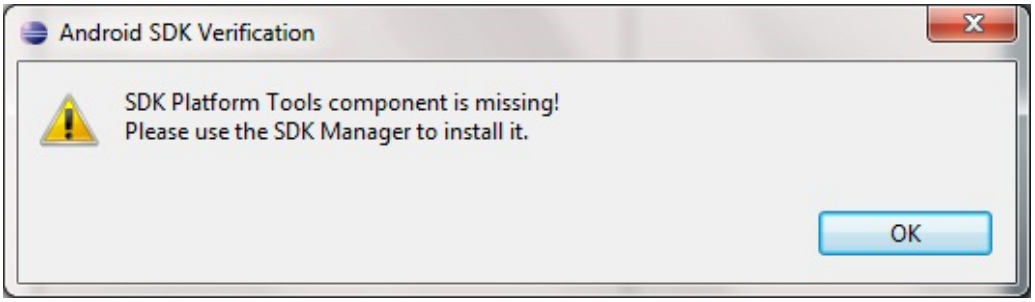
Нам интересен второй пункт - он позволяет указать, куда мы распаковали SDK в шаге 2. У нас это - **<Android>\android-sdk-windows**. Жмем Next.

И в появившемся окне выбираете, отправлять статистику в гугл или не отправлять. На работу это никак не повлияет.



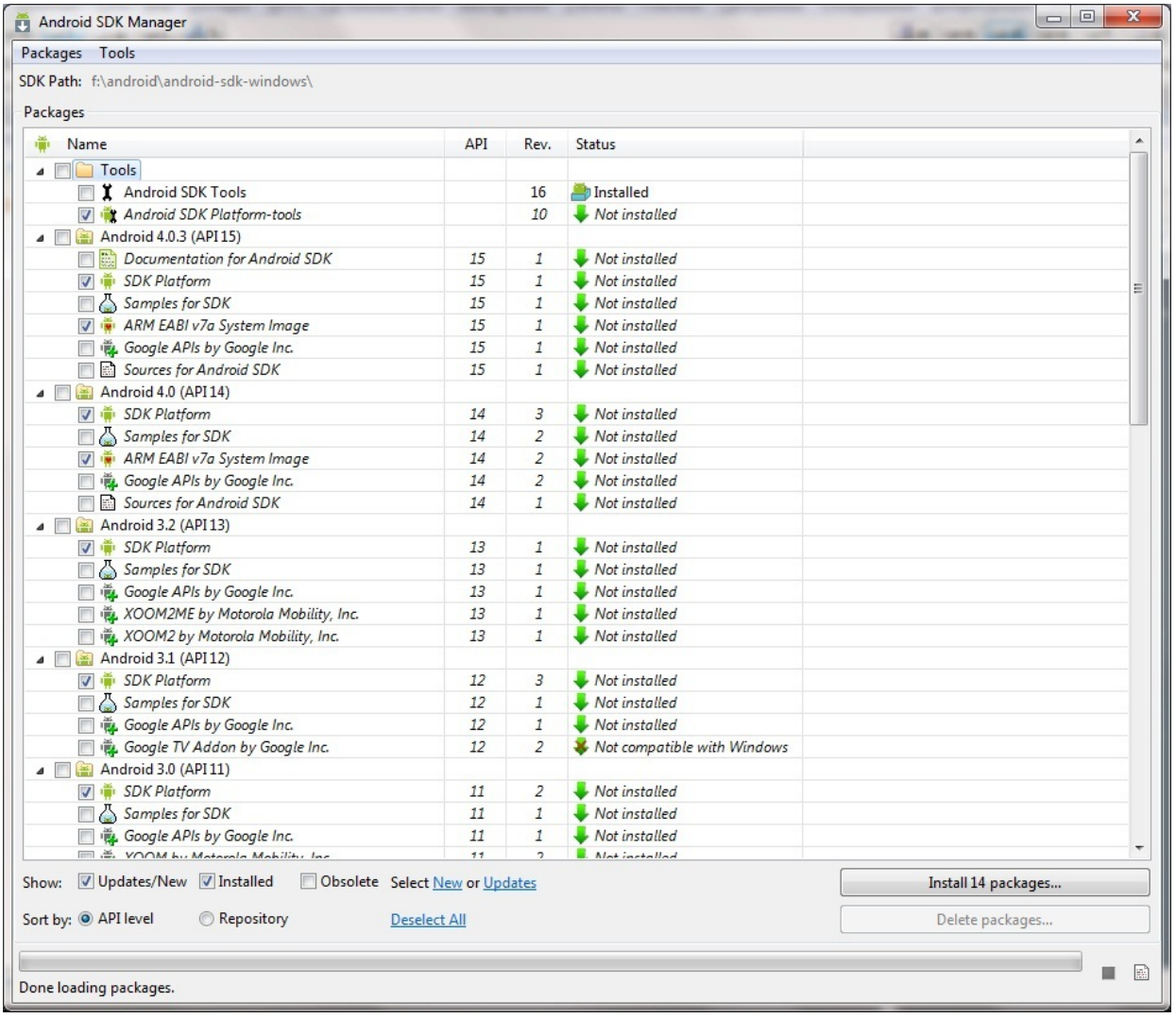
Выбираете, жмете Finish.

Далее нам сообщают, что наше скачанное SDK не содержит компонент и предлагают пройти в SDK Manager и срочно закачать. Жмем OK.



5. Платформы Android

И проваливаемся в SDK Manager.



Тут предлагается выбрать компоненты для закачки через интернет. В правой части для каждого компонента указано, установлен он уже или нет. Итак, что будем качать?

Обязательно нужно докачать неустановленное еще содержимое папки Tools. Также в папке Extras (в конце списка) найдите пункт *Android Support Library* и отметьте его, если он есть.

Для платформ 4.X выбираем *SDK Platform* (сама платформа) и *ARM EABI v7a System Image* (для работы эмулятора).

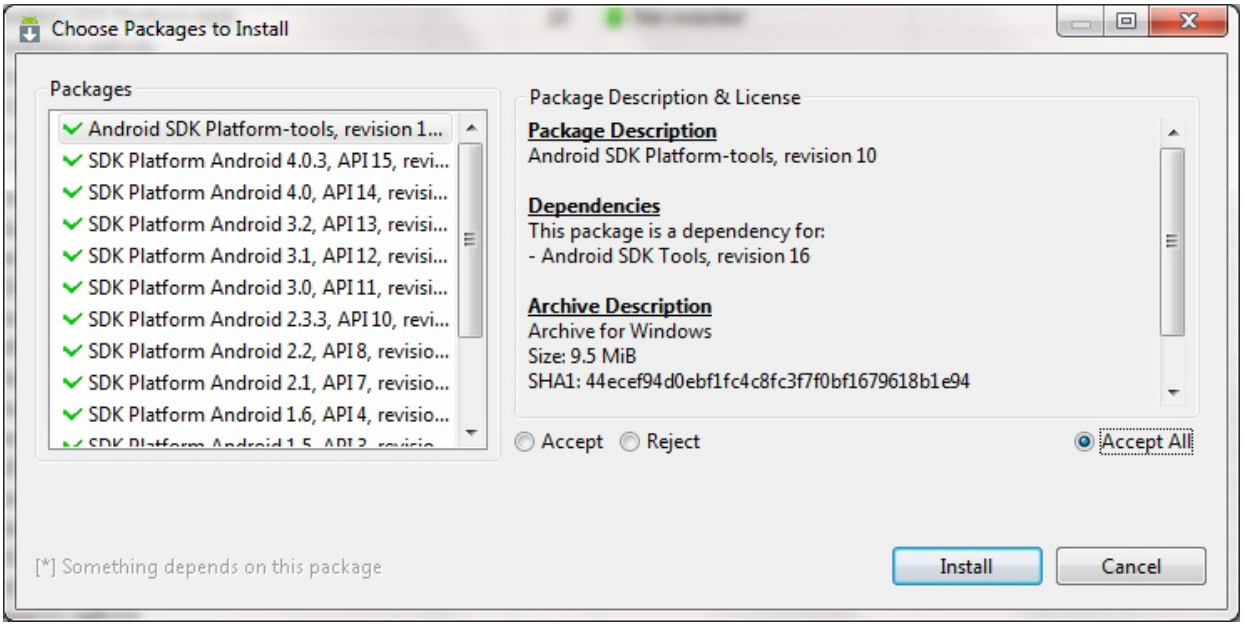
Для остальных, необходимых вам версий Android выбираете только SDK Platform. Учитывайте, что каждая платформа может весить до 150 метров!!! На работе особо не покачаешь, админы потом придут за вами :) Для первых уроков потребуется только платформа **2.3.3 (API 10)**. Можете скачать пока только ее.

Если же трафика не жалко, то добавляйте для каждой плафторомы пункты:
Samples for SDK - исходники примеров приложений
Google APIs by Google Inc. - нужно если собираетесь работать с гугл-приложениями (Map, Navigation и пр.)

Если трафика не жалко вообще - ставьте все галки. Но ждать придется долго. И займет все это дело не один гиг.

Когда все выбрали - жмем кнопку **Install X packages** справа снизу.

В новом окне подтверждаем, что согласны все это скачать - **Accept All**. Заодно здесь можно и размер посмотреть и отказаться от чего-либо.



Жмете Install - побежал индикатор и открылся лог - началась закачка. При моем выборе компонентов, я ждал минут 20. После этого в окошках появилась фраза: **Done loading packages.**

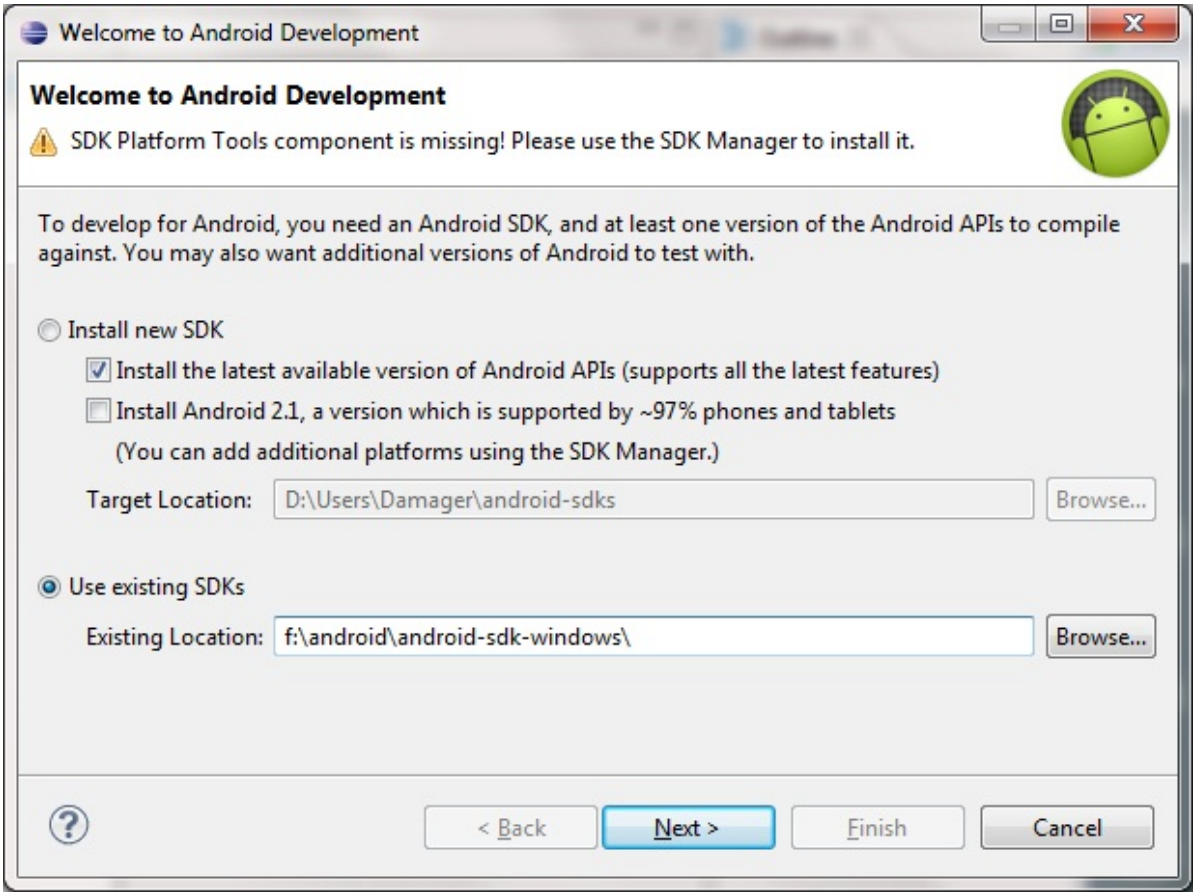
Установка завершена. Закрываем лог и SDK Manager.

Для информации - у меня новые компоненты заняли 1,5 гига.

Далее перезапускаете Eclipse и все. После выполнения этих шагов мы получили среду разработки, с помощью которой можно кодить Android-приложения. Возможно эти действия покажутся мутными и скучными, но без них, к сожалению, никак. Дальше дело пойдет веселей. Если что-то не получается или выдает ошибку - попробуйте погуглить, наверняка вы не первый сталкиваетесь с такой проблемой и в сети уже есть описание решения. Ну или пишите в форум.

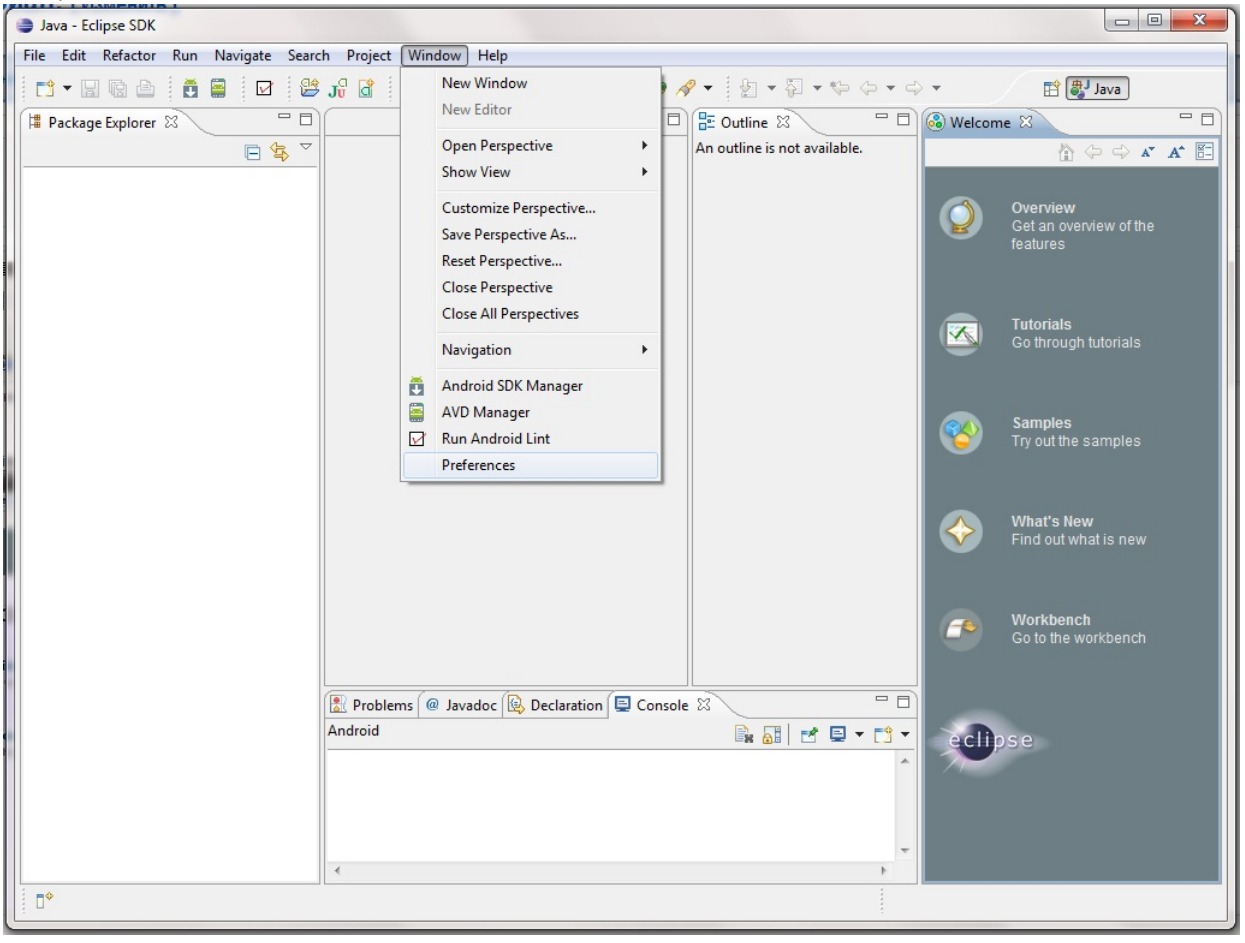
На следующем уроке мы в Eclipse настроим Android Virtual Device (AVD), создадим **наше первое приложение** и запустим его. AVD – это эмулятор смартфона с операционной системой Android, на котором можно запускать и тестировать приложения. Не подключать же свой смартфон каждый раз) Также мы рассмотрим структуру проекта приложения.

P.S. Если у вас не появилось это окно

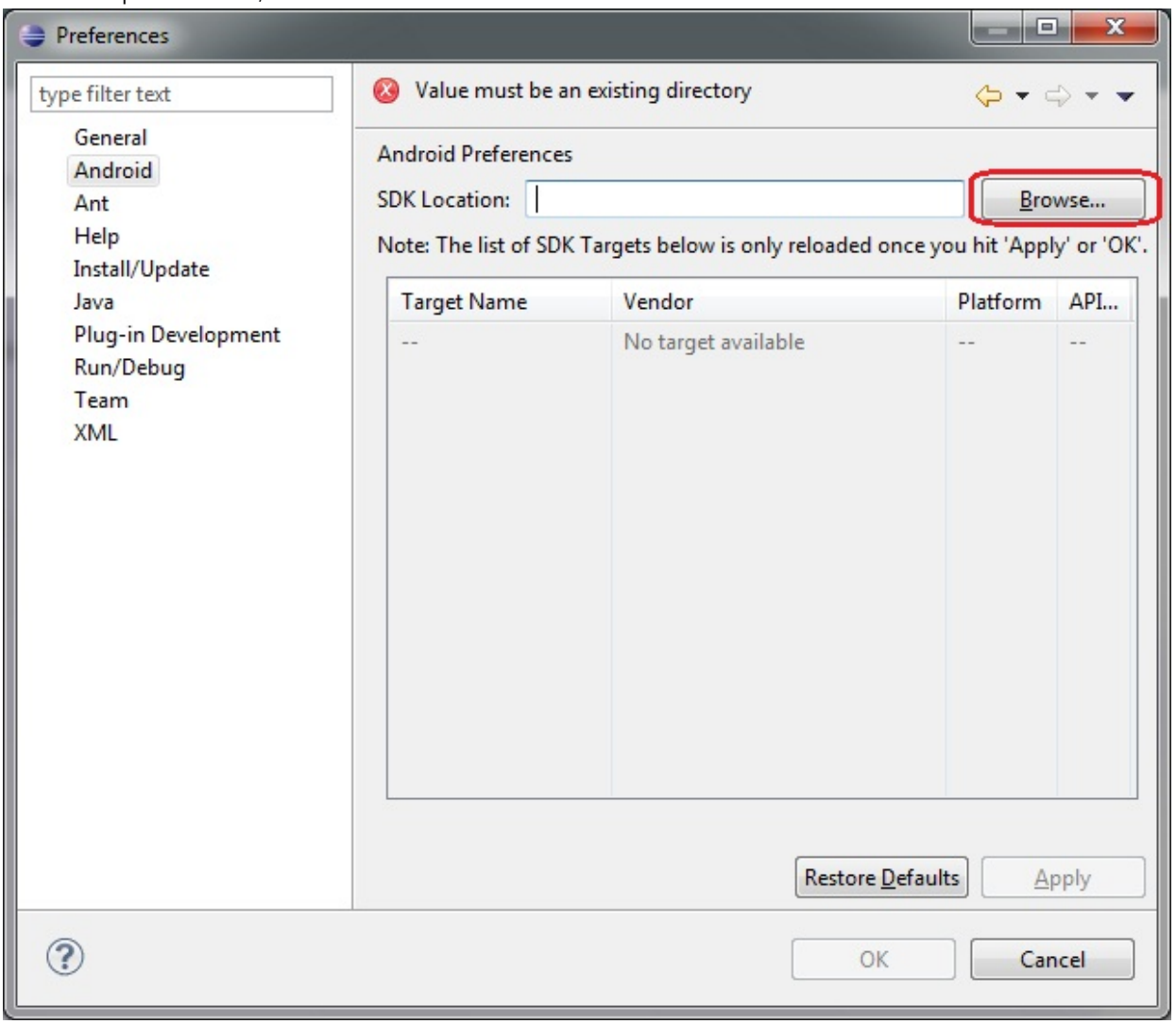


или вы его нечаянно закрыли - ничего страшного. Путь к SDK из шага 2 можно указать вручную.

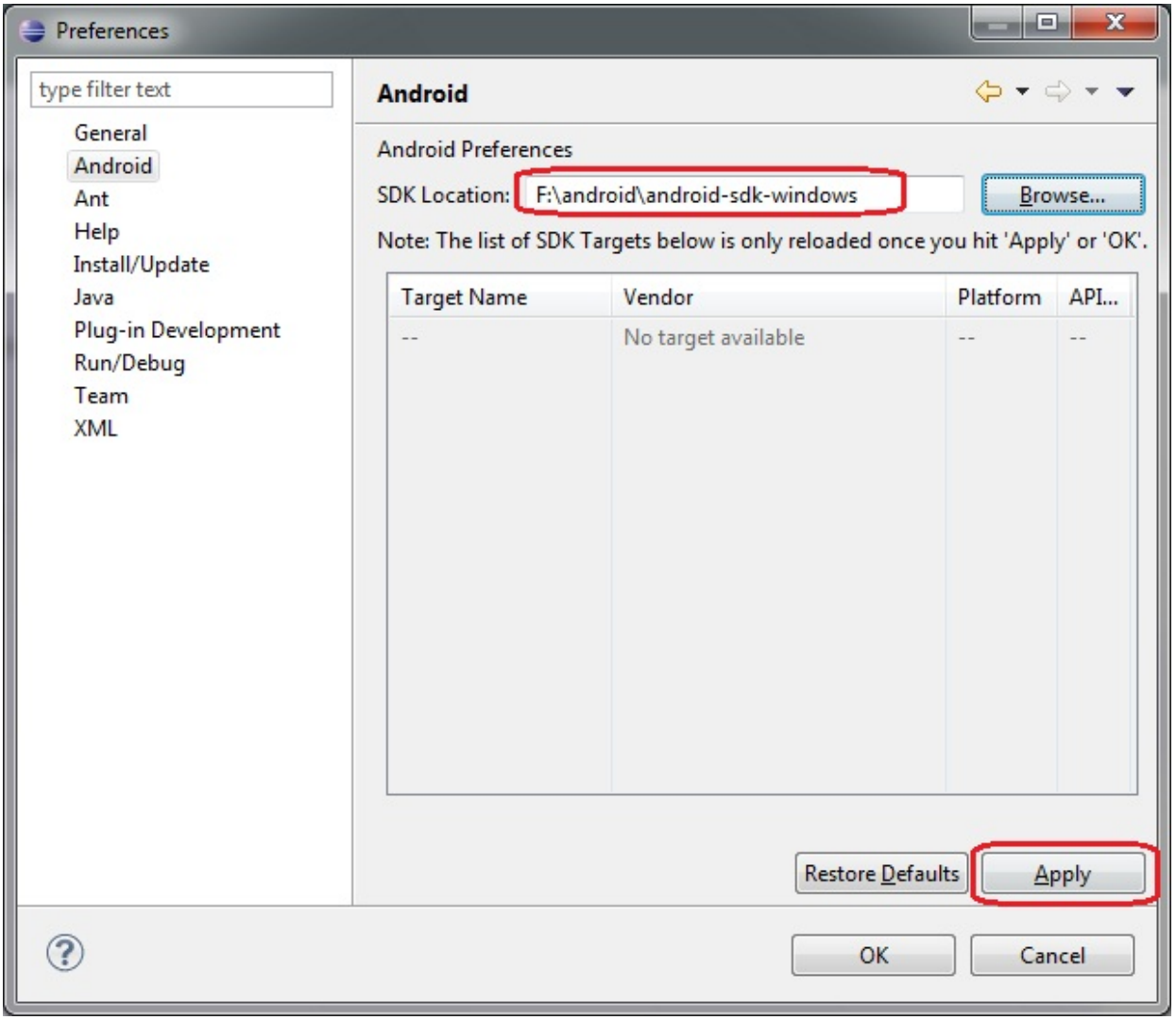
В Eclipse меню **Windows > Preferences**



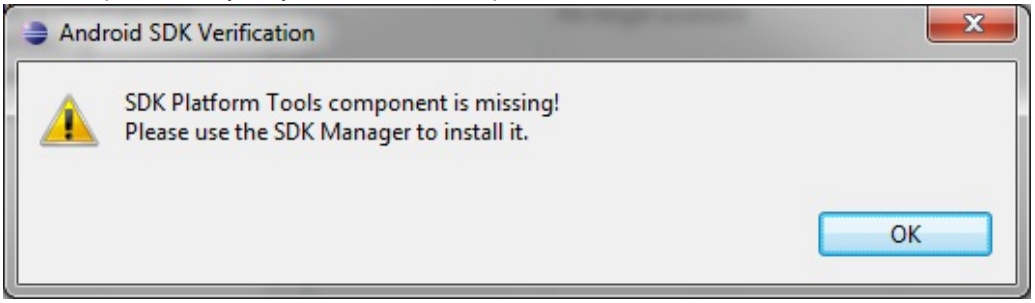
Слева выбираем **Android**, жмем Browse



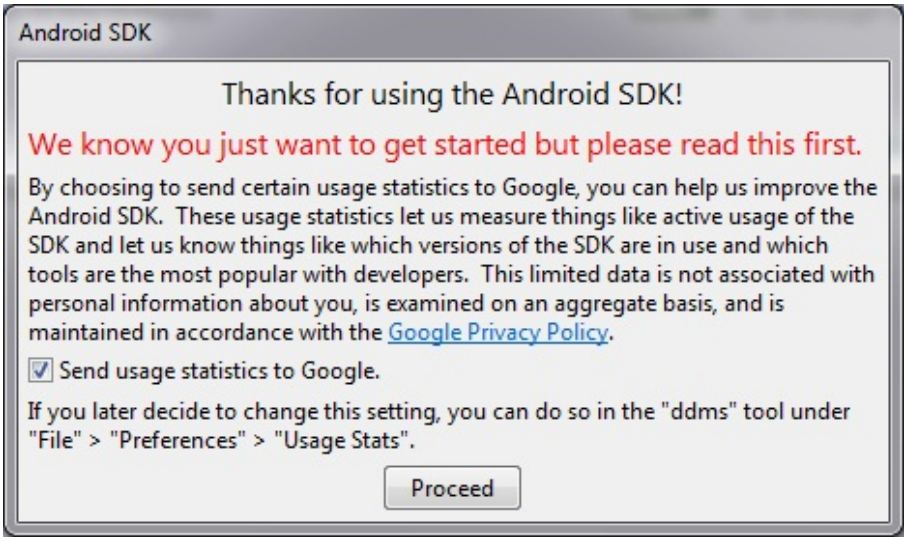
Указываем каталог, куда распаковали SDK - **<Android>\android-sdk-windows**. Жмем **Apply**.



Нам говорят, что отсутствует компонент - закрываем это сообщение и жмем **OK**



У меня еще выскочило такое окошко:



В нем сообщается, что если мы хотим улучшить продукт, то можем отправлять статистику использования. Выбор за вами, позже это всегда можно поменять в настройках. Я галку оставил, мне не жалко)

После этого можно идти в SDK Manager (в Eclipse меню **Window > Android SDK Manager**) и скачивать платформы.

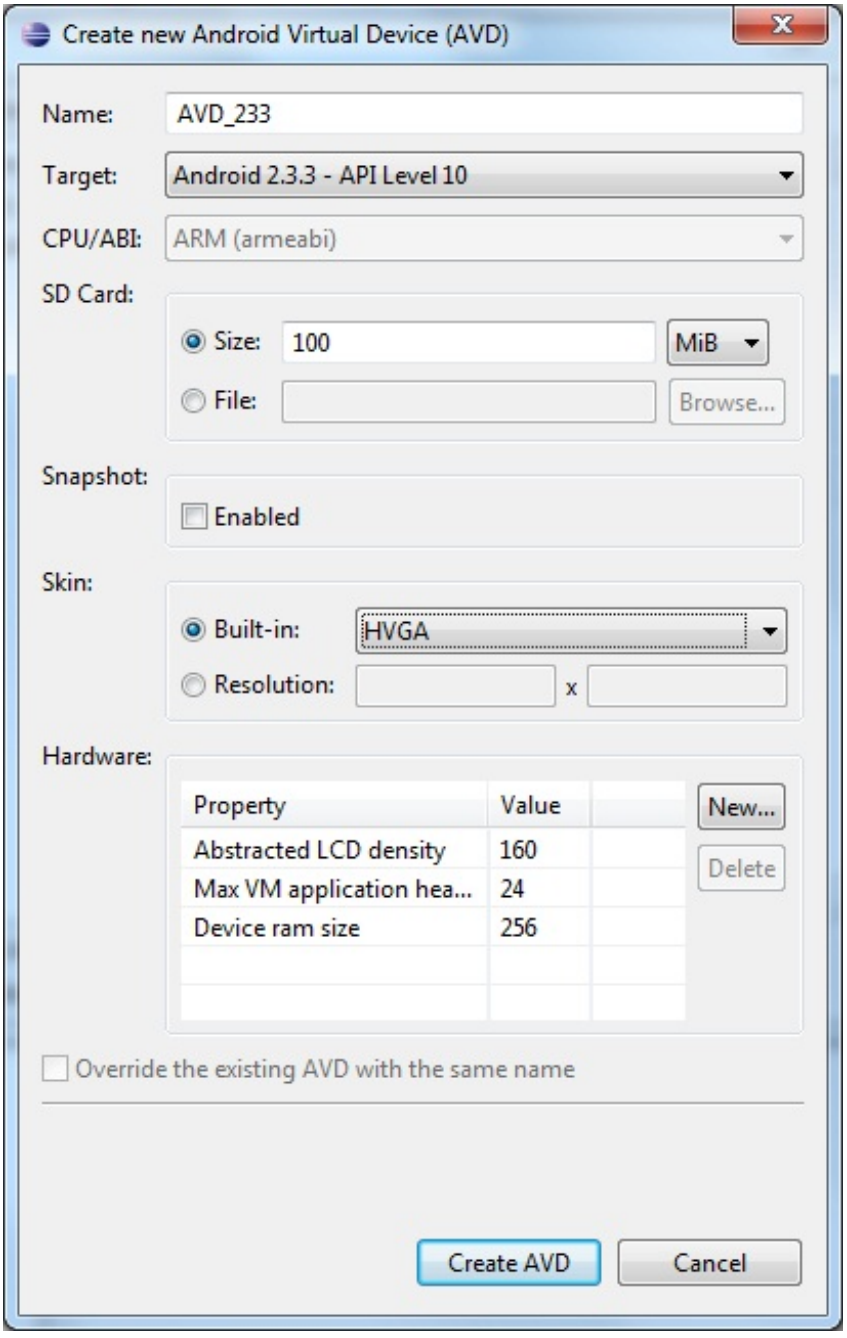
Урок 3. Создание AVD. Первое приложение. Структура Android-проекта.

Для того, чтобы тестировать приложения, нам понадобится Android Virtual Device (AVD). Это эмулятор Android-смартфона, на который Eclipse сможет устанавливать, созданные нами приложения, и запускать их там. Давайте его создадим.

Запускаем **Eclipse**
Идем в меню **Windows > Android SDK and AVD Manager** (в более свежих версиях - **Android AVD Manager**)
Слева выбираем **Virtual Devices**
Жмем кнопку **New**

В списке **Target** представлены платформы Android различных версий. Мы их закачали на прошлом уроке последним шагом. Разницу между ними можно посмотреть здесь. Слева выбираете нужную версию и смотрите информацию по ней.

Вернемся в Eclipse. Давайте выберем из списка платформу **2.3.3 (API Level 10)**. Соответственно имя зададим - **AVD_233**. Также укажем размер **SDCard = 100** и режим экрана **HVGA**.



Жмем **Create AVD**, и закрываем **AVD Manager**.

Теперь наконец-то мы можем создать наше первое приложение и посмотреть как оно работает.

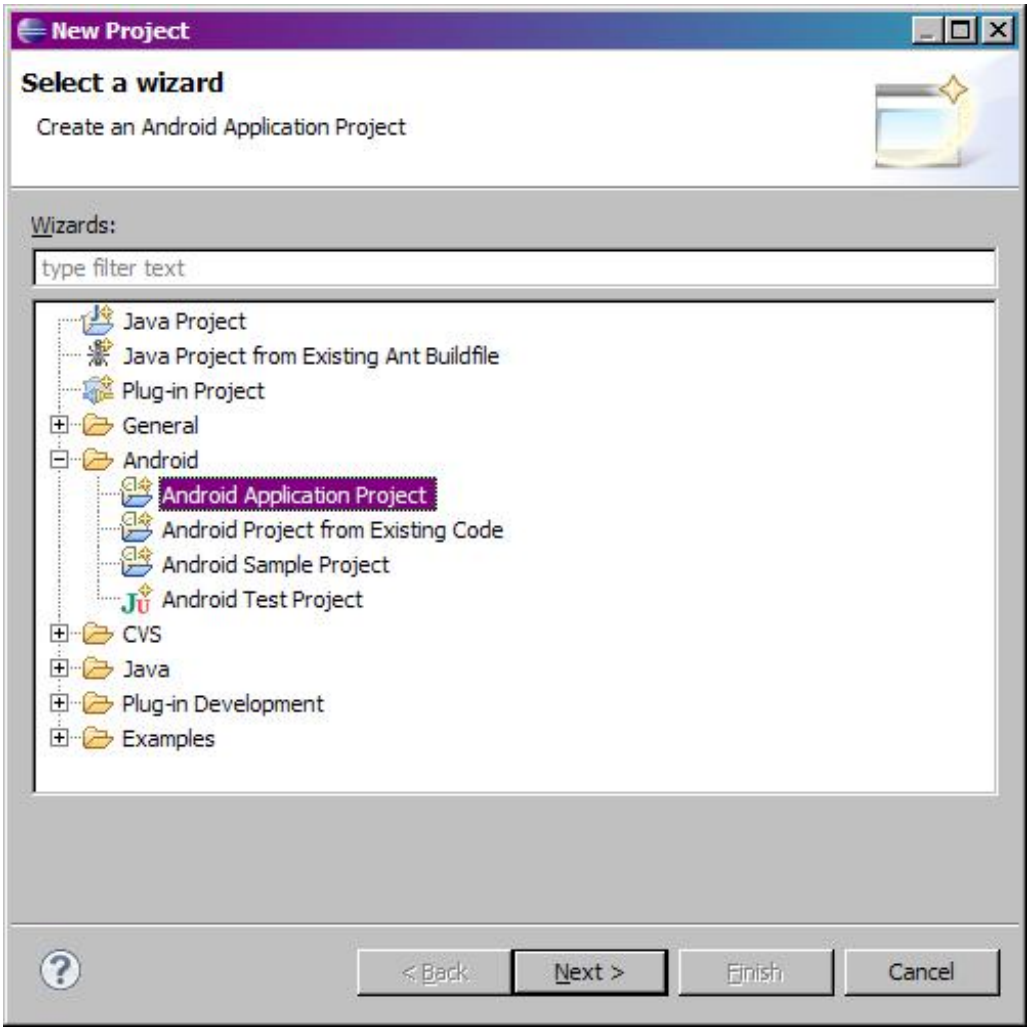
UPD от 10.07.2012 - этот урок был написан давно. На днях гугл существенно обновил визард нового проекта.

Здесь вы можете посмотреть подробно, какие возможности дает новый визард. Я же покажу, как создать простой проект.

Итак, сначала будут скрины для нового варианта визарда. Если же вы давно не обновлялись, то пролистайте чуть дальше - там будут скрины для старого визарда.

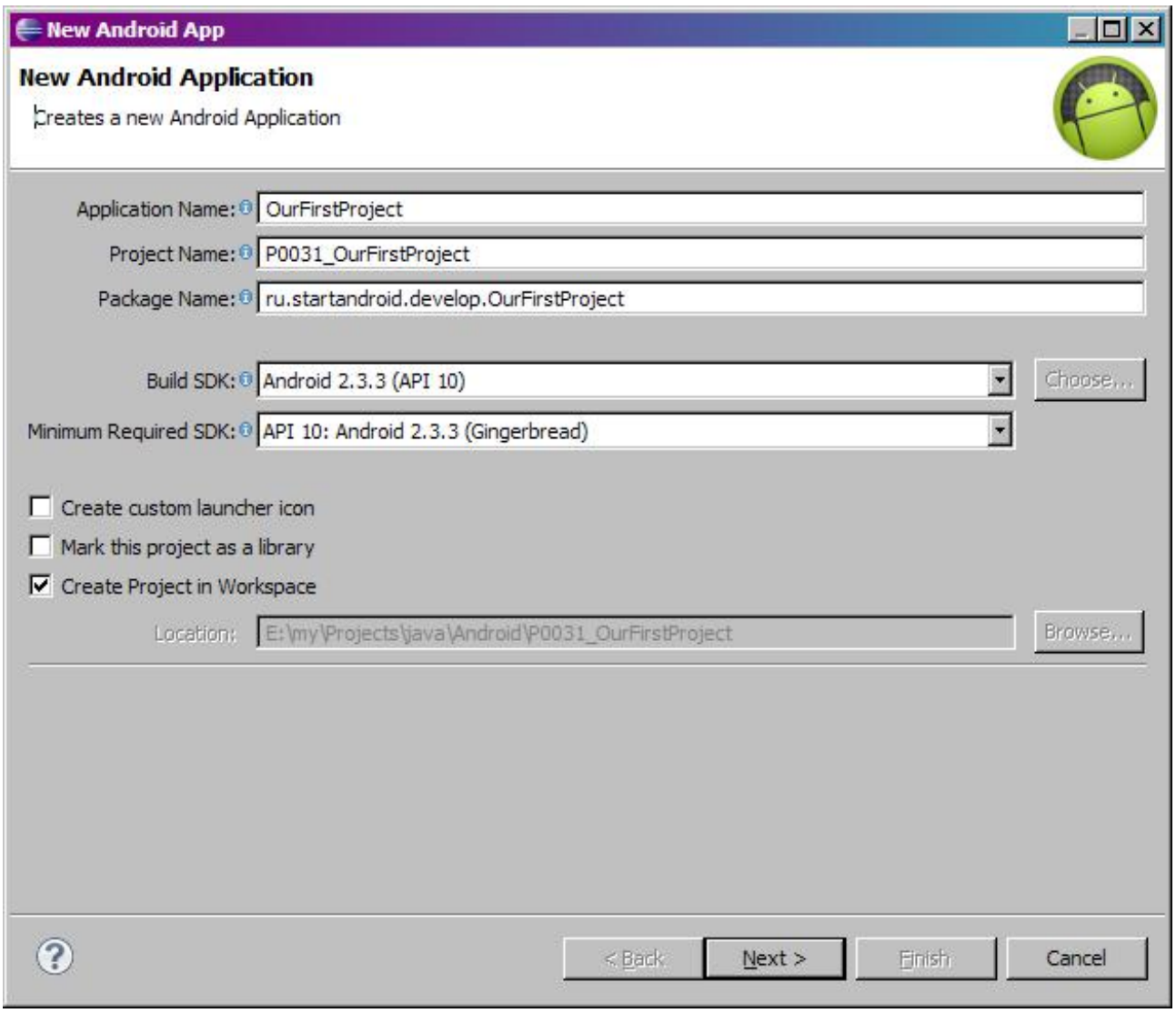
Новый вариант.

В Eclipse идем в меню **File > New > Project**



Выбираем тип проекта - **Android > Android Application Project**, жмем **Next**

Появилось окно создания проекта.



Давайте заполнять.

Начнем с **Project Name** – это имя проекта, которое будет видно в общем списке проектов слева. Т.к. проектов у нас будет много, предлагаю придумать префикс для названий. Так будет проще к ним возвращаться, и они будут красиво и удобно структурированы.

Префикс может быть таким: P<номер урока(000)><номер проекта в уроке(0)> . На номер урока выделим три цифры, на номер проекта – одну. Добавим к префиксу некое смысловое название (OurFirstProject) и получим имя нашего первого проекта в третьем уроке - **P0031_OurFirstProject**.

Application name – непосредственно имя программы, которое будет отображаться в списке приложений в смартфоне. Можно брать имя проекта без префикса

Package name – это понятие из Java, подробно можно посмотреть здесь. Вкратце – это префикс для имени классов нашего приложения. Я буду использовать **ru.startandroid.develop.<имя приложения>**

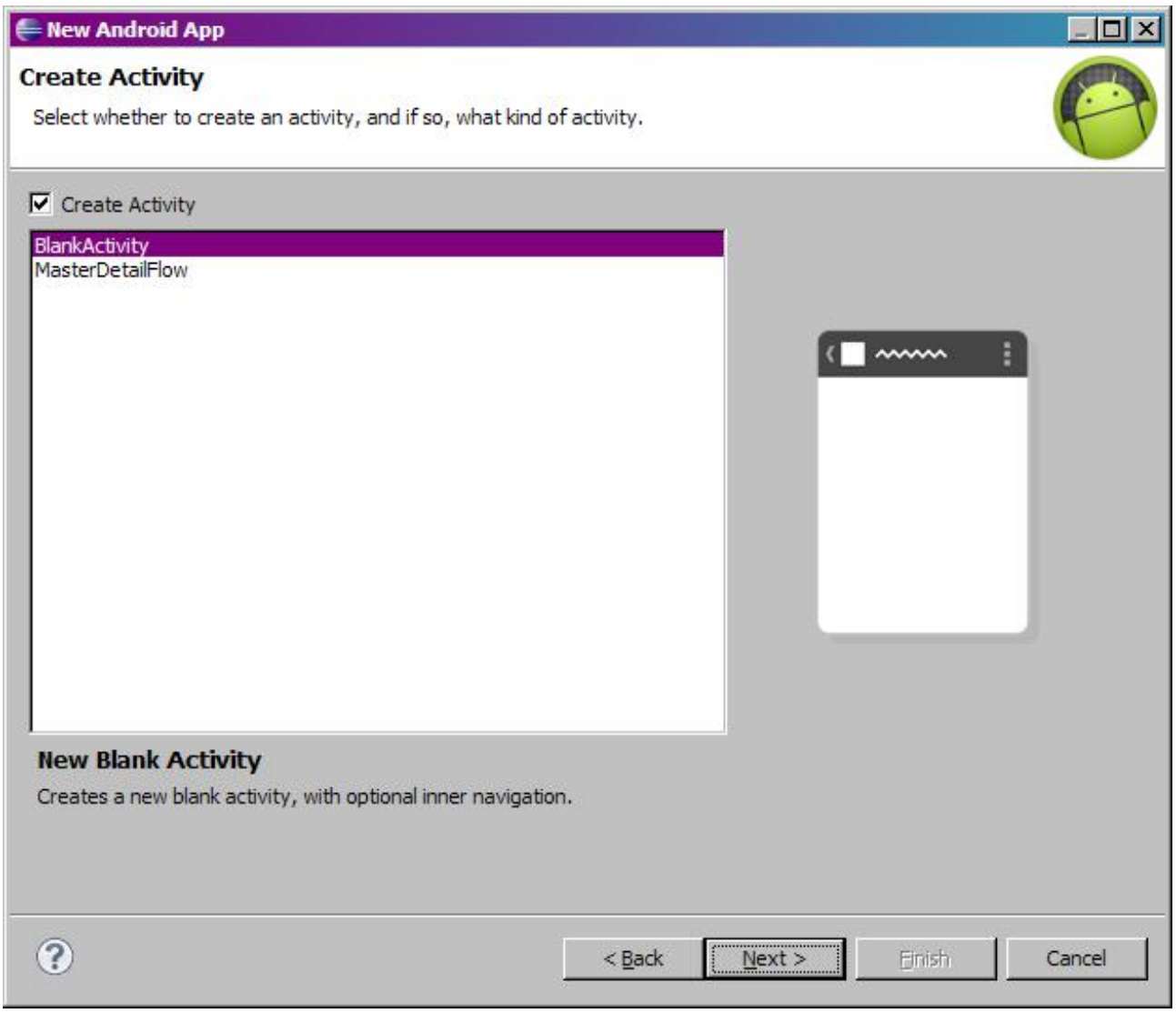
Build SDK определяет, возможности какой версии Android будет использовать приложение. Выберем ту же, что и при создании AVD – т.е. **2.3.3**.

Minimum Required SDK определяет минимальную версию Android, на которой запустится приложение. Можно сделать ее равной Build SDK - 2.3.3.

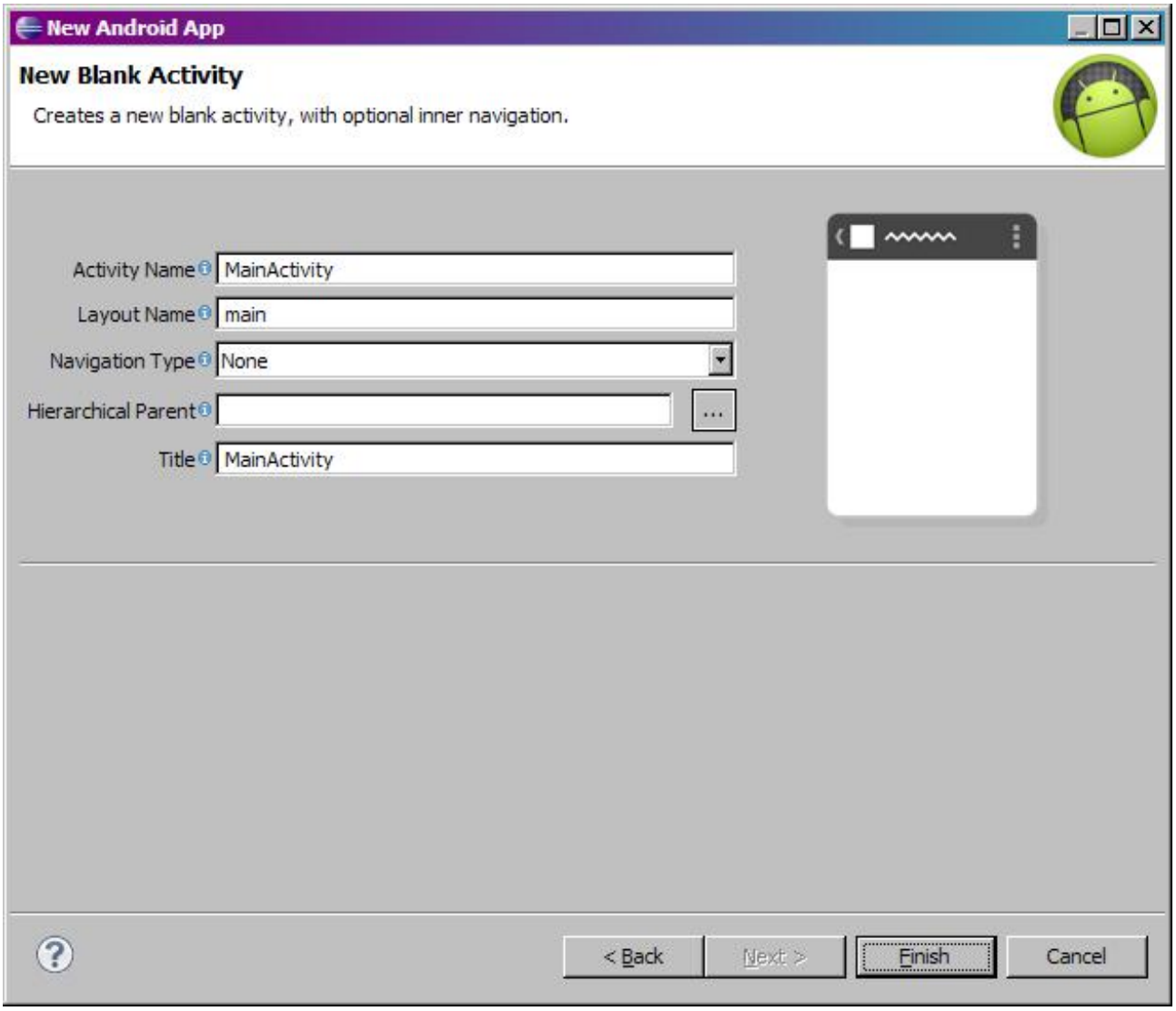
Из галок оставляете только **Create Project in Workspace**. Проект будет создан и сохранен в дефолтном Workspace.

Жмем Next.

Открывается экран создания Activity.



Здесь выбираем BlankActivity и жмем Next.



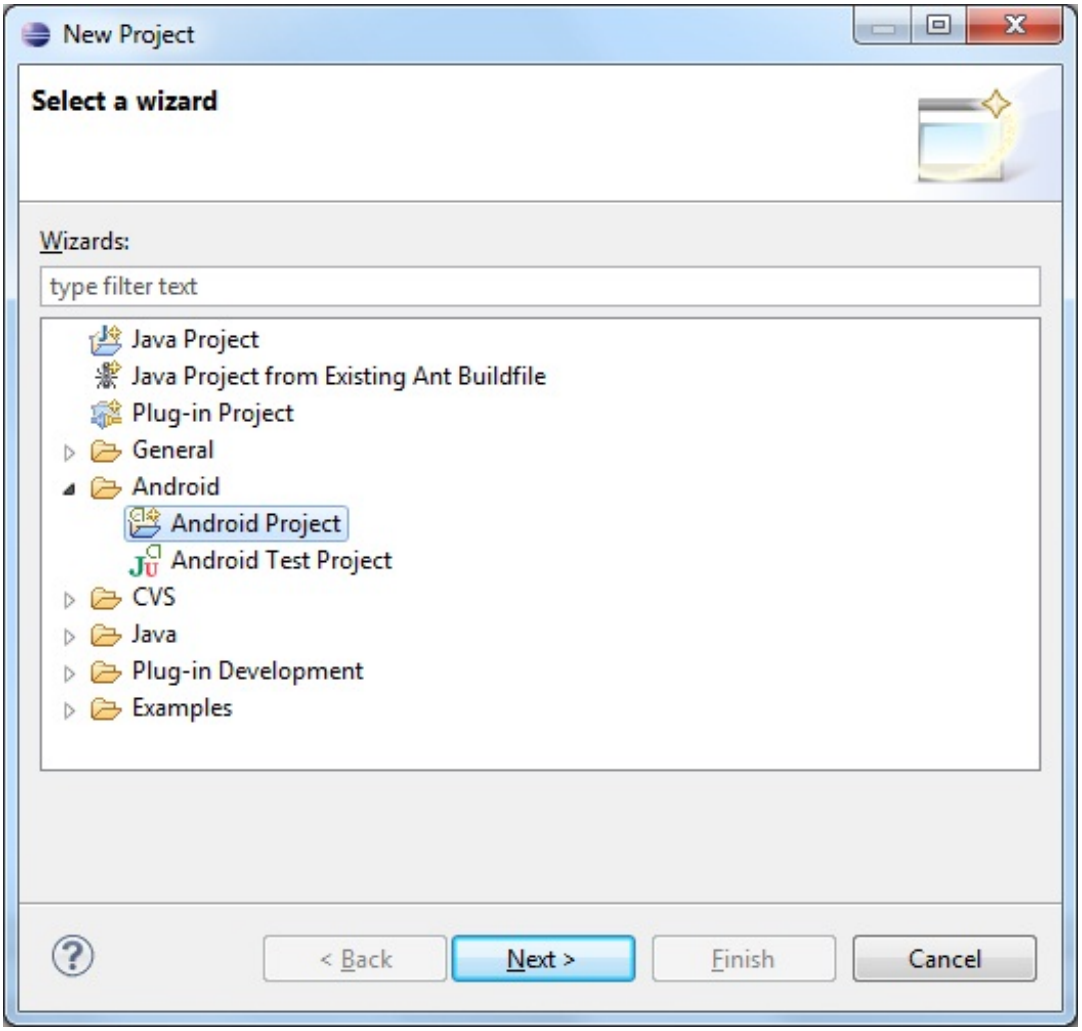
Здесь убедимся, что в поле **Activity Name** указано **MainActivity**, а в поле **Layout Name** укажем **main**. Остальные поля не трогаем.

Жмем **Finish**.

Проект создан.

Старый вариант.

В Eclipse идем в меню **File > New > Project**



Выбираем тип проекта - **Android > Android Project**, жмем **Next**

Появилось окно создания проекта. (В последующих версиях это окно разделено на три разных окна)

Давайте заполнять.

Project Name – имя проекта, которое будет видно в общем списке проектов слева. Т.к. проектов у нас будет много, предлагаю придумать префикс для названий. Так будет проще к ним возвращаться, и они будут красиво и удобно структурированы.

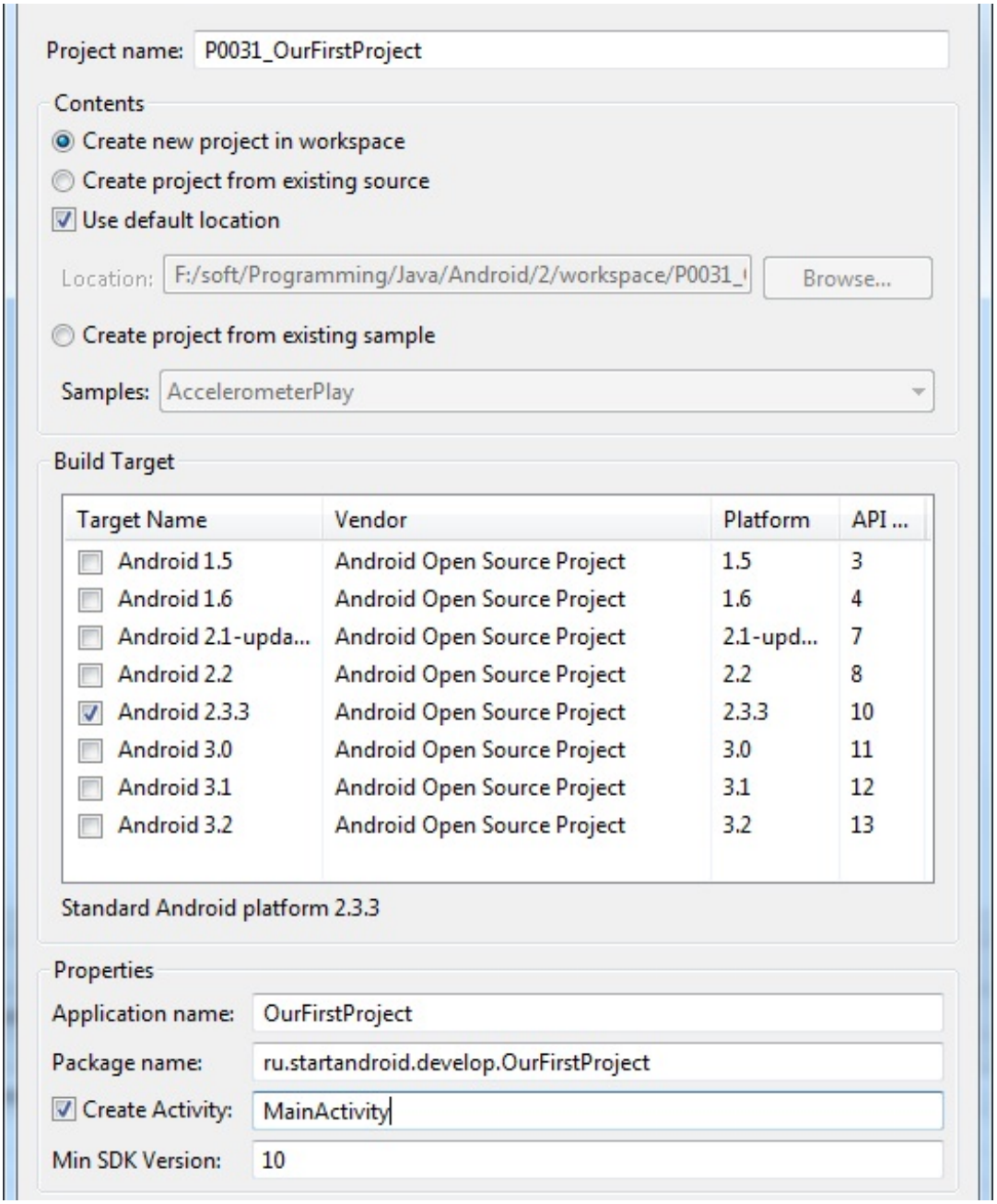
Префикс может быть таким: P<номер урока(000)><номер проекта в уроке(0)> . На номер урока выделим три цифры, на номер проекта – одну. Добавим к префиксу некое смысловое название (OurFirstProject) и получим имя нашего первого проекта в третьем уроке - **P0031_OurFirstProject**.

Build Target определяет возможности какой версии Android будет использовать приложение. Выберем ту же, что и при создании AVD – т.е. **2.3.3**.

Application name – непосредственно имя программы, которое будет отображаться в списке приложений в смартфоне. Можно брать имя проекта без префикса

Package name – это понятие из Java, подробно можно посмотреть здесь. Вкратце – это префикс для имени классов нашего приложения. Я буду использовать **ru.startandroid.develop.<имя приложения>**

Create Activity – имя класса для Activity (окна программы). Укажем, например - **MainActivity**.

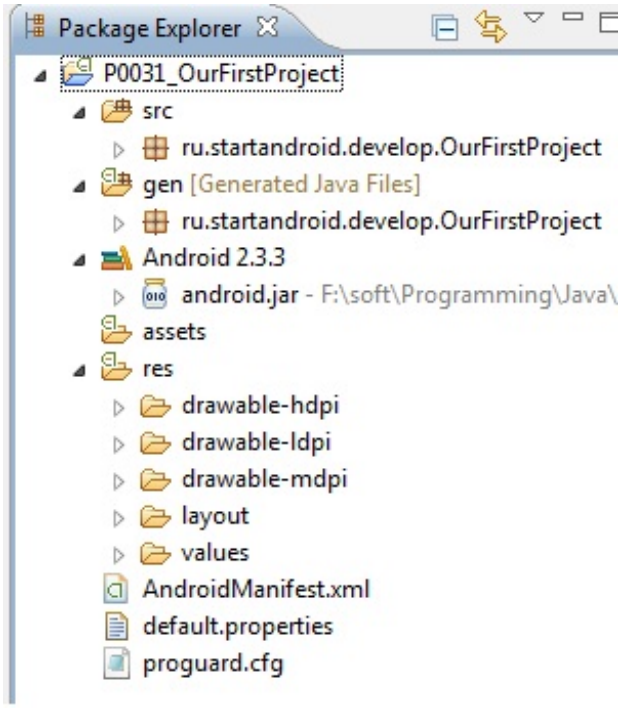


Все остальное оставляем как есть. Жмем **Finish**.

Проект создан.

Дальнейшие скрины для разных версий могут отличаться, но незначительно.

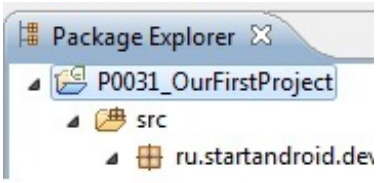
Слева появился наш проект, давайте раскроем его. Разберем наиболее важные и часто используемые папки и файлы:



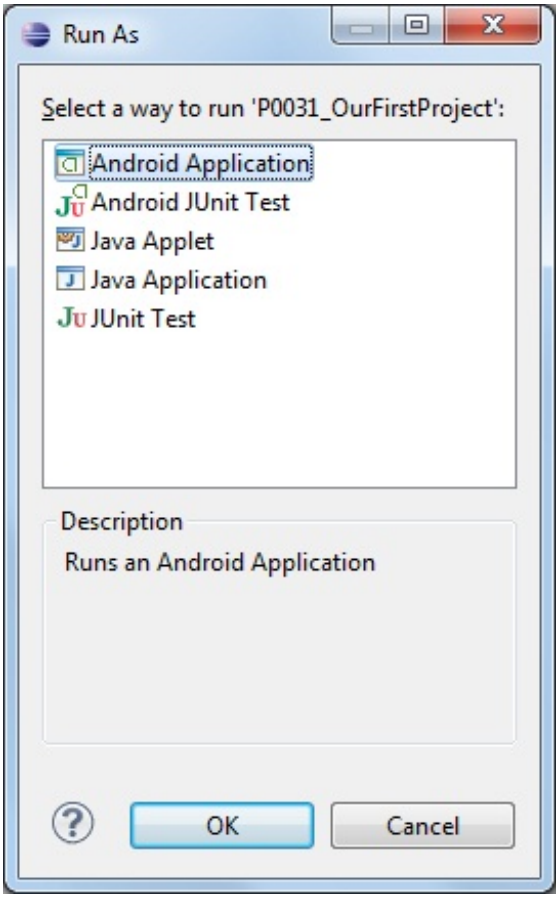
src – все, написанные нами исходные коды, будут в этой папке
gen – генерируемые средой файлы, необходимые для приложения. Здесь лучше ничего не трогать . (Если этой папки нет - что-нибудь измените в проекте и нажмите кнопку **сохранить**).
Android 2.3.3 – необходимые для приложения Android-библиотеки
assets и **res** – папки для файлов-ресурсов различного типа
AndroidManifest.xml – манифест или конфиг-файл приложения

Все это мы будем в дальнейшем использовать, и станет понятнее, что и зачем нужно.

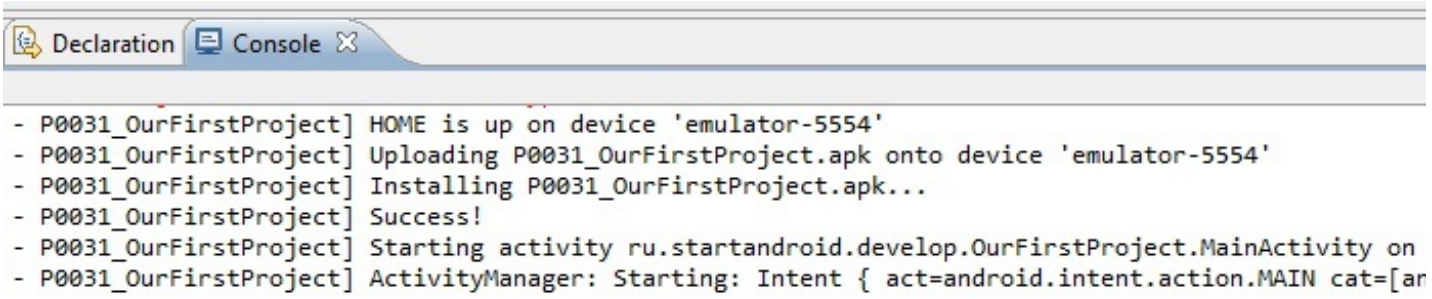
Давайте выделим имя проекта слева:



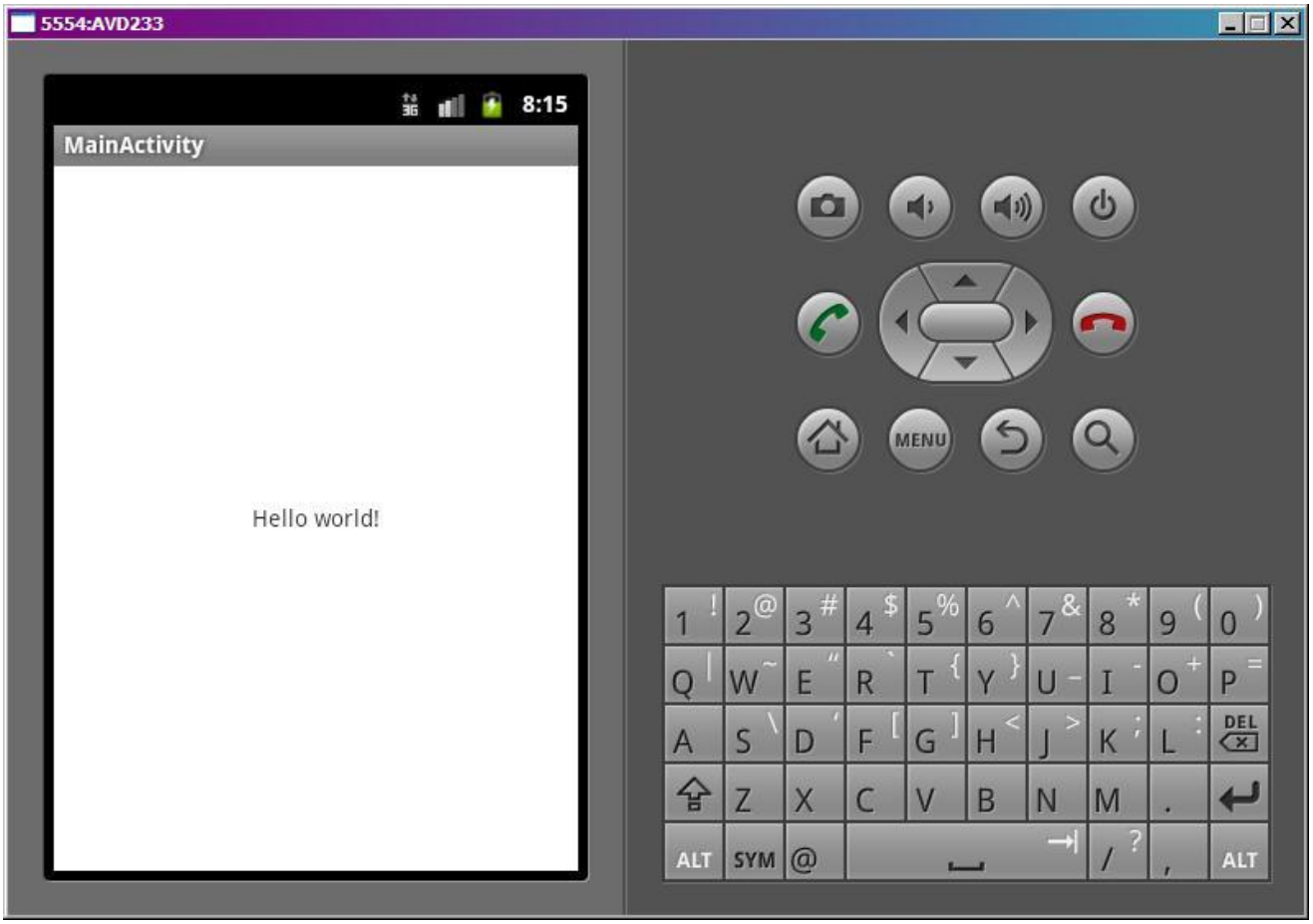
и запустим его - нажмем **CTRL+F11**, далее выбираем **Android Application**,



жмем **OK**. Запустится эмулятор, советую в это время ничего особо не трогать и не нажимать, т.к. это глючная и капризная штука. Время запуска - около минуты. Дожидаемся, пока в консоли Eclipse появятся подобные строки.



Приложение закачано на эмулятор, установлено и запущено. Снимаем блокировку экрана в эмуляторе (если она есть) и наблюдаем наше приложение:



Видно название приложения и название Activity. Мы создали и запустили первое приложение.

Если не получилось запустить и Eclipse вывел в консоль подобное сообщение: "emulator-5554 disconnected! Cancelling 'ru.startandroid.develop.OurFirstProject.MainAct activity launch'!" - то **закройте эмулятор**, и попробуйте снова. Если снова не получилось. **перезапустите Eclipse**. Если опять нет - **ребутнитесь**. Если и сейчас не работает, **удалите AVD и создайте новый**. В итоге должно заработать, пусть и не с первой попытки.

Главное - после запуска приложения (CTRL+F11) старайтесь совершать **как можно меньше движений** на компе. Я заметил четкую тенденцию - если во время запуска переключаться между различными окнами, то эмулятор запускается криво. А если просто посидеть и **подождать минутку** - то все ок. Ну и надо чтоб при этом не было включено какое-нить кодирование видео или прочие, нагружающие систему процедуры.

Подробнее про эмулятор можно почитать здесь.

P.S.

Если у вас имя пользователя русскими буквами, то будут небольшие проблемы. Как их решить, можно прочесть на форуме в ветке этого урока.

На следующем уроке будем добавлять в наше приложение различные элементы и менять их свойства.

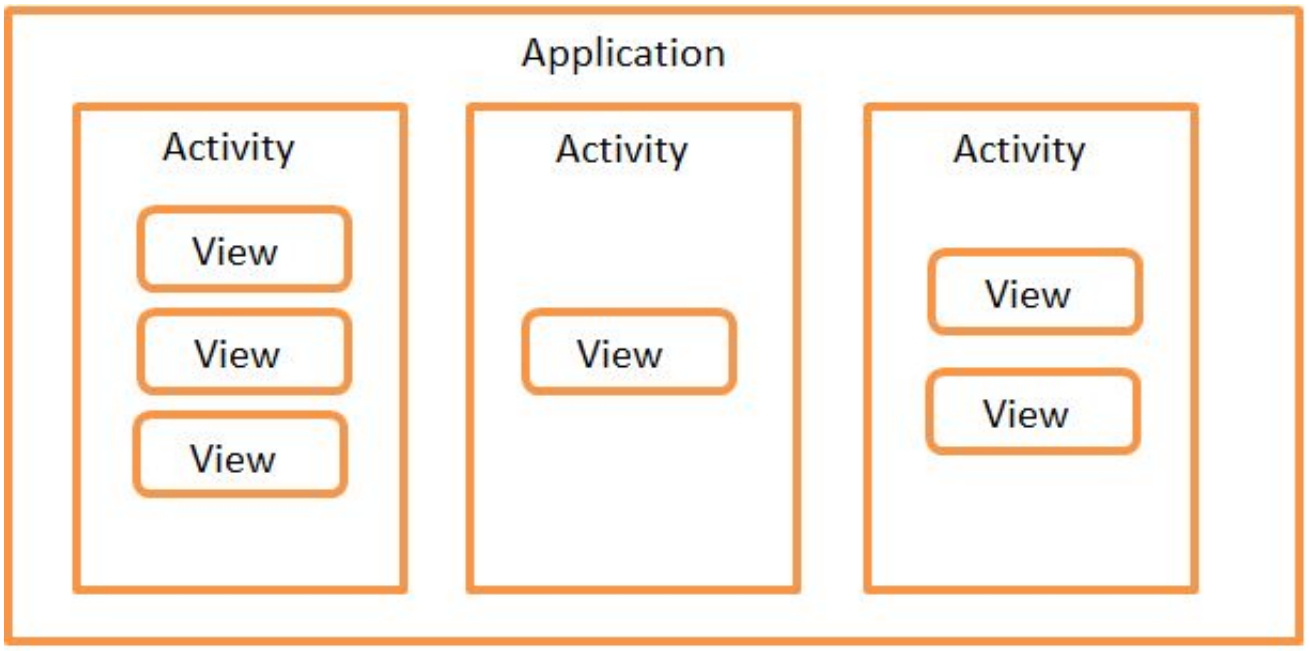
Урок 4. Элементы экрана и их свойства

Давайте посмотрим, как в Андроид приложениях формируется то, что мы видим на экране.

Если проводить аналогию с Windows, то приложение состоит из окон, называемых **Activity**. В конкретный момент времени обычно отображается одно Activity и занимает весь экран, а приложение переключается между ними. В качестве примера можно рассмотреть почтовое приложение. В нем одно Activity – список писем, другое – просмотр письма, третье – настройки ящика. При работе вы перемещаетесь по ним.

Содержимое Activity формируется из различных компонентов, называемых **View**. Самые распространенные View - это кнопка, поле ввода, чекбокс и т.д.

Примерно это можно изобразить так:



Необходимо заметить, что View обычно размещаются в **ViewGroup**. Самый распространенный пример ViewGroup – это **Layout**. Layout бывает различных типов и отвечает за то, как будут расположены его дочерние View на экране (таблицей, строкой, столбцом ...)

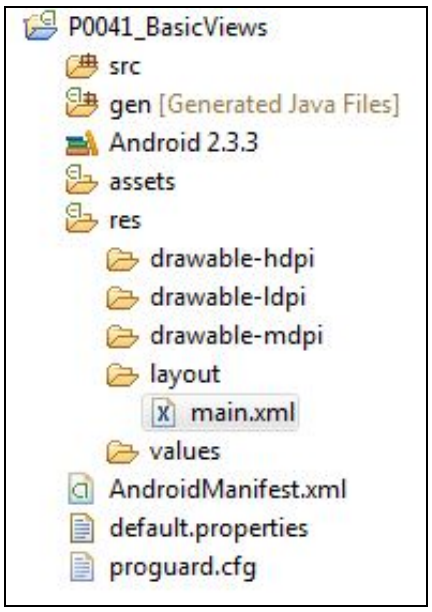
Подробней об этом можно почитать в хелпе: User Interface и Common Layout Objects.

Наверно уже запутал новыми словами и терминами, давайте посмотрим это все на практике. Создадим проект со следующими свойствами:

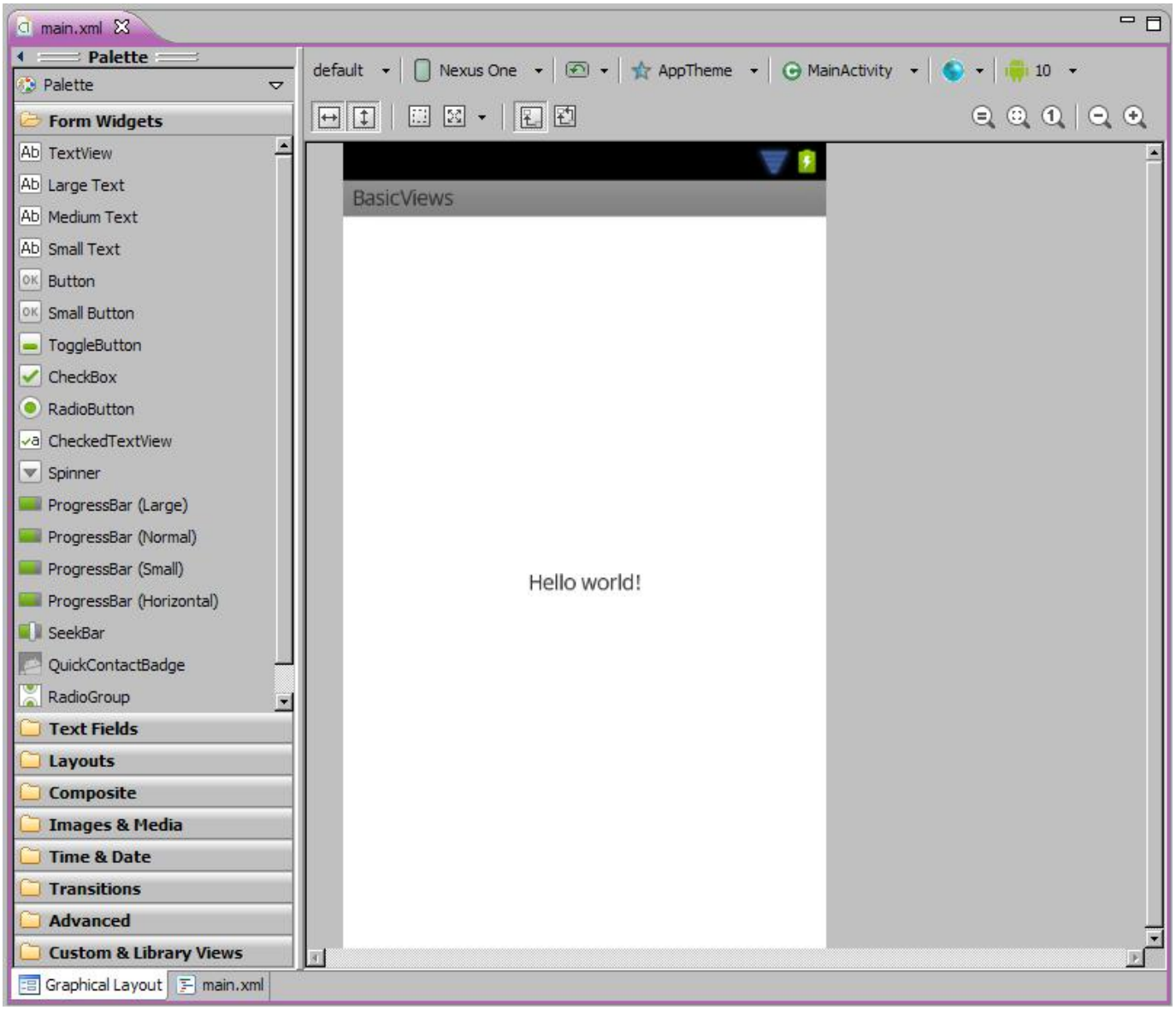
Project name: P0041_BasicViews
Build Target: Android 2.3.3
Application name: BasicViews
Package name: ru.startandroid.develop.BasicViews
Create Activity: MainActivity

Если у вас свежая версия визарда создания проекта, то **Build Target** - это то же самое, что и **Build SDK**. А в экране создания Activity не забывайте указывать **main** в поле **Layout Name**.

В нашем проекте нам интересен файл: **res > layout > main.xml**

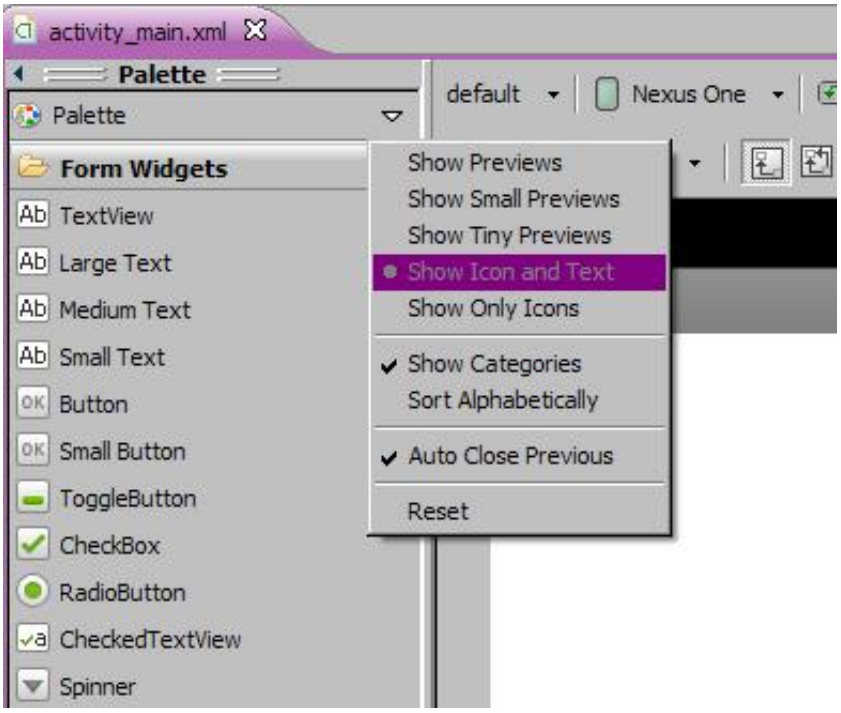


Это **layout-файл**, в нем мы определяем набор и расположение элементов View, которые хотим видеть на экране. При запуске приложения, Activity читает этот файл и отображает нам то, что мы настроили. Давайте откроем его и посмотрим, какой набор View он содержит по умолчанию.

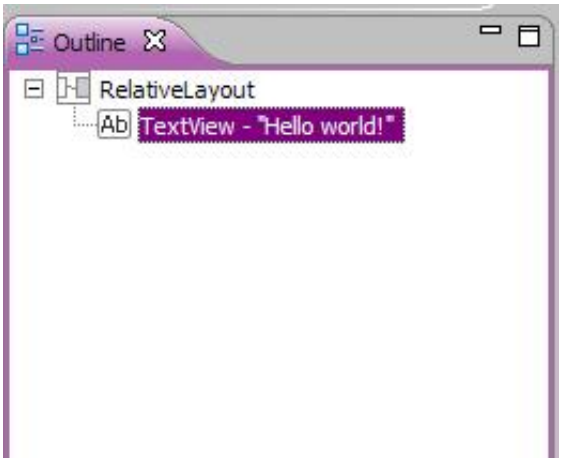


Слева видим список View, разделенный на группы. Здесь отображены все View-элементы, которые вы можете использовать в своих приложениях.

Возможно, у вас он будет немного другого вида. Это регулируется в меню Palette, чуть выше.



Обратим внимание на белый экран. Мы видим, что на экране сейчас присутствует элемент с текстом Hello world! Чтобы узнать, что это за View нажмите на этот текст. Справа во вкладке **Outline** вы видите все элементы, которые описаны в main.xml.



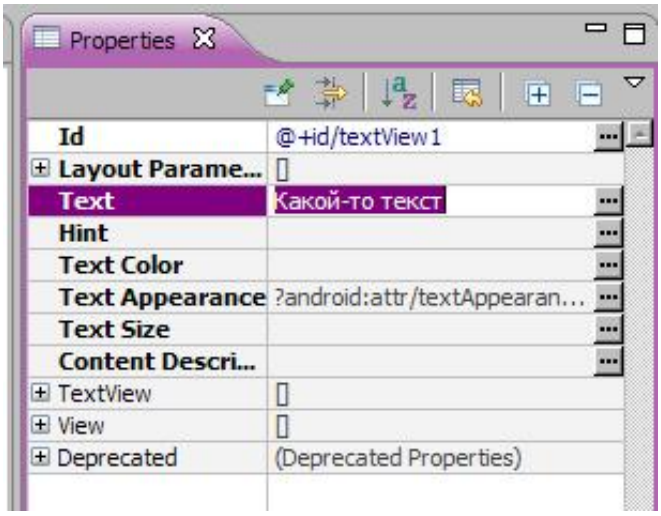
Видим, что выделенный нами элемент – это **TextView**. Обратите внимание, что он вложен в элемент **RelativeLayout** – это ViewGroup, про которые я писал выше.

Добавим еще элементов на экран, пусть это будут **Button** и **CheckBox**. Для этого просто перетащите их из списка слева на белый экран вашего будущего приложения. Также можно перетащить их на RelativeLayout во вкладке Outline, результат будет почти тот же. Кроме Button и CheckBox, добавим еще на экран **Plain Text** из группы **Text Fields**.

В Outline они появятся под названиями **button1**, **checkBox1** и **editText1**. Это **ID**, которые им были присвоены автоматически. Пока оставим их такими, позднее научимся их менять и будем делать более осмысленными.

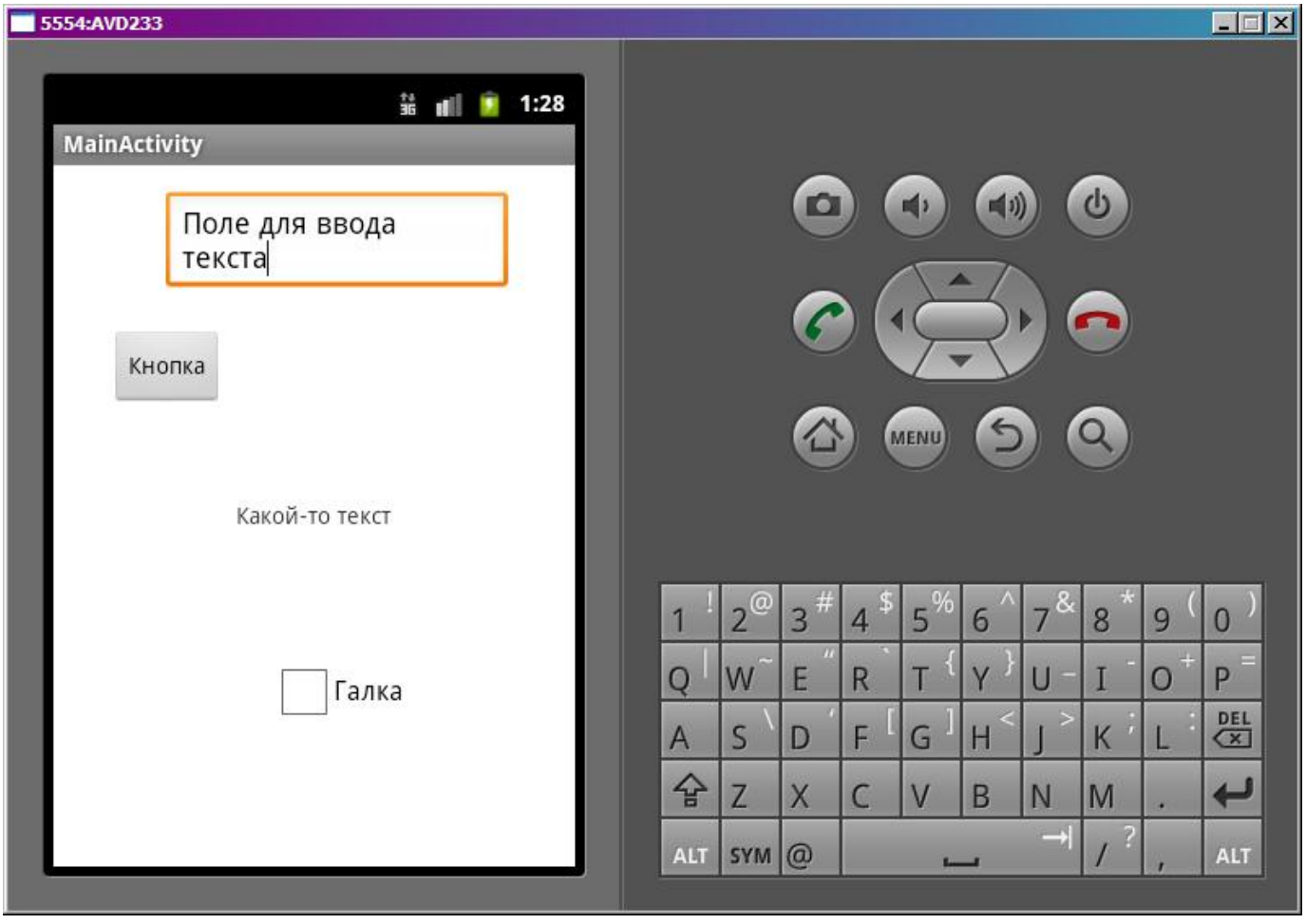
Теперь на нашем экране несколько элементов. Давайте изменим надписи на них. Во вкладке Outline жмем на TextView. Теперь нам нужна вкладка **Properties**. Она отображает свойства выделенного в Outline или на экране View-элемента. Располагается она обычно сразу под Outline.

Найдем в Properties свойство **Text**. Сейчас там стоит ссылка на текстовую константу. Где создаются эти константы мы рассмотрим в следующих уроках, а пока просто давайте напишем сюда свой текст: «Какой-то текст»



Аналогично изменим свойство Text для элементов button1, checkBox1 и editText1 на произвольные тексты. Все эти изменения пишутся в main.xml. Сохраним все CTRL+SHIFT+S и запустим CTRL+F11.

Приложение отображает нам MainActivity, а оно в свою очередь читает файл main.xml и отображает все View, которые мы создавали и настраивали.



На этом уроке мы:

узнали, что выводится на экран в Android-приложении
научились формировать layout-файл – добавлять View и настраивать их свойства

На следующем уроке мы:

рассмотрим layout-файл с другого ракурса - XML
разберем, откуда Activity знает, какой layout-файл надо читать и попробуем настроить на чтение другого файла
узнаем, какой layout-файл используется при смене ориентации экрана (горизонтальная/вертикальная)

Урок 5. Layout-файл в Activity. XML представление. Смена ориентации экрана.

Какой layout-файл используется в Activity

На прошлом уроке мы выяснили, что Activity читает layout-файл и отображает то, что в нем сконфигурировано. Теперь выясним, откуда Activity знает, какой именно layout-файл читать. А еще в этом уроке находится подсказка, как зарегиться на нашем форуме ;)

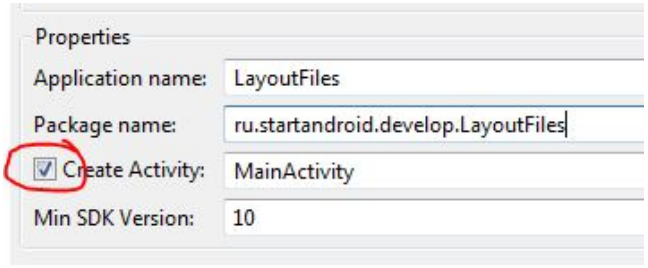
Создадим новый проект:

Project name: P0051_LayoutFiles
Build Target: Android 2.3.3
Application name: LayoutFiles
Package name: ru.startandroid.develop.LayoutFiles
Create Activity: MainActivity

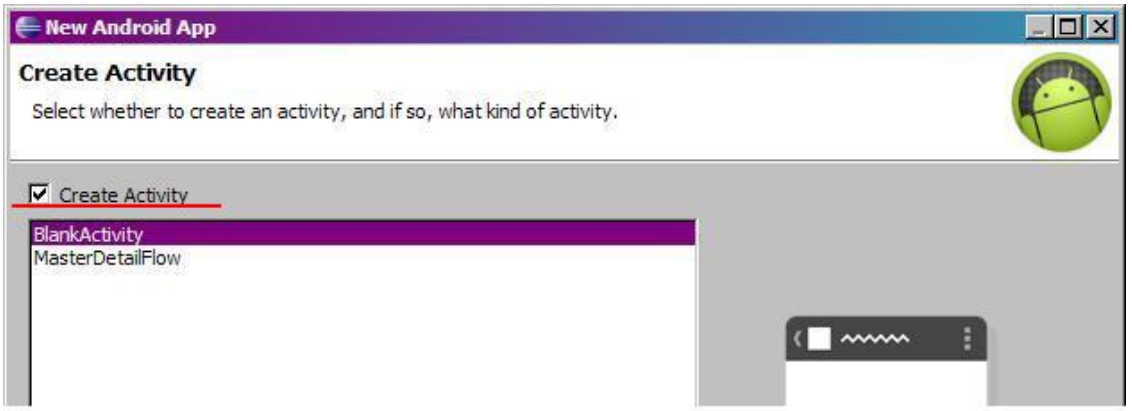
При разработке каждому Activity сопоставляется одноименный java-класс (наследник класса android.app.Activity). При запуске приложения, когда система должна показать Activity и в дальнейшем работать с ним, она будет вызывать методы этого класса. И от того, что мы в этих методах накодим, зависит поведение Activity.

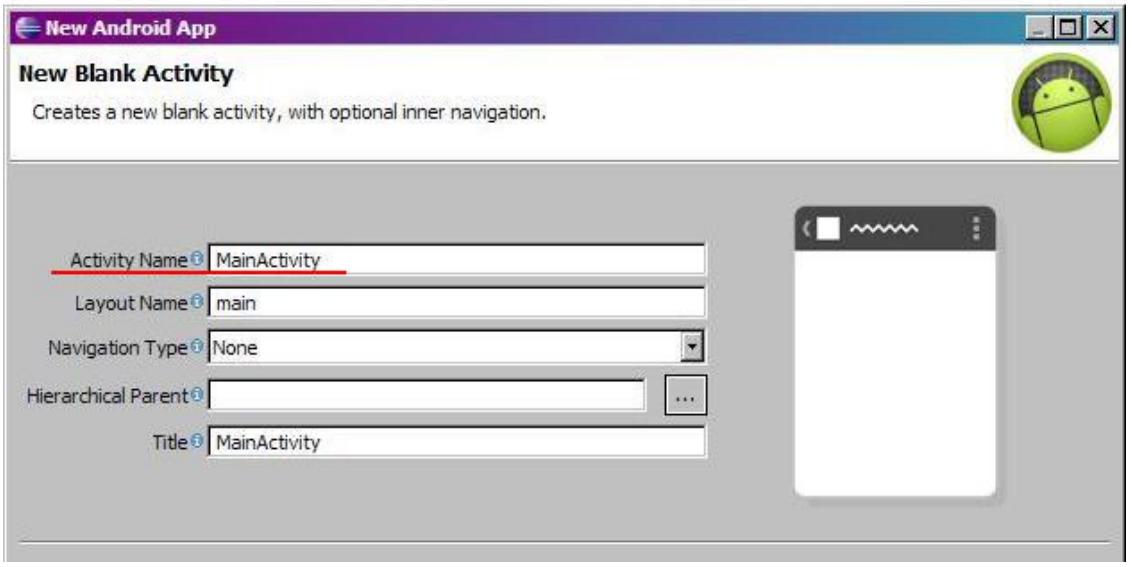
При создании проекта мы указывали, что надо создать Activity с именем **MainActivity**

При использовании старого визарда это выглядело так:



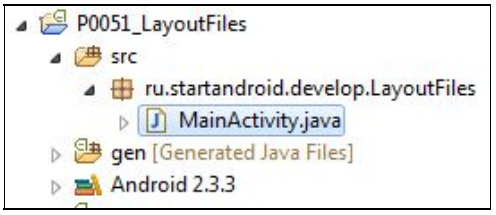
В новом визарде - чуть по другому. На одном экране галка, на другом - имя:





Мы попросили создать Activity и Eclipse создал нам соответствующий класс (в дальнейшем мы научимся их создавать самостоятельно).

Давайте посмотрим этот класс: откроем двойным кликом файл **src/ru.startandroid.develop.LayoutFiles/MainActivity.java**

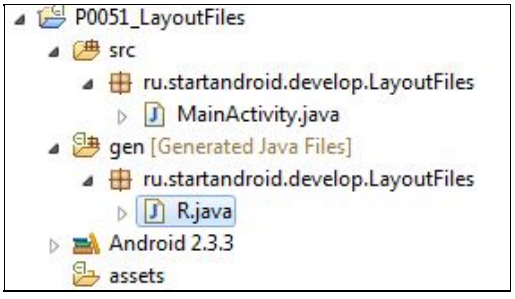


В нашем классе мы видим, что реализован метод onCreate – он вызывается, когда приложение создает и отображает Activity (на onCreateOptionsMenu пока не обращаем внимания). Посмотрим код реализации.

Первая строка:
super.onCreate(savedInstanceState); – конструктор родительского класса, выполняющий необходимые процедуры, его мы не трогаем.

Нас интересует этот код:
setContentView(R.layout.main);

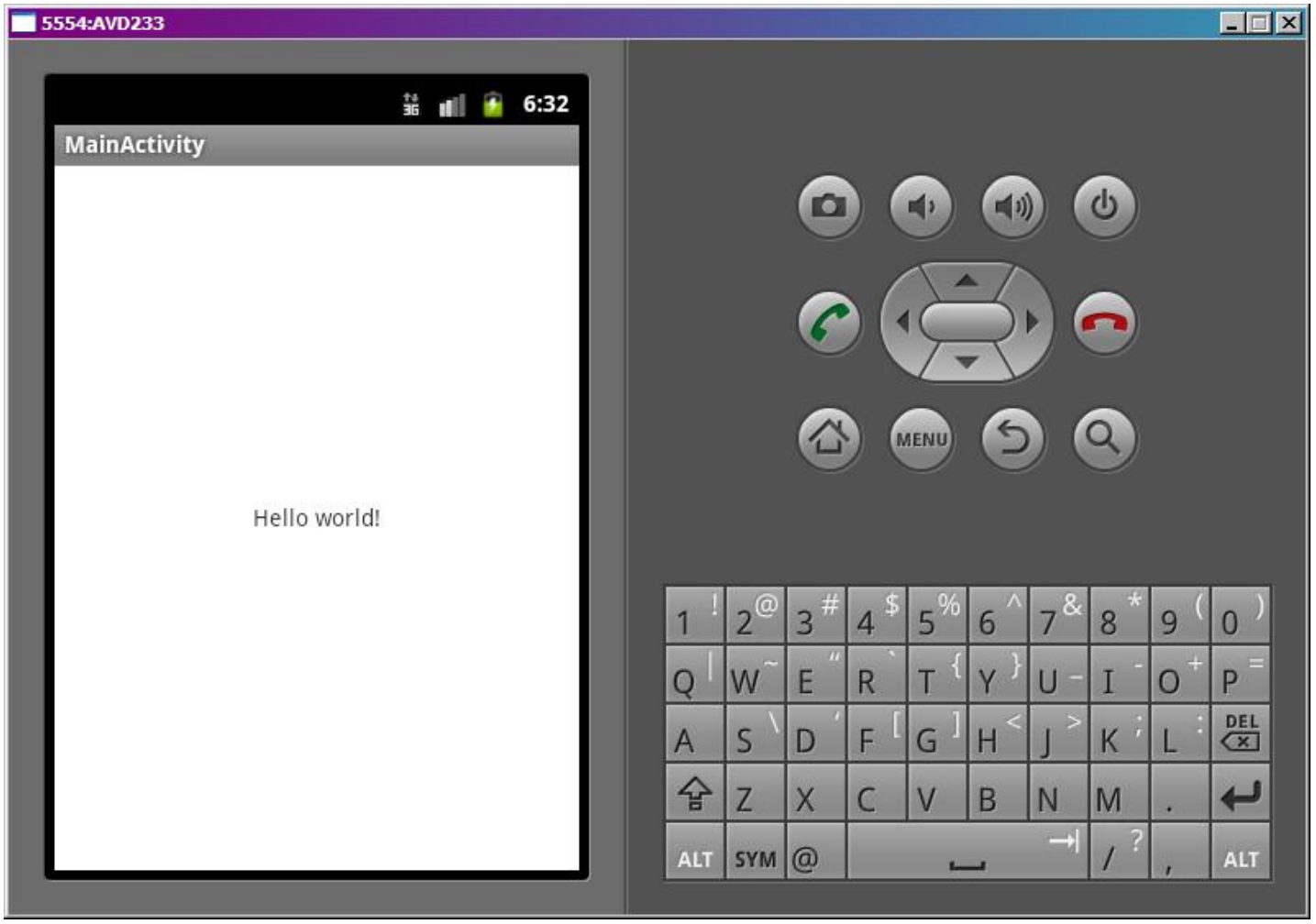
Метод setContentView(int) – устанавливает содержимое Activity из layout-файла. Но в качестве аргумента мы указываем не путь к нашему layout-файлу (res/layout/main.xml), а константу, которая является ID файла. Это константа генерируется автоматически здесь **gen/ru.startandroid.develop.LayoutFiles/R.java**.



Этот файл можно открыть и посмотреть, но менять его не стоит. В **R** классе будут храниться сгенерированные ID для всех ресурсов проекта (из папки res/*), чтобы мы могли к ним обращаться. Имена этих ID-констант совпадают с именами файлов ресурсов (без расширений).

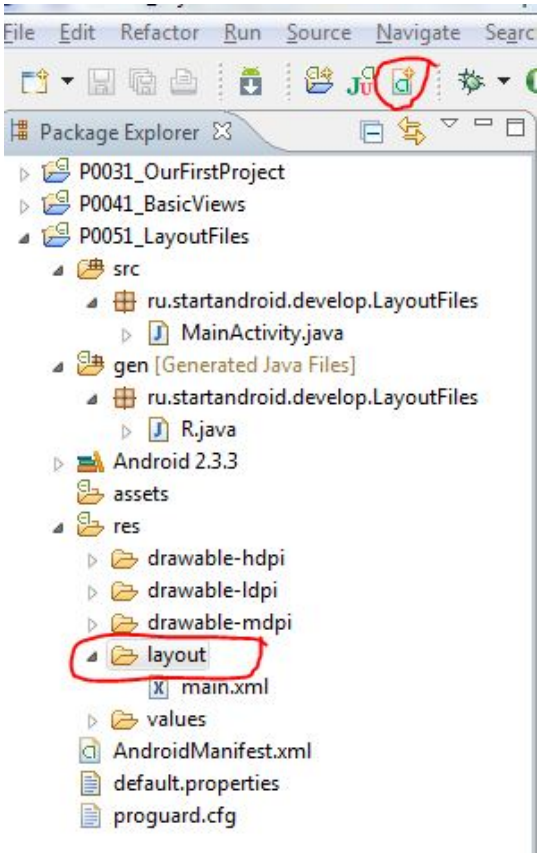
Откроем **res/layout/main.xml**, посмотрим, что там

Запустим приложение и посмотрим что оно нам покажет

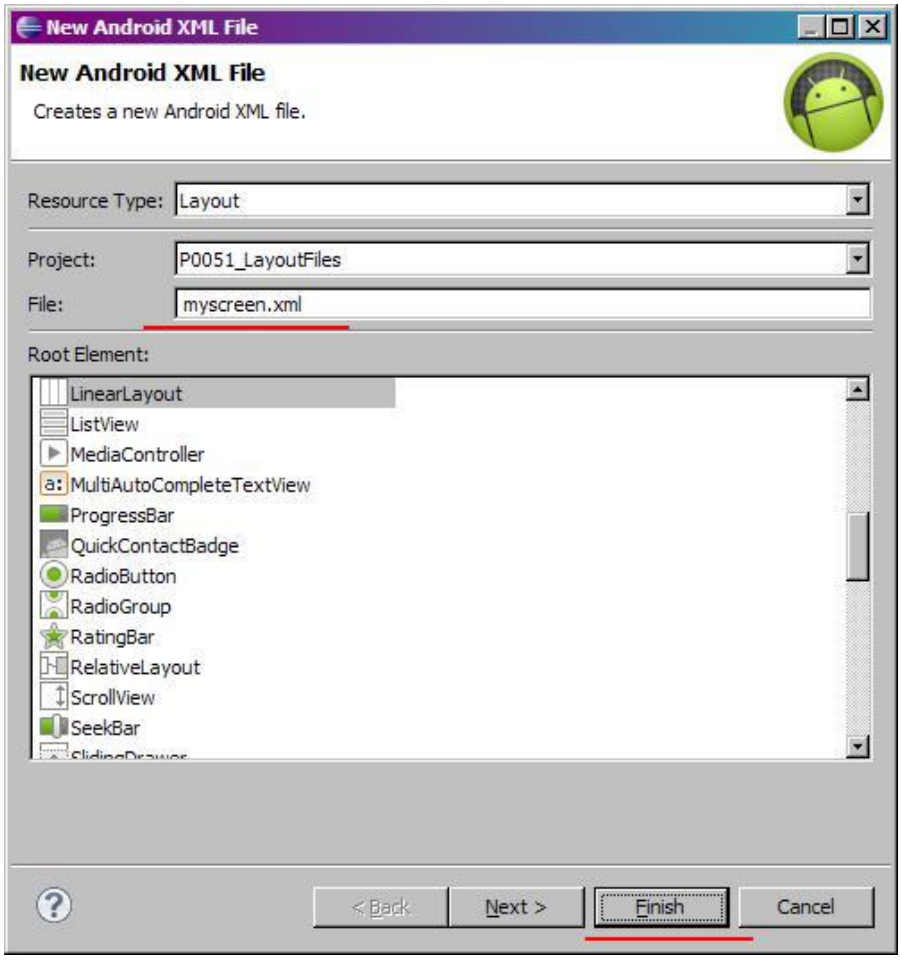


Все верно - Activity отобразил то, что прописано в **main.xml**.

Попробуем отобразить содержимое другого файла. Создадим еще один layout-файл, например **myscreen.xml**. Для этого выделим папку res/layout в нашем проекте и нажмем кнопку создания нового файла



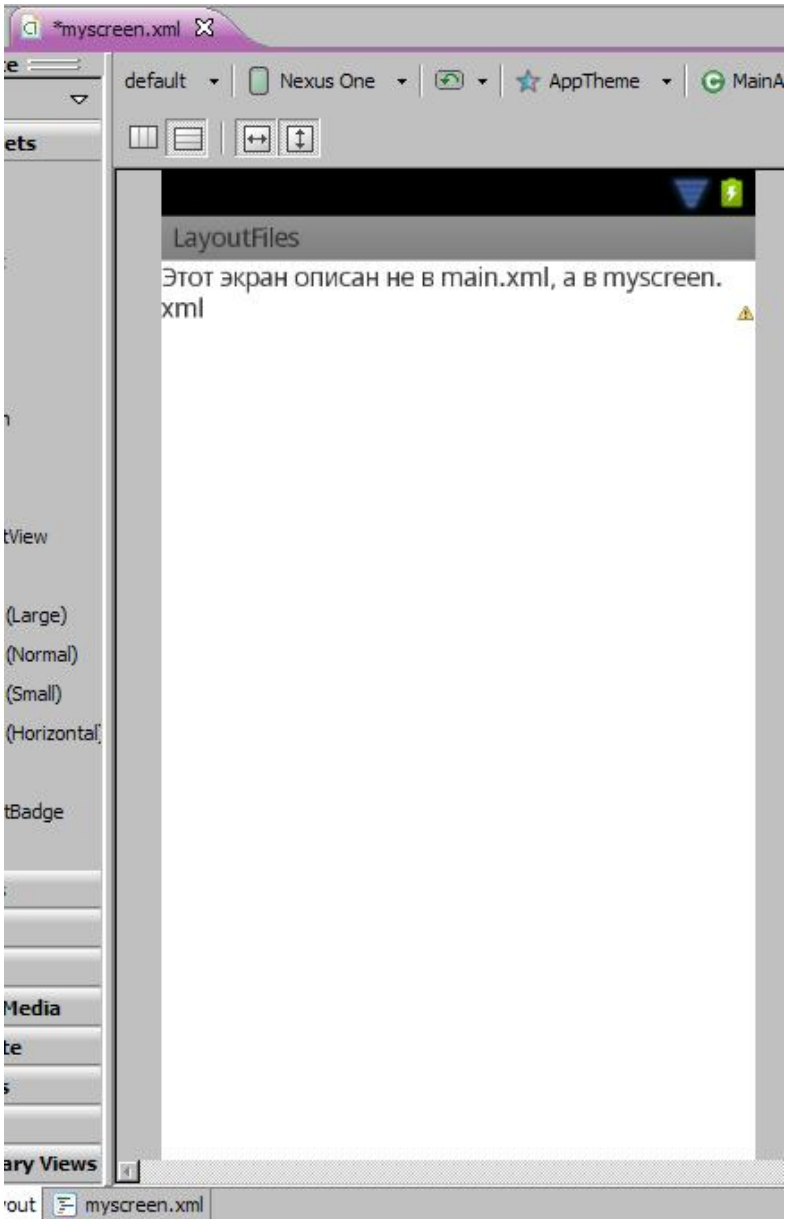
Откроется визард



В поле File вводим имя файла: myscreen.xml и жмем Finish.

Новый layout-файл должен сразу открыться. Добавим **TextView** и через **Properties** изменим его текст на: «Этот экран описан не в main.xml, а в myscreen.xml».

При этом Eclipse будет подчеркивать этот текст желтым цветом и ругаться примерно так: *[I18N] Hardcoded string "...", should use @string resource*. Это он возмущен хардкодом, который мы тут устроили. Он хочет, чтобы использовали файлы ресурсов для всех надписей. Но пока что мы это не умеем, так что игнорируем эти предупреждения.

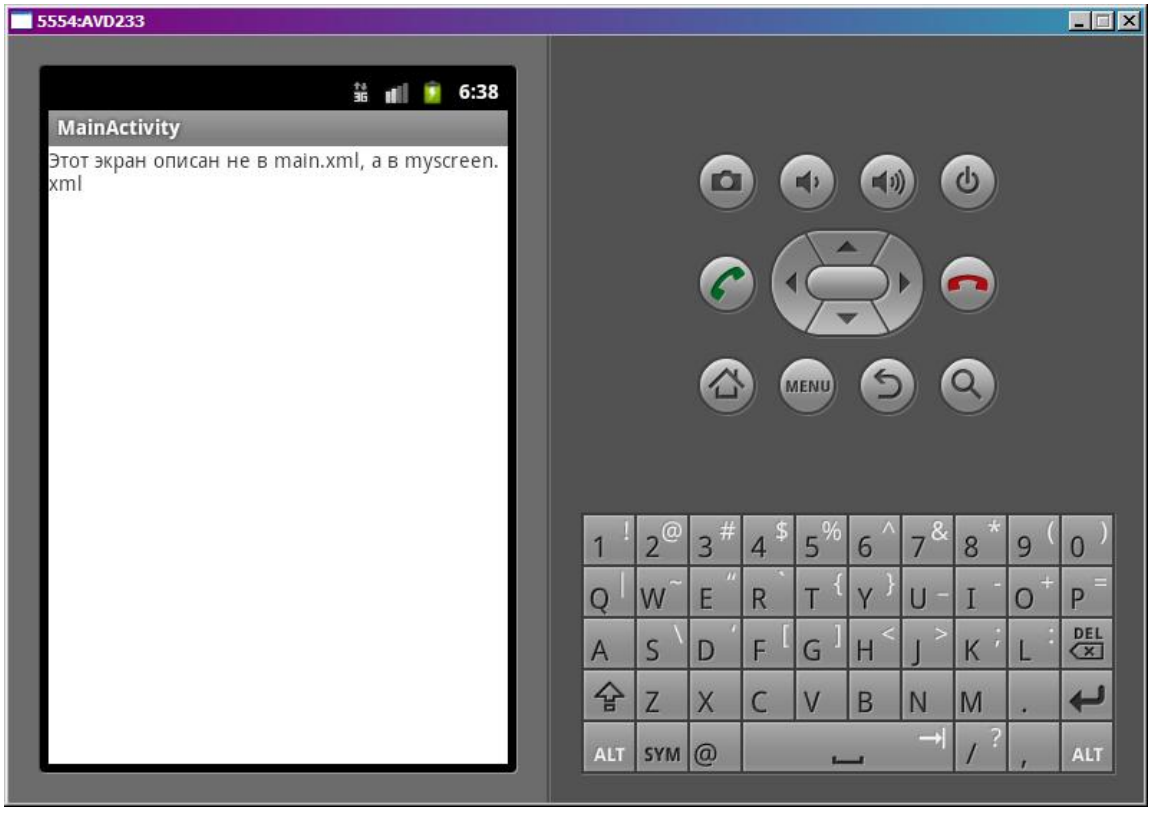


Обязательно сохраняем. Чтобы в R.java появилась новая константа для этого файла - R.layout.myscreen.

Теперь настроим так, чтобы Activity использовало новый файл **myscreen.xml**, а не **main.xml**. Откроем MainActivity.java и поменяем аргумент метода **setContentView**. Замените «**R.layout.main**», на «**R.layout.myscreen**» (ID нового layout-файла). Должно получиться так:

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.myscreen);
    }
}
```

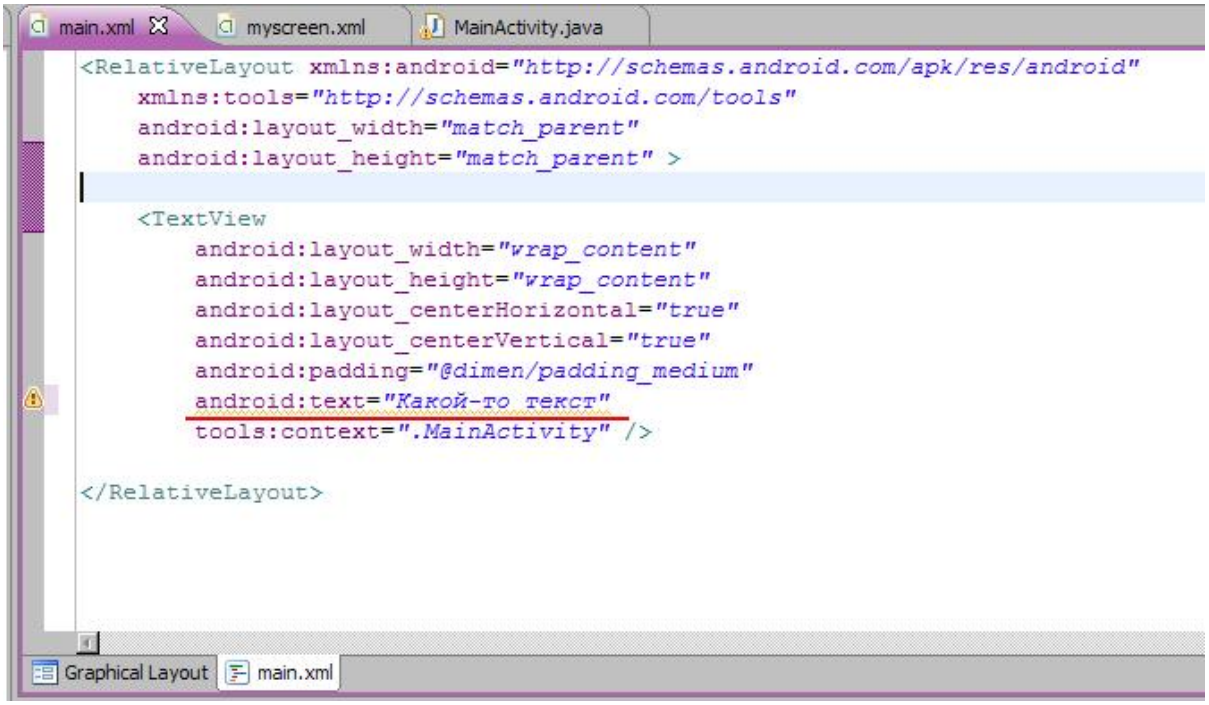
Сохраняем, запускаем приложение. Видим, что теперь оно отображает содержимое из myscreen.xml, т.к. мы явно ему это указали в методе setContentView, который выполняется при создании (onCreate) Activity



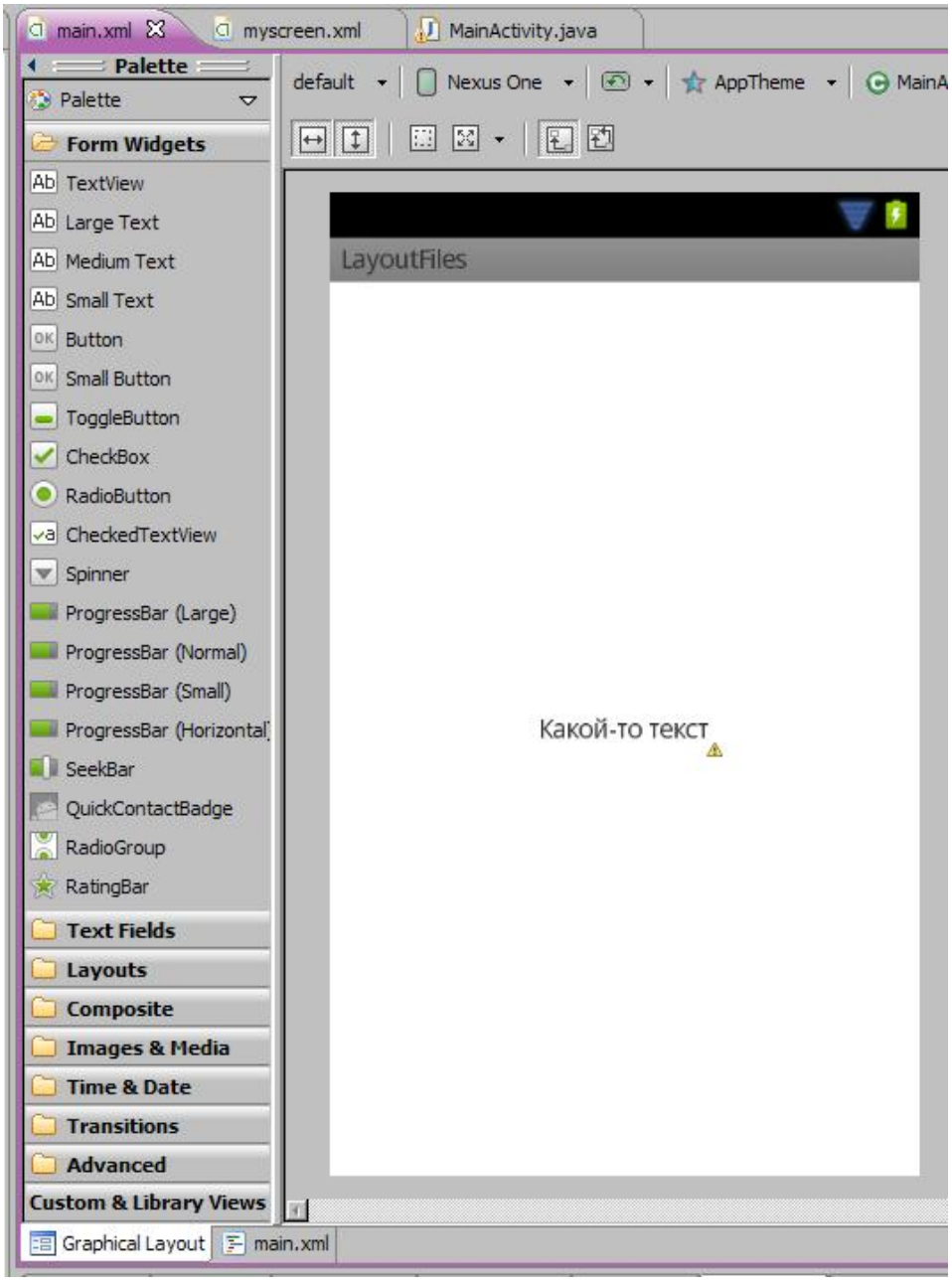
Layout-файл в виде XML

Открыв файл main.xml, вы видите его визуальное представление. Т.е. некий предпросмотр, как это будет выглядеть на экране. Снизу вы можете видеть две вкладки – **Graphical Layout** и **main.xml**. Откройте main.xml

Мы видим достаточно читабельное xml-описание всех View нашего layout-файла. Названия xml-элементов - это классы View-элементов, xml-атрибуты - это параметры View-элементов, т.е. все те параметры, что мы меняем через вкладку Properties. Также вы можете вносить изменения прямо сюда и изменения будут отображаться в Graphical Layout. Например изменим текст у TextView. Вместо ссылки на константу, вставим свой текст «Какой-то текст»



Сохраняем. Открываем Graphical Layout и наблюдаем изменения.



Обычно авторы учебников дают содержание layout-файлов именно в xml виде. Это удобно – вы можете просто скопировать фрагмент и использовать, и не надо вручную добавлять View-элементы, бегать по Properties и настраивать все руками. Я буду делать в своих проектах также.

Layout-файл при смене ориентации экрана

По умолчанию мы настраиваем **layout-файл** под **вертикальную** ориентацию экрана. Но что будет если мы повернем смартфон и включится **горизонтальная** ориентация? Давайте смотреть.

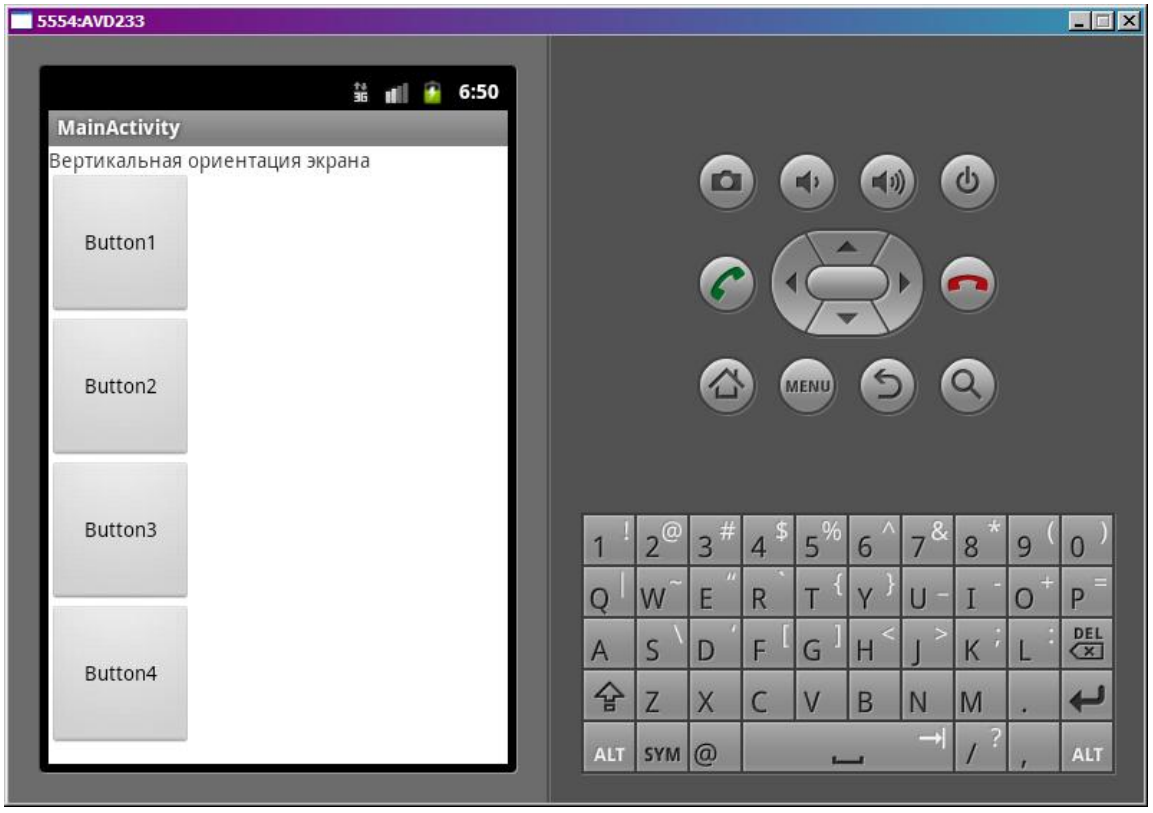
Изменим **myscreen.xml**. Добавим вертикальный ряд кнопок и изменим надпись.

xml-код (вы можете скопировать его и вставить в ваш файл):

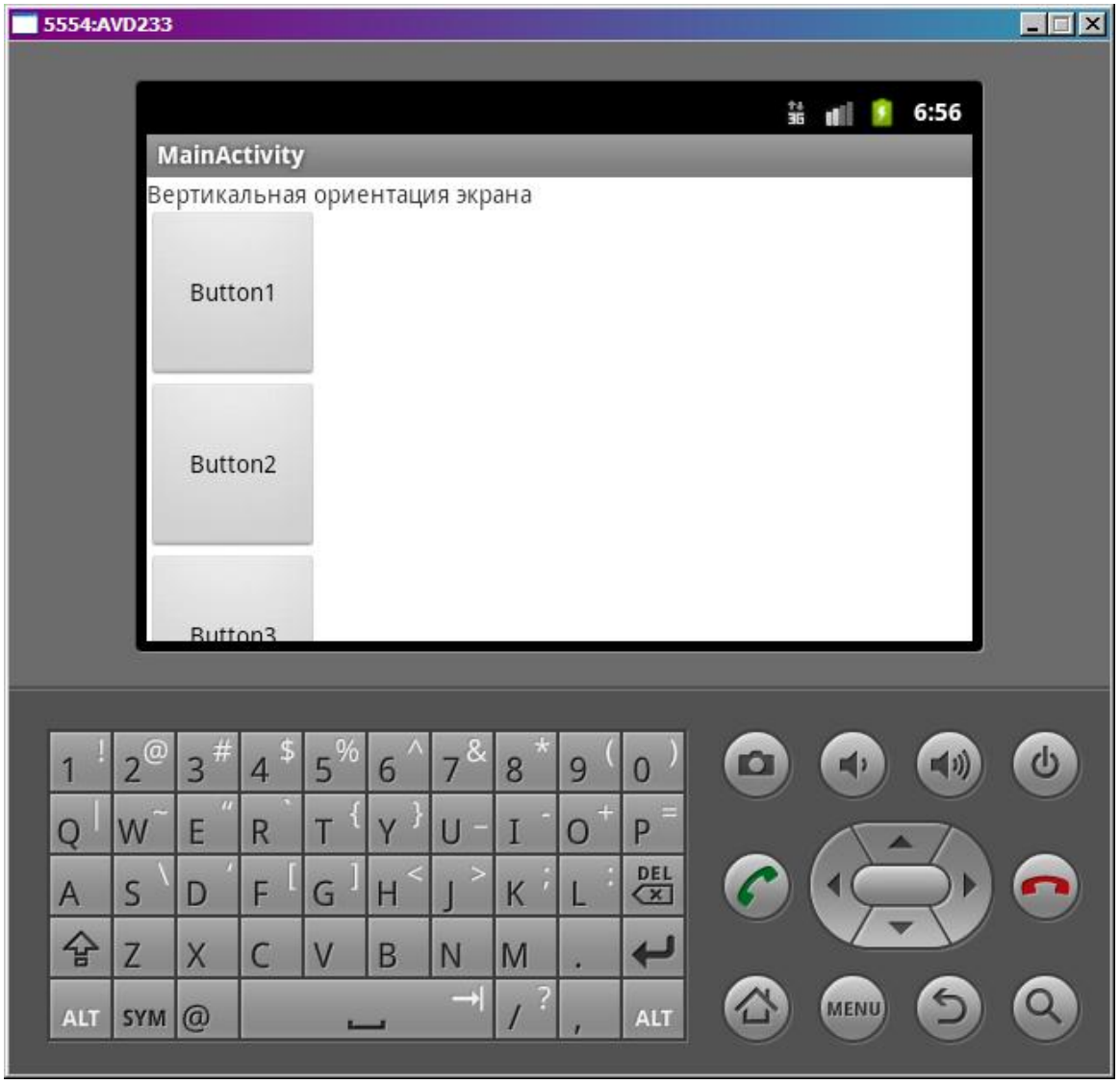
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Вертикальная ориентация экрана">
    </TextView>
    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:id="@+id/linearLayout1"
        android:orientation="vertical">
        <Button
            android:text="Button1"
            android:id="@+id/button1"
            android:layout_height="100dp"
            android:layout_width="100dp">
        </Button>
        <Button
            android:text="Button2"
            android:id="@+id/button2"
            android:layout_height="100dp"
            android:layout_width="100dp">
        </Button>
        <Button
            android:text="Button3"
            android:id="@+id/button3"
            android:layout_height="100dp"
            android:layout_width="100dp">
        </Button>
        <Button
            android:text="Button4"
            android:id="@+id/button4"
            android:layout_height="100dp"
            android:layout_width="100dp">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Обратите внимание - я добавил вертикальный **LinearLayout** и поместил в него 4 кнопки. Подробнее обсудим это на следующем уроке.

Запустим приложение. В вертикальной ориентации все ок.



Нажмем в эмуляторе CTRL+F12, ориентация сменилась на горизонтальную и наши кнопки уже не влезают в экран (у меня, кстати, эмулятор глючит и смена ориентации срабатывает не всегда)



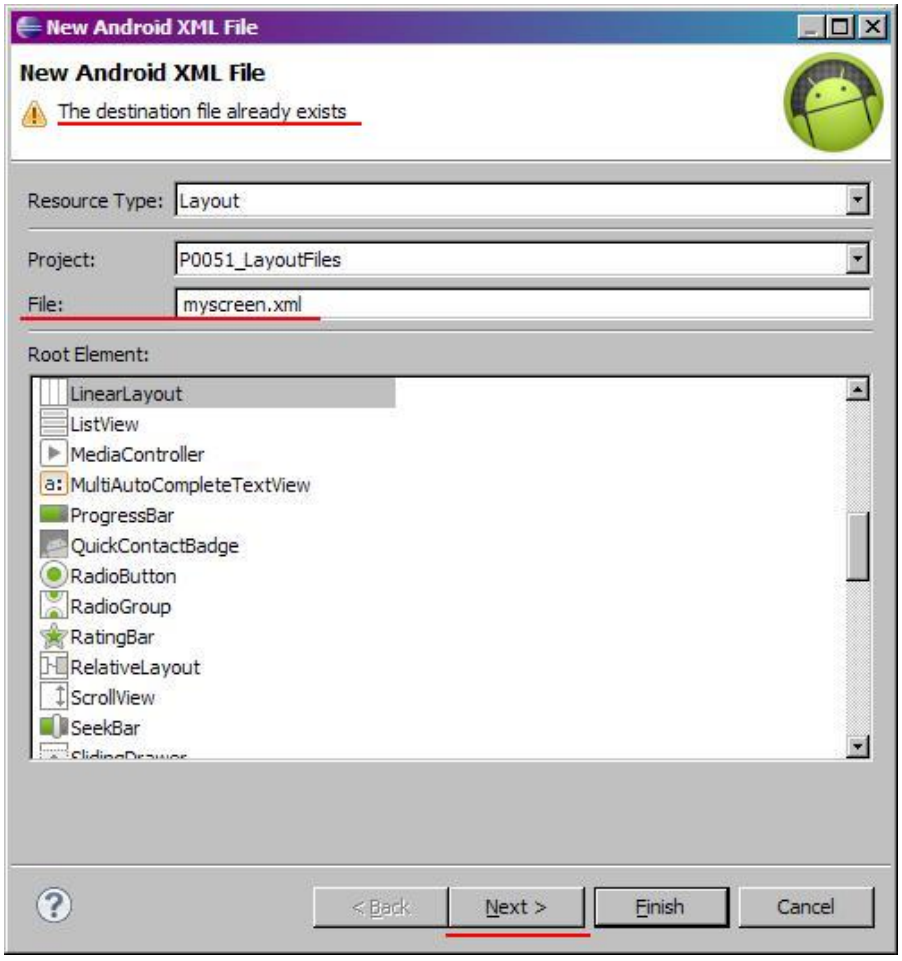
Т.е. нам необходим еще один layout-файл, который был бы заточен под горизонтальную ориентацию и в нашем случае вывел бы кнопки горизонтально.

Но как дать знать Activity, что она в вертикальной ориентации должна использовать один layout-файл, а в горизонтальной – другой? Об этом за нас уже подумали создатели Андроид. Необходимо создать папку **res/layout-land**, а в ней создать layout файл **с тем же именем**, что и основной. Этот файл будет использован в **горизонтальной** ориентации.

Создаем папку: правой кнопкой на **res**, New > Folder, Folder name = **layout-land**, жмем Finish.

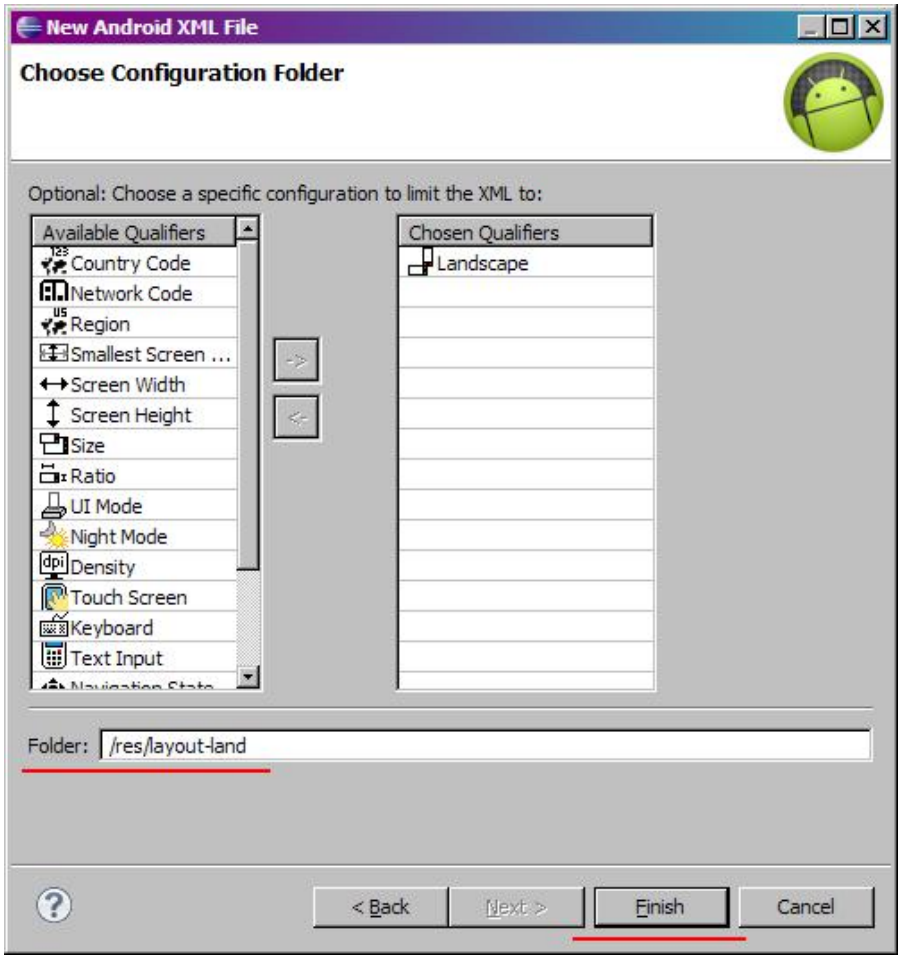
Далее создадим файл res/layout-land/myscreen.xml и настроим его под горизонтальную ориентацию. Аналогично, как и в первый раз, жмем кнопку создания файла. Откроется визард.

Вводим имя файла: **myscreen.xml**



Визард может ругнуться, что есть уже такой файл - *The destination file already exists*. Это он углядел ранее созданный файл res/layout/myscreen.xml. Нам надо ему сообщить, что новый файл предназначен для папки res/layout-land, а не res/layout. Жмем **Next**

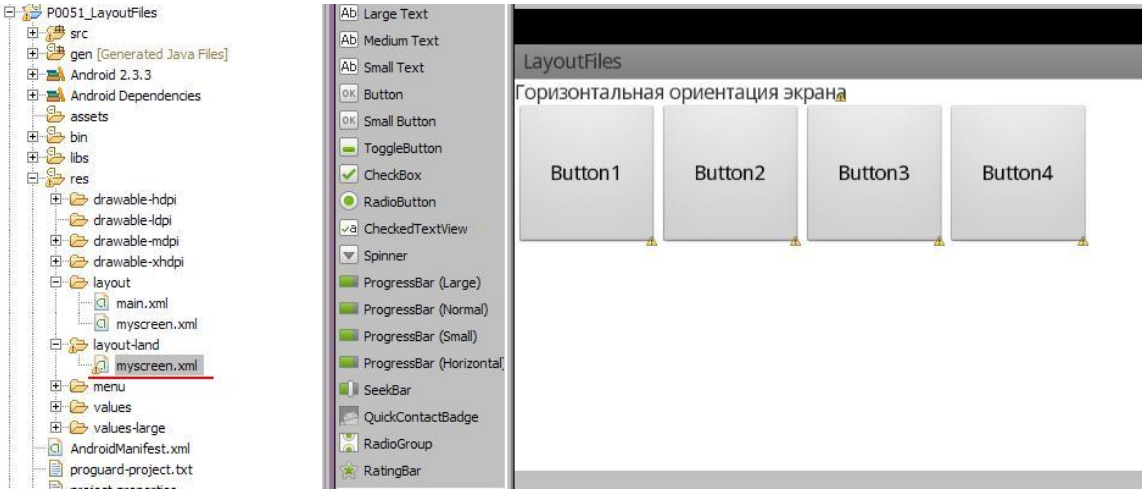
А здесь уже руками внизу допишите приставку **-land**, чтобы получилось **res/layout-land**



Как вариант, можно было руками не дописывать, а добавить из левого столбца в правый атрибут Orientation и указать ему значение Landscape. Визард все понял бы и сам дописал к пути приставку -land.

А мы сами дописали путь и визард сам добавил атрибут направо.

Жмем **Finish** и получаем готовый файл.

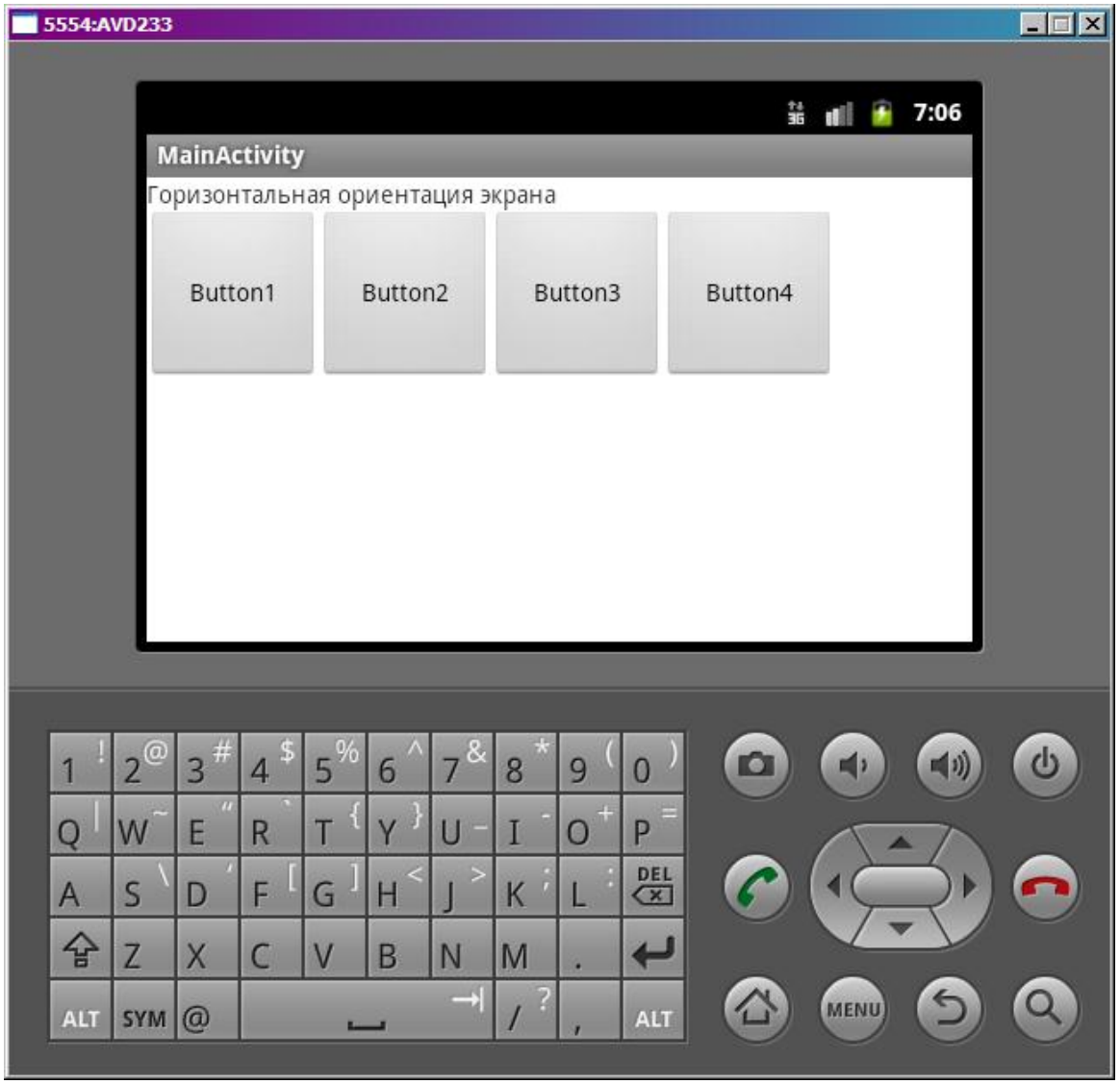


Поместите этот xml-код в файл и сохраните файл, чтобы получить такую же картинку, как выше:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Горизонтальная ориентация экрана">
```

```
    </TextView>
    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:id="@+id/linearLayout1"
        android:orientation="horizontal">
        <Button
            android:text="Button1"
            android:id="@+id/button1"
            android:layout_height="100dp"
            android:layout_width="100dp">
        </Button>
        <Button
            android:text="Button2"
            android:id="@+id/button2"
            android:layout_height="100dp"
            android:layout_width="100dp">
        </Button>
        <Button
            android:text="Button3"
            android:id="@+id/button3"
            android:layout_height="100dp"
            android:layout_width="100dp">
        </Button>
        <Button
            android:text="Button4"
            android:id="@+id/button4"
            android:layout_height="100dp"
            android:layout_width="100dp">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Запустим приложение. Activity читает layout-файл, который мы указывали в методе setContentView, т.е. myscreen.xml и отображает его содержимое. Переключим ориентацию CTRL+F12. Activity понимает, что находится в горизонтальной ориентации, ищет в папке layout-land файл myscreen.xml и использует уже его.



В этом уроке мы:

разобрали, откуда Activity знает, какой layout-файл надо читать и настроили его на чтение другого файла
рассмотрели layout-файл с другого ракурса – XML
узнали, какой layout-файл используется при смене ориентации экрана (горизонтальная/вертикальная)

В следующем уроке:

изучим основные виды layout: LinearLayout, TableLayout, RelativeLayout, AbsoluteLayout

Урок 6. Виды Layouts. Ключевые отличия и свойства.

Расположение View-элементов на экране зависит от **ViewGroup** (Layout), в которой они находятся. В этом уроке мы рассмотрим основные виды **Layout**.

LinearLayout – отображает View-элементы в виде одной строки (если он Horizontal) или одного столбца (если он Vertical). Я использовал это на прошлом уроке, когда демонстрировал использование layout-файлов при смене ориентации.

TableLayout – отображает элементы в виде таблицы, по строкам и столбцам.

RelativeLayout – для каждого элемента настраивается его положение относительно других элементов.

AbsoluteLayout – для каждого элемента указывается явная позиция на экране в системе координат (x,y)

Рассмотрим эти виды

LinearLayout (LL)

Этот вид ViewGroup по умолчанию предлагается при создании новых layout-файлов. Он действительно удобен и достаточно гибок, чтобы создавать экраны различной сложности. LL имеет свойство Orientation, которое определяет, как будут расположены дочерние элементы – горизонтальной или вертикальной линией.

Сделаем простой и наглядный пример.

Создайте проект:

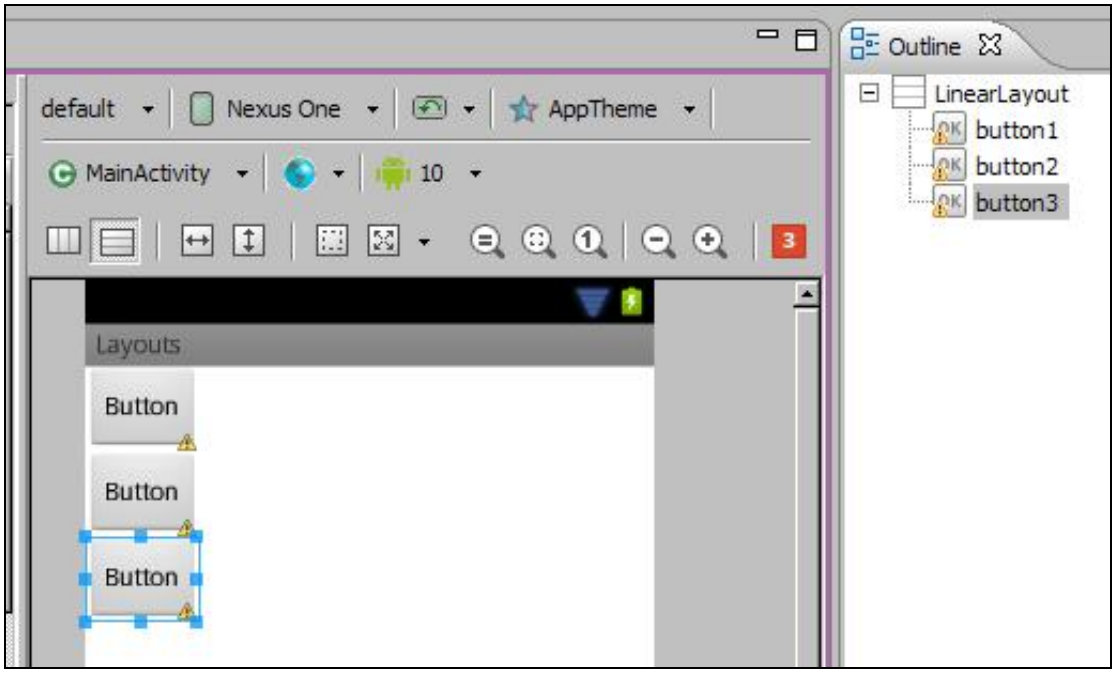
Project name: P0061_Layouts
Build Target: Android 2.3.3
Application name: Layouts
Package name: ru.startandroid.develop.layouts
Create Activity: MainActivity

Откроем layout-файл **main.xml**, и поместите в него следующий код:

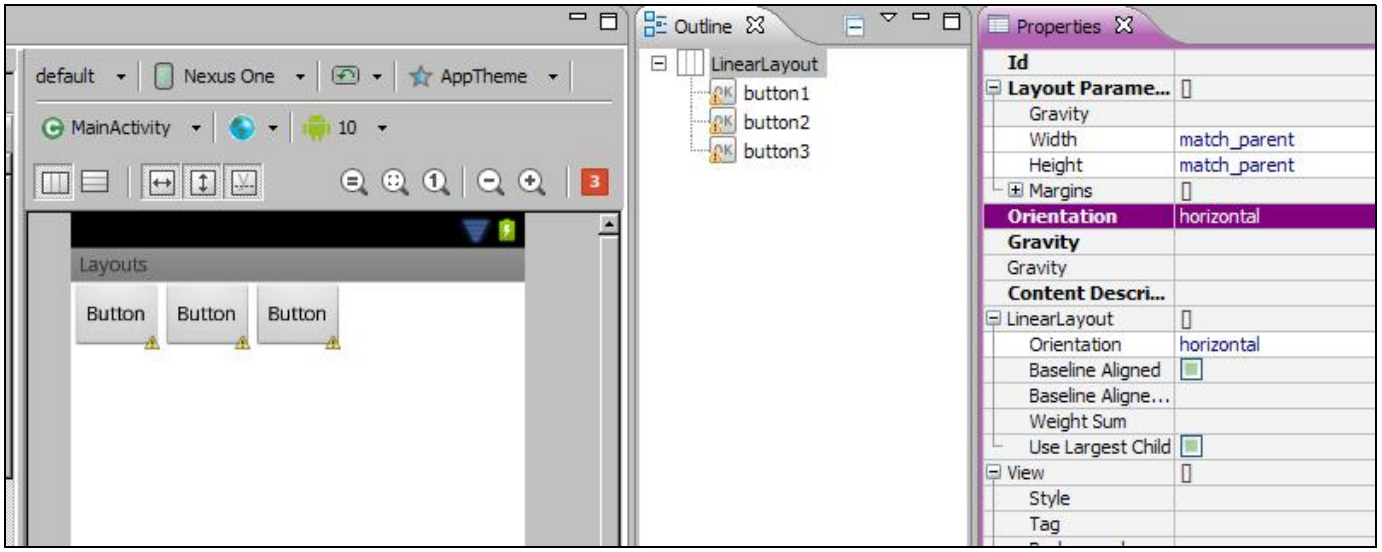
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
</LinearLayout>
```

Теперь корневой элемент у нас LinearLayout с вертикальной ориентацией.

Перетащите слева в корневой LinearLayout три кнопки. Они выстроились вертикально.

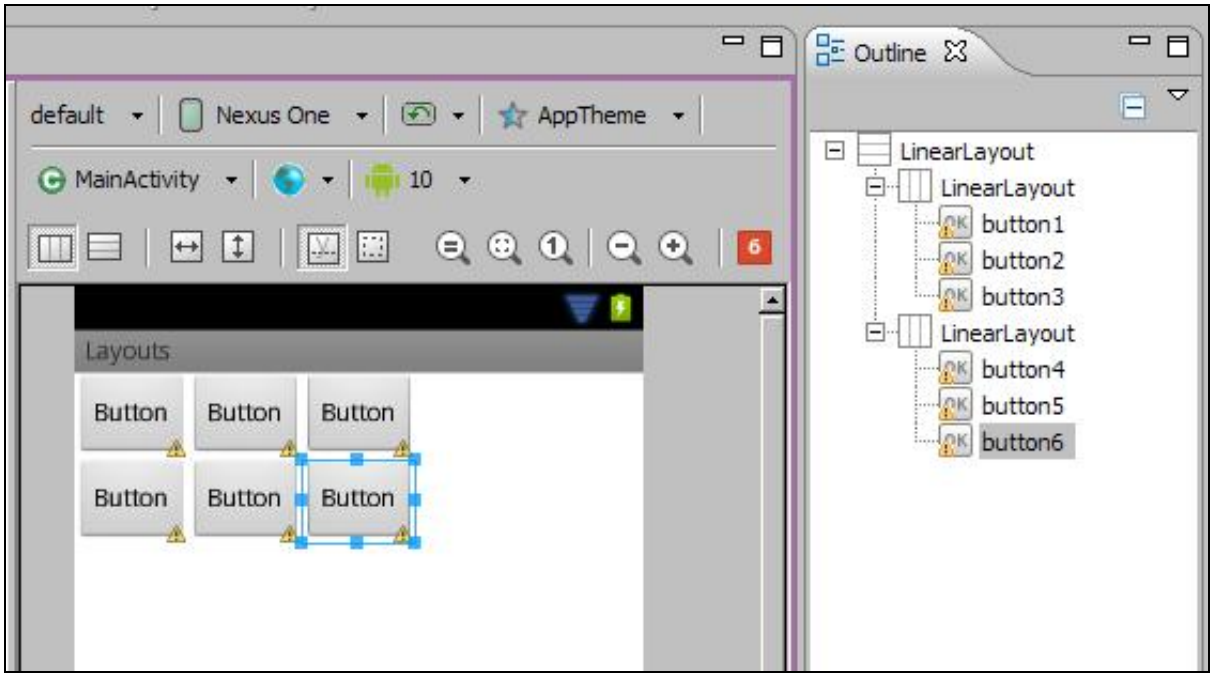


Теперь в Properties меняем для LL свойство **Orientation** на **horizontal** и сохраняем (CTRL+SHIFT+S) – кнопки выстроились горизонтально.



GroupView можно вкладывать друг в друга. Вложим в один LL два других. Удалите в main.xml все элементы (три кнопки) кроме корневого LL. Ориентацию корневого LL укажем вертикальную и добавим в него два новых горизонтальных LL. В списке элементов слева они находятся в разделе Layouts. Напоминаю, что вы можете перетаскивать элементы из списка не только на экран, но и на конкретный элемент на вкладке Outline.

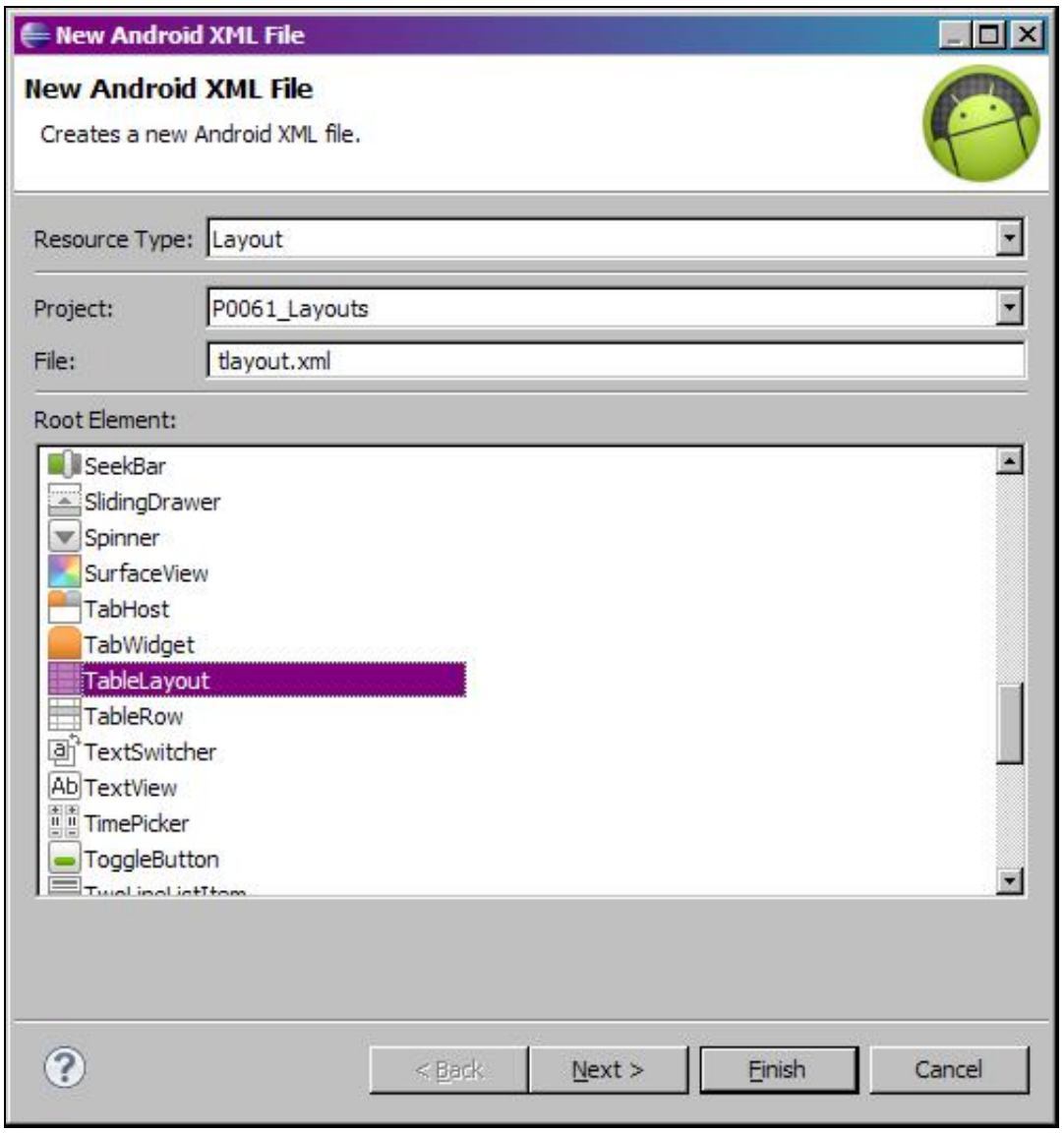
В каждый горизонтальный LL добавим по три кнопки. Получилось два горизонтальных ряда кнопок.



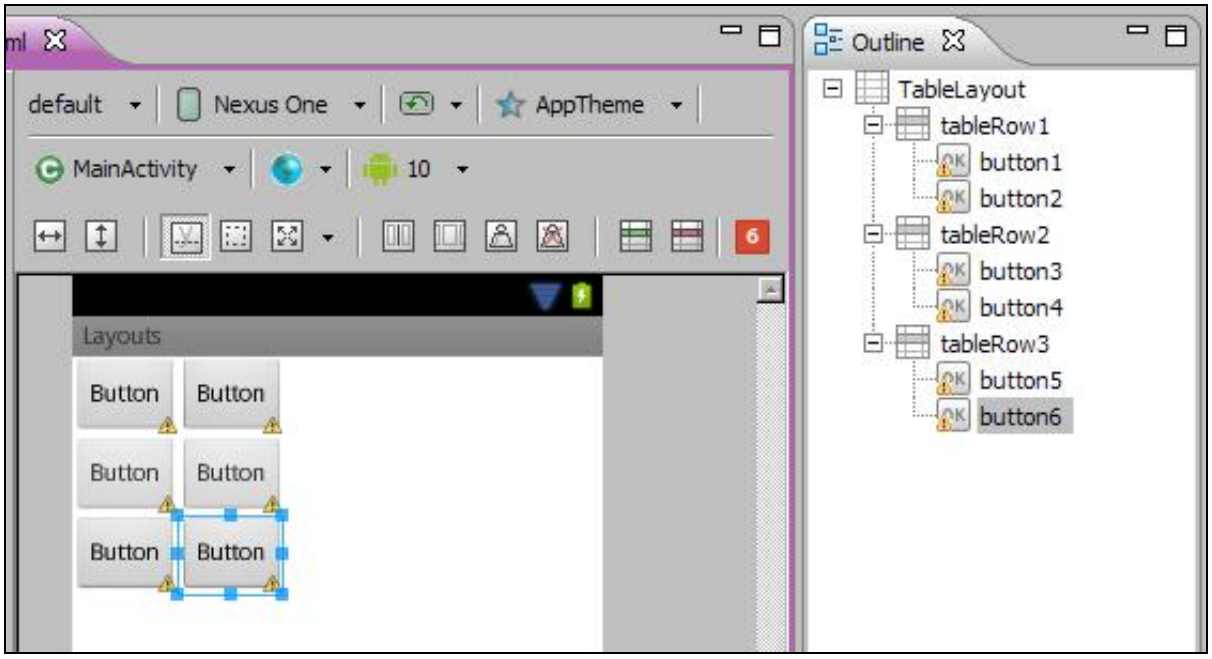
TableLayout (TL)

TL состоит из строк **TableRow** (TR). Каждая TR в свою очередь содержит View-элементы, формирующие столбцы. Т.е. кол-во View в TR - это кол-во столбцов. Но кол-во столбцов в таблице должно быть равным для всех строк. Поэтому, если в разных TR разное кол-во View-элементов (столбцов), то общее кол-во определяется по TR с максимальным кол-вом. Рассмотрим на примере.

Создадим layout-файл **tlayout.xml**. с корневым элементом TableLayout

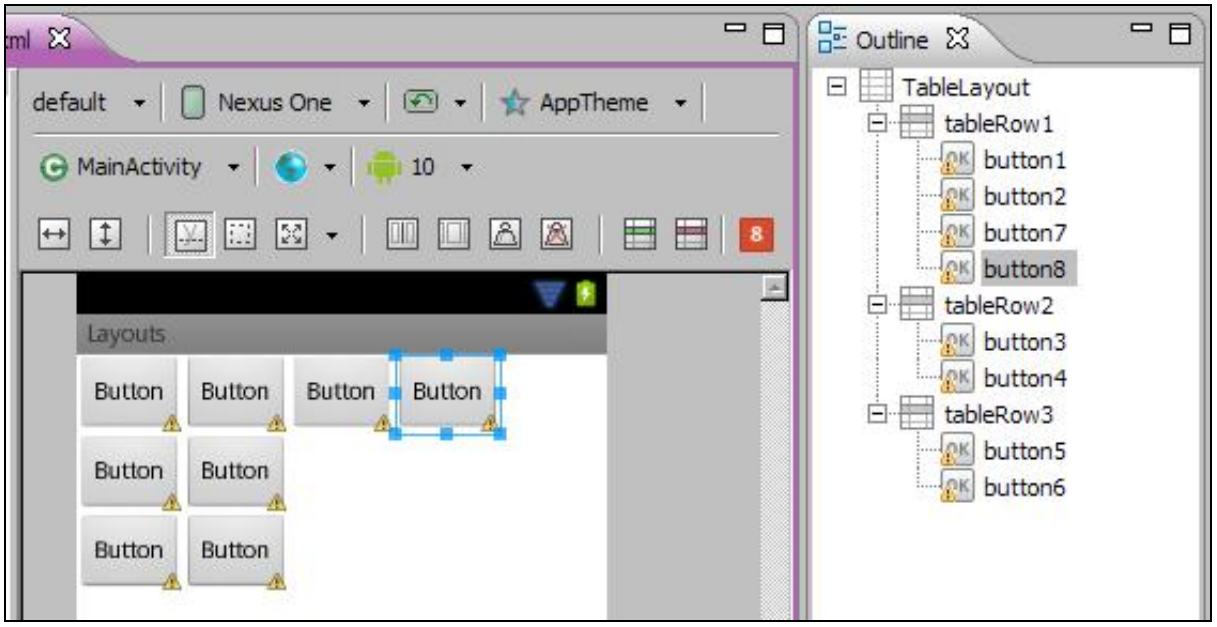


Добавим в корневой TableLayout три TableRow-строки (из раздела Layouts слева) и в каждую строку добавим по две кнопки. Результат: наша таблица имеет три строки и два столбца.

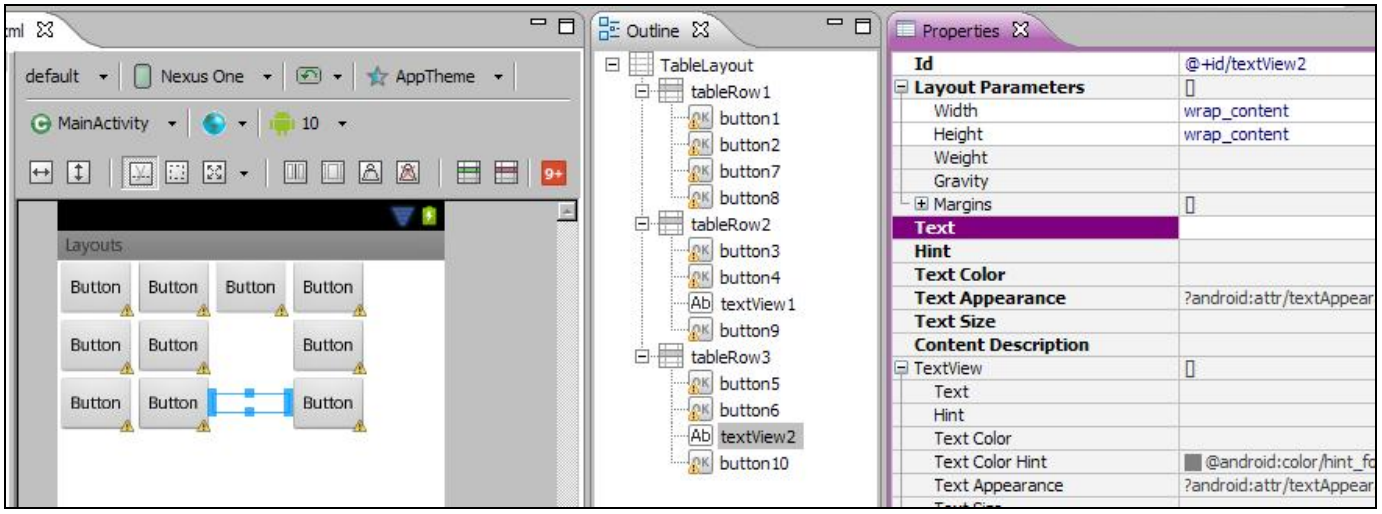


Добавим в первую строку еще пару кнопок. Кол-во столбцов для всех строк теперь равно 4, т.к. оно определяется по строке с максимальным кол-вом элементов, т.е. по первой строке. Для второй и третьей строки третий и четвертый

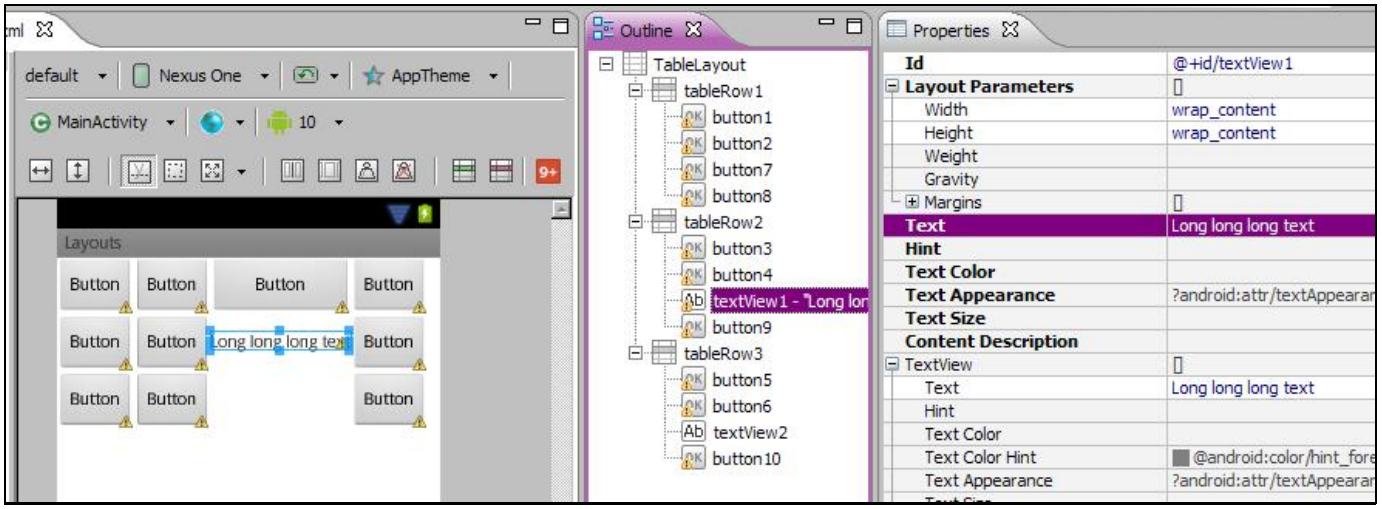
столбцы просто ничем не заполнены.



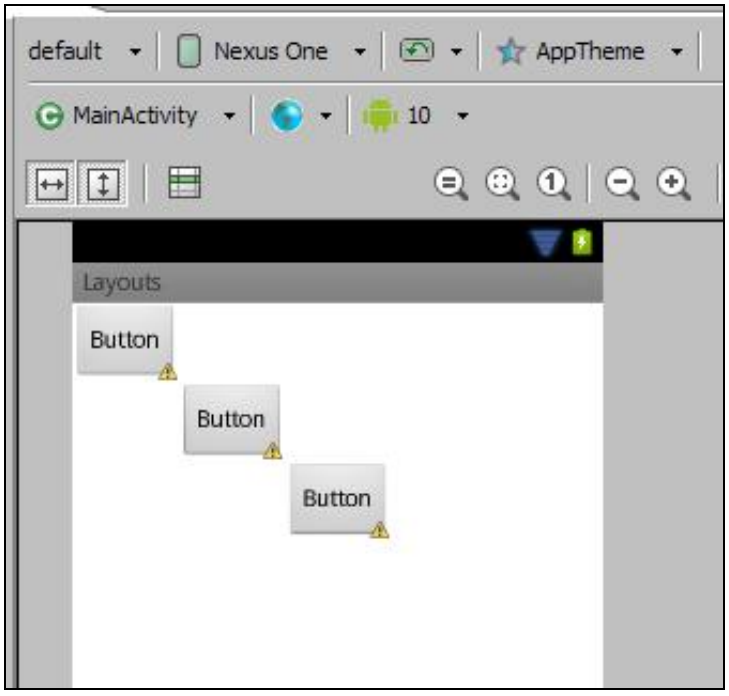
Во вторую строку добавим TextView и Button, и текст в добавленном TextView сделаем пустым. В третьей строке сделаем тоже самое. Мы видим, что эти элементы легли в третий и четвертый столбец. И т.к. TextView у нас без текста и на экране не виден, кажется что третий столбец во второй и третьей строке пустой.



Ширина столбца определяется по самому широкому элементу из этого столбца. Введем текст в один из TextView и видим, что он расширил столбец.



Я уберу элементы четвертого столбца и построю такой экран. Попробуйте сами сделать так же в качестве упражнения.



TL может содержать не только TR, но и обычные View. Добавьте, например, Button прямо в TL, а не в TR и увидите, что она растянулась на ширину всей таблицы.

RelativeLayout (RL)

В этом виде Layout каждый View-элемент может быть расположен определенным образом относительно указанного View-элемента.

Виды отношений:

1) слева, справа, сверху, снизу указанного элемента (layout_toLeftOf, layout_toRightOf, layout_above, layout_below)

2) выравненным по левому, правому, верхнему, нижнему краю указанного элемента (layout_alignLeft, layout_alignRight, layout_alignTop, layout_alignBottom)

3) выравненным по левому, правому, верхнему, нижнему краю родителя (layout_alignParentLeft,

layout_alignParentRight, layout_alignParentTop, layout_alignParentBottom)

4) выравненным по центру вертикально, по центру горизонтально, по центру вертикально и горизонтально относительно родителя (layout_centerVertical, layout_centerHorizontal, layout_centerInParent)

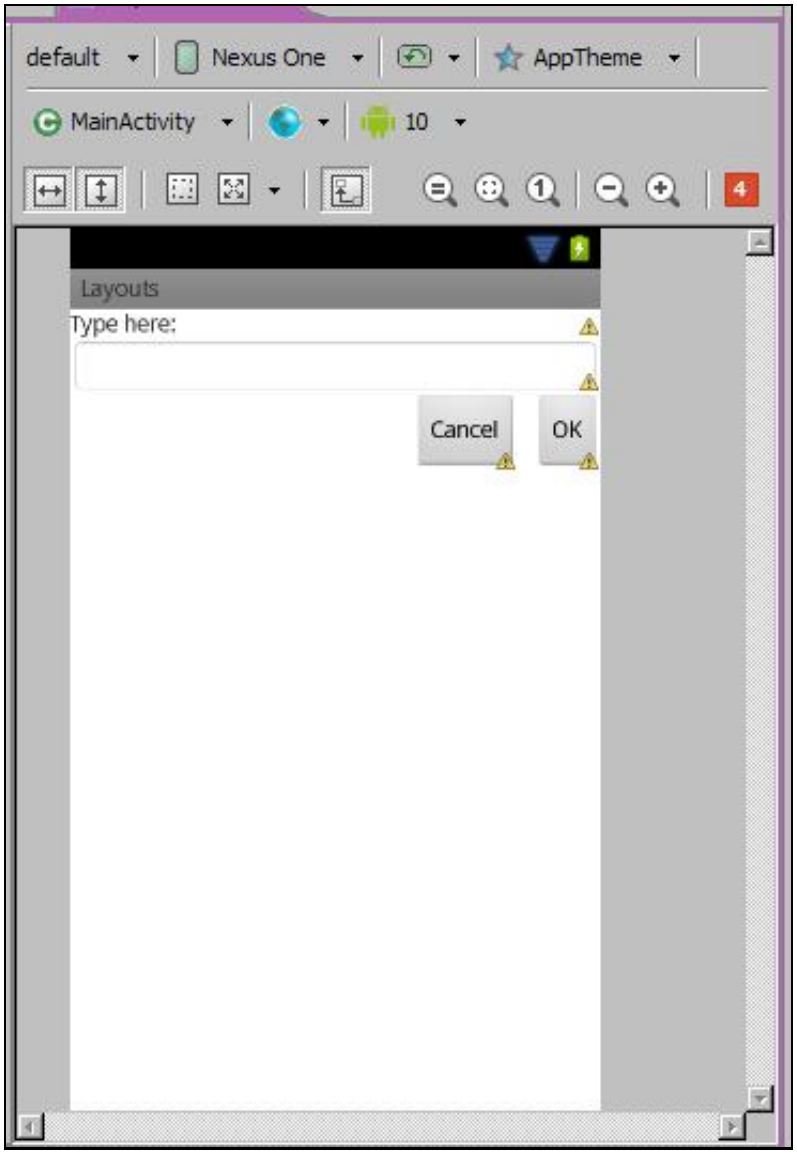
Подробно можно почитать в хелпе.

Создадим **rlayout.xml** и скопируем туда такой xml-код:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
     <TextView
          android:id="@+id/label"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:text="Type here:">
     </TextView>
     <EditText
          android:id="@+id/entry"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_below="@+id/label"
          android:background="@android:drawable/editbox_background">
     </EditText>
     <Button
          android:id="@+id/ok"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_alignParentRight="true"
          android:layout_below="@+id/entry"
          android:layout_marginLeft="10dip"
          android:text="OK">
     </Button>
     <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_alignTop="@+id/ok"
          android:layout_toLeftOf="@+id/ok"
          android:text="Cancel">
     </Button>
</RelativeLayout>
```

Здесь у нас корневой элемент - RelativeLayout.

Получился такой экран:



Нам интересен xml-код. Сразу кратко опишу незнакомые атрибуты и их значения:

android:layout_width="*match_parent*"
android:layout_height="*wrap_content*"
android:id="*@+id/entry*"

- слово **android** в названии каждого аттрибута – это namespace, я его буду опускать при объяснениях.
- id – это ID элемента,
- layout_width (ширина элемента) и **layout_height** (высота элемента) могут задаваться в асболютных значениях, а могут быть следующими: **fill_parent** (максимально возможная ширина или высота в пределах родителя) и **wrap_content** (ширина или высота определяется по содержимому элемента). В хелпе указывается, что есть еще **match_parent**. Это тоже самое, что и fill_parent. По каким-то причинам, разработчики системы решили, что название match_parent удобнее, и от fill_parent постепенно будут отказываться. А пока его оставили для совместимости. Так что запомните, что **match_parent = fill_parent** и в дальнейшем будем стараться использовать **match_parent**. Позже мы еще остановимся на этом и разберем подробнее.

Сейчас вернемся к нашим элементам. В примере мы видим TextView, EditText и два Button – OK и Cancel. Давайте подробно разберем интересующие нас атрибуты.

TextView
android:id="*@+id/label*" - ID
android:layout_width="*match_parent*" - занимает всю доступную ему ширину (хоть это и не видно на экране);
android:layout_height="*wrap_content*" - высота по содержимому;

ни к чему никак не относится

EditText
android:id="*@+id/entry*" - ID
android:layout_width="*match_parent*" - вся доступная ему ширина
android:layout_height="*wrap_content*" - высота по содержимому
android:layout_below="*@id/label*" - расположен **ниже** TextView (ссылка по ID)

Button_OK
android:id="*@+id/ok*" – ID
android:layout_width="*wrap_content*" - ширина по содержимому
android:layout_height="*wrap_content*" – высота по содержимому
android:layout_below="*@id/entry*" - расположен ниже EditText
android:layout_alignParentRight="*true*" - **выравнен по правому краю родителя**
android:layout_marginLeft="*10dip*" – имеет отступ слева (чтобы Button_Cancel был не впритык)

Button_Cancel
android:layout_width="*wrap_content*" - ширина по содержимому
android:layout_height="*wrap_content*" – высота по содержимому
android:layout_toLeftOf="*@id/ok*" - расположен **слева** от Button_OK
android:layout_alignTop="*@id/ok*" - **выравнен по верхнему краю** Button_OK

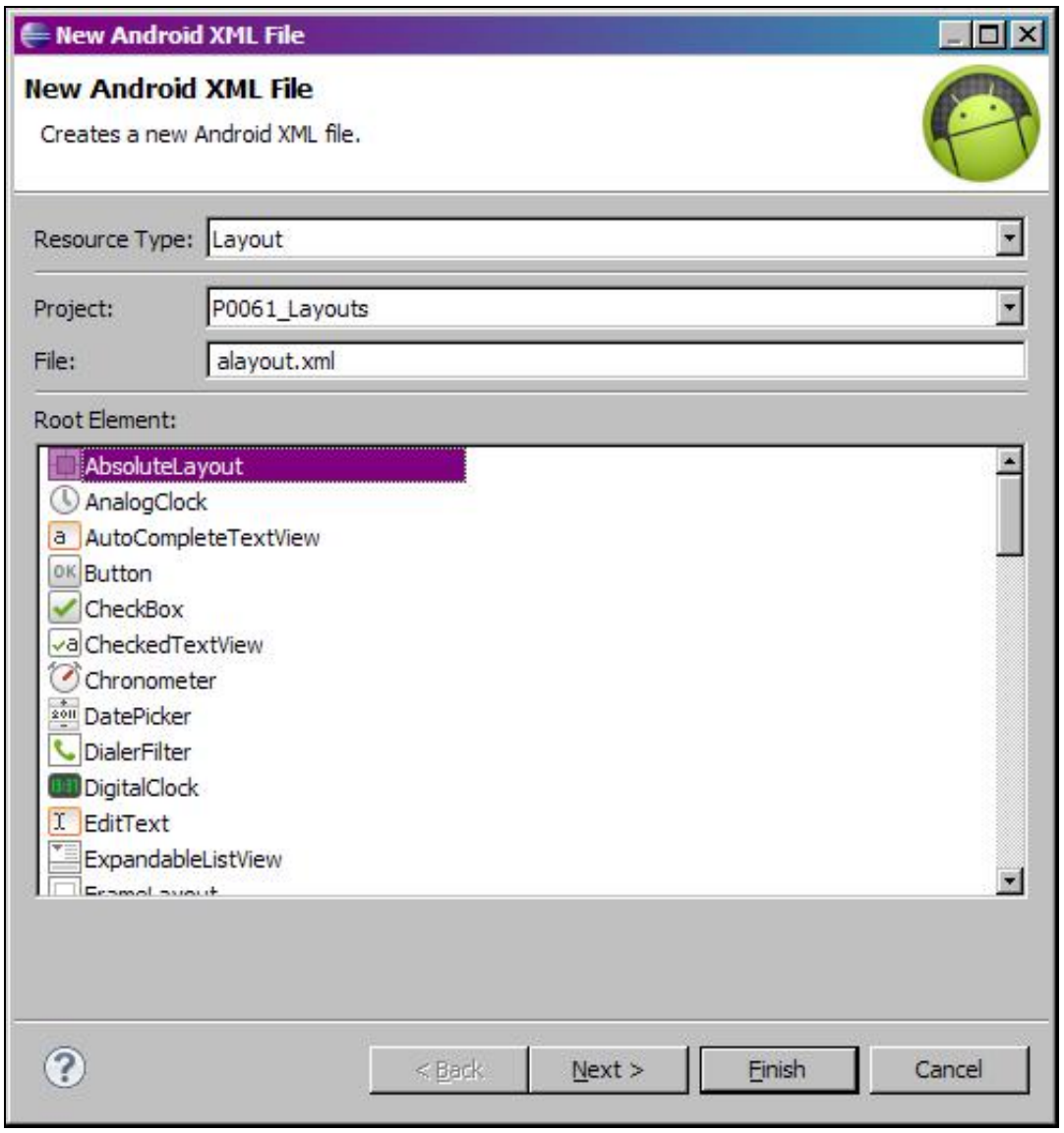
Вы можете подобавлять элементы и поэкспериментировать с их размещением.

Обратите внимание, что у View-элемента может не быть ID (android:id). Например, для TextView он обычно не нужен, т.к. они чаще всего статичны и мы к ним почти не обращаемся при работе приложения. Другое дело EditText – мы работаем с содержимым текстового поля, и Button – нам надо обрабатывать нажатия и соответственно знать, какая именно кнопка нажата. В будущем мы увидим еще одну необходимость задания ID для View-элемента.

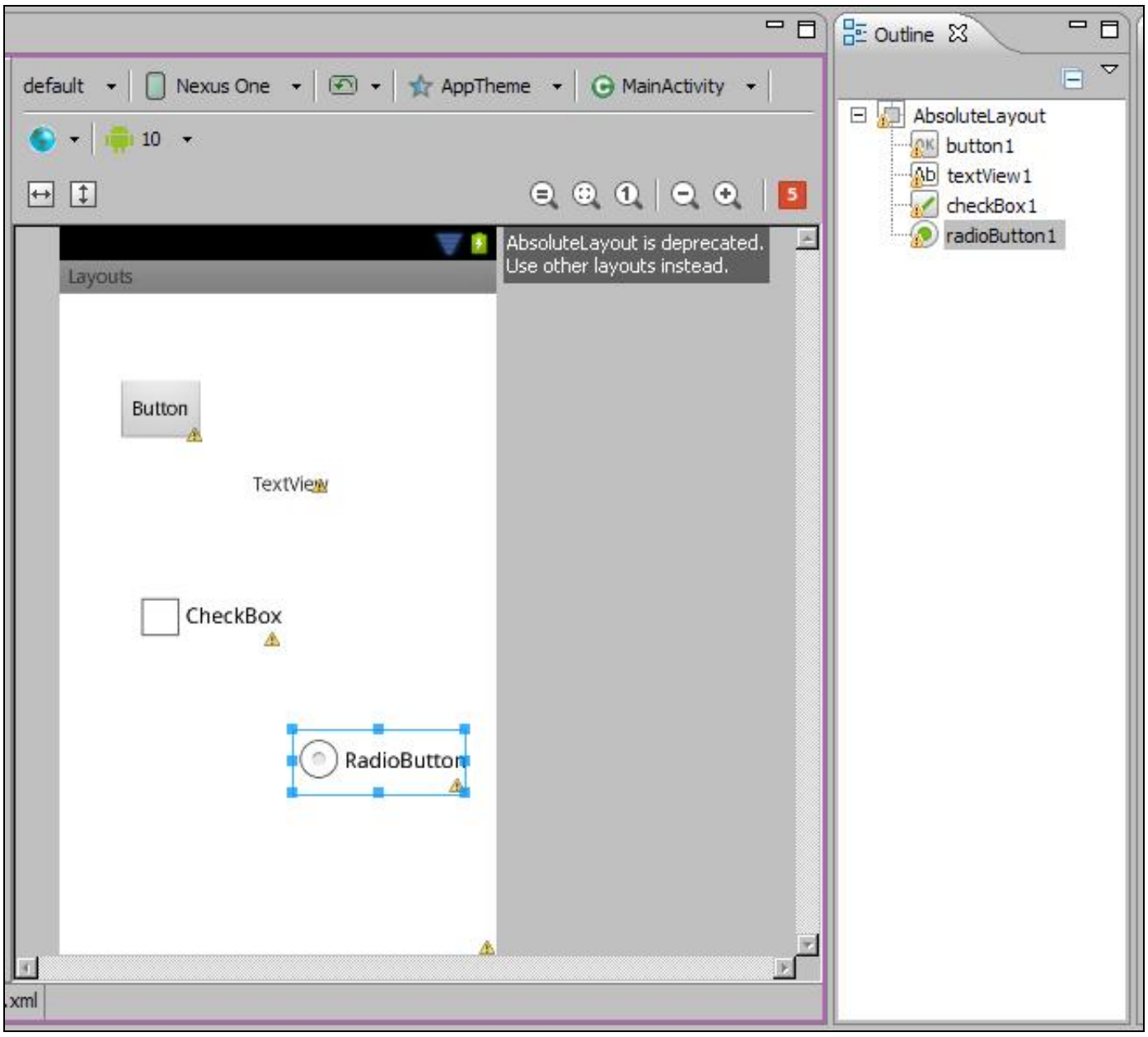
AbsoluteLayout (AL)

Обеспечивает абсолютное позиционирование элементов на экране. Вы указываете координаты для левого верхнего угла компонента.

Создадим **alayout.xml** с корневым AbsoluteLayout



Теперь попробуйте перетаскиванием подобавлять различные элементы на экран. Они не выстраиваются, как при LinearLayout или TableLayout, а ложатся там, куда вы их перетащили. Т.е. это абсолютное позиционирование.



Открываем xml-код и видим, что для задания координат используются **layout_x** и **layout_y**.

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
     <Button
           android:id="@+id/button1"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_x="42dp"
           android:layout_y="62dp"
           android:text="Button">
     </Button>
     <TextView
           android:id="@+id/textView1"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
```

```
            android:layout_x="142dp"
            android:layout_y="131dp"
            android:text="TextView">
        </TextView>
        <CheckBox
            android:id="@+id/checkBox1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_x="55dp"
            android:layout_y="212dp"
            android:text="CheckBox">
        </CheckBox>
        <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_x="171dp"
            android:layout_y="318dp"
            android:text="RadioButton">
        </RadioButton>
    </AbsoluteLayout>
```

Поначалу кажется, что это наиболее удобный и интуитивно понятный способ расположения элементов на экране - они сразу располагаются там где надо. Но это только в случае, когда вы разрабатываете для экрана с конкретным разрешением. Если открыть такое приложение на другом экране, все элементы сместятся и получится не так, как вы планировали. Поэтому этот Layout не рекомендуется использовать. И его совместимость с будущими версиями Android не гарантируется.

Есть еще много видов ViewGroup, и мы постепенно будем их осваивать. А пока нам хватит этих.

В этом уроке мы:

Рассмотрели основные виды Layout: LinearLayout, TableLayout, RelativeLayout, AbsoluteLayout

На следующем уроке:

рассмотрим подробно некоторые Layout-свойства View-элементов, которые позволяют настраивать их расположение в ViewGroup.

Урок 7. Layout параметры для View-элементов.

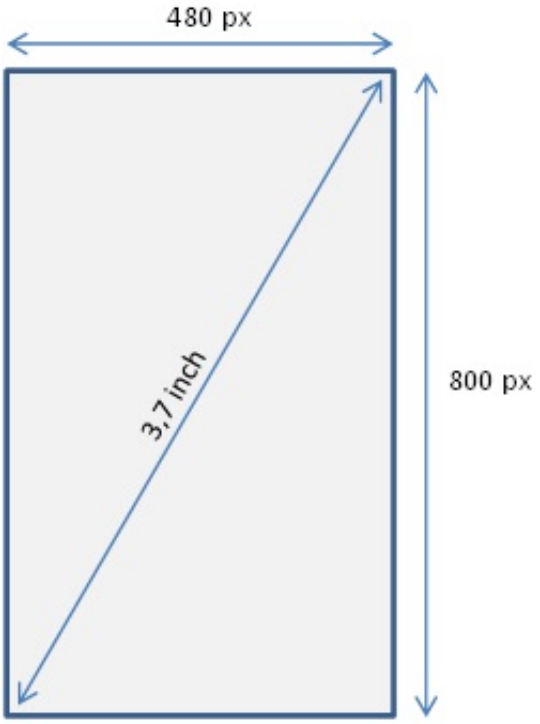
На этом уроке мы:

- разбираемся в характеристиках экрана
- рассматриваем layout параметры (высота, ширина, отступ, гравитация, вес)

Экраны

Для начала немного теории по экранам. Экран имеет такие физические характеристики как диагональ и разрешение. Диагональ – это расстояние между противоположными углами экрана, обычно измеряется в дюймах. Разрешение – кол-во точек по горизонтали и вертикали, которое экран способен отобразить, измеряется в пикселах.

Возьмем в качестве примера экран смартфона HTC Desire. Диагональ = 3,7 дюйма, разрешение = 800x480 пикселов.



Кол-во пикселов в одном дюйме называется dpi (dot per inch). Узнаем чему равно dpi в данном случае, вспомнив классику: $c^2 = a^2 + b^2$, где c – кол-во пикселей по диагонали, т.е. вмещаемое в 3,7 дюйма. a и b – стороны экрана.

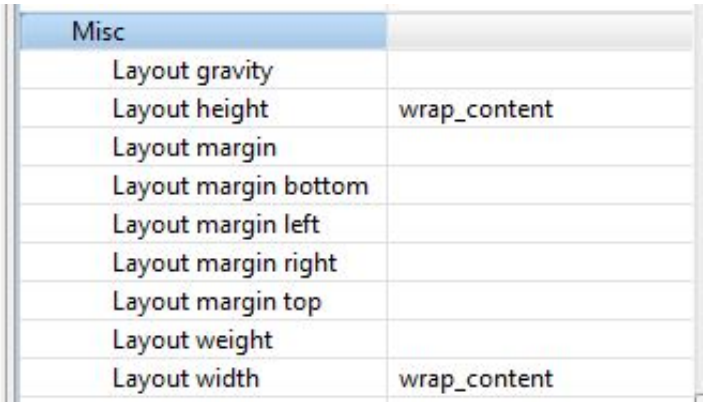
$c = 3,7 * dpi$

$(3,7 * dpi)^2 = 480^2 + 800^2$

$dpi^2 = 870400 / 13,69 = 63579$

dpi = 252. Т.е. в одном дюйме экрана помещается ряд из 252 пикселов.

Возвращаемся к теме урока. Рассмотрим подробно следующие параметры View элементов



Misc	
Layout gravity	
Layout height	wrap_content
Layout margin	
Layout margin bottom	
Layout margin left	
Layout margin right	
Layout margin top	
Layout weight	
Layout width	wrap_content

Layout width и Layout height

Про ширину (layout_width) и высоту (layout_height) мы уже немного говорили на прошлом уроке. Мы можем указывать для них абсолютные значения, а можем использовать константы. Разберем подробнее эти возможности.

АБСОЛЮТНЫЕ ЗНАЧЕНИЯ:

Используются следующие единицы измерения (ЕИ):

dp или **dip** - Density-independent Pixels. Абстрактная ЕИ, позволяющая приложениям выглядеть одинаково на различных экранах и разрешениях.

sp - Scale-independent Pixels. То же, что и dp, только используется для размеров шрифта в View элементах

pt - 1/72 дюйма, определяется по физическому размеру экрана. Эта ЕИ из типографии.

px – пиксел, не рекомендуется использовать т.к. на разных экранах приложение будет выглядеть по-разному.

mm – миллиметр, определяется по физическому размеру экрана

in – дюйм, определяется по физическому размеру экрана

Подробней о различиях и соотношениях между этими ЕИ вы можете прочесть в этом материале сайта.

КОНСТАНТЫ

match_parent (fill_parent) – означает, что элемент займет всю доступную ему в родительском элементе ширину/высоту.

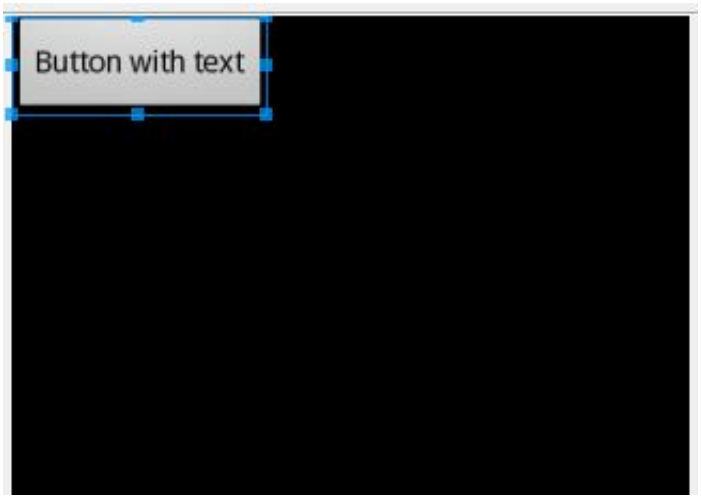
wrap_content – ширна/высота элемента будет определяться его содержимым

Создадим проект:

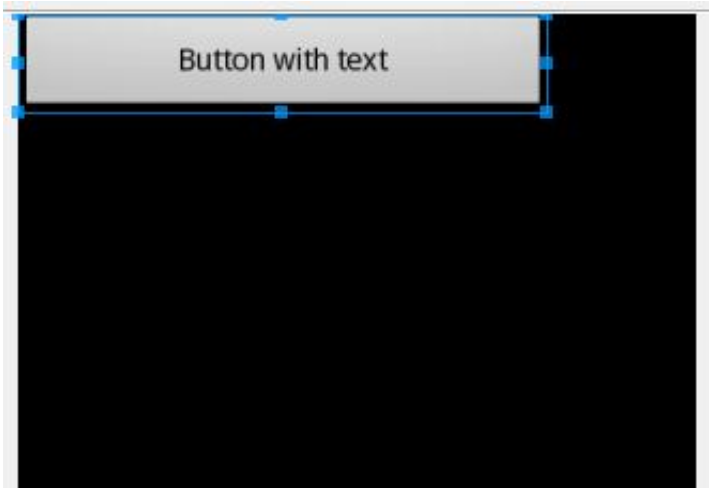
Project name: P0072_LayoutProp
Build Target: Android 2.3.3
Application name: LayoutProp
Package name: ru.startandroid.develop.layoutprop
Create Activity: MainActivity

Открываем **main.xml**. Настроим корневой **LinearLayout** на горизонтальную ориентацию, удалим **TextView**, и добавим **Button** с шириной и высотой равной **wrap_content**. Она отображается на экране и ее ширина соответствует тексту на ней.

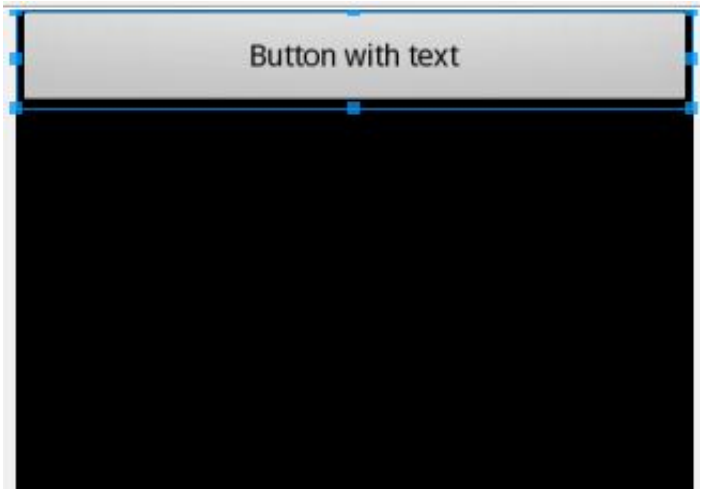
Изменим текст с «Button» на «Button with text», сохраним и посмотрим на экран.



Кнопка стала шире, т.к. ширина определяется по содержимому. Если же мы сейчас явно укажем ей ширину 250 dp, то кнопка растянется независимо от содержимого.



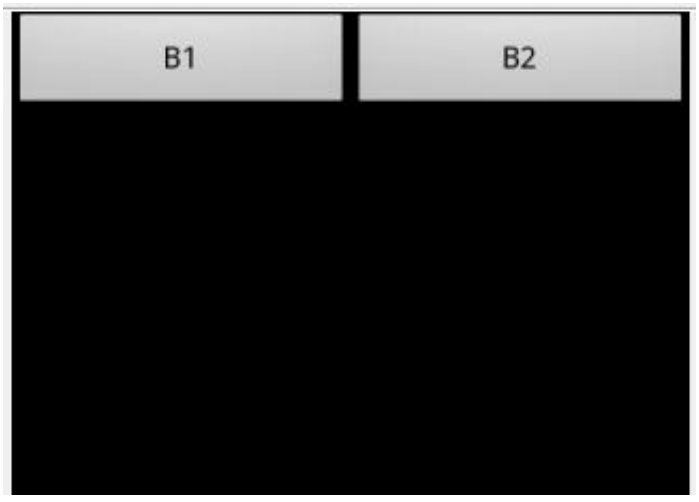
Теперь сделаем ширину равной **match_parent**. Кнопка растянулась на всю **ширину родителя**, т.е. LinearLayout. А LinearLayout в свою очередь занимет всю ширину экрана.



Если у нас родитель содержит несколько элементов и мы хотим, чтобы они заняли все пространство необходимо использовать параметр **Layout weight – вес**. Свободное пространство распределяется между элементами пропорционально их weight-значениям.

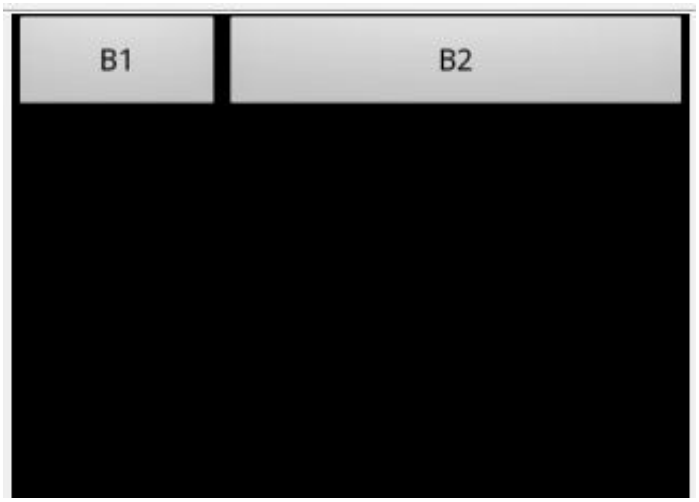
Изменим текст нашей кнопки на **B1** и добавим ей соседа по LinearLayout – вторую кнопку с текстом **B2**. Ширину для обоих поставьте **wrap_content**

Займемся дележом. Если мы хотим, чтобы кнопки поделили пространство родителя **поровну** – то для обеих укажем **weight** = 1. В этом случае кнопки равны по ширине.

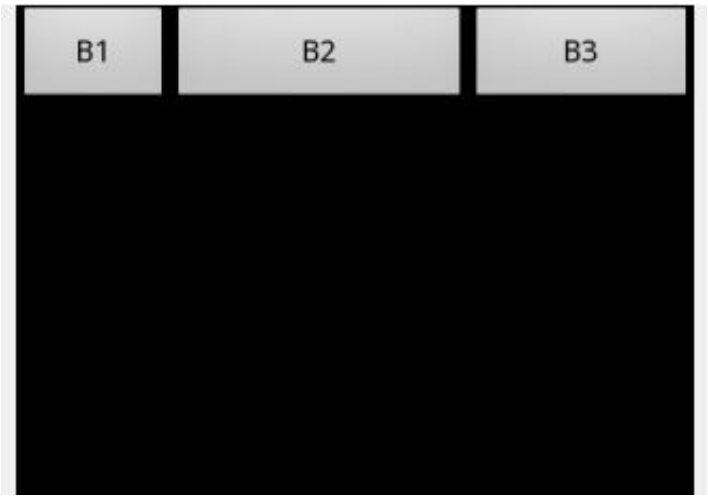


Обратите внимание, что не используются единицы измерения, указываются просто числа.

Если нужно, чтобы B1 занимала четверть, а B2 три четверти свободного пространства, то проставляем **weight** = 1 для **B1** и **weight** = 3 для **B2**.



Кол-во элементов может быть любым. Добавим еще кнопку с текстом **B3**, **weight** = 2 и **width** = wrap_content.

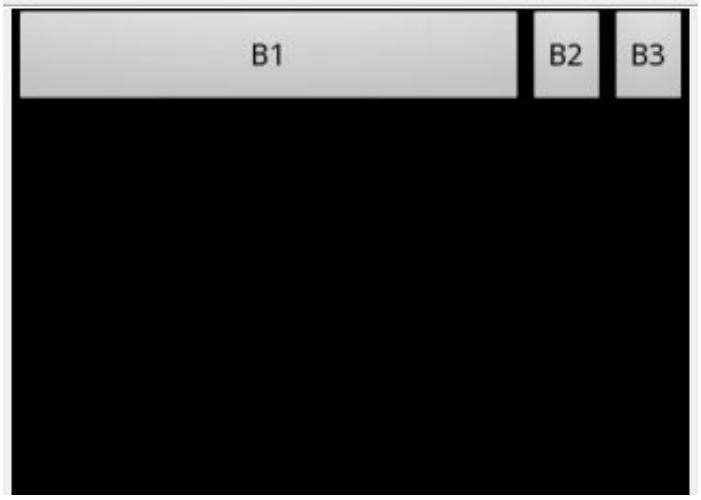


xml-код получившегося экрана:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <Button
        android:id="@+id/button1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="B1"
        android:layout_weight="1">
    </Button>
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="B2"
            android:layout_weight="3">
        </Button>
        <Button
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:id="@+id/button3"
            android:text="B3"
            android:layout_weight="2">
        </Button>
</LinearLayout>
```

Теперь для **B2** и **B3** укажите **weight** = 0. Они больше не претендуют на свободное пространство и занимают ширину по содержимому, а **B1** забирает все себе.



Разумеется, все выше сказанное применимо и для параметра высоты - **height**.

При использовании **weigth** вы можете указать значение **height** или **width** = **0dp**. В этом случае не будет учитываться содержимое элементов и результат будет более соответствующий коэффициентам веса.

Layout gravity

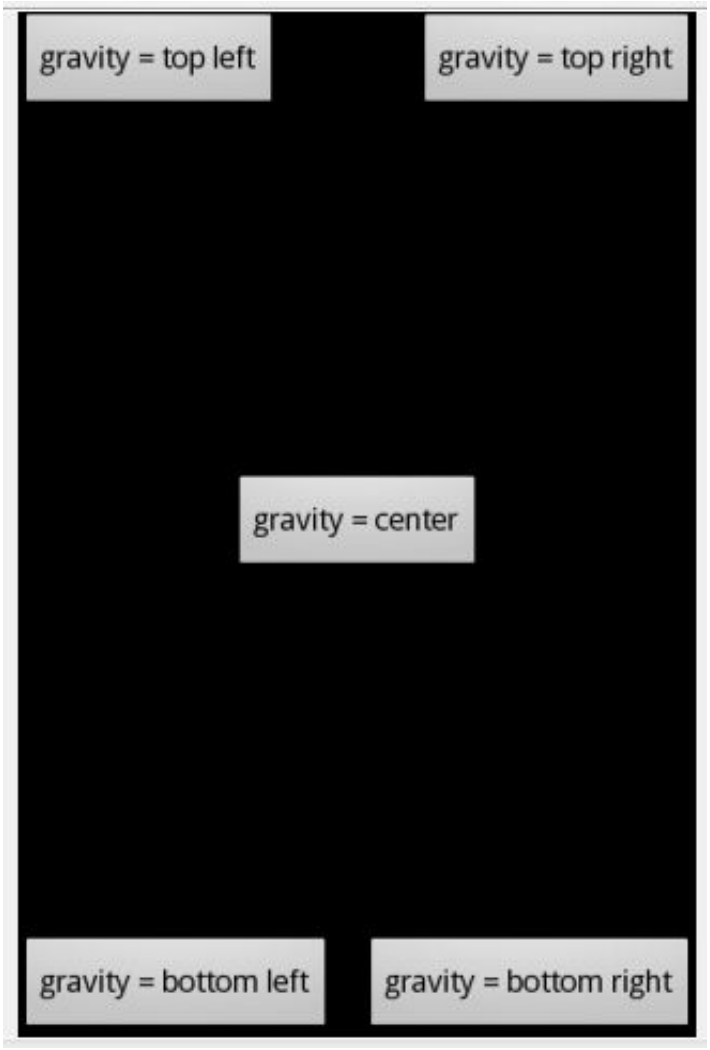
Параметр layout_gravity аналогичен выравниванию из Word или Excel. Удобнее всего продемонстрировать его с использованием **FrameLayout**. Я не описывал этот Layout на прошлом уроке, т.к. он совсем простой. Все помещаемые в него элементы он по умолчанию помещает в левый верхний угол и никак их не выстраивает. Нам это очень подходит для демонстрации настроек выравнивания.

Создадим **grlayout.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <FrameLayout
```

```
            android:id="@+id/frameLayout1"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <Button
                android:id="@+id/button1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="top|left"
                android:text="gravity = top left">
            </Button>
            <Button
                android:id="@+id/button2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="top|right"
                android:text="gravity = top right">
            </Button>
            <Button
                android:id="@+id/button3"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="bottom|left"
                android:text="gravity = bottom left">
            </Button>
            <Button
                android:id="@+id/button4"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="bottom|right"
                android:text="gravity = bottom right">
            </Button>
            <Button
                android:id="@+id/button5"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:text="gravity = center">
            </Button>
        </FrameLayout>
</LinearLayout>
```

На экране видим:



Для наглядности текст кнопки отображает ее свойства. Все очевидно и несложно.

Я честно пытался понять зачем нужны значения gravity **fill_*** и **clip_***, но так и не понял. То, что написано про них в хелпе у меня не работает. Если у вас есть сведения по этому поводу – пишите в каменты.

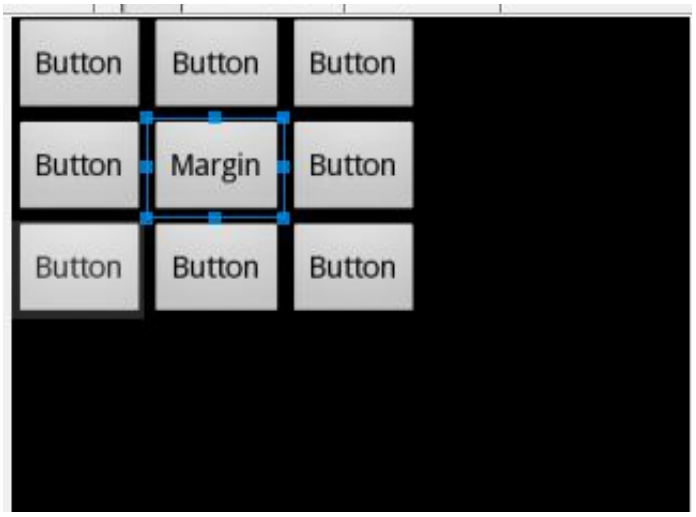
Layout margin

Параметры margin полностью аналогичны **margin** из **html**. Это отступ. Он может быть со всех сторон сразу, либо только с необходимых сторон. Продемонстрируем это на примере TableLayout. Создадим **marginlayout.xml** и нарисуем таблицу три на три с кнопками.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:id="@+id/tableLayout1">
        <TableRow
            android:id="@+id/tableRow1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
            <Button
                android:text="Button"
```

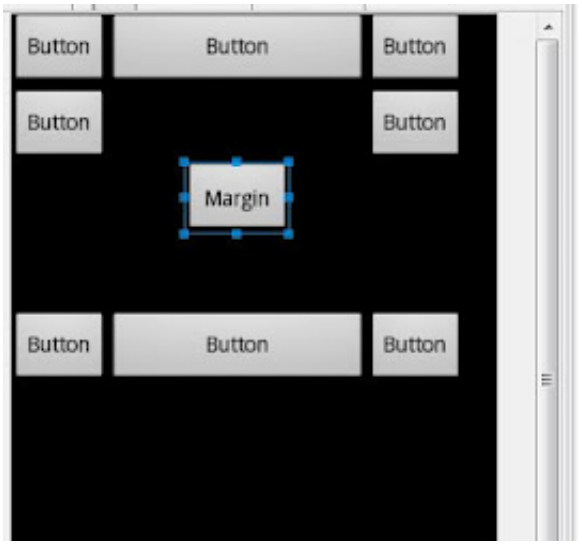
```
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
        <Button
            android:text="Button"
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
        <Button
            android:text="Button"
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
    </TableRow>
    <TableRow
        android:id="@+id/tableRow2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:text="Button"
            android:id="@+id/button4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
        <Button
            android:text="Margin"
            android:id="@+id/button5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
        <Button
            android:text="Button"
            android:id="@+id/button6"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
    </TableRow>
    <TableRow
        android:id="@+id/tableRow3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:text="Button"
            android:id="@+id/button7"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
        <Button
            android:text="Button"
            android:id="@+id/button8"
            android:layout_width="wrap_content"
```

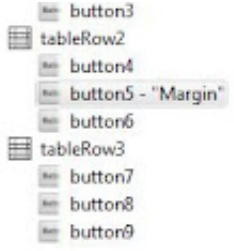
```
            android:layout_height="wrap_content">
        </Button>
        <Button
            android:text="Button"
            android:id="@+id/button9"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </Button>
    </TableRow>
</TableLayout>
</LinearLayout>
```

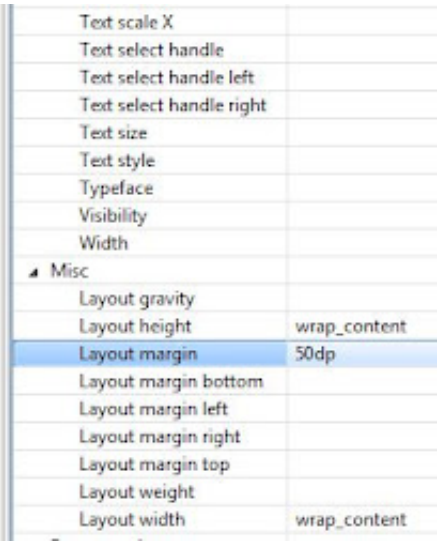


И на кнопке в центре будем экспериментировать.

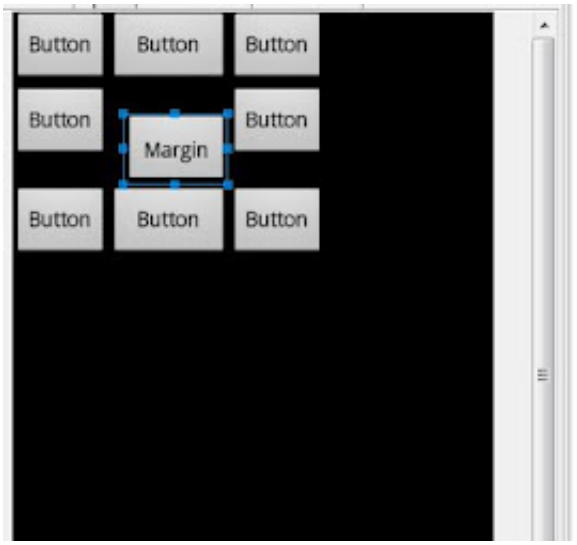
margin = 50 dp
Вокруг кнопки со всех сторон образовался отступ = 50 dp.

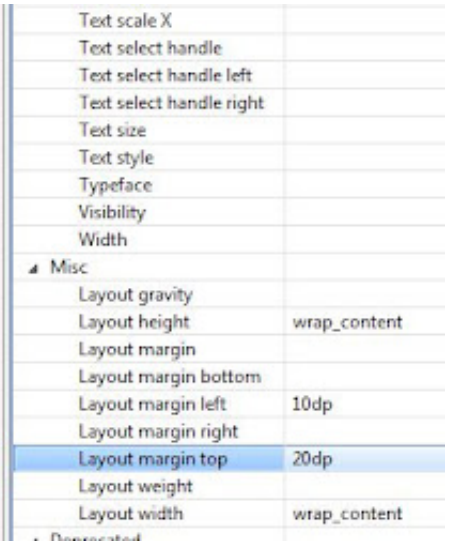




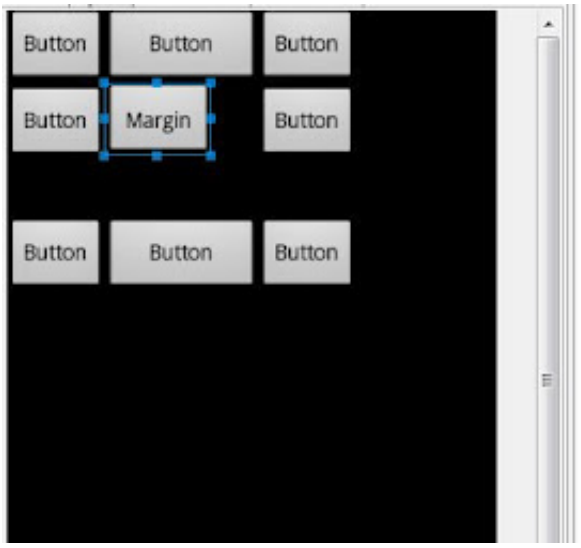


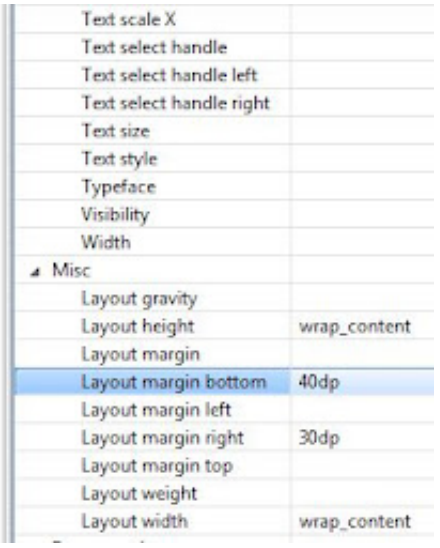
margin left = 10 dp
margin top = 20 dp
Отступ слева и сверху.





margin right = 30 dp
margin bottom = 40 dp
Отступ справа и снизу.





Урок получился большой, но полезный. Думаю, это был последний урок дизайна, моделирования и верстки и дальше мы уже начнем кодить.

Стили

Если кто использовал HTML, то наверняка слышали про каскадные стили - CSS. Стили позволяют вам группировать атрибуты элементов (кнопок, таблиц, параграфов и т.д.). Далее вы просто применяете к элементам стили, и элемент рисуется с учетом всех атрибутов стиля. И нет необходимости повторять несколько раз один и тот же код для элементов, которые должны выглядеть одинаково. Особенно это удобно в случае изменения атрибутов. Вы просто меняете один раз стиль и все элементы с этим стилем меняются.

В Android тоже есть стили и они имеют точно такое же назначение. Если у вас есть несколько элементов и вам надо, чтобы они выглядели одинаково, то вы просто создаете один стиль и применяете его к нужным элементам. В принципе, вы пока можете пока не заморачиваться этим и начать использовать стили, когда наберетесь опыта. Ну а тем кому это интересно прямо сейчас - прошу в эту ветку нашего форума. Пользователь icamys на примере

подробно разъясняет как использовать стили.

На следующем уроке:

- научимся обращаться к View-элементам из кода и менять их свойства

Единицы измерения. Чем отличается dp (dip) от px. Screen Density.

Для указания ширины, высоты и отступов View-элементов используются различные единицы измерения (ЕИ):

dp или **dip** - Density-independent Pixels. Абстрактная ЕИ, позволяющая приложениям выглядеть одинаково на различных экранах и разрешениях.

sp - Scale-independent Pixels. То же, что и dp, только используется для размеров шрифта в View элементах

pt - 1/72 дюйма, определяется по физическому размеру экрана. Эта ЕИ из типографии.

px – пиксел, не рекомендуется использовать т.к. на разных экранах приложение будет выглядеть по-разному.

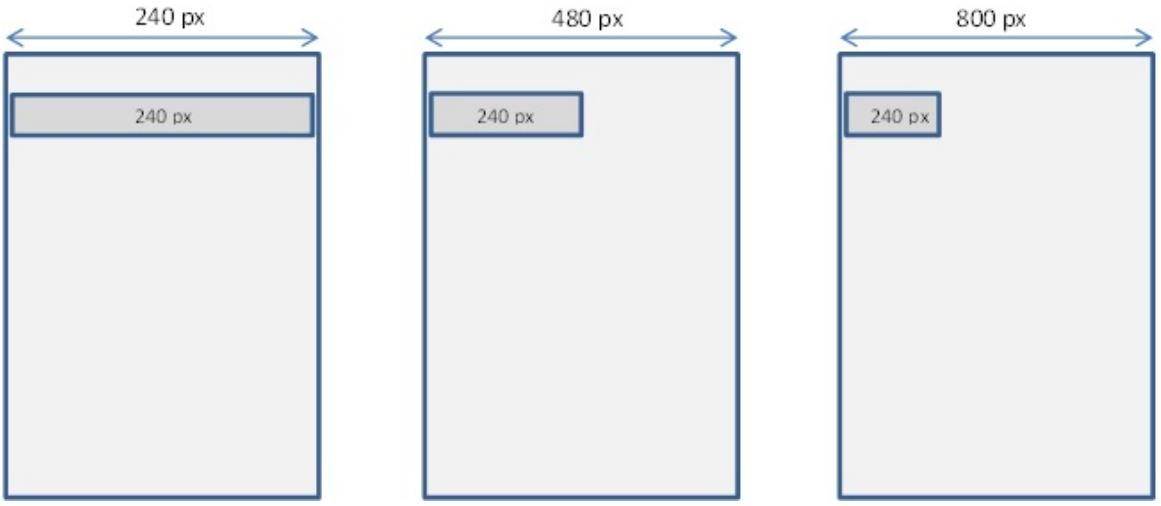
mm – миллиметр, определяется по физическому размеру экрана

in – дюйм, определяется по физическому размеру экрана

Давайте разбираться, чем они отличаются друг от друга.

in, mm и pt – неизменны относительно друг друга. Всегда **1 in = 25,4 mm** и **1 in = 72 pt**. Это классические единицы измерения. Т.е. задаете, например, кнопке ширину = 1 in и она должна отобразиться шириной в один дюйм, независимо от разрешения и диагонали экрана.

Что такое px, думаю, тоже понятно. Если у вас есть устройство с экраном шириной 480 px и вы создали кнопку шириной 240 px, то эта кнопка займет в ширину ровно пол-экрана. Но если вы откроете ваше приложение на устройстве с экраном с меньшим разрешением, то соотношение изменится, например:
- если разрешение 320x240, ширина экрана = 240 px. Кнопка займет уже не пол-экрана в ширину, а весь экран
- если же разрешение 1280x800, ширина = 800 px. Кнопка опять же будет занимать в ширину не пол-экрана, а чуть меньше трети



А ведь экран приложения – это обычно не одна кнопка, а набор из многих элементов и все они будут расползаться или сжиматься на разных разрешениях. Поэтому использовать px при разработке НЕ рекомендуется.

Для того, чтобы избежать таких ситуаций на разных разрешениях рекомендуется использовать dp (и sp). Его можно определить, как масштабируемый px. За степень масштабируемости отвечает **Screen Density**. Это коэффициент, который используется системой для вычисления значения dp. На текущий момент есть 5 значений этого коэффициента:
- low (**ldpi**) = 0,75
- medium (**mdpi**) = 1
- tv (**tvdpi**) = 1,33
- high (**hdpi**) = 1,5
- extra high (**xhdpi**) = 2

Т.е. когда для экрана стоит режим **mdpi**, то **1 dp = 1 px**. Т.е. кнопка шириной 100 dp будет выглядеть также как и кнопка шириной 100 px.

Если, например, у нас экран с низким разрешением, то используется режим **ldpi**. В этом случае **1 dp = 0,75 px**. Т.е. кнопка шириной 100 dp будет выглядеть так же как кнопка шириной 75 px.

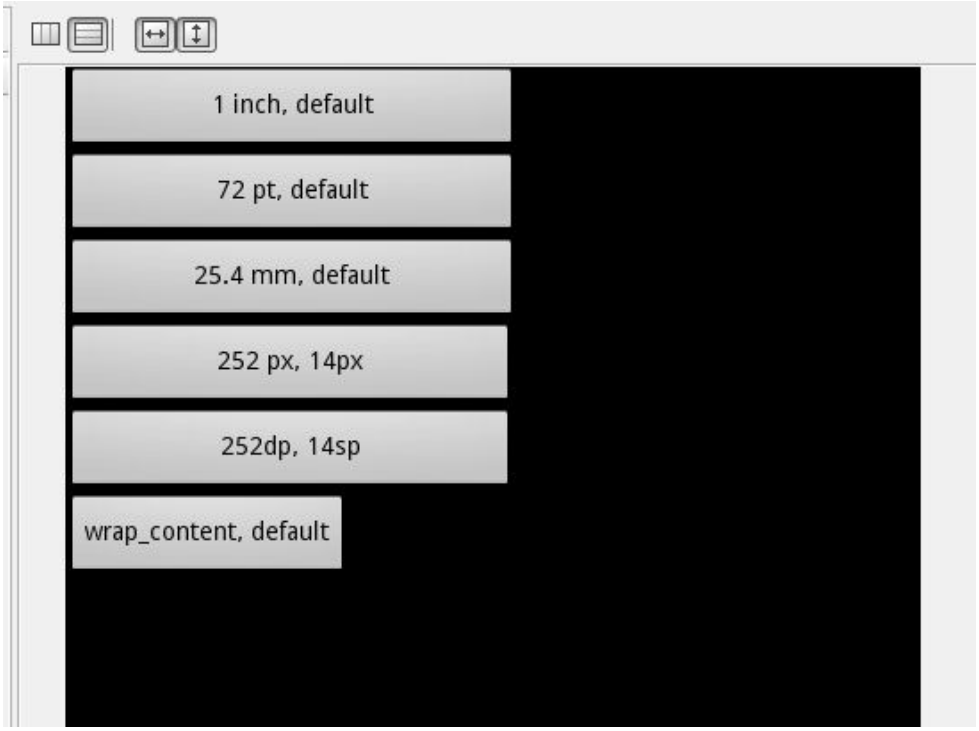
Если у нас экран с высоким разрешением, то используется режим **hdpi** или **xhdpi**. **1 dp = 1, 5 px** или **2 px**. И кнопка шириной 100 dp будет выглядеть так же как кнопка шириной 150 px или 200 px.

Т.е. при различных разрешениях используются различные Density режимы, которые позволяют приложениям масштабироваться и выглядеть если не одинаково, то, по крайне мере, похоже на всех экранах.

Рассмотрим пример. Предположим у нас есть три устройства (характеристики реальны и взяты из спецификаций):

HTC **Wildfire** S: 3,2 inch, 480x320 px, 180 dpi
HTC **Desire**: 3,7 inch, 800x480 px, 252 dpi
HTC **Flyer**: 7 inch, 1280x800 px, 170 dpi

Я создам такой экран:



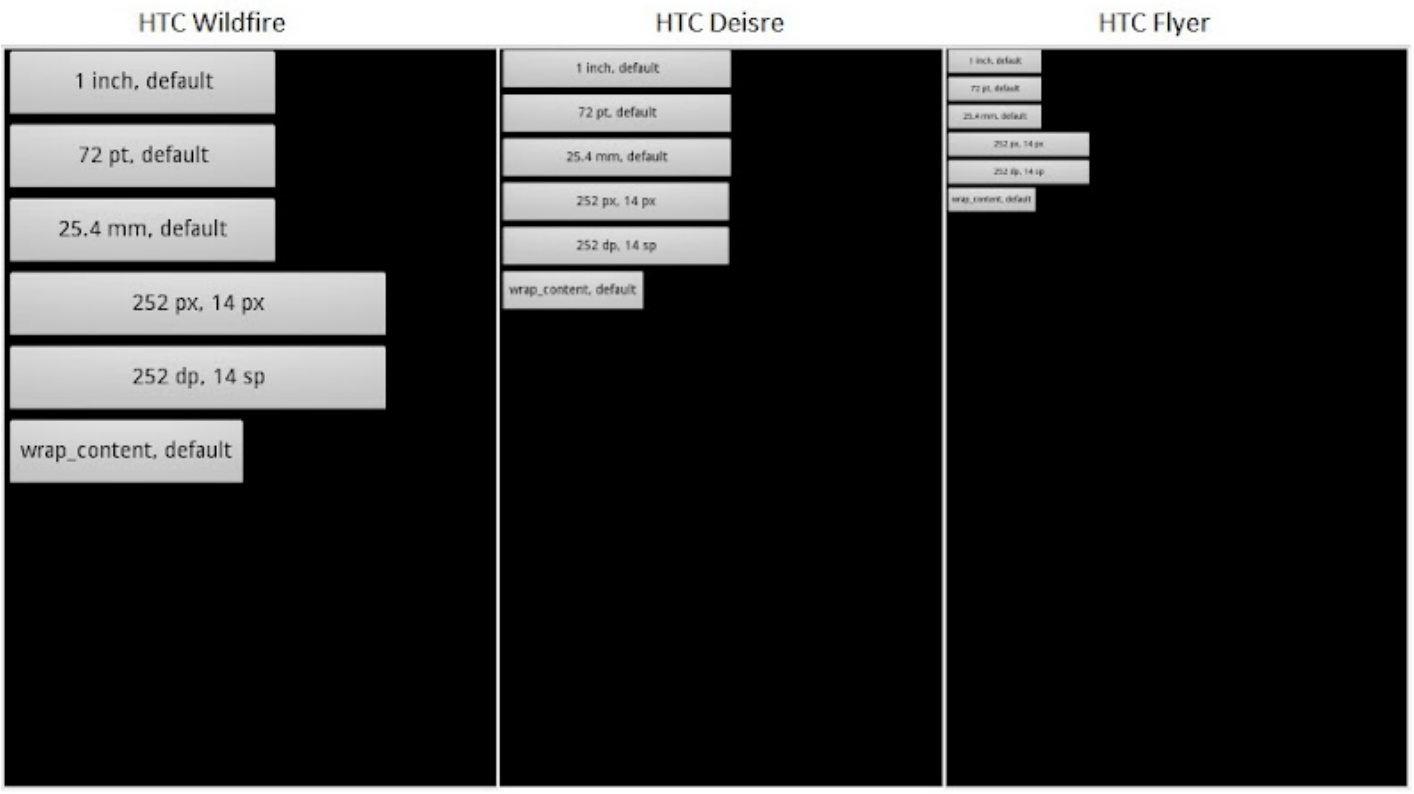
Это несколько кнопок, в которых ширина и размер шрифта определены с использованием разных единиц измерения. На каждой кнопке для наглядности я написал ее ширину (**layout_width**) и размер шрифта (**textSize**) через запятую. Обратите внимание, что ширина всех кнопок кроме последней одинакова. Так происходит потому, что 1 in = 72 pt = 25,4 mm в любом случае, а для данного экрана также 1 in = 252 px = 252 dp. Шрифты также везде одинаковы, т.к. размер шрифта по умолчанию равен 14 sp и в данном случае равен 14 px.

xml-код:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/button1"
        android:layout_height="wrap_content"
        android:layout_width="1in"
        android:text="1 inch, default">
    </Button>
    <Button
        android:id="@+id/button2"
        android:layout_height="wrap_content"
        android:layout_width="72pt"
        android:text="72 pt, default">
    </Button>
    <Button
        android:id="@+id/button3"
        android:layout_height="wrap_content"
        android:layout_width="25.4mm"
        android:text="25.4 mm, default">
    </Button>
    <Button
        android:id="@+id/button4"
        android:layout_height="wrap_content"
        android:layout_width="252px"
        android:text="252 px, 14 px"
        android:textSize="14px">
```

```
    </Button>
    <Button
        android:id="@+id/button5"
        android:layout_height="wrap_content"
        android:layout_width="252dp"
        android:textSize="14sp"
        android:text="252 dp, 14 sp">
    </Button>
    <Button
        android:id="@+id/button6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="wrap_content, default">
    </Button>
</LinearLayout>
```

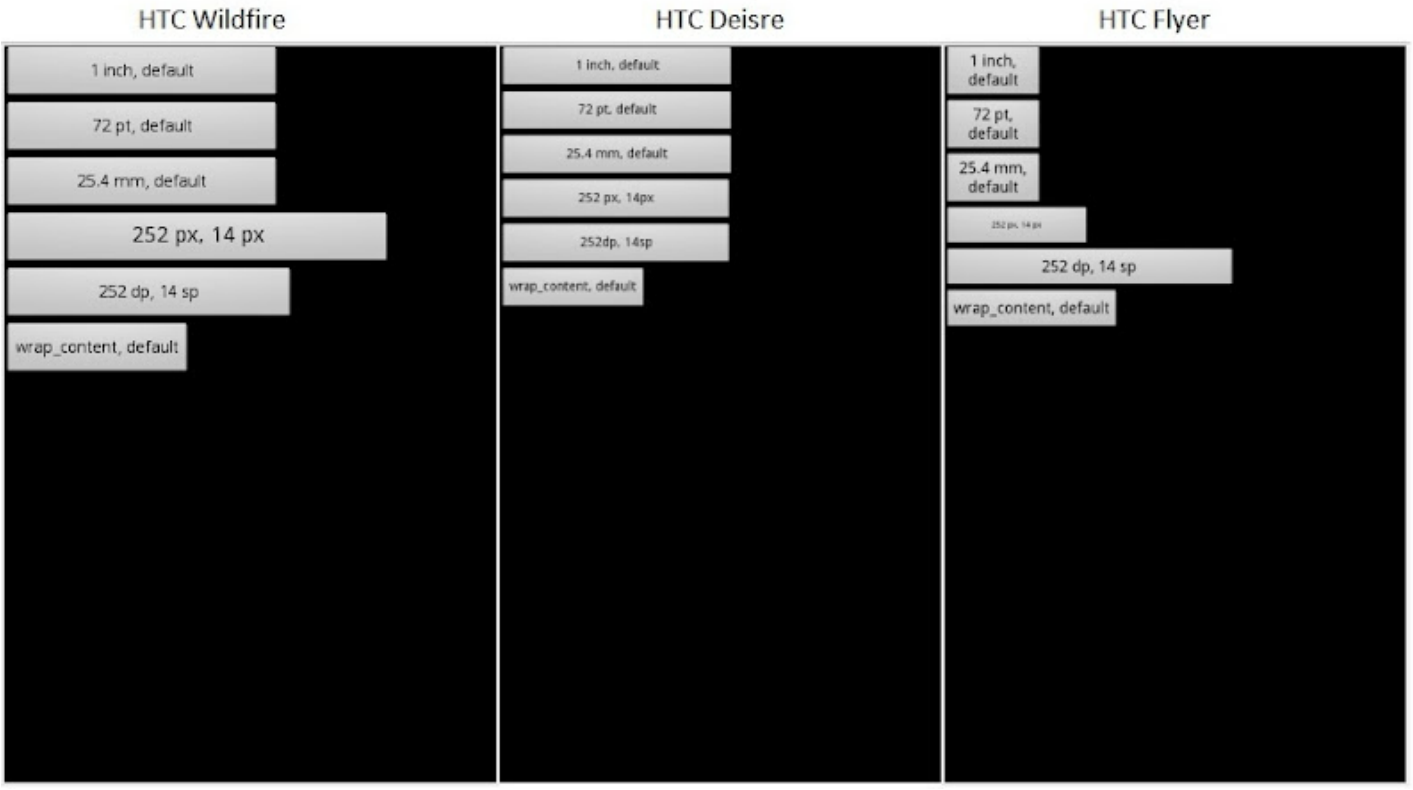
Посмотрим, как это будет выглядеть на экранах других устройств:



Для всех экранов я поставил режим **mdpi**. Скриншоты экранов смасштабированы к одному размеру для наглядности. Надо понимать, что на самом деле диагонали экранов существенно отличаются, а наша цель – добиться, чтобы приложение выглядело одинаково на различных устройствах.

Мы видим, что на всех экранах 1 px остался равен 1 dp (т.к. режим mdpi). И in, pt и mm сохранили свои пропорции (1; 72; 25,4) относительно друг друга. Но обратите внимание, что 1 in уже не равен 252 px на экранах Wildfire и Flyer. Это потому, что у этих устройств другое соотношение диагонали экрана и разрешения:
- у **Wildfire** экран с **dpi = 180**, т.е. 1 in = 180 px, поэтому первая кнопка (с шириной = 1 inch) теперь короче, чем кнопка с шириной 252 px.
- для **Flyer**, соответственно, **dpi = 170**.

Видно, что приложение выглядит достаточно по разному на трех экранах. Так было бы, если бы не существовало коэфициента Screen Density. Но он есть и давайте смотреть, чем он полезен. Я включу режим **ldpi** для **Wildfire** и **xhdpi** для **Flyer**. **Desire** оставляю в **mdpi**.



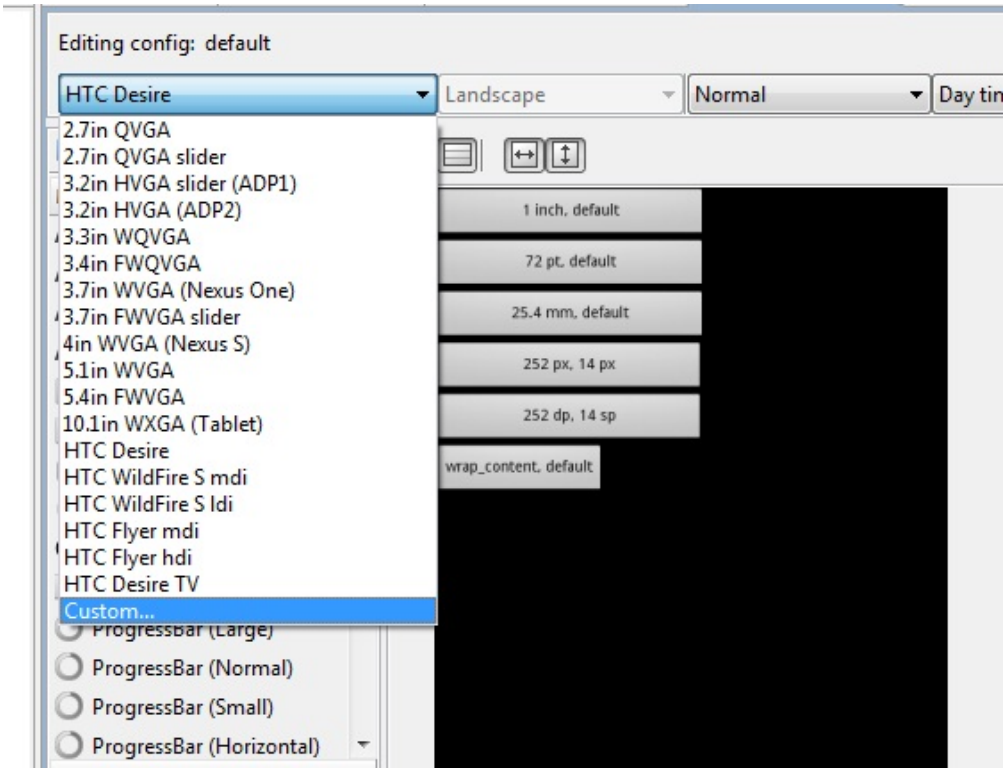
Ширина кнопок указанная в in, mm, pt неизменна, на эти единицы измерения режимы Density влияния не имеют. Нам интересны четвертая и пятая кнопки. Видим, что на экранах Wildfire и Flyer отличаются размеры px и dp, т.к. мы сменили mdpi на ldpi и xhdpi. Для Wildfire 1 dp стал равен 0,75 px, а для Flyer 1 dp = 2 px. Также видим, что изменился шрифт на кнопках, где размер шрифта был указан по умолчанию или в sp-единицах. Он так же, как и dp смасштабировался благодаря Density режимам. А шрифт, размер которого был указан в px (четвертая кнопка) оставался неизменным и на Wildfire выглядит крупным, а на Flyer – мелким.

Отлично видно, что адекватнее всего перенос на другие экраны перенесли пятая и шестая кнопки. Для пятой кнопки используются dp и sp. Для шестой кнопки – ширина = wrap_content и размер шрифта по умолчанию. А кнопки с in, mm, pt и px статичны и на разных экранах выглядят по-разному. Наверняка, есть случаи, когда необходимо использовать именно эти единицы измерения. Но в основном старайтесь использовать dp (для ширины, высоты и т.д.) и sp (для размера шрифта).

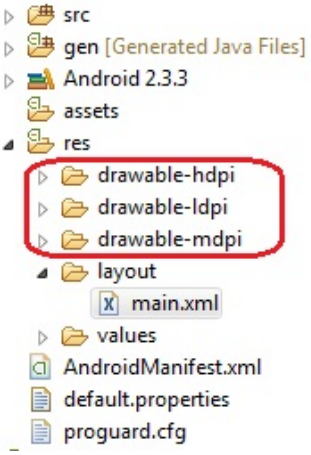
Конечно Density не дает масштабирования абсолютно пропорциального разнице в разрешениях экрана. Погрешность есть, но она невелика и является вполне приемлемой платой за способность приложения «сохранять форму» на разных устройствах.

Кем именно устанавливаются Density режимы для различных экранов – я не знаю. Но подозреваю, что производителями устройств/экранов. Еще мне интересно, можно ли эти режимы переключать при работе устройства. Думаю, чуть позже я найду ответы на эти вопросы.

Для создания этого материала я использовал различные конфигурации экранов, которые можно создавать самому:



Вы наверно обратили внимание, что в папке Андроид-проекта в Eclipse есть папки **drawable** с суффиксами hdpi, ldpi, mdpi:



Папка drawable используется для хранения изображений, а суффиксы дают понять системе из какой именно drawable использовать картинки при текущем Density режиме. Подробнее об этом можно почитать в хелпе. Кстати, там же вы найдете уже изученный нами **–land**, который связан с горизонтальной ориентацией экрана. Будем по мере изучения Андроид знакомиться с остальными.

Урок 8. Работаем с элементами экрана из кода

В этом уроке мы:

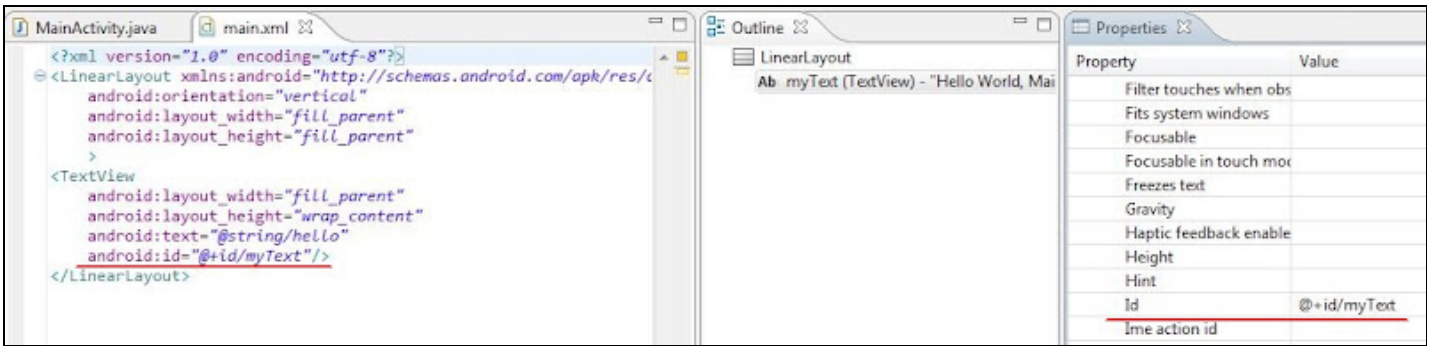
- научимся обращаться из кода к View-элементам на экране и менять их свойства

Создадим проект:

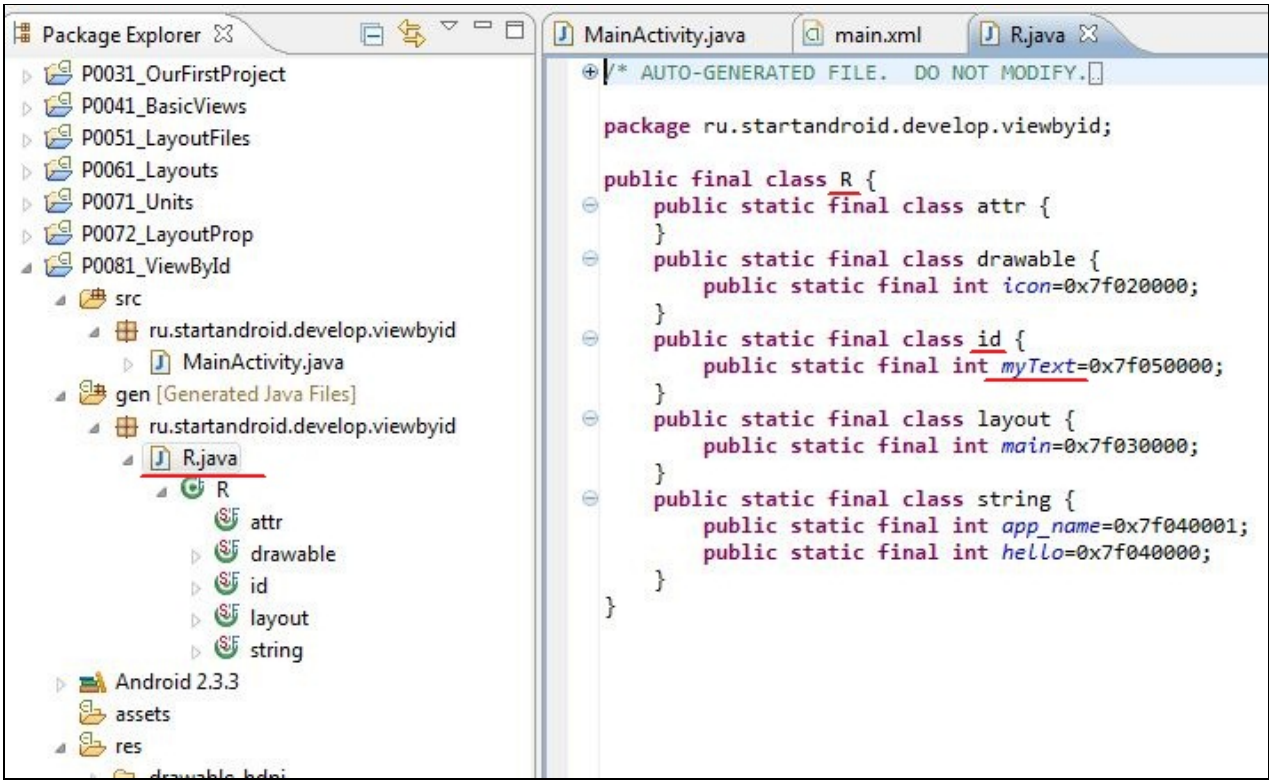
Project name: P0081_ViewById
Build Target: Android 2.3.3
Application name: ViewById
Package name: ru.startandroid.develop.viewbyid
Create Activity: MainActivity

Чтобы обратиться к элементу экрана из кода, нам нужен его **ID**. Он прописывается либо в **Properties**, либо в **layout-файлах**, как вам удобнее. Для ID существует четкий формат - **@+id/name**, где **+** означает, что это новый ресурс и он должен добавиться в **R.java** класс, если он там еще не существует.

Давайте откроем **main.xml**, для **TextView** укажем **ID = @+id/myText** и сохраним



Теперь откроем **R.java** и видим, что для класса **id** появилась константа **myText**. Т.е. чтобы к ней обратиться, надо написать **R.id.myText**.



Она связана с элементом TextView и мы можем ее использовать, чтобы обратиться к элементу программно. Для этого нам понадобится метод findViewById. Он по ID возвращает View. Давайте напишем вызов этого метода. Напомню, что пока мы пишем наш код в методе **onCreate**. Это метод, который вызывается при создании Activity. Если вдруг непонятно, куда писать, можно подсмотреть в конец урока, там я выложил код.

Откроем MainActivity.java и после строки с вызовом метода **setContentView** напишем:

```
View myTextView = findViewById(R.id.myText);
```

Если View подчеркнуто красным, то скорей всего этот класс не добавлен в секцию import. Нажмите CTRL+SHIFT+O для автоматического обновления импорта.

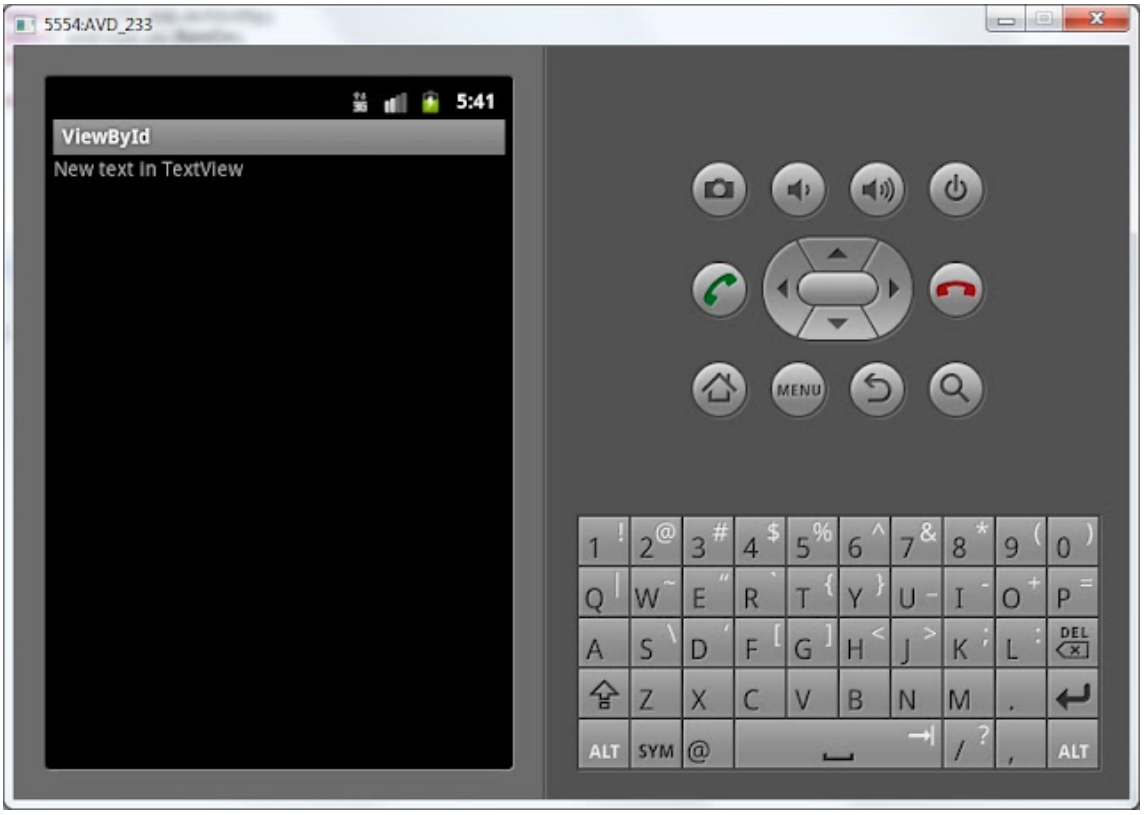
Теперь **myTextView** (типа View) – это наш TextView на экране. Но тип View – это предок для TextView (и остальных View-элементов). И он нам не подходит, если мы хотим проделывать операции соответствующие TextView. Поэтому нам необходимо преобразование View в TextView. Изменим наш код на следующий:

```
TextView myTextView = (TextView) findViewById(R.id.myText);
```

Теперь **myTextView** имеет тип **TextView**, а результат метода findViewById мы преобразуем из View в TextView. Теперь мы можем применять к myTextView методы класса TextView. Для примера возьмем метод setText. Сейчас отображаемый текст = *Hello World, MainActivity!*. Мы его программно поменяем на *New text in TextView*

```
myTextView.setText("New text in TextView");
```

Сохраняем, запускаем (CTRL+F11) и видим, что текст изменился



Добавим на экран кнопку (Button), Id = @+id/myBtn, текст оставим по умолчанию. Сохраняем - CTRL+SHIFT+S (если не сохранить, то в R.java не появится ID).

Пишем код:

```
Button myBtn = (Button) findViewById(R.id.myBtn);
```

Обратите внимание, что у меня совпадает имя объекта и ID

```
Button myBtn = (Button) findViewById(R.id.myBtn);
```

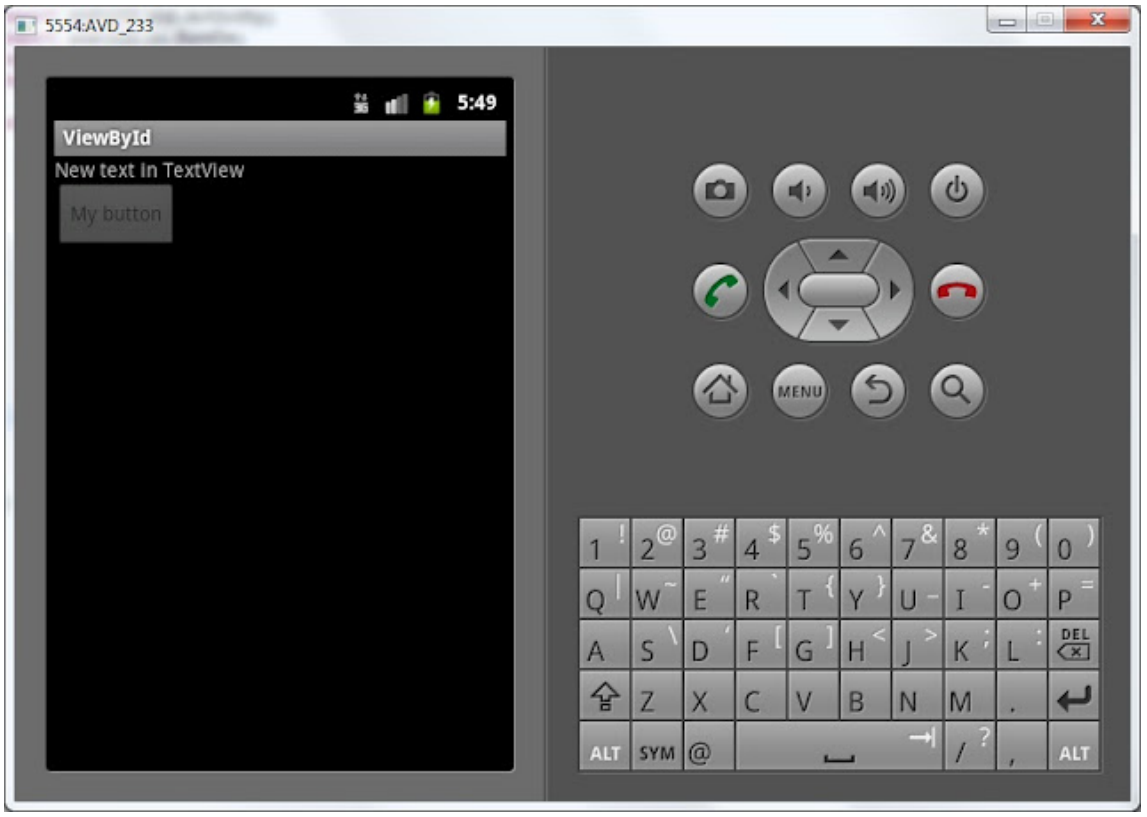
Они друг другу не мешают и так делать даже логичнее. Это остается на ваше усмотрение. Так, кнопку мы нашли, теперь давайте изменим ее текст:

```
myBtn.setText("My button");
```

Запустим приложение. Текст на кнопке поменялся, на кнопку можно понажимать, но ничего происходить не будет. Т.к. мы нигде не указывали, что надо делать при нажатии. Этим займемся на следующем уроке. А пока давайте сделаем кнопку неактивной.

```
myBtn.setEnabled(false);
```

Мы поменяли параметр **Enabled**. Теперь на кнопку нельзя нажать. Сохраним, запустим и убедимся.



Добавим CheckBox, id = @+id/myChb. По умолчанию галочка не стоит. Давайте поставим ее программно, для этого используется метод setChecked, который меняет параметр Checked.

```
CheckBox myChb = (CheckBox) findViewById(R.id.myChb);

myChb.setChecked(true);
```

Запустив приложение видим, что код сработал.

Как видите – все несложно. Используем метод **findViewById**, чтобы по **ID** получить объект соответствующий какому-либо **View-элементу** (Button, TextView, CheckBox) и далее вызываем необходимые методы объектов (setText, setEnabled, setChecked).

В итоге должен получиться такой код:

```
package ru.startandroid.develop.viewbyid;


import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.TextView;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        TextView myTextView = (TextView) findViewById(R.id.myText);
        myTextView.setText("New text in TextView");

        Button myBtn = (Button) findViewById(R.id.myBtn);
        myBtn.setText("My button");
        myBtn.setEnabled(false);

        CheckBox myChb = (CheckBox) findViewById(R.id.myChb);
        myChb.setChecked(true);
    }
```

```
}
```

На следующем уроке:

- научимся обрабатывать нажатие кнопки

Урок 9. Обработчики событий на примере Button.

В этом уроке мы:

- научимся обрабатывать нажатие кнопки и узнаем, что такое обработчик

Создадим проект:

Project name: P0091_OnClickButtons
Build Target: Android 2.3.3
Application name: OnClickButtons
Package name: ru.startandroid.develop.onclickbuttons
Create Activity: MainActivity

В layout-файл **main.xml** напишем следующее и сохраним:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="horizontal">
     <LinearLayout
          android:id="@+id/linearLayout1"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:layout_margin="30dp"
          android:orientation="vertical">
         <TextView
               android:id="@+id/tvOut"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:layout_gravity="center_horizontal"
               android:layout_marginBottom="50dp"
               android:text="TextView">
         </TextView>
         <Button
               android:id="@+id/btnOk"
               android:layout_width="100dp"
               android:layout_height="wrap_content"
               android:layout_gravity="center_horizontal"
               android:text="OK">
         </Button>
         <Button
               android:id="@+id/btnCancel"
               android:layout_width="100dp"
               android:layout_height="wrap_content"
               android:layout_gravity="center_horizontal"
               android:text="Cancel">
         </Button>
     </LinearLayout>
</LinearLayout>
```

У нас есть TextView с текстом и две кнопки: OK и Cancel. Мы сделаем так, чтобы по нажатию кнопки менялось содержимое TextView. По нажатию кнопки OK – будем выводить текст: «Нажата кнопка OK», по нажатию Cancel – «Нажата кнопка Cancel».

Открываем **MainActivity.java**. Описание объектов вынесем за пределы метода **onCreate**. Это сделано для того, чтобы мы

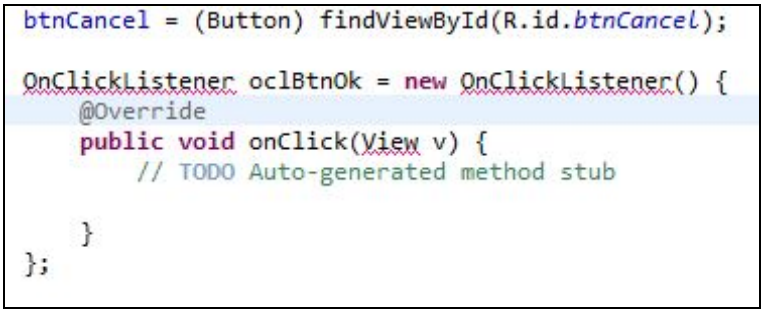
могли из любого метода обращаться к ним. В onCreate мы эти объекты заполним с помощью уже пройденного нами метода **findViewById**. В итоге должен получиться такой код:

```
public class MainActivity extends Activity {

   TextView tvOut;
   Button btnOk;
   Button btnCancel;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);

     // найдем View-элементы
     tvOut = (TextView) findViewById(R.id.tvOut);
     btnOk = (Button) findViewById(R.id.btnOk);
     btnCancel = (Button) findViewById(R.id.btnCancel);

   }
 }
```

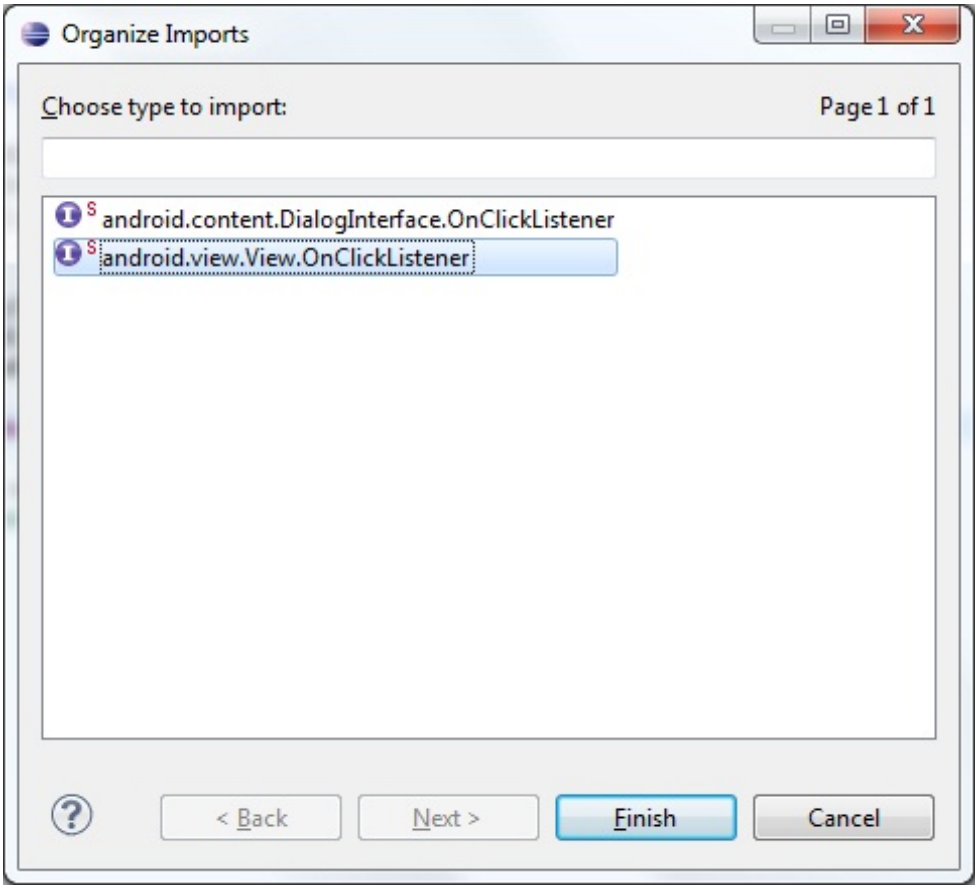
Обновляем секцию import (CTRL+SHIFT+O). Объекты tvOut, btnOk и btnCancel соответствуют View-элементам экрана и мы можем с ними работать. Нам надо научить кнопку реагировать на нажатие. Для этого у кнопки есть метод setOnClickListener (View.OnClickListener l). На вход подается объект с интерфейсом View.OnClickListener. Именно этому объекту кнопка поручит обрабатывать нажатия. Давайте создадим такой объект. Код продолжаем писать в onCreate:

```
    OnClickListener oclBtnOk = new OnClickListener() {
       @Override
       public void onClick(View v) {
         // TODO Auto-generated method stub

       }
     };
```

Eclipse подчеркивает OnClickListener красной линией



т.к. пока не знает его. Необходимо обновить секцию import. Жмем CTRL+SHIFT+O, Eclipse показывает нам, что он знает два интерфейса с именем onClickListener и предлагает выбрать. Нам нужен View.OnClickListener, т.к. метод кнопки setOnClickListener принимает на вход именно его.



Итак, мы создали объект oclBtnOk, который реализует интерфейс View.OnClickListener. Объект содержит метод onClick – это как раз то, что нам нужно. Именно этот метод будет вызван при нажатии кнопки. Мы решили, что по нажатию будем выводить текст: «Нажата кнопка OK» в TextView (tvOut). Реализуем это.

В методе onClick пишем:

```
tvOut.setText("Нажата кнопка OK");
```

Обработчик нажатия готов. Осталось «скормить» его кнопке с помощью метода setOnClickListener.

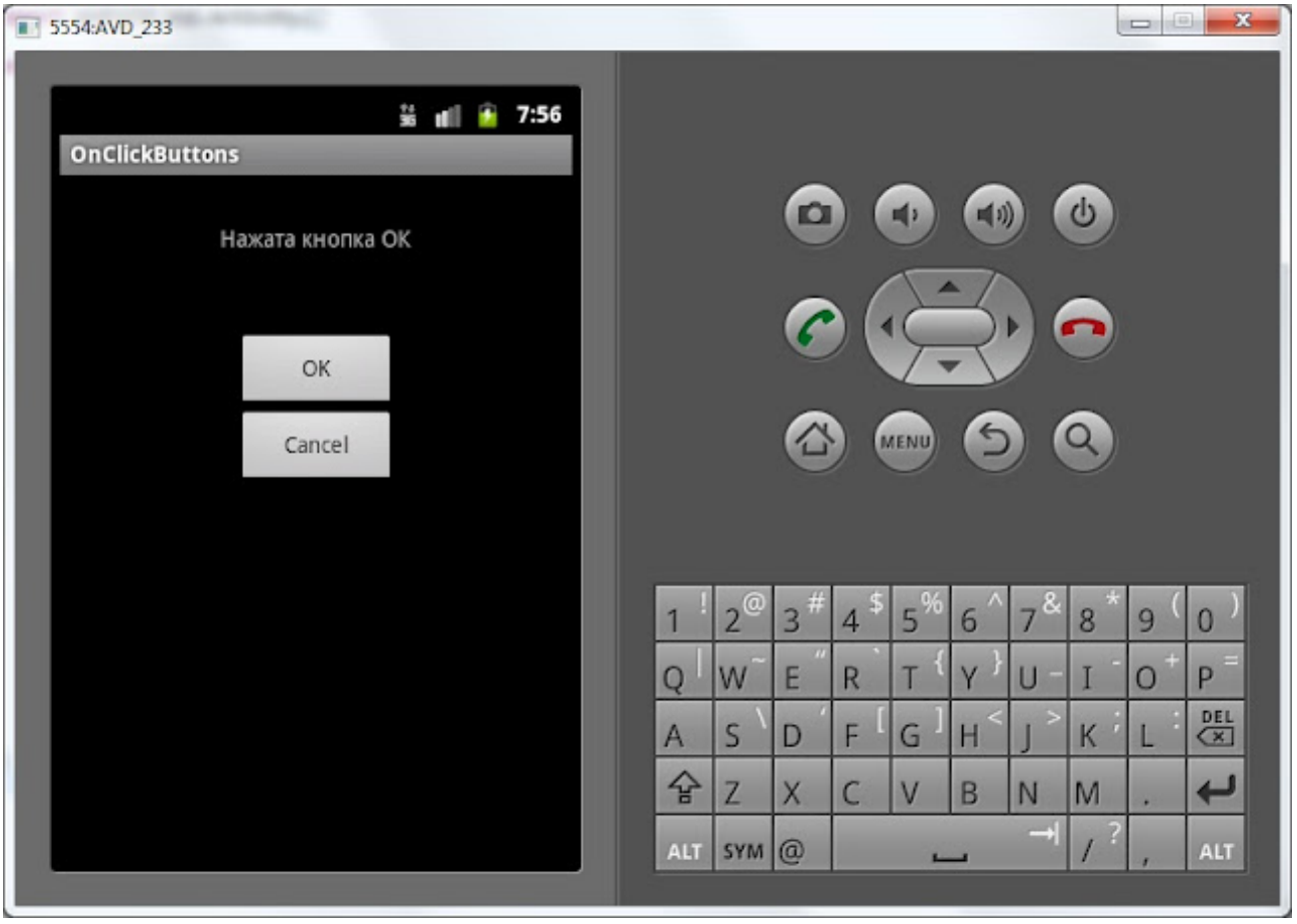
```
btnOk.setOnClickListener(oclBtnOk);
```

В итоге должен получится такой код:

```
public class MainActivity extends Activity {

   TextView tvOut;
   Button btnOk;
   Button btnCancel;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);

     // найдем View-элементы
     tvOut = (TextView) findViewById(R.id.tvOut);
     btnOk = (Button) findViewById(R.id.btnOk);
```

```
    btnCancel = (Button) findViewById(R.id.btnCancel);

    // создаем обработчик нажатия
    OnClickListener oclBtnOk = new OnClickListener() {
      @Override
      public void onClick(View v) {
        // Меняем текст в TextView (tvOut)
        tvOut.setText("Нажата кнопка OK");
      }
    };

    // присвоим обработчик кнопке OK (btnOk)
    btnOk.setOnClickListener(oclBtnOk);
  }
}
```

Все сохраняем и запускаем. Жмем на кнопку OK и видим. Что текст изменился



Нажатие на Cancel пока ни к чему не приводит, т.к. для нее мы обработчик не создали и не присвоили. Давайте сделаем это аналогично, как для кнопки OK. Сначала мы создаем обработчик:

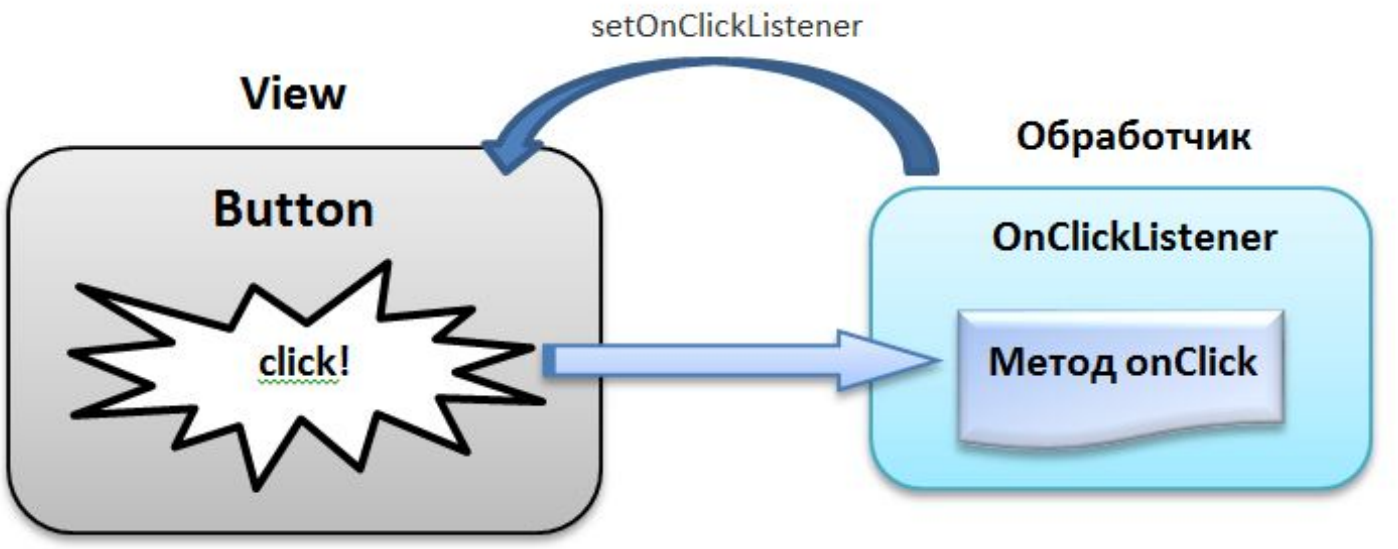
```
    OnClickListener oclBtnCancel = new OnClickListener() {
      @Override
      public void onClick(View v) {
        // Меняем текст в TextView (tvOut)
        tvOut.setText("Нажата кнопка Cancel");
      }
    };
```

Потом присваиваем его кнопке:

```
btnCancel.setOnClickListener(oclBtnCancel);
```

Сохраняем, запускаем, проверяем. Обе кнопки теперь умеют обрабатывать нажатия.

Давайте еще раз проговорим механизм обработки событий на примере нажатия кнопки. Сама кнопка обрабатывать нажатия не умеет, ей нужен обработчик (его также называют слушателем - listener), который присваивается с помощью метода setOnClickListener. Когда на кнопку нажимают, обработчик реагирует и выполняет код из метода onClick. Это можно изобразить так:



Соответственно для реализации необходимо выполнить следующие шаги:

- создаем обработчик
- заполняем метод onClick
- присваиваем обработчик кнопке

и система обработки событий готова.

На следующем уроке:

- научимся использовать один обработчик для нескольких View-элементов
- научим Activity выступать в качестве обработчика

Урок 10. Оптимизируем реализацию обработчиков.

В этом уроке мы:

- научимся использовать один обработчик для нескольких View-элементов
- научим Activity выступать в качестве обработчика

Создадим проект:
Project name: P0101_Listener
Build Target: Android 2.3.3
Application name: Listener
Package name: ru.startandroid.develop.listener
Create Activity: MainActivity

Будем работать с теми же View, что и в предыдущем уроке. Код для **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:layout_height="match_parent"
	android:layout_width="match_parent"
	android:orientation="horizontal">
	<LinearLayout
		android:id="@+id/linearLayout1"
		android:layout_height="match_parent"
		android:orientation="vertical"
		android:layout_width="match_parent"
		android:layout_margin="30dp">
		<TextView
			android:layout_width="wrap_content"
			android:text="TextView"
			android:layout_height="wrap_content"
			android:id="@+id/tvOut"
			android:layout_gravity="center_horizontal"
			android:layout_marginBottom="50dp">
		</TextView>
		<Button
			android:layout_height="wrap_content"
			android:layout_gravity="center_horizontal"
			android:id="@+id/btnOk"
			android:text="OK"
			android:layout_width="100dp">
		</Button>
		<Button
			android:layout_height="wrap_content"
			android:layout_gravity="center_horizontal"
			android:id="@+id/btnCancel"
			android:text="Cancel"
			android:layout_width="100dp">
		</Button>
	</LinearLayout>
</LinearLayout>
```

Один обработчик для двух кнопок

Итак, у нас есть TextView с текстом и две кнопки. Как и на прошлом уроке, мы сделаем так, чтобы по нажатию кнопки менялось содержимое TextView. По нажатию кнопки OK – будем выводить текст: «Нажата кнопка OK», по нажатию Cancel – «Нажата кнопка Cancel». Но сейчас мы сделаем это с помощью **одного обработчика**, который будет обрабатывать нажатия **для обеих кнопок**.

Напомню механизм обработки событий на примере нажатия кнопки. Сама кнопка обрабатывать нажатия не умеет, ей нужен обработчик (listener), который присваивается с помощью метода setOnClickListener. Когда на кнопку нажимают, обработчик реагирует и выполняет код из метода onClick.

Соответственно для реализации необходимо выполнить следующие шаги:
- создаем обработчик
- заполняем метод onClick
- присваиваем обработчик кнопке

В нашем случае мы будем присваивать один обработчик обеим кнопкам, а внутри обработчика надо будет определять, какая именно кнопка была нажата.

Подготовим объекты и создадим обработчик:

```
public class MainActivity extends Activity {

   TextView tvOut;
   Button btnOk;
   Button btnCancel;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);

     // найдем View-элементы
     tvOut = (TextView) findViewById(R.id.tvOut);
     btnOk = (Button) findViewById(R.id.btnOk);
     btnCancel = (Button) findViewById(R.id.btnCancel);

     // создание обработчика
     OnClickListener oclBtn = new OnClickListener() {
       @Override
       public void onClick(View v) {
         // TODO Auto-generated method stub

       }
     };

   }
 }
```

Давайте заполнять метод onClick. На вход ему подается объект класса **View**, это как раз то, что нам нужно. Это View на которой произошло нажатие и которая вызвала обработчик. Т.е. в нашем случае это будет либо кнопка OK либо Cancel. Нам осталось узнать **ID** этой **View** и сравнить его с нашими **R.id.btnOk** и **R.id.btnCancel**, чтобы определить какая именно это кнопка. Чтобы получить ID какой-либо View, используется метод getId. Для перебора результатов используем java-оператор switch.

Реализация метода onClick:

```
    public void onClick(View v) {
      // по id определеяем кнопку, вызвавшую этот обработчик
      switch (v.getId()) {
      case R.id.btnOk:
        // кнопка OK
        tvOut.setText("Нажата кнопка OK");
        break;
      case R.id.btnCancel:
        // кнопка Cancel
        tvOut.setText("Нажата кнопка Cancel");
        break;
      }
    }
```

Если сейчас запустить приложение и проверить, то ничего не произойдет. Обработчик то мы создали, но не присвоили его кнопкам. Обеим кнопкам присваиваем один и тот же обработчик:

```
btnOk.setOnClickListener(oclBtn);
btnCancel.setOnClickListener(oclBtn);
```

Вот теперь можем запускать и проверять, все должно работать.

Как вы понимаете, **один обработчик** может быть присвоен не двум, а **любому количеству** кнопок. И не только кнопкам. У остальных View-элементов тоже есть различные события, которые нуждаются в обработчиках. В дальнейшем мы еще будем с ними работать. А сейчас важно понять схему, как происходит обработка событий.

Отличие способа реализации на этом уроке от прошлого урока в том, что сейчас мы создали **один объект-обработчик для обеих кнопок**, а на прошлом уроке - **два объекта, по одному каждой кнопке**. Есть правило – чем меньше объектов вы создаете, тем лучше, т.к. под каждый объект выделяется память, а это достаточно ограниченный ресурс, особенно для телефонов. Поэтому создавать один обработчик для нескольких View это правильнее с точки зрения оптимизации. К тому же кода становится меньше и читать его удобнее.

Есть еще один способ создания обработчика, который вовсе не потребует создания объектов. Будет использоваться уже созданный объект – Activity

Activity, как обработчик

Кнопка присваивает себе обработчика с помощью метода setOnClickListener (View.OnClickListener l). Т.е. подойдет любой объект с интерфейсом View.OnClickListener. Почему бы классу Activity не быть таким объектом? Мы просто укажем, что Activity-класс реализует интерфейс View.OnClickListener и заполним метод onCreate.

Создадим для этого новый проект:
Project name: P0102_ActivityListener
Build Target: Android 2.3.3
Application name: ActivityListener
Package name: ru.startandroid.develop.activitylistener
Create Activity: MainActivity

Экран снова возьмем тот же самый:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_height="match_parent"
      android:layout_width="match_parent"
      android:orientation="horizontal">
      <LinearLayout
            android:id="@+id/linearLayout1"
            android:layout_height="match_parent"
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_margin="30dp">
            <TextView
                  android:layout_width="wrap_content"
                  android:text="TextView"
                  android:layout_height="wrap_content"
                  android:id="@+id/tvOut"
                  android:layout_gravity="center_horizontal"
                  android:layout_marginBottom="50dp">
            </TextView>
            <Button
                  android:layout_height="wrap_content"
                  android:layout_gravity="center_horizontal"
                  android:id="@+id/btnOk"
                  android:text="OK"
                  android:layout_width="100dp">
            </Button>
            <Button
                  android:layout_height="wrap_content"
                  android:layout_gravity="center_horizontal"
                  android:id="@+id/btnCancel"
                  android:text="Cancel"
                  android:layout_width="100dp">
            </Button>
      </LinearLayout>
</LinearLayout>
```

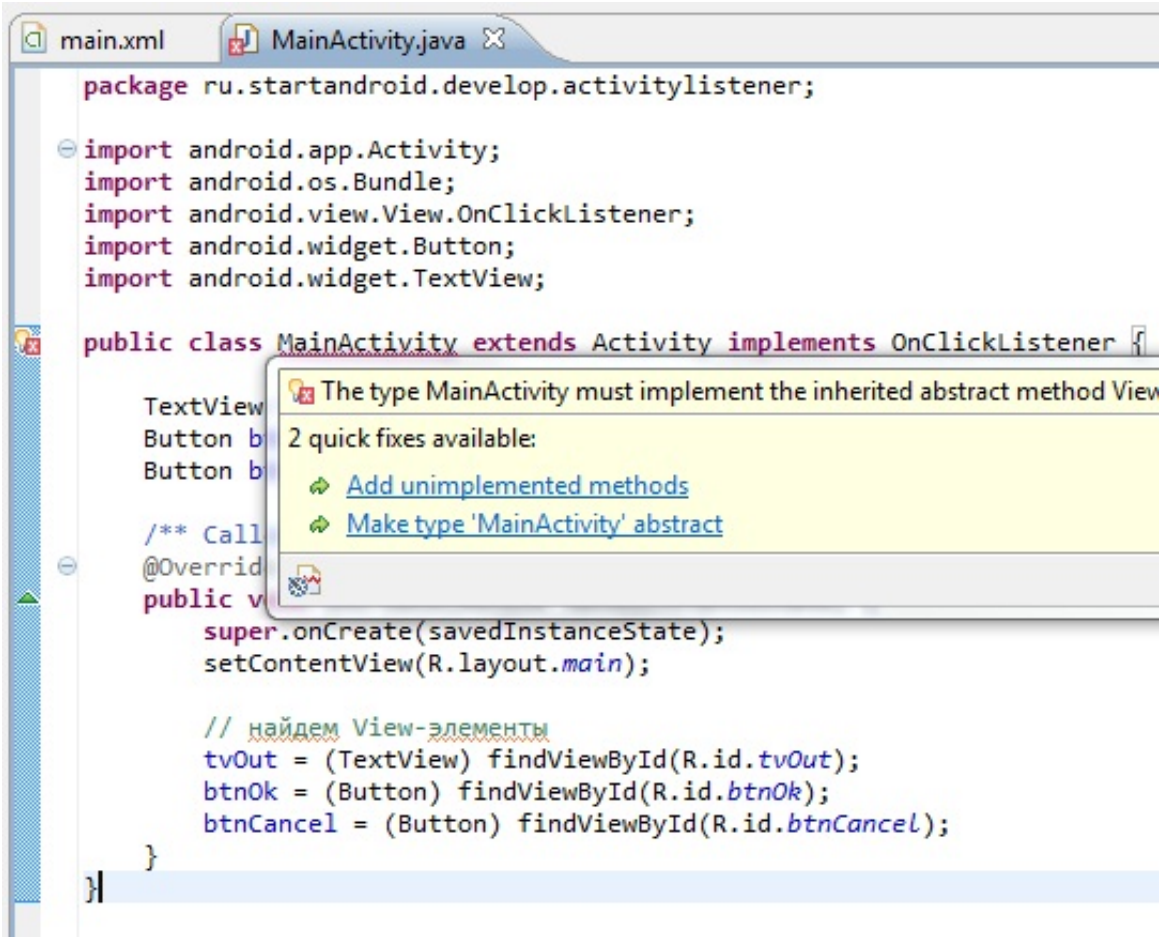
Подготовим объекты и добавим реализацию интерфейса (implements onClickListener)

```
public class MainActivity extends Activity implements OnClickListener {

    TextView tvOut;
    Button btnOk;
    Button btnCancel;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      // найдем View-элементы
      tvOut = (TextView) findViewById(R.id.tvOut);
      btnOk = (Button) findViewById(R.id.btnOk);
      btnCancel = (Button) findViewById(R.id.btnCancel);
```

```
    }
}
```

OnClickListener подчеркнут красным, т.к. его нет в импорте. Поэтому CTRL+SHIFT+O и выбираем View.OnClickListener.

Теперь Eclipse ругается на класс MainActivity. Это происходит потому, что для класса **прописан интерфейс**, но **нет реализации методов** этого интерфейса. Исправим это с помощью Eclipse. Наведите курсор на **MainAcivity** и выберите **Add unimplemented methods**



Eclipse добавит знакомый нам метод **onClick**. Только теперь этот метод будет реализован в Activity, а не в отдельном объекте-обработчике. Соответственно Activity и будет выступать обработчиком.

Заполним метод точно так же как и раньше. Ничего не изменилось. Ему на вход так же подается View (на которой произошло событие), по Id мы определим, какая именно эта View и выполним соответствующие действия:

```
  public void onClick(View v) {
    // по id определеяем кнопку, вызвавшую этот обработчик
    switch (v.getId()) {
    case R.id.btnOk:
      // кнопка ОК
      tvOut.setText("Нажата кнопка ОК");
      break;
    case R.id.btnCancel:
      // кнопка Cancel
      tvOut.setText("Нажата кнопка Cancel");
      break;
    }
  }
```

Осталось в методе onCreate присвоить обработчик кнопкам. Это будет объект this, т.е. текущий объект MainActivity.

```
    btnOk.setOnClickListener(this);
    btnCancel.setOnClickListener(this);
```

При такой реализации мы не создали ни одного лишнего объекта (Activity создается в любом случае) и затраты памяти минимальны, это рекомендуемый метод. Но, возможно, такой способ покажется сложным и непонятным, особенно если мало опыта в объектно-ориентированном программировании. В таком случае используйте ту реализацию, которая вам понятна и удобна. А со временем и опытом понимание обязательно придет.

Полный код:

```
public class MainActivity extends Activity implements OnClickListener {

   TextView tvOut;
   Button btnOk;
   Button btnCancel;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);

     // найдем View-элементы
     tvOut = (TextView) findViewById(R.id.tvOut);
     btnOk = (Button) findViewById(R.id.btnOk);
     btnCancel = (Button) findViewById(R.id.btnCancel);

     // присваиваем обработчик кнопкам
     btnOk.setOnClickListener(this);
     btnCancel.setOnClickListener(this);
   }

   @Override
   public void onClick(View v) {
     // по id определеяем кнопку, вызвавшую этот обработчик
     switch (v.getId()) {
     case R.id.btnOk:
       // кнопка OK
       tvOut.setText("Нажата кнопка OK");
       break;
     case R.id.btnCancel:
       // кнопка Cancel
       tvOut.setText("Нажата кнопка Cancel");
       break;
     }
   }

 }
```

Самая простая реализация обработчика

Есть еще один способ реализации. В layout-файле (main.xml) при описании кнопки пишем:

```
<?xml version="1.0" encoding="utf-8"?>
<Button
      android:id="@+id/btnStart"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="onClickStart"
      android:text="start">
</Button>
```

Т.е. используем атрибут onClick. В нем указываем имя метода из Activity. Этот метод и сработает при нажатии на кнопку.

Далее, добавляем этот метод в Activity (MainActivity.java). Требования к методу: public, void и на вход принимает View:

```
  public void onClickStart(View v) {
    // действия при нажати на кнопку
  }
```

В методе прописываете необходимые вам действия, и они будут выполнены при нажатии кнопки.

Урок 11. Папка res/values. Используем ресурсы приложения.

В этом уроке мы:

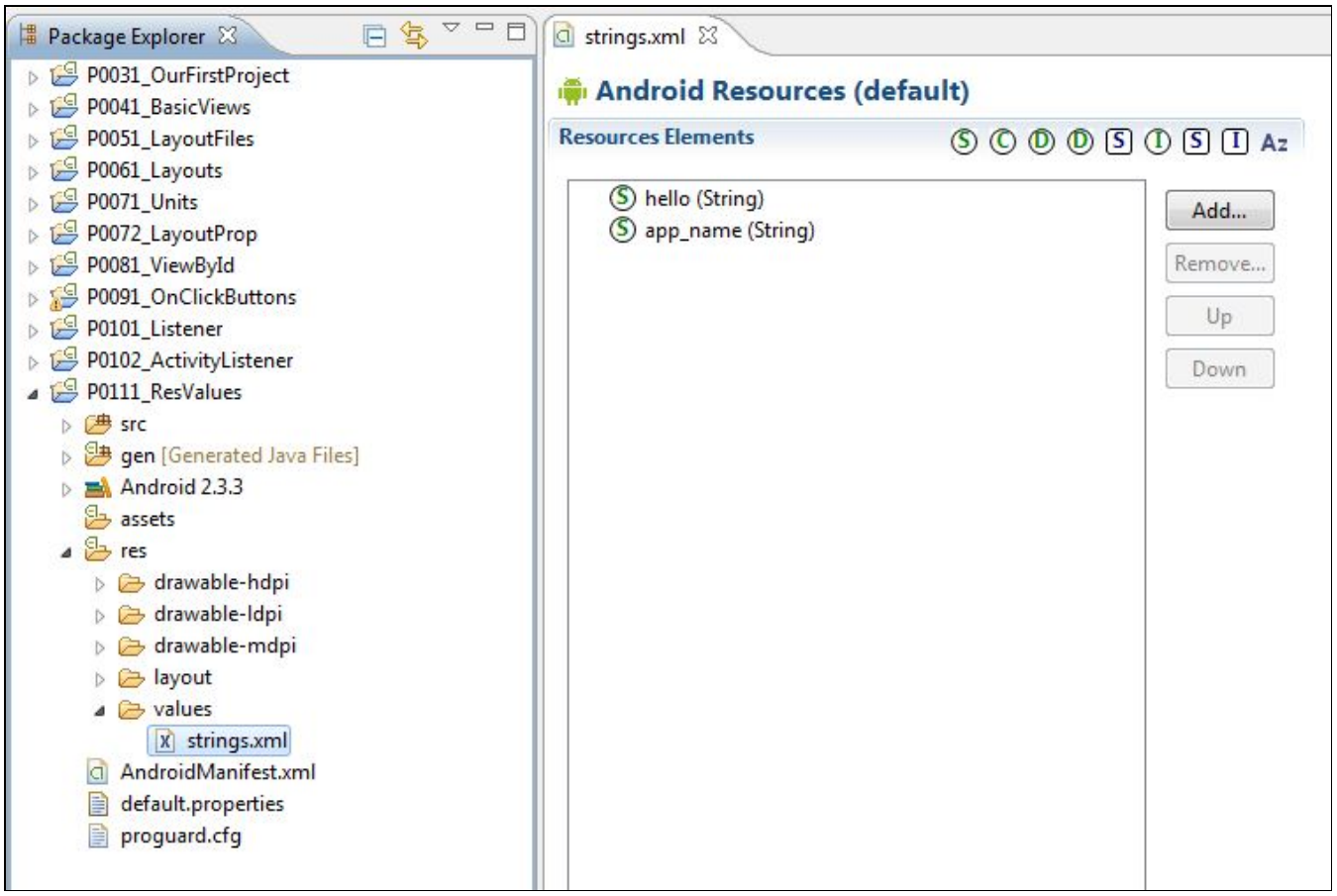
- узнаем, зачем нужна папка res/values, что в ней можно хранить и как использовать

В подпапках res хранятся различные **ресурсы** приложения. Мы уже отлично знаем про **layout-файлы** в папке **res/layout**. Я упоминал про папку **res/drawable** с density-суффиксами – в ней хранятся **картинки**. Теперь обратим внимание на папку **res/values**. Она предназначена для хранения ресурсов (констант) различных типов. Мы рассмотрим типы **String** и **Color**.

Создадим проект:

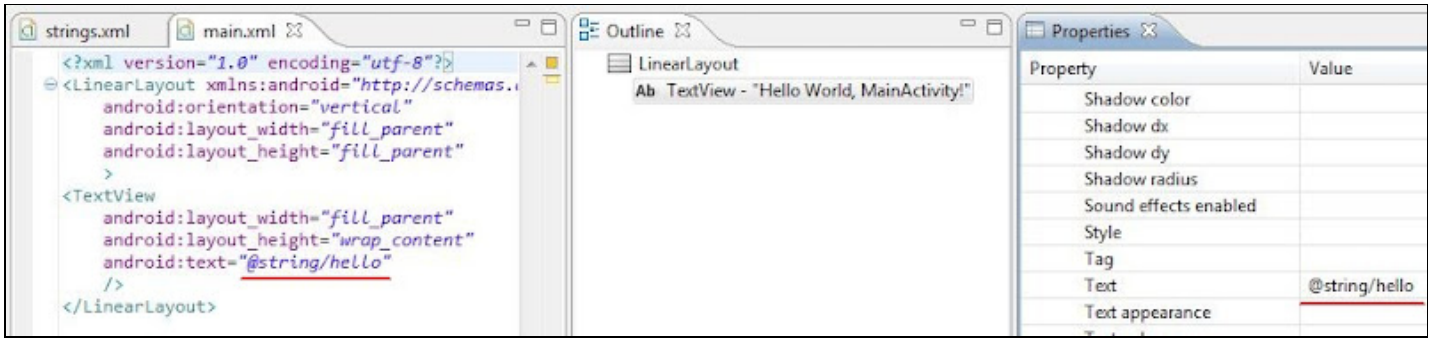
Project name: P0111_ResValues
Build Target: Android 2.3.3
Application name: ResValues
Package name: ru.startandroid.develop.resvalues
Create Activity: MainActivity

Откроем файл **res/values/strings.xml**



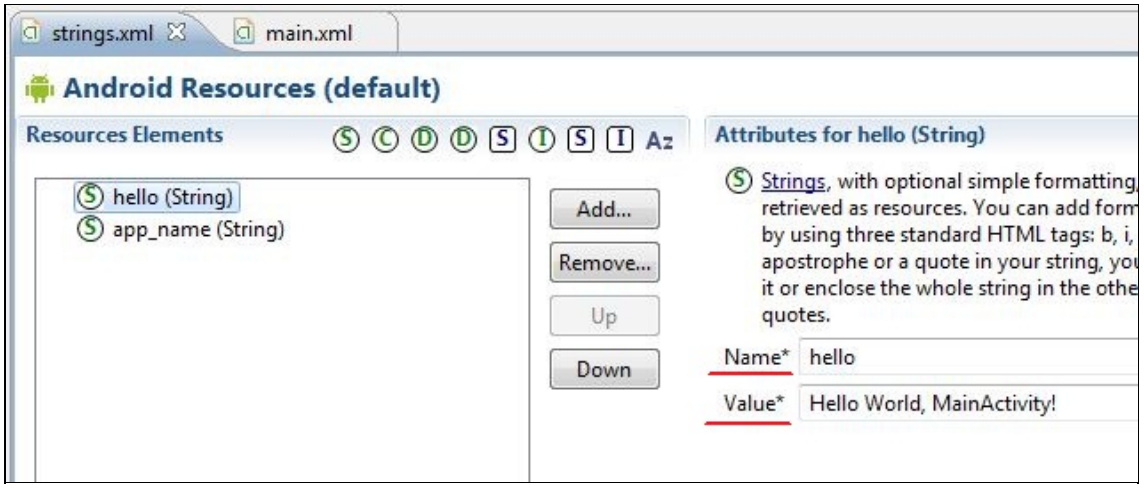
Мы видим два элемента типа String:

hello – по умолчанию он использован в свойстве Text в TextView в main.xml. И соответственно TextView отображает значение этого элемента.

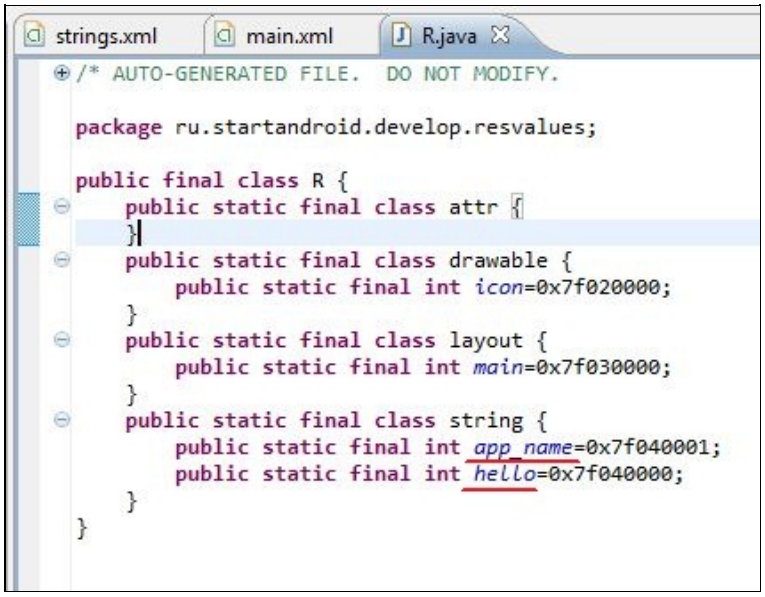


app_name – по умолчанию используется как заголовок для приложения и Activity. Это указывается в манифест-файле, который мы еще не разбирали.

На эти элементы можно кликнуть и увидеть справа, что они собой представляют: **имя** (Name) и **значение** (Value)



Name – это **ID**. Оно должно быть уникальным, и для него в **R.java** создается константа, чтобы мы могли иметь доступ к этому String-элементу.



Если мы посмотрим XML-содержимое файла strings.xml (вкладка снизу – аналогично как для main.xml), то видим, что там все прозрачно и просто. Попробуем и мы использовать ресурсы.

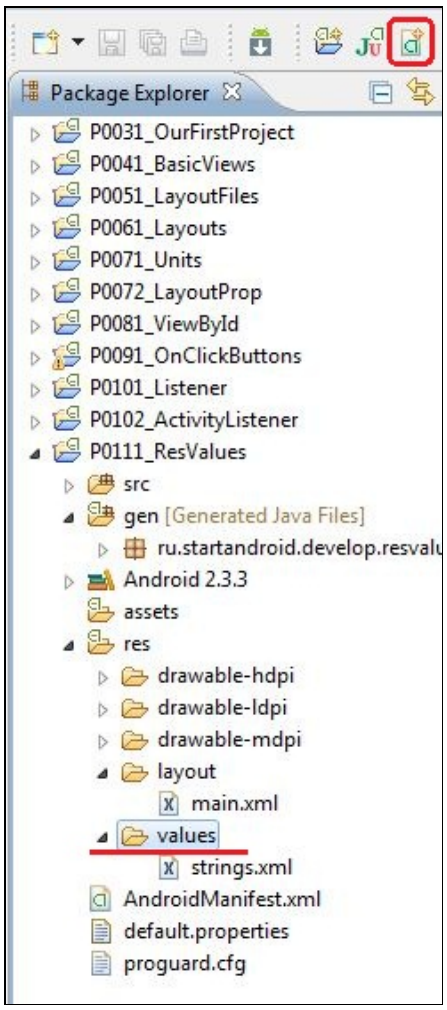
Для начала создадим такой экран в **main.xml**:

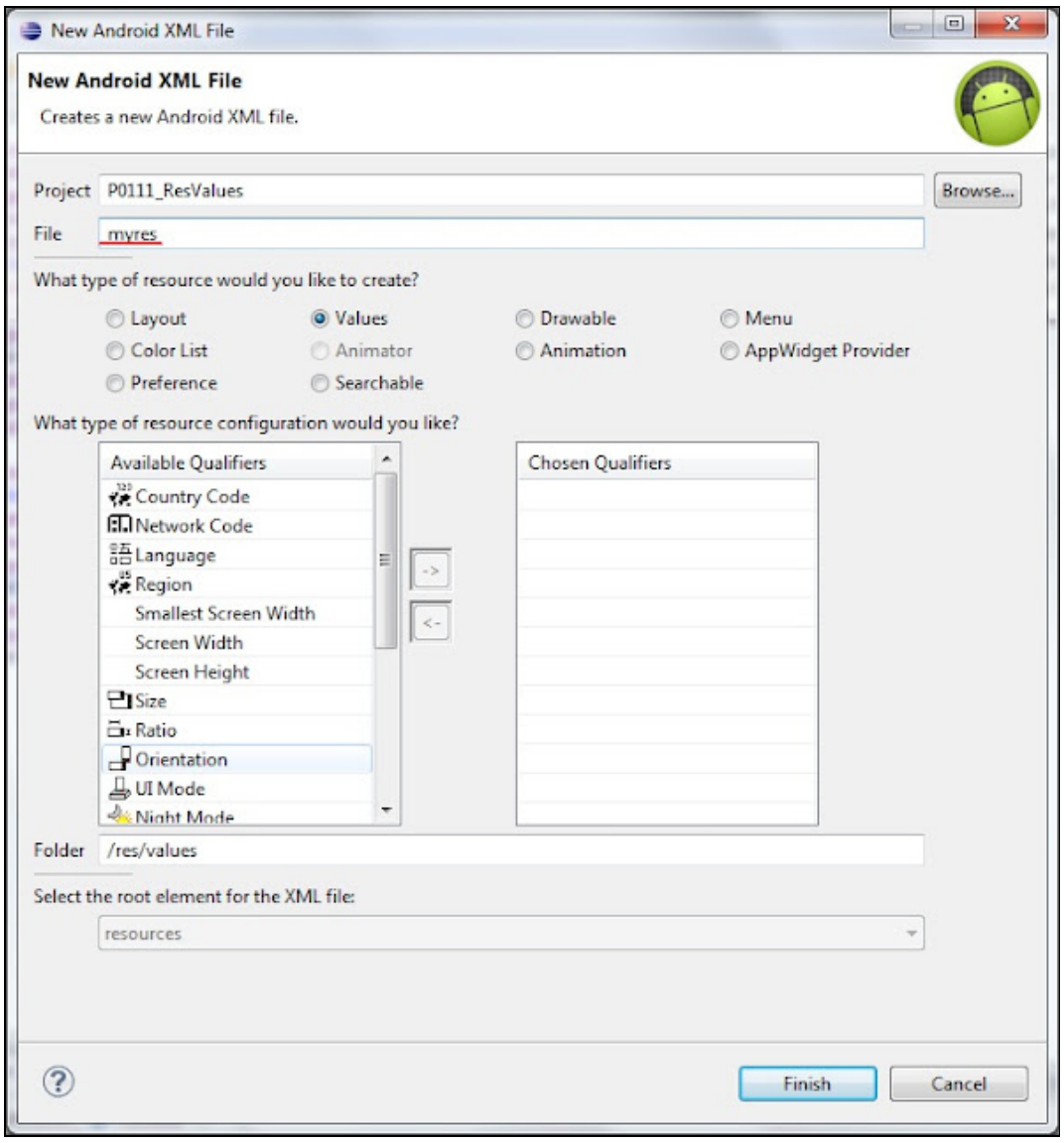
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:layout_height="match_parent"
	android:layout_width="match_parent"
	android:orientation="vertical">
	<LinearLayout
		android:layout_width="match_parent"
		android:id="@+id/llTop"
		android:orientation="vertical"
		android:layout_weight="1"
		android:layout_height="match_parent">
		<TextView
			android:text="TextView"
			android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:id="@+id/tvTop"
            android:layout_marginTop="30dp">
        </TextView>
        <Button
            android:text="Button"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:id="@+id/btnTop"
            android:layout_width="wrap_content">
        </Button>
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:id="@+id/llBottom"
        android:orientation="vertical"
        android:layout_weight="1"
        android:layout_height="match_parent">
        <TextView
            android:text="TextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:id="@+id/tvBottom"
            android:layout_marginTop="30dp">
        </TextView>
        <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:id="@+id/btnBottom">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Экран разделен на две одинаковые половины, содержащие LinearLayout, Button и TextView. Для **LinearLayout** мы укажем **фоновый цвет**, а для **TextView** и **Button** – изменим **текст**. Реализуем это с помощью ресурсов. Причем View-элементы **верхней** части мы будем настраивать вручную через **properties**, а **нижнюю** часть попробуем настроить **программно**.

Давайте создадим свой файл с ресурсами в папке values, название пусть будет myres.


Package Explorer
P0031_OurFirstProject
P0041_BasicViews
P0051_LayoutFiles
P0061_Layouts
P0071_Units
P0072_LayoutProp
P0081_ViewById
P0091_OnClickButtons
P0101_Listener
P0102_ActivityListener
P0111_ResValues
src
gen [Generated Java Files]
ru.startandroid.develop.resvalu
Android 2.3.3
assets
res
drawable-hdpi
drawable-ldpi
drawable-mdpi
layout
main.xml
values
strings.xml
AndroidManifest.xml
default.properties
proguard.cfg



После создания открылся редактор файла. Добавлять элемент просто – жмем кнопку **Add** и выбираем **тип**, а справа пишем **имя** и **значение**. Создадим 4 String-элемента и 2 Color-элемента:

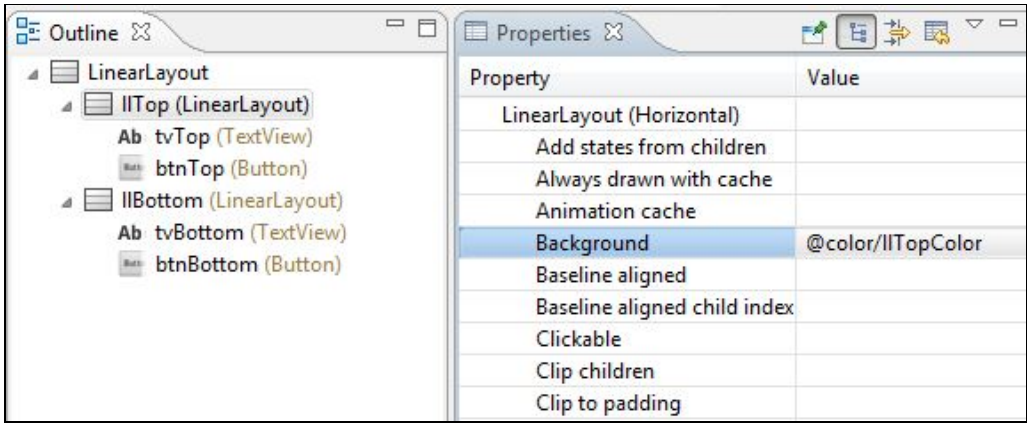
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="tvTopText">Верхний текст</string>
    <string name="btnTopText">Верхняя кнопка</string>
    <string name="tvBottomText">Нижний текст</string>
    <string name="btnBottomText">Нижняя кнопка</string>
    <color name="llTopColor">#336699</color>
    <color name="llBottomColor">#339966</color>
</resources>
```

Для практики можете создать вручную, а можете просто вставить этот текст в содержимое myres.xml. Не забудьте сохранить. Заглянем в R.java, убедимся, что здесь все появилось:

```
public final class R {
    public static final class attr {
    }
    public static final class color {
        public static final int llBottomColor=0x7f050001;
        public static final int llTopColor=0x7f050000;
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int btnBottom=0x7f060005;
        public static final int btnTop=0x7f060002;
        public static final int llBottom=0x7f060003;
        public static final int llTop=0x7f060000;
        public static final int tvBottom=0x7f060004;
        public static final int tvTop=0x7f060001;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040005;
        public static final int btnBottomText=0x7f040003;
        public static final int btnTopText=0x7f040001;
        public static final int hello=0x7f040004;
        public static final int tvBottomText=0x7f040002;
        public static final int tvTopText=0x7f040000;
    }
}
```

Ок, ресурсы созданы, настроим View-элементы на их использование. Сначала верхние:

llTop – в Properties находим свойство **Background**, жмем кнопку выбора (три точки), в ветке **Color** выделяем **llTopColor** и жмем **OK**



tvTop – для свойства Text откройте окно выбора и найдите там **tvTopText**.

btnTop - для свойства Text откройте окно выбора и найдите там **btnTopText**.

Цвет верхней части изменился и тексты поменялись на те, что мы указывали в myres.xml.

Чтобы изменить нижнюю часть, будем кодить. Сначала находим элементы, потом присваиваем им значения.

```
public class MainActivity extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    LinearLayout llBottom = (LinearLayout) findViewById(R.id.llBottom);
    TextView tvBottom = (TextView) findViewById(R.id.tvBottom);
    Button btnBottom = (Button) findViewById(R.id.btnBottom);
```

```
        llBottom.setBackgroundResource(R.color.llBottomColor);
        tvBottom.setText(R.string.tvBottomText);
        btnBottom.setText(R.string.btnBottomText);

    }
}
```

Обратите внимание на то, что для смены текста используется метод **setText**. Только это не тот же setText, что мы использовали, когда задавали текст напрямую. Этот **на вход** принимает **ID** и мы используем R.java, который хранит ID всех наших ресурсов. Т.е. методы называются одинаково, но на вход принимают разные параметры. Это нормальное явление в Java.

Сохраняем, запускаем и проверяем. Теперь тексты и цвета взяты из файла ресурсов. Вы можете изменить содержимое myres.xml (например текст для верхней кнопки), сохранить, запустить приложение и увидите изменения.

Иногда необходимо в коде получить не ID ресурса, а его значение. Это делается следующим образом:

```
        getResources().getString(R.string.tvBottomText);
```

Выражение вернет текст «Нижний текст», соответствующий String-ресурсу с name = **tvBottomText**.

Напоследок скажу пару слов об организации файлов для хранения ресурсов. Мы сейчас создали String и Color ресурсы **в одном файле** myres.xml, но рекомендуется их разделять **по разным файлам** (например strings.xml, colors.xml ...), и в дальнейшем я буду следовать этой рекомендации. Для этого есть причины, позже мы в этом убедимся.

Имена ресурсов сквозные для всех файлов в папке res/values. Т.е. вы не можете в разных файлах создать ресурс с одним именем и типом.

Имена файлов ресурсов могут быть произвольными и файлов можно создавать сколько угодно. В R.java попадут все ресурсы из этих файлов.

На следующем уроке:

- рассмотрим логи приложения и всплывающие сообщения

Урок 12. Логи и всплывающие сообщения

В этом уроке мы:

- рассмотрим логи приложения и всплывающие сообщения

Создадим проект:

Project name: P0121_LogAndMess
Build Target: Android 2.3.3
Application name: LogAndMess
Package name: ru.startandroid.develop.logandmess
Create Activity: MainActivity

Создадим в **main.xml** экран, знакомый нам по прошлым урокам про обработчики:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:layout_height="match_parent"
	android:layout_width="match_parent"
	android:orientation="horizontal">
	<LinearLayout
		android:id="@+id/linearLayout1"
		android:layout_height="match_parent"
		android:orientation="vertical"
		android:layout_width="match_parent"
		android:layout_margin="30dp">
		<TextView
			android:layout_width="wrap_content"
			android:text="TextView"
			android:layout_height="wrap_content"
			android:id="@+id/tvOut"
			android:layout_gravity="center_horizontal"
			android:layout_marginBottom="50dp">
		</TextView>
		<Button
			android:layout_height="wrap_content"
			android:layout_gravity="center_horizontal"
			android:id="@+id/btnOk"
			android:text="OK"
			android:layout_width="100dp">
		</Button>
		<Button
			android:layout_height="wrap_content"
			android:layout_gravity="center_horizontal"
			android:id="@+id/btnCancel"
			android:text="Cancel"
			android:layout_width="100dp">
		</Button>
	</LinearLayout>
</LinearLayout>
```

Алгоритм приложения будет тот же. По нажатию кнопок меняется текст. Обработчик - Activity.

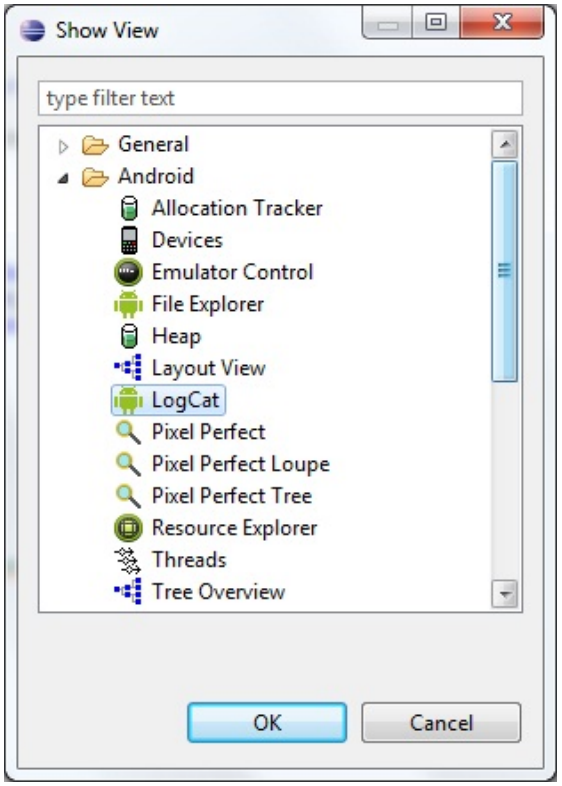
```
public class MainActivity extends Activity implements OnClickListener {

  TextView tvOut;
  Button btnOk;
  Button btnCancel;

  /** Called when the activity is first created. */
```

```
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // найдем View-элементы
    tvOut = (TextView) findViewById(R.id.tvOut);
    btnOk = (Button) findViewById(R.id.btnOk);
    btnCancel = (Button) findViewById(R.id.btnCancel);

    // присваиваем обработчик кнопкам
    btnOk.setOnClickListener(this);
    btnCancel.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
    // по id определеяем кнопку, вызвавшую этот обработчик
    switch (v.getId()) {
    case R.id.btnOk:
      // кнопка OK
      tvOut.setText("Нажата кнопка OK");
      break;
    case R.id.btnCancel:
      // кнопка Cancel
      tvOut.setText("Нажата кнопка Cancel");
      break;
    }
  }

}
```

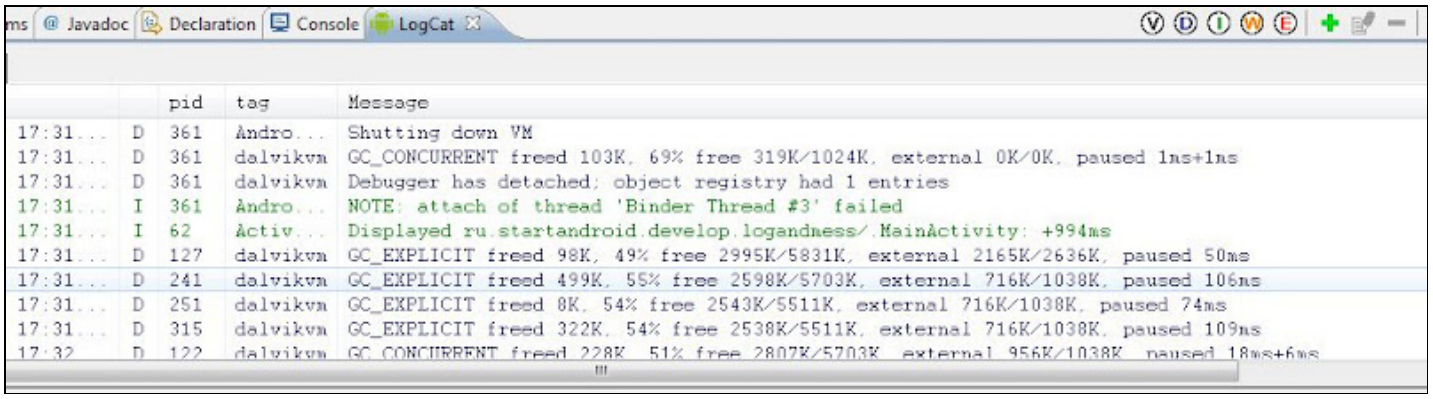
Сохраним, запустим. Убедимся, что все работает.

Логи приложения

Когда вы тестируете работу приложения, вы можете видеть ***логи*** работы. Они отображаются в окне **LogCat**. Чтобы отобразить окно откройте меню **Window > Show View > Other** ... В появившемся окне выберите **Android > LogCat**



Должна появится вкладка LogCat



Рассмотрим эту вкладку подробней. Логи имеют разные уровни важности: **ERROR**, **WARN**, **INFO**, **DEBUG**, **VERBOSE** (по убыванию). Кнопки V D I W E (в кружках) – это фильтры и соответствуют типам логов. Опробуйте их и обратите внимание, что фильтр показывает логи не только своего уровня, но и уровней более высокой важности. Также вы можете создавать, редактировать и удалять свои фильтры – это мы рассмотрим чуть дальше.

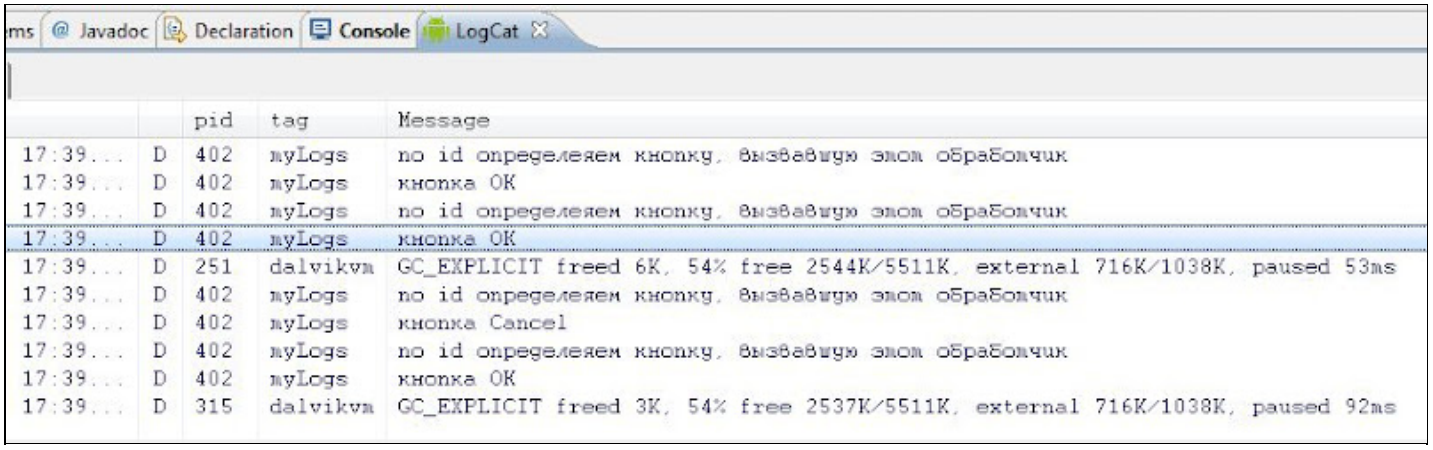
Давайте смотреть, как самим писать логи. Делается это совсем несложно с помощью класса Log и его методов Log.v() Log.d() Log.i() Log.w() and Log.e(). Названия методов соответствуют уровню логов, которые они запишут.

Изменим код **MainActivity.java**. Возьмем все каменты из кода и добавим в DEBUG-логи с помощью метода Log.d. Метод требует на вход **тэг** и **текст** сообщения. Тэг – это что-то типа метки, чтобы легче было потом в куче системных логов найти именно наше сообщение. Добавим описание тега (TAG) и запишем все тексты каментов в лог.

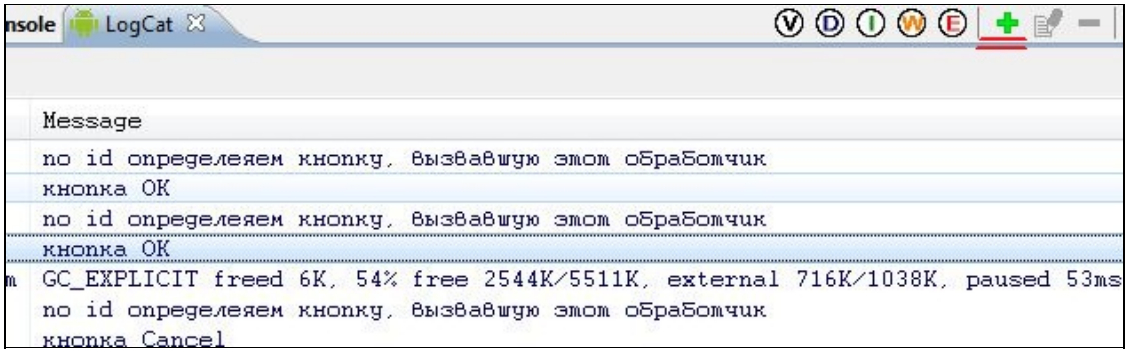
```
public class MainActivity extends Activity implements OnClickListener {

  TextView tvOut;
  Button btnOk;
  Button btnCancel;

  private static final String TAG = "myLogs";

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // найдем View-элементы
    Log.d(TAG, "найдем View-элементы");
    tvOut = (TextView) findViewById(R.id.tvOut);
    btnOk = (Button) findViewById(R.id.btnOk);
    btnCancel = (Button) findViewById(R.id.btnCancel);

    // присваиваем обработчик кнопкам
    Log.d(TAG, "присваиваем обработчик кнопкам");
    btnOk.setOnClickListener(this);
    btnCancel.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
    // по id определеяем кнопку, вызвавшую этот обработчик
    Log.d(TAG, "по id определеяем кнопку, вызвавшую этот обработчик");
    switch (v.getId()) {
    case R.id.btnOk:
      // кнопка ОК
      Log.d(TAG, "кнопка ОК");
      tvOut.setText("Нажата кнопка ОК");
      break;
    case R.id.btnCancel:
      // кнопка Cancel
      Log.d(TAG, "кнопка Cancel");
      tvOut.setText("Нажата кнопка Cancel");
      break;
```

```
        }
    }

}
```

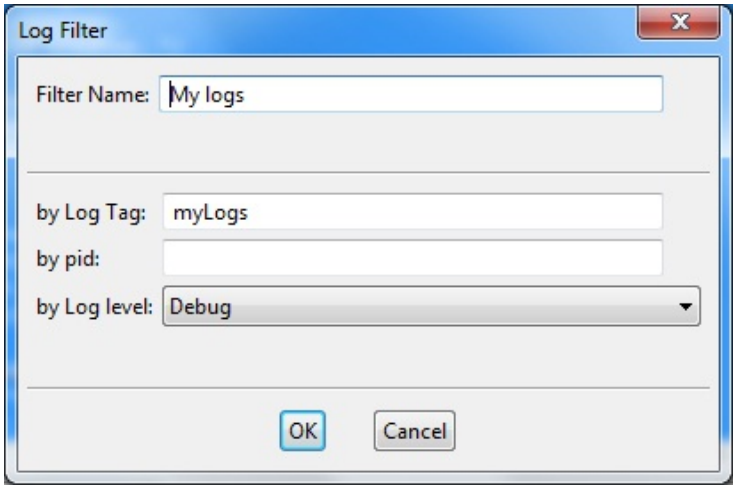
Eclipse ругнется, что не знает класс Log. Обновите импорт (CTRL+SHIFT+O) и, если спросит, выберите **android.util.Log**. Запустим приложение, понажимаем кнопки и посмотрим логи



Видно, что все отлично записалось. Чтобы сделать просмотр удобней, создадим свой фильтр. Жмем значок +



Имя фильтра произвольное, например, «**My logs**». **Log Tag** – это как раз значение константы TAG, которая описан в нашем коде и использовалась в методе Log.d, т.е. - "**myLogs**". **Pid** оставляем **пустым**, это id процесса. **Уровень** поставим **Debug**



и жмем OK. Появилась новая вкладка My logs, на которой отображаются логи, соответствующие только что созданному фильтру.

Мы помещали в лог текст, но разумеется, вы можете писать, например, значения интересующих вас переменных (приведенные к типу String).

Иногда бывает, что логи не отображаются во вкладке LogCat, хотя AVD запущен, приложение работает без проблем. В таком случае должно

помочь следующее: в Eclipse идем в меню Window > Open Perspective > Other > DDMS. Откроется немного другой набор окон чем обычно. Там найдите вкладку Devices и в ней должно быть видно ваше AVD-устройство, кликните на него и логи должны появиться. Чтобы вернуться в разработку: Window > Open Perspective > Java.

Всплывающие сообщения

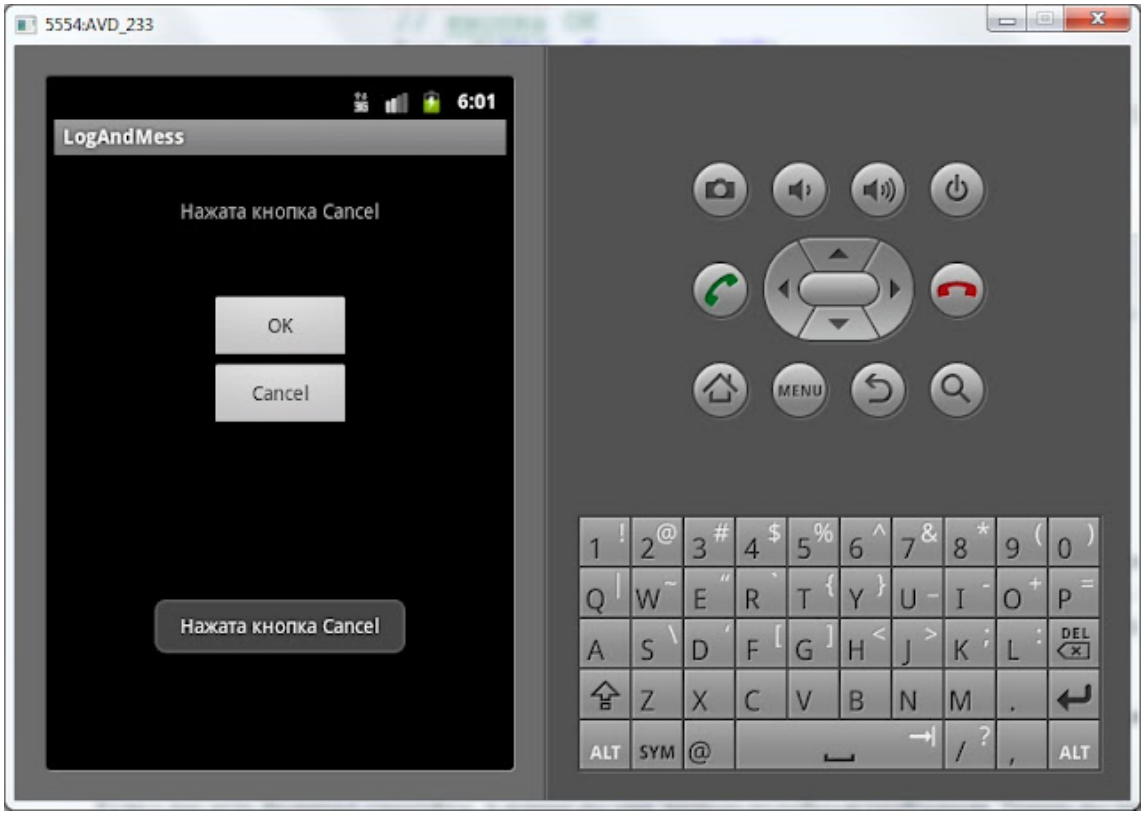
Приложение может показывать **всплывающие сообщения** с помощью класса **Toast**. Давайте подредактируем метод onClick. Сделаем так, чтобы всплывало сообщение о том, какая кнопка была нажата.

```
public void onClick(View v) {
  // по id определеяем кнопку, вызвавшую этот обработчик
  Log.d(TAG, "по id определеяем кнопку, вызвавшую этот обработчик");
  switch (v.getId()) {
  case R.id.btnOk:
    // кнопка OK
    Log.d(TAG, "кнопка OK");
    tvOut.setText("Нажата кнопка OK");
    Toast.makeText(this, "Нажата кнопка OK", Toast.LENGTH_LONG).show();
    break;
  case R.id.btnCancel:
    // кнопка Cancel
    Log.d(TAG, "кнопка Cancel");
    tvOut.setText("Нажата кнопка Cancel");
    Toast.makeText(this, "Нажата кнопка Cancel", Toast.LENGTH_LONG).show();
    break;
  }
}
```

Разберем синтаксис вызова. Статический метод makeText создает View-элемент Toast. Параметры метода:

- **context** – пока не будем вдаваться в подробности, что это такое и используем текущую Activity, т.е. this.
- **text** – текст, который надо показать
- **duration** – продолжительность показа (Toast.LENGTH_LONG - длинная, Toast.LENGTH_SHORT - короткая)

Toast создан и чтобы он отобразился на экране, вызывается метод **show**(). Сохраняем, запускаем, проверяем.



Если у вас есть Андроид-смартфон, я думаю вы уже видели подобные сообщения. Теперь вы знаете, как это делается)

На следующем уроке:

- создаем пункты меню

Урок 13. Создание простого меню

В этом уроке мы:

- создаем пункты меню

Что такое меню, думаю, нет смысла рассказывать. Оно отображается при нажатии кнопки Menu. Давайте создадим свое.

Создаем проект:

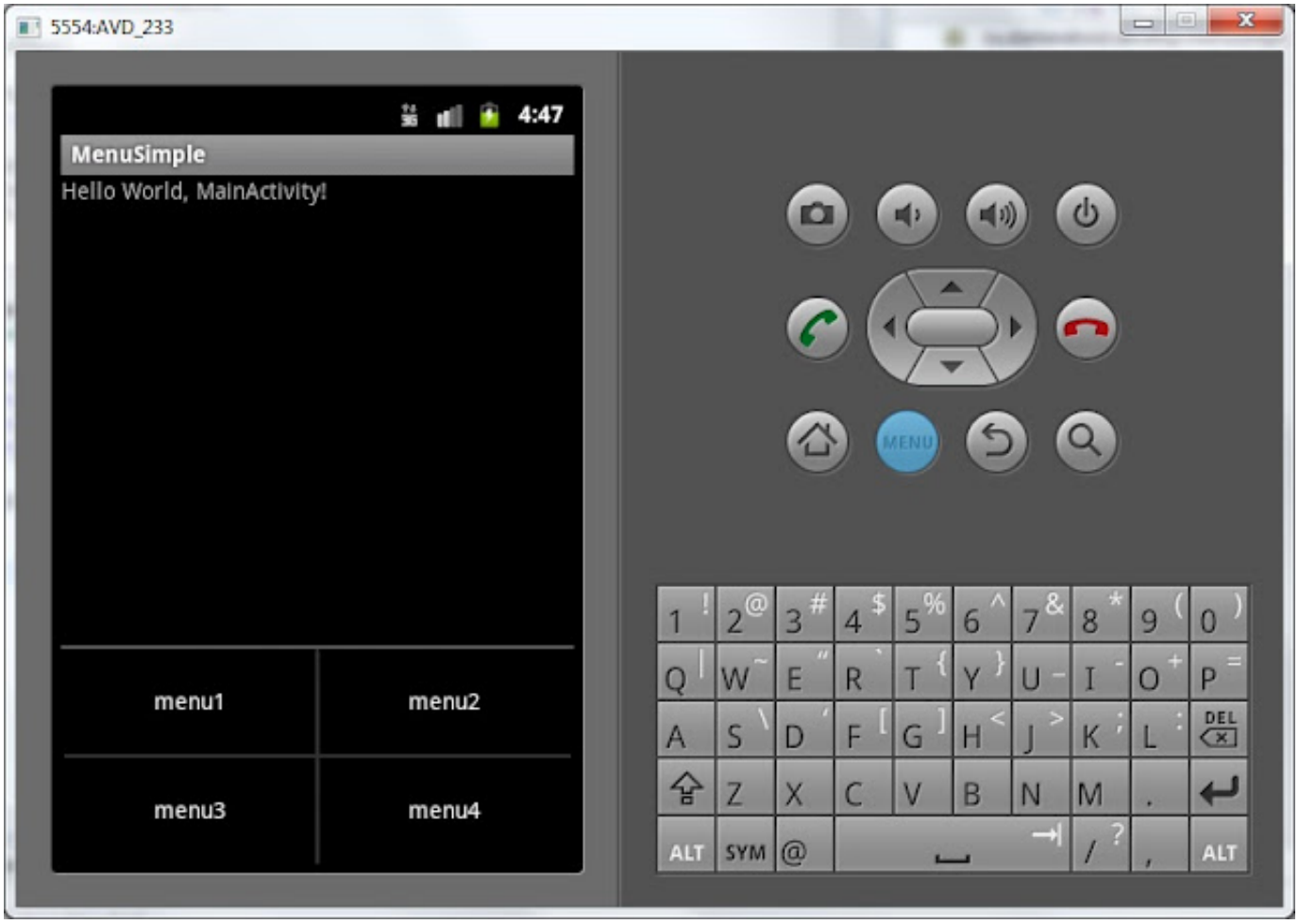
Project name: P0131_MenuSimple
Build Target: Android 2.3.3
Application name: MenuSimple
Package name: ru.startandroid.develop.menusimple
Create Activity: MainActivity

Откроем **MainActivity.java**. За создание меню отвечает метод onCreateOptionsMenu. На вход ему подается объект типа **Menu**, в который мы и будем добавлять свои пункты. Добавляются они просто, методом add. На вход методу подается **текст** пункта меню. Добавим 4 пункта.

```
    public boolean onCreateOptionsMenu(Menu menu) {
      // TODO Auto-generated method stub

      menu.add("menu1");
      menu.add("menu2");
      menu.add("menu3");
      menu.add("menu4");

      return super.onCreateOptionsMenu(menu);
    }
```

Метод onCreateOptionsMenu должен вернуть результат типа **boolean**. **True** – меню показывать, **False** – не показывать. Т.е. можно было бы накодить проверку какого-либо условия, и по итогам этой проверки не показывать меню передавая False. Пока нам это не нужно, поэтому поручаем этот выбор конструктору суперкласса, по умолчанию он возвращает True.

Сохраним все, запустим приложение и нажмем кнопку меню на эмуляторе.



Появилось 4 пункта меню. Нажатие на них ни к чему не приводит, т.к. не реализован обработчик. Обработчиком является Activity, а метод зовется onOptionsItemSelected. На вход ему передается пункт меню, который был нажат – **MenuItem**. Определить, какое именно меню было нажато можно по методу getTitle. Давайте выводить всплывающее сообщение с текстом нажатого пункта меню. На выходе метода надо возвращать **boolean**. И мы снова предоставляем это суперклассу.

```
    public boolean onOptionsItemSelected(MenuItem item) {
      // TODO Auto-generated method stub
      Toast.makeText(this, item.getTitle(), Toast.LENGTH_SHORT).show();
      return super.onOptionsItemSelected(item);
    }
```

Полный код:

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
      // TODO Auto-generated method stub
```

```
        menu.add("menu1");
        menu.add("menu2");
        menu.add("menu3");
        menu.add("menu4");

        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // TODO Auto-generated method stub
        Toast.makeText(this, item.getTitle(), Toast.LENGTH_SHORT).show();
        return super.onOptionsItemSelected(item);
    }

}
```

Определять нажатый пункт меню по тексту – это не самый лучший вариант. Далее будем делать это по **ID**. Но для этого надо немного по другому создавать меню.

На следующем уроке:

- создаем пункты меню с ID
- группируем и сортируем пункты меню

Урок 14. Меню, группы, порядок. MenuInflater и xml-меню.

В этом уроке мы:

- создаем пункты меню с ID
- группируем и сортируем пункты меню

На прошлом уроке мы рассмотрели простейший способ создания меню методом add(CharSequence title), на вход мы подавали только текст. Рассмотрим другую реализацию этого метода - add(int groupId, int itemId, int order, CharSequence title). У этого метода **4 параметра** на вход:
- **groupId** - идентификатор группы, частью которой является пункт меню
- **itemId** - ID пункта меню
- **order** - для задания последовательности показа пунктов меню
- **title** - текст, который будет отображен

Чтоб показать как используются все эти параметры, создадим приложение. На экране будет **TextView** и **CheckBox**:
- TextView будет отображать какой пункт меню был выбран
- CheckBox будет определять показывать обычное меню или расширенное. Это будет реализовано с помощью групп меню.

Сразу уточню, понятия "обычное" и "расширенное" - это не Андроид-понятия, а просто мои названия. Т.е. когда запущено приложение и пользователь жмет кнопку меню, он видит "обычное" меню. Если же он включит CheckBox, то будет отображаться "расширенное" меню, в котором больше пунктов.

Создаем проект:

Project name: P0141_MenuAdv
Build Target: Android 2.3.3
Application name: MenuAdv
Package name: ru.startandroid.develop.menuadv
Create Activity: MainActivity

Откроем **main.xml**, присвоим **ID** существующему **TextView**, сотрем его текст и создадим **CheckBox**. Код:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"
	android:layout_width="fill_parent"
	android:layout_height="fill_parent">
	<CheckBox
		android:layout_width="wrap_content"
		android:layout_height="wrap_content"
		android:id="@+id/chbExtMenu"
		android:text="расширенное меню">
	</CheckBox>
	<TextView
		android:layout_width="fill_parent"
		android:layout_height="wrap_content"
		android:id="@+id/textView">
	</TextView>
```

```
</LinearLayout>
```

Открываем **MainActivity.java** и класс **MainActivity** заполняем следующим кодом:

```
public class MainActivity extends Activity {

  // Элементы экрана
  TextView tv;
  CheckBox chb;


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // находим эдементы
        tv = (TextView) findViewById(R.id.textView);
        chb = (CheckBox) findViewById(R.id.chbExtMenu);

    }

    // создание меню
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
      // TODO Auto-generated method stub
      // добавляем пункты меню
      menu.add(0, 1, 0, "add");
      menu.add(0, 2, 0, "edit");
      menu.add(0, 3, 3, "delete");
      menu.add(1, 4, 1, "copy");
      menu.add(1, 5, 2, "paste");
      menu.add(1, 6, 4, "exit");

      return super.onCreateOptionsMenu(menu);
    }

    // обновление меню
    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
      // TODO Auto-generated method stub
      // пункты меню с ID группы = 1 видны, если в CheckBox стоит галка
      menu.setGroupVisible(1, chb.isChecked());
      return super.onPrepareOptionsMenu(menu);
    }

    // обработка нажатий
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
      // TODO Auto-generated method stub
      StringBuilder sb = new StringBuilder();

      // Выведем в TextView информацию о нажатом пункте меню
      sb.append("Item Menu");
      sb.append("\r\n groupId: " + String.valueOf(item.getGroupId()));
      sb.append("\r\n itemId: " + String.valueOf(item.getItemId()));
      sb.append("\r\n order: " + String.valueOf(item.getOrder()));
      sb.append("\r\n title: " + item.getTitle());
```

```
        tv.setText(sb.toString());

        return super.onOptionsItemSelected(item);
    }
}
```

Не забудьте **обновить импорт** (CTRL+SHIFT+O).

Давайте разбирать написанное. Мы используем следующие **методы**:

onCreateOptionsMenu - вызывается только **при первом показе** меню. Создает меню и более не используется. Здесь мы добавляем к меню пункты.

onPrepareOptionsMenu - вызывается **каждый раз** перед отображением меню. Здесь мы вносим изменения в уже созданное меню, если это необходимо

onOptionsItemSelected - вызывается **при нажатии** пункта меню. Здесь мы определяем какой пункт меню был нажат.

В методе **onCreateOptionsMenu** мы добавляем 6 пунктов меню. Обратим внимание на параметры метода **Add**.

Первый параметр – **ID группы**. В первых трех пунктах он равен нулю, в оставшихся трех – 1. Т.е. пункты меню **copy**, **paste** и **exit** объединены в группу с ID = **1**. Визуально это никак не проявляется - они не отличаются цветом или еще чем-либо. ID группы мы будем использовать в реализации **onPrepareOptionsMenu**.

Второй параметр – **ID пункта меню**. В обработчике используется для определения какой пункт меню был нажат. Будем использовать его в **onOptionsItemSelected**.

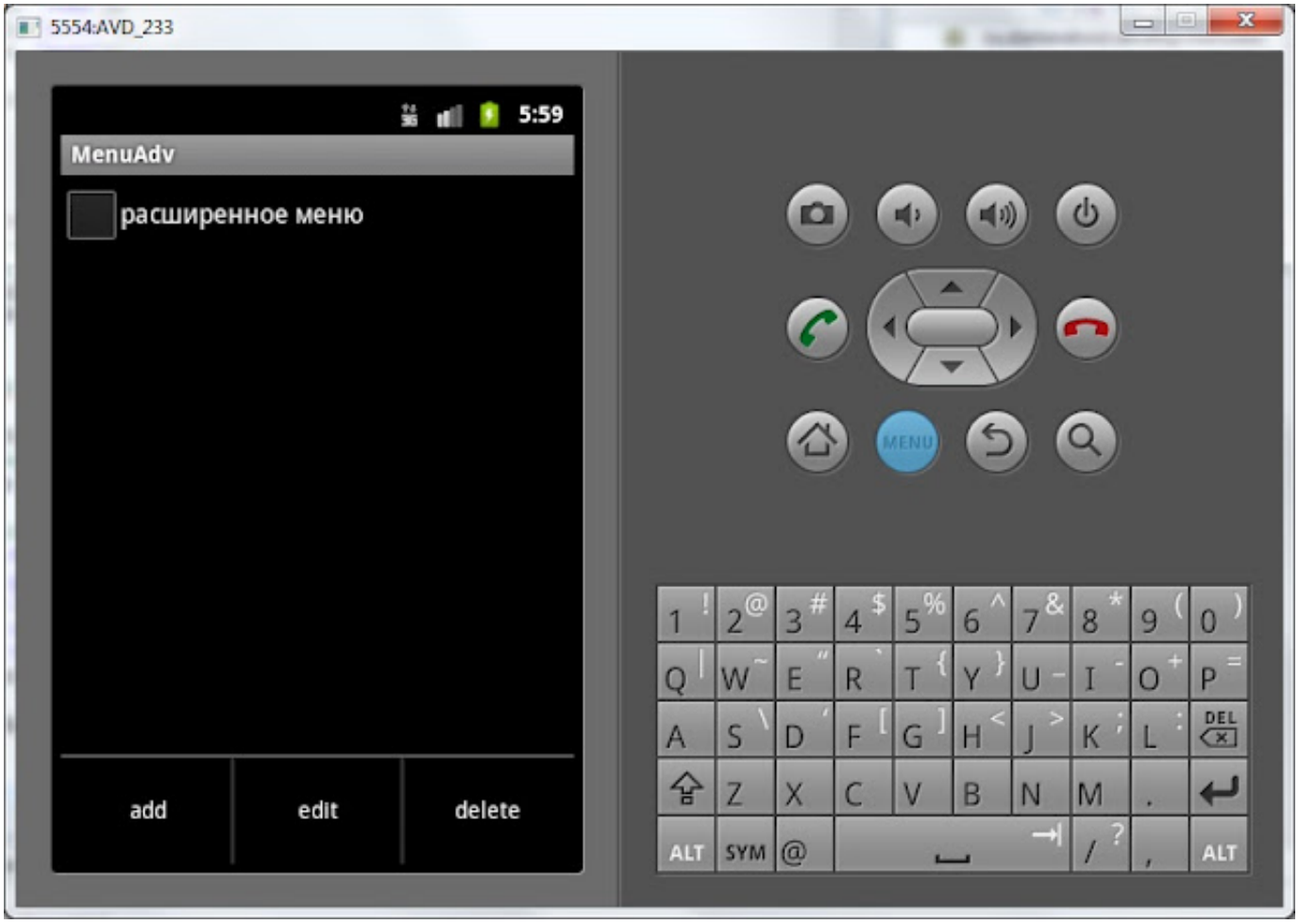
Третий параметр – определяет **позицию пункта** меню. Этот параметр используется для определения порядка пунктов при отображении меню. Используется **сортировка по возрастанию**, т.е. от меньшего **order** к большему.

Четвертый параметр – **текст**, который будет отображаться на пункте меню. Тут все понятно.

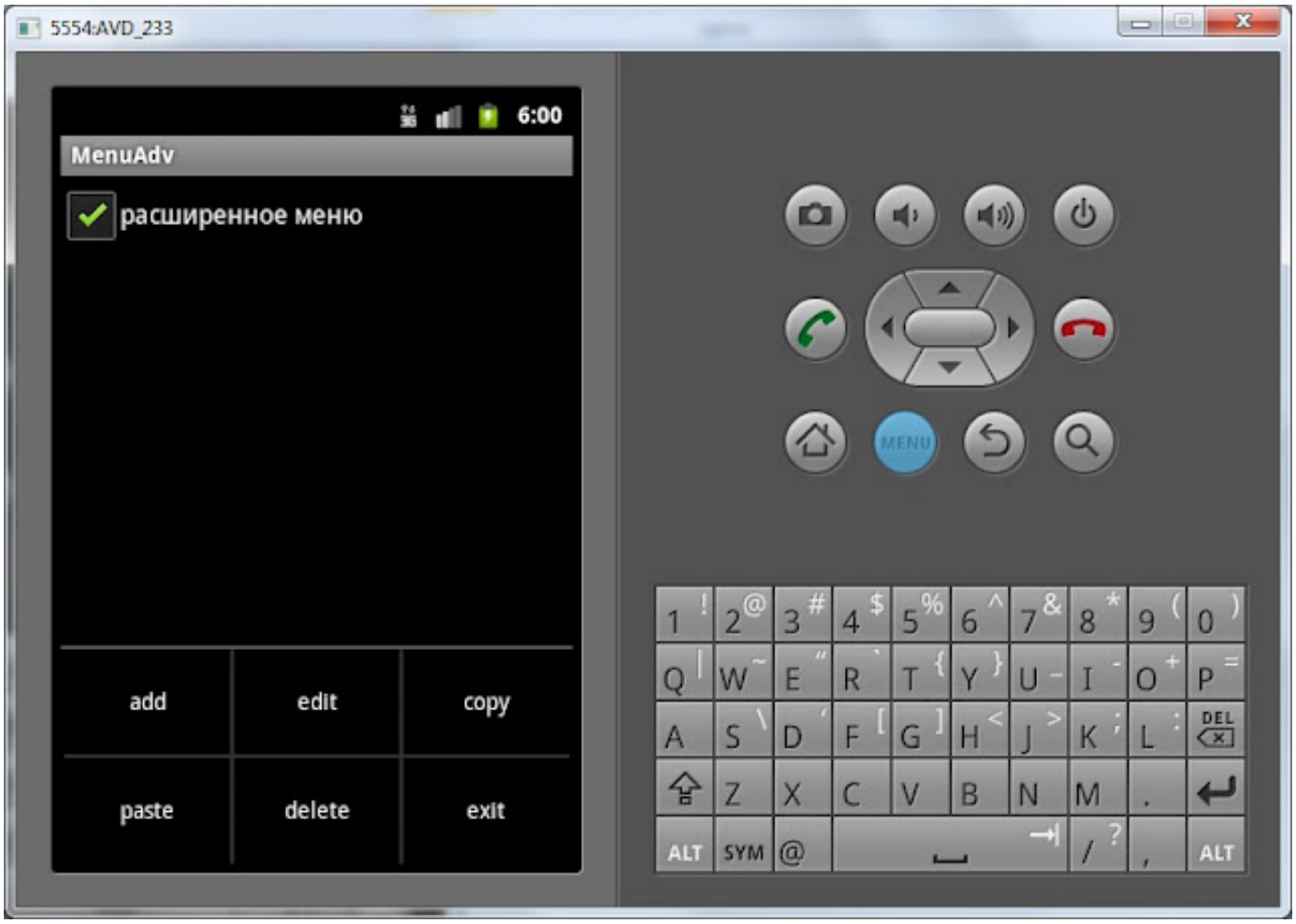
В метод **onPrepareOptionsMenu** передается объект **Menu** и мы можем работать с ним. В данном примере вызываем setGroupVisible. Этот метод позволяет **скрывать\отображать** пункты меню. На вход подается два параметра – **ID группы** и **boolean**-значение. В качестве ID группы мы пишем – **1** (та самая группа с ID = 1, в которой находятся пункты copy, paste и exit), а в качестве boolean параметра используем состояние **CheckBox**. Если он включен, то пункты меню (из **группы с ID = 1**) будут отображаться, если выключен – не будут.

Сохраним все и запустим приложение.

"**Обычное**" меню:



"**Расширенное**" меню



В зависимости от состояния **CheckBox** в меню видно **3** или **6** пунктов.

Обратите внимание на **порядок** пунктов. Они отсортированы по параметру **order** по **возрастанию**. Если **order** у нескольких пунктов **совпадает**, то эти пункты размещаются **в порядке их создания** в методе onCreateOptionsMenu.

При нажатии на какой-либо пункт меню срабатывает метод **onOptionsItemSelected**. В нем мы выводим в **TextView** информацию о нажатом пункте. Можете сверить эту информацию с тем, что мы кодили при создании пунктов меню. Все параметры должны совпадать. Порядок, для удобства, я сделал такой же как и в методе add: **groupId**, **itemId**, **order**, **title**.

Попробуйте добавить еще несколько пунктов в меню, чтобы их стало **больше шести**. И обратите внимание, как они отобразятся.

Для упрощения кода я использовал напрямую цифры для ID групп и ID пунктов меню. А вообще рекомендуется использовать константы, в дальнейшем буду использовать их.

XML-меню

Есть еще один, более удобный и предпочтительный способ создания меню - с использованием xml-файлов, аналогично layout-файлам при создании экрана. Чтобы получить меню, которые мы создавали программно на этом уроке, надо создать в папке res/menu файл **mymenu.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
     xmlns:android="http://schemas.android.com/apk/res/android">
    <item
         android:id="@+id/menu_add"
         android:title="add">
    </item>
    <item
         android:id="@+id/menu_edit"
         android:title="edit">
    </item>
    <item
         android:id="@+id/menu_delete"
         android:orderInCategory="3"
         android:title="delete">
    </item>
    <group
         android:id="@+id/group1">
        <item
             android:id="@+id/menu_copy"
             android:orderInCategory="1"
             android:title="copy">
        </item>
        <item
             android:id="@+id/menu_paste"
             android:orderInCategory="2"
             android:title="paste">
        </item>
        <item
```

```
            android:id="@+id/menu_exit"
            android:orderInCategory="4"
            android:title="exit">
        </item>
    </group>
</menu>
```

item - это пункт меню, **group** - группа. В атрибутах **ID** используем ту же схему, что и в ID экранных компонентов, т.е. пишем @+id/<your_ID> и Eclipse сам создаст эти ID в R.java. Атрибут **orderInCategory** - это порядок пунктов, а **title** - текст.

В методе **onCreateOptionsMenu** нам теперь не надо вручную кодить создание каждого пункта, мы просто свяжем menu, который нам дается на вход и наш xml-файл.

```
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.mymenu, menu);
    return super.onCreateOptionsMenu(menu);
  }
```

С помощью метода **getMenuInflater** мы получаем MenuInflater и вызываем его метод inflate. На вход передаем наш файл **mymenu.xml** из папки res/menu и объект **menu**. MenuInflater берет объект menu и наполняет его пунктами согласно файлу mymenu.xml.

Если захотите скрыть группу, выполняете тот же метод setGroupVisible и передаете туда R.id.group1 в качестве ID группы.

Подробно атрибуты для xml-файла меню можно посмотреть здесь.

Я вам рекомендую опробовать и потестить оба способа созданию меню. Программное создание гибче, а xml сокращает код.

На следующем уроке:

- создадим контекстное меню

Урок 15. Контекстное меню

В этом уроке мы:

- создадим контекстное меню

Контекстное меню вызывается в Андроид длительным нажатием на каком-либо экранном компоненте. Обычно оно используется в списках. Когда на экран выводится список однородных объектов (например письма в почт.ящике) и, чтобы выполнить действие с одним из этих объектов, мы вызываем контекстное меню для него. Но т.к. списки мы еще не проходили, сделаем пример попроще и будем вызывать контекстное меню для TextView.

Создадим проект:

Project name: P0151_ContextMenu
Build Target: Android 2.3.3
Application name: ContextMenu
Package name: ru.startandroid.develop.contextmenu
Create Activity: MainActivity

Откроем main.xml и нарисуем там два TextView:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"
	android:layout_width="fill_parent"
	android:layout_height="fill_parent">
	<TextView
		android:layout_height="wrap_content"
		android:textSize="26sp"
		android:layout_width="wrap_content"
		android:id="@+id/tvColor"
		android:layout_marginBottom="50dp"
		android:layout_marginTop="50dp"
		android:text="Text color">
	</TextView>
	<TextView
		android:layout_width="fill_parent"
		android:layout_height="wrap_content"
		android:textSize="22sp"
		android:id="@+id/tvSize"
		android:text="Text size">
	</TextView>
</LinearLayout>
```

Для первого TextView мы сделаем контекстное меню, с помощью которого будем менять цвет текста. Для второго – будем менять размер текста.

Принцип создания контекстного меню похож на создание обычного меню. Но есть и отличия.

Метод создания onCreateContextMenu вызывается каждый раз перед показом меню. На вход ему передается:

- **ContextMenu**, в который мы будем добавлять пункты
- **View** - элемент экрана, для которого вызвано контекстное меню
- **ContextMenu.ContextMenuInfo** – содержит доп.информацию, когда контекстное меню вызвано для элемента списка. Пока мы это не используем, но когда будем изучать списки увидим, что штука полезная.

Метод обработки onContextItemSelected аналогичный методу **onOptionsItemSelected** для обычного меню. На вход передается **MenuItem** – пункт меню, который был нажат.

Также нам понадобится третий метод registerForContextMenu. На вход ему передается **View** и это означает, что для этой View необходимо создавать контекстное меню. Если не выполнить этот метод, контекстное меню для View создаваться не будет.

Давайте кодить, открываем **MainActivity.java**. Опишем и найдем **TextView** и укажем, что необходимо создавать для них контекстное меню.

```
TextView tvColor, tvSize;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      tvColor = (TextView) findViewById(R.id.tvColor);
      tvSize = (TextView) findViewById(R.id.tvSize);

      // для tvColor и tvSize необходимо создавать контекстное меню
      registerForContextMenu(tvColor);
      registerForContextMenu(tvSize);

  }
```

Теперь опишем создание контекстных меню. Используем константы для хранения ID пунктов меню.

```
final int MENU_COLOR_RED = 1;
final int MENU_COLOR_GREEN = 2;
final int MENU_COLOR_BLUE = 3;

final int MENU_SIZE_22 = 4;
final int MENU_SIZE_26 = 5;
final int MENU_SIZE_30 = 6;
```

И создаем

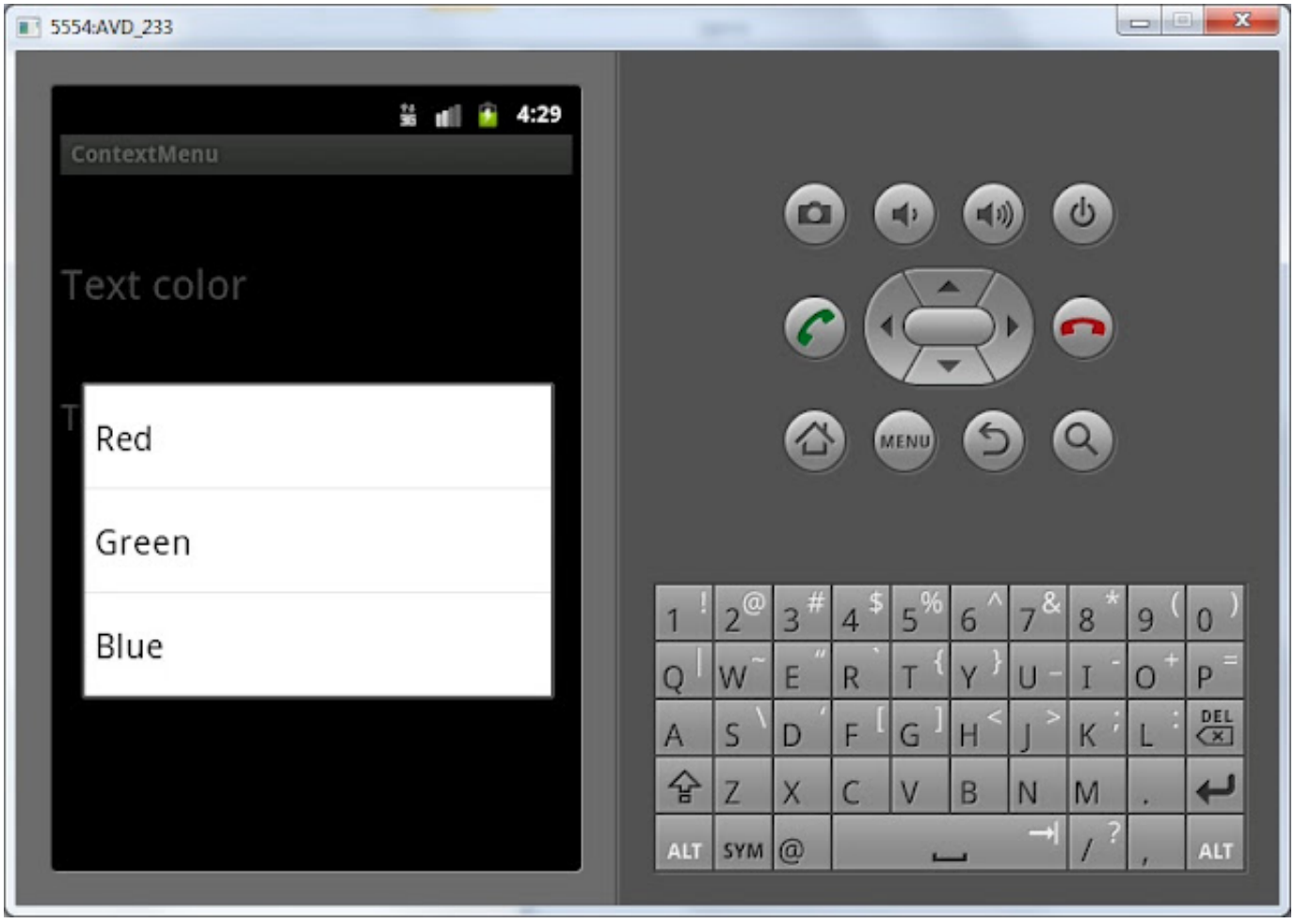
```
  @Override
  public void onCreateContextMenu(ContextMenu menu, View v,
```

```
      ContextMenuInfo menuInfo) {
    // TODO Auto-generated method stub
    switch (v.getId()) {
  case R.id.tvColor:
    menu.add(0, MENU_COLOR_RED, 0, "Red");
    menu.add(0, MENU_COLOR_GREEN, 0, "Green");
    menu.add(0, MENU_COLOR_BLUE, 0, "Blue");
    break;
  case R.id.tvSize:
    menu.add(0, MENU_SIZE_22, 0, "22");
    menu.add(0, MENU_SIZE_26, 0, "26");
    menu.add(0, MENU_SIZE_30, 0, "30");
    break;
  }
  }
```

Обратите внимание, что мы по **ID** определяем **View**, для которого вызвано **контекстное меню** и в зависимости от этого создаем определенное меню. Т.е. если контекстное меню вызвано для **tvColor**, то мы создаем **меню с перечислением цветов**, а если для **tvSize** – **с размерами шрифта**.

В качестве ID пунктов мы использовали константы. Группировку и сортировку не используем, поэтому используем нули в качестве соответствующих параметров.

Можно все сохранить и запустить. При долгом нажатии на TextView должны появляться контекстные меню.



Но нажатие на них ничего не дает, т.к. не мы не прописали обработку в методе onContextItemSelected. Давайте пропишем:

```
    @Override
    public boolean onContextItemSelected(MenuItem item) {
      // TODO Auto-generated method stub
      switch (item.getItemId()) {
      // пункты меню для tvColor
    case MENU_COLOR_RED:
      tvColor.setTextColor(Color.RED);
      tvColor.setText("Text color = red");
      break;
    case MENU_COLOR_GREEN:
      tvColor.setTextColor(Color.GREEN);
      tvColor.setText("Text color = green");
      break;
    case MENU_COLOR_BLUE:
      tvColor.setTextColor(Color.BLUE);
      tvColor.setText("Text color = blue");
      break;
    // пункты меню для tvSize
    case MENU_SIZE_22:
      tvSize.setTextSize(22);
      tvSize.setText("Text size = 22");
      break;
    case MENU_SIZE_26:
      tvSize.setTextSize(26);
      tvSize.setText("Text size = 26");
      break;
    case MENU_SIZE_30:
      tvSize.setTextSize(30);
      tvSize.setText("Text size = 30");
      break;
      }
      return super.onContextItemSelected(item);
    }
```

В этом методе мы определяем по **ID**, какой пункт меню был нажат. И выполняем соответствующие действия: меняем цвет текста для tvColor или размер шрифта для tvSize. Сохраняем, запускаем и проверяем, что контекстные меню теперь реагируют на нажатия и делают то, что от них требуется.

Для расширения кругозора я хотел бы еще кое-что написать по этой теме. Возможно, это покажется пока сложноватым, так что если вдруг будет непонятно, ничего страшного. Итак, мысли вслух.

Мы использовали метод registerForContextMenu (View view) для включения контекстного меню для определенного View. Этот метод принадлежит классу Activity. Я посмотрел исходники этого метода, там написано следующее:

```
  public void registerForContextMenu(View view) {
    view.setOnCreateContextMenuListener(this);
  }
```

Вспоминаем наш урок по обработчикам и смотрим хелп по методу setOnCreateContextMenuListener (View.OnCreateContextMenuListener l). Получается, что **View** в качестве обработчика создания контекстного меню использует объект **this**. В данном случае, этот код в **Activity**, значит this – это Activity и есть. Т.е. когда View хочет **показать контекстное меню**, оно обращается к **обработчику** (Activity), а он уже выполняет свой метод

onCreateContextMenu. Т.е. тот же самый принцип, что и при обычном нажатии (Click).

И строка в MainActivity.java:

```
    registerForContextMenu(tvColor);
```

абсолютно равнозначна этой строке:

```
tvColor.setOnCreateContextMenuListener(this);
```

Вообще мы можем создать свой объект, реализующий интерфейс View.OnCreateContextMenuListener и использовать его вместо Activity в качестве обработчика создания контекстного меню.

Не забывайте, что для контекстного меню вы также можете использовать XML-способ, который был рассмотрен в конце прошлого урока. Попробуйте сделать этот же урок, но уже с использованием XML-меню.

Полный код урока:

```
public class MainActivity extends Activity {

  final int MENU_COLOR_RED = 1;
  final int MENU_COLOR_GREEN = 2;
  final int MENU_COLOR_BLUE = 3;

  final int MENU_SIZE_22 = 4;
  final int MENU_SIZE_26 = 5;
  final int MENU_SIZE_30 = 6;

  TextView tvColor, tvSize;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvColor = (TextView) findViewById(R.id.tvColor);
    tvSize = (TextView) findViewById(R.id.tvSize);

    // для tvColor и tvSize необходимо создавать контекстное меню
    registerForContextMenu(tvColor);
    registerForContextMenu(tvSize);
  }

  @Override
  public void onCreateContextMenu(ContextMenu menu, View v,
      ContextMenuInfo menuInfo) {
    // TODO Auto-generated method stub
    switch (v.getId()) {
    case R.id.tvColor:
      menu.add(0, MENU_COLOR_RED, 0, "Red");
      menu.add(0, MENU_COLOR_GREEN, 0, "Green");
```

```
      menu.add(0, MENU_COLOR_BLUE, 0, "Blue");
      break;
    case R.id.tvSize:
      menu.add(0, MENU_SIZE_22, 0, "22");
      menu.add(0, MENU_SIZE_26, 0, "26");
      menu.add(0, MENU_SIZE_30, 0, "30");
      break;
    }
  }

  @Override
  public boolean onContextItemSelected(MenuItem item) {
    // TODO Auto-generated method stub
    switch (item.getItemId()) {
    // пункты меню для tvColor
    case MENU_COLOR_RED:
      tvColor.setTextColor(Color.RED);
      tvColor.setText("Text color = red");
      break;
    case MENU_COLOR_GREEN:
      tvColor.setTextColor(Color.GREEN);
      tvColor.setText("Text color = green");
      break;
    case MENU_COLOR_BLUE:
      tvColor.setTextColor(Color.BLUE);
      tvColor.setText("Text color = blue");
      break;
    // пункты меню для tvSize
    case MENU_SIZE_22:
      tvSize.setTextSize(22);
      tvSize.setText("Text size = 22");
      break;
    case MENU_SIZE_26:
      tvSize.setTextSize(26);
      tvSize.setText("Text size = 26");
      break;
    case MENU_SIZE_30:
      tvSize.setTextSize(30);
      tvSize.setText("Text size = 30");
      break;
    }
    return super.onContextItemSelected(item);
  }
}
```

На следующем уроке:

- рисуем экран программно, а не через layout-файл

Урок 16. Программное создание экрана. LayoutParams

В этом уроке мы:

- рисуем экран программно, а не через layout-файл

До этого мы создавали экран с помощью **layout-файлов**. Но то же самое мы можем делать и **программно**.

Создадим проект:

Project name: P0161_DynamicLayout
Build Target: Android 2.3.3
Application name: DynamicLayout
Package name: ru.startandroid.develop.dinamiclayout
Create Activity: MainActivity

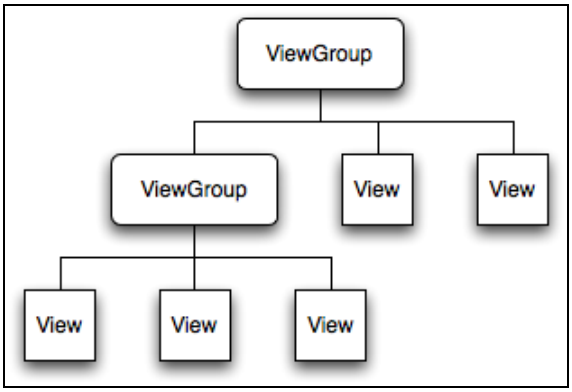
Открываем **MainActivity.java** и обратим внимание на строку:

```
        setContentView(R.layout.main);
```

Напомню, что в этой строке мы указываем, что **Activity** в качестве экрана будет использовать **layout-файл main.xml**. Есть другая реализация этого метода, которая на вход принимает не layout-файл, а **View**-элемент и делает его корневым. В layout-файлах корневой элемент обычно **LinearLayout**, мы тоже используем его.

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // создание LinearLayout
        LinearLayout linLayout = new LinearLayout(this);
        // установим вертикальную ориентацию
        linLayout.setOrientation(LinearLayout.VERTICAL);
        // создаем LayoutParams
        LayoutParams linLayoutParam = new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
        // устанавливаем linLayout как корневой элемент экрана
        setContentView(linLayout, linLayoutParam);
    }
```

Обновим импорт – **CTRL+SHIFT+O**. Eclipse предложит нам выбрать, какой именно **LayoutParams** мы будем использовать. Тут надо остановиться подробнее. Вспомним теорию про экраны. Экран состоит из ViewGroup и вложенных в них View.

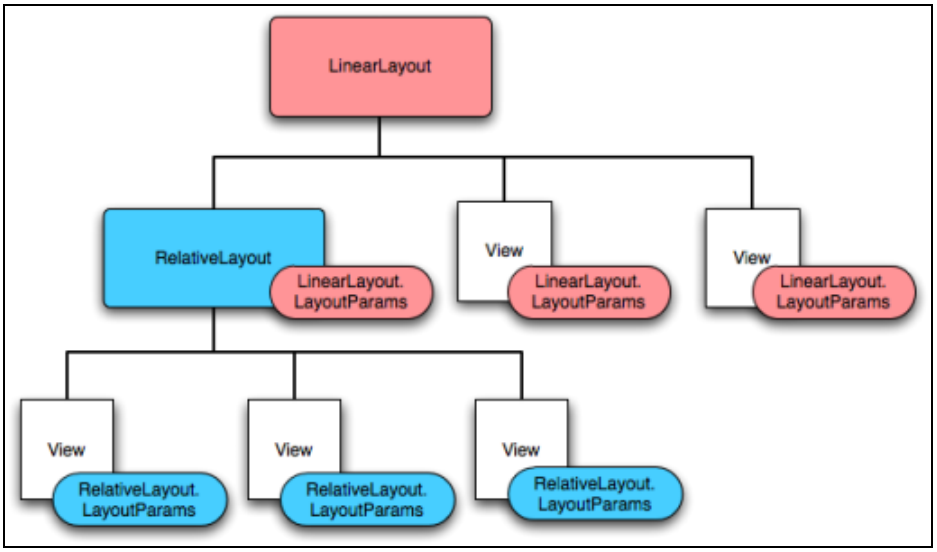


Известные нам примеры **ViewGroup** – это **LinearLayout**, **TableLayout**, **RelativeLayout** и т.д. Каждая их этих ViewGroup имеет вложенный класс LayoutParams. Базовым для этих LayoutParams является ViewGroup.LayoutParams.

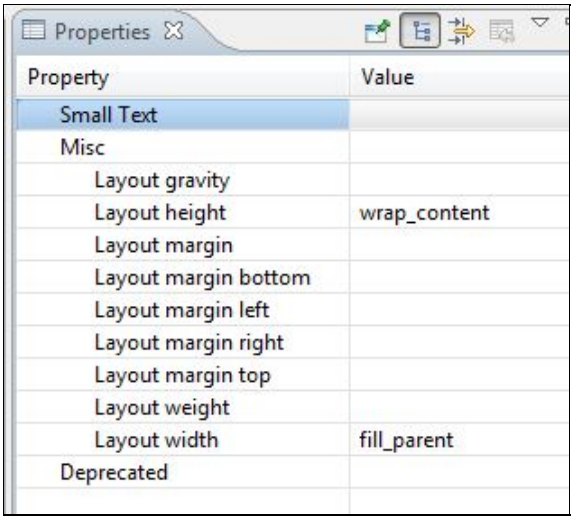
ViewGroup.LayoutParams имеет всего два атрибута: **height** и **width**. Его подкласс ViewGroup.MarginLayoutParams наследует два этих атрибута и имеет свои четыре: **bottomMargin**, **leftMargin**, **rightMargin**, **topMargin**. Класс LinearLayout.LayoutParams в свою очередь является подклассом ViewGroup.MarginLayoutParams, наследует от него уже 6 аттрибутов и добавляет свои два: **gravity** и **weight**.

Т.е. объект **LinearLayout** имеет вложенный класс **LinearLayout.LayoutParams** с layout-аттрибутами. И эти аттрибуты распространяются на

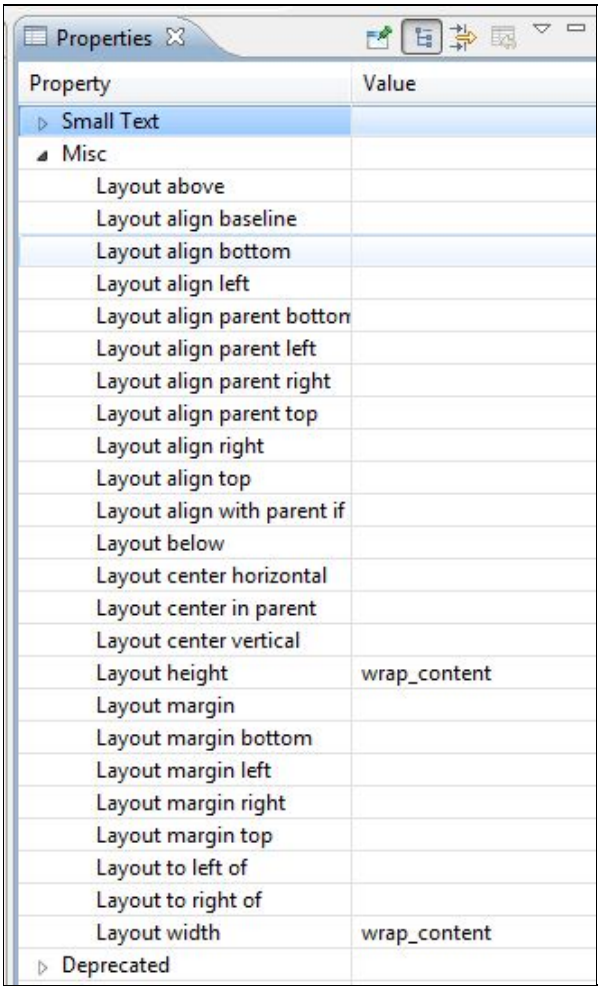
все дочерние View и ViewGroup.



Т.е. View, находящаяся в LinearLayout имеет один набор layout-параметров:

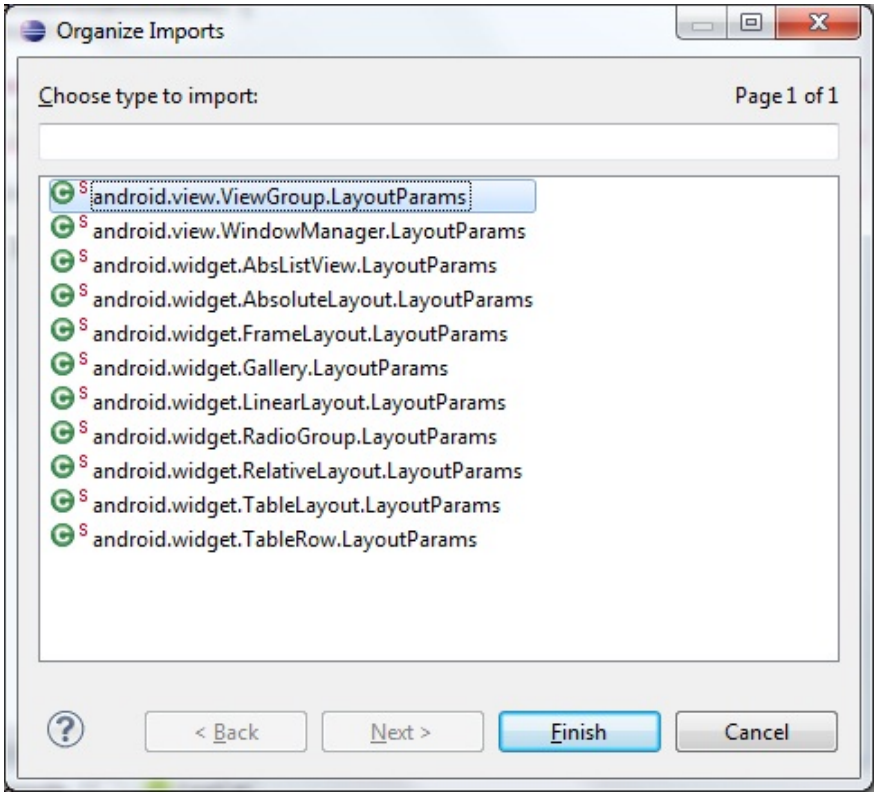


а View из RelativeLayout – другой:



Есть и общие элементы, т.к. родители у этих ViewGroup одни.

Вернемся в Eclipse, он ждет нашего выбора. Используем базовый класс ViewGroup.LayoutParams



Давайте разберем код. Мы создаем **LinearLayout** и ставим **вертикальную** ориентацию. Далее создаем **LayoutParams**. Конструктор на вход принимает два параметра: **width** и **height**. Мы оба ставим **MATCH_PARENT**. Далее вызывается метод setContentView. На вход ему подается

LinearLayout и **LayoutParams**. Это означает, что корневым элементом **Activity** будет **LinearLayout** с layout-свойствами из **LayoutParams**.

Если мы сейчас запустим приложение, то ничего не увидим, т.к. LinearLayout – прозрачен. Давайте добавлять в LinearLayout View-компоненты.

```
        LayoutParams lpView = new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);

        TextView tv = new TextView(this);
        tv.setText("TextView");
        tv.setLayoutParams(lpView);
        linLayout.addView(tv);

        Button btn = new Button(this);
        btn.setText("Button");
        linLayout.addView(btn, lpView);
```

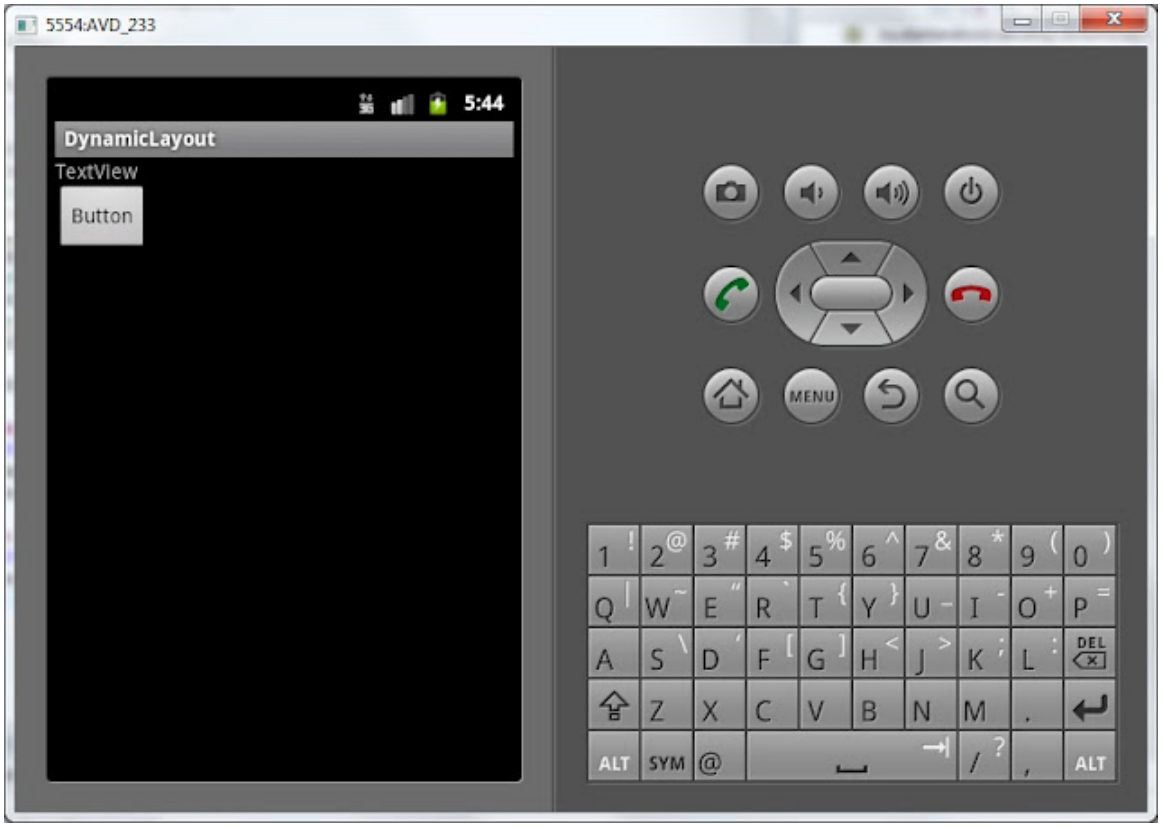
Мы снова создаем объект **LayoutParams** с атрибутами **width** = **wrap_content** и **height** = **wrap_content**. Теперь если мы присвоим этот объект какому-либо View, то это **View** будет иметь **ширину** и **высоту по содержимому**.

Далее мы создаем **TextView**, настраиваем его текст, присваиваем ему выше созданный LayoutParams и добавляем в LinearLayout с помощью метода addView (View child).

С **Button** аналогично – создаем, правим текст, а затем используем другую реализацию метода addView (View child, ViewGroup.LayoutParams params), которая одновременно добавляет **Button** в **LinearLayout** и присваивает для Button указанный **LayoutParams**. Результат будет тот же, что и с TextView, но вместо двух строк кода получилась одна.

Обратите внимание, что для **двух объектов View** я использовал **один объект LayoutParams** - lpView. И если я теперь буду менять свойства этого объекта, меняться будут оба View.

Сохраним и запустим приложение. Видим, что компоненты на экране появились. И видно, что их высота и ширина определена по содержимому (wrap_content).



Объект **lpView** имеет базовый тип **android.view.ViewGroup.LayoutParams**. А значит позволит настроить только ширину и высоту. Но для **View** в **LinearLayout** доступны, например, отступ слева или выравнивание по правому краю. И если мы хотим их задействовать, значит надо использовать **LinearLayout.LayoutParams**:

```
        LinearLayout.LayoutParams leftMarginParams = new LinearLayout.LayoutParams(
```

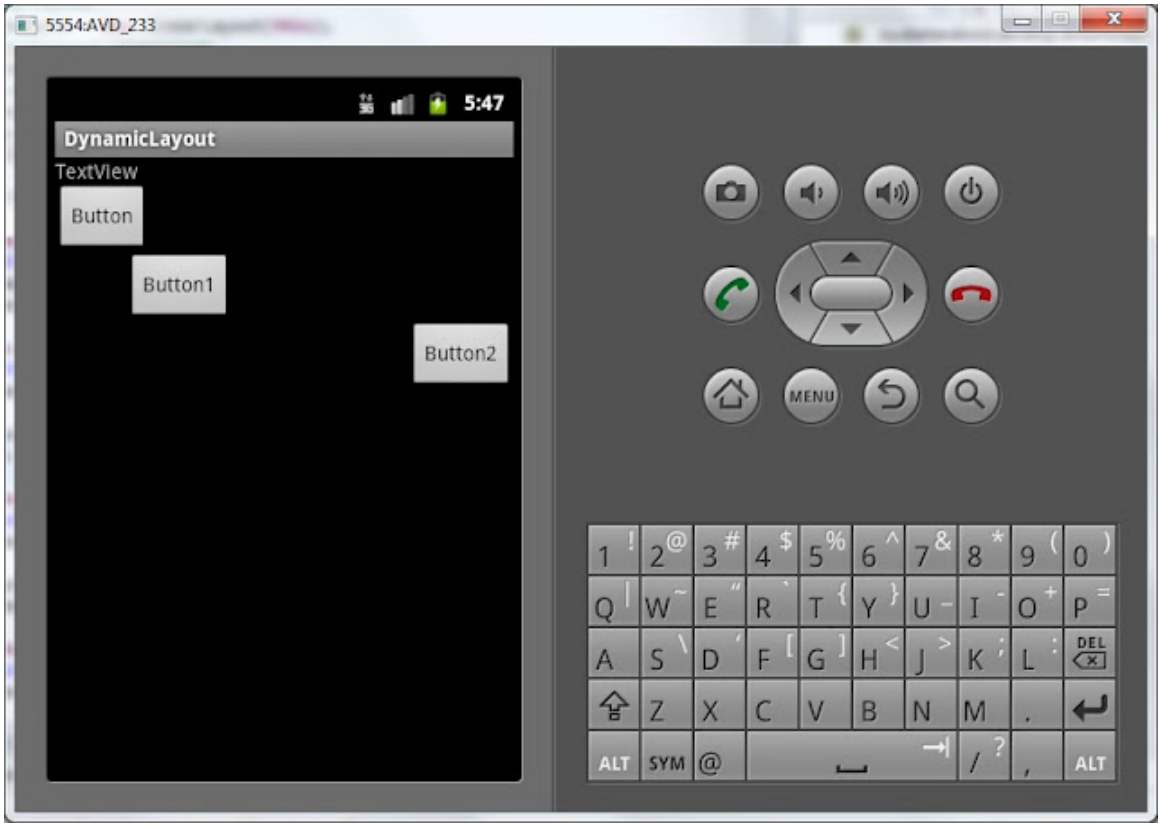
```
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        leftMarginParams.leftMargin = 50;

        Button btn1 = new Button(this);
        btn1.setText("Button1");
        linLayout.addView(btn1, leftMarginParams);
```

Смотрим код. Мы создаем объект типа **LinearLayout.LayoutParams** с помощью такого же конструктора, как и для обычного LayoutParams, указывая **width** и **height**. Затем мы указываем **отступ слева** = 50. Отступ здесь указывается в **пикселах**. Далее схема та же: создаем объект, настраиваем текст и добавляем его в LinearLayout с присвоением LayoutParams.

Аналогично добавим компонент с выравниванием:

```
        LinearLayout.LayoutParams rightGravityParams = new LinearLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        rightGravityParams.gravity = Gravity.RIGHT;

        Button btn2 = new Button(this);
        btn2.setText("Button2");
        linLayout.addView(btn2, rightGravityParams);
```

Сохраним и запустим. Button1 имеет отступ 50px. А Button2 выравнена по правому краю:



Вероятно, эта тема будет не очень понятна с первого раза. Поэтому на следующих двух уроках мы закрепим эти знания и попрактикуемся в добавлении элементов на экран и их настройке.

Полный код урока:

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        // создание LinearLayout
        LinearLayout linLayout = new LinearLayout(this);
        // установим вертикальную ориентацию
        linLayout.setOrientation(LinearLayout.VERTICAL);
        // создаем LayoutParams
        LayoutParams linLayoutParam = new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
        // устанавливаем linLayout как корневой элемент экрана
        setContentView(linLayout, linLayoutParam);

        LayoutParams lpView = new LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);

        TextView tv = new TextView(this);
        tv.setText("TextView");
        tv.setLayoutParams(lpView);
        linLayout.addView(tv);

        Button btn = new Button(this);
        btn.setText("Button");
        linLayout.addView(btn, lpView);


        LinearLayout.LayoutParams leftMarginParams = new LinearLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        leftMarginParams.leftMargin = 50;

        Button btn1 = new Button(this);
        btn1.setText("Button1");
        linLayout.addView(btn1, leftMarginParams);


        LinearLayout.LayoutParams rightGravityParams = new LinearLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT);
        rightGravityParams.gravity = Gravity.RIGHT;

        Button btn2 = new Button(this);
        btn2.setText("Button2");
        linLayout.addView(btn2, rightGravityParams);
    }
}
```

На следующем уроке:

- добавляем компоненты на экран во время работы приложения

Урок 17. Создание View-компонент в рабочем приложении

В этом уроке мы:

- добавляем компоненты на экран прямо из приложения

На прошлом уроке мы создавали компоненты в методе Activity.onCreate, т.е. при создании приложения. На этом уроке будем создавать уже в работающем приложении. Создавать будем Button-ы, т.к. они наглядней всего отображаются. Будем указывать текст, который будет отображен на кнопке и выравнивание: слева, по центру или справа. Также предусмотрим возможность удаления созданных элементов.

Создадим проект:

Project name: P0171_DynamicLayout2
Build Target: Android 2.3.3
Application name: DynamicLayout2
Package name: ru.startandroid.develop.dynamiclayout2
Create Activity: MainActivity

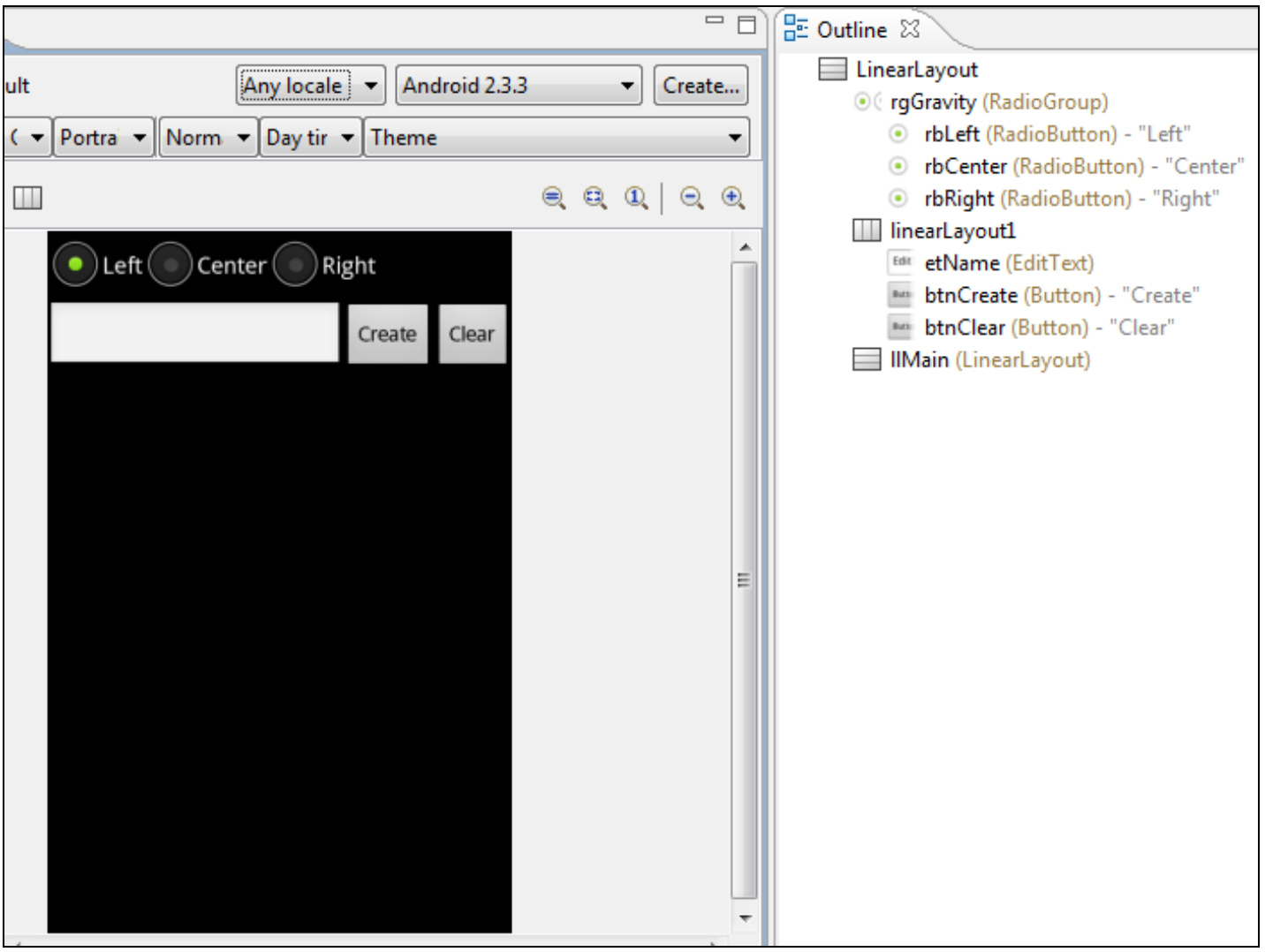
Создадим экран, который поможет нам создавать View-компоненты. Открываем **main.xml** и пишем там следующее:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"
	android:layout_width="fill_parent"
	android:layout_height="fill_parent">
	<RadioGroup
		android:layout_height="wrap_content"
		android:layout_width="match_parent"
		android:orientation="horizontal"
		android:id="@+id/rgGravity">
		<RadioButton
			android:layout_height="wrap_content"
			android:layout_width="wrap_content"
			android:checked="true"
			android:text="Left"
			android:id="@+id/rbLeft">
		</RadioButton>
		<RadioButton
			android:layout_height="wrap_content"
			android:layout_width="wrap_content"
			android:text="Center"
			android:id="@+id/rbCenter">
		</RadioButton>
		<RadioButton
			android:layout_height="wrap_content"
			android:layout_width="wrap_content"
			android:text="Right"
			android:id="@+id/rbRight">
		</RadioButton>
	</RadioGroup>
	<LinearLayout
```

```
            android:id="@+id/linearLayout1"
            android:layout_width="match_parent"
            android:orientation="horizontal"
            android:layout_height="wrap_content">
            <EditText
                android:layout_height="wrap_content"
                android:layout_width="wrap_content"
                android:layout_weight="1"
                android:id="@+id/etName"
                android:fadeScrollbars="true">
            </EditText>
            <Button
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Create"
                android:id="@+id/btnCreate">
            </Button>
            <Button
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Clear"
                android:id="@+id/btnClear">
            </Button>
        </LinearLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/llMain"
            android:orientation="vertical">
        </LinearLayout>
</LinearLayout>
```

Рассмотрим подробно экран.

rgGravity – это RadioGroup, с тремя RadioButton (**rbLeft**, **rbCenter**, **rbRight**). Этот компонент мы используем для выбора выравнивания создаваемого компонента
etName – текстовое поле, здесь будем указывать текст, который будет отображаться на созданном компоненте
btnCreate – кнопка, запускающая процесс создания.
btnClear – кнопка, стирающая все, что создали
llMain – вертикальный LinearLayout, в котором будут создаваться компоненты



Экран готов, давайте кодить реализацию. Открываем **MainActivity.java**. Начнем с того, что опишем и найдем все необходимые нам компоненты. Кстати, у нас есть пара кнопок, которые мы будем использовать, значит им нужен обработчик. В качестве **обработчика** назначим **Activity** (т.е. необходимо дописать: implements OnClickListener) и создадим пустой пока метод обработки **onClick**:

```
public class MainActivity extends Activity implements OnClickListener{

  LinearLayout llMain;
  RadioGroup rgGravity;
  EditText etName;
  Button btnCreate;
  Button btnClear;

int wrapContent = LinearLayout.LayoutParams.WRAP_CONTENT;


  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    llMain = (LinearLayout) findViewById(R.id.llMain);
    rgGravity = (RadioGroup) findViewById(R.id.rgGravity);
    etName = (EditText) findViewById(R.id.etName);

    btnCreate = (Button) findViewById(R.id.btnCreate);
```

```
    btnCreate.setOnClickListener(this);

    btnClear = (Button) findViewById(R.id.btnClear);
    btnClear.setOnClickListener(this);
  }


  @Override
  public void onClick(View v) {
    // TODO Auto-generated method stub

  }
}
```

Я также создал переменную **wrapContent** и буду хранить в ней значение LinearLayout.LayoutParams.WRAP_CONTENT. Делаю это только для снижения громоздкости кода.

Теперь опишем процесс создания Button-компонента заполнив метод onClick:

```
@Override
  public void onClick(View v) {
    switch (v.getId()) {
    case R.id.btnCreate:
      // Создание LayoutParams с шириной и высотой по содержимому
      LinearLayout.LayoutParams lParams = new LinearLayout.LayoutParams(
          wrapContent, wrapContent);
      // переменная для хранения значения выравнивания
      // по умолчанию пусть будет LEFT
      int btnGravity = Gravity.LEFT;
      // определяем, какой RadioButton "чекнут" и
      // соответственно заполняем btnGravity
      switch (rgGravity.getCheckedRadioButtonId()) {
      case R.id.rbLeft:
        btnGravity = Gravity.LEFT;
        break;
      case R.id.rbCenter:
        btnGravity = Gravity.CENTER_HORIZONTAL;
        break;
      case R.id.rbRight:
        btnGravity = Gravity.RIGHT;
        break;
      }
      // переносим полученное значение выравнивания в LayoutParams
      lParams.gravity = btnGravity;

      // создаем Button, пишем текст и добавляем в LinearLayout
      Button btnNew = new Button(this);
      btnNew.setText(etName.getText().toString());
      llMain.addView(btnNew, lParams);

      break;
    }
  }
```

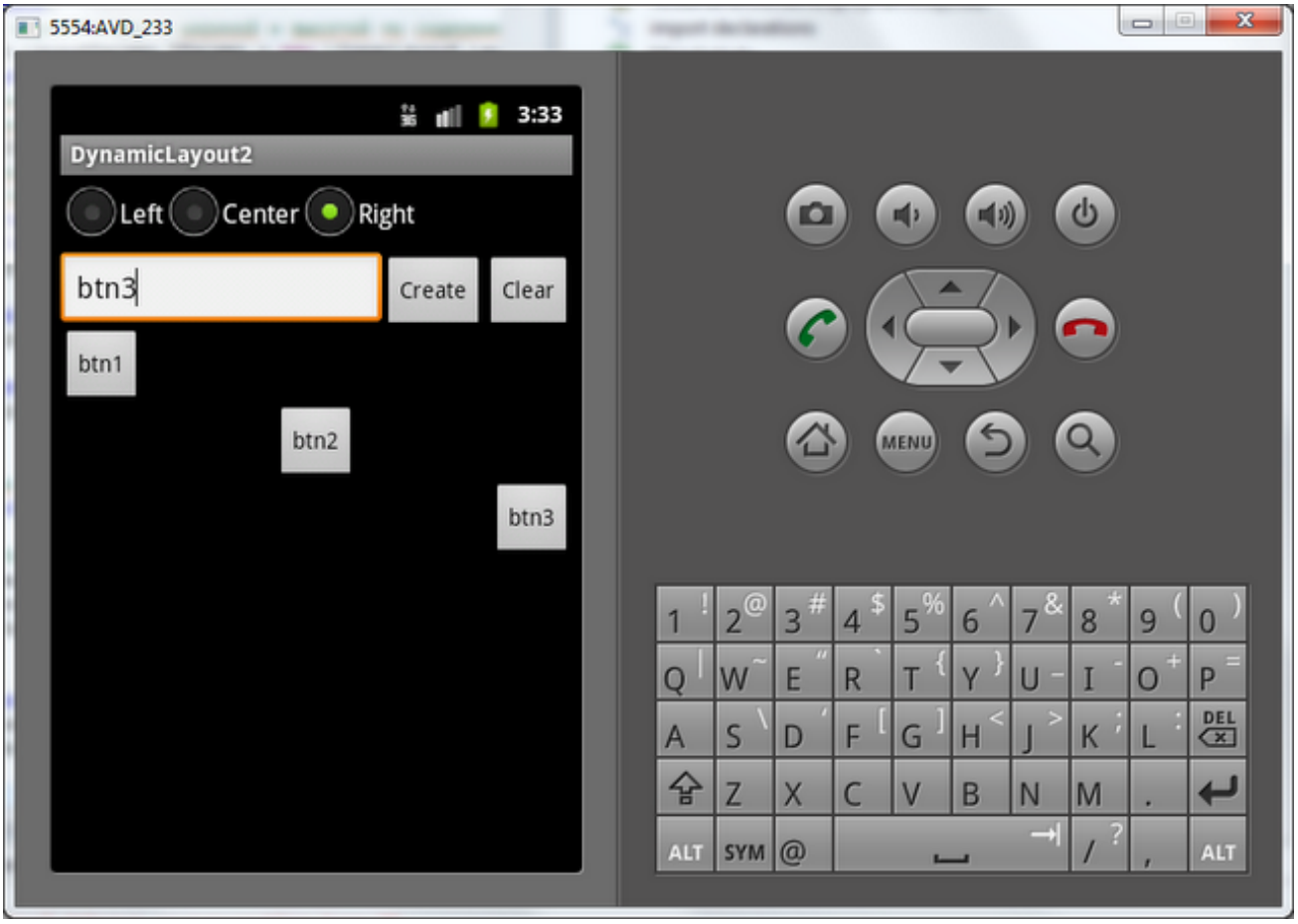
Разберем написанное. Для начала мы проверяем, что была нажата кнопка **btnCreate** – т.е. кнопка создания. Затем создаем **LayoutParams** с высотой и шириной по содержанию. Здесь я использовал переменную, про которую писал выше – wrapContent. Иначе получилось бы довольно громоздко.

Далее создаем переменную **btnGravity**, в которую по умолчанию запишем значение выравнивания LEFT. Для определения,

какой RadioButton выделен в данный момент, используем метод **getCheckedRadioButtonId** – он для RadoiGroup возвращает ID «чекнутого» RadioButton-а. Мы его сравниваем с нашими тремя ID и заносим соответствующее значение в переменную btnGravity. Скидываем это значение в gravity у LayoutParams.

Далее создаем кнопку и присваиваем ей текст из **etName**. Обратите внимание, что недостаточно написать getText, т.к. это не даст текста. Необходимо еще вызвать метод toString. Ну и в конце добавляем созданный Button в наш LinearLayout.

Сохраним все и запустим приложение. Добавим несколько кнопок.



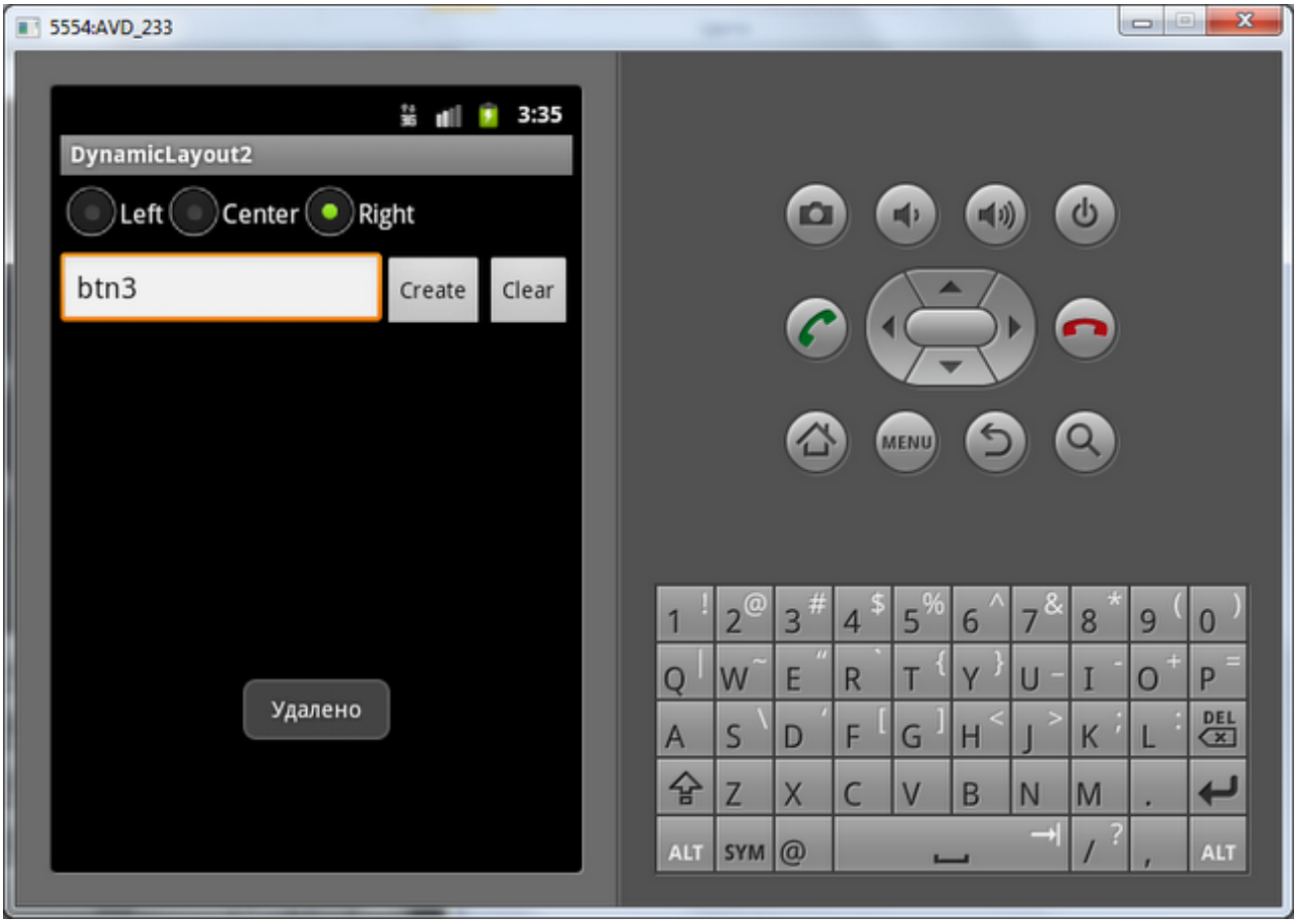
Кнопки должны появляться с указанным выравниванием и текстом.

Когда вводите текст, снизу появляется клавиатура и закрывает обзор. Чтобы она исчезла, надо нажать кнопку Back (Назад) на эмуляторе или ESC на обычной клавиатуре. Если клавиатура появляется японская с иероглифами, вызовите контекстное меню для поля ввода (долгое нажатие левой кнопкой мыши), нажмите Input method и выберите из списка Android Keyboard.

Осталось нереализованной кнопка **Clear**, которая призвана удалять все созданное. Для этого нам необходимо дополнить метод **onClick**, добавим в **switch** (v.getId()) еще один case:

```
    case R.id.btnClear:
      llMain.removeAllViews();
      Toast.makeText(this, "Удалено", Toast.LENGTH_SHORT).show();
      break;
```

Метод **removeAllViews** удаляет все дочерние View-компоненты с нашего LinearLayout. С помощью **Toast** выводим на экран **сообщение** об успехе. Сохраним, запустим и проверим. Добавляем несколько кнопок, жмем кнопку **Clear** и наблюдаем результат:



В итоге у нас получилось очень даже динамическое приложение, которое умеет менять само себя.

Полный код урока:

```
public class MainActivity extends Activity implements OnClickListener {

  LinearLayout llMain;
  RadioGroup rgGravity;
  EditText etName;
  Button btnCreate;
  Button btnClear;

  int wrapContent = LinearLayout.LayoutParams.WRAP_CONTENT;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    llMain = (LinearLayout) findViewById(R.id.llMain);
    rgGravity = (RadioGroup) findViewById(R.id.rgGravity);
    etName = (EditText) findViewById(R.id.etName);

    btnCreate = (Button) findViewById(R.id.btnCreate);
    btnCreate.setOnClickListener(this);

    btnClear = (Button) findViewById(R.id.btnClear);
    btnClear.setOnClickListener(this);
  }
```

```
  @Override
  public void onClick(View v) {
    switch (v.getId()) {
    case R.id.btnCreate:
      // Создание LayoutParams с шириной и высотой по содержимому
      LinearLayout.LayoutParams lParams = new LinearLayout.LayoutParams(
          wrapContent, wrapContent);
      // переменная для хранения значения выравнивания
      // по умолчанию пусть будет LEFT
      int btnGravity = Gravity.LEFT;
      // определяем, какой RadioButton "чекнут" и
      // соответственно заполняем btnGravity
      switch (rgGravity.getCheckedRadioButtonId()) {
      case R.id.rbLeft:
        btnGravity = Gravity.LEFT;
        break;
      case R.id.rbCenter:
        btnGravity = Gravity.CENTER_HORIZONTAL;
        break;
      case R.id.rbRight:
        btnGravity = Gravity.RIGHT;
        break;
      }
      // переносим полученное значение выравнивания в LayoutParams
      lParams.gravity = btnGravity;

      // создаем Button, пишем текст и добавляем в LinearLayout
      Button btnNew = new Button(this);
      btnNew.setText(etName.getText().toString());
      llMain.addView(btnNew, lParams);

      break;

    case R.id.btnClear:
      llMain.removeAllViews();
      Toast.makeText(this, "Удалено", Toast.LENGTH_SHORT).show();
      break;

    }
  }

}
```

На следующем уроке:

- изменяем layout-параметры для уже существующих компонентов экрана

Урок 18. Меняем layoutParams в рабочем приложении

В этом уроке мы:

- изменяем layout-параметры для уже существующих компонентов экрана

Мы умеем создавать экранные компоненты и настраивать для них расположение с помощью **LayoutParams**. В этом уроке разберемся, как изменять layout-параметры уже существующих компонентов.

Менять мы будем вес – **weight**. Нарисуем **SeekBar** (регулятор или «ползунок») и **две кнопки**. И будем **регулировать** пространство занимаемое кнопками, используя параметр веса.

Создадим проект:

Project name: P0181_DynamicLayout3
Build Target: Android 2.3.3
Application name: DynamicLayout3
Package name: ru.startandroid.develop.dynamiclayout3
Create Activity: MainActivity

Открываем **main.xml** и создаем такой экран:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
     <SeekBar
           android:layout_height="wrap_content"
           android:layout_width="match_parent"
           android:max="100"
           android:progress="50"
           android:layout_marginTop="20dp"
           android:id="@+id/sbWeight">
     </SeekBar>
     <LinearLayout
           android:id="@+id/linearLayout1"
           android:layout_width="match_parent"
           android:orientation="horizontal"
           android:layout_height="wrap_content"
           android:layout_marginTop="30dp">
          <Button
                android:layout_height="wrap_content"
                android:id="@+id/btn1"
                android:text="Button1"
                android:layout_weight="1"
                android:layout_width="wrap_content">
          </Button>
          <Button
```

```
            android:layout_height="wrap_content"
            android:id="@+id/btn2"
            android:text="Button2"
            android:layout_weight="1"
            android:layout_width="wrap_content">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Мы используем компонент SeekBar. Он похож на полосу прокрутки и позволяет задавать какое-либо значение из диапазона. У этого компонента есть свойства **max** и **progress**. **Max** – это какое значение выдает SeekBar, когда он выкручен на **максимум**. **Progress** – это **текущее значение** ползунка. Максимум сделаем = **100**, а текущее значение будет на половине – **50**.

Кнопки у нас с шириной по содержимому и **вес для обоих = 1**. Они **поровну** делят пространство LinearLayout, в котором находятся.

Осталось только написать нужный код, чтобы все заработало. Открываем **MainActivity.java**, опишем и найдем компоненты и получим доступ к их LayoutParams.

```
public class MainActivity extends Activity {

  SeekBar sbWeight;
  Button btn1;
  Button btn2;

  LinearLayout.LayoutParams lParams1;
  LinearLayout.LayoutParams lParams2;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    sbWeight = (SeekBar) findViewById(R.id.sbWeight);

    btn1 = (Button) findViewById(R.id.btn1);
    btn2 = (Button) findViewById(R.id.btn2);

    lParams1 = (LinearLayout.LayoutParams) btn1.getLayoutParams();
    lParams2 = (LinearLayout.LayoutParams) btn2.getLayoutParams();
  }
}
```

Мы используем метод getLayoutParams для получения LayoutParams компонента. Но этот метод возвращает базовый **ViewGroup.LayoutParams**, а нам нужен **LinearLayout.LayoutParams**, поэтому делаем преобразование. В результате - **lParams1** и **lParams2** теперь являются **LayoutParams** для компонентов **btn1** и **btn2**. Т.е. работая, например, с lParams1 мы влияем на btn1. Сейчас мы это используем.

Для SeekBar нужен будет обработчик, который будет реагировать на изменения. Это мы поручим Activity. Для этого надо добавить к описанию класса **implements OnSeekBarChangeListener**:

```
public class MainActivity extends Activity implements OnSeekBarChangeListener {
```

А также надо добавить методы обработчика, которые теперь обязана реализовывать Activity.

```
@Override
public void onProgressChanged(SeekBar seekBar, int progress,
    boolean fromUser) {
  // TODO Auto-generated method stub

}

@Override
public void onStartTrackingTouch(SeekBar seekBar) {
  // TODO Auto-generated method stub

}

@Override
public void onStopTrackingTouch(SeekBar seekBar) {
  // TODO Auto-generated method stub

}
```

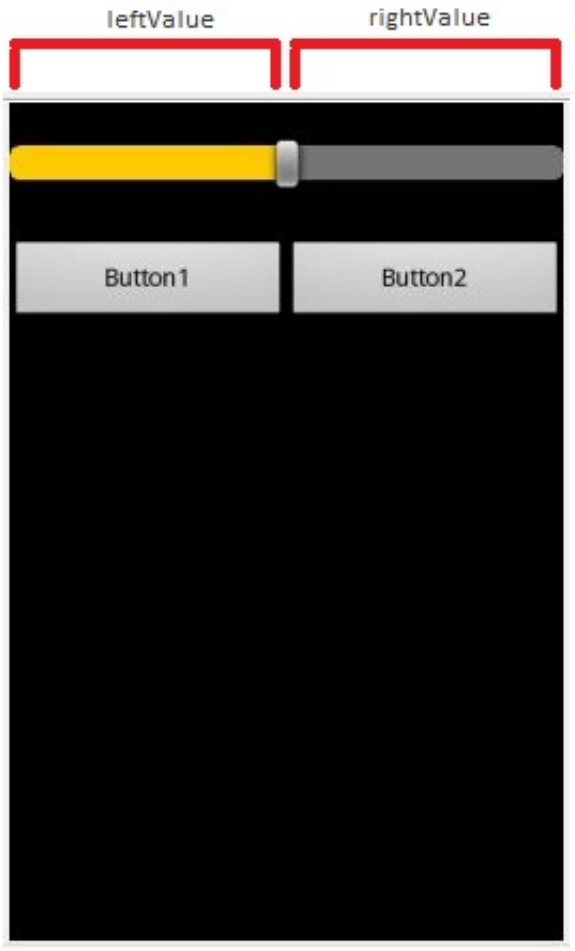
Обработчик содержит три метода. Из названий понятно, что:
- onStartTrackingTouch срабатывает, когда начинаем тащить ползунок
- onProgressChanged срабатывает все время, пока значение меняется
- onStopTrackingTouch срабатывает, когда отпускаем ползунок

Мы будем использовать метод onProgressChanged. Так изменения будут видны во время перетаскивания ползунка.

```
@Override
public void onProgressChanged(SeekBar seekBar, int progress,
    boolean fromUser) {
  int leftValue = progress;
  int rightValue = seekBar.getMax() - progress;
  // настраиваем вес
  lParams1.weight = leftValue;
  lParams2.weight = rightValue;
  // в текст кнопок пишем значения переменных
  btn1.setText(String.valueOf(leftValue));
  btn2.setText(String.valueOf(rightValue));
}
```

переменная **leftValue** – текущее значение SeekBar, т.е. то что слева от ползунка
переменная **rightValue** – то, что справа от ползунка, т.е. из максимума вычесть текущее значение.

Соответственно эти значения и используем как вес. Чем ползунок **левее**, тем **меньше leftValue** и **больше rightValue**, а значит **меньше ширина btn1** и **больше ширина btn2**. И наоборот.



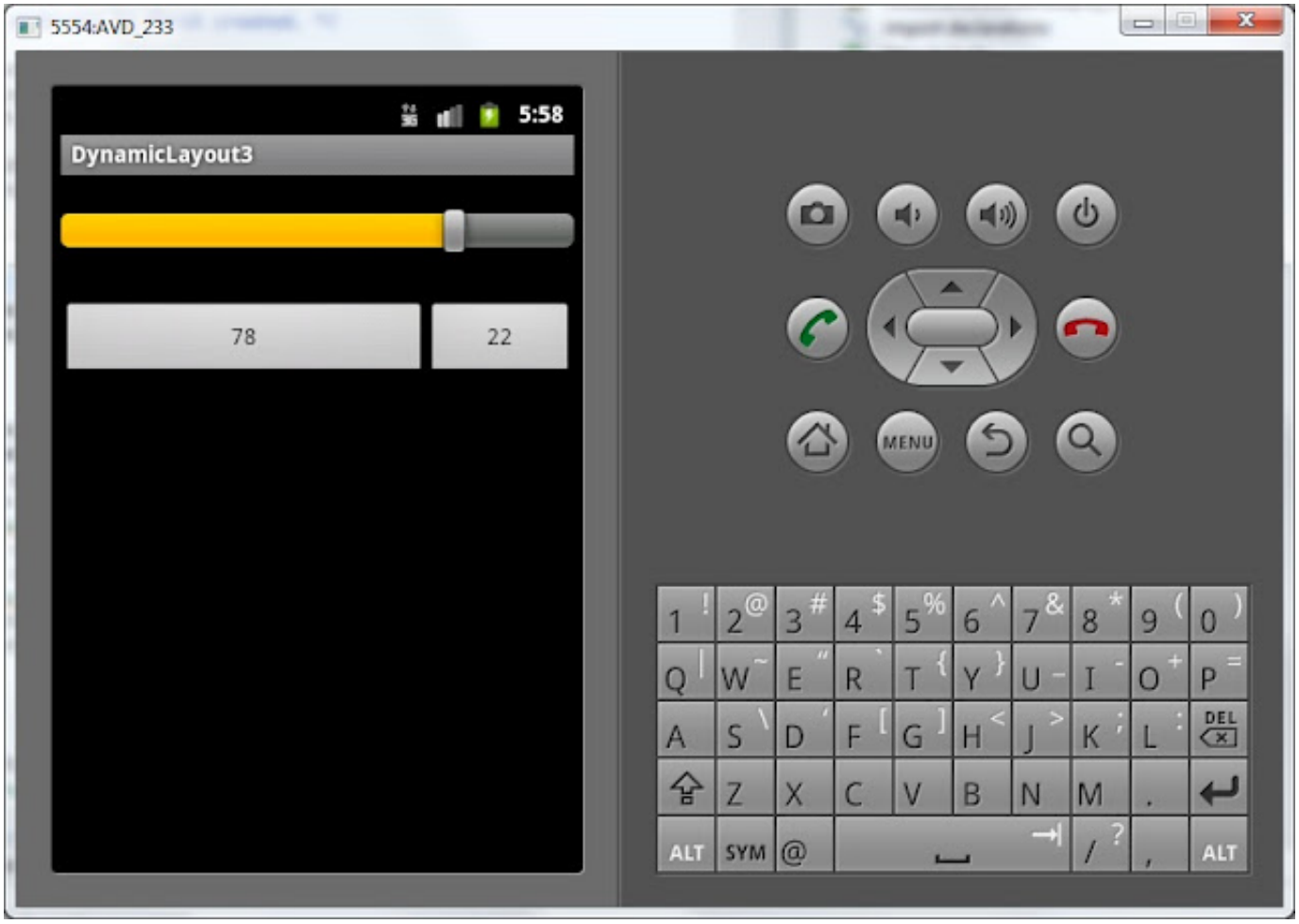
Также для наглядности в текст кнопок будем записывать значения переменных.

Ну и конечно не забываем, что надо обработчик (Activity) присвоить View-компоненту, события которого необходимо обрабатывать:

```
setContentView(R.layout.main);

sbWeight = (SeekBar) findViewById(R.id.sbWeight);
sbWeight.setOnSeekBarChangeListener(this);

btn1 = (Button) findViewById(R.id.btn1);
```

(Обратите внимание. Я ввожу новый прием подачи кода. То, что подчеркнуто – это новый код, а обычный шрифт – уже существующий код. Вам надо найти существующий код и дописать к нему новый, чтобы получился этот фрагмент.)

Все сохраним и запустим приложение. Перетаскивая ползунок, меняем размеры кнопок:



Выглядит эффектно, я считаю) И кода - всего несколько строк.

Есть небольшой нюанс. Как верно заметили в каментах, если просто написать код *lParams1.weight = 1,* то компонент не изменится. Необходимо дописать код: *btn1.requestLayout()*. Тогда кнопка прочтет Layout и перерисуется. Этот метод уже вызывается в setText, поэтому мы его здесь явно не вызываем.

Теперь мы знаем достаточно много, и на следующих уроках попробуем написать первое осмысленное приложение – калькулятор.

Полный код урока:

```
public class MainActivity extends Activity implements OnSeekBarChangeListener {

  SeekBar sbWeight;
  Button btn1;
  Button btn2;

  LinearLayout.LayoutParams lParams1;
  LinearLayout.LayoutParams lParams2;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.main);

    sbWeight = (SeekBar) findViewById(R.id.sbWeight);
    sbWeight.setOnSeekBarChangeListener(this);

    btn1 = (Button) findViewById(R.id.btn1);
    btn2 = (Button) findViewById(R.id.btn2);

    lParams1 = (LinearLayout.LayoutParams) btn1.getLayoutParams();
    lParams2 = (LinearLayout.LayoutParams) btn2.getLayoutParams();
  }

  @Override
  public void onProgressChanged(SeekBar seekBar, int progress,
      boolean fromUser) {
    int leftValue = progress;
    int rightValue = seekBar.getMax() - progress;
    // настраиваем вес
    lParams1.weight = leftValue;
    lParams2.weight = rightValue;
    // в текст кнопок пишем значения переменных
    btn1.setText(String.valueOf(leftValue));
    btn2.setText(String.valueOf(rightValue));
  }

  @Override
  public void onStartTrackingTouch(SeekBar seekBar) {
    // TODO Auto-generated method stub
  }

  @Override
  public void onStopTrackingTouch(SeekBar seekBar) {
    // TODO Auto-generated method stub
  }
}
```

На следующем уроке:

- пишем приложение калькулятор

Урок 19. Пишем простой калькулятор

В этом уроке мы:

- пишем приложение - калькулятор

Попробуем написать простейший калькулятор, который берет два числа и проводит с ними операции сложения, вычитания, умножения или деления. Результат отображает в виде полного выражения.

Создадим проект:

Project name: P0191_SimpleCalculator
Build Target: Android 2.3.3
Application name: SimpleCalculator
Package name: ru.startandroid.develop.simplecalculator
Create Activity: MainActivity

Откроем **main.xml** и нарисуем экран:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:orientation="vertical"
	android:layout_width="fill_parent"
	android:layout_height="fill_parent">
	<LinearLayout
		android:layout_width="match_parent"
		android:layout_height="wrap_content"
		android:id="@+id/linearLayout1"
		android:layout_marginLeft="10pt"
		android:layout_marginRight="10pt"
		android:layout_marginTop="3pt">
		<EditText
			android:layout_weight="1"
			android:layout_height="wrap_content"
			android:layout_marginRight="5pt"
			android:id="@+id/etNum1"
			android:layout_width="match_parent"
			android:inputType="numberDecimal">
		</EditText>
		<EditText
			android:layout_height="wrap_content"
			android:layout_weight="1"
			android:layout_marginLeft="5pt"
			android:id="@+id/etNum2"
			android:layout_width="match_parent"
			android:inputType="numberDecimal">
		</EditText>
	</LinearLayout>
	<LinearLayout
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/linearLayout2"
            android:layout_marginTop="3pt"
            android:layout_marginLeft="5pt"
            android:layout_marginRight="5pt">
            <Button
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:text="+"
                android:textSize="8pt"
                android:id="@+id/btnAdd">
            </Button>
            <Button
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:text="-"
                android:textSize="8pt"
                android:id="@+id/btnSub">
            </Button>
            <Button
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:text="*"
                android:textSize="8pt"
                android:id="@+id/btnMult">
            </Button>
            <Button
                android:layout_height="wrap_content"
                android:layout_width="match_parent"
                android:layout_weight="1"
                android:text="/"
                android:textSize="8pt"
                android:id="@+id/btnDiv">
            </Button>
        </LinearLayout>
        <TextView
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:layout_marginLeft="5pt"
            android:layout_marginRight="5pt"
            android:textSize="12pt"
            android:layout_marginTop="3pt"
            android:id="@+id/tvResult"
            android:gravity="center_horizontal">
        </TextView>
</LinearLayout>
```

Тут есть **два поля ввода**, **4 кнопки** и **текстовое поле для вывода**. Обратите внимание на атрибут **inputType** для **EditText**. Он задает **тип содержимого**. Я указал **numberDecimal** – т.е. в поле получится ввести **только цифры и**

запятую, буквы он не пропустит. Это удобно, не надо самому кодить различные проверки.

Для **TextView** указан атрибут **gravity**. Он указывает, как будет **расположен текст** в TextView. Не путайте с layout_gravity, который отвечает за размещение TextView в ViewGroup.

Теперь нам надо читать содержимое полей, определять какую кнопку нажали и выводить нужный результат. Открываем **MainActivity.java** и пишем код

```
public class MainActivity extends Activity implements OnClickListener {

  EditText etNum1;
  EditText etNum2;

  Button btnAdd;
  Button btnSub;
  Button btnMult;
  Button btnDiv;

  TextView tvResult;

  String oper = "";

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // находим элементы
    etNum1 = (EditText) findViewById(R.id.etNum1);
    etNum2 = (EditText) findViewById(R.id.etNum2);

    btnAdd = (Button) findViewById(R.id.btnAdd);
    btnSub = (Button) findViewById(R.id.btnSub);
    btnMult = (Button) findViewById(R.id.btnMult);
    btnDiv = (Button) findViewById(R.id.btnDiv);

    tvResult = (TextView) findViewById(R.id.tvResult);

    // прописываем обработчик
    btnAdd.setOnClickListener(this);
    btnSub.setOnClickListener(this);
    btnMult.setOnClickListener(this);
    btnDiv.setOnClickListener(this);

  }

  @Override
  public void onClick(View v) {
    // TODO Auto-generated method stub
    float num1 = 0;
    float num2 = 0;
    float result = 0;

    // Проверяем поля на пустоту
    if (TextUtils.isEmpty(etNum1.getText().toString())
        || TextUtils.isEmpty(etNum2.getText().toString())) {
      return;
    }

    // читаем EditText и заполняем переменные числами
```

```
    num1 = Float.parseFloat(etNum1.getText().toString());
    num2 = Float.parseFloat(etNum2.getText().toString());

    // определяем нажатую кнопку и выполняем соответствующую операцию
    // в oper пишем операцию, потом будем использовать в выводе
    switch (v.getId()) {
    case R.id.btnAdd:
      oper = "+";
      result = num1 + num2;
      break;
    case R.id.btnSub:
      oper = "-";
      result = num1 - num2;
      break;
    case R.id.btnMult:
      oper = "*";
      result = num1 * num2;
      break;
    case R.id.btnDiv:
      oper = "/";
      result = num1 / num2;
      break;
    default:
      break;
    }

    // формируем строку вывода
    tvResult.setText(num1 + " " + oper + " " + num2 + " = " + result);
  }
}
```

Думаю, все понятно по каментам. Читаем значения, определяем кнопку, выполняем операцию и выводим в текстовое поле. Обработчиком нажатий на кнопки выступает **Activity**.

Все сохраним и запустим.

Давайте для большего функционала сделаем **меню** с пунктами **очистки полей** и **выхода из приложения**. Пункты будут называться **Reset** и **Quit**.

Добавим **две константы** – это будут **ID** пунктов меню.

```
public class MainActivity extends Activity implements OnClickListener {

final int MENU_RESET_ID = 1;
final int MENU_QUIT_ID = 2;

EditText etNum1;
```

(добавляете только подчеркнутый код)

И напишем код **создания** и **обработки** меню:

```
// создание меню
@Override
public boolean onCreateOptionsMenu(Menu menu) {
// TODO Auto-generated method stub
menu.add(0, MENU_RESET_ID, 0, "Reset");
menu.add(0, MENU_QUIT_ID, 0, "Quit");
return super.onCreateOptionsMenu(menu);
}

// обработка нажатий на пункты меню
@Override
public boolean onOptionsItemSelected(MenuItem item) {
// TODO Auto-generated method stub
```

```
switch (item.getItemId()) {
case MENU_RESET_ID:
// очищаем поля
etNum1.setText("");
etNum2.setText("");
tvResult.setText("");
break;
case MENU_QUIT_ID:
// выход из приложения
finish();
break;
}
return super.onOptionsItemSelected(item);
}
```

Сохраним все, запустим. Появилось два пункта меню:
Reset – очищает все поля
Quit – закрывает приложение

В качестве самостоятельной работы вы можете реализовать **проверку деления на ноль**. И выводить какое-нить сообщение с помощью Toast или прямо в поле результата.

На следующем уроке:

- рассмотрим анимацию View-компонентов

Урок 20. Анимация

В этом уроке мы:

- рассмотрим анимацию View-компонентов

Перед серьезными темами я решил все таки рассмотреть еще одну интересную и, на мой взгляд, несложную тему. Правда рассмотрю я только вершину и в дебри не полезу. Тема – анимация. Мы научимся проделывать следующие трансформации с обычными View-компонентами:
- менять прозрачность
- менять размер
- перемещать
- поворачивать

Создадим проект:

Project name: P0201_SimpleAnimation
Build Target: Android 2.3.3
Application name: SimpleAnimation
Package name: ru.startandroid.develop.p0201simpleanimation
Create Activity: MainActivity

Трансформации конфигурируются в XML файлах, затем в коде программы считываются и присваиваются View-элементам. Я не буду дублировать хелп и все расписывать, а сразу перейду к практике.

В нашем проекте есть папка **res**. Надо в ней создать папку **anim**. Сделать это можно, например, так: правой кнопкой на **res** и в меню выбираем **New** -> **Folder**. В папке **anim** надо создать файлы. Делается это аналогично: правой кнопкой на **anim** и в меню выбираем **New** -> **File**. В этих файлах будем конфигурировать анимацию.

Создаем следующие файлы в папке res/anim:

Имя файла: myalpha.xml

Содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<alpha
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:fromAlpha="0.0"
     android:toAlpha="1.0"
     android:duration="3000">
</alpha>
```

Описание трансформации: меняется прозрачность с 0 до 1 в течение трех секунд.

Имя файла: myscale.xml

Содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<scale
```

```
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:fromXScale="0.1"
      android:toXScale="1.0"
      android:fromYScale="0.1"
      android:toYScale="1.0"
      android:pivotX="50%"
      android:pivotY="50%"
      android:duration="3000">
</scale>
```

Описание трансформации: изменение размера с 0.1 от оригинальной ширины и высоты до 1. Точка, относительно которой будет производиться масштабирование, лежит ровно посередине объекта (pivotX, pivotY). Продолжительность – 3 сек.

Имя файла: mytrans.xml

Содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<translate
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:fromXDelta="-150"
      android:toXDelta="0"
      android:fromYDelta="-200"
      android:toYDelta="0"
      android:duration="3000">
</translate>
```

Описание трансформации: перемещение с -150 относительно текущей позиции по оси X и -200 по оси Y в текущую позицию (0,0). Продолжительность – 3 сек.

Имя файла: myrotate.xml

Содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<rotate
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:fromDegrees="0"
      android:toDegrees="360"
      android:duration="3000">
</rotate>
```

Описание трансформации: поворот относительно левого верхнего угла (т.к. не указаны pivotX, pivotY) на 360 градусов в течение трех секунд

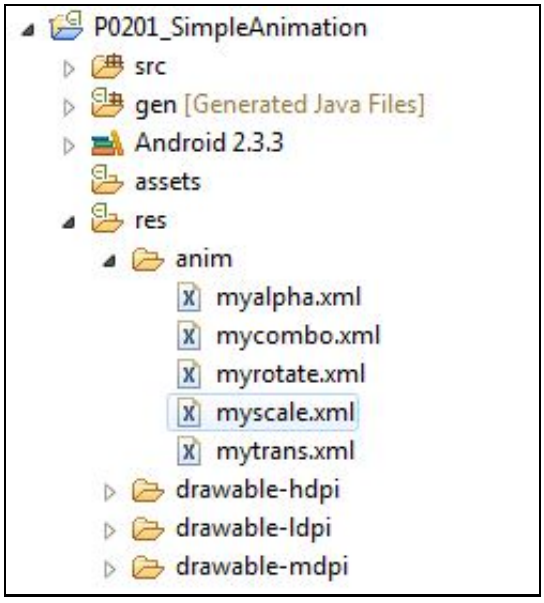
Имя файла: mycombo.xml

Содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<set
     xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
         android:fromDegrees="0"
         android:toDegrees="360"
         android:duration="3000"
         android:pivotX="50%"
         android:pivotY="50%">
    </rotate>
    <scale
         android:fromXScale="0.1"
         android:toXScale="1.0"
         android:fromYScale="0.1"
         android:toYScale="1.0"
         android:pivotX="50%"
         android:pivotY="50%"
         android:duration="3000">
    </scale>
</set>
```

Описание трансформации: одновременно увеличение размера и вращение в течение трех секунд. Обратите внимание, для комбинации трансформ использован тег <set>

Итак, мы создали 5 файлов анимации.



И теперь можем применять их к View-компонентам.

Открываем **main.xml** и создадим экран:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:id="@+id/frameLayout1"
```

```
     android:layout_height="match_parent">
    <TextView
         android:text="TextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center_vertical|center_horizontal"
         android:id="@+id/tv"
         android:textSize="38sp">
    </TextView>
</FrameLayout>
```

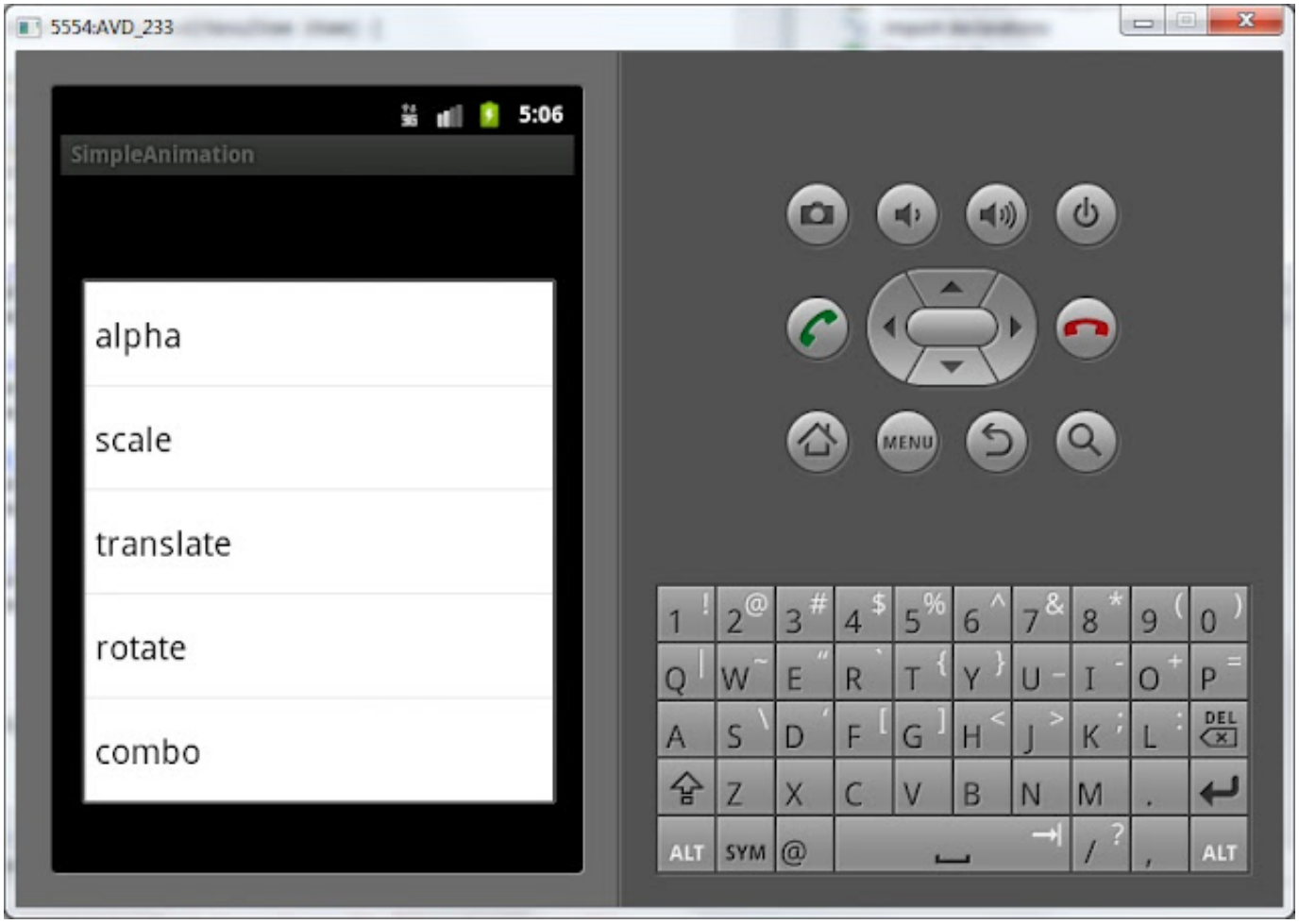
По центру экрана находится **TextView**, к нему и будем применять трансформации. Для этого создадим контекстное меню для TextView, добавим пункты меню, соответствующие нашим наборам и при нажатии будем запускать анимацию.

```
package ru.startandroid.develop.p0201simpleanimation;

import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.MenuItem;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.TextView;

public class MainActivity extends Activity {

  // константы для ID пунктов меню
  final int MENU_ALPHA_ID = 1;
  final int MENU_SCALE_ID = 2;
  final int MENU_TRANSLATE_ID = 3;
  final int MENU_ROTATE_ID = 4;
  final int MENU_COMBO_ID = 5;

  TextView tv;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tv = (TextView) findViewById(R.id.tv);
    // регистрируем контекстное меню для компонента tv
    registerForContextMenu(tv);
  }

  @Override
  public void onCreateContextMenu(ContextMenu menu, View v,
      ContextMenuInfo menuInfo) {
    // TODO Auto-generated method stub
    switch (v.getId()) {
```

```
    case R.id.tv:
      // добавляем пункты
      menu.add(0, MENU_ALPHA_ID, 0, "alpha");
      menu.add(0, MENU_SCALE_ID, 0, "scale");
      menu.add(0, MENU_TRANSLATE_ID, 0, "translate");
      menu.add(0, MENU_ROTATE_ID, 0, "rotate");
      menu.add(0, MENU_COMBO_ID, 0, "combo");
      break;
    }
    super.onCreateContextMenu(menu, v, menuInfo);
  }

  @Override
  public boolean onContextItemSelected(MenuItem item) {
    Animation anim = null;
    // определяем какой пункт был нажат
    switch (item.getItemId()) {
    case MENU_ALPHA_ID:
      // создаем объект анимации из файла anim/myalpha
      anim = AnimationUtils.loadAnimation(this, R.anim.myalpha);
      break;
    case MENU_SCALE_ID:
      anim = AnimationUtils.loadAnimation(this, R.anim.myscale);
      break;
    case MENU_TRANSLATE_ID:
      anim = AnimationUtils.loadAnimation(this, R.anim.mytrans);
      break;
    case MENU_ROTATE_ID:
      anim = AnimationUtils.loadAnimation(this, R.anim.myrotate);
      break;
    case MENU_COMBO_ID:
      anim = AnimationUtils.loadAnimation(this, R.anim.mycombo);
      break;
    }
    // запускаем анимацию для компонента tv
    tv.startAnimation(anim);
    return super.onContextItemSelected(item);
  }
}
```

Анимация читается из xml-файла методом AnimationUtils.loadAnimation, на выходе получается объект типа Animation. Его используем в методе startAnimation, который запускает анимацию.

Все сохраним и запустим приложение. Вызываем контекстное меню для TextView, и тестируем анимации



Я использовал не все возможности и параметры. Есть еще, например, параметр **android:startOffset** – он указывает задержку при старте анимации. Т.е. если указать android:startOffset="1000", то анимация начнется через секунду. Это удобно использовать если вы делаете набор трансформ (<set>) и вам надо чтобы они запускались не одновременно, а в определенном порядке. Также полезный параметр **android:repeatCount** – это количество повторов.

Рекомендую поиграть параметрами в XML файлах и посмотреть, что получается.

На следующем уроке:

- создаем в приложении второй экран

Урок 21. Создание и вызов Activity

В этом уроке мы:

- создадим и вызовем второе Activity в приложении

Мы подобрались к очень интересной теме. На всех предыдущих уроках мы создавали приложения, которые содержали только **один экран** (Activity). Но если вы пользуетесь смартфоном с Android, то вы замечали, что экранов в приложении обычно больше. Если рассмотреть, например, почтовое приложение, то в нем есть следующие экраны: список аккаунтов, список писем, просмотр письма, создание письма, настройки и т.д. Пришла и нам пора научиться создавать **многоэкранные приложения**.

Создадим проект:

Project name: P0211_TwoActivity
Build Target: Android 2.3.3
Application name: TwoActivity
Package name: ru.startandroid.develop.p0211twoactivity
Create Activity: MainActivity

Откроем main.xml и создадим такой экран:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
     <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Go to Activity Two"
          android:id="@+id/btnActTwo">
     </Button>
</LinearLayout>
```

На экране одна кнопка, по нажатию которой будем вызывать второй экран.

Открываем MainActivity.java и пишем код:

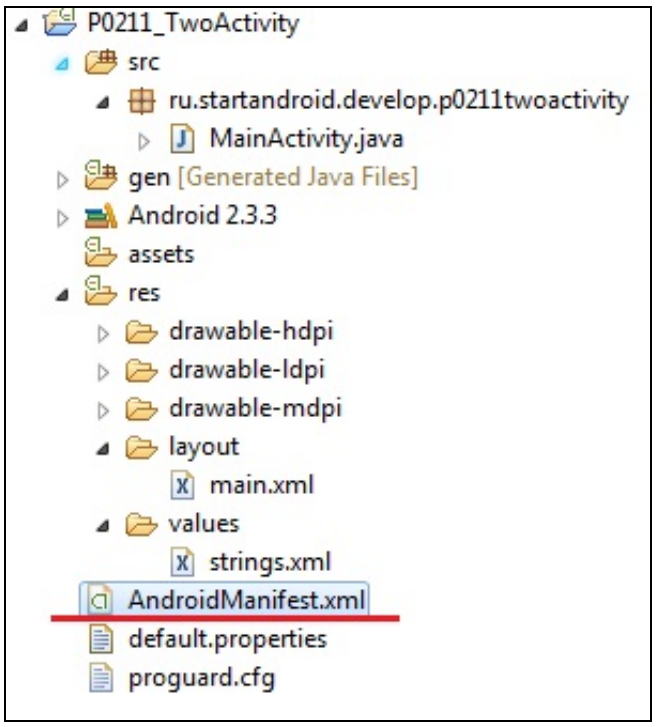
```
package ru.startandroid.develop.p0211twoactivity;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener {
```

```
  Button btnActTwo;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    btnActTwo = (Button) findViewById(R.id.btnActTwo);
    btnActTwo.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
    switch (v.getId()) {
    case R.id.btnActTwo:
      // TODO Call second activity
      break;
    default:
      break;
    }
  }
}
```

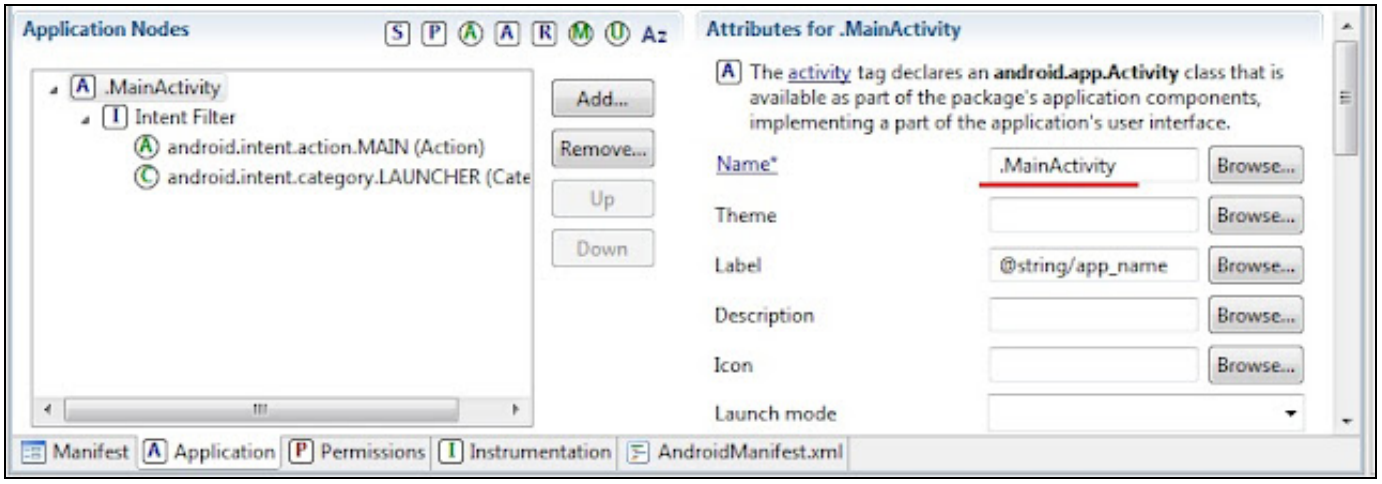
Мы определили кнопку **btnActTwo** и присвоили ей Activity в качестве обработчика. Реализация метода **onClick** для кнопки пока заполнена частично - определяем, какая кнопка была нажата. Чуть позже здесь мы будем **вызывать второй экран**. Но сначала этот второй экран надо **создать**.

Если помните, при создании проекта у нас по умолчанию создается Activity.

От нас требуется только указать **имя** этого Activity – обычно мы пишем здесь MainActivity. Давайте разбираться, что при этом происходит. Мы уже знаем, что **создается одноименный класс** MainActivity.java – который отвечает за поведение Activity. Но, кроме этого, Activity «**регистрируется**» в системе с помощью **манифест-файла** - AndroidManifest.xml.



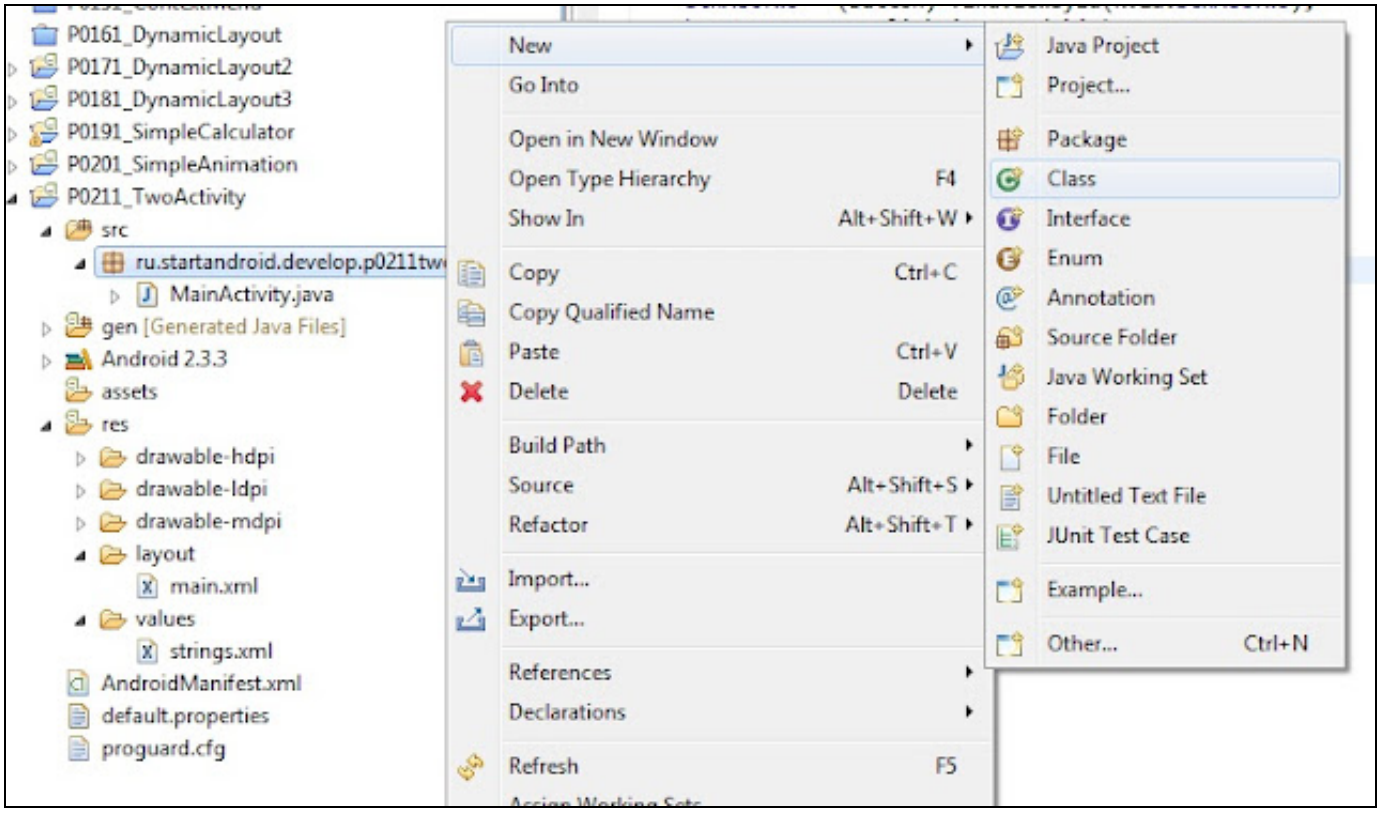
Давайте откроем этот файл:



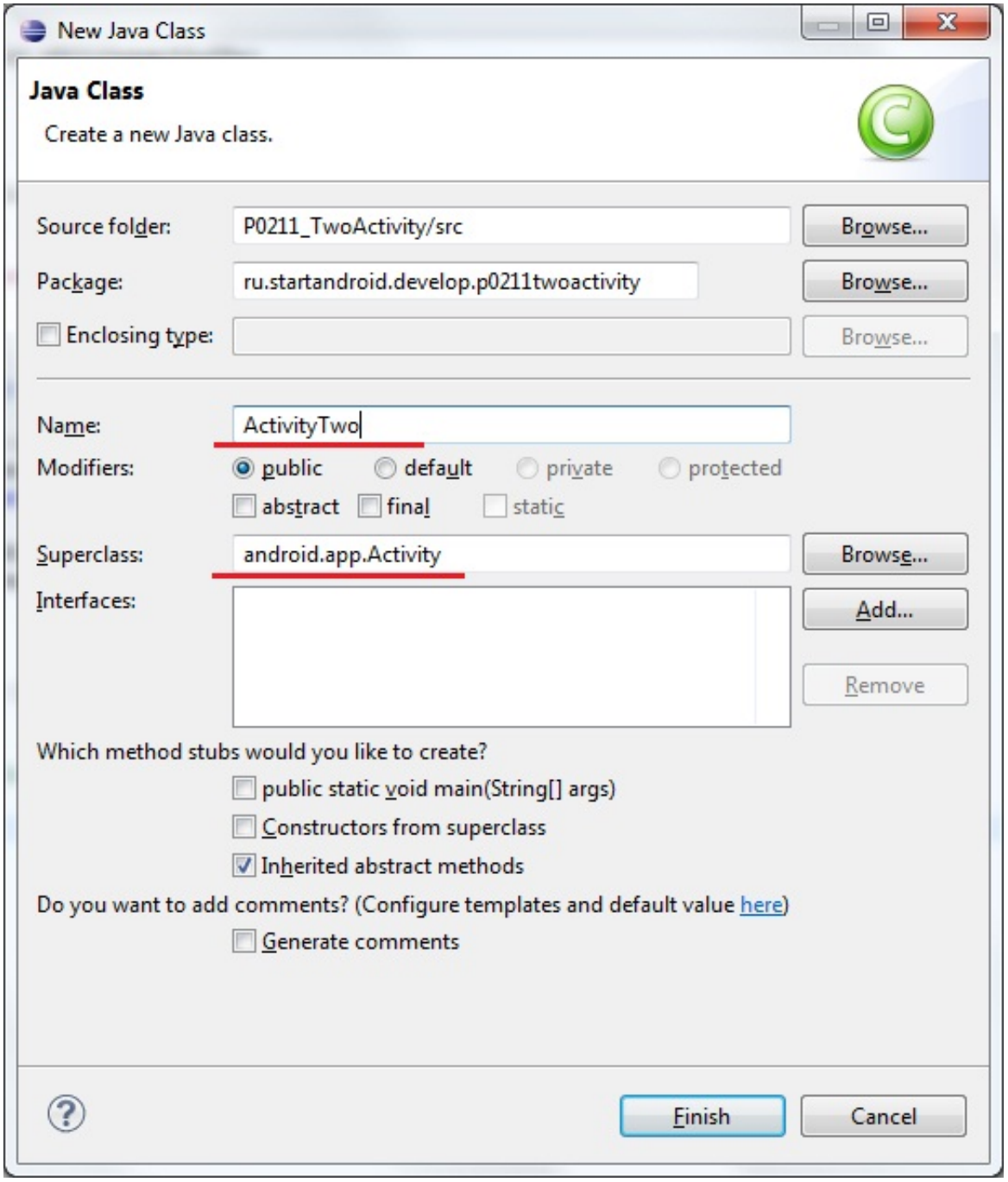
Нас интересует вкладка **Application**. Слева мы видим **MainActivity**. Если его раскрыть, внутри видим Intent Filter с определенными параметрами. Пока мы не знаем что это и зачем, сейчас нам это не нужно. Забегая вперед, скажу, что android.intent.action.MAIN показывает системе, что Activity является основной и будет первой отображаться при запуске приложения. А android.intent.category.LAUNCHER означает, что приложение будет отображено в общем списке приложений Android.

Справа в поле **Name** написано **.MainActivity**. Это имя класса, который отвечает за работу Activity (это же можно считать и именем Activity).

Итак, если мы хотим **создать** еще одно **Activity**, надо **создать класс** и **прописать Activity в AndroidManifest.xml**. Чтобы создать класс, жмем правой кнопкой на package **ru.startandroid.develop.p0201twoactivity** в папке проекта и выбираем **New** -> **Class**.



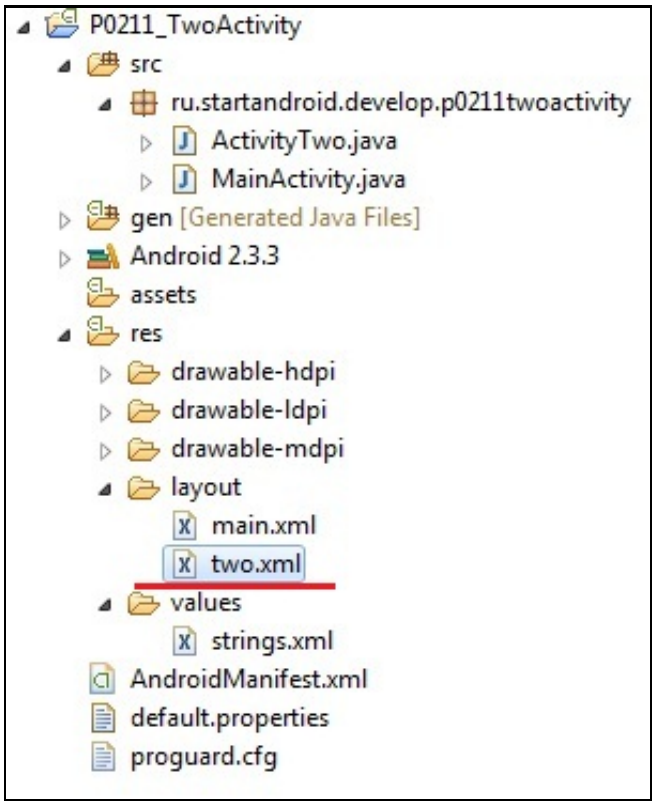
В появившемся окне вводим имя класса – **ActivityTwo**, и суперкласс – **android.app.Activity**.



Класс ActivityTwo создан. Он абсолютно пустой. Нам надо реализовать метод **onCreate**, который вызывается при создании Activity:

```
package ru.startandroid.develop.p0211twoactivity;

import android.app.Activity;
import android.os.Bundle;


public class ActivityTwo extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
  }
}
```

Не хватает вызова метода **setContentView**, который указал бы классу, чем заполнять экран. Этому методу на вход требуется **layout-файл**. Давайте создадим его в папке layout, там же где и main.xml. Назовем файл **two.xml**



Заполним этот файл следующим кодом:

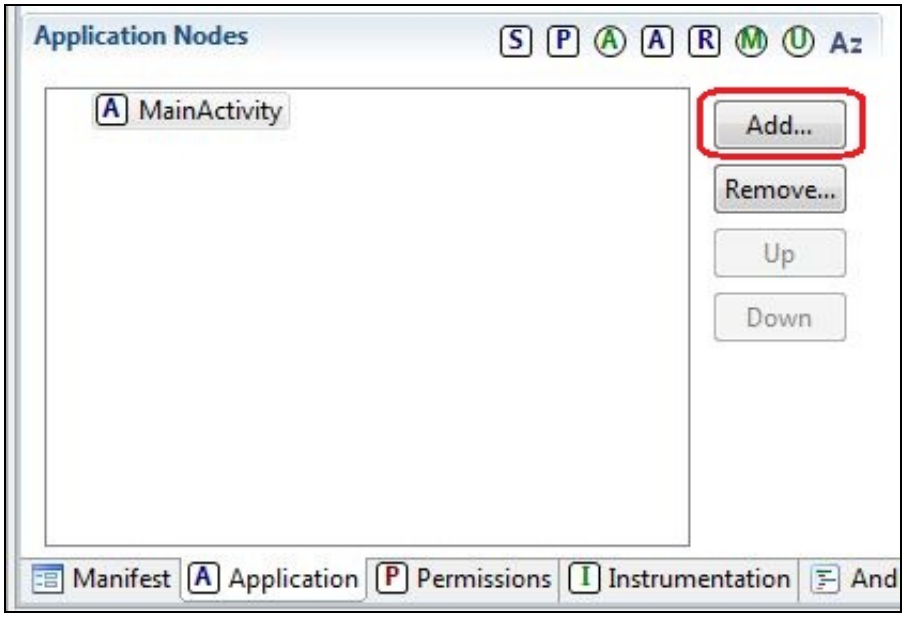
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
     <TextView
          android:id="@+id/textView1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="This is Activity Two">
     </TextView>
</LinearLayout>
```

Экран будет отображать **TextView** с текстом "This is Activity Two". Используем файл two.xml в методе setContentView в **ActivityTwo.java**

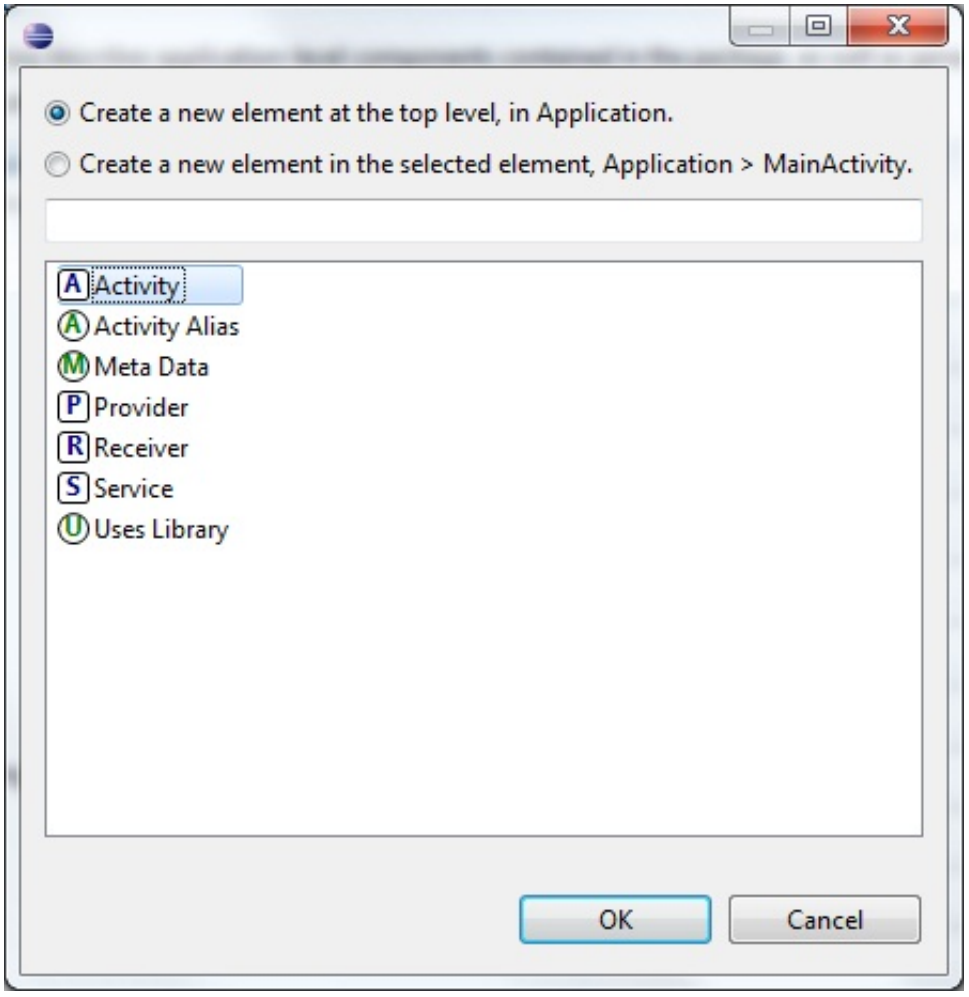
```
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.two);
  }
```

(добавляете только подчеркнутый код)

Сохраните все. Класс ActivityTwo готов, при создании он выведет на экран то, что мы настроили в layout-файле two.xml. Теперь надо прописать Activity в манифесте. Открываем AndroidManifest.xml, вкладка Application. Жмем кнопку **Add**.



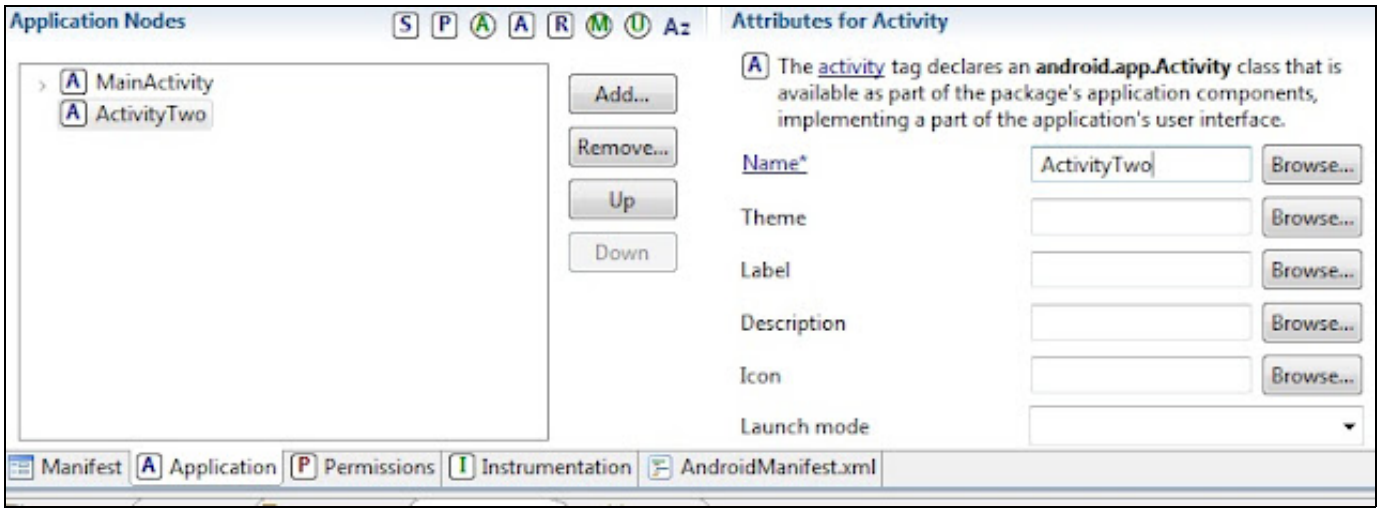
Далее, в появившемся окне сверху выберите пункт «Create a new element at the top level ...» (если есть выбор) , а из списка выбираем **Activity**.



Жмем **OK**, Activity создалось и появилось в списке. Осталось указать ему класс, который будет отвечать за его работу. Справа в поле **Name** можем **написать вручную имя класса**, который создавали – "ActivityTwo". А можем **нажать Browse** и **выбрать** его же из списка (надо будет немного подождать пока список сформируется). Кроме этого

больше ничего указывать и заполнять пока не надо. Сохраним все.

Теперь в **манифесте** прописаны **два Activity**, и каждое ссылается на **свой класс**.

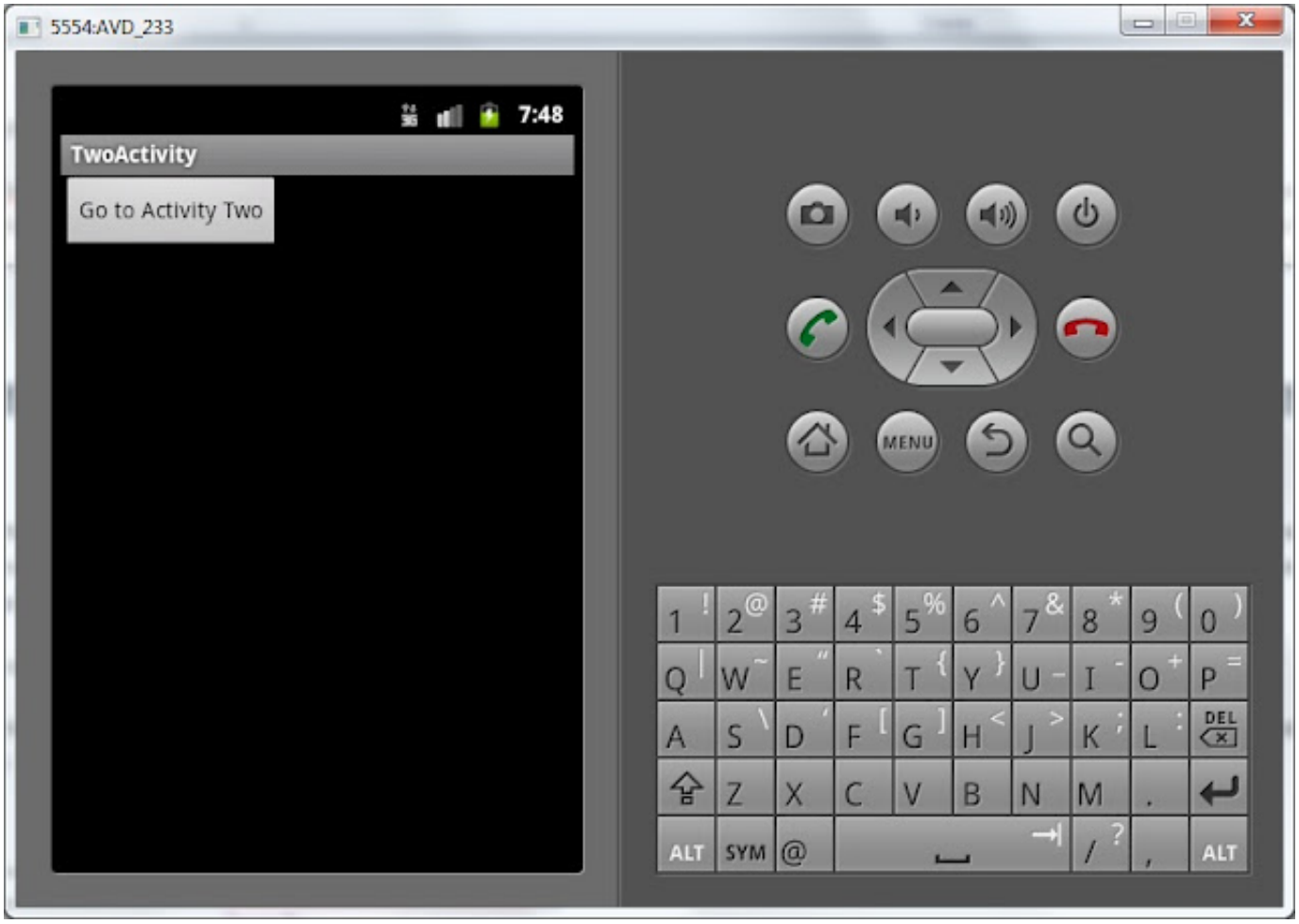


Нам осталось вернуться в MainActivity.java и довершить реализацию метода onClick (нажатие кнопки), а именно - прописать вызов ActivityTwo. Открываем MainActivity.java и добавляем строки:

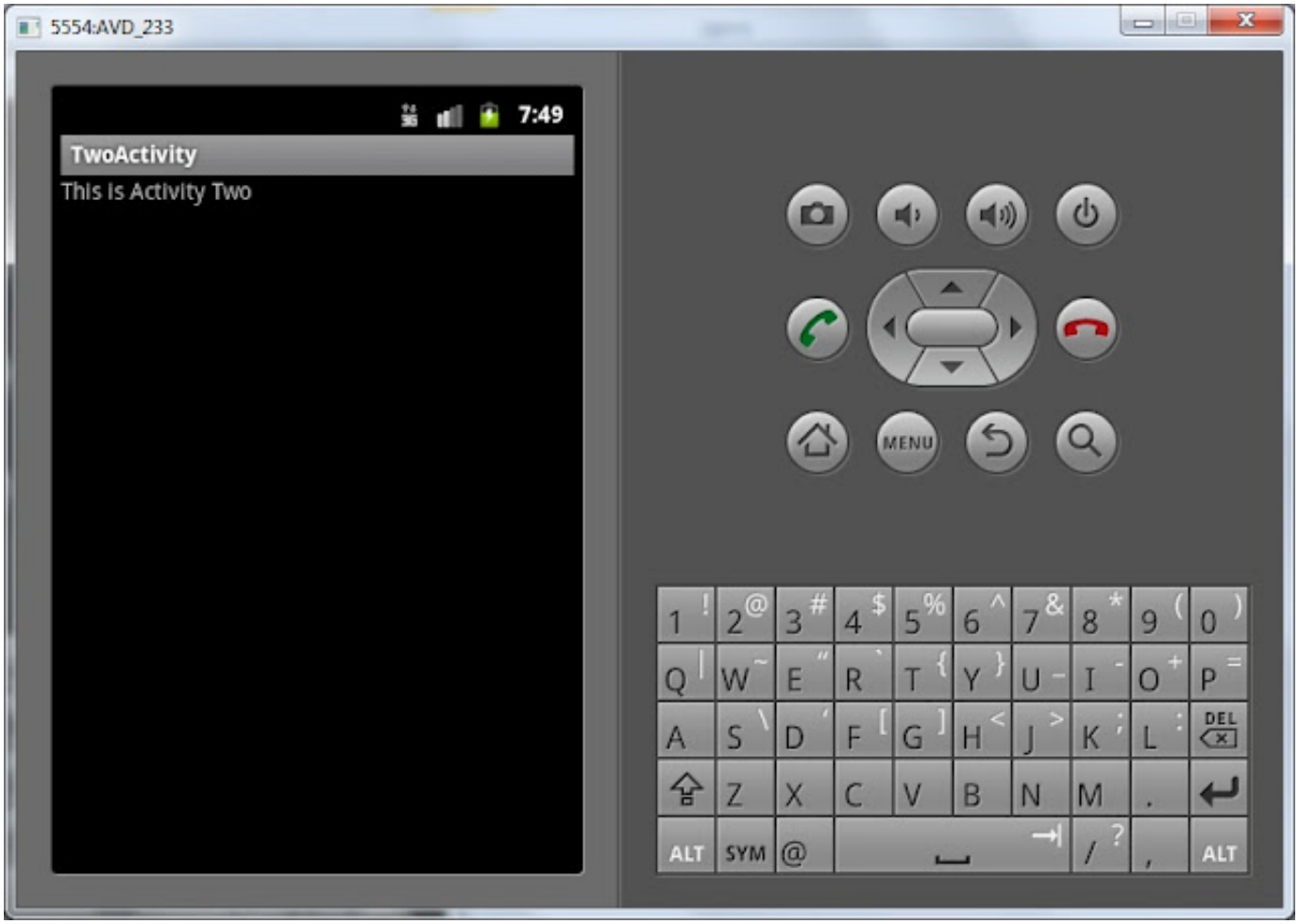
```
    case R.id.btnActTwo:
      Intent intent = new Intent(this, ActivityTwo.class);
      startActivity(intent);
      break;
```

(добавляете только подчеркнутый код)

Обновите импорт (CTRL+SHIFT+O), сохраните все и можем всю эту конструкцию запускать. При запуске появляется MainActivity



Нажимаем на кнопку и переходим на ActivityTwo



Код вызова Activity пока не объясняю и теорией не гружу, урок итак получился сложным. Получилось много текста и скриншотов, но на самом деле процедура минутная. Поначалу, возможно, будет непонятно, но постепенно втянемся. Создадим штук 5-6 новых Activity в разных проектах и тема уляжется в голове.

Пока попробуйте несколько раз пройти мысленно эту цепочку действий и усвоить, что для создания Activity необходимо создать класс (который наследует android.app.Activity) и создать соответствующую запись в манифест-файле.

На следующем уроке:

- разбираемся в коде урока 21
- теория по Intent и Intent Filter (не пропустите, тема очень важная)
- немного о Context

Урок 22. Intent, Intent Filter, Context - теория

В этом уроке:

- разбираемся в коде урока 21
- теория по Intent и Intent Filter
- немного о Context

На прошлом уроке (№ 21) мы создали **приложение**, которое содержит **два Activity**. Напомню, что **для создания** Activity, необходимо:
- **создать класс**, с суперклассом android.app.Activity
- **создать Activity-запись в манифесте** и указать ей созданный класс в поле Name

Надеюсь прошлый урок не вызвал особых трудностей и процедура создания Activity примерно уложилась в голове. Теперь мы можем обратить внимание на **код вызова Activity**.

```
        Intent intent = new Intent(this, ActivityTwo.class);
        startActivity(intent);
```

Мы использовали объект **Intent**. О нем можно почитать здесь, здесь и здесь. Правда инфа достаточно сложна для понимания с нуля. Я попробую своими словами объяснить.

Что такое Intent

В нашем случае **Intent** – это **объект**, в котором мы **прописываем**, **какое Activity** нам необходимо **вызвать**. После чего мы передаем этот Intent-объект методу **startActivity**, который находит **соответствующее Activity** и показывает его. При создании Intent мы использовали конструктор **Intent (Context packageContext, Class cls)** с двумя параметрами.

Первый параметр – это Context. Если помните, когда мы программно создавали View в одном из прошлых уроков, мы тоже использовали в конструкторах объект Context. **Activity** является **подклассом Context**, поэтому мы можем использовать ее – **this**. Вкратце, **Context** – это объект, который **предоставляет** доступ к **базовым функциям** приложения таким как: доступ к ресурсам, к файловой системе, вызов Activiy и т.д. Я думаю, в дальнейшем мы рассмотрим примеры, где явно увидим, зачем Context передается и как используется.

Второй параметр – **имя класса**. Напомню, что при создании записи Activity в манифест-файле мы указываем имя класса. И теперь если мы укажем тот же класс в Intent – то система, просмотрев манифест-файл обнаружит соответствие и покажет соответствующий Activity.

В этом можно легко убедится. Мы удалим запись об Activity из манифест-файла и попробуем его после этого вызвать. Откройте проект из прошлого урока **P0211_TwoActivity**, откройте **манифест-файл**, вкладка **Application** и удалите запись об **ActivityTwo** с помощью кнопки **Remove**. Сохраните все, запустите приложение и попробуйте вызвать Activity кнопкой “Go to Activity Two”. Приложение выдаст ошибку. Если посмотреть логи, то видим следующий текст:

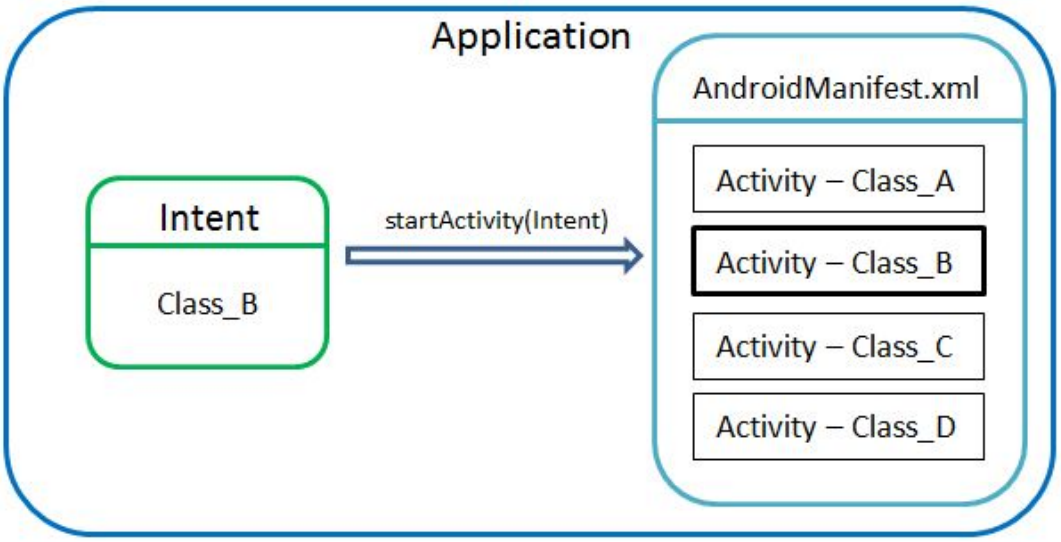
ERROR/AndroidRuntime(367): android.content.ActivityNotFoundException: Unable to find explicit activity class {ru.startandroid.develop.p0211twoactivity/ru.startandroid.develop.p0211twoactivity.ActivityTwo}; have you declared this activity in your AndroidManifest.xml?

(Логи - вкладка LogCat в Eclipse. Если не видно такой, то идем в меню Window -> Show View -> Other, папка Android -> LogCat)

Система говорит нам, что не нашла такого Activity класса и любезно подсказывает, что, возможно, он **не прописан в манифест-файле**. Снова пропишите Activity в манифест-файле, все сохраните и запускайте. Теперь должно работать.

Явный вызов

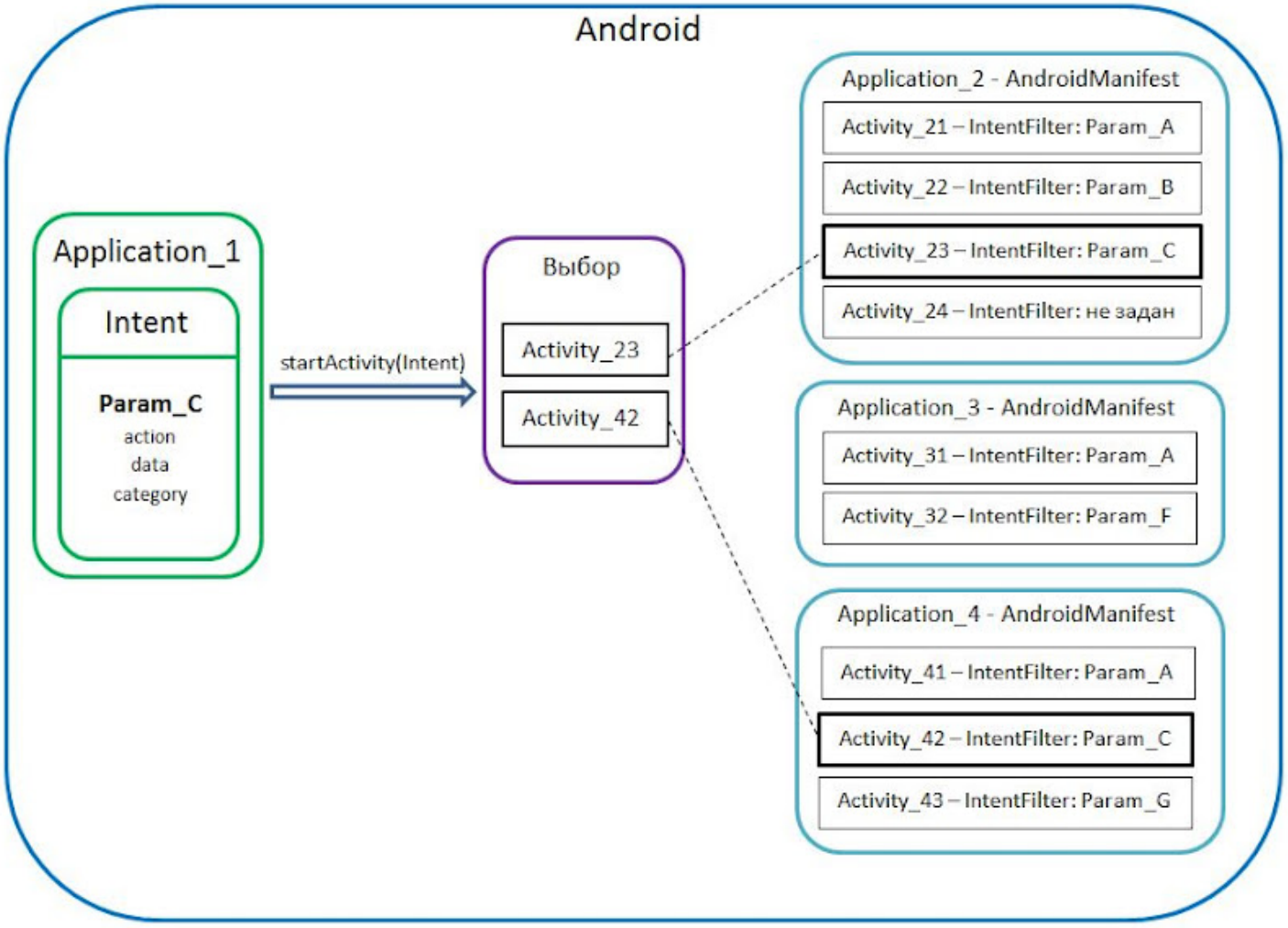
Вызов Activity с помощью такого Intent – это **явный** вызов. Т.е. с помощью класса мы **явно** указываем какое Activity хотели бы увидеть. Это обычно используется **внутри одного приложения**. Схематично это можно изобразить так:



Здесь мы создаем **Intent**, в качестве параметра передаем ему **класс Class_B**. Далее вызываем метод **startActivity** с созданным **Intent** в качестве параметра. Метод проверяет **AndroidManifest** на наличие **Activity** связанной с классом **Class_B** и если находит, то отображает. Все это в пределах одного приложения.

Неявный вызов

Существует также **неявный** вызов Activity. Он отличается тем, что при создании **Intent** мы используем **не класс**, а **заполняем** параметры **action**, **data**, **category** определенными значениями. Комбинация этих значений определяют цель, которую мы хотим достичь. Например: отправка письма, открытие гиперссылки, редактирование текста, просмотр картинки, звонок по определенному номеру и т.д. В свою очередь для **Activity** мы прописываем **Intent Filter** - это набор тех же параметров: **action**, **data**, **category** (но значения уже свои - зависят от того, что умеет делать Activity). И если параметры нашего Intent **совпадают** с условиями этого фильтра, то Activity вызывается. Но при этом поиск уже идет **по всем Activity всех приложений** в системе. Если находится **несколько**, то система предоставляет вам **выбор**, какой именно программой вы хотите воспользоваться. Схематично это можно изобразить так:



В Application_1 создается **Intent**, заполняются параметры **action**, **data**, **category**. Для удобства, получившийся набор параметров назовем **Param_C**. С помощью **startActivity** этот **Intent** отправляется на поиски **подходящей** Activity, которая сможет выполнить то, что нам нужно (т.е. то, что определено с помощью Param_C). В системе есть разные приложения, и в каждом из них несколько Activity. Для некоторых Activity определен Intent Filter (наборы Param_A, Param_B и т.д.), для некоторых нет. Метод startActivity **сверяет** набор параметров **Intent** и наборы параметров **Intent Filter** для каждой **Activity**. Если наборы **совпадают** (Param_C для обоих), то Activity считается **подходящей**.

Если в итоге нашлась только одна Activity – она и отображается. Если же нашлось **несколько** подходящих Activity, то пользователю выводится **список**, где он может сам выбрать какое приложение ему использовать.

Например, если в системе установлено несколько музыкальных плееров, и вы запускаете mp3, то система выведет вам список Activity, которые умеют играть музыку и попросит выбрать, какое из них использовать. А те Activity, которые умеют редактить текст, показывать картинки, звонить и т.п. будут проигнорированы.

Если для Activity не задан Intent Filter (Activity_24 на схеме), то Intent с параметрами ему никак не подойдет, и оно тоже будет проигнорировано.

Если проводить аналогии - можно сравнить Intent с ключом, а Intent Filter с замкОм, за которым сидит прекрасное Activity)))

Мы будем постепенно узнавать нюансы этого механизма и **значения** которыми заполняются параметры action, data и category в Intent и Intent Filter. Сейчас важно понять, что в случае **неявного** вызова одно приложение посылает **Intent**, а все другие сверяют его параметры со своими **Activity -> Intent Filter**. Intent – это базовое понятие системы Android и без него нам никуда. Оно применяется не только для Activity. Но об этом позднее.

Ну вот, хотел написать пару вводных слов, а получилось достаточно подробное объяснение со схемами и примерами) Надеюсь, что у меня получилось донести смысл технологии Intent-ов. В дальнейшем будем практиковаться и закрепим тему.

На следующем уроке:

- Activity LifeCycle – поведение Activity при создании, вызове, закрытии

Урок 23. Activity Lifecycle. В каких состояниях может быть Activity

В этом уроке:

- Activity LifeCycle – поведение Activity при создании, вызове, закрытии

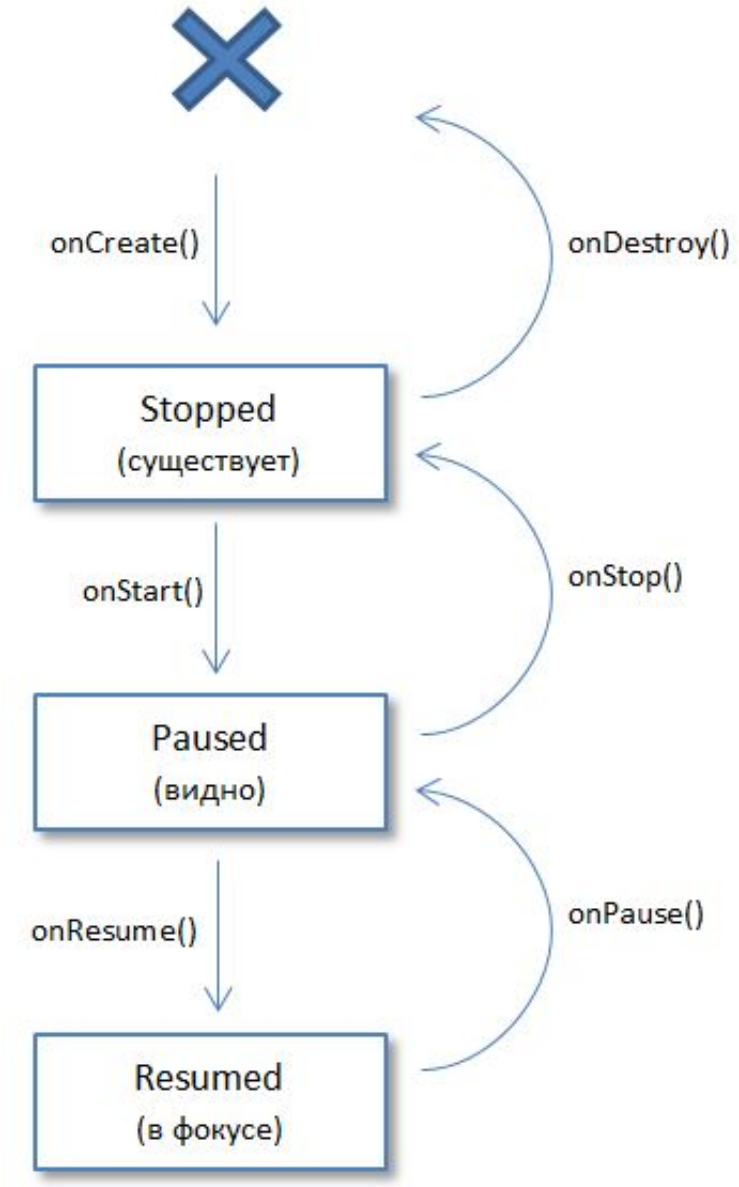
Теория

При работе приложения, мы **создаем** новые **Activity** и **закрываем** старые, **сворачиваем** приложение, снова **открываем** и т.д. Activity умеет обрабатывать все эти движения. Это необходимо, например, для освобождения ресурсов или сохранения данных. В хелпе достаточно подробно это описано.

Созданное при работе приложения **Activity** может быть в одном из **трех состояний**:

Resumed - Activity видно на экране, оно находится в фокусе, пользователь может с ним взаимодействовать. Это состояние также иногда называют Running.
Paused - Activity не в фокусе, пользователь не может с ним взаимодействовать, но его видно (оно перекрыто другим Activity, которое занимает не весь экран или полупрозрачно).
Stopped - Activity не видно (полностью перекрывается другим Activity), соответственно оно не в фокусе и пользователь не может с ним взаимодействовать.

Когда Activity переходит из одного **состояния** в другое, система вызывает различные его **методы**, которые мы можем заполнять своим кодом. Схематично это можно изобразить так:



Для упрощения понимания я дал краткое описание состояний в скобках под названиями. А крестом обозначил отсутствие Activity.

Итак, мы имеем следующие методы Activity, которые вызывает система:

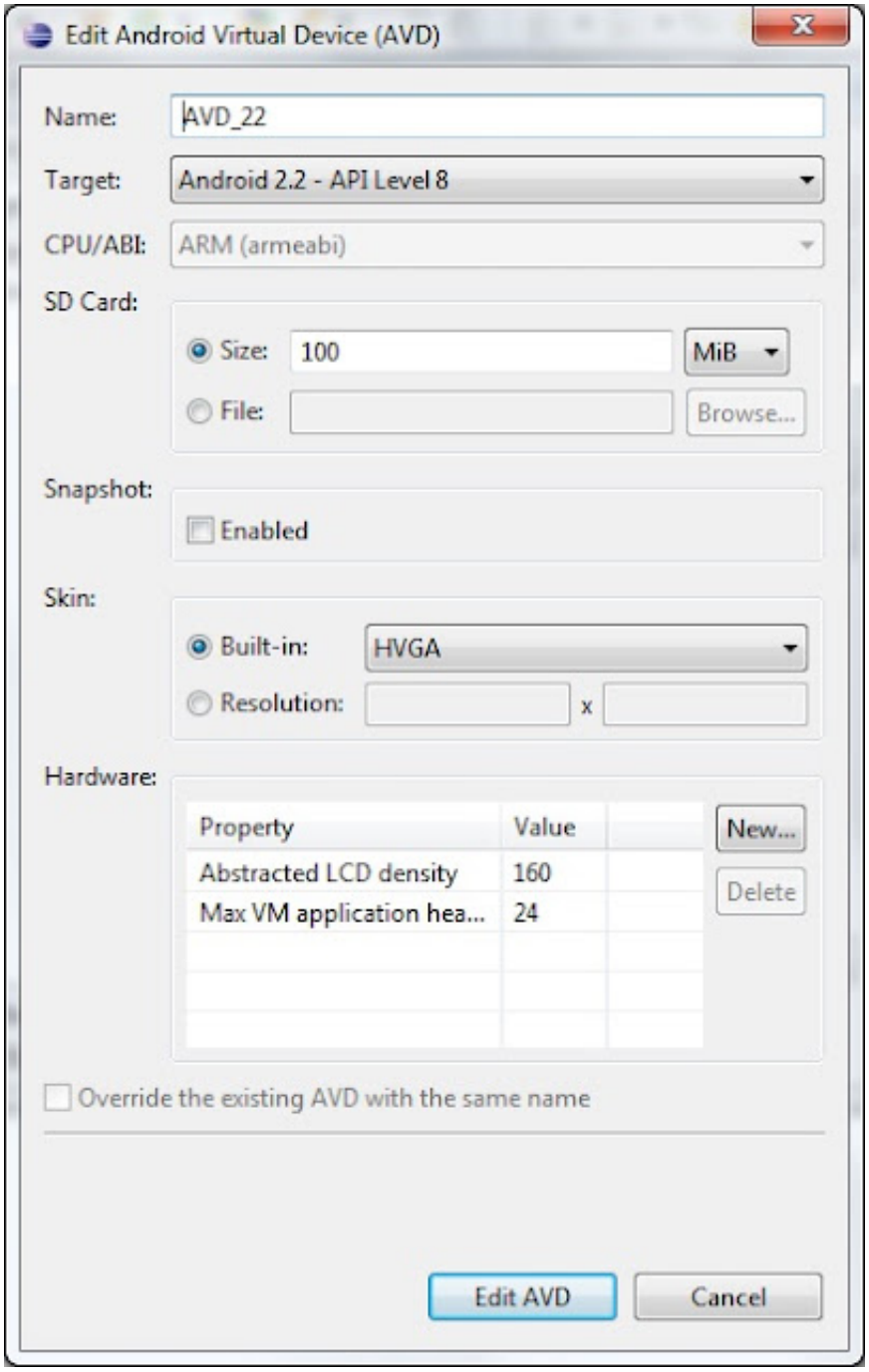
onCreate() – вызывается при первом создании Activity
onStart() – вызывается перед тем, как Activity будет видно пользователю
onResume() – вызывается перед тем как будет доступно для активности пользователя (взаимодействие)

onPause() – вызывается перед тем, как будет показано другое Activity
onStop() – вызывается когда Activity становится не видно пользователю
onDestroy() – вызывается перед тем, как Activity будет уничтожено

Т.е. эти методы **НЕ** вызывают смену состояния. Наоборот, смена состояния Activity является триггером, который вызывает эти методы. Тем самым нас уведомляют о смене, и мы можем реагировать соответственно. Посмотрим на практике, когда и в каком порядке вызываются эти методы.

Практика

В этом уроке нам надо будет эмулировать событие смены ориентации экрана. Но эмулятор с Android 2.3 делает это криво, поэтому в проекте будем использовать версию 2.2. Для этого надо создать новое AVD по версии 2.2



Создадим проект (обратите внимание, используем Android 2.2.):

Project name: P0231_OneActivityState
Build Target: Android 2.2
Application name: OneActivityState
Package name: ru.startandroid.develop.p0231oneactivitystate
Create Activity: MainActivity

Layout не меняем, нам он сейчас не важен. Открываем **MainActivity.java**, там как обычно код по умолчанию:

```
package ru.startandroid.develop.p0231oneactivitystate;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

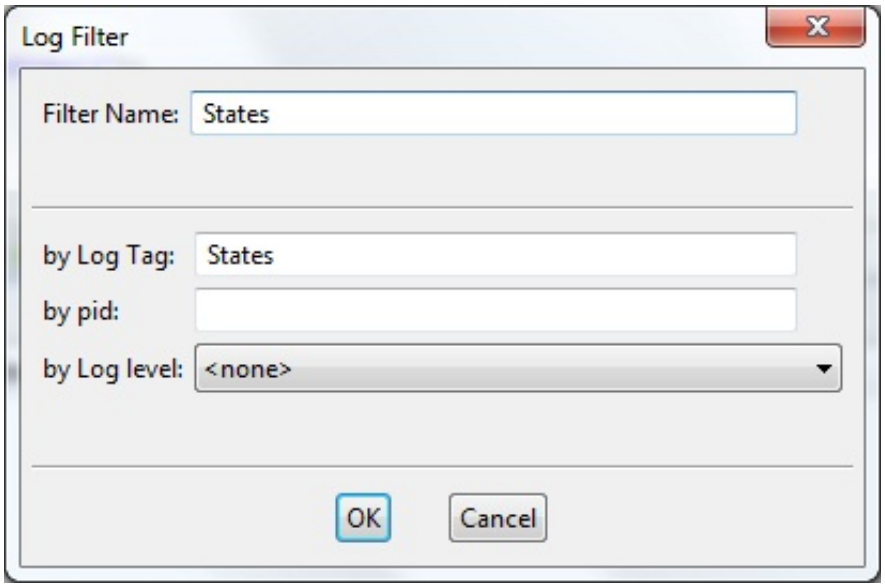
Мы видим, что реализован уже знакомый нам по схеме метод **onCreate**. Повторюсь, важно понимать, что этот метод **НЕ создает** Activity. Создание – это дело системы. Т.е. система сама создает Activity, а нам дает возможность немного поучаствовать и выполнить свой код в методе onCreate(). Мы этой возможностью пользуемся и говорим системе, что Activity должна отобразить экран из R.layout.main.

Добавим все остальные **методы** из схемы, и в каждый добавим запись в **лог**.

```
package ru.startandroid.develop.p0231oneactivitystate;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

  final String TAG = "States";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.d(TAG, "MainActivity: onCreate()");
    }

  @Override
  protected void onStart() {
    super.onStart();
    Log.d(TAG, "MainActivity: onStart()");
  }

  @Override
  protected void onResume() {
    super.onResume();
    Log.d(TAG, "MainActivity: onResume()");
  }

  @Override
  protected void onPause() {
    super.onPause();
    Log.d(TAG, "MainActivity: onPause()");
  }

  @Override
  protected void onStop() {
    super.onStop();
    Log.d(TAG, "MainActivity: onStop()");
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "MainActivity: onDestroy()");
  }
}
```

В каментах подсказали важное замечание! При реализации этих методов обязательно вызывайте соответствующие методы супер-класса и обязательно перед вашим кодом. См. код выше. Каждый метод содержит вызов метода супер-класса и свой код расположен после этих вызовов.

Теперь, когда методы будут вызываться, мы будем видеть это в логах. Настроим фильтр на тег «States», чтобы не искать свои сообщения в общей куче логов. Как это делается мы проходили в уроке 12



Все сохраним и запустим приложение. После того, как запустилось, смотрим лог:

MainActivity: onCreate()
MainActivity: onStart()
MainActivity: onResume()

Activity создалось, прошло два состояния (Stopped, Paused) и теперь находится в третьем состоянии - Resumed. Т.е. оно создалось (onCreate), отобразилось (onStart) и получило возможность взаимодействовать с пользователем (onResume).

Теперь нажмем кнопку Back на эмуляторе. Activity закрылось. Смотрим лог:

MainActivity: onPause()
MainActivity: onStop()
MainActivity: onDestroy()

Activity проделывает путь, обратный созданию. Сначала теряет фокус (onPaused), затем исчезает с экрана (onStop), затем полностью уничтожается (onDestroy).

Смена ориентации экрана

Посмотрим, как ведет себя Activity, когда происходит смена ориентации экрана. Запустите снова приложение (либо найдите его в списке приложений в системе на эмуляторе, либо снова нажмите CTRL+F11 в Eclipse). В логах снова отобразились три метода, вызванные при создании. Теперь в эмуляторе нажмите CTRL+F12, ориентация сменилась. Кажется, что ничего особенного не произошло, но смотрим логи и видим:

MainActivity: onPause()

MainActivity: onStop()
MainActivity: onDestroy()
MainActivity: onCreate()
MainActivity: onStart()
MainActivity: onResume()

Activity полностью уничтожается и снова создается. При этом обычно выполняются процедуры сохранения и восстановления данных, чтобы не потерялись данные, и приложение сохранило свой вид. Про то, как это делается, мы будем говорить в последующих уроках.

Также есть еще метод **onRestart**. Он вызывается перед методом **onStart**, если Activity не создается с нуля, а восстанавливается из состояния **Stoped**. Его мы рассмотрим в следующем уроке.

Обычно в учебниках эта тема дается по-другому. Но мне это шаблонное объяснение кажется недостаточно понятным, поэтому я написал свое. Как всегда, надеюсь, что у меня получилось раскрыть тему)

Советую вам после этого урока прочитать хелп, ссылку на который я дал в самом начале урока. Там все очень хорошо написано. И знания лучше усвоятся. Пока что, главное – это понять в какой момент, какой метод вызывается. А уже дальше мы будем разбираться, как это можно использовать и что там кодить.

На следующем уроке:

- изучаем смену состояния на примере двух Activity

Урок 24. Activity Lifecycle, пример смены состояний с двумя Activity

В этом уроке:

- изучаем смену состояния на примере двух Activity

На прошлом уроке мы рассмотрели, какие **состояния** проходит **Activity** за время своего существования и какие **методы** при этом вызываются. Но мы видели Activity только в состоянии **Resumed** (т.е. его видно, и оно в фокусе). На этом уроке на примере **двух Activity** попробуем понять, в каком случае Activity может остаться в состоянии **Stopped**, т.е. не видно и не в фокусе, но существует в памяти.

Создадим проект:

Project name: P0241_TwoActivityState
Build Target: Android 2.3.3
Application name: TwoActivityState
Package name: ru.startandroid.develop.p0241twoactivitystate
Create Activity: MainActivity

В main.xml пишем следующее:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
     <TextView
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="@string/hello">
     </TextView>
     <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Go to Activity Two"
          android:id="@+id/btnActTwo">
     </Button>
</LinearLayout>
```

Кнопка “Go to Activity Two” будет вызывать второе Activity.

Откроем MainActivity.java и пишем туда все **методы**, на этот раз, включая **onRestart**, и в методах прописываем запись в **логи**. Также описываем и находим **кнопку**, присваиваем ей **обработчик**. В методе **onClick** пока ничего не пишем.

```
package ru.startandroid.develop.p0241twoactivitystate;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener {

  final String TAG = "States";

  Button btnActTwo;
```

```
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    btnActTwo = (Button) findViewById(R.id.btnActTwo);
    btnActTwo.setOnClickListener(this);

    Log.d(TAG, "MainActivity: onCreate()");
  }

  @Override
  protected void onRestart() {
    super.onRestart();
    Log.d(TAG, "MainActivity: onRestart()");
  }

  @Override
  protected void onStart() {
    super.onStart();
    Log.d(TAG, "MainActivity: onStart()");
  }

  @Override
  protected void onResume() {
    super.onResume();
    Log.d(TAG, "MainActivity: onResume()");
  }

  @Override
  protected void onPause() {
    super.onPause();
    Log.d(TAG, "MainActivity: onPause()");
  }

  @Override
  protected void onStop() {
    super.onStop();
    Log.d(TAG, "MainActivity: onStop()");
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "MainActivity: onDestroy()");
  }

  @Override
  public void onClick(View v) {
  }
}
```

Какие методы и в каком порядке выполняются при работе одного Activity, мы видели на прошлом уроке. Сейчас нам интересно поведение при **двух** Activity, поэтому создаем второе Activity. Назовем ее **ActivityTwo**. Вспоминаем прошлые уроки: надо **создать класс** с таким именем и с **суперклассом** android.app.Activity, и прописать новое Activity в **манифест**-файле. Также надо создать layout-файл, назовем его two.xml и заполним этим кодом:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
     <TextView
          android:id="@+id/textView1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="This is Activity Two">
```

```
    </TextView>
  </LinearLayout>
```

Просто TextView с текстом, чтобы было понятно, что это ActivityTwo.

Создаем класс. Код ActivityTwo.java:

```
package ru.startandroid.develop.p0241twoactivitystate;


import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class ActivityTwo extends Activity {

  final String TAG = "States";

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.two);
    Log.d(TAG, "ActivityTwo: onCreate()");
  }

  @Override
  protected void onRestart() {
    super.onRestart();
    Log.d(TAG, "ActivityTwo: onRestart()");
  }

  @Override
  protected void onStart() {
    super.onStart();
    Log.d(TAG, "ActivityTwo: onStart()");
  }

  @Override
  protected void onResume() {
    super.onResume();
    Log.d(TAG, "ActivityTwo: onResume()");
  }

  @Override
  protected void onPause() {
    super.onPause();
    Log.d(TAG, "ActivityTwo: onPause()");
  }

  @Override
  protected void onStop() {
    super.onStop();
    Log.d(TAG, "ActivityTwo: onStop()");
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "ActivityTwo: onDestroy()");
  }
}
```

Не забудьте добавить запись об **ActivityTwo** в **манифест**. И теперь мы можем дописать код метода **onClick** в **MainActivity.java,** прописав там вызов ActivityTwo

```
  @Override
  public void onClick(View v) {
    Intent intent = new Intent(this, ActivityTwo.class);
    startActivity(intent);
```

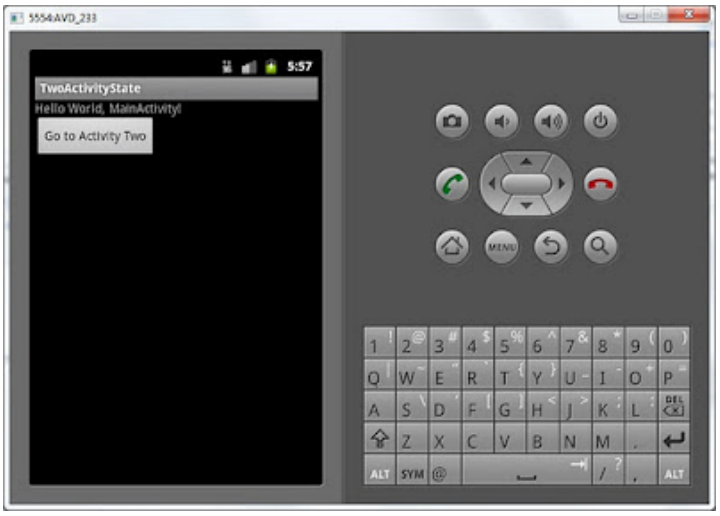
```
}
```

(добавляете только подчеркнутый код).

Фильтр логов должен был остаться с прошлого урока. Используем его. Если нет, то создайте фильтр по тегу **States**.

Все сохраним и приступим к испытаниям.

Шаг1. Запускаем приложение. Появилось MainActivity.



Логи:

MainActivity: onCreate()
MainActivity: onStart()
MainActivity: onResume()

Все, как и в прошлый раз - вызываются **три** метода. Activity проходит через состояния Stopped, Paused и остается в состоянии Resumed.

Шаг 2. Жмем кнопку «Go to Activity Two» на экране и появляется ActivityTwo.

Логи:

MainActivity: onPause()
ActivityTwo: onCreate()
ActivityTwo: onStart()
ActivityTwo: onResume()
MainActivity: onStop()

Давайте разбираться. Вызов MainActivity.onPause означает, что MainActivity теряет фокус и переходит в состояние Paused. Затем создается (onCreate), отображается (onStart) и получает фокус (onResume) ActivityTwo. Затем перестает быть видно (onStop) MainActivity. Обратите внимание, что не вызывается onDestroy для MainActivity, а значит, оно не уничтожается. MainActivity остается в памяти, в состоянии Stopped. А ActivityTwo – находится в состоянии Resumed. Его видно и оно в фокусе, с ним можно взаимодействовать.

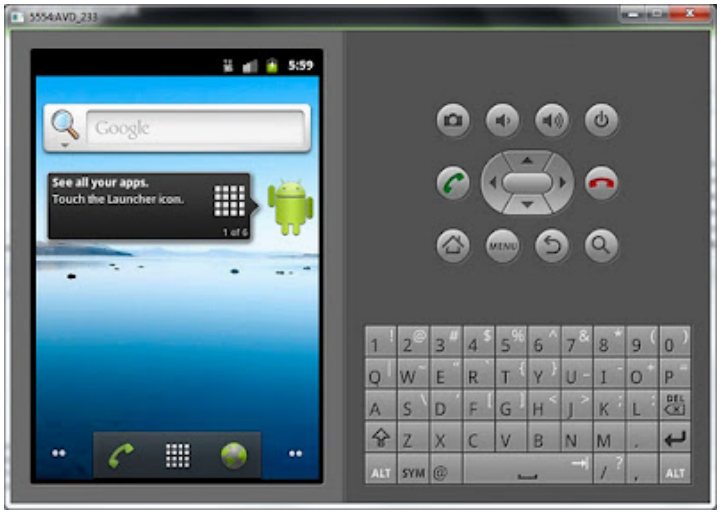
Шаг 3. Жмем кнопку Назад (Back) на эмуляторе. Мы вернулись в MainActivity.

Логи:

ActivityTwo: onPause()
MainActivity: onRestart()
MainActivity: onStart()
MainActivity: onResume()
ActivityTwo: onStop()
ActivityTwo: onDestroy()

ActivityTwo.onPause означает, что ActivityTwo теряет фокус и переходит в состояние Paused. MainActivity теперь должна восстановиться из статуса Stopped. В конце прошлого урока я написал: «Метод onRestart вызывается перед методом onStart, если Activity не создается с нуля, а восстанавливается из состояния Stoped» – это как раз наш случай, MainActivity не было уничтожено системой, оно висело в памяти. Поэтому вызывается MainActivity.onRestart. Далее вызываются методы MainActivity.onStart и MainActivity.onResume – значит MainActivity перешло в состояние Paused (отобразилось) и Resumed (получило фокус). Ну и вызов методов onStop и onDestroy означает, что ActivityTwo было переведено в статус Stopped (потеряло видимость) и было уничтожено.

Шаг 4. Жмем еще раз Назад и наше приложение закрылось.



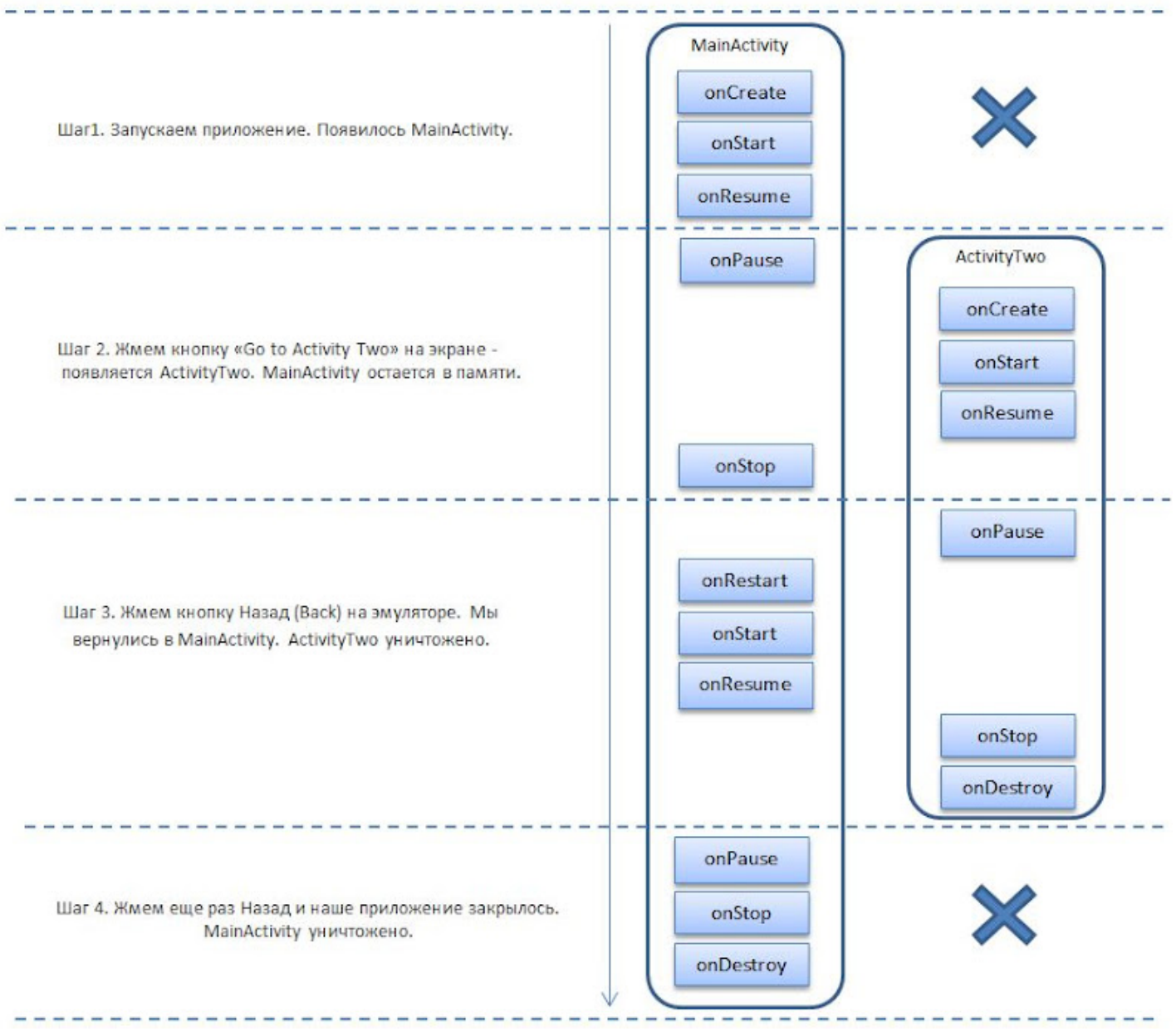
Логи:

MainActivity: onPause()

MainActivity: onStop()
MainActivity: onDestroy()

Логи показывают, что MainActivity перешло в состояние Paused, Stopped и было уничтожено.

Если с первого раза непонятно, попробуйте прогнать алгоритм **несколько раз** и сверяйтесь со **схемой** с прошлого урока урока. Она достаточно наглядная и поможет разобраться. Попробуйте расписать всю схему на бумаге и нарисовать смену статусов Activity. Я здесь тоже приведу схему шагов для наглядности.



Мы увидели, что Activity не обязательно уничтожается, когда его не видно, а может оставаться в памяти. В связи с этим, думаю, наверняка возник вопрос: почему на шаге 2 MainActivity исчезло с экрана, но осталось висеть в памяти и не было уничтожено? Ведь на шаге 3 было уничтожено ActivityTwo после того, как оно пропало с экрана. А на шаге 4 было в итоге уничтожено и MainActivity. Почему шаг 2 стал исключением?

Об этом мы поговорим на следующем уроке, т.к. этот и так получился слишком заумным. Но тема очень важная и одна из ключевых для понимания принципов работы Android.

Если чего-то не получилось, пишите в каменты.

На следующем уроке:

- немного теории по Task
- фиксируем Activity в состоянии Paused

Урок 25. Task. Что это такое и как формируется

В этом уроке:

- немного теории по Task
- фиксируем Activity в состоянии Paused

В этом уроке узнаем, куда помещается Activity, пока его не видно. И откуда оно достается при нажатии кнопки назад. В хелпе об этом написано достаточно понятно. Я сделаю краткий перевод основной части этого хелпа и использую их схемы.

Task

Мы уже знаем, что приложение может содержать несколько Activity. И что Activity умеет вызывать Activity из других приложений с помощью Intent и Intent Filter. Если вы хотите отправить письмо из вашего приложения, вы вызываете Activity почтовой программы и передаете ей данные. Письмо уходит и вы возвращаетесь в ваше приложение. Создается ощущение, что все это происходило в рамках одного приложения. Такая «бесшовность» достигается засчет того, что оба Activity (ваше и почтовое) были в одном Task.

Прежде, чем продолжу объяснять, хочу сразу привести аналогию, чтобы тему легче было понять. В скобках я буду давать понятия-аналоги из Android.

Механизм организации Activity в Android очень схож по реализации с навигацией в браузере. Вы находитесь в одной вкладке(Task) и открываете страницы (Activity) переходя по ссылкам (Intent). В любой момент можете вернуться на предыдущую страницу, нажав кнопку Назад. Но кнопка Вперед отсутствует, т.к. страница, на которой была нажата кнопка Назад, стирается из памяти. И надо снова нажимать ссылку, если хотим попасть на нее. Если вам надо открыть что-то новое, вы создаете новую вкладку и теперь уже в ней открываете страницы, переходите по ссылкам, возвращаетсь назад. В итоге у вас есть несколько вкладок. Большинство из них на заднем фоне, а одна (активная, с которой сейчас работаете) – на переднем.

В итоге список аналогий браузера и Android таков:

Браузер – Android
Вкладка с историей посещений – Task
Страница – Activity
Ссылка – Intent

Теперь вам будет более понятен текст про Task.

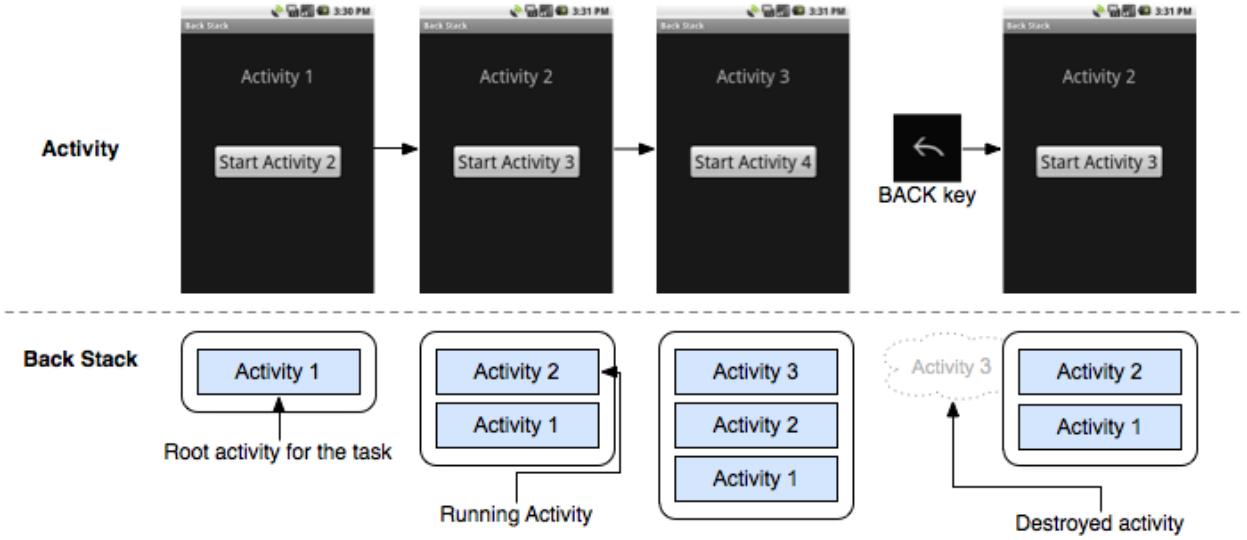
Task – группа из нескольких Activity, с помощью которых пользователь выполняет определенную операцию. Обычно стартовая позиция для создания Task – это экран Домой (Home).

Находясь в Home вы вызываете какое-либо приложение из списка приложений или через ярлык. Создается Task. И Activity приложения (которое отмечено как MAIN в манифест-файле) помещается в этот Task как корневое. Task выходит на передний фон. Если же при вызове приложения, система обнаружила, что в фоне уже существует Task, соответствующий этому приложению, то она выведет его на передний план и создавать ничего не будет.

Когда Activity_A вызывает Activity_B, то Activity_B помещается на верх (в топ) Task и получает фокус. Activity_A остается в Task, но находится в состоянии Stopped (его не видно и оно не в фокусе). Далее, если пользователь жмет Back находясь в Activity_B, то Activity_B удаляется из Task и уничтожается. А Activity_A оказывается теперь на верху Task и получает фокус.

В каком порядке открывались (добавлялись в Task) Activity, в таком порядке они и содержатся в Task. Они никак специально не сортируются и не упорядочиваются внутри. Набор Activity в Task еще называют back stack. Я буду называть его просто - стэк.

Схема (с офиц.сайта) демонстрирует пример:



В верхней части то, что видит пользователь. В нижней – содержимое Task. Видно, как при вызове новых Activity они добавляются в верх стэка. А если нажата кнопка Назад, то верхнее Activity из стэка удаляется и отображается предыдущее Activity.

Допустим у нас есть Task с несколькими Activity. Он на переднем фоне, мы с ним работаем сейчас.

- если мы нажмем кнопку Home, то ничего не будет удалено, все Activity сохранятся в этом Task-е, а сам Task просто уйдет на задний фон и его всегда можно будет вызвать оттуда, снова вызвав приложение, Activity которого является корневым для Task-а. Либо можно удерживать кнопку Home и мы увидим как раз список Task-ов, которые расположены на заднем фоне.

- если же в активном Task-е несколько раз нажимать кнопку Назад, то в итоге в стэке не останется Activity, пустой Task будет удален и пользователь увидит экран Home.

Там еще как всегда куча нюансов и сложностей, но мы пока остановимся на этом и в дебри не полезем. Этих знаний вполне хватит, чтобы ответить на вопросы предыдущего урока: *почему на шаге 2 MainActivity исчезло с экрана, но осталось висеть в памяти и не было уничтожено? Ведь на шаге 3 было уничтожено ActivityTwo после того, как оно пропало с экрана. А на шаге 4 было в итоге уничтожено и MainActivity. Почему шаг 2 стал исключением?*

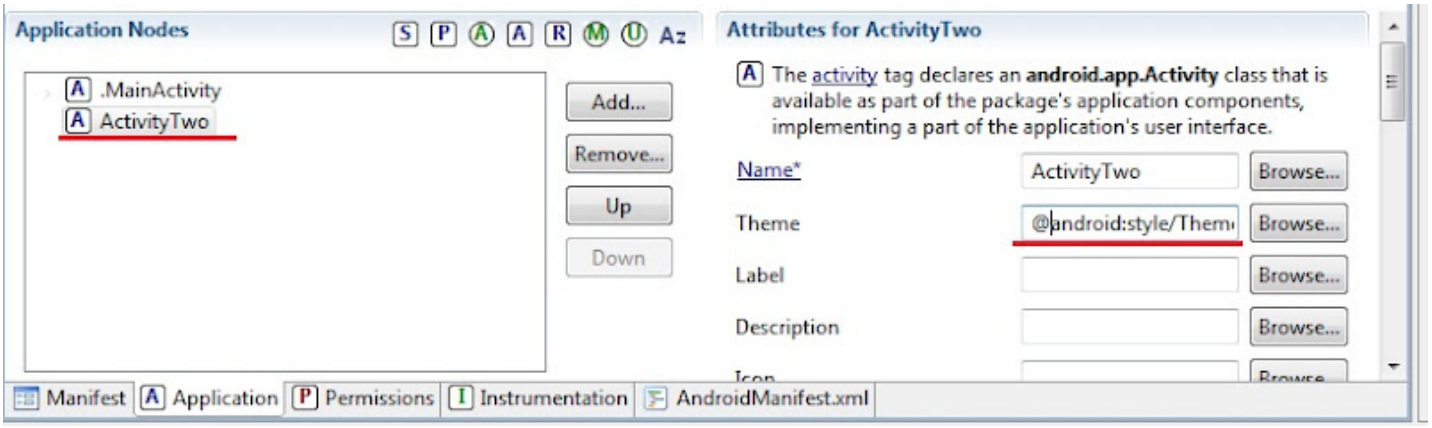
Теперь вы знаете, почему. Потому, что на шаге 2 MainActivity осталось в стэке, а ActivityTwo вставилось на верх стэка и получило фокус. Ну а на шаге 3 и 4 были удалены Activity из верха стэка, в Task не осталось Activity, и мы увидели экран Home.

Если бы мы на шаге 3 нажали не Back, а Home, то Task с обоими Activity ушел бы задний фон и ничего не было бы уничтожено.

Paused

Теперь давайте откроем проект с прошлого урока P0241_TwoActivityState. Мы хотели поймать состояние Paused для Activity. Это состояние означает, что Activity не в фокусе, но оно видно, пусть и частично. Мы можем этого добиться, если присвоим диалоговый стиль для ActivityTwo. Оно отобразится как всплывающее окно и под ним будет частично видно MainActivity – оно и будет в статусе Paused. Давайте реализуем.

Для этого открываем AndroidManifest.xm, вкладка Application, находим там ActivityTwo и справа в поле Theme пишем такой текст: *@android:style/Theme.Dialog*



Все сохраняем и запускаем приложение.

Появилось MainActivity

Логи:

MainActivity: onCreate()
MainActivity: onStart()
MainActivity: onResume()

Все верно.

Вызываем ActivityTwo.

Логи:

MainActivity: onPause()
ActivityTwo: onCreate()
ActivityTwo: onStart()
ActivityTwo: onResume()

Видим, что не был вызван метод onStop для MainActivity, а значит приложение не было переведено в состояние Stopped и находится в режиме Paused.

Нажмем Back.

Логи:

ActivityTwo: onPause()
MainActivity: onResume()
ActivityTwo: onStop()
ActivityTwo: onDestroy()

MainActivity восстановилось одним лишь вызовом onResume, а onStart не понадобился, т.к. оно было в состоянии Paused, а не Stopped.

Мы четко увидели разницу между этим примером и им же на прошлом уроке. И MainActivity у нас был в состоянии Paused.

Далее можно нажать Back, а можно Home - вы уже знаете, что произойдет в обоих случаях. По логам можно убедиться в этом.

Чтобы вернуть ActivityTwo нормальный режим отображения, зайдите снова в манифест и удалите строку из поля Theme.

Кстати, у вас уже вполне достаточно знаний, чтобы создать приложение с кучей Activity, прописать вызовы и поиграться, посмотреть логи. Тем самым закрепите темы LifeCycle и Task.

На следующем уроке:

- вызываем Activity используя неявный вызов и Intent Filter

Урок 26. Intent Filter - практика

В этом уроке:

- вызываем Activity, используя неявный вызов и Intent Filter

Последние уроки получились перегруженными теорией. Эту теорию необходимо понять, чтобы не было проблем с практикой. Эти темы являются основой – Task, Lifecycle, Intent. Если что-либо осталось непонятно, то вы всегда можете снова открыть и перечитать материал. А далее мы будем реализовывать примеры, которые будут эту теорию подтверждать и все станет понятнее.

На прошлых уроках мы научились **вызывать Activity** с помощью **Intent** и явного указания **класса**. Также мы знаем, что есть и другой способ вызова Activity – **неявный**. Он основан на том, что Activity вызывается не по имени, а по функционалу. Т.е. мы хотим выполнить определенные действия, создаем и настраиваем соответствующий **Intent** и отправляем его **искать** те Activity, которые могли бы справиться с нашей задачей.

Давайте посмотрим, как это реализуется на практике. Мы создадим приложение, которое будет отображать нам текущее время или дату. Сделаем мы это с помощью трех Activity:
- первое будет содержать две кнопки: Show time и Show date
- второе будет отображать время
- третье будет отображать дату

Нажатие на кнопку Show time будет вызывать второе Activity, а нажатие на кнопку Show date – третье Activity. Но **реализуем** мы это не через прямое указание классов Activity в Intent, а через **Intent Filter**.

Создадим проект:

Project name: P0261_IntentFilter
Build Target: Android 2.3.3
Application name: IntentFilter
Package name: ru.startandroid.develop.p0261intentfilter
Create Activity: MainActivity

Открываем **main.xml** и рисуем две кнопки:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="horizontal">
    <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:id="@+id/btnTime"
         android:text="Show time">
    </Button>
    <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:id="@+id/btnDate"
         android:text="Show date">
    </Button>
</LinearLayout>
```

Пишем реализацию **MainActivity.java**:

```
package ru.startandroid.develop.p0261intentfilter;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button btnTime = (Button) findViewById(R.id.btnTime);
        Button btnDate = (Button) findViewById(R.id.btnDate);

        btnTime.setOnClickListener(this);
        btnDate.setOnClickListener(this);
    }

  @Override
  public void onClick(View v) {
```

```
    Intent intent;

    switch(v.getId()) {
    case R.id.btnTime:
      intent = new Intent("ru.startandroid.intent.action.showtime");
      startActivity(intent);
      break;
    case R.id.btnDate:
      intent = new Intent("ru.startandroid.intent.action.showdate");
      startActivity(intent);
      break;
    }
  }
}
```

В коде мы определили кнопки и присвоили им **Activity** как **обработчик** нажатий. В методе **onCilck** мы определяем какая кнопка была нажата и **создаем Intent**.

Для создания Intent используем конструктор: Intent (String action). Т.е. мы при создании заполняем атрибут объекта Intent, который называется **action**. Это обычная строковая константа. Action обычно указывает **действие**, которое мы хотим произвести. Например, есть следующие системные action-константы: ACTION_VIEW - просмотр, ACTION_EDIT – редактирование, ACTION_PICK – выбор из списка, ACTION_DIAL – сделать звонок.

Если действие производится с чем-либо, то в пару к **action** идет еще один Intent-атрибут – **data**. В нем мы можем указать какой-либо объект: пользователь в адресной книге, координаты на карте, номер телефона и т.п. Т.е. **action** указывает **что делать**, а **data** – **с чем делать**.

Про **data** мы еще поговорим на следующих уроках, а пока будем использовать только **action**. Выше я уже перечислил некоторые системные action-константы, но мы можем использовать и свой action.

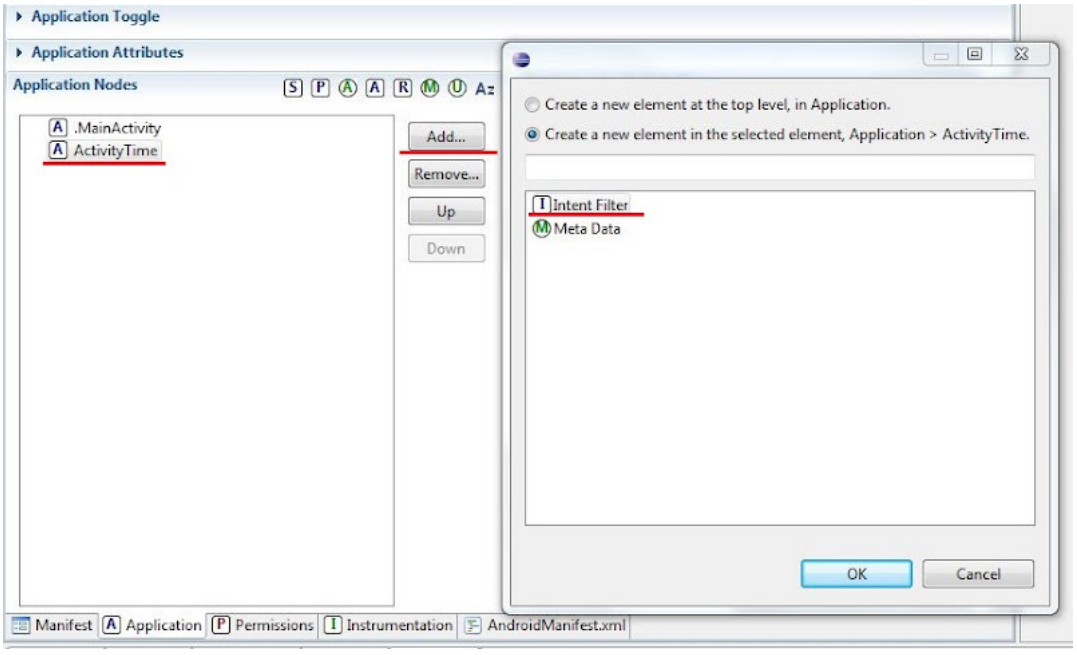
Как вы видите из кода, я придумал и использую такие action:
ru.startandroid.intent.action.showtime
ru.startandroid.intent.action.showdate

Первый будет означать, что я хочу вызвать Activity, которое мне покажет текущее время. Второй – Activity с датой.

Здесь надо четко понимать следующее: **action** – это просто **текст**. И я мог с таким же успехом придумать action *abcdefg123456*. Но текст *showtime* – отражает то, что я хочу сделать, он нагляднее и понятнее. А префикс *ru.startandroid.intent.action* я использую, чтобы не было коллизий. В системе может быть приложение, которое уже использует action *showtime* - я не хочу с ним пересекаться. Поэтому мой action – это *ru.startandroid.intent.action.showtime*.

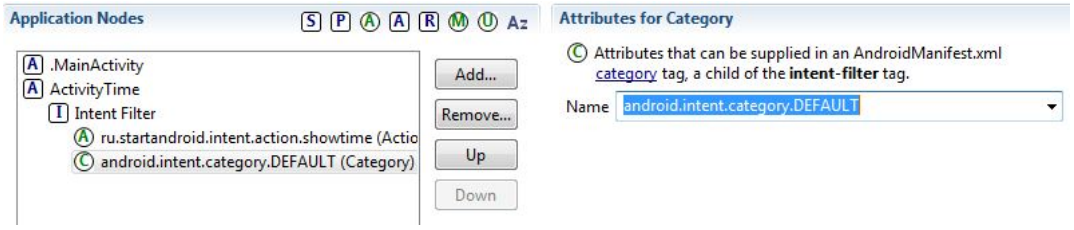
Итак, мы создали **Intent** с **action** и запустили его в систему искать Activity. Чтобы Activity подошла, надо чтобы ее **Intent Filter** содержал атрибут **action** с тем же **значением**, что и **action** в **Intent**. Значит нам осталось создать две **Activity**, настроить их **Intent Filter** и реализовать отображение времени и даты.

Activity создается как обычно - создаем класс **ActivityTime** с суперклассом **android.app.Activity** и прописываем его в манифесте как Activity. После того, как прописали в манифесте, надо будет там же создать Intent Filter. Для этого выделяем **ActivityTime**, жмем **Add**, выбираем **Intent Filter** и жмем **OK**.



Далее в Intent Filter аналогично создаем **Action** и в поле **Name** прописываем *ru.startandroid.intent.action.showtime*

Также в **Intent Filter** надо создать **Category** и в поле **name** выбрать из списка *android.intent.category.DEFAULT*. Пока не будем разбираться детально зачем она нужна. Но без этого вызов **startActivity(Intent)** не найдет Activity.



Создадим layout для нового Activity, назовем его **time.xml**:

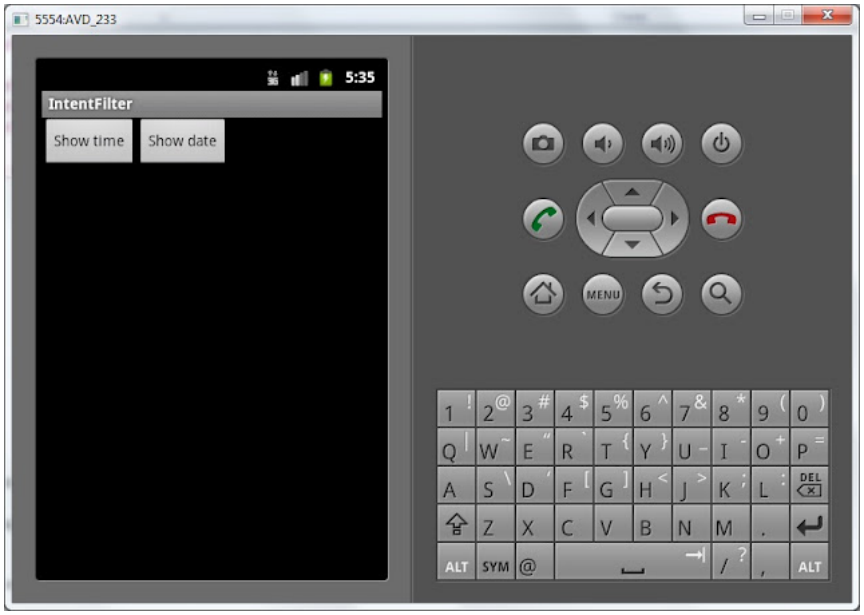
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:id="@+id/tvTime"
         android:text="TextView"
         android:layout_gravity="center_horizontal"
         android:layout_marginTop="20dp"
         android:textSize="30sp">
    </TextView>
</LinearLayout>
```

Пишем код в **ActivityTime.java**:

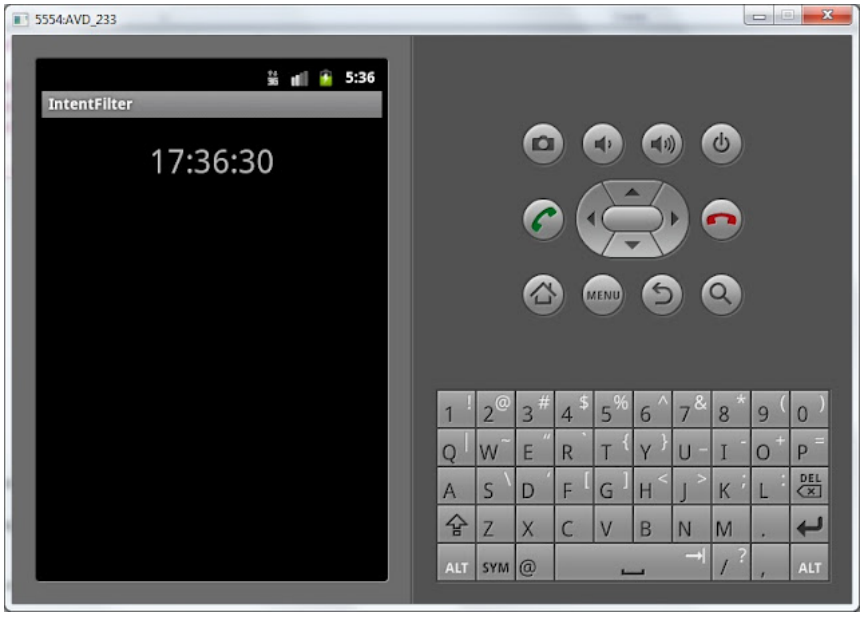
```
package ru.startandroid.develop.p0261intentfilter;

import java.sql.Date;
import java.text.SimpleDateFormat;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class ActivityTime extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.time);

    SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
    String time = sdf.format(new Date(System.currentTimeMillis()));

    TextView tvTime = (TextView) findViewById(R.id.tvTime);
    tvTime.setText(time);
  }
}
```

Тут все просто - вычисляем текущее время и показываем его в TextView.

Все сохраним и запустим приложение.



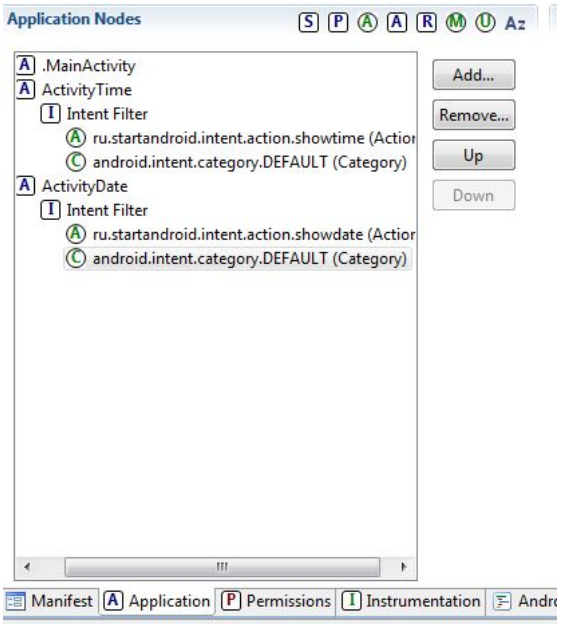
Жмем кнопку Show time:



отобразилось время. Т.е. **Intent** с **action** = *ru.startandroid.intent.action.showtime* нашел и отобразил **Activity**, у которого **action** также равен *ru.startandroid.intent.action.showtime* в **Intent Filter**.

Вернемся назад (кнопка **Back**) и нажмем теперь кнопку **Show date**. Приложение выдаст ошибку, т.к. оно не смогло найти Activity, которое соответствовало бы Intent с **action** = *ru.startandroid.intent.action.showdate* (мы создали только для showtime).

Давайте создадим такое Activity, назовем его **ActivityDate**. Действия все те же самые, что и при создании ActivityTime:
- создание класса
- создание Activity в манифесте и создание для него Intent Filter (с action = *ru.startandroid.intent.action.showdate* и category = *android.intent.category.DEFAULT*)



Layout-файл назовем **date.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:id="@+id/tvDate"
         android:text="TextView"
         android:layout_gravity="center_horizontal"
         android:layout_marginTop="20dp"
         android:textSize="30sp">
    </TextView>
</LinearLayout>
```

Код **ActivityDate.java**:

```
package ru.startandroid.develop.p0261intentfilter;

import java.sql.Date;
import java.text.SimpleDateFormat;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class ActivityDate extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.date);

    SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy");
    String date = sdf.format(new Date(System.currentTimeMillis()));

    TextView tvDate = (TextView) findViewById(R.id.tvDate);
    tvDate.setText(date);
  }
}
```

Все сохраняем, запускаем приложение, жмем **Show date** и видим дату. Это значит, что **Intent** с **action** = *ru.startandroid.intent.action.showdate* нашел и отобразил **ActivityDate** подходящее ему по **Intent Filter**.

Чтобы закрепить тему, проделаем еще кое-то. Если помните, в уроке №22, где я рассказывал про Intent Filter, я говорил, что **один Intent** может найти **несколько** подходящих **Activity**. В этом случае пользователю предоставляется **выбор**, какое Activity использовать. Давайте сами спровоцируем такой случай. Мы сделаем еще одно Activity, которое будет реагировать на Intent с action = *ru.startandroid.intent.action.showdate*. И будет отображать текущую дату аналогично ActivityDate. Но формат отображения даты будет немного другой.

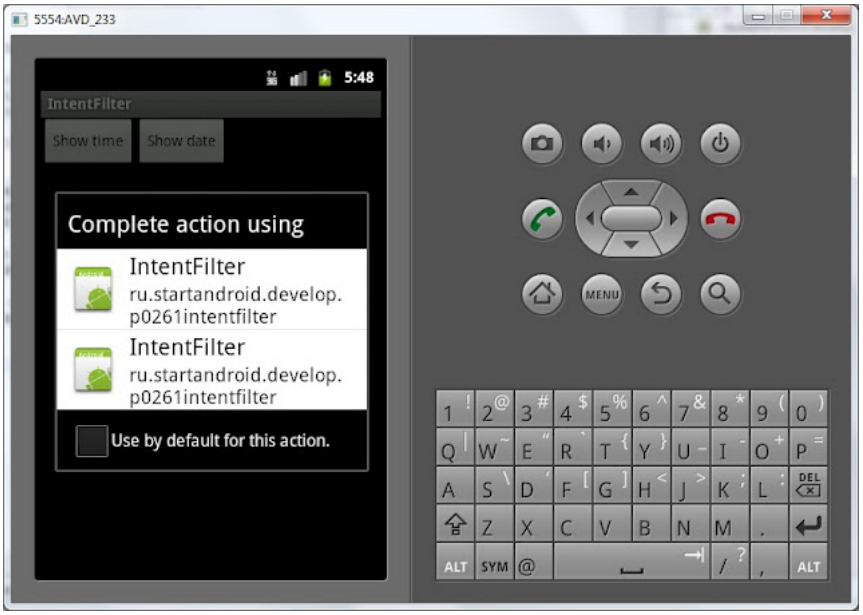
Давайте создадим такое Activity, назовем его **ActivityDateEx**. Действия все те же самые, что и при создании ActivityDate:
- создание класса
- создание Activity в манифесте и создание для него Intent Filter (с action = *ru.startandroid.intent.action.showdate* и category = *android.intent.category.DEFAULT*)

Новый layout-файл создавать не будем, используем уже существующий **date.xml**. В принципе, все три Activity у нас могли использовать один layout, т.к. они совершенно одинаковы – один TextView.

Код **ActivityDateEx.java**:

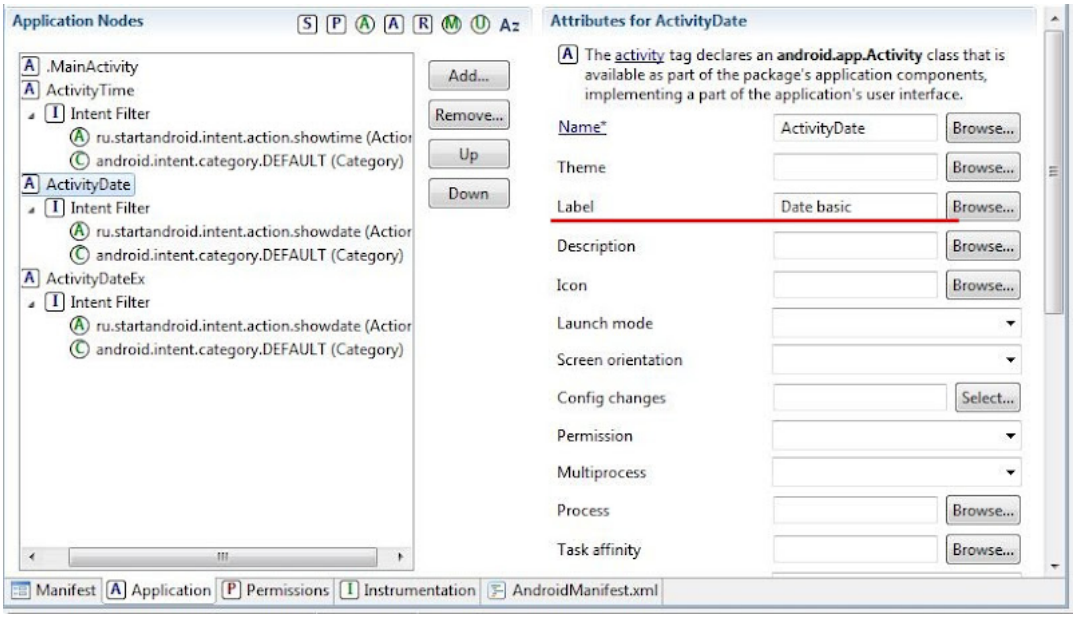
```
package ru.startandroid.develop.p0261intentfilter;

import java.sql.Date;
import java.text.SimpleDateFormat;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class ActivityDateEx extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.date);

    SimpleDateFormat sdf = new SimpleDateFormat("EEE, MMM d, yyyy");
    String date = sdf.format(new Date(System.currentTimeMillis()));

    TextView tvDate = (TextView) findViewById(R.id.tvDate);
    tvDate.setText(date);
  }
}
```

Как видим, отличие от ActivityDate только в формате даты. Сохраним все и запустим. Жмем Show date и видим такой выбор:



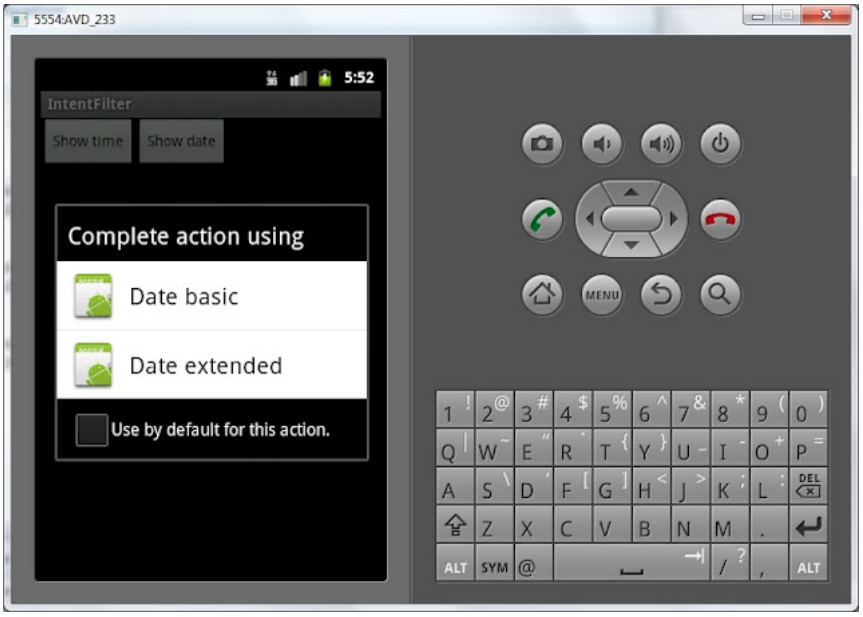
Intent нашел **два Activity**, но показал для каждого из них название родительского приложения и package. В нашем случае – оба Activity из нашего приложения, поэтому текст одинаков и не разберешь, какое из них какое. Давайте пофиксим это, прописав нормальные имена.

Нажмите Back, чтобы закрыть диалог выбора. Идем в **манифест** и для **Activity** пропишем **label**:



Date basic для **ActivityDate**
Date extended для **ActivityDateEx**

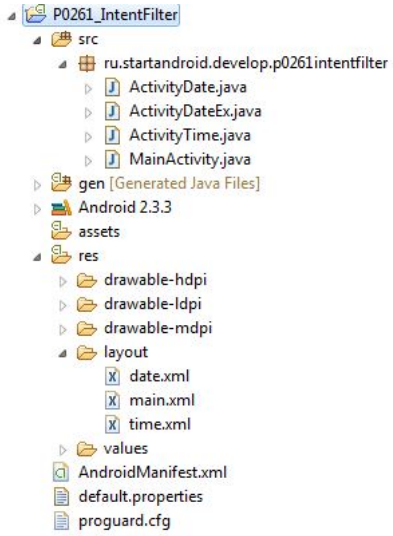
Сохраняем и запускаем. Жмем Show date и видим такой выбор:



Так значительно лучше. Жмем Date extended и видим дату в расширенном формате на ActivityDateEx.

Итак, мы создавали и посылали **Intent** с **action**. Этот **Intent** находил **Activity** с подходящим **Intent Filter** и отображал его. Если находил **несколько** – давал **выбор**. Примеры отлично показывают механизм.

Если запутались, чего и где создавать, привожу скрин проекта и содержимое манифеста.



Содержимое манифеста (вкладка AndroidManifest.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<manifest  xmlns:android="http://schemas.android.com/apk/res/android"  package="ru.startandroid.develop.p0261intentfilter"  android:versionCode="1"  an
    <uses-sdk  android:minSdkVersion="10"></uses-sdk>
    <application  android:icon="@drawable/icon"  android:label="@string/app_name">
        <activity  android:name=".MainActivity"  android:label="@string/app_name">
            <intent-filter>
                <action  android:name="android.intent.action.MAIN"></action>
                <category  android:name="android.intent.category.LAUNCHER"></category>
            </intent-filter>
        </activity>
        <activity  android:name="ActivityTime">
            <intent-filter>
                <action  android:name="ru.startandroid.intent.action.showtime"></action>
                <category  android:name="android.intent.category.DEFAULT"></category>
            </intent-filter>
        </activity>
        <activity  android:name="ActivityDate"  android:label="Date basic">
            <intent-filter>
                <action  android:name="ru.startandroid.intent.action.showdate"></action>
                <category  android:name="android.intent.category.DEFAULT"></category>
            </intent-filter>
        </activity>
        <activity  android:name="ActivityDateEx"  android:label="Date extended">
            <intent-filter>
                <action  android:name="ru.startandroid.intent.action.showdate"></action>
                <category  android:name="android.intent.category.DEFAULT"></category>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

На следующем уроке:

- читаем action из Intent

Урок 27. Читаем action из Intent

В этом уроке:

- читаем action из Intent

На прошлом уроке мы сделали наглядный пример, показывающий, как связаны между собой **action**, **Intent** и **Intent Filter**. На этом уроке продолжим тему. Есть две новости: хорошая и хорошая)

Первая новость. **Intent Filter** может содержать в себе несколько **action**. Тем самым Activity дает понять, что она способна на **несколько действий**. Например, не только просмотр картинки, но и редактирование. Получается, что Activity может подойти разным Intent с разными action.

Вторая новость. **Activity**, которое было вызвано с помощью Intent, имеет **доступ** к этому **Intent** и может **прочесть** его **атрибуты**. Т.е. может узнать какой action использовался.

Мы сделаем следующее: создадим **Activity** и настроим **Intent Filter** на **action** = *ru.startandroid.intent.action.showtime* и **action** = *ru.startandroid.intent.action.showdate*. Тем самым мы обозначаем, что это Activity способно и время показать и дату. Далее мы будем создавать Intent либо с action = *ru.startandroid.intent.action.showtime*, либо с *ru.startandroid.intent.action.showdate*. Они **оба** будут вызывать **одно Activity**. А чтобы **Activity** знало показывать ему дату или время, мы будем **читать action** из **Intent** и по нему определять.

В общем, сейчас начнем делать и все станет понятно)

Создадим проект:

Project name: P0271_GetIntentAction
Build Target: Android 2.3.3
Application name: GetIntentAction
Package name: ru.startandroid.develop.p0271getintentaction
Create Activity: MainActivity

Открываем **main.xml** и рисуем две кнопки:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="horizontal">
    <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:id="@+id/btnTime"
          android:text="Show time">
    </Button>
    <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:id="@+id/btnDate"
          android:text="Show date">
    </Button>
</LinearLayout>
```

Экран точно такой же как и в прошлом уроке.

Код для **MainActivity.java**:

```
package ru.startandroid.develop.p0271getintentaction;
import android.app.Activity;
import android.content.Intent;
```

```
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button btnTime = (Button) findViewById(R.id.btnTime);
        Button btnDate = (Button) findViewById(R.id.btnDate);

        btnTime.setOnClickListener(this);
        btnDate.setOnClickListener(this);
    }

  @Override
  public void onClick(View v) {
    Intent intent;

    switch(v.getId()) {
    case R.id.btnTime:
      intent = new Intent("ru.startandroid.intent.action.showtime");
      startActivity(intent);
      break;
    case R.id.btnDate:
      intent = new Intent("ru.startandroid.intent.action.showdate");
      startActivity(intent);
      break;
    }
  }
}
```

Код тоже полностью из прошлого урока. Определяем кнопки, присваиваем **обработчик** – **Activity**, и вызываем **Intent** по нажатиям. Теперь мы сделаем Activity, которая будет ловить оба этих Intent.

Для начала создадим layout-файл **info.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
    <TextView
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:id="@+id/tvInfo"
          android:text="TextView"
          android:layout_gravity="center_horizontal"
          android:layout_marginTop="20dp"
          android:textSize="30sp">
    </TextView>
</LinearLayout>
```

На экране у нас один TextView.

Создаем **Activity**, назовем его просто **Info**.

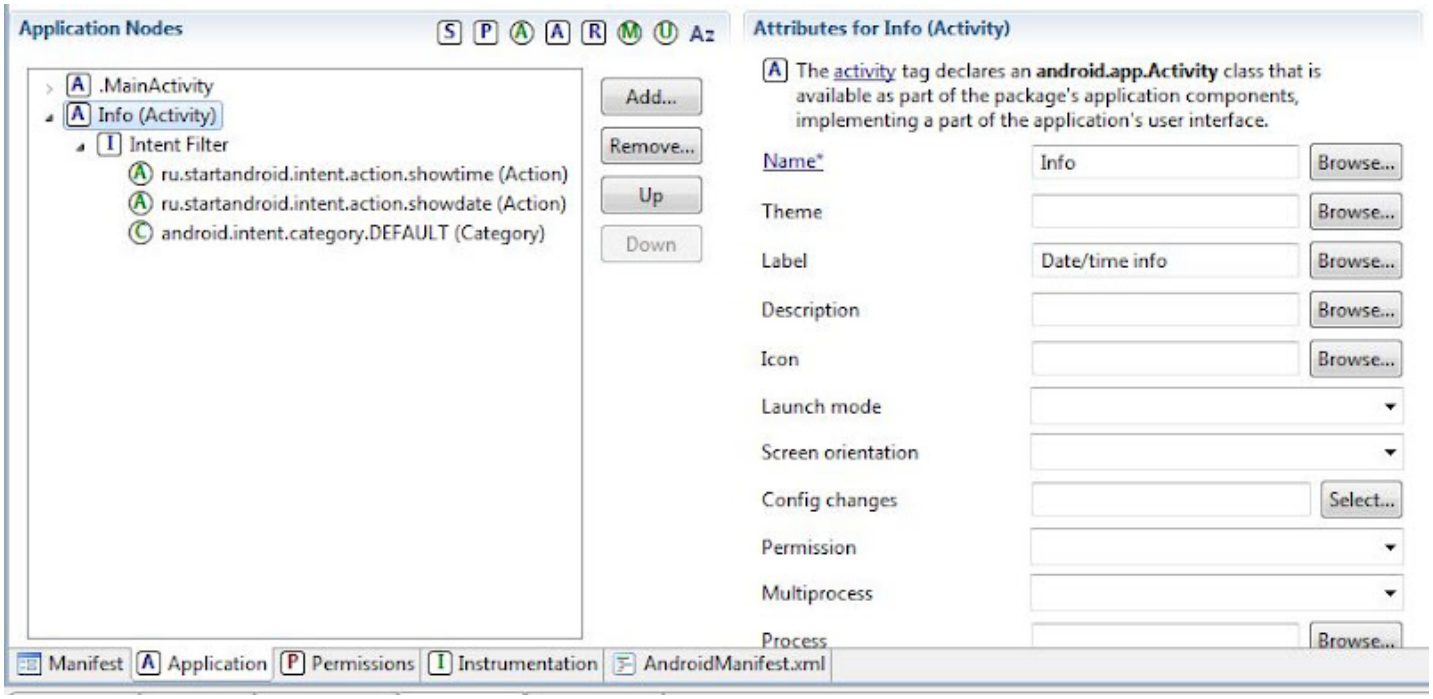
Код **Info.java**:

```
package ru.startandroid.develop.p0271getintentaction;
```

```
import java.sql.Date;
import java.text.SimpleDateFormat;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class Info extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.info);

    // получаем Intent, который вызывал это Activity
    Intent intent = getIntent();
    // читаем из него action
    String action = intent.getAction();

    String format = "", textInfo = "";

    // в зависимости от action заполняем переменные
    if (action.equals("ru.startandroid.intent.action.showtime")) {
      format = "HH:mm:ss";
      textInfo = "Time: ";
    }
    else if (action.equals("ru.startandroid.intent.action.showdate")) {
      format = "dd.MM.yyyy";
      textInfo = "Date: ";
    }

    // в зависимости от содержимого переменной format
    // получаем дату или время в переменную datetime
    SimpleDateFormat sdf = new SimpleDateFormat(format);
    String datetime = sdf.format(new Date(System.currentTimeMillis()));

    TextView tvDate = (TextView) findViewById(R.id.tvInfo);
    tvDate.setText(textInfo + datetime);
  }
}
```

Мы получаем **Intent** с помощью метода getIntent(), читаем из него **action** и в зависимости от значения формируем и выводим на экран текст.

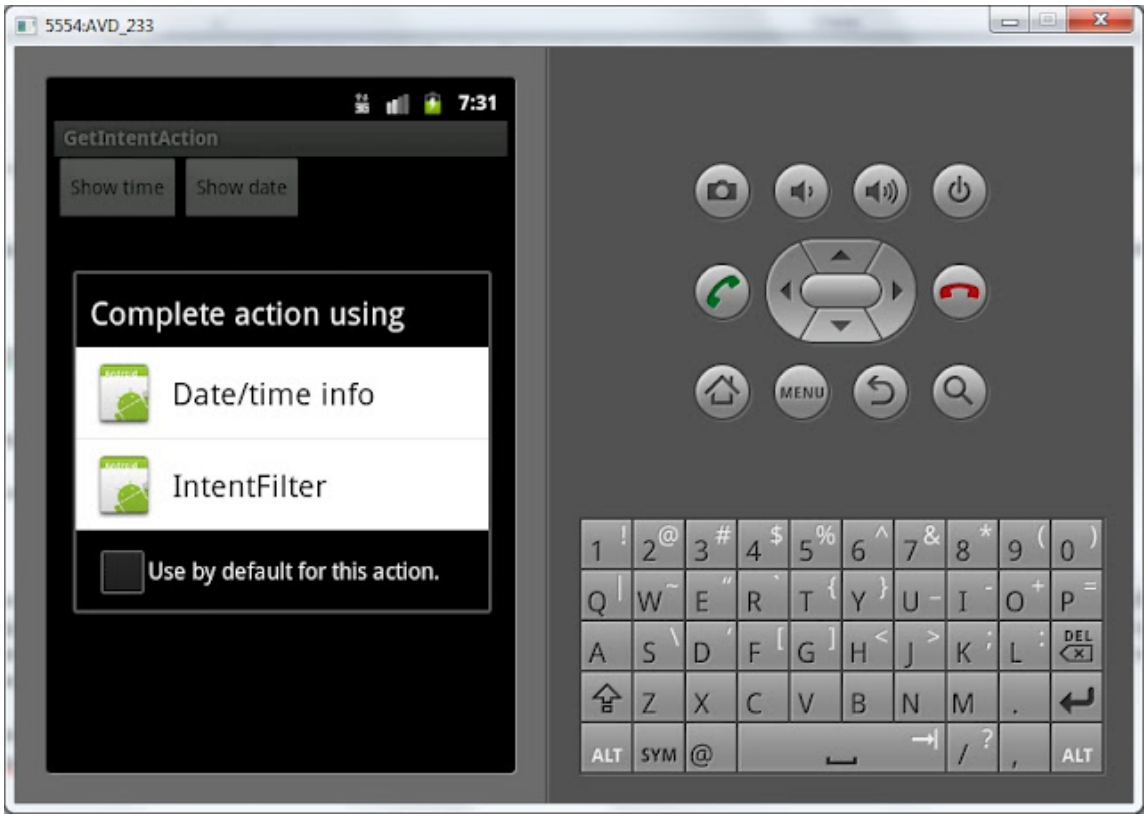
Не забываем прописать новое Activity в **манифесте** и создать ему **Intent Filter** с двумя **action** и **category**. И **label** укажите Date/time info.



Intent Filter для Info содержит **два action**. А значит если придет **Intent** с **любым** из них – то **Activity** будет **вызвана**.

Все сохраним и запустим.

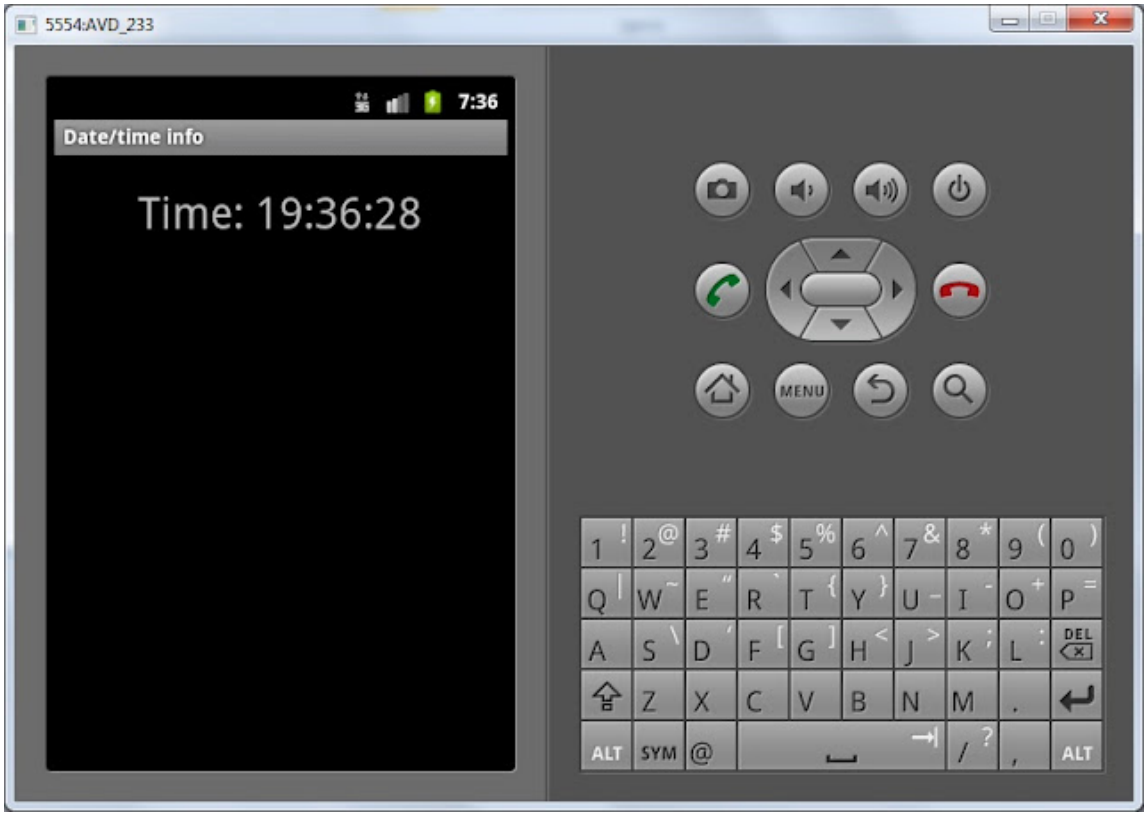
Жмем кнопку Show time. Скорее всего вы увидите следующее:



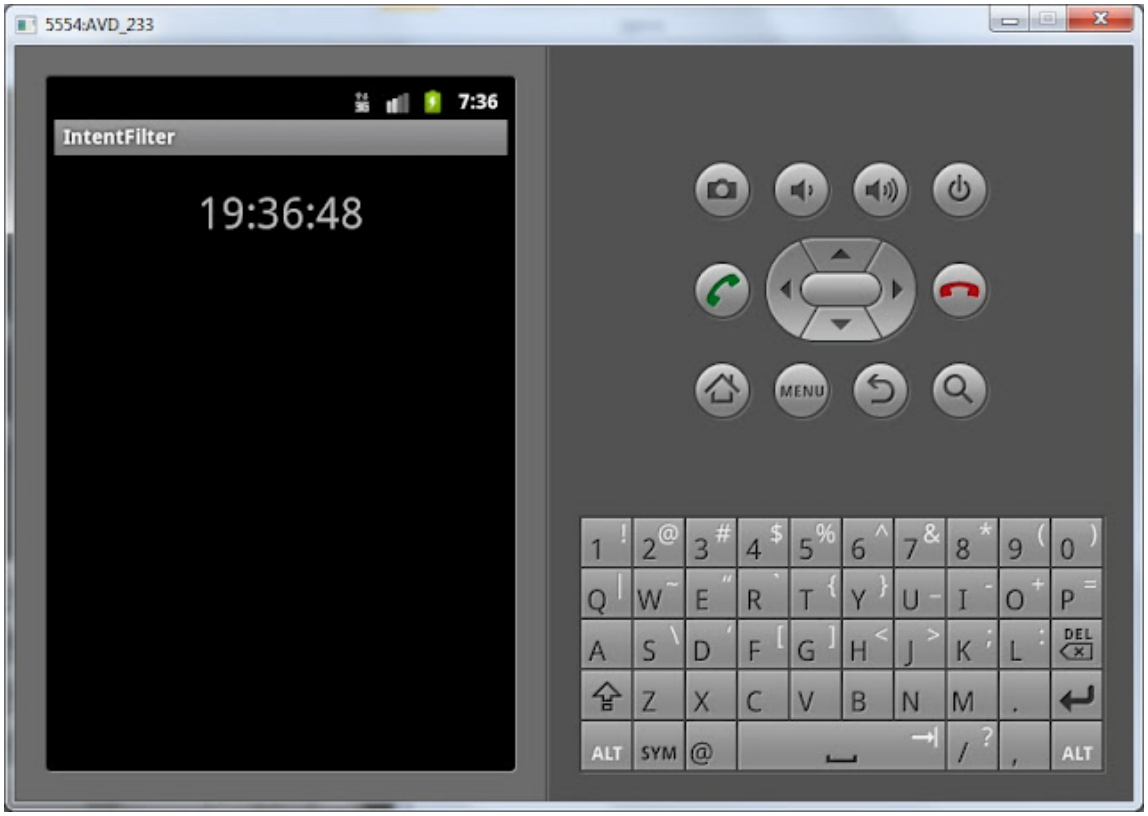
Система предлагает нам выбор. Т.е. **Intent** с action = *ru.startandroid.intent.action.showtime* нашел **два** подходящих **Activity**. То, которое *Date/time info* мы создали только что, тут все понятно. Вопрос – откуда второе с названием *IntentFilter*. Ответ – это ActivityTime из прошлого урока. Называется оно *IntentFilter* потому, что на прошлом уроке мы не прописали в манифесте **label** для этого Activity и система по умолчанию отображает название приложения.

Если же система не отобразила диалог выбора, значит вы, либо не создавали приложение с прошлого урока, либо пересоздали AVD, либо где-то очепятка в коде.

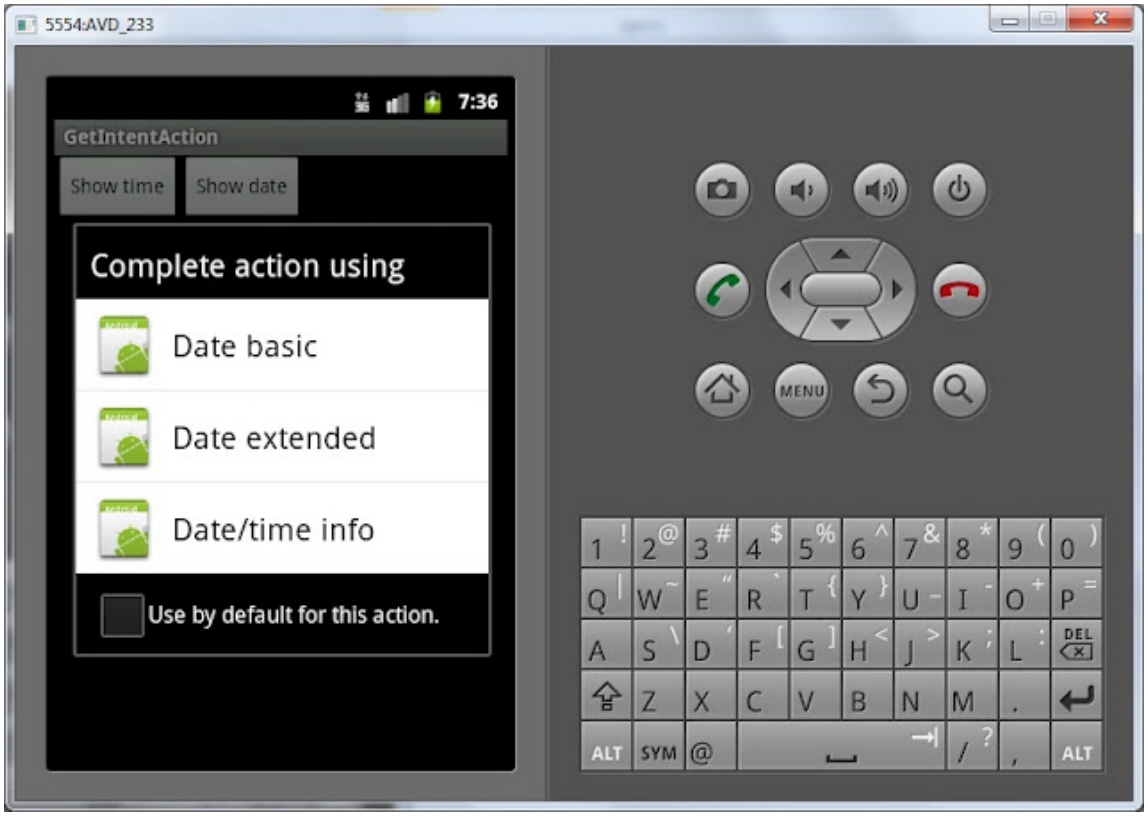
Выбираем *Date/time info* и видим то, что только что кодили. **Activity** определило, что **Intent** был с **action** = *ru.startandroid.intent.action.showtime* и показало время с текстом *Time:*



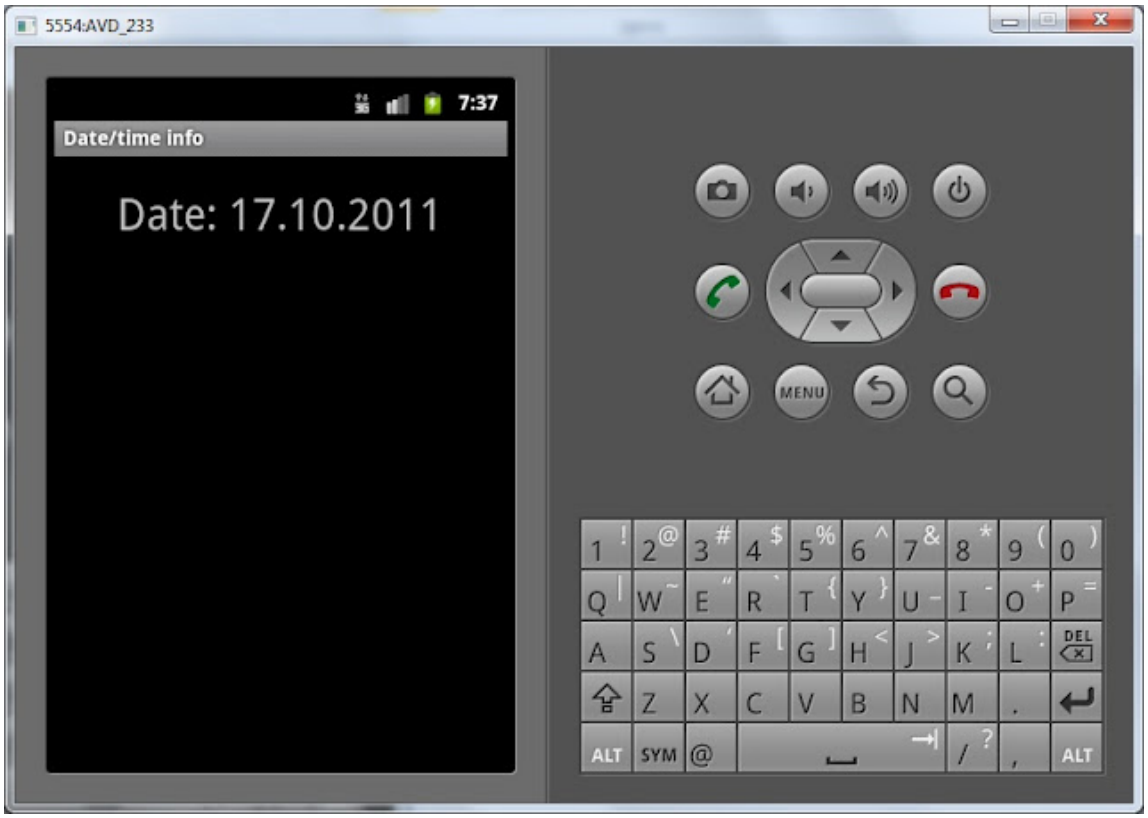
Если же выбрать *IntentFilter* увидим Activity с прошлого урока:



Теперь попробуем нажать кнопку Show date: видим такой выбор:

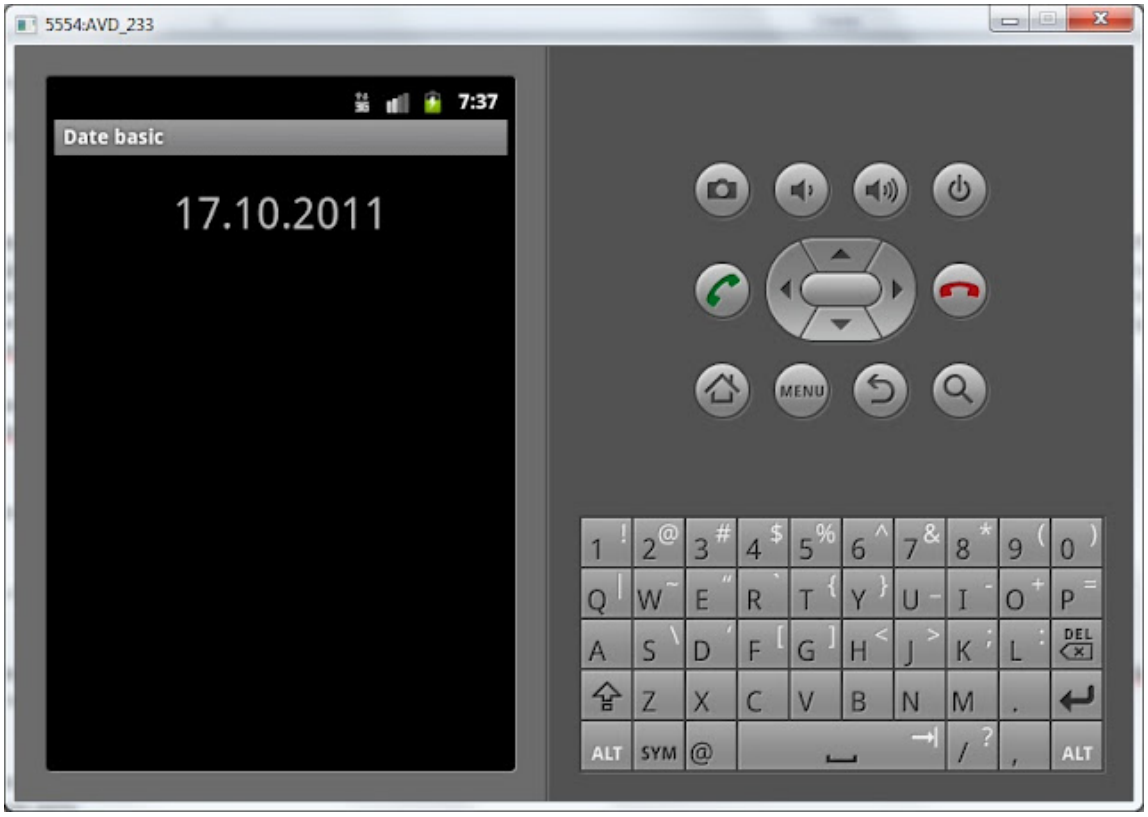


Снова видим наше *Date/time info* и **два Activity** с **прошлого** урока. Они все содержат **action** = *ru.startandroid.intent.action.showdate* в **Intent Filter** и нам надо выбирать. Выберем *Date/time info* и видим дату с текстом *Date:*

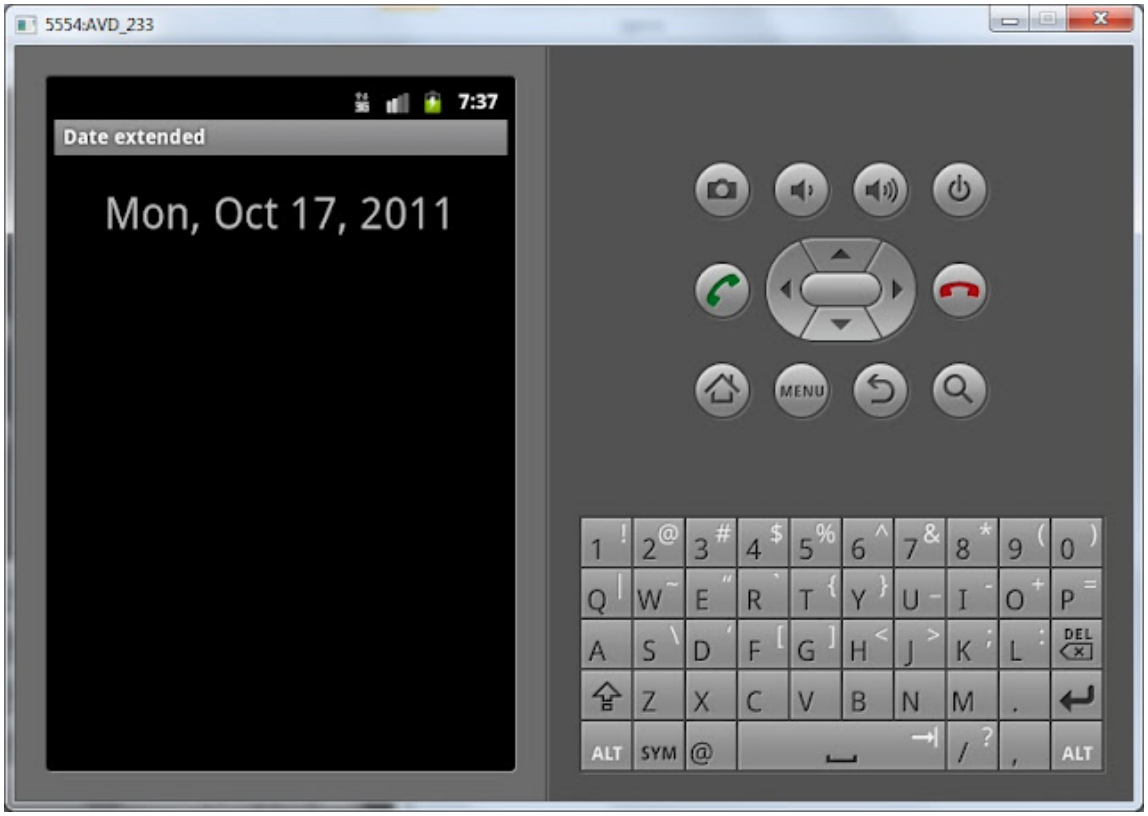


Если же выбирать *Date basic* или *Date extended* увидим то, что делали на прошлом уроке :

обычную дату



и расширенную



Мы увидели, что **одно Activity** может быть вызвано с помощью **Intent** с **разными action**. **Activity** может прочесть **action** и выполнить необходимые действия.

Также мы убедились, что **Intent** ищет **Activity** по **всем приложениям** в системе. В этот раз он нашел **Activity** из приложения, которое мы делали на прошлом уроке.

На следующем уроке:

- передаем данные с помощью Intent

Урок 28. Extras - передаем данные с помощью Intent

В этом уроке:

- передаем данные с помощью Intent

На прошлых урокам мы узнали, что такое **Intent** и как им пользоваться. Из одного Activity мы просто вызывали другое, передавая **action**. Теперь научимся передавать **данные**. Сделаем простейшее приложение. На первом экране мы будем вводить наше имя и фамилию, а второй экран будет эти данные отображать. Передавать **данные** будем **внутри Intent**.

Создадим проект:

Project name: P0281_IntentExtras
Build Target: Android 2.3.3
Application name: IntentExtras
Package name: ru.startandroid.develop.p0281intentextras
Create Activity: MainActivity

Открываем **main.xml** и рисуем экран с полями и кнопкой отправки:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
           android:layout_width="fill_parent"
           android:layout_height="wrap_content"
           android:layout_marginTop="10dp"
           android:gravity="center_horizontal"
           android:text="Input your Name">
     </TextView>
     <TableLayout
           android:id="@+id/tableLayout1"
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:layout_margin="10dp"
           android:stretchColumns="1">
          <TableRow
               android:id="@+id/tableRow1"
               android:layout_width="match_parent"
               android:layout_height="wrap_content">
              <TextView
                   android:id="@+id/textView1"
                   android:layout_width="match_parent"
                   android:layout_height="wrap_content"
                   android:text="First Name">
              </TextView>
```

```
            <EditText
                 android:id="@+id/etFName"
                 android:layout_width="match_parent"
                 android:layout_height="wrap_content"
                 android:layout_marginLeft="5dp">
                <requestFocus>
                </requestFocus>
            </EditText>
        </TableRow>
        <TableRow
             android:id="@+id/tableRow2"
             android:layout_width="match_parent"
             android:layout_height="wrap_content">
            <TextView
                 android:id="@+id/textView2"
                 android:layout_width="match_parent"
                 android:layout_height="wrap_content"
                 android:text="Last Name">
            </TextView>
            <EditText
                 android:id="@+id/etLName"
                 android:layout_width="match_parent"
                 android:layout_height="wrap_content"
                 android:layout_marginLeft="5dp">
            </EditText>
        </TableRow>
    </TableLayout>
    <Button
         android:id="@+id/btnSubmit"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center_horizontal"
         android:text="Submit">
    </Button>
</LinearLayout>
```

В **EditText** будем вводить имя и фамилию, а кнопка **Submit** будет вызывать другой экран и передавать ему эти данные.

Пишем код для **MainActivity.java**:

```
package ru.startandroid.develop.p0281intentextras;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity implements OnClickListener {
```

```
  EditText etFName;
  EditText etLName;

  Button btnSubmit;


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        etFName = (EditText) findViewById(R.id.etFName);
        etLName = (EditText) findViewById(R.id.etLName);

        btnSubmit = (Button) findViewById(R.id.btnSubmit);
        btnSubmit.setOnClickListener(this);

    }


  @Override
  public void onClick(View v) {
    Intent intent = new Intent(this, ViewActivity.class);
    intent.putExtra("fname", etFName.getText().toString());
    intent.putExtra("lname", etLName.getText().toString());
    startActivity(intent);
  }
}
```

Определяем поля ввода и кнопку. Кнопке присваиваем обработчик – Activity (this). Рассмотрим реализацию метода **onClick**. Мы **создаем Intent** с использованием класса, а не action. Если помните, с такого способа мы начинали знакомство с Intent. Напомню - это означает, что система просмотрит манифест файл нашего приложения, и если найдет Activity с таким классом – отобразит его. ViewActivity пока не создан, поэтому код будет подчеркнут красным. Это не мешает нам сохранить файл. Чуть позже мы создадим это Activity и ошибка исчезнет.

Итак, Intent создан, смотрим код дальше. Используется метод putExtra. Он имеет множество вариаций и аналогичен методу **put** для **Map**, т.е. **добавляет** к объекту **пару**. Первый параметр – это **ключ**(имя), второй - **значение**.

Мы поместили в Intent два объекта с именами: *fname* и *lname*. *fname* содержит значение поля etFName, *lname* – значение поля etLName. Остается только **отправить** укомплектованный **Intent** с помощью метода **startActivity**.

Теперь создадим второе Activity. Назовем его **ViewActivity**.

Создаем для него layout-файл **view.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView
         android:id="@+id/tvView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="20dp"
            android:text="TextView"
            android:textSize="20sp">
        </TextView>
    </LinearLayout>
```

Здесь просто TextView, который будет отображать пришедшие данные.

Создаем класс **ViewActivity**. И пишем код:

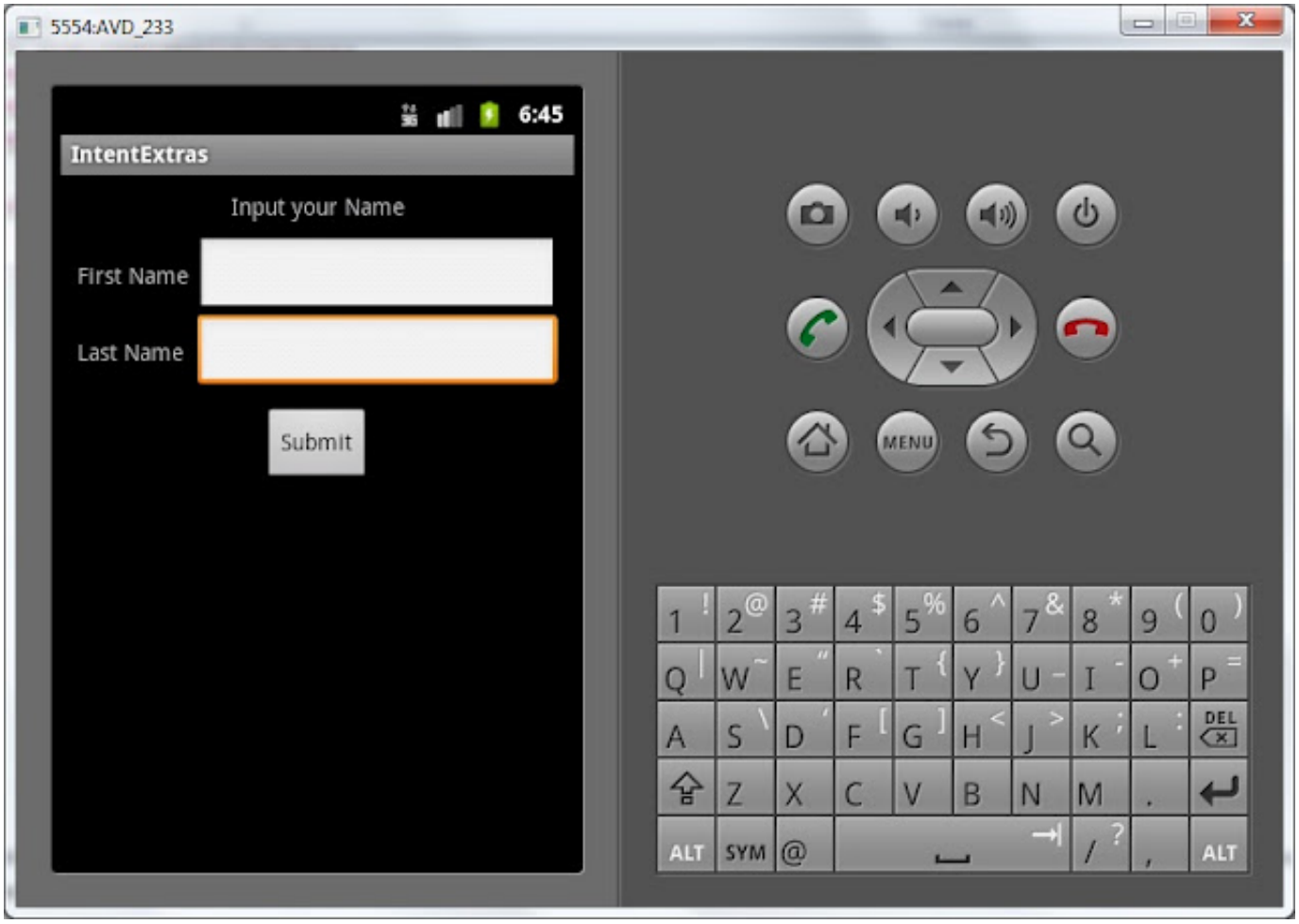
```
package ru.startandroid.develop.p0281intentextras;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class ViewActivity extends Activity {

  TextView tvView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.view);

    tvView = (TextView) findViewById(R.id.tvView);

    Intent intent = getIntent();

    String fName = intent.getStringExtra("fname");
    String lName = intent.getStringExtra("lname");

    tvView.setText("Your name is: " + fName + " " + lName);
  }
}
```

Находим TextView, затем получаем **Intent** и **извлекаем** из него String-объекты с именами *fname* и *lname*. Это те самые объекты, которые мы помещали в коде MainActivity.java. Формируем строку вывода в TextView с использованием полученных данных.

Не забудьте прописать ViewActivity в манифесте. На этот раз никаких Intent Filter не нужно, т.к. мы точно знаем имя класса Activity и используем явный вызов.

Все сохраним и запустим.

Видим такой экран:



Заполняете поля как пожелаете. Я напишу *John* в поле **First Name** и *Smith* в поле **Last Name**.

Жмем Submit:

ViewActivity отобразилось, считало данные из Intent и вывело их на экран.

Поместить в Intent можно данные не только типа String. В списке методов Intent можно посмотреть все многообразие типов, которые умеет принимать на вход метод putExtra.

На следующем уроке:

- вызываем Activity с возвратом результата

Урок 29. Вызываем Activity и получаем результат. Метод startActivityForResult

В этом уроке:

- вызываем Activity с возвратом результата

Бывает необходимость **вызвать Activity**, **выполнить** на нем какое-либо **действие** и **вернуться** с **результатом**. Например – при создании SMS. Вы жмете кнопку «добавить адресата», система показывает экран со списком из адресной книги, вы выбираете нужного вам абонента и возвращаетесь в экран создания SMS. Т.е. вы **вызвали экран выбора** абонента, а он **вернул** вашему экрану **результат**.

Об этом можно почитать здесь и здесь.

Давайте посмотрим на практике. Создадим приложение с двумя экранами. С первого экрана будем вызвать второй экран, там вводить данные, нажимать кнопку и возвращаться на первый экран с введенными данными. Например, будем таким образом запрашивать имя.

Создадим проект:

Project name: P0291_SimpleActivityResult
Build Target: Android 2.3.3
Application name: SimpleActivityResult
Package name: ru.startandroid.develop.p0291simpleactivityresult
Create Activity: MainActivity

Открываем **main.xml** и нарисуем такой экран:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
          android:id="@+id/btnName"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="center_horizontal"
          android:layout_margin="20dp"
          android:text="Input name">
     </Button>
     <TextView
          android:id="@+id/tvName"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="center_horizontal"
          android:text="Your name is ">
     </TextView>
</LinearLayout>
```

На экране **TextView**, который будет отображать имя, и **кнопка**, которая будет вызывать экран для ввода.

Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p0291simpleactivityresult;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity implements OnClickListener {

  TextView tvName;
  Button btnName;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        tvName = (TextView) findViewById(R.id.tvName);
        btnName = (Button) findViewById(R.id.btnName);
        btnName.setOnClickListener(this);

    }

  @Override
  public void onClick(View v) {
    Intent intent = new Intent(this, NameActivity.class);
    startActivityForResult(intent, 1);
  }


  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (data == null) {return;}
    String name = data.getStringExtra("name");
    tvName.setText("Your name is " + name);
  }
}
```

Определяем TextView и кнопку, настраиваем обработчик. В методе обработчика **onClick** создаем **Intent**, указываем **класс** второго **Acivity** (которое создадим чуть позже, на ошибку не обращайте внимания). Для отправки используем startActivityForResult. Отличие от обычного **startActivity** в том, что **MainActivity** становится «**родителем**» для **NameActivity**. И когда NameActivity закрывается, вызывается метод onActivityResult в MainActivity, тем самым давая нам знать, что закрылось Activity, которое мы вызывали методом **startActivityForResult**.

В **startActivityForResult** в качестве параметров мы передаем **Intent** и **requestCode**. **requestCode** – необходим для идентификации. В этом уроке мы его укажем, но не будем использовать по назначению. В следующем же уроке

разберемся подробнее, зачем он нужен.

В **onActivityResult** мы видим следующие параметры:
requestCode – тот же идентификатор, что и в startActivityForResult. По нему определяем, с какого Activity пришел результат.
resultCode – код возврата. Определяет успешно прошел вызов или нет.
data – Intent, в котором возвращаются данные

requestCode и **resultCode** мы пока использовать не будем, подробнее рассмотрим их на следующем уроке. А из **data** мы будем получать объект по имени *name* и выводить значение в **TextView**.

Если мы извлекаем из Intent объект с именем *name*, значит надо, чтобы кто-то его туда положил. Этим займется **NameActivity**.

Создадим экран **name.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
     <LinearLayout
          android:id="@+id/linearLayout1"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_margin="10dp">
         <TextView
              android:id="@+id/textView1"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Name">
         </TextView>
         <EditText
              android:id="@+id/etName"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_marginLeft="10dp"
              android:layout_weight="1">
             <requestFocus>
             </requestFocus>
         </EditText>
     </LinearLayout>
     <Button
          android:id="@+id/btnOK"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="center_horizontal"
          android:text="OK">
     </Button>
</LinearLayout>
```

В поле ввода будем вводить имя и жать кнопку OK.

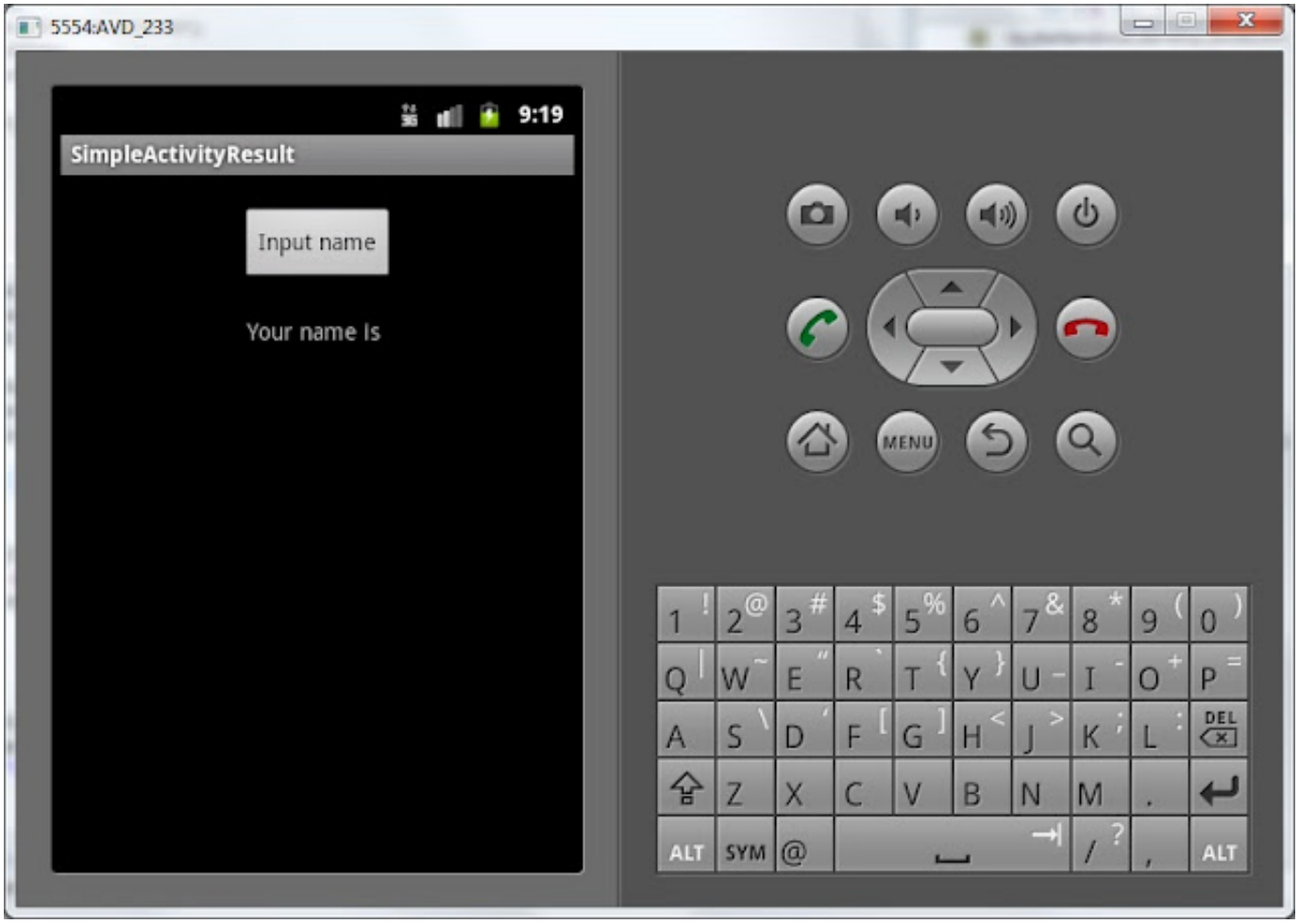
Создаем класс **NameActivity** и прописываем его в манифесте:

```
package ru.startandroid.develop.p0291simpleactivityresult;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class NameActivity extends Activity implements OnClickListener {

  EditText etName;
  Button btnOK;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    // TODO Auto-generated method stub
    super.onCreate(savedInstanceState);
    setContentView(R.layout.name);

    etName = (EditText) findViewById(R.id.etName);
    btnOK = (Button) findViewById(R.id.btnOK);
    btnOK.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
    Intent intent = new Intent();
    intent.putExtra("name", etName.getText().toString());
    setResult(RESULT_OK, intent);
    finish();
  }
}
```

Определяем поле ввода и кнопку, прописываем обработчик. В методе onClick мы **создаем Intent** и **помещаем** в него **данные** из поля ввода под именем name. Обратите внимание, мы никак **не адресуем** этот Intent. Т.е. ни **класс**, ни **action** мы **не указываем**. И получается, что непонятно куда пойдет этот Intent. Но метод setResult знает, куда его адресовать - в «**родительское**» **Activity**, в котором был вызван метод **startActivityForResult**. Также в **setResult** мы передаем константу **RESULT_OK**, означающую **успешное завершение** вызова. И именно она передастся в параметр **resultCode** метода **onActivityResult** в MainActivity.java. Это мы подробнее разберем на следующем уроке. Далее методом **finish** мы завершаем работу NameActivity, чтобы результат ушел в MainActivity.

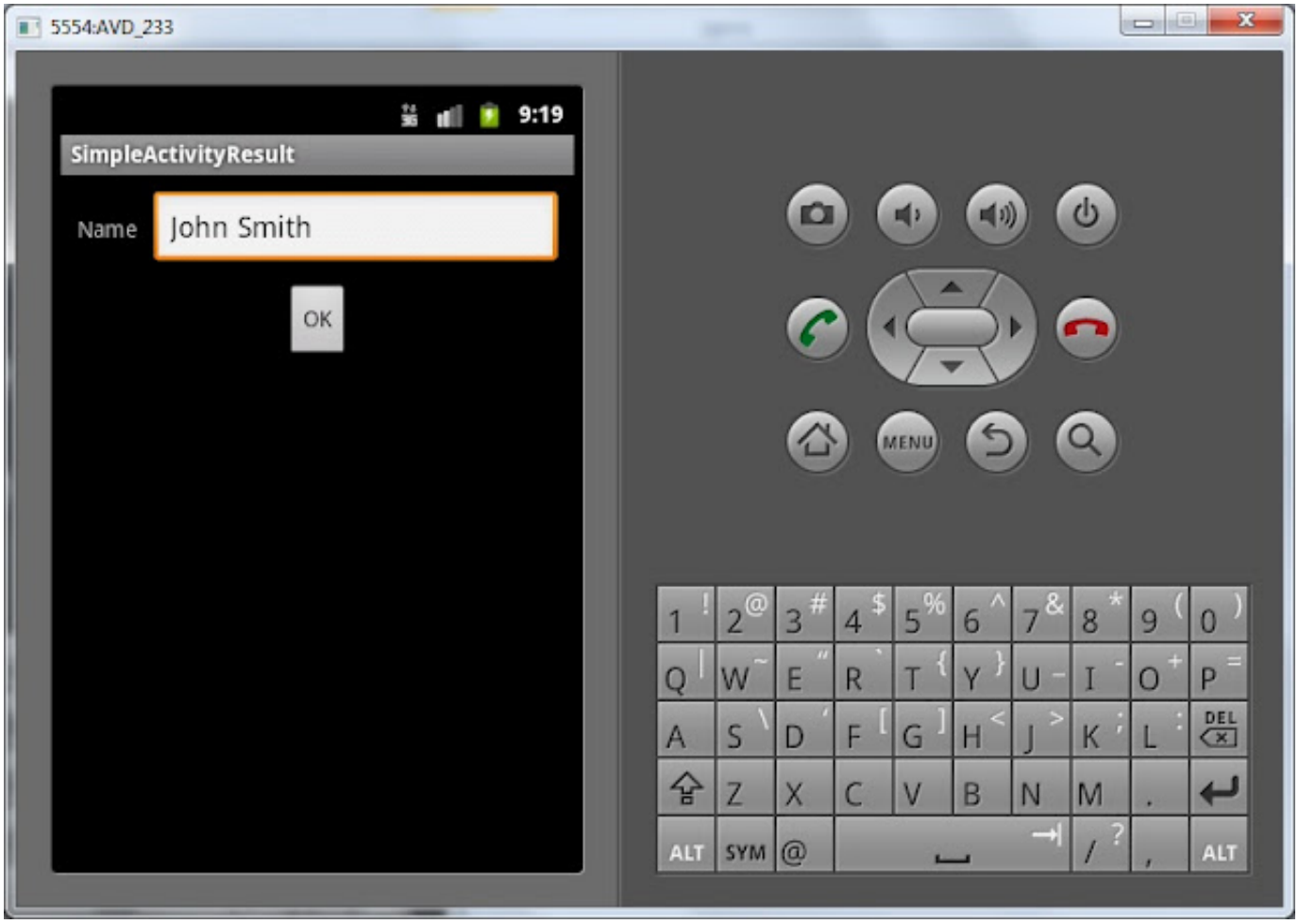
Все сохраним и запустим приложение.

Видим первый экран:

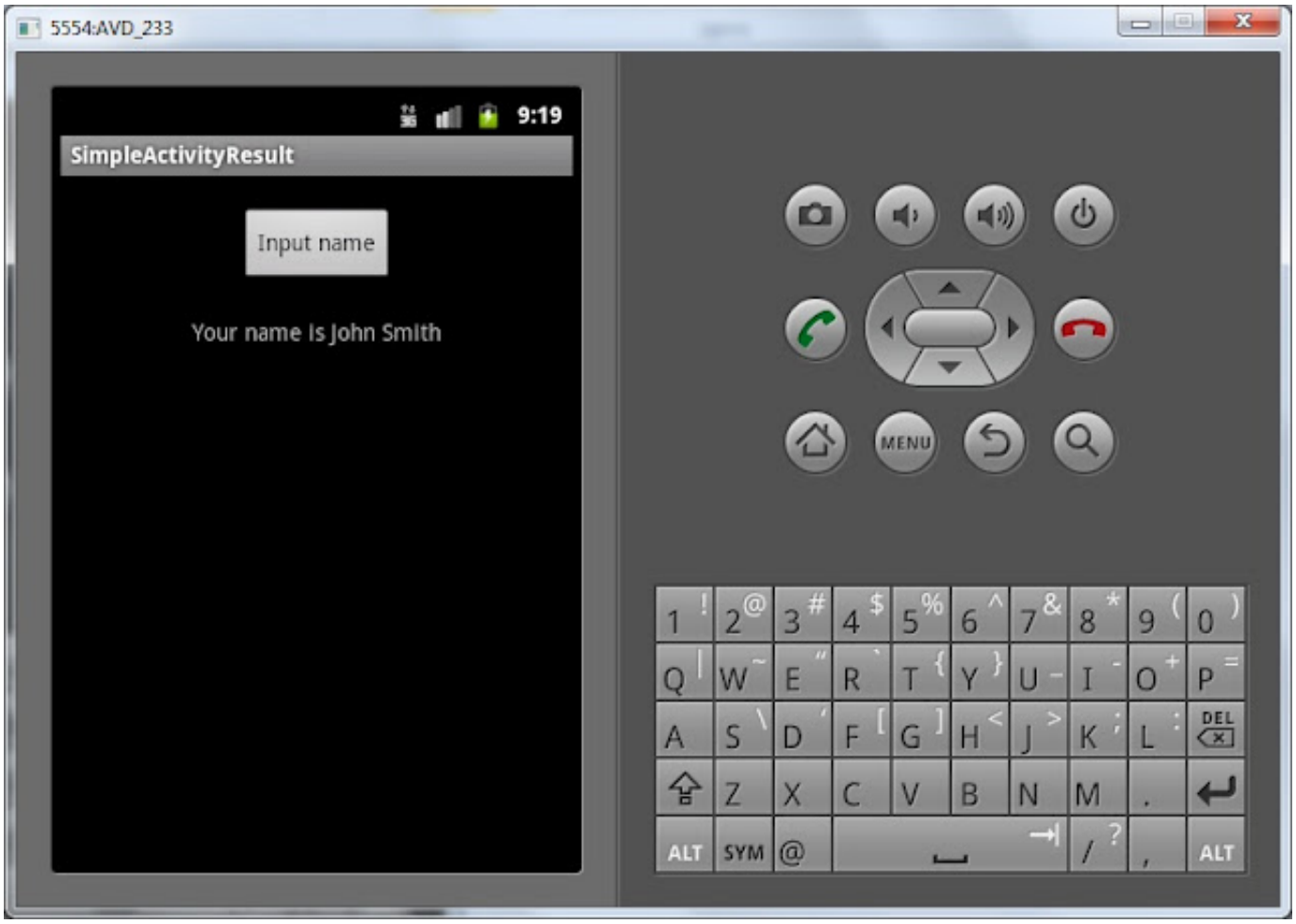


Жмем кнопку, чтобы попасть на экран ввода имени.

Вводим имя и жмем OK

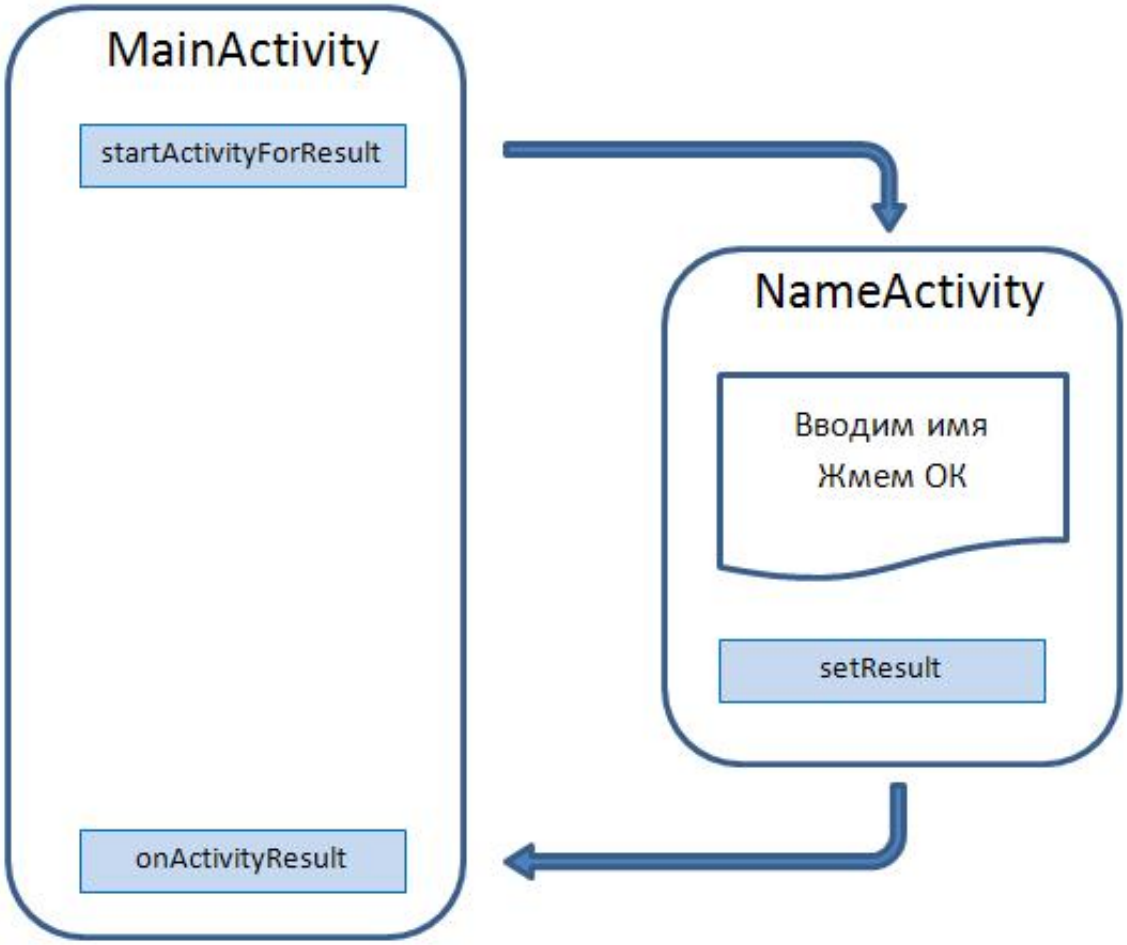


Снова первый экран, отобразивший полученные данные.



Попробуем подытожить. В **MainActivity** мы **создали Intent** с явным указанием на класс NameActivity. Запустили этот Intent с помощью метода **startActivityForResult**. NameActivity отобразилось, мы ввели имя и нажали кнопку. Создался **Intent**, в который **поместилось** введенное нами **имя**. Метод **setResult** знает, что Intent надо вернуть в **Activity**, которое выполнило вызов **startActivityForResult**, т.е. – MainActivity. В MainActivity за прием результатов с вызванных Activity отвечает метод **onActivityResult**. В нем мы распаковали Intent и отобразили полученные данные в TextView.

Пока необходимо просто понять схему вызова и возврата.



На следующем уроке мы сделаем расширенный и более показательный пример использования этой технологии.

На следующем уроке:

- разбираемся, зачем нужны requestCode и resultCode в onActivityResult

Урок 30. Подробнее про onActivityResult. Зачем нужны requestCode и resultCode

В этом уроке:

- разбираемся, зачем нужны requestCode и resultCode в onActivityResult

На прошлом уроке мы поверхностно рассмотрели, как вызвать **Activity**, и как сделать так, чтобы она **вернула результат**. Рассмотрим немного подробней этот механизм. Создадим приложение, которое будет вызывать два разных Activity и получать от них результат. Как мы помним, результат приходит в метод onActivityResult. И **requestCode** используется, чтобы отличать друг от друга пришедшие результаты. А **resultCode** – позволяет определить успешно прошел вызов или нет.

Создадим проект:

Project name: P0301_ActivityResult
Build Target: Android 2.3.3
Application name: ActivityResult
Package name: ru.startandroid.develop.p0301activityresult
Create Activity: MainActivity

Нарисуем экран в **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
           android:id="@+id/tvText"
           android:layout_width="fill_parent"
           android:layout_height="wrap_content"
           android:layout_marginTop="20dp"
           android:gravity="center_horizontal"
           android:text="Hello World"
           android:textSize="20sp">
     </TextView>
     <LinearLayout
           android:id="@+id/linearLayout1"
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:layout_margin="20dp"
           android:orientation="horizontal">
          <Button
                android:id="@+id/btnColor"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginRight="5dp"
```

```
            android:layout_weight="1"
            android:text="Color">
        </Button>
        <Button
            android:id="@+id/btnAlign"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="5dp"
            android:layout_weight="1"
            android:text="Alignment">
        </Button>
    </LinearLayout>
</LinearLayout>
```

На экране **TextView** с текстом. И **две кнопки** для **выбора цвета** шрифта и **выравнивания** текста в TextView. Нажатие на кнопку будет **вызывать** Activity для выбора и получать обратно **результат**.

Давайте начнем кодить в **MainActivity.java**:

```
package ru.startandroid.develop.p0301activityresult;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity implements OnClickListener {

  TextView tvText;
  Button btnColor;
  Button btnAlign;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        tvText = (TextView) findViewById(R.id.tvText);

        btnColor = (Button) findViewById(R.id.btnColor);
        btnAlign = (Button) findViewById(R.id.btnAlign);

        btnColor.setOnClickListener(this);
        btnAlign.setOnClickListener(this);

    }

  @Override
  public void onClick(View v) {
    // TODO Auto-generated method stub

  }
}
```

Определили экранные элементы, прописали обработчик кнопкам и пока остановимся на этом.

Создадим два других Activity. Начнем с **Activity** для **выбора цвета**. Создадим layout-файл **color.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="horizontal">
     <Button
          android:id="@+id/btnRed"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_margin="5dp"
          android:layout_weight="1"
          android:text="Red">
     </Button>
     <Button
          android:id="@+id/btnGreen"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_margin="5dp"
          android:layout_weight="1"
          android:text="Green">
     </Button>
     <Button
          android:id="@+id/btnBlue"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_margin="5dp"
          android:layout_weight="1"
          android:text="Blue">
     </Button>
</LinearLayout>
```

Создаем класс ColorActivity. **ColorActivity.java**:

```
package ru.startandroid.develop.p0301activityresult;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class ColorActivity extends Activity implements OnClickListener {

  Button btnRed;
```

```
  Button btnGreen;
  Button btnBlue;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.color);

    btnRed = (Button) findViewById(R.id.btnRed);
    btnGreen = (Button) findViewById(R.id.btnGreen);
    btnBlue = (Button) findViewById(R.id.btnBlue);

    btnRed.setOnClickListener(this);
    btnGreen.setOnClickListener(this);
    btnBlue.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
    Intent intent = new Intent();
    switch (v.getId()) {
    case R.id.btnRed:
      intent.putExtra("color", Color.RED);
      break;
    case R.id.btnGreen:
      intent.putExtra("color", Color.GREEN);
      break;
    case R.id.btnBlue:
      intent.putExtra("color", Color.BLUE);
      break;
    }
    setResult(RESULT_OK, intent);
    finish();
  }
}
```

Как обычно определяем элементы, присваиваем обработчик кнопкам и реализуем onClick. В onClick мы создаем **Intent**, затем **определяем**, **кнопка** с каким цветом была нажата и помещаем в Intent объект с именем *color* и **значением** цвета. Ставим статус **RESULT_OK**, указываем, что надо вернуть объект **intent** в качестве результата и **закрываем** Activity. Для значения цветов используем системные константы.

Аналогично создаем Activity для выбора выравнивания.

Layout-файл **align.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="horizontal">
    <Button
         android:id="@+id/btnLeft"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:layout_margin="5dp"
```

```
            android:layout_weight="1"
            android:text="Left">
        </Button>
        <Button
            android:id="@+id/btnCenter"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="5dp"
            android:layout_weight="1"
            android:text="Center">
        </Button>
        <Button
            android:id="@+id/btnRight"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="5dp"
            android:layout_weight="1"
            android:text="Right">
        </Button>
    </LinearLayout>
```

AlignActivity.java:

```
package ru.startandroid.develop.p0301activityresult;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Gravity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class AlignActivity extends Activity implements OnClickListener {

  Button btnLeft;
  Button btnCenter;
  Button btnRight;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.align);

    btnLeft = (Button) findViewById(R.id.btnLeft);
    btnCenter = (Button) findViewById(R.id.btnCenter);
    btnRight = (Button) findViewById(R.id.btnRight);

    btnLeft.setOnClickListener(this);
    btnCenter.setOnClickListener(this);
    btnRight.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
```

```
    Intent intent = new Intent();
    switch (v.getId()) {
    case R.id.btnLeft:
      intent.putExtra("alignment", Gravity.LEFT);
      break;
    case R.id.btnCenter:
      intent.putExtra("alignment", Gravity.CENTER);
      break;
    case R.id.btnRight:
      intent.putExtra("alignment", Gravity.RIGHT);
      break;
    }
    setResult(RESULT_OK, intent);
    finish();
  }
}
```

Здесь все аналогично, как и в ColorActivity. Только работаем не с цветами, а с **выравниванием**. Не забудьте прописать оба Activity в манифесте.

Теперь можем завершить код в MainActivity.java. Добавим пару своих **констант** в класс для удобства:

```
  final int REQUEST_CODE_COLOR = 1;
  final int REQUEST_CODE_ALIGN = 2;
```

Эти константы далее будем использовать в качестве requestCode.

Допишем метод onClick:

```
  @Override
  public void onClick(View v) {
    Intent intent;
    switch (v.getId()) {
    case R.id.btnColor:
      intent = new Intent(this, ColorActivity.class);
      startActivityForResult(intent, REQUEST_CODE_COLOR);
      break;
    case R.id.btnAlign:
      intent = new Intent(this, AlignActivity.class);
      startActivityForResult(intent, REQUEST_CODE_ALIGN);
      break;
    }
  }
```

Мы определяем, какая **кнопка** была нажата и посылаем **Intent** с ожиданием возврата результата. Два вызова отличаются **классом** вызываемого Activity и параметром **requestCode** в методе **startActivityForResult**. При вызове **ColorActivity** используем константу **REQUEST_CODE_COLOR**, а при вызове **AlignActivity** - **REQUEST_CODE_ALIGN**. Эту константу мы обратно получим в методе обработки результата – **onActivityResult**, и по ней сможем определить из **какого** именно **Activity** пришел **результат**.

Давайте реализуем метод onActivityResult в MainActivity.java:

```
  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // запишем в лог значения requestCode и resultCode
```

```
    Log.d("myLogs", "requestCode = " + requestCode + ", resultCode = " + resultCode);
    // если пришло ОК
    if (resultCode == RESULT_OK) {
      switch (requestCode) {
      case REQUEST_CODE_COLOR:
        int color = data.getIntExtra("color", Color.WHITE);
        tvText.setTextColor(color);
        break;
      case REQUEST_CODE_ALIGN:
        int align = data.getIntExtra("alignment", Gravity.LEFT);
        tvText.setGravity(align);
        break;
      }
    // если вернулось не ОК
    } else {
      Toast.makeText(this, "Wrong result", Toast.LENGTH_SHORT).show();
    }
  }
```

Для наглядности пишем в лог значения переменных.

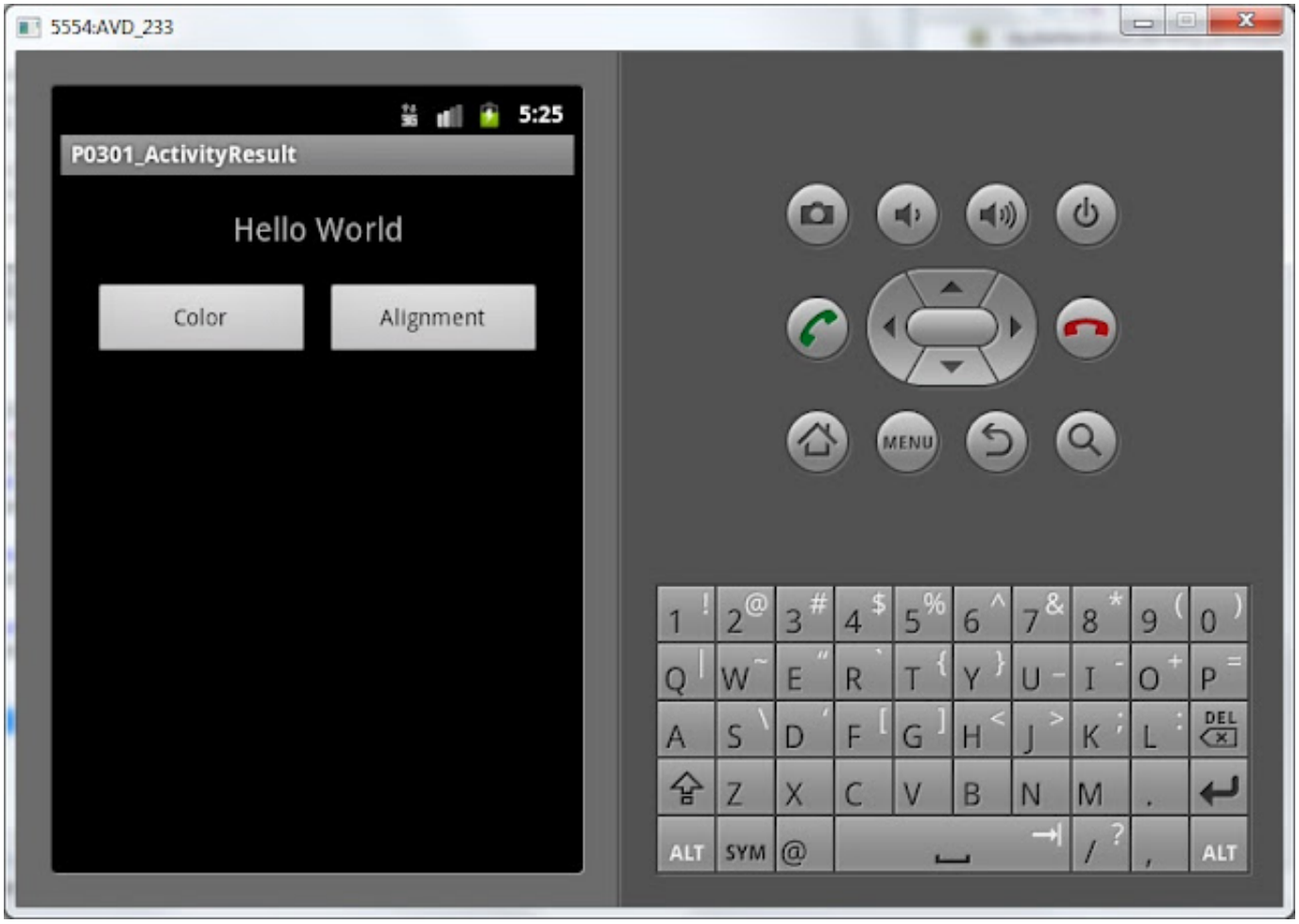
Вспоминаем, что в ColorActivity и AlignActivity в методе setResult мы ставили статус **RESULT_OK** при отправке результата. Значит в **onActivityResult** нам надо ожидать этот статус, как обозначение успешного окончания вызова.

Если вызов прошел успешно (resultCode = RESULT_OK), то мы смотрим значение **requestCode**. Если оно равно константе **REQUEST_CODE_COLOR**, то вспоминаем, что мы использовали эту константу в методе **startActivityForResult**, когда отправляли запрос на **выбор цвета**. Значит, нам пришел **результат** этого **выбора**. Мы берем **Intent** (data) и **извлекаем** из него значение объекта с именем *color* и присваиваем это значение **цвету** текста в TextView. Константа **Color.WHITE** в методе **getIntExtra** означает значение по умолчанию. Т.е. если в Intent не найдется объекта с именем *color*, то метод вернет белый (white) цвет.

Аналогично для **REQUEST_CODE_ALIGN**. Эту константу мы использовали для запроса выбора **выравнивания**. И если в методе **onActivityResult** параметр **requestCode** = этой константе, значит пришел ответ на запрос выравнивания. И мы считываем это значение из Intent и присваиваем его атрибуту Gravity для TextView.

Если **resultCode** не равен **RESULT_OK**, значит что-то пошло не так. Выводим на экран соответствующее сообщение. Этот случай может наступить, например, если на экране выбора не делать выбор, а нажать кнопку Назад.

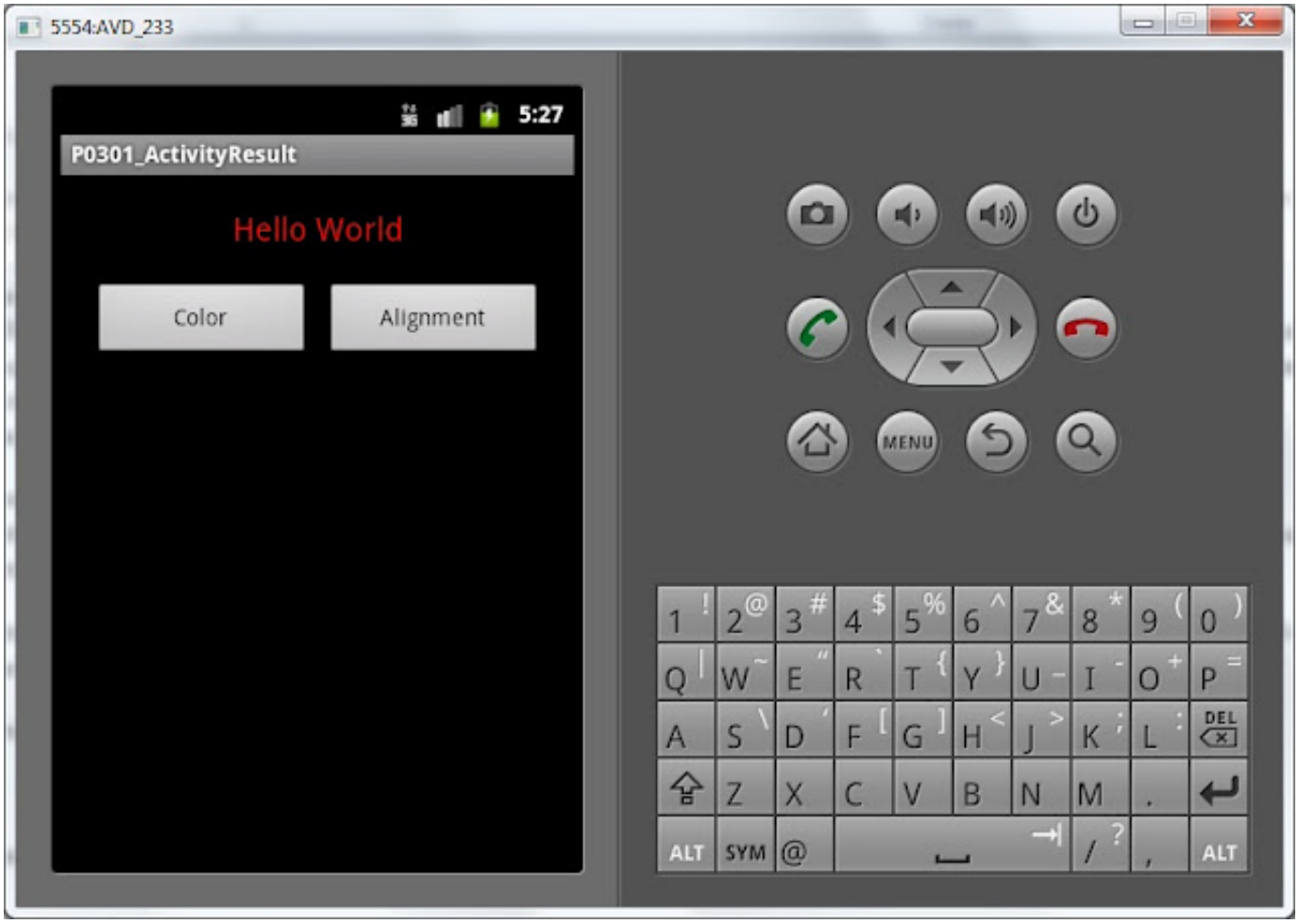
Давайте все сохраним и запустим приложение.



Нажмем Color

и выберем, например Red

Цвет изменился



смотрим лог:

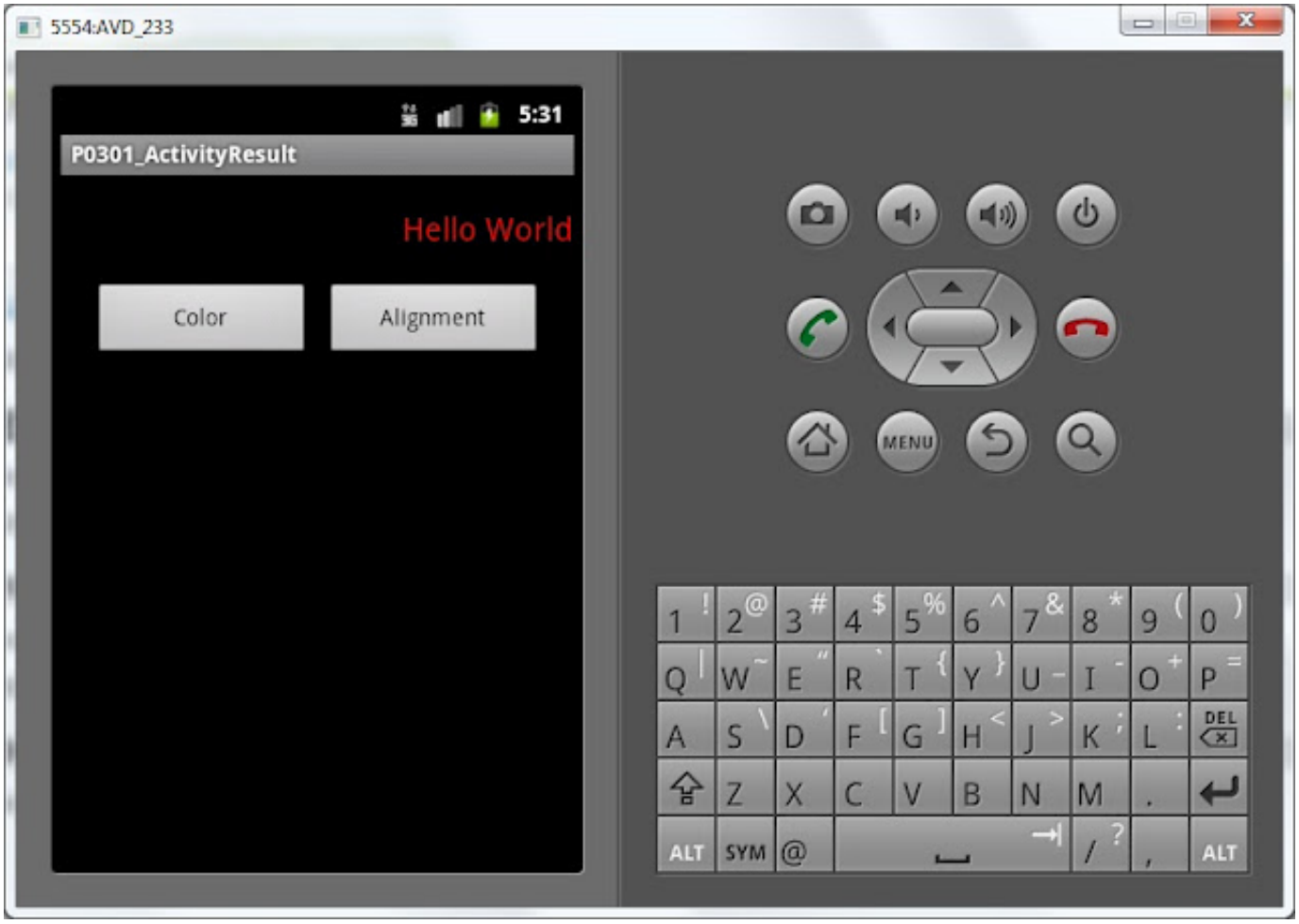
requestCode = 1, resultCode = -1

requestCode пришедший в метод **onActivityResult** равен 1. Все верно, это значение константы **REQUEST_CODE_COLOR**, которое мы использовали при вызове.

resultCode = -1 – это значение системной константы RESULT_OK

Т.е. все верно, пришел ответ на запрос цвета, и его статус = **RESULT_OK**.

Теперь жмем Alignment и выбираем Right, получаем выравнивание вправо:



Смотрим лог:

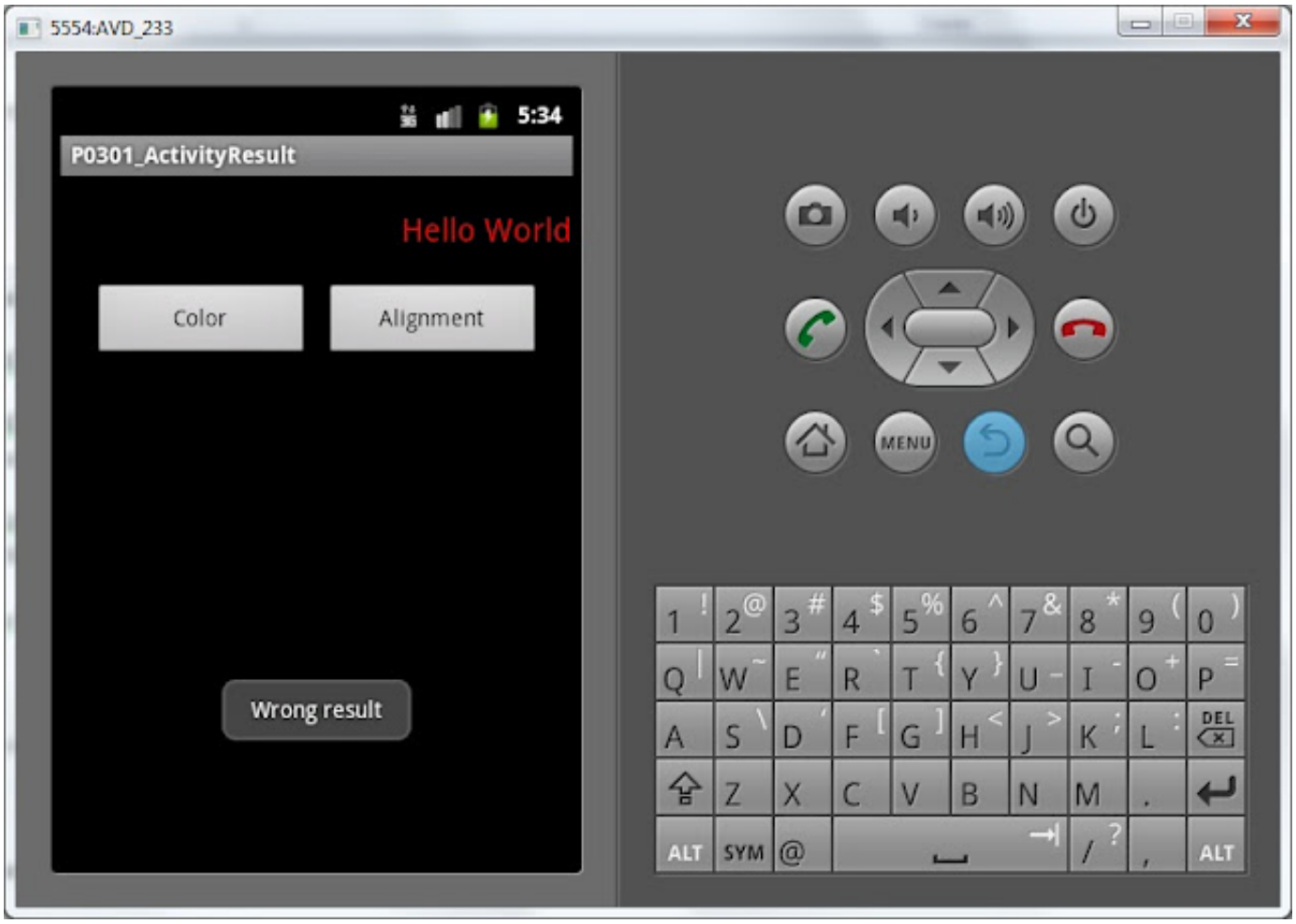
requestCode = 2, resultCode = -1

requestCode = 2, что равно константе **REQUEST_CODE_ALIGN**. Значит пришел ответ на запрос выравнивания.

resultCode = -1, т.е. RESULT_OK.

Теперь снова жмем Color

но вместо того, чтобы выбрать цвет нажмем кнопку назад



Отобразилось наше сообщение об ошибке. Смотрим логи:

requestCode = 1, resultCode = 0

requestCode = 1 – все верно, мы запрашивали цвет (**REQUEST_CODE_COLOR**)

resultCode = 0, это значение константы RESULT_CANCELED, значит вызов прошел неудачно

Ограничений на значение статуса в методе setResult нет. RESULT_OK и RESULT_CANCELED – системные общепринятые константы. Но вы можете свободно использовать свои значения, если в этом есть необходимость.

Итак, подведем итог.

requestCode – это в некотором роде ID запроса. Задается в методе **startActivityForResult**, и проверяется потом в **onActivityResult**, чтобы точно знать, на какой вызов пришел ответ.

resultCode – статус вызова. Задается в методе **setResult**, и проверяется в **onActivityResult**, чтобы понять насколько успешно прошел вызов. Если при вызове что-то пошло не так, то вернется системная константа **RESULT_CANCELED**.

Полный код **MainActivity.java**:

```
package ru.startandroid.develop.p0301activityresult;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.util.Log;
import android.view.Gravity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener {

  final int REQUEST_CODE_COLOR = 1;
  final int REQUEST_CODE_ALIGN = 2;

  TextView tvText;
  Button btnColor;
  Button btnAlign;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvText = (TextView) findViewById(R.id.tvText);

    btnColor = (Button) findViewById(R.id.btnColor);
    btnAlign = (Button) findViewById(R.id.btnAlign);
```

```
    btnColor.setOnClickListener(this);
    btnAlign.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
    Intent intent;
    switch (v.getId()) {
    case R.id.btnColor:
      intent = new Intent(this, ColorActivity.class);
      startActivityForResult(intent, REQUEST_CODE_COLOR);
      break;
    case R.id.btnAlign:
      intent = new Intent(this, AlignActivity.class);
      startActivityForResult(intent, REQUEST_CODE_ALIGN);
      break;
    }
  }

  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // запишем в лог значения requestCode и resultCode
    Log.d("myLogs", "requestCode = " + requestCode + ", resultCode = " + resultCode);
    // если пришло OK
    if (resultCode == RESULT_OK) {
      switch (requestCode) {
      case REQUEST_CODE_COLOR:
        int color = data.getIntExtra("color", Color.WHITE);
        tvText.setTextColor(color);
        break;
      case REQUEST_CODE_ALIGN:
        int align = data.getIntExtra("alignment", Gravity.LEFT);
        tvText.setGravity(align);
        break;
      }
      // если вернулось не OK
    } else {
      Toast.makeText(this, "Wrong result", Toast.LENGTH_SHORT).show();
    }
  }
}
```

На следующем уроке:

- узнаем что такое URI
- вызываем системные приложения (браузер, звонилка, карта)

Урок 31. Зачем у Intent есть атрибут data. Что такое Uri. Вызываем системные приложения

В этом уроке:

- узнаем, что такое Uri и Intent-атрибут data
- вызываем системные приложения (браузер, звонилка, карта)

Мы знаем, что **Intent** имеет атрибут **action**. С помощью этого атрибута обычно дается указание **действия**. Например, **просмотр** или **редактирование**. Но действие обычно совершается не просто так, а с чем-либо. Значит кроме указания действия, мы должны указывать на **объект**, с которым эти действия нужно произвести. Для этого **Intent** имеет атрибут **data**.

Один из способов присвоения значения этому атрибуту – метод setData (Uri data) у объекта **Intent**. На вход этому методу подается объект Uri.

Uri – это объект, который берет **строку**, **разбирает** ее на составляющие и хранит в себе эту информацию. Строка, конечно, должна быть не любая, а составлена в соответствии с этим документом RFC 2396. Uri имеет кучу **методов**, которые позволяют **извлекать** из распарсенной **строки** отдельные **элементы**.

Я создам объект Uri из строки, а в лог буду выводить название метода и (через двоеточие) значение, которое он возвращает. Например возьмем такую строку - http адрес:

```
Uri uri = Uri.parse("http://developer.android.com/reference/android/net/Uri.html");
```

Смотрим, чего нам возвращают методы:

uri.getScheme(): http
uri.getSchemeSpecificPart(): //developer.android.com/reference/android/net/Uri.html
uri.getAuthority(): developer.android.com
uri.getHost(): developer.android.com
uri.getPath(): /reference/android/net/Uri.html
uri.getLastPathSegment(): Uri.html

Понятия Scheme, Authority, Host, Path и пр. – взяты из RFC дока, ссылку на который я дал выше. Там можно найти их полное описание, понять что они означают и свериться с тем, что нам вернул Uri.

Рассмотрим еще примеры:

FTP

```
Uri uri = Uri.parse("ftp:// bob@google.com:80/data/files");
```

(*Код, написанный выше, идет одной строкой на самом деле. Здесь идет пробел из-за особенностей разметки*)

uri.getScheme(): ftp
uri.getSchemeSpecificPart(): // bob@google.com:80/data/files
uri.getAuthority(): bob@google.com:80
uri.getHost(): google.com
uri.getPort(): 80
uri.getPath(): /data/files
uri.getLastPathSegment(): files
uri.getUserInfo(): bob

Координаты

```
Uri uri = Uri.parse("geo:55.754283,37.62002");
```

uri.getScheme(): geo
uri.getSchemeSpecificPart(): 55.754283,37.62002

Здесь уже получилось выделить только Scheme и SchemeSpecificPart.

Номер телефона

```
Uri uri = Uri.parse("tel:12345");
```

uri.getScheme(): tel
uri.getSchemeSpecificPart():12345

Аналогично, получилось выделить только две части из строки.

Контакт из адресной книги

```
Uri uri = Uri.parse("content://contacts/people/1");
```

uri.getScheme(): content
uri.getSchemeSpecificPart(): //contacts/people/1
uri.getAuthority(): contacts
uri.getPath(): /people/1
uri.getLastPathSegment(): 1

В этом примере **Scheme** равен *content*. Это особый тип данных – Content Provider. Он позволяет любой программе давать доступ к своим данным, а другим программам – читать и менять эти данные. Эту тему мы рассмотрим позднее, и сами будем создавать такой тип данных.

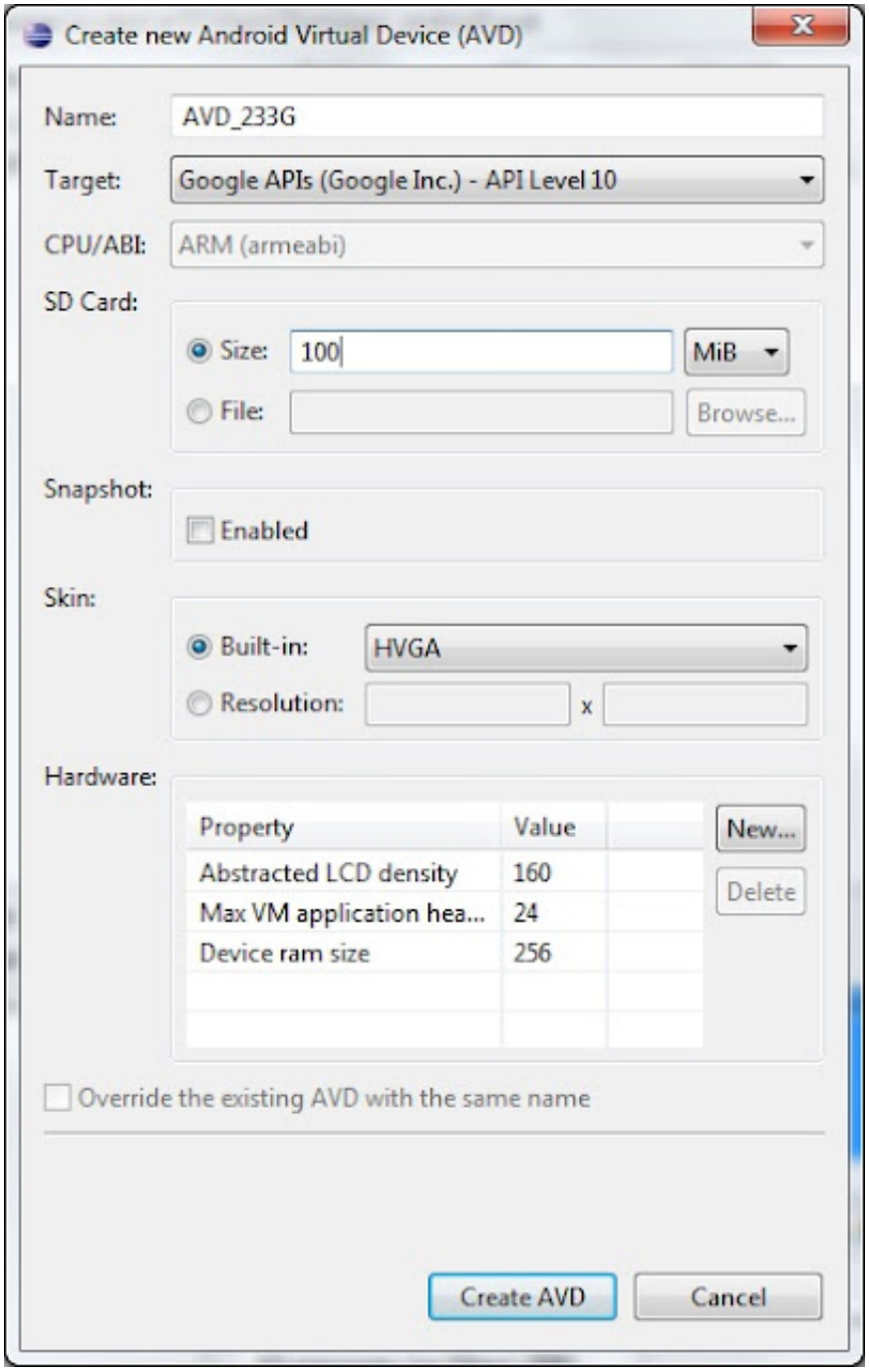
Здесь можно посмотреть какие стандартные Uri поддерживаются.

Примеры показывают, что **Uri** можно создать из абсолютно разных строк: http-адрес, ftp-адрес, координаты, номер телефона, контакт из адресной книги.

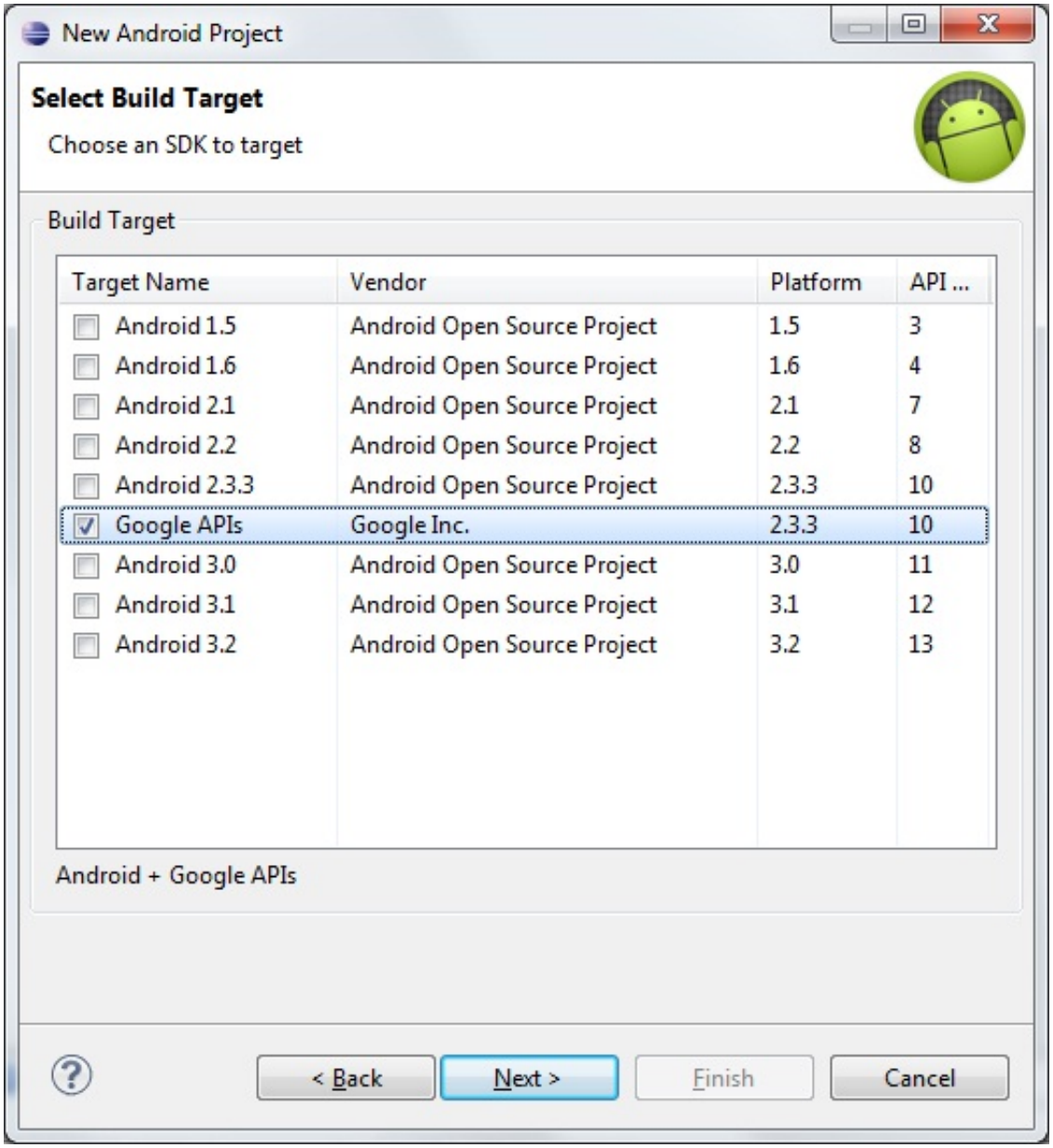
Тип содержимого можно определить по **Scheme**. И этот же **Scheme** можно настроить в **Intent Filter** и отсеивать **Intent**, только с нужным нам типом данных в **Uri**, например только http. Этим мы еще займемся позднее, а пока сделаем простой пример, в котором будем формировать Intent с action и data, отправлять его и смотреть, что получится. Попробуем просмотреть следующее: http-адрес, координаты на карте и открыть окно набора номера.

Чтобы посмотреть координаты на карте, необходимо приложение Google Maps. Его нет в стандартных Android платформах. Нам нужны дополнения от Google. Как их установить, я расписал здесь. После обновления может немного поменяться интерфейс. Не теряйтесь)

Создайте AVD на платформе Google APIs с API Level 10. Назовите его на ваше усмотрение.



Создадим проект. Обратите внимание, используем платформу Google APIs версии 2.3.3



Project name: P0311_SimpleIntents
Build Target: Google APIs 2.3.3
Application name: SimpleIntents
Package name: ru.startandroid.develop.p0311simpleintents
Create Activity: MainActivity

Если у вас не получилось установить Google APIs, то создавайте проект как обычно - с платформой Android 2.3.3. Просто не будет работать вызов Google Maps в этом примере.

Сформируем экран **main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
	xmlns:android="http://schemas.android.com/apk/res/android"
	android:layout_width="fill_parent"
	android:layout_height="fill_parent"
	android:orientation="horizontal">
	<Button
		android:id="@+id/btnWeb"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:text="Web">
        </Button>
        <Button
            android:id="@+id/btnMap"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:text="Map">
        </Button>
        <Button
            android:id="@+id/btnCall"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:text="Call">
        </Button>
</LinearLayout>
```

На экране три кнопки. Первая будет открывать веб-страницу, вторая - карту, третья – звонилку.

Пишем код в **MainActivity.java**:

```
package ru.startandroid.develop.p0311simpleintents;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener {

  Button btnWeb;
  Button btnMap;
  Button btnCall;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    btnWeb = (Button) findViewById(R.id.btnWeb);
    btnMap = (Button) findViewById(R.id.btnMap);
    btnCall = (Button) findViewById(R.id.btnCall);

    btnWeb.setOnClickListener(this);
```

```
    btnMap.setOnClickListener(this);
    btnCall.setOnClickListener(this);
  }

  @Override
  public void onClick(View v) {
    Intent intent;
    switch (v.getId()) {
    case R.id.btnWeb:
      intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://developer.android.com"));
      startActivity(intent);
      break;
    case R.id.btnMap:
      intent = new Intent();
      intent.setAction(Intent.ACTION_VIEW);
      intent.setData(Uri.parse("geo:55.754283,37.62002"));
      startActivity(intent);
      break;
    case R.id.btnCall:
      intent = new Intent(Intent.ACTION_DIAL);
      intent.setData(Uri.parse("tel:12345"));
      startActivity(intent);
      break;
    }
  }
}
```

Я использовал три разных способа создания Intent-а и задания его атрибутов.

В случае **btnWeb** я использовал конструктор Intent (String action, Uri uri). Он создает **Intent** и на вход сразу принимает **action** и **data**. Мы используем стандартный системный **action** – ACTION_VIEW. Это константа в классе **Intent** – означает, что мы хотим **просмотреть** что-либо. В качестве **data** мы подаем объект **Uri**, созданный из веб-ссылки: http://developer.android.com. И если попытаться описать словами наш код, то получится так: этот Intent означает, что мы хотим посмотреть содержимое этой ссылки и ищем Activity, которая могла бы нам помочь.

В случае **btnMap** использовался конструктор Intent(). Он просто создает **Intent**. А в следующих строках мы уже присваиваем ему атрибуты **action** и **data**. **action** – снова ACTION_VIEW, а в качестве **data** мы создаем **Uri** из пары координат - *55.754283,37.62002*. Этот **Intent** означает, что мы хотим посмотреть на карте указанные координаты.

В случае **btnCall** используем конструктор Intent (String action). На вход ему сразу подается **action**, а **data** указывается позже. **action** в данном случае – ACTION_DIAL – открывает звонилку и набирает номер, указанный в **data**, но не начинает звонок. В **data** – помещаем **Uri**, созданный из номера телефона *12345*.

Три этих способа приводят к одному результату - **Intent** с заполненными атрибутами **action** и **data**. Какой из них использовать - решать вам в зависимости от ситуации.

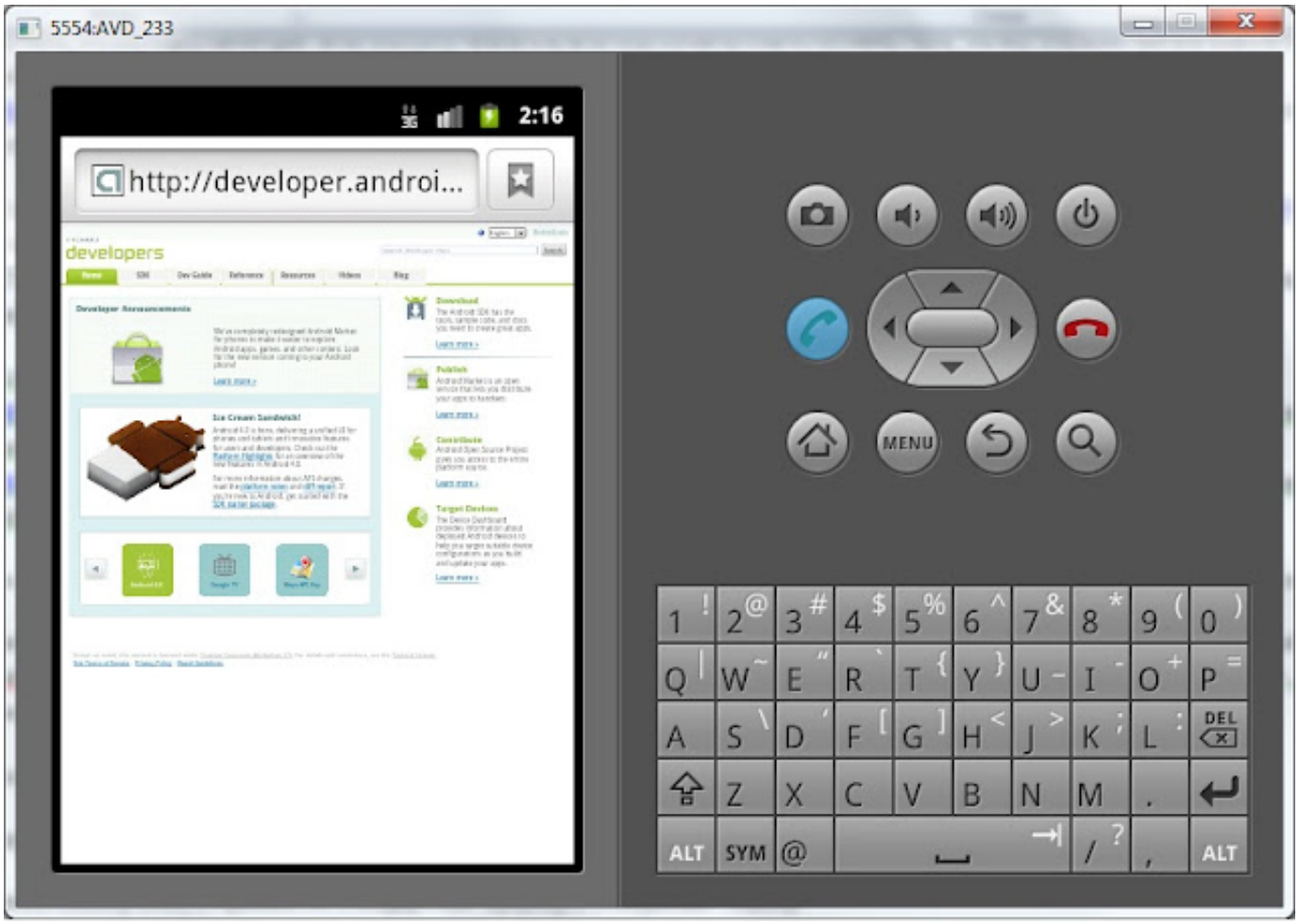
Т.к. нашему приложению понадобится интернет, чтобы открыть ссылку и посмотреть карту, надо чтобы на вашем компе интернет был.

Также в файле **манифеста** приложения, на вкладке **Permission** добавьте элемент **Uses Permission** и справа в поле **Name** выберите *android.permission.INTERNET*. Это даст приложению доступ в интернет. Правда у меня почему-то и без этого все работает ... Пока не понял почему.

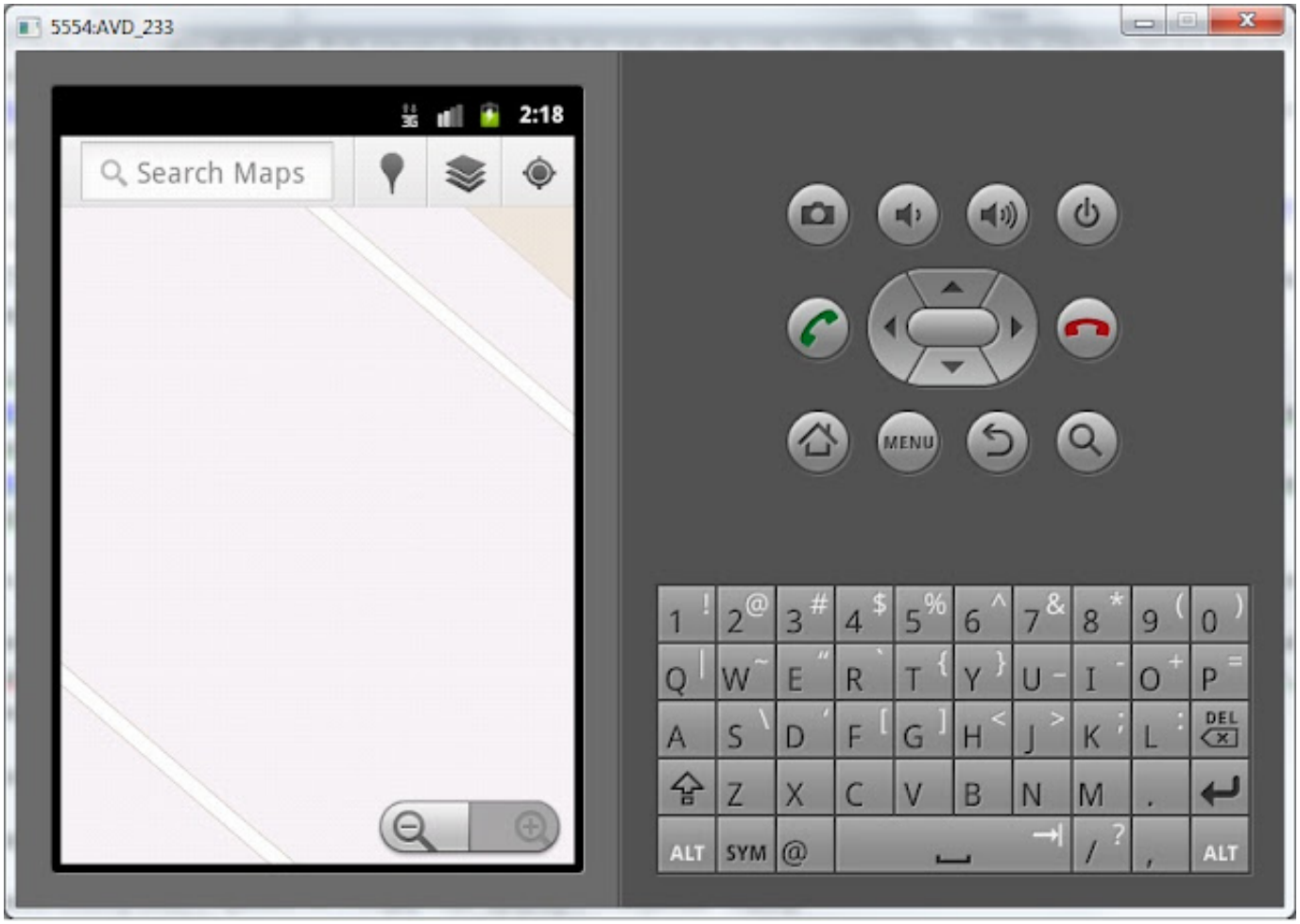
Все сохраняем и запускаем приложение

Жмем кнопку Web,

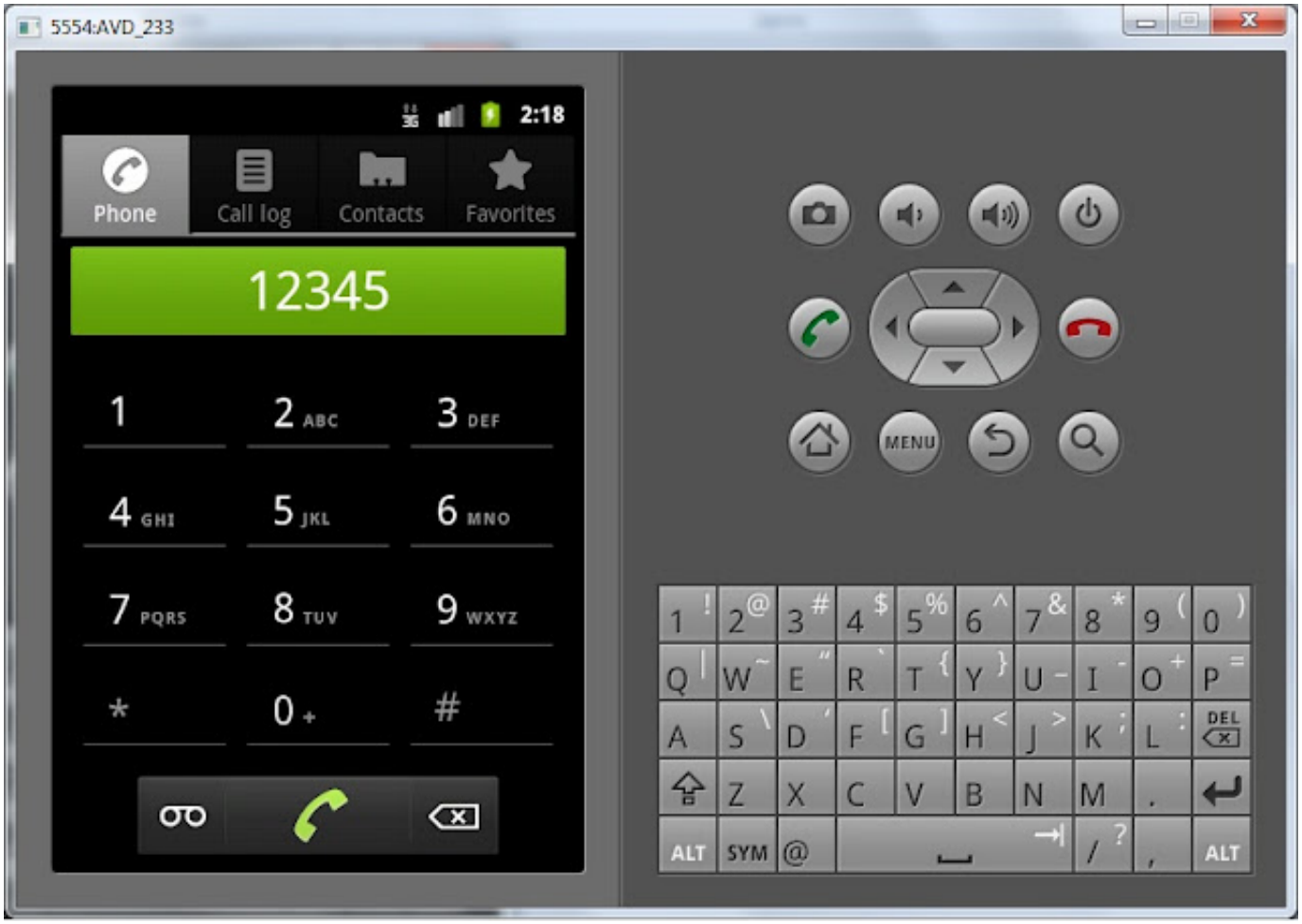
открывается стандартный браузер и отображает содержимое страницы по ссылке



Возвращаемся, жмем Map. Отображается карта. Уменьшите масштаб, чтобы понять, координаты какого места я использовал в примере)



Возвращаемся, жмем Call. Отображается стандартный экран набора номера и видим, что номер, который мы указывали в data, уже набран. Нам остается только нажать кнопку звонка.



Скорее всего, сейчас есть много вопросов типа «Что будет если … ». На некоторые из них сразу могу ответить и предлагаю вам поэкспериментировать в текущем приложении:

1) Что будет, если указать координаты без приставки *geo*:

Система ругнется, что не нашла подходящего Activity (см. логи). Т.к. в Activity карты настроен Intent Filter, который (как я думаю) настроен на data с Schema = *geo*.

Аналогично не сработает звонилка, если указать номер без приставки *tel*.

2) Что будет, если в координатах оставить *geo*, но координаты указать кривые?

Если мы попробуем посмотреть, например, такие координаты *geo:a,b*, то карта запустится, но скажет нам *Unable to load the URL*. Т.е. данные подошли по Schema, но оказались некорректными.

3) Что будет, если координаты указать верно, но action использовать не ACTION_VIEW, а ACTION_EDIT.

Получается, что мы хотим отредактировать место на карте заданное этими координатами. Но система говорит нам, что она не нашла такое Activity. Потому что приложение Google Maps ожидает Intent с action = ACTION_VIEW и оно сможет показать нам это место на карте. А на редактирование оно не подписывалось)

Необходимо понять, что все приложения в системе заточены под конкретные действия с конкретными типами

данных. И если вы попробуете позвонить на адрес сайта, или открыть на карте номер телефона – то система просто не найдет приложения, способные на это.

На следующем уроке:

- пишем простой браузер

Урок 32. Пишем простой браузер

В этом уроке:

- пишем простой браузер

На прошлом уроке мы увидели, что если вызвать **Intent** с **action** = **ACTION_VIEW** и **data** = **Uri**-объект с **http**-адресом, то запускается **браузер** и отображает содержимое страницы по этому http-адресу. Мы можем самостоятельно сделать простейший браузер, который будет реагировать на такой Intent и просто отобразит страницу. Для этого надо настроить **Intent Filter** и использовать компонент **WebView**.

На первом экране приложения у нас будет кнопка, отправляющая Intent. На втором экране будет WebView.

Создадим проект:

Project name: P0321_SimpleBrowser
Build Target: Android 2.3.3
Application name: SimpleBrowser
Package name: ru.startandroid.develop.p0321simplebrowser
Create Activity: MainActivity

Рисуем **main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnWeb"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="web">
    </Button>
</LinearLayout>
```

На экране просто кнопка

Кодим **MainActivity.java:**

```
package ru.startandroid.develop.p0321simplebrowser;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

public class MainActivity extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    (findViewById(R.id.btnWeb)).setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.ya.ru")));
      }
    });
  }
}
```

Код немного непривычен. Обратите внимание я нигде не описываю объект класса **Button**. Метод **findViewById** возвращает **View**, и это **View** поддерживает метод **setOnClickListener**, который я вызываю. А в методе **setOnClickListener** я создаю объект, реализующий интерфейс **OnClickListener** и в нем пишу код в **onClick**. Также я создаю объект **Intent** не отдельно, а прямо в методе **startActivity**. Кода меньше получилось, чем обычно. Может быть вам подойдет такой вариант.

Итак, мы по **нажатию** на кнопку запускаем **Intent**, который означает, что мы хотим просмотреть сайт http://www.ya.ru.

Создадим второе Activity. Сначала layout-файл **browser.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <WebView
         android:id="@+id/webView"
         android:layout_width="match_parent"
         android:layout_height="match_parent">
```

```
    </WebView>
</LinearLayout>
```

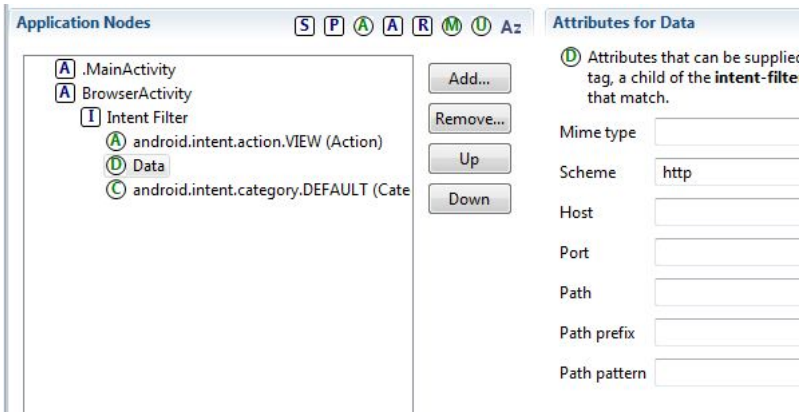
На экране компонент **WebView**.

Создаем **BrowserActivity.java**:

```
package ru.startandroid.develop.p0321simplebrowser;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.webkit.WebView;

public class BrowserActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.browser);

    WebView webView = (WebView) findViewById(R.id.webView);
    Uri data = getIntent().getData();
    webView.loadUrl(data.toString());
  }
}
```

Определяем **WebView**, читаем **data** из **Intent** и передаем **строку** в **WebView**.

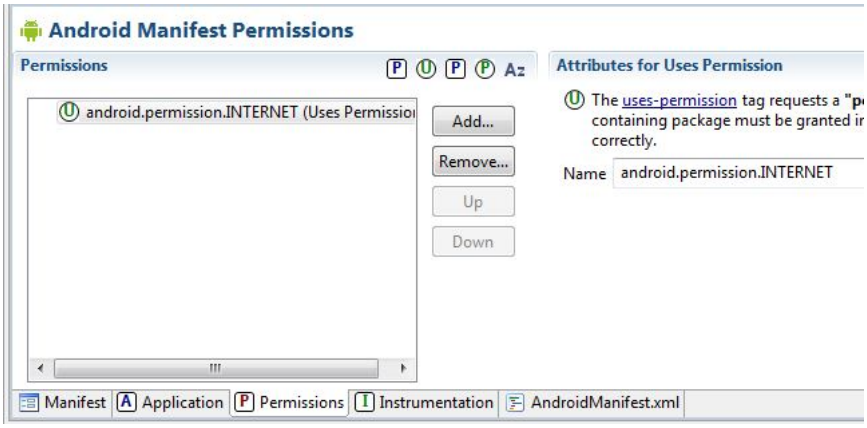
Теперь пропишем **Activity** в **манифесте**. К нему нам надо будет добавить **Intent Filter**, в нем указать **action** = **ACTION_VIEW**. А для **data** мы видим несколько параметров, используем **Scheme** = *http*.



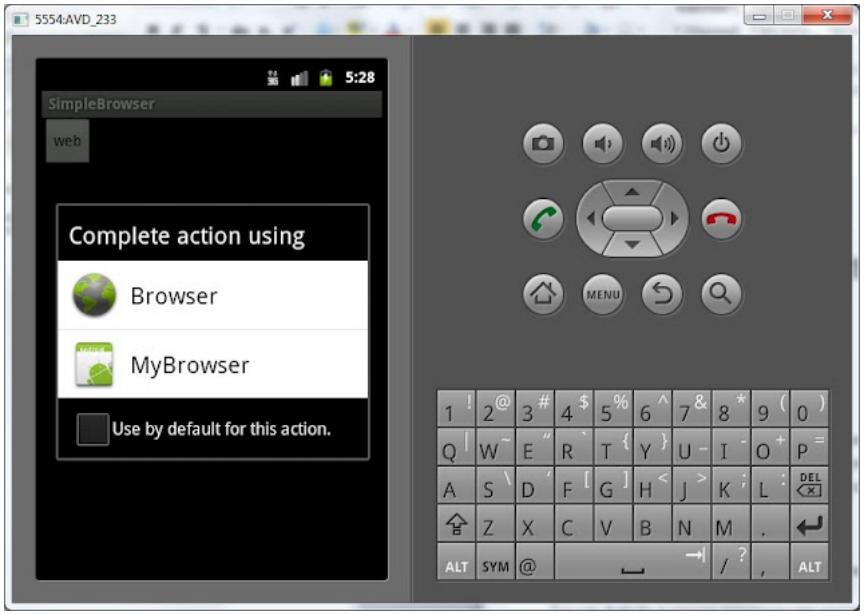
Это значит, что **Uri** объект в **Intent** должен содержать http-адрес.

Не забываем про **Category** = **Default**. **Label** для BrowserActivity укажите, например, *MyBrowser*.

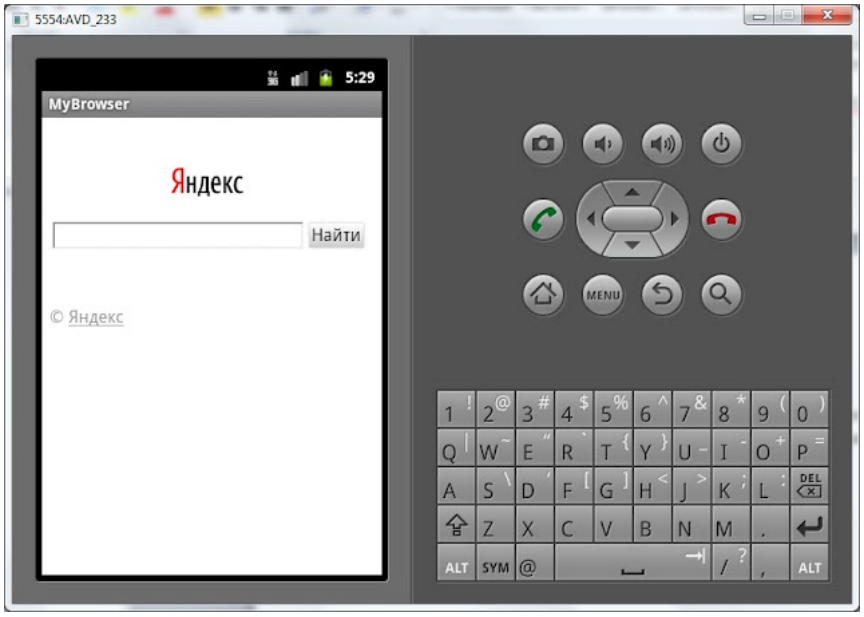
Также в манифесте надо добавить **Uses Permission** = **android.permission.INTERNET** на вкладке **Permissions**. Чтобы система дала приложению доступ в интернет.



Все сохраним и запустим приложение. Жмем кнопку и видим выбор: система предлагает нам на выбор **системный браузер** и **наш**, только что сделанный. Т.е. **Intent** с запросом на просмотр http-адреса нашел в системе **два** Activity, которые в своих **Intent Filter** заявили, что умеют отображать http-адреса.



Выбираем наше MyBrowser и видим страницу.



Мы увидели, что **Activity** в наших приложениях могут **обрабатывать** не только **наши** придуманные action, но и **системные**. И, тем самым, создавать альтернативу системным приложениям.

Но, как вы понимаете, мы запросто могли в нашем Activity не использовать WebView и не показывать страницу. Можно было использовать TextView и в нем просто отобразить в виде текста адрес из data. Или накодить http-запрос, который скачал бы эту страницу и отобразил ее html-содержимое. Мы могли вообще забить на http-адрес и показать какую-нибудь картинку левую или просто темный экран.

Т.е. для Activity можно создать Intent Filter, который будет сообщать системе, что приложение умеет что-то, но, при этом, внутри Activity будет какая-нибудь ерунда. Это уже вопросы программерской этики, здравого смысла и адекватности)

Полный код манифест-файла:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="ru.startandroid.develop.p0321simplebrowser" android:versionCode="1" a
    <uses-sdk android:minSdkVersion="10"></uses-sdk>
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <application android:icon="@drawable/ic_launcher" android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"></action>
                <category android:name="android.intent.category.LAUNCHER"></category>
            </intent-filter>
        </activity>
        <activity android:label="MyBrowser" android:name="BrowserActivity">
            <intent-filter>
                <action android:name="android.intent.action.VIEW"></action>
                <data android:scheme="http"></data>
                <category android:name="android.intent.category.DEFAULT"></category>
            </intent-filter>
        </activity>
```

```
    </application>
</manifest>
```

На следующем уроке:

- хранение данных с помощью Preferences

Урок 33. Хранение данных. Preferences.

В этом уроке:

- хранение данных с помощью Preferences

Хватит об Intent и Activity. Поговорим о хранении данных. В Android есть несколько способов хранения данных:

Preferences - в качестве аналогии можно привести виндовые INI-файлы

SQLite - база данных, таблицы

обычные файлы - внутренние и внешние (на SD карте)

Мы начнем с самого простого – **Preferences**. Значения сохраняются в виде пары: **имя**, **значение**. Так же, как и например extras в Intent.

Разработаем приложение. В нем будет поле для ввода текста и две кнопки – **Save** и **Load**. По нажатию на **Save** мы будем **сохранять** значение из поля, по нажатию на **Load** – **загружать**.

Создадим проект:

Project name: P0331_SharedPreferences
Build Target: Android 2.3.3
Application name: SharedPreferences
Package name: ru.startandroid.develop.p0331sharedpreferences
Create Activity: MainActivity

Откроем **main.xml** и создадим такой экран:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <EditText
          android:id="@+id/etText"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
         <requestFocus>
         </requestFocus>
     </EditText>
     <LinearLayout
          android:id="@+id/linearLayout1"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:orientation="horizontal">
         <Button
              android:id="@+id/btnSave"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Save">
        </Button>
        <Button
            android:id="@+id/btnLoad"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Load">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Поле ввода и две кнопки.

Теперь пишем код в **MainActivity.java**:

```
package ru.startandroid.develop.p0331sharedpreferences;

import android.app.Activity;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener {

  EditText etText;
  Button btnSave, btnLoad;

  SharedPreferences sPref;

  final String SAVED_TEXT = "saved_text";


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        etText = (EditText) findViewById(R.id.etText);

        btnSave = (Button) findViewById(R.id.btnSave);
        btnSave.setOnClickListener(this);

        btnLoad = (Button) findViewById(R.id.btnLoad);
        btnLoad.setOnClickListener(this);
    }

  @Override
  public void onClick(View v) {
```

```
    switch (v.getId()) {
    case R.id.btnSave:
      saveText();
      break;
    case R.id.btnLoad:
      loadText();
      break;
    default:
      break;
    }
  }

  void saveText() {
    sPref = getPreferences(MODE_PRIVATE);
    Editor ed = sPref.edit();
    ed.putString(SAVED_TEXT, etText.getText().toString());
    ed.commit();
    Toast.makeText(this, "Text saved", Toast.LENGTH_SHORT).show();
  }

  void loadText() {
    sPref = getPreferences(MODE_PRIVATE);
    String savedText = sPref.getString(SAVED_TEXT, "");
    etText.setText(savedText);
    Toast.makeText(this, "Text loaded", Toast.LENGTH_SHORT).show();
  }
}
```

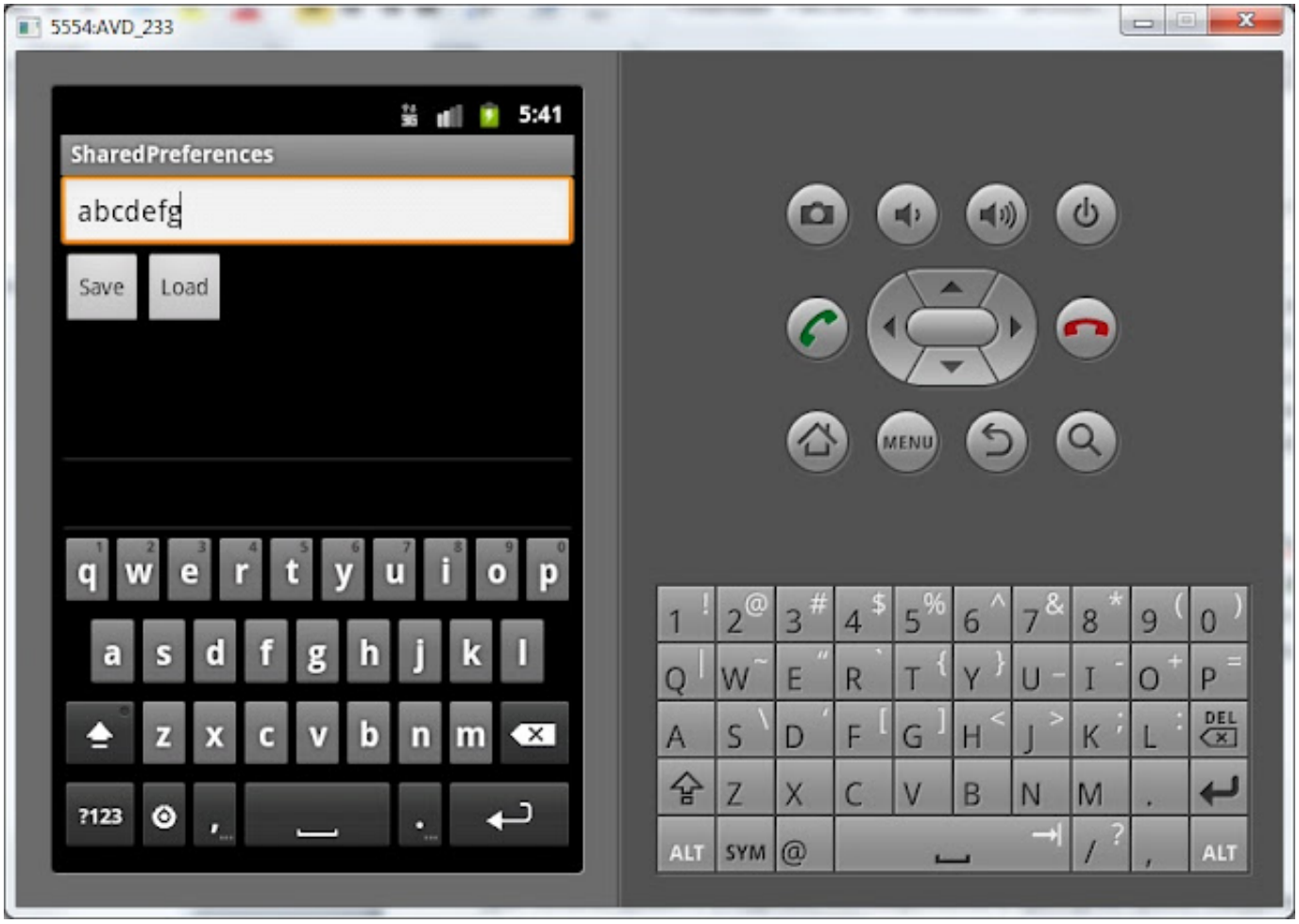
Определение элементов экрана, присвоение обработчиков и реализация onClick – тут все понятно и как обычно. Нам интересны **методы**, которые мы вызываем в onClick

saveText – сохранение данных. Сначала с помощью метода getPreferences получаем объект **sPref** класса **SharedPreferences**, который позволяет **работать** с **данными** (читать и писать). Константа MODE_PRIVATE используется для настройки **доступа** и означает, что после сохранения, данные будут видны только **этому** приложению. Далее, чтобы редактировать данные, необходим объект **Editor** – получаем его из sPref. В метод putString указываем **наименование** переменной – это константа **SAVED_TEXT**, и **значение** – содержимое поля **etText**. Чтобы данные сохранились, необходимо выполнить commit. И для наглядности выводим **сообщение**, что данные сохранены.

loadText – загрузка данных. Так же, как и saveText, с помощью метода **getPreferences** получаем объект sPref класса **SharedPreferences**. **MODE_PRIVATE** снова указывается, хотя и используется только при записи данных. Здесь Editor мы не используем, т.к. нас интересует только чтение данных. Читаем с помощью метода getString – в параметрах указываем константу - это **имя,** и **значение** по умолчанию (пустая строка). Далее пишем значение в поле ввода **etText** и выводим **сообщение**, что данные считаны.

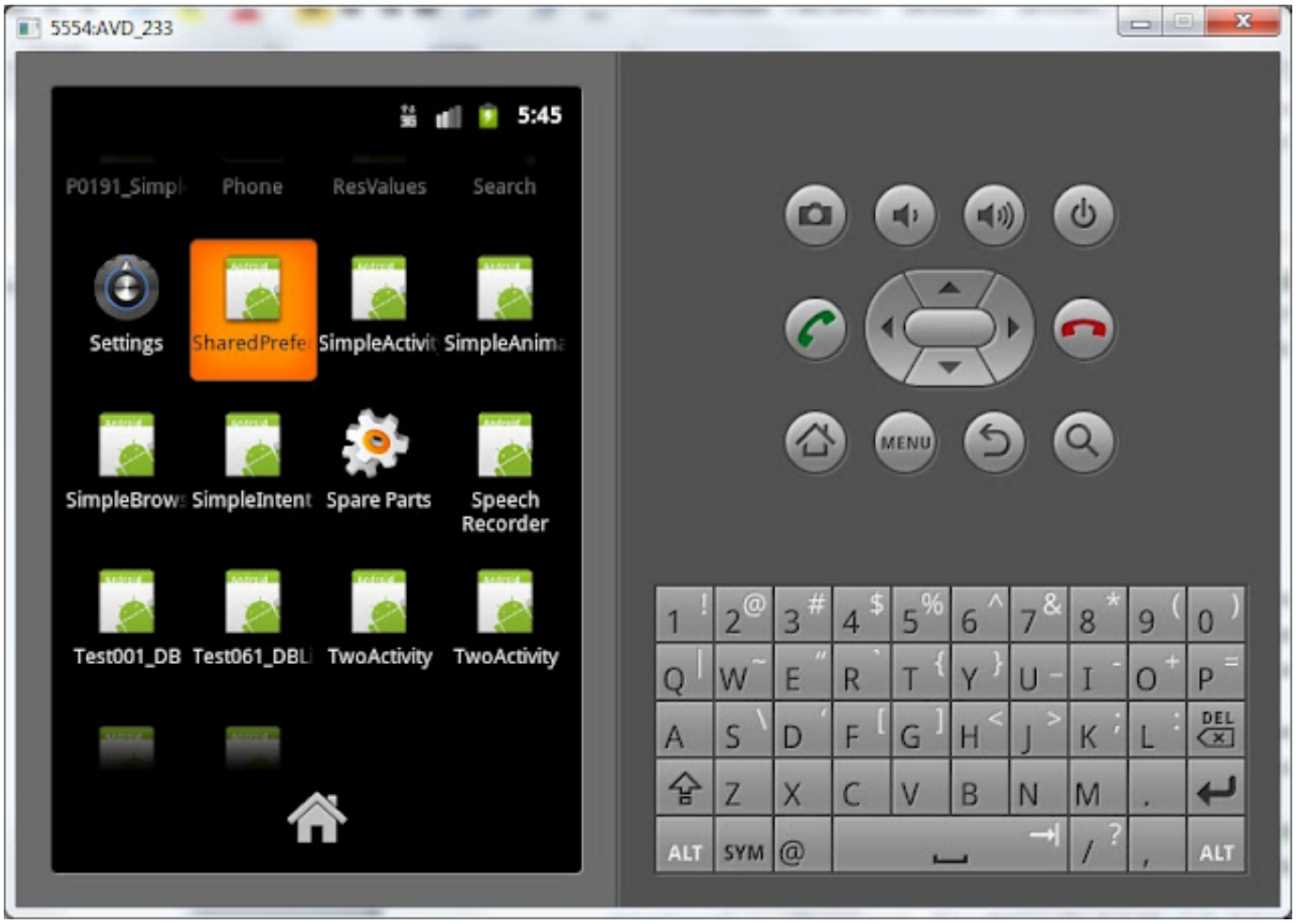
Все сохраняем, запускаем приложение.

Для начала, давайте убедимся, что сохранение в принципе нужно. Введите какой-нить текст в поле ввода



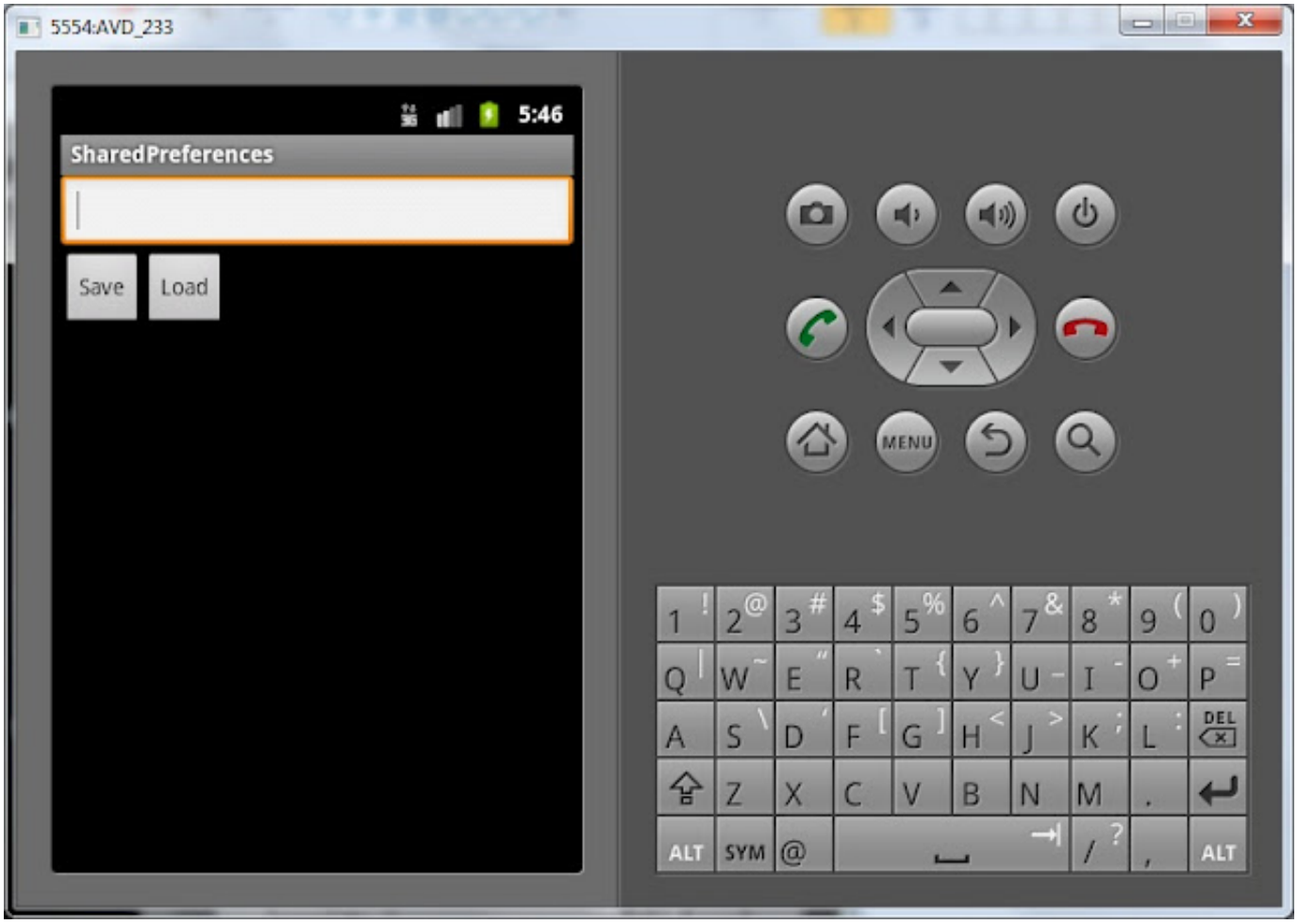
и не нажимая кнопку **Save** закройте приложение кнопкой **Назад**.

Теперь найдите приложение в общем списке приложений эмулятора

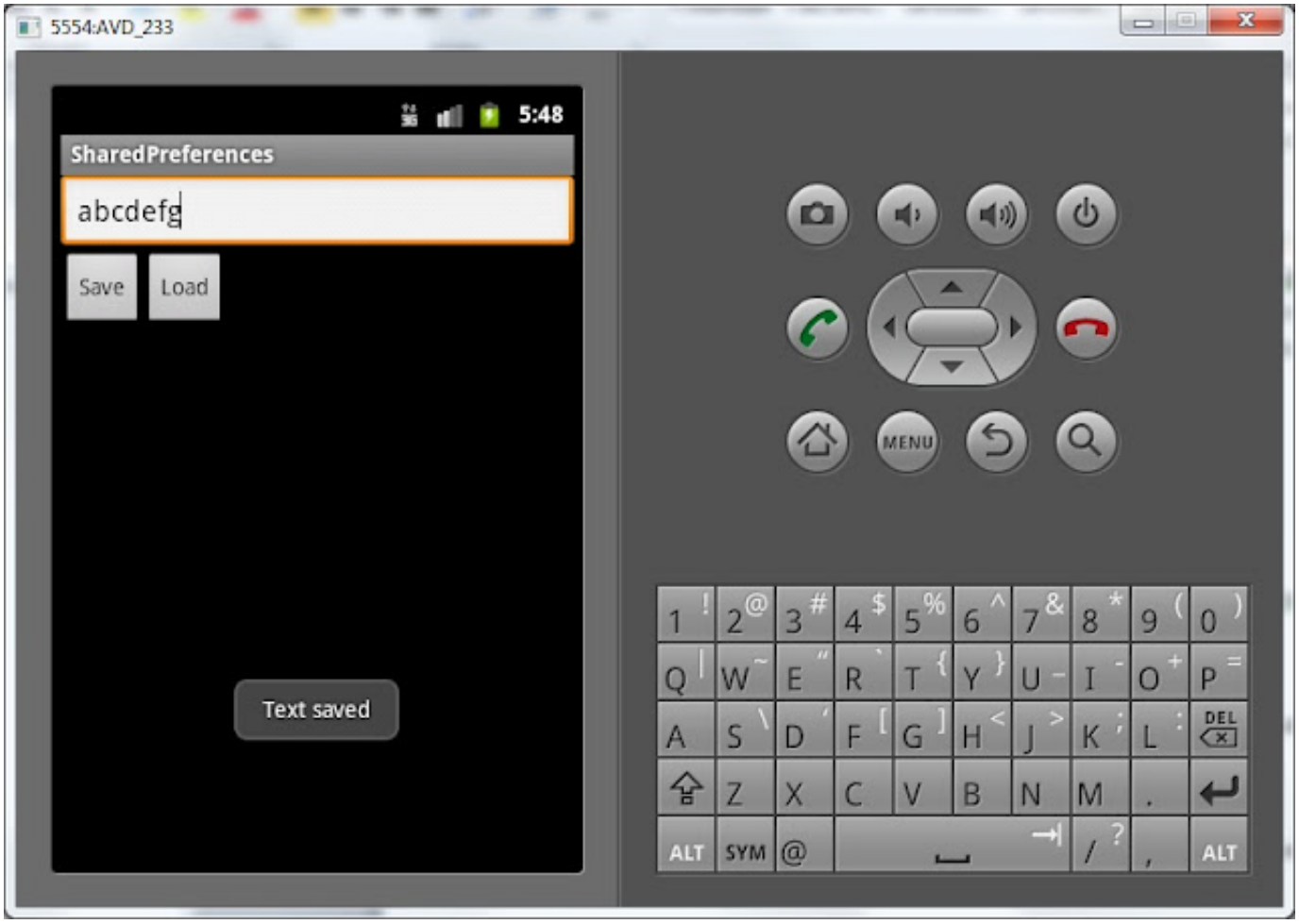


и запустите снова.

Поле ввода пустое. То, что мы вводили – пропало при закрытии программы. Нажатие на **Load** тоже ничего не даст – мы ничего не сохраняли.

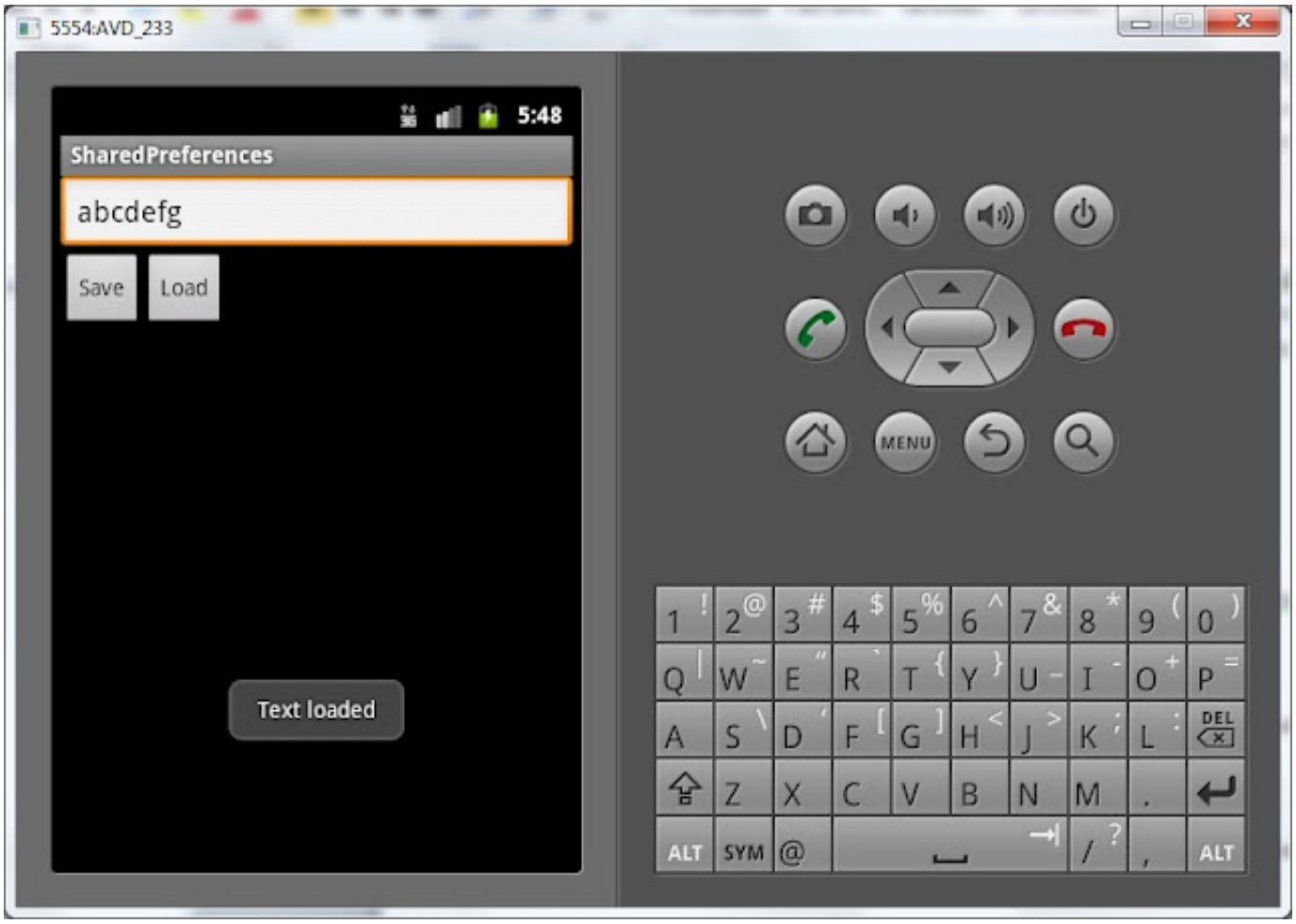


Давайте попробуем сохранять. Снова введите значение и нажмите **Save**.



Значение сохранилось в системе.

Теперь закроем приложение (**Назад**), снова откроем и нажмем **Load**. Значение считалось и отобразилось.



Давайте сделаем так, чтобы сохранение и загрузка происходили **автоматически** при **закрытии** и **открытии** приложения и не надо было жать кнопки. Для этого метод **loadText** будем вызывать в **onCreate**.

```
public void onCreate(Bundle savedInstanceState) {

    ...

    btnLoad = (Button) findViewById(R.id.btnLoad);
    btnLoad.setOnClickListener(this);

    loadText();
}
```

(Добавляете только жирный курсивный текст)

а метод saveText - в onDestroy

```
  @Override
  protected void onDestroy() {
    saveText();
    super.onDestroy();
  }
```

Все сохраним, запустим. Теперь можно вводить данные, закрывать приложение, снова открывать и данные не потеряются. Кнопки **Save** и **Load** также работают. В какой момент сохранять данные в ваших приложениях – решать только вам. По **нажатию** кнопки, при **закрытии** программы или еще по какому-либо событию. Главное – теперь вы это умеете.

Еще немного слов по этой теме.

Preferences-данные сохраняются в файлы и вы можете посмотреть их. Для этого в Eclipse откройте меню **Window** > **Show View** > **Other** и выберите **Android** > **File Explorer**. Отобразилась файловая система эмулятора. Открываем *data/data/ru.startandroid.develop.p0331sharedpreferences/shared_prefs* и видим там файл **MainActivity.xml**. Если его выгрузить на комп и открыть - увидим следующее:

```
<?xml version="1.0" encoding="utf-8"?>
<map>
    <string  name="saved_text">abcdefg</string>
</map>
```

Все верно, имя - *saved_text* и значение - *abcdefg*.

Обратите внимание, что в пути к файлу используется наш package.

Теперь разберемся, откуда взялось наименование файла **MainActivity.xml**. Кроме метода **getPreferences**, который мы использовали, есть метод getSharedPreferences. Он выполняет абсолютно те же функции, но позволяет указать

имя файла для хранения данных. Т.е., например, если бы мы в **saveText** использовали для получение SharedPreferences такой код:

```
sPref = getSharedPreferences("MyPref", MODE_PRIVATE);
```

То данные сохранились бы в файле **MyPref.xml**, а не в **MainActivity.xml**.

Теперь если мы посмотрим исходники метода **getPreferences**, то видим следующее:

```
public SharedPreferences getPreferences(int mode) {
  return getSharedPreferences(getLocalClassName(), mode);
}
```

Используется метод **getSharedPreferences**, а в качестве имени файла берется имя класса текущего **Activity**. Отсюда и появилось имя файла **MainActivity.xml**.

В итоге:

- используете **getPreferences**, если работаете с данными для текущего Activity и не хотите выдумывать имя файла.

- используете **getSharedPreferences**, если сохраняете, например, данные - общие для нескольких Activity и сами выбираете имя файла для сохранения.

Кстати, в File Explorer вы можете видеть юниксовые rwx-права доступа к файлу. Попробуйте при сохранении данных использовать не **MODE_PRIVATE**, а MODE_WORLD_READABLE или MODE_WORLD_WRITEABLE и посмотрите, как будут меняться права.

Полный код **MainActivity.java**:

```
package ru.startandroid.develop.p0331sharedpreferences;

import android.app.Activity;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity implements OnClickListener {

  EditText etText;
  Button btnSave, btnLoad;

  SharedPreferences sPref;

  final String SAVED_TEXT = "saved_text";
```

```
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        etText = (EditText) findViewById(R.id.etText);

        btnSave = (Button) findViewById(R.id.btnSave);
        btnSave.setOnClickListener(this);

        btnLoad = (Button) findViewById(R.id.btnLoad);
        btnLoad.setOnClickListener(this);

        loadText();
    }

  @Override
  public void onClick(View v) {
    switch (v.getId()) {
    case R.id.btnSave:
      saveText();
      break;
    case R.id.btnLoad:
      loadText();
      break;
    default:
      break;
    }
  }

  void saveText() {
    sPref = getPreferences(MODE_PRIVATE);
    Editor ed = sPref.edit();
    ed.putString(SAVED_TEXT, etText.getText().toString());
    ed.commit();
    Toast.makeText(this, "Text saved", Toast.LENGTH_SHORT).show();
  }

  void loadText() {
    sPref = getPreferences(MODE_PRIVATE);
    String savedText = sPref.getString(SAVED_TEXT, "");
    etText.setText(savedText);
    Toast.makeText(this, "Text loaded", Toast.LENGTH_SHORT).show();
  }

  @Override
  protected void onDestroy() {
    saveText();
    super.onDestroy();
  }

}
```

На следующем уроке:

- хранение данных с помощью SQLite

Урок 34. Хранение данных. SQLite

В этом уроке:

- хранение данных с помощью SQLite

На прошлом уроке мы рассмотрели самый простой способ хранения данных - **Preferences**. Но способ этот достаточно ограничен и для хранения большого количества структурированных данных неудобен. На этом уроке рассмотрим SQLite. Это база данных с таблицами и запросами - все как в обычных БД.

Для начала, немного теории по взаимодействию приложения и БД.

В приложении, при подключении к БД мы указываем **имя** БД и **версию**. При этом могут возникнуть следующие ситуации:

1) БД **не существует**. Это может быть например в случае первичной установки программы. В этом случае приложение должно само **создать** БД и все таблицы в ней. И далее оно уже работает с только что созданной БД.

2) БД **существует**, но ее версия **устарела**. Это может быть в случае обновления программы. Например новой версии программы нужны дополнительные поля в старых таблицах или новые таблицы. В этом случае приложение должно **апдейтить** существующие таблицы и создать новые, если это необходимо.

3) БД **существует** и ее версия **актуальна**. В этом случае приложение успешно **подключается** к БД и работает.

Как вы понимаете, фраза "приложение должно" равнозначна фразе "разработчик должен", т.е. это наша задача. Для обработки описанных выше ситуаций нам надо создать **класс**, являющийся наследником для SQLiteOpenHelper. Назовем его DBHelper. Этот класс предоставит нам методы для **создания** или **обновления** БД в случаях ее **отсутствия** или **устаревания**.

onCreate - метод, который будет вызван, если БД, к которой мы хотим подключиться – не существует

onUpgrade - будет вызван в случае, если мы пытаемся подключиться к БД более новой версии, чем существующая

Давайте накидаем простое приложение – справочник контактов, которое будет хранить **имя** и **email**. **Вводить** данные будем на **экране** приложения, а для **отображения** информации используем **логи**. Обычно для этого используется List (список) – но мы эту тему пока не знаем. Да и не хочется перегружать приложение. Главное – освоить приемы работы с БД.

Создадим проект:

Project name: P0341_SimpleSQLite
Build Target: Android 2.3.3
Application name: SimpleSQLite
Package name: ru.startandroid.develop.p0341simplesqlite
Create Activity: MainActivity

Нарисуем экран для ввода записей и очистки таблицы. Открываем **main.xml** и пишем:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
```

```
 android:orientation="vertical">
<LinearLayout
     android:id="@+id/linearLayout1"
     android:layout_width="match_parent"
     android:layout_height="wrap_content">
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Name"
         android:layout_marginLeft="5dp"
         android:layout_marginRight="5dp">
    </TextView>
    <EditText
         android:id="@+id/etName"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_weight="1">
        <requestFocus>
        </requestFocus>
    </EditText>
</LinearLayout>
<LinearLayout
     android:id="@+id/linearLayout3"
     android:layout_width="match_parent"
     android:layout_height="wrap_content">
    <TextView
         android:id="@+id/textView2"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Email"
         android:layout_marginLeft="5dp"
         android:layout_marginRight="5dp">
    </TextView>
    <EditText
         android:id="@+id/etEmail"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_weight="1">
    </EditText>
</LinearLayout>
<LinearLayout
     android:id="@+id/linearLayout2"
     android:layout_width="match_parent"
     android:layout_height="wrap_content">
    <Button
         android:id="@+id/btnAdd"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Add">
    </Button>
    <Button
         android:id="@+id/btnRead"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
            android:text="Read">
        </Button>
        <Button
            android:id="@+id/btnClear"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Clear">
        </Button>
    </LinearLayout>
</LinearLayout>
```

Пара полей для ввода и кнопки добавления записи, вывода существующих записей и очистки таблицы.

Открываем **MainActivity.java** и пишем:

```
package ru.startandroid.develop.p0341simplesqlite;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity implements OnClickListener {

  final String LOG_TAG = "myLogs";

  Button btnAdd, btnRead, btnClear;
  EditText etName, etEmail;

  DBHelper dbHelper;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    btnAdd = (Button) findViewById(R.id.btnAdd);
    btnAdd.setOnClickListener(this);

    btnRead = (Button) findViewById(R.id.btnRead);
    btnRead.setOnClickListener(this);

    btnClear = (Button) findViewById(R.id.btnClear);
    btnClear.setOnClickListener(this);

    etName = (EditText) findViewById(R.id.etName);
    etEmail = (EditText) findViewById(R.id.etEmail);
```

```
    // создаем объект для создания и управления версиями БД
    dbHelper = new DBHelper(this);
}


@Override
public void onClick(View v) {

    // создаем объект для данных
    ContentValues cv = new ContentValues();

    // получаем данные из полей ввода
    String name = etName.getText().toString();
    String email = etEmail.getText().toString();

    // подключаемся к БД
    SQLiteDatabase db = dbHelper.getWritableDatabase();


    switch (v.getId()) {
    case R.id.btnAdd:
      Log.d(LOG_TAG, "--- Insert in mytable: ---");
      // подготовим данные для вставки в виде пар: наименование столбца - значение

      cv.put("name", name);
      cv.put("email", email);
      // вставляем запись и получаем ее ID
      long rowID = db.insert("mytable", null, cv);
      Log.d(LOG_TAG, "row inserted, ID = " + rowID);
      break;
    case R.id.btnRead:
      Log.d(LOG_TAG, "--- Rows in mytable: ---");
      // делаем запрос всех данных из таблицы mytable, получаем Cursor
      Cursor c = db.query("mytable", null, null, null, null, null, null);

      // ставим позицию курсора на первую строку выборки
      // если в выборке нет строк, вернется false
      if (c.moveToFirst()) {

        // определяем номера столбцов по имени в выборке
        int idColIndex = c.getColumnIndex("id");
        int nameColIndex = c.getColumnIndex("name");
        int emailColIndex = c.getColumnIndex("email");

        do {
          // получаем значения по номерам столбцов и пишем все в лог
          Log.d(LOG_TAG,
              "ID = " + c.getInt(idColIndex) +
              ", name = " + c.getString(nameColIndex) +
              ", email = " + c.getString(emailColIndex));
          // переход на следующую строку
          // а если следующей нет (текущая - последняя), то false - выходим из цикла
        } while (c.moveToNext());
      } else
        Log.d(LOG_TAG, "0 rows");
      break;
    case R.id.btnClear:
      Log.d(LOG_TAG, "--- Clear mytable: ---");
      // удаляем все записи
```

```
        int clearCount = db.delete("mytable", null, null);
        Log.d(LOG_TAG, "deleted rows count = " + clearCount);
        break;
      }
      // закрываем подключение к БД
      dbHelper.close();
    }



    class DBHelper extends SQLiteOpenHelper {

      public DBHelper(Context context) {
        // конструктор суперкласса
        super(context, "myDB", null, 1);
      }

      @Override
      public void onCreate(SQLiteDatabase db) {
        Log.d(LOG_TAG, "--- onCreate database ---");
        // создаем таблицу с полями
        db.execSQL("create table mytable ("
            + "id integer primary key autoincrement,"
            + "name text,"
            + "email text" + ");");
      }

      @Override
      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

      }
    }

}
```

Куча новых незнакомых слов в коде. Давайте разбираться.

В методе Activity - **onCreate** мы определяем объекты, присваиваем обработчики и создаем объект dbHelper класса **DBHelper** для управления БД. Сам класс будет описан ниже.

Далее смотрим метод Activity – **onClick**, в котором мы обрабатываем нажатия на кнопки.

Класс ContentValues используется для указания **полей** таблицы и **значений**, которые мы в эти поля будем вставлять. Мы создаем объект **cv**, и позже его используем. Далее мы записываем в переменные значения из **полей** ввода. Затем, с помощью метода getWritableDatabase подключаемся к БД и получаем объект SQLiteDatabase. Он позволит нам работать с БД. Мы будем использовать его методы **insert** – вставка записи, **query** – чтение, **delete** – удаление. У них много разных параметров на вход, но мы пока используем самый минимум.

Далее смотрим, какая кнопка была нажата:

btnAdd – добавление записи в таблицу *mytable*. Мы заполняем объект **cv** парами: **имя** поля и **значение**. И (при вставке записи в таблицу) в указанные поля будут вставлены соответствующие значения. Мы заполняем поля *name* и *email*. *id* у нас заполнится автоматически (primary key autoincrement). Вызываем метод insert – передаем ему **имя таблицы** и объект **cv** с вставляемыми значениями. Второй аргумент метода используется, при вставке в таблицу пустой строки. Нам это сейчас не нужно, поэтому передаем null. Метод **insert** возвращает **ID** вставленной строки, мы

его сохраняем в **rowID** и выводим в **лог**.

btnRead – чтение всех записей из таблицы *mytable*. Для чтения используется метод query. На вход ему подается **имя таблицы**, список запрашиваемых полей, условия выборки, группировка, сортировка. Т.к. нам нужны все данные во всех полях без сортировок и группировок - мы используем везде **null**. Только имя таблицы указываем. Метод возвращает нам объект класса Cursor. Его можно рассматривать как таблицу с данными. Метод moveToFirst – делает **первую** запись в Cursor **активной** и заодно проверяет, есть ли вообще записи в нем (т.е. выбралось ли что-либо в методе **query**). Далее мы получаем порядковые номера столбцов в Cursor по их именам с помощью метода getColumnIndex. Эти номера потом используем для чтения данных в методах getInt и getString и выводим данные в лог. С помощью метода moveToNext мы перебираем все строки в Cursor пока не добираемся до последней. Если же записей не было, то выводим в лог соответствующее сообщение – *0 rows*.

btnClear – очистка таблицы. Метод delete удаляет записи. На вход передаем **имя таблицы** и **null** в качестве условий для удаления, а значит удалится все. Метод возвращает **кол-во удаленных** записей.

После этого закрываем соединение с БД методом close.

Класс **DBHelper** является **вложенным** в **MainActivity** и описан в конце кода. Как я уже писал выше, этот класс должен наследовать класс **SQLiteOpenHelper**.

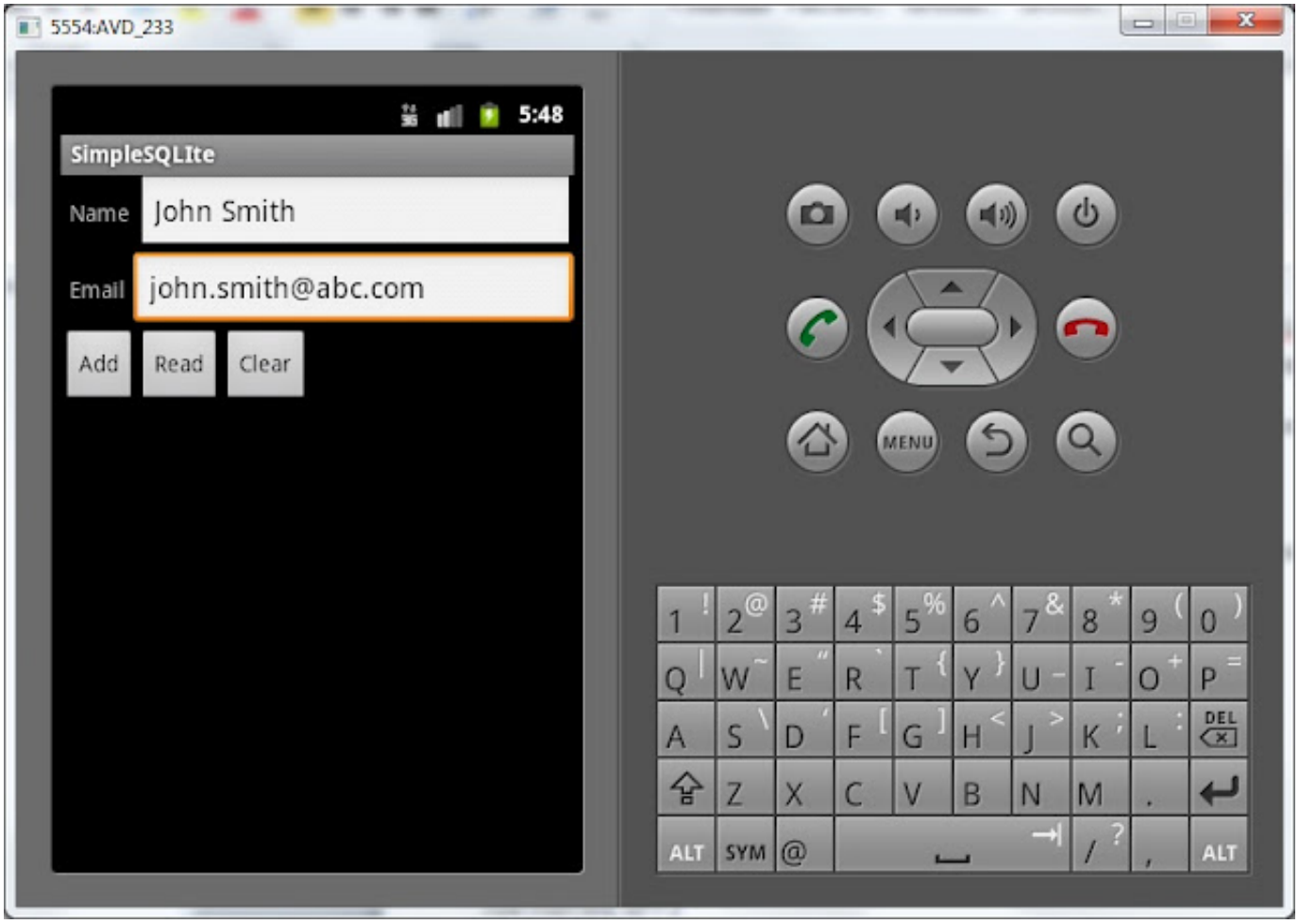
В **конструкторе** мы вызываем конструктор суперкласса и передаем ему:
context - контекст
mydb - название базы данных
null – объект для работы с курсорами, нам пока не нужен, поэтому null
1 – версия базы данных

В методе **onCreate** этого класса мы используем метод execSQL объекта SQLiteDatabase для выполнения SQL-запроса, который создает таблицу. Напомню – этот метод вызывается, если БД не существует и ее надо создавать. По запросу видно, что мы создаем таблицу *mytable* с полями *id*, *name* и *email*.

Метод **onUpgrade** пока не заполняем, т.к. используем одну версию БД и менять ее не планируем.

Все сохраним и запустим приложение. Будем работать с БД и смотреть логи, которые покажут, какие методы выполняются, и что в них происходит.

Введем чего-нить в поля ввода и нажмем **Add**.



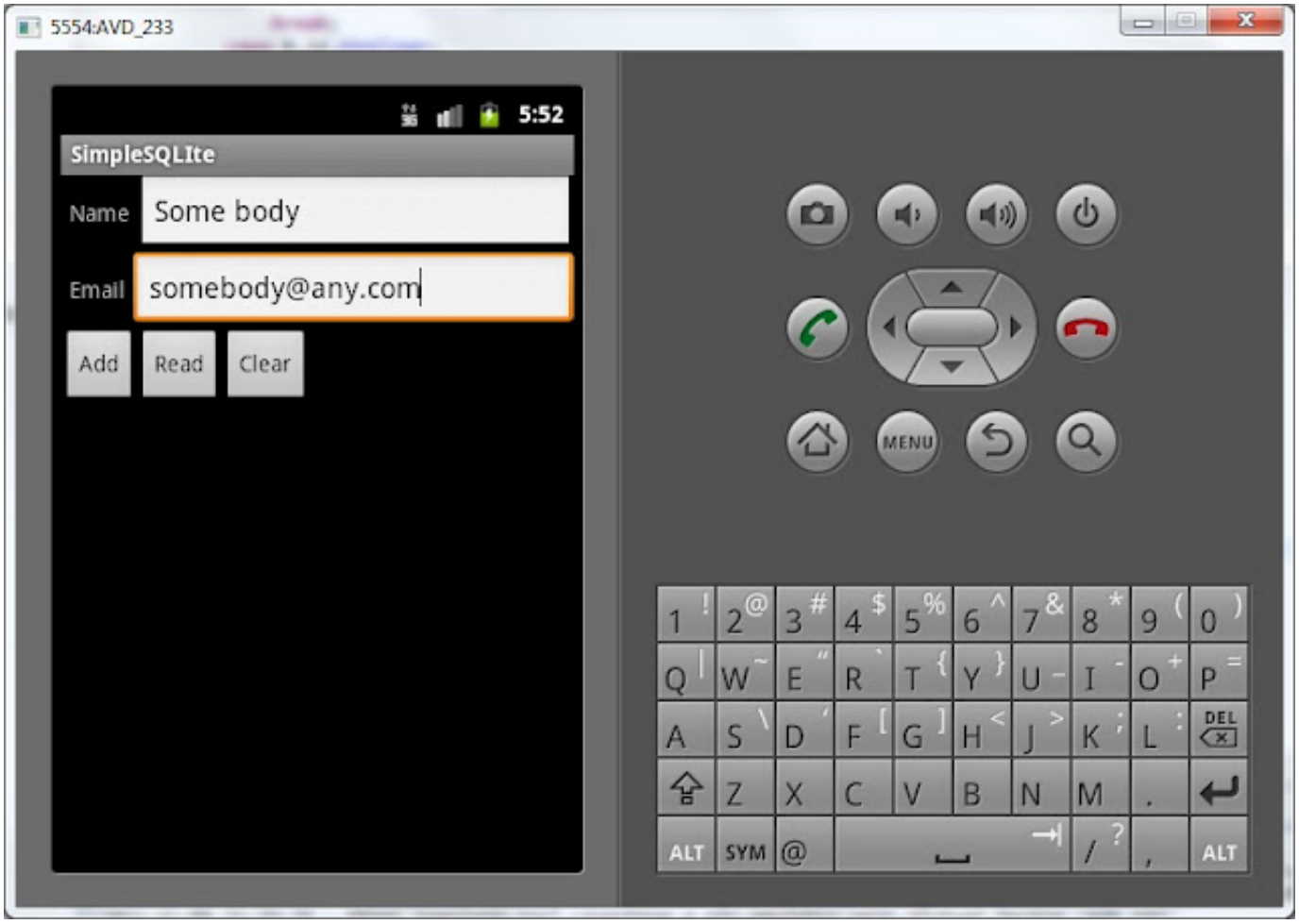
Смотрим лог:

--- onCreate database ---
--- Insert in mytable: ---
row inserted, ID = 1

Мы видим, что вызывался метод **onCreate** в классе **DBHelper**, а значит выполнялся скрипт по **созданию** таблицы. Это произошло потому, что это **первый** запуск приложения и БД еще **не была создана**. Теперь БД существует и с ней можно работать.

Далее видим, что вызывался метод вставки записи и вернул **ID** = 1.

Вставим еще какую-нибудь запись.



Смотрим лог:

--- Insert in mytable: ---
row inserted, ID = 2

На этот раз **onCreate не вызывался**, т.к. БД уже **существует**. Вставилась запись с ID = 2.

Давайте посмотрим содержимое таблицы - нажмем кнопку **Read** и посмотрим лог:

--- Rows in mytable: ---
ID = 1, name = John Smith, email = john.smith@abc.com
ID = 2, name = Some body, email = somebody@any.com

Мы видим записи, которые вставляли. Тут все верно.

Теперь очистим таблицу - нажмем **Clear**. Смотрим лог:

--- Clear mytable: ---
deleted rows count = 2

Удалено две записи, все верно. Если теперь посмотрим содержимое таблицы – кнопка Read:

--- Rows in mytable: ---
0 rows

Записей нет.

В этой теме важно понять, что для работы с БД мы использовали два класса:

- **DBHelper**, наследующий **SQLiteOpenHelper**. В его **конструкторе** мы вызываем конструктор супер-класса и указываем имя и версию БД. Метод **getWritableDatabase** выполняет подключение к базе данных и возвращает нам объект **SQLiteDatabase** для работы с ней. Метод **close** закрывает подключение к БД. В случае, когда БД отсутствует или устарела, класс предоставляет нам самим реализовать создание или обновление в методах **onCreate** и **onUpgrate**.

- **SQLiteDatabase**. Содержит методы для работы с данными – т.е. **вставка**, **обновление**, **удаление** и **чтение**.

Файл базы можно найти в **File Explorer**, как и на прошлом уроке. Путь к нему *data/data/ru.startandroid.develop.p0341simpelsqlite/databases/myDB*.

На следующем уроке продолжим это приложение. Добавим возможность обновления и удаления конкретных записей.

На следующем уроке:

- используем методы query и delete с указанием условия

Урок 35. SQLite. Методы update и delete с указанием условия

В этом уроке:

- используем методы query и delete с указанием условия

На прошлом уроке мы разобрали, как **вставить** запись, **считать** все записи из таблицы и **очистить** таблицу. Теперь посмотрим, как **обновить** и **удалить** конкретную запись.

Новый проект создавать не будем, используем **P0341_SimpleSQLite** с прошлого урока. Немного поменяем экран, добавим **поле** для ввода **ID** и **кнопки** для **обновления** и **удаления**.

Перепишем **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <LinearLayout
         android:id="@+id/linearLayout4"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:layout_marginTop="5dp">
        <TextView
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="ID"
             android:layout_marginLeft="5dp"
             android:layout_marginRight="25dp">
        </TextView>
        <EditText
             android:id="@+id/etID"
             android:layout_width="70dp"
             android:layout_height="wrap_content"
             android:layout_marginTop="2dp">
        </EditText>
        <Button
             android:id="@+id/btnUpd"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="Update">
        </Button>
        <Button
             android:id="@+id/btnDel"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="Delete">
        </Button>
    </LinearLayout>
```

```
<LinearLayout
     android:id="@+id/linearLayout1"
     android:layout_width="match_parent"
     android:layout_height="wrap_content">
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Name"
         android:layout_marginLeft="5dp"
         android:layout_marginRight="5dp">
    </TextView>
    <EditText
         android:id="@+id/etName"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_weight="1">
        <requestFocus>
        </requestFocus>
    </EditText>
</LinearLayout>
<LinearLayout
     android:id="@+id/linearLayout3"
     android:layout_width="match_parent"
     android:layout_height="wrap_content">
    <TextView
         android:id="@+id/textView2"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Email"
         android:layout_marginLeft="5dp"
         android:layout_marginRight="5dp">
    </TextView>
    <EditText
         android:id="@+id/etEmail"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_weight="1">
    </EditText>
</LinearLayout>
<LinearLayout
     android:id="@+id/linearLayout2"
     android:layout_width="match_parent"
     android:layout_height="wrap_content">
    <Button
         android:id="@+id/btnAdd"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Add">
    </Button>
    <Button
         android:id="@+id/btnRead"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Read">
```

```
        </Button>
        <Button
            android:id="@+id/btnClear"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Clear">
        </Button>
    </LinearLayout>
</LinearLayout>
```

По нажатию кнопки **Update** мы будем читать содержимое полей **Name** и **Email**, и обновлять запись в таблице, для которой **id** = значению из поля **ID**. По нажатию кнопки **Delete** будем удалять запись из таблицы по **id** = значению из поля **ID**. Экран получился, конечно, не самый лучший с точки зрения эргономики и юзабилити, но у нас тут не кружок юных дизайнеров, а серьезное изучение БД.

Подредактируем **MainActivity.java**. Добавим описание и определение новых экранных элементов, присвоение обработчиков для кнопок.

```
final String LOG_TAG = "myLogs";

Button btnAdd, btnRead, btnClear, btnUpd, btnDel;
EditText etName, etEmail, etID;

...

public void onCreate(Bundle savedInstanceState) {

  ...

  btnClear = (Button) findViewById(R.id.btnClear);
  btnClear.setOnClickListener(this);

  btnUpd = (Button) findViewById(R.id.btnUpd);
  btnUpd.setOnClickListener(this);

  btnDel = (Button) findViewById(R.id.btnDel);
  btnDel.setOnClickListener(this);

  etName = (EditText) findViewById(R.id.etName);
  etEmail = (EditText) findViewById(R.id.etEmail);
  etID = (EditText) findViewById(R.id.etID);

  // создаем объект для создания и управления версиями БД
  dbHelper = new DBHelper(this);
}
```

(Добавляете только жирный курсивный текст. Если не видно, увеличьте масштаб)

Теперь дополним реализацию **onClick**:

```
public void onClick(View v) {

  // создаем объект для данных
```

```
    ContentValues cv = new ContentValues();

    // получаем данные из полей ввода
    String name = etName.getText().toString();
    String email = etEmail.getText().toString();
    String id = etID.getText().toString();

    // подключаемся к БД
    SQLiteDatabase db = dbHelper.getWritableDatabase();

    switch (v.getId()) {
    case R.id.btnAdd:
      ...
    case R.id.btnRead:
      ...
    case R.id.btnClear:
      ...
    case R.id.btnUpd:
      if (id.equalsIgnoreCase("")) {
        break;
      }
      Log.d(LOG_TAG, "--- Update mytabe: ---");
      // подготовим значения для обновления
      cv.put("name", name);
      cv.put("email", email);
      // обновляем по id
      int updCount = db.update("mytable", cv, "id = ?",
          new String[] { id });
      Log.d(LOG_TAG, "updated rows count = " + updCount);
      break;
    case R.id.btnDel:
      if (id.equalsIgnoreCase("")) {
        break;
      }
      Log.d(LOG_TAG, "--- Delete from mytabe: ---");
      // удаляем по id
      int delCount = db.delete("mytable", "id = " + id, null);
      Log.d(LOG_TAG, "deleted rows count = " + delCount);
      break;
    }
    // закрываем подключение к БД
    dbHelper.close();
  }
```

(Добавляете только жирный курсивный текст)

Мы добавляем переменную **id**, пишем в нее значение поля **etID**. В **switch** добавляем две новые ветки:

btnUpd – **обновление** записи в mytable. Проверяем, что значение **id** не пустое, заполняем **cv** данными для апдейта и **обновляем** запись. Для этого используется метод **update**. На вход ему подается **имя** таблицы, заполненный **ContetnValues** с значениями для обновления, строка **условия** (Where) и массив **аргументов** для строки условия. В строке условия я использовал знак *?*. При запросе к БД вместо этого знака будет подставлено значение из массива **аргументов**, в нашем случае это – значение переменной **id**. Если знаков *?* в строке условия несколько, то им будут сопоставлены значения из массива по порядку. Метод **update** возвращает нам **кол-во обновленных** записей, которое мы выводим в лог.

btnDel – **удаление** записи из mytable. Проверяем, что **id** не пустое и вызываем метод **delete**. На вход передаем **имя**

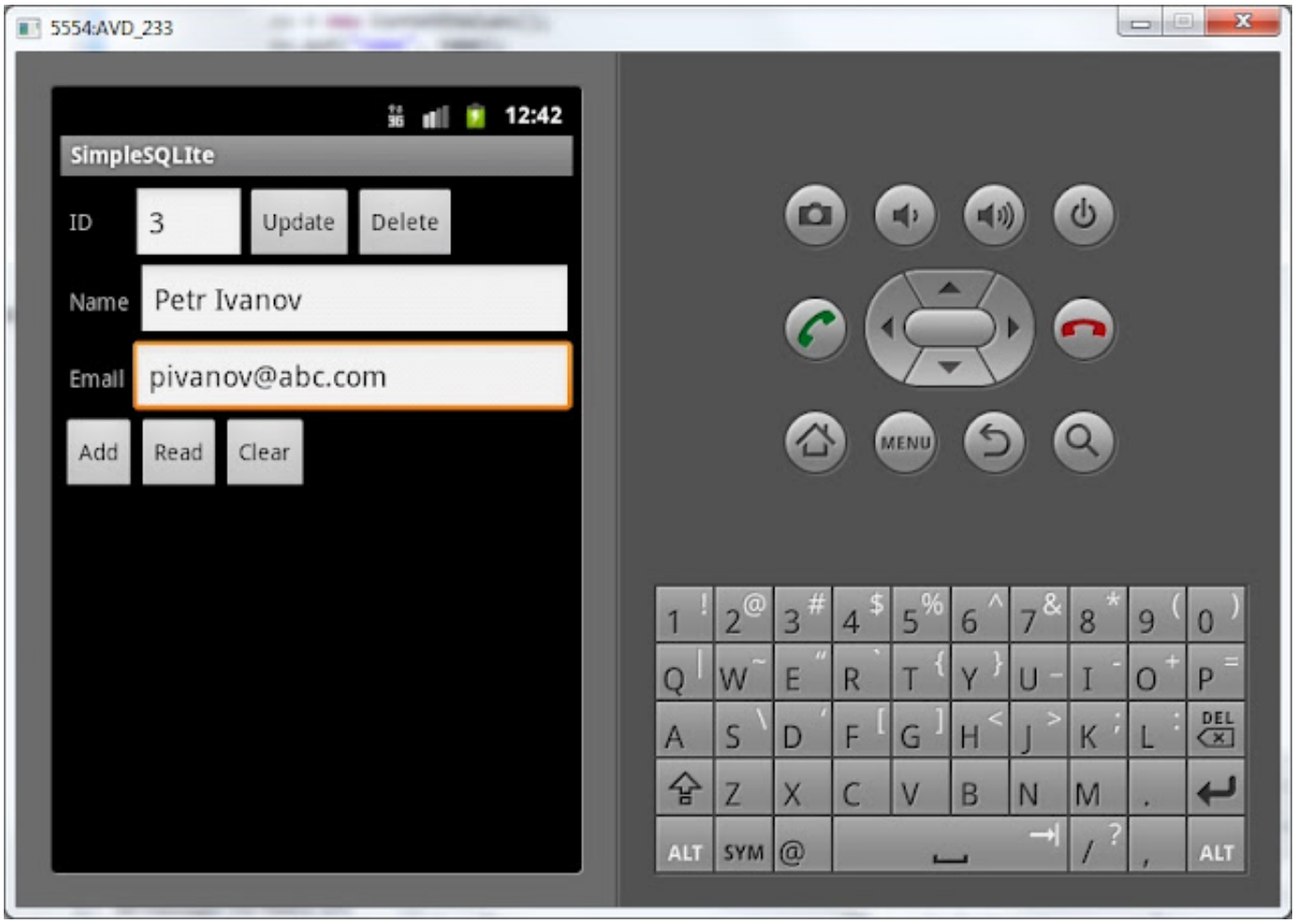
таблицы, строку **условия** и массив **аргументов** для условия. Метод delete возвращает кол-во удаленных строк, которое мы выводим в лог.

Обратите внимание, что **условия** и для **update** и для **delete** у меня одинаковые, а именно ***id*** = *значение из поля* ***etID***. Но реализовал я их немного по-разному. Для **update** использовал символ *?* в строке условия и **массив** аргументов. А для **delete** вставил значение сразу в строку **условия**. Таким образом, я просто показал **способы формирования условия**. А вы уже используйте тот, что больше нравится или лучше в конкретной ситуации.

Все сохраним и запустим. На прошлом уроке мы добавляли пару записей, но потом их удалили. Так что я буду добавлять снова. **Добавлю** пару записей, нажму **Read**, вижу в логе:

ID = 3, name = Ivan Petrov, email = ipetrov@abc.com
ID = 4, name = Anton Sidorov, email = asidorov@def.com

Теперь попробуем обновить запись с **ID**=3. Для этого вводим 3 в поле **ID** и новые данные в поля **Name** и **Email**:



Жмем **Update**, смотрим лог:

-- Update mytabe: ---
updated rows count = 1

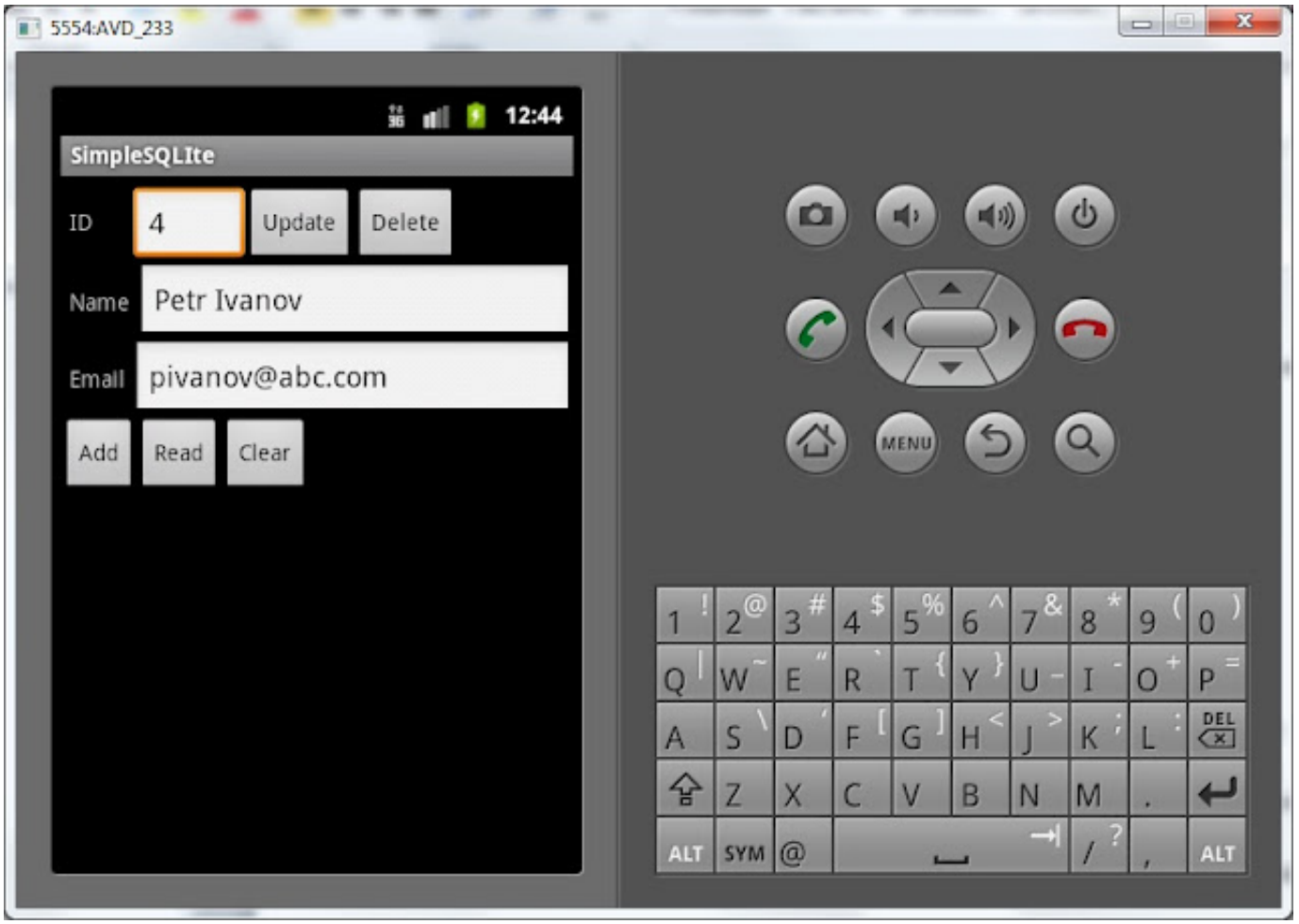
обновилась одна запись, все верно.

Нажмем **Read** и убедимся в этом. Лог:

ID = 3, name = Petr Ivanov, email = pivanov@abc.com
ID = 4, name = Anton Sidorov, email = asidorov@def.com

Запись действительно обновилась.

Теперь давайте удалим запись с **ID** = 4. Вводим 4 в поле **ID**



Жмем **Delete**, смотрим лог:

--- Delete from mytabe: ---
deleted rows count = 1

одна запись удалена.

Жмем **Read**, смотрим лог:

--- Rows in mytable: ---
ID = 3, name = Petr Ivanov, email = pivanov@abc.com

осталась одна запись.

Если попробовать удалить запись с **пустым** полем **ID**, то ничего не будет, т.к. мы реализовали проверку. Если же попробовать удалить запись с несуществующим **ID**, то метод **delete** вернет **0**. Т.е. ничего не было удалено, т.к. не нашлось записей для удаления.

Теперь вы умеете **читать**, **вставлять**, **удалять** и **изменять** записи в SQLite.

Полный код **MainActivity.java**:

```java
package ru.startandroid.develop.p0341simplesqlite;

import ru.startandroid.develop.p0341simpelsqlite.R;
import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity implements OnClickListener {

  final String LOG_TAG = "myLogs";

  Button btnAdd, btnRead, btnClear, btnUpd, btnDel;
  EditText etName, etEmail, etID;

  DBHelper dbHelper;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    btnAdd = (Button) findViewById(R.id.btnAdd);
    btnAdd.setOnClickListener(this);

    btnRead = (Button) findViewById(R.id.btnRead);
    btnRead.setOnClickListener(this);

    btnClear = (Button) findViewById(R.id.btnClear);
    btnClear.setOnClickListener(this);

    btnUpd = (Button) findViewById(R.id.btnUpd);
    btnUpd.setOnClickListener(this);

    btnDel = (Button) findViewById(R.id.btnDel);
    btnDel.setOnClickListener(this);

    etName = (EditText) findViewById(R.id.etName);
    etEmail = (EditText) findViewById(R.id.etEmail);
    etID = (EditText) findViewById(R.id.etID);

    // создаем объект для создания и управления версиями БД
    dbHelper = new DBHelper(this);
  }

  public void onClick(View v) {

    // создаем объект для данных
    ContentValues cv = new ContentValues();
```

```
// получаем данные из полей ввода
String name = etName.getText().toString();
String email = etEmail.getText().toString();
String id = etID.getText().toString();

// подключаемся к БД
SQLiteDatabase db = dbHelper.getWritableDatabase();

switch (v.getId()) {
case R.id.btnAdd:
  Log.d(LOG_TAG, "--- Insert in mytable: ---");
  // подготовим данные для вставки в виде пар: наименование столбца -
  // значение
  cv.put("name", name);
  cv.put("email", email);
  // вставляем запись и получаем ее ID
  long rowID = db.insert("mytable", null, cv);
  Log.d(LOG_TAG, "row inserted, ID = " + rowID);
  break;
case R.id.btnRead:
  Log.d(LOG_TAG, "--- Rows in mytable: ---");
  // делаем запрос всех данных из таблицы mytable, получаем Cursor
  Cursor c = db.query("mytable", null, null, null, null, null, null);

  // ставим позицию курсора на первую строку выборки
  // если в выборке нет строк, вернется false
  if (c.moveToFirst()) {

    // определяем номера столбцов по имени в выборке
    int idColIndex = c.getColumnIndex("id");
    int nameColIndex = c.getColumnIndex("name");
    int emailColIndex = c.getColumnIndex("email");

    do {
      // получаем значения по номерам столбцов и пишем все в лог
      Log.d(LOG_TAG,
          "ID = " + c.getInt(idColIndex) + ", name = "
              + c.getString(nameColIndex) + ", email = "
              + c.getString(emailColIndex));
      // переход на следующую строку
      // а если следующей нет (текущая - последняя), то false -
      // выходим из цикла
    } while (c.moveToNext());
  } else
    Log.d(LOG_TAG, "0 rows");
  break;
case R.id.btnClear:
  Log.d(LOG_TAG, "--- Clear mytable: ---");
  // удаляем все записи
  int clearCount = db.delete("mytable", null, null);
  Log.d(LOG_TAG, "deleted rows count = " + clearCount);
  break;
case R.id.btnUpd:
  if (id.equalsIgnoreCase("")) {
    break;
  }
  Log.d(LOG_TAG, "--- Update mytabe: ---");
  // подготовим значения для обновления
  cv.put("name", name);
  cv.put("email", email);
```

```
        // обновляем по id
        int updCount = db.update("mytable", cv, "id = ?",
            new String[] { id });
        Log.d(LOG_TAG, "updated rows count = " + updCount);
        break;
      case R.id.btnDel:
        if (id.equalsIgnoreCase("")) {
          break;
        }
        Log.d(LOG_TAG, "--- Delete from mytabe: ---");
        // удаляем по id
        int delCount = db.delete("mytable", "id = " + id, null);
        Log.d(LOG_TAG, "deleted rows count = " + delCount);
        break;
      }
      // закрываем подключение к БД
      dbHelper.close();
    }

    class DBHelper extends SQLiteOpenHelper {

      public DBHelper(Context context) {
        // конструктор суперкласса
        super(context, "myDB", null, 1);
      }

      public void onCreate(SQLiteDatabase db) {
        Log.d(LOG_TAG, "--- onCreate database ---");
        // создаем таблицу с полями
        db.execSQL("create table mytable ("
            + "id integer primary key autoincrement,"
            + "name text,"
            + "email text" + ");");
      }

      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

      }
    }

}
```

На следующем уроке:

- подробно разбираем метод чтения данных query
- используем сортировку, группировку, условия, having

Урок 36. SQLite. Подробнее про метод query. Условие, сортировка, группировка

В этом уроке:

- подробно разбираем метод чтения данных query
- используем сортировку, группировку, условия, having

На прошлых уроках мы использовали метод **query** для чтения всех данных из таблицы. Мы использовали только имя таблицы в качестве входного параметра и получали все записи. Но у query есть и другие параметры:

columns – список полей, которые мы хотим получить
selection – строка условия WHERE
selectionArgs – массив аргументов для selection. В selection можно использовать знаки *?*, а которые будут заменены этими значениями.
groupBy - группировка
having – использование условий для агрегатных функций
orderBy - сортировка

Попробуем на примере их использовать. Создадим приложение – **справочник стран**. Возьмем **десять стран** и сохраним в БД их **наименование**, **количество** населения и **регион**. Реализуем в приложении следующие функции:

- вывод всех записей
- вывод значения агрегатной функции (SUM, MIN, MAX, COUNT)
- вывод стран с населением, больше чем указано
- группировка стран по региону
- вывод регионов с населением больше, чем указано
- сортировка стран по наименованию, населению или региону

Выводить все данные снова будем в лог.

Создадим проект:

Project name: P0361_SQLiteQuery
Build Target: Android 2.3.3
Application name: SQLiteQuery
Package name: ru.startandroid.develop.p0361sqlitequery
Create Activity: MainActivity

Открываем layout-файл **main.xml** и пишем:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="Справочник стран"
```

```
        android:textSize="14sp"
        android:gravity="center_horizontal"
        android:layout_marginBottom="5dp"
        android:layout_marginTop="5dp">
    </TextView>
    <Button
        android:id="@+id/btnAll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Все записи"
        android:layout_marginTop="5dp">
    </Button>
    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp">
        <Button
            android:id="@+id/btnFunc"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Функция">
        </Button>
        <EditText
            android:id="@+id/etFunc"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1">
            <requestFocus>
            </requestFocus>
        </EditText>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/linearLayout2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp">
        <Button
            android:id="@+id/btnPeople"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Население >">
        </Button>
        <EditText
            android:id="@+id/etPeople"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:inputType="number">
        </EditText>
    </LinearLayout>
    <Button
        android:id="@+id/btnGroup"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Население по региону"
        android:layout_marginTop="5dp">
    </Button>
    <LinearLayout
        android:id="@+id/linearLayout4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp">
        <Button
            android:id="@+id/btnHaving"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Население по региону >">
        </Button>
        <EditText
            android:id="@+id/etRegionPeople"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:inputType="number">
        </EditText>
    </LinearLayout>
    <LinearLayout
        android:id="@+id/linearLayout3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp">
        <Button
            android:id="@+id/btnSort"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Сортировка">
        </Button>
        <RadioGroup
            android:id="@+id/rgSort"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
            <RadioButton
                android:id="@+id/rName"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:checked="true"
                android:text="Наименование">
            </RadioButton>
            <RadioButton
                android:id="@+id/rPeople"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Население">
            </RadioButton>
            <RadioButton
                android:id="@+id/rRegion"
                android:layout_width="wrap_content"
```

```
                android:layout_height="wrap_content"
                android:text="Регион">
            </RadioButton>
        </RadioGroup>
    </LinearLayout>
</LinearLayout>
```

6 кнопок – 6 функций, которые мы планируем реализовать. Поля для ввода значений, где это необходимо. Для **сортировки** используем **RadioGroup**.

Код для **MainActivity.java**:

```
package ru.startandroid.develop.p0361sqlitequery;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.RadioGroup;

public class MainActivity extends Activity implements OnClickListener {

  final String LOG_TAG = "myLogs";

  String name[] = { "Китай", "США", "Бразилия", "Россия", "Япония",
      "Германия", "Египет", "Италия", "Франция", "Канада" };
  int people[] = { 1400, 311, 195, 142, 128, 82, 80, 60, 66, 35 };
  String region[] = { "Азия", "Америка", "Америка", "Европа", "Азия",
      "Европа", "Африка", "Европа", "Европа", "Америка" };

  Button btnAll, btnFunc, btnPeople, btnSort, btnGroup, btnHaving;
  EditText etFunc, etPeople, etRegionPeople;
  RadioGroup rgSort;

  DBHelper dbHelper;
  SQLiteDatabase db;

  /** Called when the activity is first created. */

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    btnAll = (Button) findViewById(R.id.btnAll);
    btnAll.setOnClickListener(this);

    btnFunc = (Button) findViewById(R.id.btnFunc);
    btnFunc.setOnClickListener(this);
```

```
    btnPeople = (Button) findViewById(R.id.btnPeople);
    btnPeople.setOnClickListener(this);

    btnSort = (Button) findViewById(R.id.btnSort);
    btnSort.setOnClickListener(this);

    btnGroup = (Button) findViewById(R.id.btnGroup);
    btnGroup.setOnClickListener(this);

    btnHaving = (Button) findViewById(R.id.btnHaving);
    btnHaving.setOnClickListener(this);

    etFunc = (EditText) findViewById(R.id.etFunc);
    etPeople = (EditText) findViewById(R.id.etPeople);
    etRegionPeople = (EditText) findViewById(R.id.etRegionPeople);

    rgSort = (RadioGroup) findViewById(R.id.rgSort);

    dbHelper = new DBHelper(this);
    // подключаемся к базе
    db = dbHelper.getWritableDatabase();

    // проверка существования записей
    Cursor c = db.query("mytable", null, null, null, null, null, null);
    if (c.getCount() == 0) {
      ContentValues cv = new ContentValues();
      // заполним таблицу
      for (int i = 0; i < 10; i++) {
        cv.put("name", name[i]);
        cv.put("people", people[i]);
        cv.put("region", region[i]);
        Log.d(LOG_TAG, "id = " + db.insert("mytable", null, cv));
      }
    }
    dbHelper.close();
    // эмулируем нажатие кнопки btnAll
    onClick(btnAll);

  }

  public void onClick(View v) {

    // подключаемся к базе
    db = dbHelper.getWritableDatabase();

    // данные с экрана
    String sFunc = etFunc.getText().toString();
    String sPeople = etPeople.getText().toString();
    String sRegionPeople = etRegionPeople.getText().toString();

    // переменные для query
    String[] columns = null;
    String selection = null;
    String[] selectionArgs = null;
    String groupBy = null;
    String having = null;
    String orderBy = null;
```

```
// курсор
Cursor c = null;

// определяем нажатую кнопку
switch (v.getId()) {
// Все записи
case R.id.btnAll:
  Log.d(LOG_TAG, "--- Все записи ---");
  c = db.query("mytable", null, null, null, null, null, null);
  break;
// Функция
case R.id.btnFunc:
  Log.d(LOG_TAG, "--- Функция " + sFunc + " ---");
  columns = new String[] { sFunc };
  c = db.query("mytable", columns, null, null, null, null, null);
  break;
// Население больше, чем
case R.id.btnPeople:
  Log.d(LOG_TAG, "--- Население больше " + sPeople + " ---");
  selection = "people > ?";
  selectionArgs = new String[] { sPeople };
  c = db.query("mytable", null, selection, selectionArgs, null, null,
      null);
  break;
// Население по региону
case R.id.btnGroup:
  Log.d(LOG_TAG, "--- Население по региону ---");
  columns = new String[] { "region", "sum(people) as people" };
  groupBy = "region";
  c = db.query("mytable", columns, null, null, groupBy, null, null);
  break;
// Население по региону больше чем
case R.id.btnHaving:
  Log.d(LOG_TAG, "--- Регионы с населением больше " + sRegionPeople
      + " ---");
  columns = new String[] { "region", "sum(people) as people" };
  groupBy = "region";
  having = "sum(people) > " + sRegionPeople;
  c = db.query("mytable", columns, null, null, groupBy, having, null);
  break;
// Сортировка
case R.id.btnSort:
  // сортировка по
  switch (rgSort.getCheckedRadioButtonId()) {
  // наименование
  case (R.id.rName):
    Log.d(LOG_TAG, "--- Сортировка по наименованию ---");
    orderBy = "name";
    break;
  // население
  case (R.id.rPeople):
    Log.d(LOG_TAG, "--- Сортировка по населению ---");
    orderBy = "people";
    break;
  // регион
  case (R.id.rRegion):
    Log.d(LOG_TAG, "--- Сортировка по региону ---");
    orderBy = "region";
    break;
  }
  c = db.query("mytable", null, null, null, null, null, orderBy);
```

```
        break;
      }

      if (c != null) {
        if (c.moveToFirst()) {
          String str;
          do {
            str = "";
            for (String cn : c.getColumnNames()) {
              str = str.concat(cn + " = "
                  + c.getString(c.getColumnIndex(cn)) + "; ");
            }
            Log.d(LOG_TAG, str);

          } while (c.moveToNext());
        }
      } else
        Log.d(LOG_TAG, "Cursor is null");

      dbHelper.close();
    }

    class DBHelper extends SQLiteOpenHelper {

      public DBHelper(Context context) {
        // конструктор суперкласса
        super(context, "myDB", null, 1);
      }

      public void onCreate(SQLiteDatabase db) {
        Log.d(LOG_TAG, "--- onCreate database ---");
        // создаем таблицу с полями
        db.execSQL("create table mytable ("
            + "id integer primary key autoincrement," + "name text,"
            + "people integer," + "region text" + ");");
      }

      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

      }
    }

}
```

Никаких проверок на вводимые с экрана значения я не делал, чтобы не перегружать код. Он и так немаленький получился, но ничего сложного в нем нет.

Три массива данных **name**, **people**, **region**. Это **наименования** стран, их **население** (в тысячах) и **регионы**, к которым страны относятся. По этим данным мы будем заполнять таблицу.

В методе **onCreate** мы определяем и находим экранные элементы, присваиваем обработчики, создаем объект **dbHelper** для управления БД, подключаемся к базе и получаем объект **db** для работы с БД, проверяем наличие записей в таблице, если нет ничего – заполняем ее данными, закрываем соединение и эмулируем нажатие кнопки **Все записи** - чтобы сразу вывести весь список.

В методе **onClick** – подключаемся к базе, читаем данные с экранных полей в переменные, описываем переменные, которые будем использовать в методе **query**, и курсор, и смотрим, какая кнопка была нажата.

btnAll – вывод всех записей. Вызываем метод query с **именем** таблицы и **null** для остальных параметров. Это уже

знакомо, делали на прошлом уроке.

btnFunc – вывод значения агрегатной функции (или любого поля). Использую параметр **columns**, в который надо записать **поля**, которые я хотел бы получить из таблицы, т.е. то, что обычно перечисляется после слова **SELECT** в SQL-запросе. **columns** имеет тип **String[]** – массив строк. Создаем массив из одного значения, которое считано с поля **etFunc** на экране. Запускаем query.

btnPeople – вывод стран с населением больше введенного на экране количества. Используем **selection** для формирования условия. При этом используем один аргумент - *?*. Значение аргумента задаем в **selectionArgs** – это **sPeople** – содержимое поля **etPeople**. Запускаем query.

btnGroup – группировка стран по регионам и вывод общее количество населения. Используем **columns** для указания столбцов, которые хотели бы получить – **регион** и **сумма** населения. В **groupBy** указываем, что **группировка** будет по **региону**. Запускаем query.

btnHaving – вывод регионов с населением больше указанного числа. Полностью аналогично случаю с группировкой, но добавляется условие в параметре **having** – сумма населения региона должна быть меньше **sRegionPeople** (значение **etRegionPeople** с экрана).

btnSort – сортировка стран. Определяем какой **RadioButton** включен и соответственно указываем в **orderBy** поле для сортировки данных. Запускаем query.

В выше описанных случаях мы запускали **query** и получали объект с класса **Cursor**. Далее мы проверяем, что он существует и в нем есть записи (**moveToFirst**). Если все ок, то мы запускаем перебор записей в цикле **do ... while (c.moveToNext())**. Для каждой записи перебираем названия полей (**getColumnNames**), получаем по каждому полю его номер и извлекаем данные методом **getString**. Формируем список полей и значений в переменную **str**, которую потом выводим в лог. После всего этого закрываем соединение.

Ну и в конце кода идет описание вложенного класса **DBHelper**. Тут ничего не изменилось с прошлых уроков. Только при **создании таблицы** используются другие **поля**.

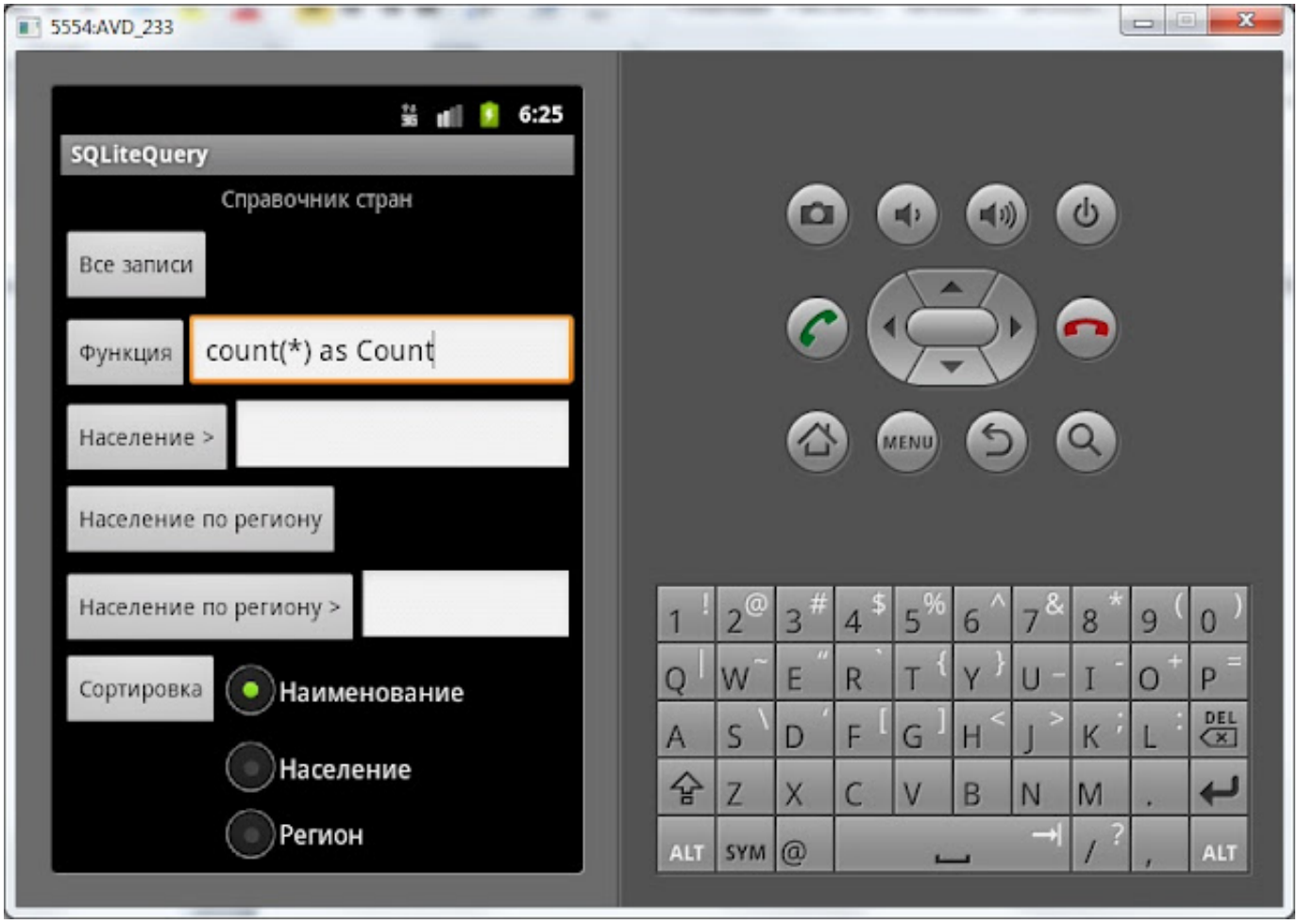
Сохраняем все и запускаем приложение.

В лог при запуске вывелись все записи, как если бы мы нажали кнопку «Все записи».

--- Все записи ---
id = 1; name = Китай; people = 1400; region = Азия;
id = 2; name = США; people = 311; region = Америка;
id = 3; name = Бразилия; people = 195; region = Америка;
id = 4; name = Россия; people = 142; region = Европа;
id = 5; name = Япония; people = 128; region = Азия;
id = 6; name = Германия; people = 82; region = Европа;
id = 7; name = Египет; people = 80; region = Африка;
id = 8; name = Италия; people = 60; region = Европа;
id = 9; name = Франция; people = 66; region = Европа;
id = 10; name = Канада; people = 35; region = Америка;

Т.е. таблица заполнена данными, можно работать.

Попробуем использовать агрегатную функцию. Например – получим **кол-во записей**. Вводим значение:

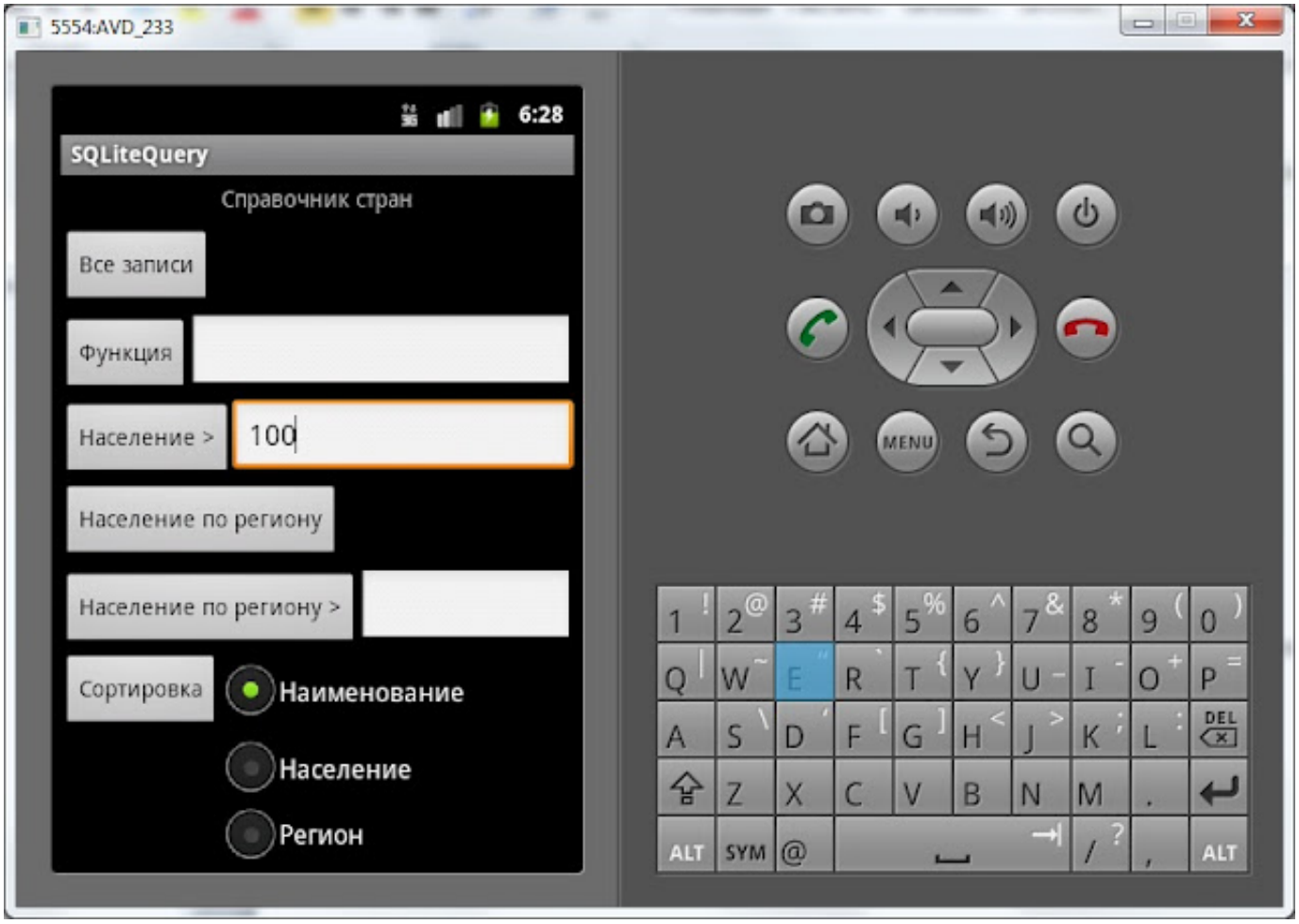


жмем кнопку **Функция**. Смотрим лог:

--- Функция count(*) as Count ---
Count = 10;

Все верно, 10 записей в таблице.

Покажем страны с населением больше 100 млн. Вводим **100** и жмем **Население >**



Лог:

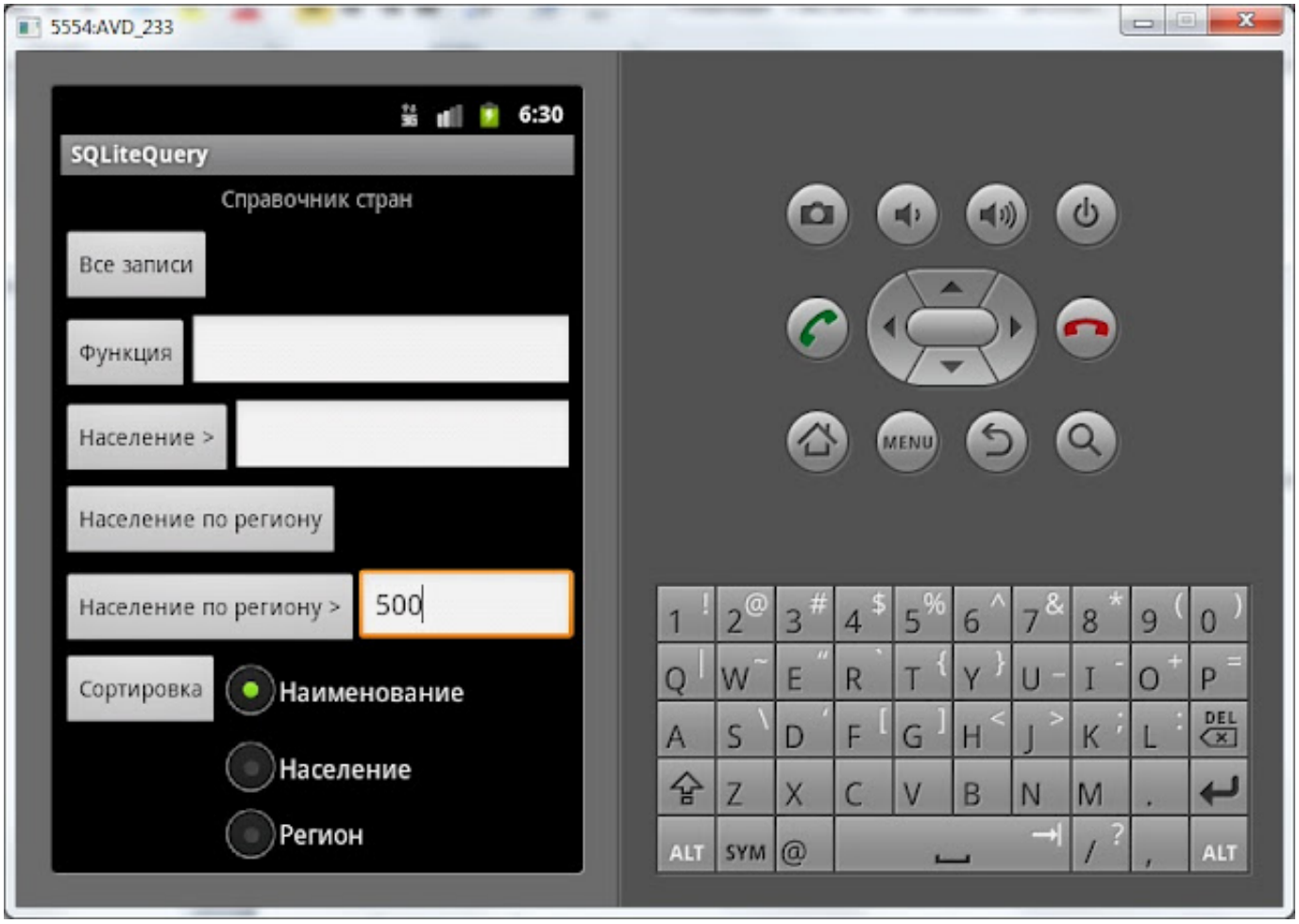
--- Население больше 100 ---
id = 1; name = Китай; people = 1400; region = Азия;
id = 2; name = США; people = 311; region = Америка;
id = 3; name = Бразилия; people = 195; region = Америка;
id = 4; name = Россия; people = 142; region = Европа;
id = 5; name = Япония; people = 128; region = Азия;

Сгруппируем страны по региону и покажем население регионов. Нажмем кнопку **Население по региону**

Лог:

--- Население по региону ---
region = Азия; people = 1528;
region = Америка; people = 541;
region = Африка; people = 80;
region = Европа; people = 350;

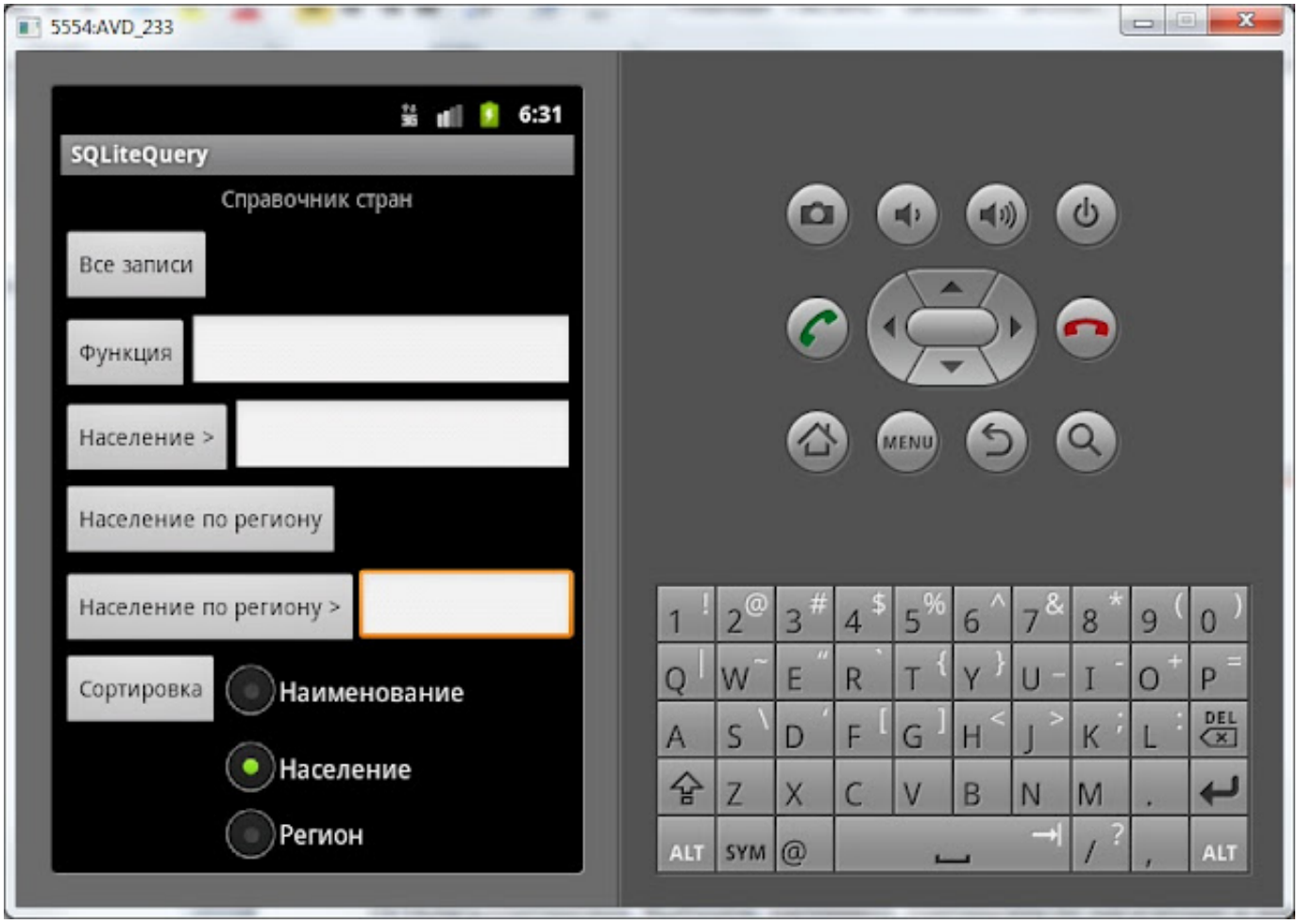
Теперь отобразим только те регионы, в которых население выше 500 млн.чел. Вводим **500** и жмем **Население по региону >**



Лог:

--- Регионы с населением больше 500 ---
region = Азия; people = 1528;
region = Америка; people = 541;

Осталась сортировка. Выберем, например, сортировку по **населению** и жмем кнопку **Сортировка**



Лог:

--- Сортировка по населению ---
id = 10; name = Канада; people = 35; region = Америка;
id = 8; name = Италия; people = 60; region = Европа;
id = 9; name = Франция; people = 66; region = Европа;
id = 7; name = Египет; people = 80; region = Африка;
id = 6; name = Германия; people = 82; region = Европа;
id = 5; name = Япония; people = 128; region = Азия;
id = 4; name = Россия; people = 142; region = Европа;
id = 3; name = Бразилия; people = 195; region = Америка;
id = 2; name = США; people = 311; region = Америка;
id = 1; name = Китай; people = 1400; region = Азия;

По умолчанию **сортировка** идет по **возрастанию**.

Все работает так, как и должно. На этих примерах мы использовали все основные **параметры** метода **query**. Кроме описанных параметров, у метода query есть также реализации с использованием параметров **limit** и **distinct**. Я не стал их здесь отдельно показывать. Расскажу на словах:

limit – **строковый** параметр, указывается в формате *[offset], rows*. Т.е. если в **query** в качестве **limit** передать строку "*5*" - то запрос выдаст только **пять** первых записей. Если же передать "*3,5*", то запрос выдаст **пять** записей, начиная с **четвертой** (НЕ с третьей).

distinct – это **boolean**-параметр, удаление дубликатов. Может быть **true** или **false**.

Надеюсь, что метод query, который сначала казался большим скоплением параметров, стал понятен и прост.

На следующем уроке:

- читаем данные из связанных таблиц
- используем rawQuery

Урок 37. Запросы из связанных таблиц. INNER JOIN в SQLite. Метод rawQuery.

В этом уроке:

- читаем данные из связанных таблиц
- используем rawQuery

Мы достаточно подробно разобрали метод **query**. Но не рассмотрели, как с его помощью выполнять запросы для **связанных таблиц**. Создадим простое приложение, которое будет делать запрос из двух таблиц и выводить результат в лог. Таблицы будут **people** и **position**. В первую (people) запишем список **людей**, во вторую (position) – список **должностей**. И для каждого **человека** в **people** будет прописан **id должности** из **position**.

Создадим проект:

Project name: P0371_SQLiteInnerJoin
Build Target: Android 2.3.3
Application name: SQLiteInnerJoin
Package name: ru.startandroid.develop.p0371sqliteinnerjoin
Create Activity: MainActivity

Экран вообще использовать не будем, поэтому **main.xml** даже не трогаем. Открываем **MainActivity.java** и пишем код:

```
package ru.startandroid.develop.p0371sqliteinnerjoin;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  // данные для таблицы должностей
  int[] position_id = { 1, 2, 3, 4 };
  String[] position_name = { "Директор", "Программер", "Бухгалтер", "Охранник" };
  int[] position_salary = { 15000, 13000, 10000, 8000 };

  // данные для таблицы людей
  String[] people_name = { "Иван", "Марья", "Петр", "Антон", "Даша", "Борис", "Костя", "Игорь" };
  int[] people_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Подключаемся к БД
    DBHelper dbh = new DBHelper(this);
    SQLiteDatabase db = dbh.getWritableDatabase();
```

```
  // Описание курсора
  Cursor c;

  // выводим в лог данные по должностям
  Log.d(LOG_TAG, "--- Table position ---");
  c = db.query("position", null, null, null, null, null, null);
  logCursor(c);
  Log.d(LOG_TAG, "--- ---");

  // выводим в лог данные по людям
  Log.d(LOG_TAG, "--- Table people ---");
  c = db.query("people", null, null, null, null, null, null);
  logCursor(c);
  Log.d(LOG_TAG, "--- ---");

  // выводим результат объединения
  // используем rawQuery
  Log.d(LOG_TAG, "--- INNER JOIN with rawQuery---");
  String sqlQuery = "select PL.name as Name, PS.name as Position, salary as Salary "
      + "from people as PL "
      + "inner join position as PS "
      + "on PL.posid = PS.id "
      + "where salary > ?";
  c = db.rawQuery(sqlQuery, new String[] {"12000"});
  logCursor(c);
  Log.d(LOG_TAG, "--- ---");

  // выводим результат объединения
  // используем query
  Log.d(LOG_TAG, "--- INNER JOIN with query---");
  String table = "people as PL inner join position as PS on PL.posid = PS.id";
  String columns[] = { "PL.name as Name", "PS.name as Position", "salary as Salary" };
  String selection = "salary < ?";
  String[] selectionArgs = {"12000"};
  c = db.query(table, columns, selection, selectionArgs, null, null, null);
  logCursor(c);
  Log.d(LOG_TAG, "--- ---");

  // закрываем БД
  dbh.close();
}

// вывод в лог данных из курсора
void logCursor(Cursor c) {
  if (c != null) {
    if (c.moveToFirst()) {
      String str;
      do {
        str = "";
        for (String cn : c.getColumnNames()) {
          str = str.concat(cn + " = " + c.getString(c.getColumnIndex(cn)) + "; ");
        }
        Log.d(LOG_TAG, str);
      } while (c.moveToNext());
    }
  } else
    Log.d(LOG_TAG, "Cursor is null");
}

// класс для работы с БД
class DBHelper extends SQLiteOpenHelper {
```

```
    public DBHelper(Context context) {
      super(context, "myDB", null, 1);
    }

    public void onCreate(SQLiteDatabase db) {
      Log.d(LOG_TAG, "--- onCreate database ---");

      ContentValues cv = new ContentValues();

      // создаем таблицу должностей
      db.execSQL("create table position ("
          + "id integer primary key,"
          + "name text," + "salary integer"
          + ");");

      // заполняем ее
      for (int i = 0; i < position_id.length; i++) {
        cv.clear();
        cv.put("id", position_id[i]);
        cv.put("name", position_name[i]);
        cv.put("salary", position_salary[i]);
        db.insert("position", null, cv);
      }

      // создаем таблицу людей
      db.execSQL("create table people ("
          + "id integer primary key autoincrement,"
          + "name text,"
          + "posid integer"
          + ");");

      // заполняем ее
      for (int i = 0; i < people_name.length; i++) {
        cv.clear();
        cv.put("name", people_name[i]);
        cv.put("posid", people_posid[i]);
        db.insert("people", null, cv);
      }
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
  }

}
```

Разбираем код. Сначала идут несколько **массивов** с **данными** для таблиц. Обратите внимание, для **должностей** мы будем указывать **id** при заполнении таблиц. Это сделано для того, чтобы мы знали эти номера и могли их использовать в таблице **людей** для указания id должности.

В методе Activity **onCreate** мы создаем объект для **управления** БД и **подключаемся** к БД. Далее используя знакомый нам **query** выводим в лог данные из таблиц **position** и **people**.

Для вывода объединения таблиц используем rawQuery. Это несложный метод, который принимает на вход **SQL-запрос** и список **аргументов** для условия WHERE (если необходимо). Мы сформировали запрос на объединение двух таблиц и вывода **имени**, **должности** и **зарплаты** человека. Условие выборки: **ЗП** должна быть **больше 12000**. Мы использовали аргументы для формирования условия.

Далее снова выводим объединение таблиц, но используем обычный **query**. В **table** записываем все таблицы, их алиасы и

условие **JOIN**. В **columns** – все нужные поля с использованием алиасов. Ну и в **selection** и **selectionArgs** пишем условие выборки – **ЗП меньше 12000**.

Наш метод **logCursor** получает на вход **Cursor** и выводит в лог все **содержимое**. Тут все знакомо с прошлых уроков.

В классе **DBHelper** кодим **создание** таблиц и, на этот раз, здесь же их **наполнение** данными.

Все сохраним и запустим. Смотрим лог:

--- onCreate database ---
--- Table position ---
id = 1; name = Директор; salary = 15000;
id = 2; name = Программер; salary = 13000;
id = 3; name = Бухгалтер; salary = 10000;
id = 4; name = Охранник; salary = 8000;
--- ---
--- Table people ---
id = 1; name = Иван; posid = 2;
id = 2; name = Марья; posid = 3;
id = 3; name = Петр; posid = 2;
id = 4; name = Антон; posid = 2;
id = 5; name = Даша; posid = 3;
id = 6; name = Борис; posid = 1;
id = 7; name = Костя; posid = 2;
id = 8; name = Игорь; posid = 4;
--- ---

Вывели данные по таблицам раздельно.

--- INNER JOIN with rawQuery---
Name = Иван; Position = Программер; Salary = 13000;
Name = Петр; Position = Программер; Salary = 13000;
Name = Антон; Position = Программер; Salary = 13000;
Name = Борис; Position = Директор; Salary = 15000;
Name = Костя; Position = Программер; Salary = 13000;
--- ---

Вывели данные из rawQuery. ЗП > 12000

--- INNER JOIN with query---
Name = Марья; Position = Бухгалтер; Salary = 10000;
Name = Даша; Position = Бухгалтер; Salary = 10000;
Name = Игорь; Position = Охранник; Salary = 8000;
--- ---

Вывели данные из query. ЗП < 12000

Как видите, запросы из связанных таблиц в **SQLite** – не проблема и ничем не отличается от обычных БД.

Что использовать **rawQuery** или **query** – решать вам и зависит от ситуации. Хотя, навскидку я не могу придумать преимущества одного над другим в той или иной ситуации. Но наверно они есть.

Данные по ЗП и должностям являются выдуманными, любое совпадение – случайно. И, разумеется, ничего против бухгалтеров и охранников я не имею))

На следующем уроке:

- используем транзакции при работе с БД

Урок 38. Транзакции в SQLite. Небольшой FAQ по SQLite.

В этом уроке:

- используем транзакции при работе с БД

Что такое БД-**транзакция**, думаю объяснять особо не надо. Она используется при работе с данными по принципу «**все или ничего**». Т.е., например, вам нужно вставить пачку данных. Но вставить надо так, чтобы или все вставилось или ничего не вставилось. И если в процессе половина записей прошла, а другая нет – должна быть возможность **откатить** изменения.

Напишем простое приложение и исследуем возможности **SQLite** в этом плане.

Создадим проект:

Project name: P0381_SQLiteTransaction
Build Target: Android 2.3.3
Application name: SQLiteTransaction
Package name: ru.startandroid.develop.p0381sqlitetransaction
Create Activity: MainActivity

Открываем **MainActivity.java** и пишем:

```
package ru.startandroid.develop.p0381sqlitetransaction;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  DBHelper dbh;
  SQLiteDatabase db;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.d(LOG_TAG, "--- onCreate Activity ---");
    dbh = new DBHelper(this);
    myActions();
  }

  void myActions() {
    db = dbh.getWritableDatabase();
    delete(db, "mytable");
    insert(db, "mytable", "val1");
    read(db, "mytable");
    dbh.close();
```

```
  }

  void insert(SQLiteDatabase db, String table, String value) {
    Log.d(LOG_TAG, "Insert in table " + table + " value = " + value);
    ContentValues cv = new ContentValues();
    cv.put("val", value);
    db.insert(table, null, cv);
  }

  void read(SQLiteDatabase db, String table) {
    Log.d(LOG_TAG, "Read table " + table);
    Cursor c = db.query(table, null, null, null, null, null, null);
    if (c != null) {
      Log.d(LOG_TAG, "Records count = " + c.getCount());
      if (c.moveToFirst()) {
        do {
          Log.d(LOG_TAG, c.getString(c.getColumnIndex("val")));
        } while (c.moveToNext());
      }
    }
  }

  void delete(SQLiteDatabase db, String table) {
    Log.d(LOG_TAG, "Delete all from table " + table);
    db.delete(table, null, null);
  }

  // класс для работы с БД
  class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context) {
      super(context, "myDB", null, 1);
    }

    public void onCreate(SQLiteDatabase db) {
      Log.d(LOG_TAG, "--- onCreate database ---");

      db.execSQL("create table mytable ("
          + "id integer primary key autoincrement,"
          + "val text"
          + ");");
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
  }

}
```

Разбираем код. Я создал несколько методов, где сгруппировал операции: **insert** – набор операций для вставки записи, **read** – чтение всех записей, **delete** – удаление всех записей. Класс **DBHelper** – для управления БД. Интересовать нас будет метод **myActions**. Сейчас в нем мы подключаемся к БД, очищаем таблицу mytable, вставляем строку с значением *val1*, выводим в лог все записи из таблицы и отключаемся.

Все сохраним, запустим приложение. Смотрим лог:

--- onCreate Activity ---
--- onCreate database ---
Delete all from table mytable

Insert in table mytable value = val1
Read table mytable
Records count = 1
val1

Все верно, запись вставилась и отобразилась.

Теперь попробуем использовать транзакцию. Поправим код **myActions** на этот:

```
  void myActions() {
    db = dbh.getWritableDatabase();
    delete(db, "mytable");
    db.beginTransaction();
    insert(db, "mytable", "val1");
    db.endTransaction();
    insert(db, "mytable", "val2");
    read(db, "mytable");
    dbh.close();
  }
```

Мы подключаемся к базе, чистим таблицу, открываем транзакцию методом beginTransaction, вставляем *val1*, закрываем транзакцию методом endTransaction, вставляем *val2*, выводим содержимое в лог и отключаемся. Все сохраняем, запускаем и смотрим лог:

--- onCreate Activity ---
Delete all from table mytable
Insert in table mytable value = val1
Insert in table mytable value = val2
Read table mytable
Records count = 1
val2

По логу видно, что вставляли мы **две** записи, но прошла только **вторая**. Та, которая была в **транзакции** – не записалась. Это произошло потому, что мы явно не указали, что транзакция должна быть успешно **закрыта**. Если этого не сделать, то при закрытии транзакции все операции **отменятся**. Давайте исправимся. Снова перепишем **myActions**:

```
  void myActions() {
    db = dbh.getWritableDatabase();
    delete(db, "mytable");
    db.beginTransaction();
    insert(db, "mytable", "val1");
    db.setTransactionSuccessful();
    insert(db, "mytable", "val2");
    db.endTransaction();
    insert(db, "mytable", "val3");
    read(db, "mytable");
    dbh.close();
  }
```

Подключаемся к БД, чистим таблицу, открываем транзакцию, вставляем *val1*, подтверждаем успешность транзакции методом setTransactionSuccessful, вставляем *val2*, закрываем транзакцию, вставляем *val3*, выводим содержимое и отключаемся.

Сохраняем, запускаем, смотрим лог:

--- onCreate Activity ---
Delete all from table mytable
Insert in table mytable value = val1
Insert in table mytable value = val2

Insert in table mytable value = val3
Read table mytable
Records count = 3
val1
val2
val3

Вставились все три записи. Обратите внимание - несмотря на то, что *val2* мы вставляли уже после подтверждения успешности транзакции, запись вставилась, вошла в эту транзакцию.

Транзакция при открытии ставит **блокировку** на базу. Убедимся в этом, попробуем создать новое подключение к БД во время транзакции. Перепишем **myActions**:

```
void myActions() {
  try {
    db = dbh.getWritableDatabase();
    delete(db, "mytable");

    db.beginTransaction();
    insert(db, "mytable", "val1");

    Log.d(LOG_TAG, "create DBHelper");
    DBHelper dbh2 = new DBHelper(this);
    Log.d(LOG_TAG, "get db");
    SQLiteDatabase db2 = dbh2.getWritableDatabase();
    read(db2, "mytable");
    dbh2.close();

    db.setTransactionSuccessful();
    db.endTransaction();

    read(db, "mytable");
    dbh.close();

  } catch (Exception ex) {
    Log.d(LOG_TAG, ex.getClass() + " error: " + ex.getMessage());
  }
}
```

Подключаемся к базе, чистим таблицу, открываем транзакцию, вставляем запись, создаем новое подключение к БД - **db2**, читаем содержимое вторым подключением, закрываем второе подключение, успешно закрываем транзакцию, читаем содержимое первым подключением, закрываем первое подключение.

Все сохраним и запустим. Смотрим лог:

--- onCreate Activity ---
Delete all from table mytable
Insert in table mytable value = val1
create DBHelper
get db
class android.database.sqlite.SQLiteException error: database is locked

Мы видим, что при попытке создать второе подключение к базе произошла ошибка **SQLiteException** – база **заблокирована** открытой транзакцией. Если вы теперь закомментите или удалите строки управления транзакцией и снова выполните код, то все пройдет успешно, т.к. никаких блокировок не будет.

Наверняка есть некоторые вопросы по этой теме. Попробую здесь же ответить на некоторые.

close

Метод **close** есть и у **SQLiteDatabase** и у **SQLiteOpenHelper**. Какая между ними разница? Каким из них пользоваться для закрытия подключения?

Тут надо понимать один момент – объект **SQLiteOpenHelper** всегда предоставляет только **одно** подключение. Попробую объяснить этот механизм. У объекта **SQLiteOpenHelper** есть внутренний атрибут **mDatabase** типа **SQLiteDatabase**. Когда мы вызываем метод **getWritableDatabase**, объект **SQLiteOpenHelper** проверяет: если **mDatabase** не null и не закрыт, то он и идет в качестве **return**. Иначе **SQLiteOpenHelper** выполняет подключение к БД, записывает новый **SQLiteDatabase**-объект в **mDatabase** и возвращает нам его. Т.е. метод **getWritableDatabase** либо возвращает **существующее подключение** к БД, либо создает новое в случае отсутствия подключения. Когда же выполняется метод **close** для **SQLiteOpenHelper**, то происходит вызов **close** для **mDatabase** и выполняется код **mDatabase = null**.

Рассмотрим на примере. Снова меняем метод myActions:

```
void myActions() {
  db = dbh.getWritableDatabase();
  SQLiteDatabase db2 = dbh.getWritableDatabase();
  Log.d(LOG_TAG, "db = db2 - " + db.equals(db2));
  Log.d(LOG_TAG, "db open - " + db.isOpen() + ", db2 open - " + db2.isOpen());
  db2.close();
  Log.d(LOG_TAG, "db open - " + db.isOpen() + ", db2 open - " + db2.isOpen());
}
```

Сначала мы получаем **db**. При этом **dbh** проверяет свой внутренний атрибут **mDatabase**. Т.к. это первая попытка подключения, то **mDatabase** пуст, поэтому внутри **dbh** производится подключение и в **mDatabase** записывается свежесозданный **SQLiteDatabase**, и он же и возвращается в **db** из метода **getWritableDatabse** .

Затем мы из того же **dbh** получаем **db2**. **dbh** снова проверяет свой внутренний **mDatabse**, видит, что он уже **не null и не закрыт**, и возвращает нам его в наш **db2**. В итоге **db** и **db2** равны и ссылаются на один и тот же объект. Проверяем это с помощью метода **equals**. Далее проверим, что **db** и **db2 открыты**. Потом **закроем** только **db2**, и еще раз проверим на **открытость оба** объекта.

Сохраняем, запускаем, смотрим лог:

--- onCreate Activity ---
db = db2 – true
db open - true, db2 open – true
db open - false, db2 open - false

Видим, что **equals** вернул **true**. Затем видно, что **db** и **db2 открыты**. А после закрытия **db2** видим, что закрыты оба объекта. Все оттого, что «оба объекта» – это всего лишь **две ссылки** на **один объект**.

Если в коде вместо **db2.close()** поставить **dbh.close()** - эффект будет тот же. **dbh** вызовет метод **close** для **mDatabase** и обнулит его - **mDatabase = null**. А **db** и **db2** будут ссылаться на закрытый **SQLiteDatabase**.

Я думаю, что правильнее вызывать **close** для **SQLiteOpenHelper**, а не для **SQLiteDatabase**. Т.к. гарантировано **закрывается** текущее открытое соединение и **обнуляется** внутренняя ссылка на объект.

Если вам надо получить второе открытое подключение к БД, то надо создавать новый экземпляр **DBHelper** и вызывать **getWritableDatabase**. Мы так делали чуть выше в примере с блокировкой транзакции.

read write

В чем разница между **getWritableDatabase** и **getReadableDatabase**? Судя по хелпу, в обычной ситуации **оба** метода возвращают **одно и то же**. И оба позволят **читать** и **менять** БД. В случае же, например, проблемы **отсутствия свободного места** на устройстве, метод **getReadableDatabase** вернет БД только **для чтения**, а **getWritableDatabase** выдаст **ошибку**.

_id, как имя поля-идентификатора

В различных источниках при работе с БД в качестве наименования поля-идентификатора в таблице используют не просто id, а _id. Почему?

Ответ нашелся в доках по Cursor-адаптерам. Цитата: "The Cursor must include a column named "_id" or this class will not work.". Т.е. если вы планируете использовать Cursor-адаптеры, то необходимо, чтобы таблица содержала поле _id, иначе адаптер не будет работать.

Блокировка

Метод открытия транзакции **beginTransaction** ставит блокировку в режиме **EXCLUSIVE**. Т.е. БД блокируется и на чтение и на запись для других подключений. В SDK Android версии старше 2.3.3 появился метод **beginTransactionNonExclusive**, который ставит блокировку в режиме **IMMEDIATE**. Я подозреваю, что это позволит читать данные другим подключениям.

Если есть желание подробнее погрузиться в тему, вам сюда.

Синтаксис

И кстати, рекомендуемая форма для использования транзакций такая:

```
db.beginTransaction();
try {
  ...
  db.setTransactionSuccessful();
} finally {
  db.endTransaction();
}
```

Это очень важно! Т.е. если вы **открыли** транзакцию, **выполнили** какие-либо действия и **не закрыли** транзакцию, то все операции будут считаться **неуспешными** и изменения не будут внесены в БД. Поэтому **закрытие** транзакции **необходимо** выполнять и **finally** нам это гарантирует.

На следующем уроке:

- меняем версию и обновляем структуру БД в onUpgrade

Урок 39. onUpgrade. Обновляем БД в SQLite

В этом уроке:

- меняем версию и обновляем структуру БД в onUpgrade

С **развитием** приложения может возникнуть необходимость **изменения структуры БД**, которую оно использует. На одном из прошлых уроков я упоминал, что для этого используется метод **onUpgrade** класса **SQLiteOpenHelper**. Этот метод вызывается, если **существующая** версия БД **отличается** от той, к которой мы пытаемся **подключиться**. **Версию** мы обычно **указывали** при вызове **конструктора** супер-класса **SQLiteOpenHelper** в конструкторе DBHelper.

Попробуем воспользоваться методом onUpgrade и посмотреть, как происходит переход на новую версию БД. Для этого напишем небольшое приложение, аналогичное одному из приложений с прошлых уроков – про сотрудников и должности.

Первая версия БД будет содержать только таблицу **people** с именем сотрудника и его должностью. Но такая таблица будет не совсем корректна. Если вдруг у нас изменится название должности, придется обновлять все соответствующие записи в people. Поэтому мы решаем изменить БД и организовать данные немного по-другому.

Во **второй** версии добавим таблицу **position** с названием должности и зарплатой. И в таблице **people** вместо названия должности пропишем соответствующий **ID** из **position**.

Создадим проект:

Project name: P0391_SQLiteOnUpgradeDB
Build Target: Android 2.3.3
Application name: SQLiteOnUpgradeDB
Package name: ru.startandroid.develop.p0391sqliteonupgradedb
Create Activity: MainActivity

Экран снова не используем, будем выводить все в лог.

Открываем **MainActivity.java** и кодим:

```
package ru.startandroid.develop.p0391sqliteonupgradedb;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final String DB_NAME = "staff"; // имя БД
  final int DB_VERSION = 1; // версия БД

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    DBHelper dbh = new DBHelper(this);
    SQLiteDatabase db = dbh.getWritableDatabase();
    Log.d(LOG_TAG, " --- Staff db v." + db.getVersion() + " --- ");
    writeStaff(db);
    dbh.close();
  }

  // запрос данных и вывод в лог
  private void writeStaff(SQLiteDatabase db) {
    Cursor c = db.rawQuery("select * from people", null);
    logCursor(c, "Table people");
  }

  // вывод в лог данных из курсора
  void logCursor(Cursor c, String title) {
    if (c != null) {
      if (c.moveToFirst()) {
        Log.d(LOG_TAG, title + ". " + c.getCount() + " rows");
        StringBuilder sb = new StringBuilder();
        do {
          sb.setLength(0);
          for (String cn : c.getColumnNames()) {
            sb.append(cn + " = "
                + c.getString(c.getColumnIndex(cn)) + "; ");
          }
          Log.d(LOG_TAG, sb.toString());
        } while (c.moveToNext());
      }
    } else
      Log.d(LOG_TAG, title + ". Cursor is null");
  }

  // класс для работы с БД
  class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context) {
      super(context, DB_NAME, null, DB_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
      Log.d(LOG_TAG, " --- onCreate database --- ");

      String[] people_name = { "Иван", "Марья", "Петр", "Антон", "Даша",
          "Борис", "Костя", "Игорь" };
      String[] people_positions = { "Программер", "Бухгалтер",
          "Программер", "Программер", "Бухгалтер", "Директор",
          "Программер", "Охранник" };

      ContentValues cv = new ContentValues();

      // создаем таблицу людей
      db.execSQL("create table people ("
          + "id integer primary key autoincrement,"
          + "name text, position text);");

      // заполняем ее
```

```
      for (int i = 0; i < people_name.length; i++) {
        cv.clear();
        cv.put("name", people_name[i]);
        cv.put("position", people_positions[i]);
        db.insert("people", null, cv);
      }
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
  }
}
```

Код несложен. Я сгруппировал операции по **выводу в лог данных** из Cursor – метод **logCursor**. Метод **writeStaff** – выбирает данные из таблицы people и вызывает метод для вывода данных в лог. В методе Activity **onCreate** мы создаем объект **DBHelper**, **подключаемся** к БД, выводим в лог **версию** БД, вызываем **writeStaff** и **отключаемся**.

В **DBHelper** все как обычно. В конструкторе вызываем конструктор супер-класса. Обратите внимание, **DB_VERSION** = 1 – мы будем подключаться к базе версии 1. В методе **onCreate** создаем таблицу и заполняем ее.

Все сохраним и запустим приложение. Смотрим лог:

--- onCreate database ---
--- Staff db v.1 ---
Table people. 8 rows
id = 1; name = Иван; position = Программер;
id = 2; name = Марья; position = Бухгалтер;
id = 3; name = Петр; position = Программер;
id = 4; name = Антон; position = Программер;
id = 5; name = Даша; position = Бухгалтер;
id = 6; name = Борис; position = Директор;
id = 7; name = Костя; position = Программер;
id = 8; name = Игорь; position = Охранник;

БД **создалась**, версия = **1** и данные из таблицы вывелись в лог. Приложение работает, все ок. Но тут мы (внезапно!) понимаем, что при проектировании структуры БД была допущена ошибка. Записывать название должности в поле таблицы people – неправильно. К тому же у нас еще добавляются данные по зарплатам. Надо создать таблицу должностей - position, и использовать из нее id в таблице people. Тем самым структура нашей БД меняется и мы присваиваем ей версию – **2**.

Но наше приложение уже установлено у пользователей. Оно уже создало БД версии 1, и в этой БД уже есть данные. Мы не можем просто удалить существующие таблицы и создать новые, т.к. возможно пользователь уже хранит там свои данные. Нам надо будет написать скрипты для обновления без потери данных.

План обновления такой:

- создаем и заполняем данными таблицу **position**
- добавляем в таблицу **people** столбец – **posid** для хранения **id** из **position**
- заполняем **people.posid** данными из **position** в зависимости от значения **people.position**
- удаляем столбец **people.position**

Давайте менять **MainActivity.java**. Наше приложение теперь будет ориентировано на БД версии 2. Укажем это, изменив значение константы **DB_VERSION** на 2:

```
  final int DB_VERSION = 2; // версия БД
```

Метод **writeStaff** перепишем таким образом:

```
  private void writeStaff(SQLiteDatabase db) {
    Cursor c = db.rawQuery("select * from people", null);
    logCursor(c, "Table people");

    c = db.rawQuery("select * from position", null);
    logCursor(c, "Table position");

    String sqlQuery = "select PL.name as Name, PS.name as Position, salary as Salary "
      + "from people as PL "
      + "inner join position as PS "
      + "on PL.posid = PS.id ";
    c = db.rawQuery(sqlQuery, null);
    logCursor(c, "inner join");
  }
```

Будем выводить в лог данные из таблиц **people**, **position** и их **объединения**.

Реализуем метод обновления - **onUpgrade** в **DBHelper**:

```
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
      Log.d(LOG_TAG, " --- onUpgrade database from " + oldVersion
          + " to " + newVersion + " version --- ");

      if (oldVersion == 1 && newVersion == 2) {

        ContentValues cv = new ContentValues();

        // данные для таблицы должностей
        int[] position_id = { 1, 2, 3, 4 };
        String[] position_name = { "Директор", "Программер",
            "Бухгалтер", "Охранник" };
        int[] position_salary = { 15000, 13000, 10000, 8000 };

        db.beginTransaction();
        try {
          // создаем таблицу должностей
          db.execSQL("create table position ("
              + "id integer primary key,"
              + "name text, salary integer);");

          // заполняем ее
          for (int i = 0; i < position_id.length; i++) {
            cv.clear();
            cv.put("id", position_id[i]);
            cv.put("name", position_name[i]);
            cv.put("salary", position_salary[i]);
            db.insert("position", null, cv);
          }

          db.execSQL("alter table people add column posid integer;");

          for (int i = 0; i < position_id.length; i++) {
```

```
            cv.clear();
            cv.put("posid", position_id[i]);
            db.update("people", cv, "position = ?",
                new String[] { position_name[i] });
        }

        db.execSQL("create temporary table people_tmp ("
            + "id integer, name text, position text, posid integer);");

        db.execSQL("insert into people_tmp select id, name, position, posid from people;");
        db.execSQL("drop table people;");

        db.execSQL("create table people ("
            + "id integer primary key autoincrement,"
            + "name text, posid integer);");

        db.execSQL("insert into people select id, name, posid from people_tmp;");
        db.execSQL("drop table people_tmp;");

        db.setTransactionSuccessful();
      } finally {
        db.endTransaction();
      }
    }
  }
```

Все в соответствии с планом обновления, который я приводил выше. Есть пара нюансов.

Во-первых, используем **БД-транзакцию**. Т.е. нам надо чтобы на БД накатились **все** наши обновления. А в случае **ошибки** в процессе обновления - **все изменения должны быть отменены** и БД должна остаться прежней. Тут транзакции очень выручают.

Во-вторых, в SQLite **нельзя** просто так **удалить** столбец, приходится создавать временную таблицу, перекидывать туда данные, удалять оригинал, создавать его снова с нужной структурой, скидывать в него данные из временной таблицы и удалять временную таблицу. Подробнее об этом можно почитать тут - How do I add or delete columns from an existing table in SQLite.

Наше приложение **обновилось**. И теперь, при запуске, оно попытается подключиться к БД версии **2**, но увидит, что существующая версия = **1** и вызовет метод **onUpgrade**, дав нам возможность внести необходимые изменения в структуру БД. Но это произойдет в случае обновления приложения. А что будет если пользователь поставит наше новое приложение на свежий смартфон первый раз?

В этом случае приложение также попытается подключиться к БД версии **2**. Но т.к. приложение только что установлено, то БД еще не существует. Приложение **создаст** БД и присвоит ей версию номер **2**, т.к. оно умеет работать именно с такой версией. При создании будет вызван метод **onCreate** в **DBHelper**. Значит, в нем мы должны прописать код, который будет создавать нам БД версии **2** – т.е. обновленную таблицу **people** и новую таблицу **position**.

Пишем **onCreate** в DBHelper:

```
  public void onCreate(SQLiteDatabase db) {
    Log.d(LOG_TAG, " --- onCreate database --- ");

    String[] people_name = { "Иван", "Марья", "Петр", "Антон", "Даша",
        "Борис", "Костя", "Игорь" };
    int[] people_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };

    // данные для таблицы должностей
    int[] position_id = { 1, 2, 3, 4 };
```

```
        String[] position_name = { "Директор", "Программер", "Бухгалтер",
            "Охранник" };
        int[] position_salary = { 15000, 13000, 10000, 8000 };

        ContentValues cv = new ContentValues();

        // создаем таблицу должностей
        db.execSQL("create table position (" + "id integer primary key,"
            + "name text, salary integer" + ");");

        // заполняем ее
        for (int i = 0; i < position_id.length; i++) {
          cv.clear();
          cv.put("id", position_id[i]);
          cv.put("name", position_name[i]);
          cv.put("salary", position_salary[i]);
          db.insert("position", null, cv);
        }

        // создаем таблицу людей
        db.execSQL("create table people ("
            + "id integer primary key autoincrement,"
            + "name text, posid integer);");

        // заполняем ее
        for (int i = 0; i < people_name.length; i++) {
          cv.clear();
          cv.put("name", people_name[i]);
          cv.put("posid", people_posid[i]);
          db.insert("people", null, cv);
        }
      }
```

Создание и заполнение данными двух таблиц. Все понятно.

Теперь можно все сохранить и запустить приложение.

Смотрим лог:

--- onUpgrade database from 1 to 2 version ---
--- Staff db v.2 ---
Table people. 8 rows
id = 1; name = Иван; posid = 2;
id = 2; name = Марья; posid = 3;
id = 3; name = Петр; posid = 2;
id = 4; name = Антон; posid = 2;
id = 5; name = Даша; posid = 3;
id = 6; name = Борис; posid = 1;
id = 7; name = Костя; posid = 2;
id = 8; name = Игорь; posid = 4;
Table position. 4 rows
id = 1; name = Директор; salary = 15000;
id = 2; name = Программер; salary = 13000;
id = 3; name = Бухгалтер; salary = 10000;
id = 4; name = Охранник; salary = 8000;
inner join. 8 rows

Name = Иван; Position = Программер; Salary = 13000;
Name = Марья; Position = Бухгалтер; Salary = 10000;
Name = Петр; Position = Программер; Salary = 13000;
Name = Антон; Position = Программер; Salary = 13000;
Name = Даша; Position = Бухгалтер; Salary = 10000;
Name = Борис; Position = Директор; Salary = 15000;
Name = Костя; Position = Программер; Salary = 13000;
Name = Игорь; Position = Охранник; Salary = 8000;

Видим, что вызывался **onUpgrade** и **обновил** нам БД с версии **1** на **2**. Далее выводим все данные, чтобы убедиться, что обновление прошло корректно.

Можно также убедиться, что новый onCreate в DBHelper корректно отработает. Для этого надо удалить файл БД и запустить приложение. Приложение не найдет БД и создаст ее сразу в новом формате и с версией 2.

Сценарий выдуманный, есть к чему придраться и о чем поспорить, но смысл не в этом. Смысл в том, что мы увидели, как происходит **обновление** БД, если приложение запросило новую версию. Поначалу, возможно, покажется запутанным этот механизм создания и обновления. Но сложного реально ничего нет. С опытом придет полное понимание.

Еще хочу отметить, что у объекта Cursor есть метод close(), который освобождает занимаемые им ресурсы. Не забывайте про него.

Думаю, теперь можно смело сказать, что работу с **SQLite** в Android мы изучили достаточно основательно. И в дальнейших уроках сможем свободно использовать эти знания.

Полный код **MainActivity.java**:

```
package ru.startandroid.develop.p0391sqliteonupgradedb;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final String DB_NAME = "staff"; // имя БД
  final int DB_VERSION = 2; // версия БД

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.main);

    DBHelper dbh = new DBHelper(this);
    SQLiteDatabase db = dbh.getWritableDatabase();
    Log.d(LOG_TAG, " --- Staff db v." + db.getVersion() + " --- ");
    writeStaff(db);
    dbh.close();
  }

  // запрос данных и вывод в лог
  private void writeStaff(SQLiteDatabase db) {
    Cursor c = db.rawQuery("select * from people", null);
    logCursor(c, "Table people");

    c = db.rawQuery("select * from position", null);
    logCursor(c, "Table position");

    String sqlQuery = "select PL.name as Name, PS.name as Position, salary as Salary "
        + "from people as PL "
        + "inner join position as PS "
        + "on PL.posid = PS.id ";
    c = db.rawQuery(sqlQuery, null);
    logCursor(c, "inner join");
  }

  // вывод в лог данных из курсора
  void logCursor(Cursor c, String title) {
    if (c != null) {
      if (c.moveToFirst()) {
        Log.d(LOG_TAG, title + ". " + c.getCount() + " rows");
        StringBuilder sb = new StringBuilder();
        do {
          sb.setLength(0);
          for (String cn : c.getColumnNames()) {
            sb.append(cn + " = "
                + c.getString(c.getColumnIndex(cn)) + "; ");
          }
          Log.d(LOG_TAG, sb.toString());
        } while (c.moveToNext());
      }
    } else
      Log.d(LOG_TAG, title + ". Cursor is null");
  }

  // класс для работы с БД
  class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context) {
      super(context, DB_NAME, null, DB_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
      Log.d(LOG_TAG, " --- onCreate database --- ");

      String[] people_name = { "Иван", "Марья", "Петр", "Антон", "Даша",
          "Борис", "Костя", "Игорь" };
      int[] people_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };

      // данные для таблицы должностей
      int[] position_id = { 1, 2, 3, 4 };
```

```
    String[] position_name = { "Директор", "Программер", "Бухгалтер",
        "Охранник" };
    int[] position_salary = { 15000, 13000, 10000, 8000 };

    ContentValues cv = new ContentValues();

    // создаем таблицу должностей
    db.execSQL("create table position (" + "id integer primary key,"
        + "name text, salary integer" + ");");

    // заполняем ее
    for (int i = 0; i < position_id.length; i++) {
      cv.clear();
      cv.put("id", position_id[i]);
      cv.put("name", position_name[i]);
      cv.put("salary", position_salary[i]);
      db.insert("position", null, cv);
    }

    // создаем таблицу людей
    db.execSQL("create table people ("
        + "id integer primary key autoincrement,"
        + "name text, posid integer);");

    // заполняем ее
    for (int i = 0; i < people_name.length; i++) {
      cv.clear();
      cv.put("name", people_name[i]);
      cv.put("posid", people_posid[i]);
      db.insert("people", null, cv);
    }
  }

  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.d(LOG_TAG, " --- onUpgrade database from " + oldVersion
        + " to " + newVersion + " version --- ");

    if (oldVersion == 1 && newVersion == 2) {

      ContentValues cv = new ContentValues();

      // данные для таблицы должностей
      int[] position_id = { 1, 2, 3, 4 };
      String[] position_name = { "Директор", "Программер",
          "Бухгалтер", "Охранник" };
      int[] position_salary = { 15000, 13000, 10000, 8000 };

      db.beginTransaction();
      try {
        // создаем таблицу должностей
        db.execSQL("create table position ("
            + "id integer primary key,"
            + "name text, salary integer);");

        // заполняем ее
        for (int i = 0; i < position_id.length; i++) {
          cv.clear();
          cv.put("id", position_id[i]);
          cv.put("name", position_name[i]);
          cv.put("salary", position_salary[i]);
```

```
            db.insert("position", null, cv);
          }

          db.execSQL("alter table people add column posid integer;");

          for (int i = 0; i < position_id.length; i++) {
            cv.clear();
            cv.put("posid", position_id[i]);
            db.update("people", cv, "position = ?",
                new String[] { position_name[i] });
          }

          db.execSQL("create temporary table people_tmp ("
              + "id integer, name text, position text, posid integer);");

          db.execSQL("insert into people_tmp select id, name, position, posid from people;");
          db.execSQL("drop table people;");

          db.execSQL("create table people ("
              + "id integer primary key autoincrement,"
              + "name text, posid integer);");

          db.execSQL("insert into people select id, name, posid from people_tmp;");
          db.execSQL("drop table people_tmp;");

          db.setTransactionSuccessful();
        } finally {
          db.endTransaction();
        }
      }
    }

  }
}
```

На следующем уроке:

- разбираем как можно использовать LayoutInflater

Урок 40. LayoutInflater. Учимся использовать.

В этом уроке:

- разбираем как можно использовать LayoutInflater

После изучения SQLite самое время приступить к изучению списков – List. Но перед этим полезно будет узнать про **LayoutInflater**. Это знание пригодится нам в создании расширенных списков. Также перед этим уроком рекомендую снова прочесть урок про LayoutParams, освежить знания.

LayoutInflater – это класс, который умеет из содержимого layout-файла создать View-элемент. Метод который это делает называется **inflate**. Есть несколько реализаций этого метода с различными параметрами. Но все они используют друг друга и результат их выполнения один – **View**.

Мы рассмотрим эту реализацию – public View inflate (int resource, ViewGroup root, boolean attachToRoot)

Как видим, на вход метод принимает три параметра:

resource - ID layout-файла, который будет использован для создания View. Например - R.layout.main
root – родительский ViewGroup-элемент для создаваемого View. LayoutParams от этого ViewGroup присваиваются создаваемому View.
attachToRoot – присоединять ли создаваемый View к root. Если true, то root становится родителем создаваемого View. Т.е. это равносильно команде root.addView(View). Если false – то создаваемый View просто получает LayoutParams от root, но его дочерним элементом не становится.

Посмотрим на практике.

Создадим проект:

Project name: P0401_LayoutInflater
Build Target: Android 2.3.3
Application name: LayoutInflater
Package name: ru.startandroid.develop.p0401layoutinflater
Create Activity: MainActivity

Открываем **main.xml** и рисуем такой экран:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <LinearLayout
          android:id="@+id/linLayout"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_margin="20dp">
         <TextView
              android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="Linear Layout: ">
        </TextView>
    </LinearLayout>
    <RelativeLayout
         android:id="@+id/relLayout"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:layout_margin="20dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Relative Layout: ">
        </TextView>
    </RelativeLayout>
</LinearLayout>
```

На экране две ViewGroup - **linLayout** и **relLayout**. В них по TextView с соответствующим текстом.

Создадим еще один layout-файл **text.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/tvLayout"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:text="Layout with TextView">
</TextView>
```

Тут просто **TextView** без всяких **ViewGroup**. На нем мы и будем испытывать **LayoutInflater**.

Открываем **MainActivity.java** и пишем код:

```
package ru.startandroid.develop.p0401layoutinflater;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

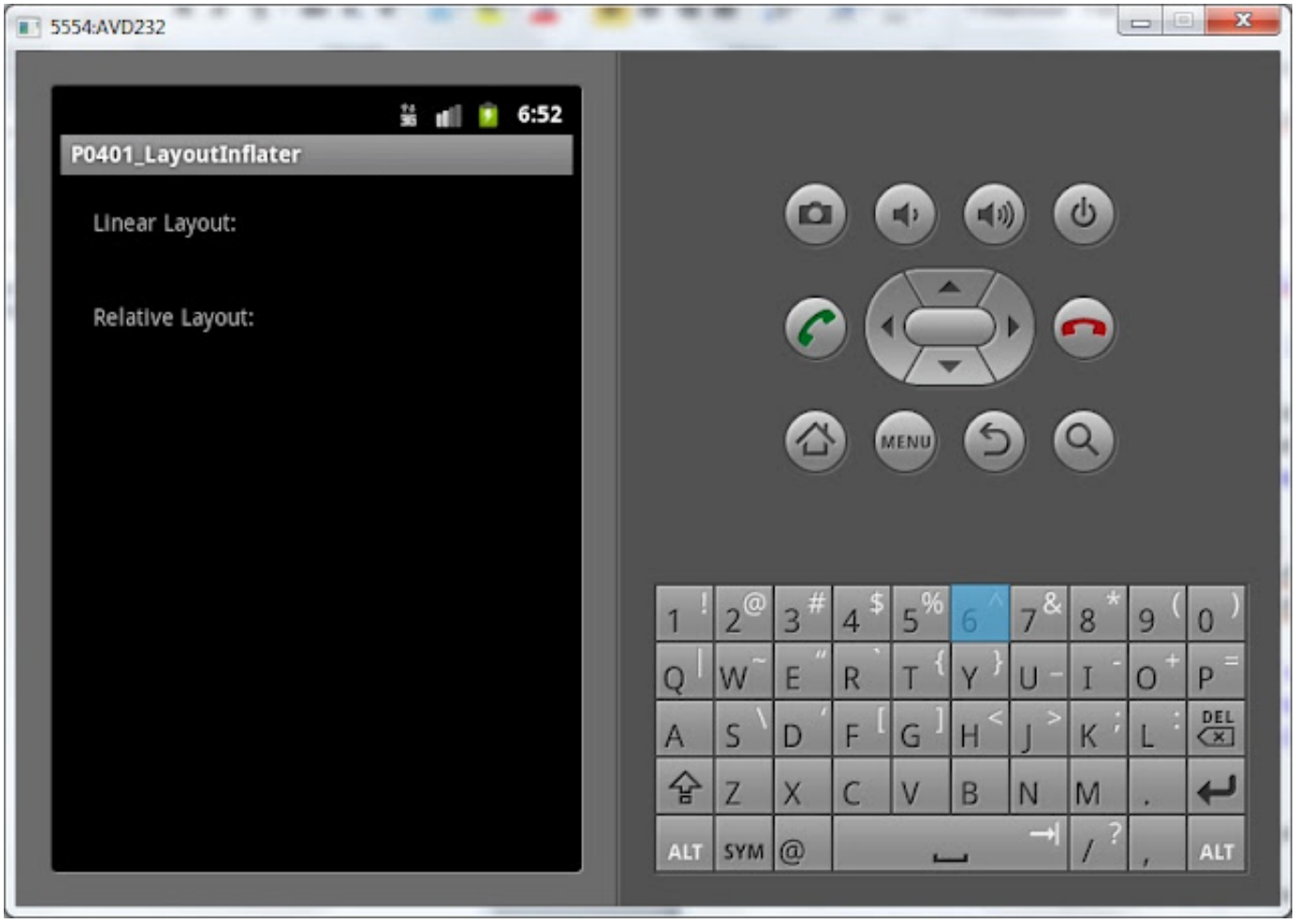
```
        setContentView(R.layout.main);

        LayoutInflater ltInflater = getLayoutInflater();
        View view = ltInflater.inflate(R.layout.text, null, false);
        LayoutParams lp = view.getLayoutParams();

        Log.d(LOG_TAG, "Class of view: " + view.getClass().toString());
        Log.d(LOG_TAG, "LayoutParams of view is null: " + (lp == null));
        Log.d(LOG_TAG, "Text of view: " + ((TextView) view).getText());
    }
}
```

Мы получаем **LayoutInflater** методом getLayoutInflater, используем его для получения **View**-элемента из layout-файла **text.xml** и считываем **LayoutParams** у свежесозданного view.

Обратите внимание, на параметры, которые мы использовали для метода **inflate**. Мы указали **ID** layout-ресурса, передали **null** в качестве родительского элемента и, соответственно, привязка к родителю - **false**.

Все сохраним и запустим.

На экране ничего не изменилось. Т.к. мы конвертнули layout в view, но никуда его не поместили. Он просто висит в памяти.



Смотрим лог:

Class of view: class android.widget.TextView
LayoutParams of view is null: true
Text of view: Layout with TextView

Мы видим класс созданного элемента - **TextView**. Все верно - этот элемент и был в файле **text.xml**. Далее видим **null** вместо **LayoutParams**. Это произошло потому, что родителя в методе inflate мы указали **null**. А именно от родителя **view** и должен был получить **LayoutParams**. Третья строка лога показывает текст TextView. Он тот же, что и в layout-файле **text.xml** – все верно.

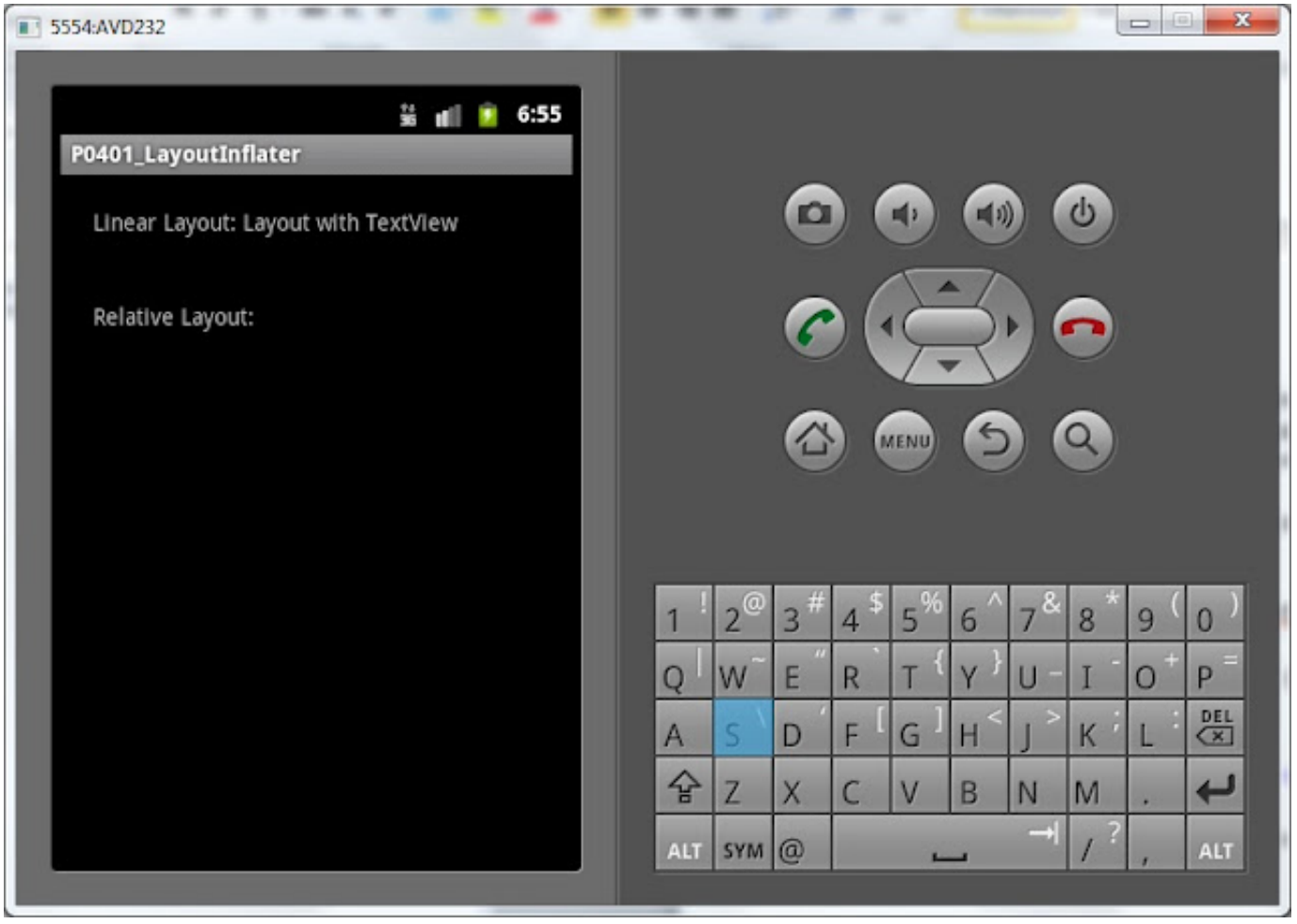
Давайте немного изменим программу. Будем добавлять наш созданный элемент в **linLayout** из **main.xml**. Делается это просто – командой **addView**.

```
LayoutParams lp = view.getLayoutParams();

LinearLayout linLayout = (LinearLayout) findViewById(R.id.linLayout);
linLayout.addView(view);

Log.d(LOG_TAG, "Class of view: " + view.getClass().toString());
```

(добавляете только ***жирный курсивный код***)

Мы нашли **linLayout** с экрана и добавили в него созданный с помощью LayoutInflater элемент.

Сохраняем, запускаем. Видим, что элемент добавился на экран в linLayout.

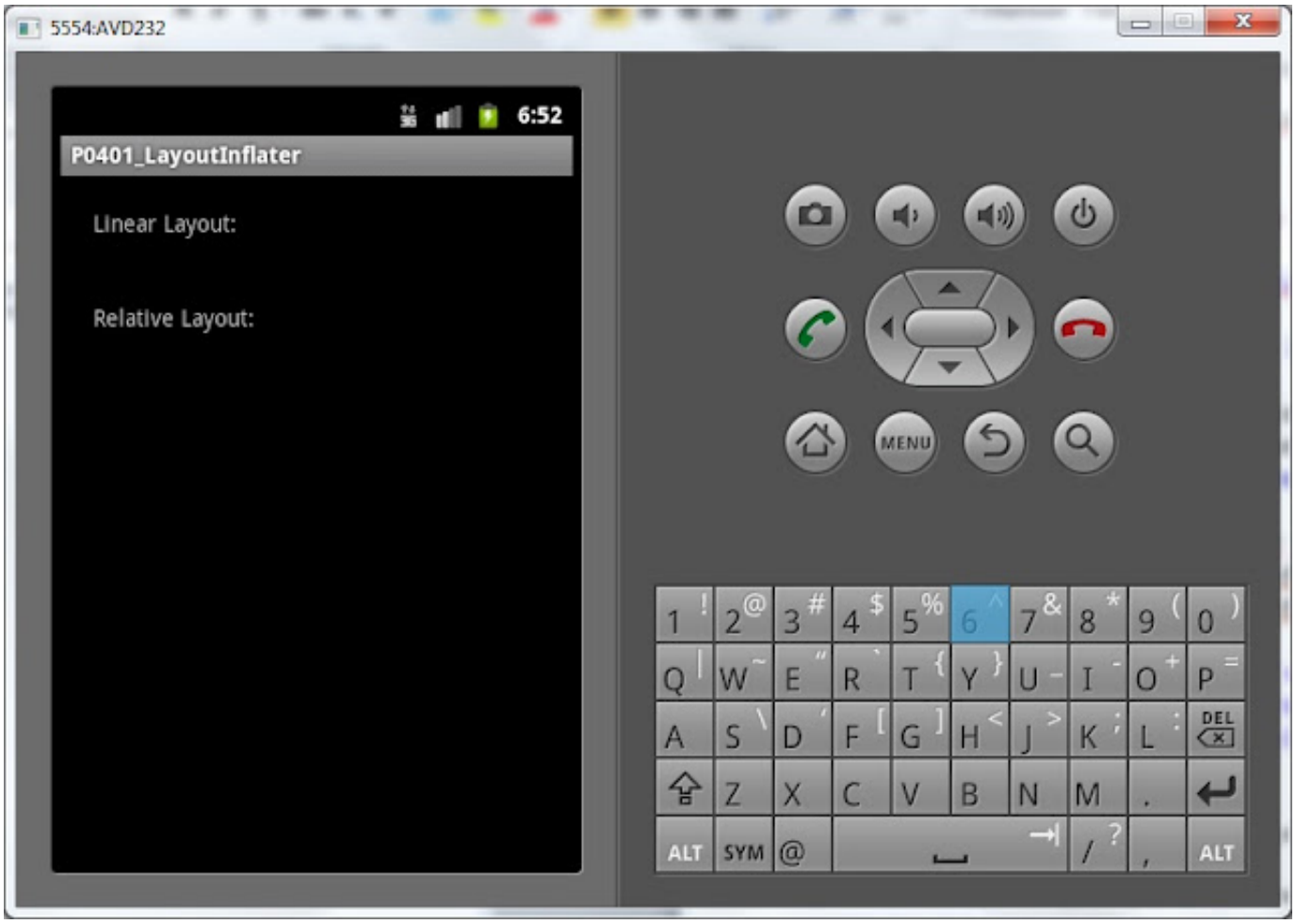


Теперь давайте попробуем указать родителя (**root**) при вызове метода **inflate**. Перепишем метод **onCreate**:

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  LayoutInflater ltInflater = getLayoutInflater();

  LinearLayout linLayout = (LinearLayout) findViewById(R.id.linLayout);
  View view1 = ltInflater.inflate(R.layout.text, linLayout, false);
  LayoutParams lp1 = view1.getLayoutParams();

  Log.d(LOG_TAG, "Class of view1: " + view1.getClass().toString());
  Log.d(LOG_TAG, "Class of layoutParams of view1: " + lp1.getClass().toString());
  Log.d(LOG_TAG, "Text of view1: " + ((TextView) view1).getText());

  RelativeLayout relLayout = (RelativeLayout) findViewById(R.id.relLayout);
  View view2 = ltInflater.inflate(R.layout.text, relLayout, false);
  LayoutParams lp2 = view2.getLayoutParams();

  Log.d(LOG_TAG, "Class of view2: " + view2.getClass().toString());
  Log.d(LOG_TAG, "Class of layoutParams of view2: " + lp2.getClass().toString());
  Log.d(LOG_TAG, "Text of view2: " + ((TextView) view2).getText());
}
```

Мы находим элементы **linLayout** и **relLayout** с экрана и с помощью **LayoutInflater** создаем два View-элемента из layout-файла text.xml. Для **первого** указываем root – **linLayout**, для **второго** – **relLayout**. Но третий параметр **attachToRoot** оставляем **false**. Это значит, что созданный **View**-элемент **получит LayoutParams** от **root**-элемента, но **не добавится** к нему.

Все сохраним, запустим. На экране ничего не поменялось. Т.к. мы ни к чему новые элементы не добавляли и **attachToRoot** = **false**.



Смотрим лог:

Class of view1: class android.widget.TextView
Class of layoutParams of view1: class android.widget.LinearLayout$LayoutParams
Text of view1: Layout with TextView
Class of view2: class android.widget.TextView
Class of layoutParams of view2: class android.widget.RelativeLayout$LayoutParams
Text of view2: Layout with TextView

По логам видно, что класс созданных элементов – **TextView**. А класс **LayoutParams** различается. В **первом** случае – это **LinearLayout$LayoutParams**, т.к. в качестве **root** элемента в методе inflate мы указали **linLayout**, а это объект класса **LinearLayout**. Во втором случае класс **LayoutParams** у созданного элемента - **RelativeLayout$LayoutParams**. Потому, что в качестве root указали **relLayout** (класс **RelativeLayout**).

Теперь у нас два варианта, как добавить созданные **view1** и **view2** на экран.

1) Снова использовать методы **addView**

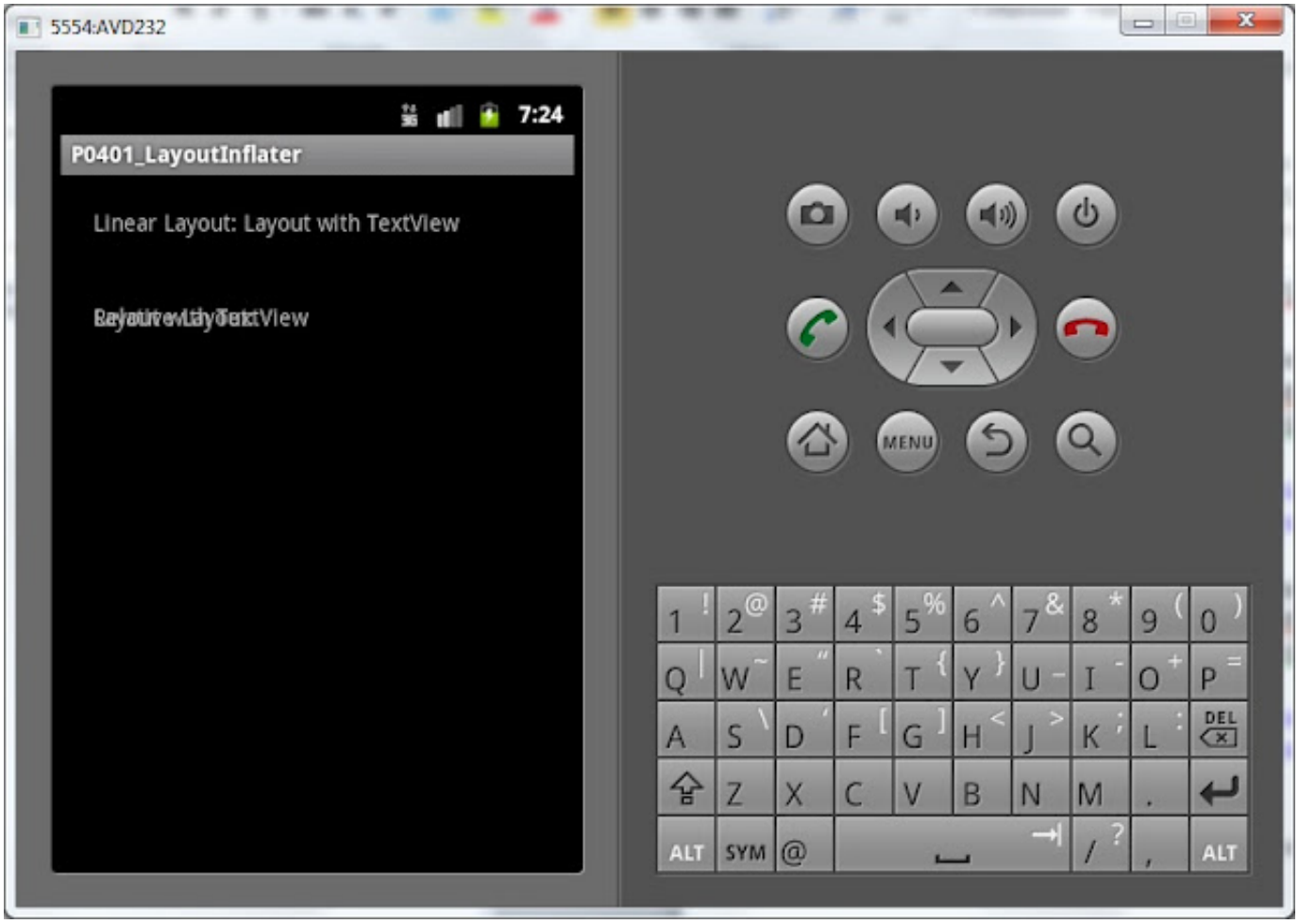
2) Передавать **true** в качестве третьего параметра метода **inflate**. Тогда созданный **View**-элемент будет добавлен к root.

Выберем второй вариант и внесем изменения в код:

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  LayoutInflater ltInflater = getLayoutInflater();

  LinearLayout linLayout = (LinearLayout) findViewById(R.id.linLayout);
  View view1 = ltInflater.inflate(R.layout.text, linLayout, true);
  LayoutParams lp1 = view1.getLayoutParams();

  Log.d(LOG_TAG, "Class of view1: " + view1.getClass().toString());
  Log.d(LOG_TAG, "Class of layoutParams of view1: " + lp1.getClass().toString());

  RelativeLayout relLayout = (RelativeLayout) findViewById(R.id.relLayout);
  View view2 = ltInflater.inflate(R.layout.text, relLayout, true);
  LayoutParams lp2 = view2.getLayoutParams();

  Log.d(LOG_TAG, "Class of view2: " + view2.getClass().toString());
  Log.d(LOG_TAG, "Class of layoutParams of view2: " + lp2.getClass().toString());
}
```

Передаем **true** в качестве **третьего** параметра в методе **inflate** и убираем строки выведения в лог текстов из TextView. Сейчас будет понятно почему.

Все сохраним и запустим приложение.



Как видим, созданные **TextView** появились в своих родителях, которых мы указали в методе **inflate**. В **RelativeLayout** элементы наложились друг на друга, т.к. мы не настроили расположение. В данный момент это не существенно.

Смотрим лог:

Class of view1: class android.widget.LinearLayout
Class of layoutParams of view1: class android.widget.LinearLayout$LayoutParams
Class of view2: class android.widget.RelativeLayout
Class of layoutParams of view2: class android.widget.LinearLayout$LayoutParams

Обратите внимание на класс элементов. В первом случае - это **LinearLayout**, а во втором - **RelativeLayout**. Т.е. метод **inflate** вернул нам **не созданные** из layout-файла View-элементы, а те, что мы указывали как **root**. А **созданные** из layout-файла **View** элементы он **добавил** в **root** как дочерние аналогично команде **addView**. Это произошло потому, что мы указали **true** в третьем параметре (attachToRoot) метода inflate.

Соответственно **LayoutParams** для **view1** и **view2** будет **LinearLayout$LayoutParams**, т.к. **linLayout** и **relLayout** имеют родителя **LinearLayout**. И **LayoutParams** берут от него.

Для закрепления темы на следующем уроке сделаем пример поинтереснее.

На следующем уроке:

- делаем свой вариант списка

Урок 41. Используем LayoutInflater для создания списка

В этом уроке:

- делаем свой вариант списка

На прошлом уроке мы узнали, зачем нужен класс **LayoutInflater** и сделали небольшой пример, на котором подробно рассмотрели метод **inflate** и его параметры. Для закрепления темы сделаем еще один, чуть более сложный пример.

Мы сделаем свой аналог **списка**. Для начала придумаем данные. Пусть это снова будет **штатное расписание** с именами работников, должностями и зряплатой) Т.е. каждый пункт нашего списка будет содержать три текстовых не редактируемых поля - **name**, **position**, **salary**. А пункты мы разместим в виде вертикального списка.

Для реализации нам понадобятся **два layout**-файла:
main.xml - основной экран для Activity, контейнер для пунктов списка
item.xml - экран с FrameLayout и тремя текстовыми полями в нем. Это будет пункт списка.

Приложение будет параллельно перебирать **три массива данных**, **создавать** для каждой тройки **View**-элемент из layout-файла **item.xml**, **заполнять** его данными и **добавлять** в вертикальный **LinearLayout** в **main.xml**.

Создадим проект:

Project name: P0411_LayoutInflaterList
Build Target: Android 2.3.3
Application name: LayoutInflaterList
Package name: ru.startandroid.develop.p0411layoutinflaterlist
Create Activity: MainActivity

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Staff list"
          android:layout_gravity="center_horizontal">
     </TextView>
     <ScrollView
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:id="@+id/scroll">
         <LinearLayout
              android:id="@+id/linLayout"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
```

```
            android:orientation="vertical">
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

ScrollView обеспечит нам **прокрутку** списка, если все пункты не влезут в экран. А в нем **LinearLayout**, в который мы будем добавлять элементы.

Экран **item.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="50dp"
     android:layout_marginTop="10dp">
    <TextView
         android:id="@+id/tvName"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="TextView"
         android:layout_gravity="top|center_horizontal"
         android:textSize="24sp">
    </TextView>
    <TextView
         android:id="@+id/tvPosition"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="TextView"
         android:layout_gravity="bottom|left"
         android:layout_marginLeft="5dp">
    </TextView>
    <TextView
         android:id="@+id/tvSalary"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="TextView"
         android:layout_gravity="bottom|right"
         android:layout_marginRight="5dp">
    </TextView>
</FrameLayout>
```

FrameLayout, и три **TextView** в нем.

Кодим реализацию. **MainActivity.java**:

```
package ru.startandroid.develop.p0411layoutinflaterlist;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
```

```
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends Activity {

  String[] name = { "Иван", "Марья", "Петр", "Антон", "Даша", "Борис",
      "Костя", "Игорь" };
  String[] position = { "Программер", "Бухгалтер", "Программер",
      "Программер", "Бухгалтер", "Директор", "Программер", "Охранник" };
  int salary[] = { 13000, 10000, 13000, 13000, 10000, 15000, 13000, 8000 };

  int[] colors = new int[2];

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    colors[0] = Color.parseColor("#559966CC");
    colors[1] = Color.parseColor("#55336699");

    LinearLayout linLayout = (LinearLayout) findViewById(R.id.linLayout);

    LayoutInflater ltInflater = getLayoutInflater();

    for (int i = 0; i < name.length; i++) {
      Log.d("myLogs", "i = " + i);
      View item = ltInflater.inflate(R.layout.item, linLayout, false);
      TextView tvName = (TextView) item.findViewById(R.id.tvName);
      tvName.setText(name[i]);
      TextView tvPosition = (TextView) item.findViewById(R.id.tvPosition);
      tvPosition.setText("Должность: " + position[i]);
      TextView tvSalary = (TextView) item.findViewById(R.id.tvSalary);
      tvSalary.setText("Оклад: " + String.valueOf(salary[i]));
      item.getLayoutParams().width = LayoutParams.MATCH_PARENT;
      item.setBackgroundColor(colors[i % 2]);
      linLayout.addView(item);
    }
  }
}
```

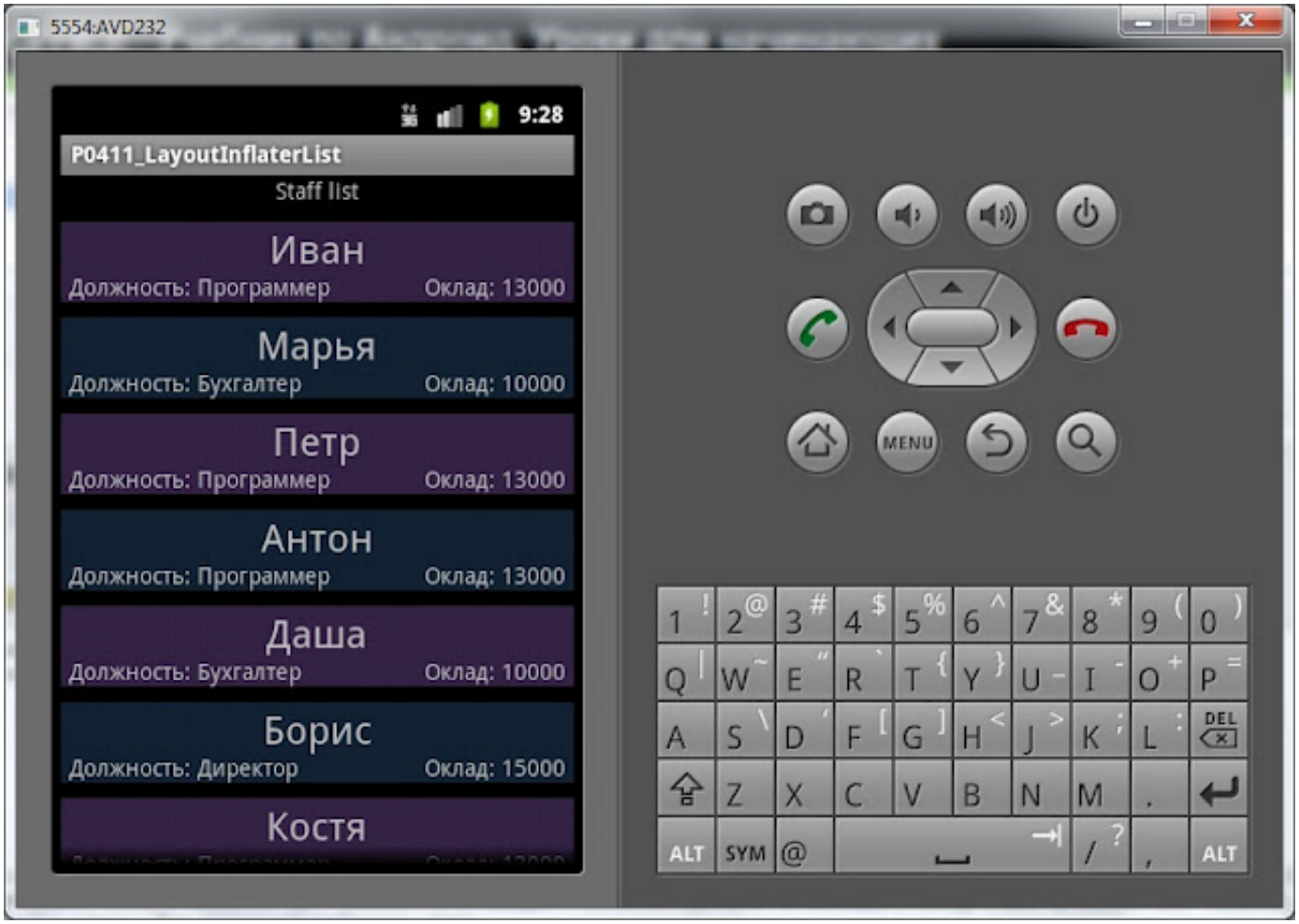
Не так уж много нужно кода, чтобы сделать несложный список. Мы запускаем **цикл** по кол-ву элементов в массивах данных. В каждой итерации **создаем View**-элемент item из layout-файла **item.xml**. В нашем случае **item** - это **FrameLayout**, который содержит три **TextView**. Мы их находим в **созданном item** и **заполняем данными** из массивов.

В методе **inflate** мы указали **root** - **linLayout**, чтобы получить от него **LayoutParams** и далее использовать для настройки ширины. Также для наглядности раскрашиваем пункты методом setBackgroundColor.

Обратите внимание - третий параметр **inflate** мы указали **false**. Т.е. мы **не стали сразу добавлять** создаваемый View-элемент к linLayout, а делаем это в конце кода методом **addView**. Этому есть объяснение. Если бы мы указали true - то метод добавил бы item к linLayout и вернул бы нам **linLayout**, общий для всех пунктов списка. Через **linLayout** заполнять **TextView** необходимым нам текстом было бы затруднительно. Поэтому мы получаем пункт

item (FrameLayout), заполняем его **TextView** данными и только потом помещаем к остальным пунктам в **linLayout** методом addView.

Все сохраним и запустим:



Список удался и работает прокрутка.

Урок получился короткий, но полезный. На всякий случай хочу заметить, что это еще не классический Android-список называемый List. Но этот пример значительно облегчит понимание списка. Т.к. принцип схож. Для построения List мы также должны будем предоставлять массив данных и layout-файл для пунктов. Этим и займемся на следующем уроке.

Этот урок был придуман и написан на высоте 10 км., в самолете на пути в Москву. Через ряд, слева от меня сидел парень с рюкзаком java, распечатанными доками, ноутом и чего-то кодил под Android.

Android повсюду :)

На следующем уроке:

- используем ListView для построения списка

Урок 42. Список - ListView

В этом уроке:

- используем ListView для построения списка

Перед тем, как начать говорить про компонент **ListView**, предлагаю вспомнить еще раз прошлый урок и механизм построения списка, который мы там использовали. Мы перебирали массив данных, в каждой итерации создавали **пункт** списка, заполняли его **данными** и помещали в **список**.

При создании **ListView** создавать пункты за нас будет **адаптер**. Адаптеру нужны от нас **данные** и **layout-ресурс** пункта списка. Далее мы присваиваем **адаптер** списку **ListView**. Список при построении запрашивает у адаптера пункты, адаптер их создает (используя данные и layout) и возвращает списку. В итоге мы видим готовый список.

Есть различные типы списков и адаптеров. Мы пока что рассмотрим простейший вариант.

Создадим проект:

Project name: P0421_SimpleList
Build Target: Android 2.3.3
Application name: SimpleList
Package name: ru.startandroid.develop.p0421simplelist
Create Activity: MainActivity

Открываем **main.xml** и добавим на экран компонент **ListView** (вкладка Composite):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
    <TextView
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="@string/hello">
    </TextView>
    <ListView
          android:id="@+id/lvMain"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
    </ListView>
</LinearLayout>
```

ListView – это и есть компонент списка.

Теперь надо создать **адаптер**. Открываем **MainActivity.java** и пишем:

```
package ru.startandroid.develop.p0421simplelist;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {

  String[] names = { "Иван", "Марья", "Петр", "Антон", "Даша", "Борис",
      "Костя", "Игорь", "Анна", "Денис", "Андрей" };

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // находим список
    ListView lvMain = (ListView) findViewById(R.id.lvMain);

    // создаем адаптер
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, names);

    // присваиваем адаптер списку
    lvMain.setAdapter(adapter);

  }
}
```

Вы не поверите, но это весь код, необходимый для создания списка)

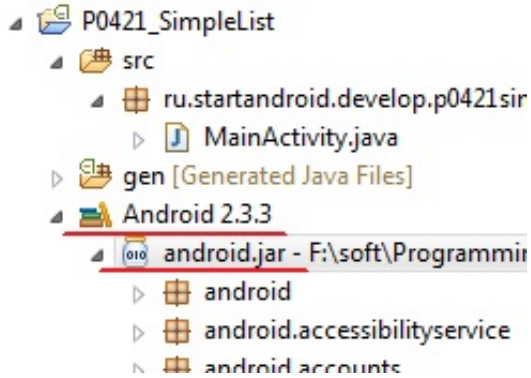
В качестве данных используем массив имен. В onCreate мы находим список, создаем адаптер и присваиваем адаптер списку. Давайте разберемся, как создали адаптер.

Мы использовали этот конструктор: public ArrayAdapter (Context context, int textViewResourceId, T[] objects)

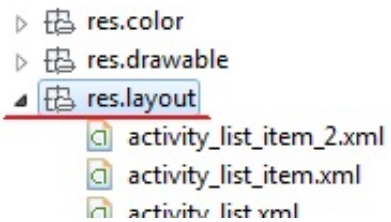
и передали ему следующие параметры:

this – контекст
android.R.layout.simple_list_item_1 – это системный layout-файл, который представляет собой TextView
names – массив данных, которые мы хотим вывести в список

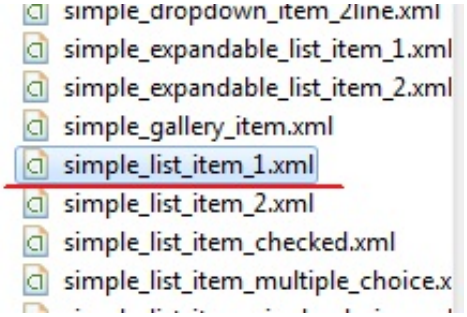
Мы можем посмотреть содержимое использованного **simple_list_item_1**. Для этого в вашем проекте найдите пункт **Android 2.3.3**., раскройте его, и раскройте **android.jar**



Проматывайте в самый низ и открывайте **res.layout**.



И внутри находим используемый нами **simple_list_item_1**



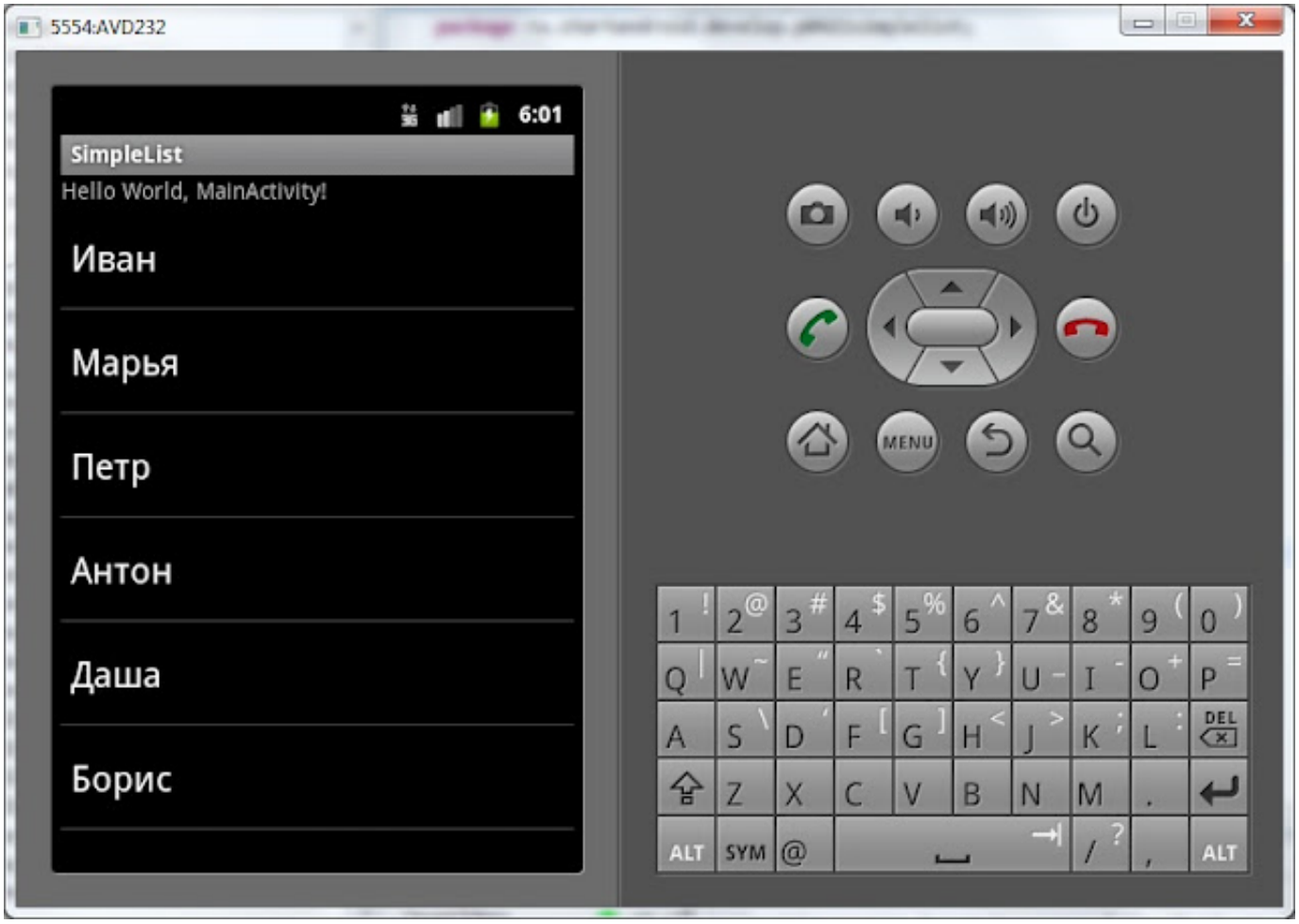
Двойной клик на него и смотрим содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@android:id/text1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:gravity="center_vertical"
      android:paddingLeft="6dip"
      android:minHeight="?android:attr/listPreferredItemHeight">
</TextView>
```

Обычный TextView с набором параметров.

Когда список при формировании запрашивает очередной пункт, адаптер берет этот Layout-ресурс **simple_list_item_1**, прогоняет его через **LayoutInflater** и получает **View**, преобразует View к TextView, присваивает ему **текст** из массива данных и отдает **списку**.

Все сохраним и запустим. Видим список из наших данных.



Использование системного layout-ресурса simple_list_item_1 хорошо тем, что нам не надо самим layout рисовать. Однако, если нас не устраивает то, как выглядит список с использованием simple_list_item_1 в качестве пункта списка, мы можем создать **свой layout-ресурс**.

Создадим layout-файл **my_list_item.xml** в папке res/layout нашего проекта:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/textView1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center_horizontal"
      android:padding="5dp"
      android:text="TextView"
      android:textColor="#00FF00"
      android:textSize="24sp">
</TextView>
```

TextView с указанием цвета и размера шрифта, выравнивания текста и отступов.

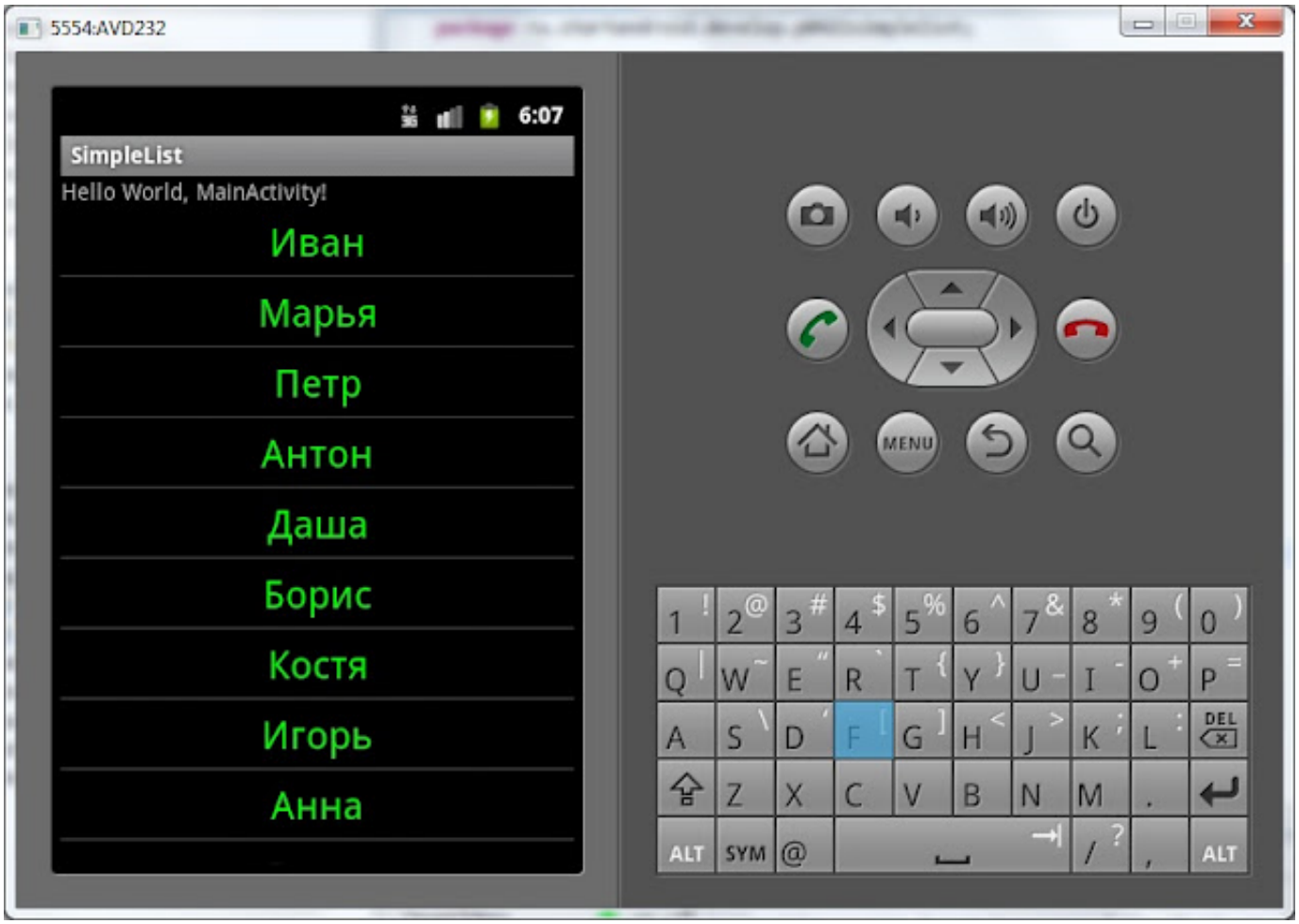
Изменим существующий код, укажем адаптеру наш созданный layout-ресурс **my_list_item**:

```
    // создаем адаптер
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
```

```
        R.layout.my_list_item, names);
```

Теперь адаптер будет использовать его при создании пунктов списка.

Все сохраним и запустим. Видим наш зеленый список:



В layout-ресурсе для пункта списка вместо **TextView** вы можете использовать какой-нибудь его производный класс – например **Button**. Главное, чтобы объект прошел **преобразование** к **TextView**. Адаптер присвоит ему текст методом **setText** и отдаст списку.

Немного про Context

На одном из прошлых уроков я говорил, что **Context** (контекст) используется для доступа к **базовым** функциям приложения. В этом уроке у нас получилось хорошее подтверждение этим словам.

ArrayAdapter использует LayoutInflater, чтобы конвертнуть layout-ресурс в View. Но получение объекта LayoutInflater – это одна из **базовых** функций и она **недоступна** для класса **ArrayAdapter**. Поэтому мы в ArrayAdapter в качестве контекста передаем ссылку на Activity (Activity имеет доступ к базовым функциям через восходящую иерархию классов). А класс ArrayAdapter внутри себя **использует** переданный ему **контекст**, чтобы вызвать LayoutInflater. Без контекста он не смог бы это сделать.

На следующем уроке:

- используем список ListView для одиночного и множественного выбора элементов

Урок 43. Одиночный и множественный выбор в ListView

В этом уроке:

- используем список ListView для одиночного и множественного выбора элементов

Бывает необходимость предоставить пользователю возможность выбрать один или несколько элементов из списка. Давайте посмотрим, как это можно реализовать.

Создадим проект:

Project name: P0431_SimpleListChoice
Build Target: Android 2.3.3
Application name: SimpleListChoice
Package name: ru.startandroid.develop.p0431simplelistchoice
Create Activity: MainActivity

Нарисуем экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
          android:id="@+id/btnChecked"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Get checked items">
     </Button>
     <ListView
          android:id="@+id/lvMain"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
     </ListView>
</LinearLayout>
```

Список **lvMain** и кнопка **btnChecked**, по нажатию, на которую будем выводить в лог отмеченные в списке элементы.

Предлагаю вспомнить, что у нас есть файлы ресурсов и мы можем их использовать. Найдем в нашем проекте файл с ресурсами **res/values/strings.xml** и добавим туда массив строк с именами. В итоге у меня получился файл с таким содержимым:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="hello">Hello World, MainActivity!</string>
    <string  name="app_name">SimpleListChoice</string>
    <string-array  name="names">
        <item>Иван</item>
        <item>Марья</item>
```

```
        <item>Петр</item>
        <item>Антон</item>
        <item>Даша</item>
        <item>Борис</item>
        <item>Костя</item>
        <item>Игорь</item>
        <item>Анна</item>
        <item>Денис</item>
        <item>Вадим</item>
        <item>Ольга</item>
        <item>Сергей</item>
    </string-array>
</resources>
```

Из этого списка мы будем получать массив имен. Это удобнее и правильнее, чем перечислять все элементы массива в java-коде.

Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p0431simplelistchoice;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;

public class MainActivity extends Activity implements OnClickListener {

  final String LOG_TAG = "myLogs";

  ListView lvMain;
  String[] names;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    lvMain = (ListView) findViewById(R.id.lvMain);
    // устанавливаем режим выбора пунктов списка
    lvMain.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    // Создаем адаптер, используя массив из файла ресурсов
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
        this, R.array.names,
        android.R.layout.simple_list_item_single_choice);
    lvMain.setAdapter(adapter);

    Button btnChecked = (Button) findViewById(R.id.btnChecked);
    btnChecked.setOnClickListener(this);

    // получаем массив из файла ресурсов
```

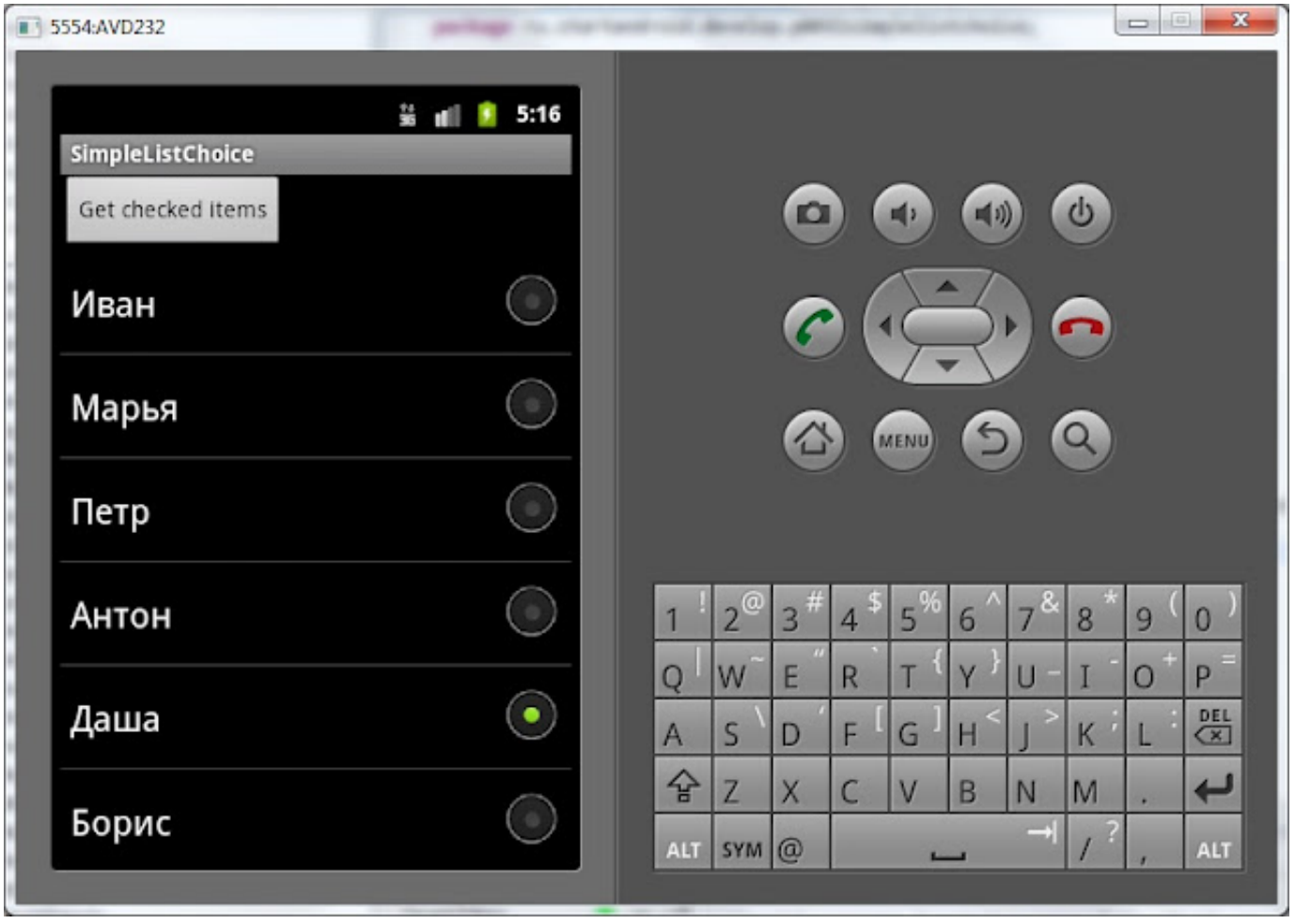
```
    names = getResources().getStringArray(R.array.names);
  }

  public void onClick(View arg0) {
    // пишем в лог выделенный элемент
    Log.d(LOG_TAG, "checked: " + names[lvMain.getCheckedItemPosition()]);
  }
}
```

Мы устанавливаем для списка режим выбора - **CHOICE_MODE_SINGLE**. Это значит, что список будет хранить **позицию** последнего **нажатого** пункта и мы всегда можем запросить у него эту информацию. Далее мы создаем **адаптер**, но не через обычный конструктор, а с использованием метода createFromResource. Параметры на вход почти те же, что и в обычном конструкторе, только вместо массива данных, мы указываем **массив строк в ресурсах**, который мы создали чуть раньше. В качестве layout-ресурса для пунктов используем системный **simple_list_item_single_choice**. Он как раз заточен под такое использование.

Далее мы, находим кнопку **btnChecked** и присваиваем ей **Activity** в качестве **обработчика**. И в конце считываем наш **массив имен** из **файла-ресурса** в массив строк.

В **обработчике** нажатия кнопки мы выводим в **лог имя** из массива. В качестве индекса используем позицию пункта в списке. Последовательность элементов в массиве и в списке совпадают.

Все сохраняем, запускаем и видим список. Выделяем какой-нить пункт:



Жмем кнопку **Get checked items** и смотрим лог:

checked: Даша

Все верно.

Теперь чуть изменим код программы и получим список с множественным выбором.

```
    // устанавливаем режим выбора пунктов списка
    lvMain.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
    // Создаем адаптер, используя массив из файла ресурсов
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
        this, R.array.names,
        android.R.layout.simple_list_item_multiple_choice);
```

(замените только жирный курсивный текст)

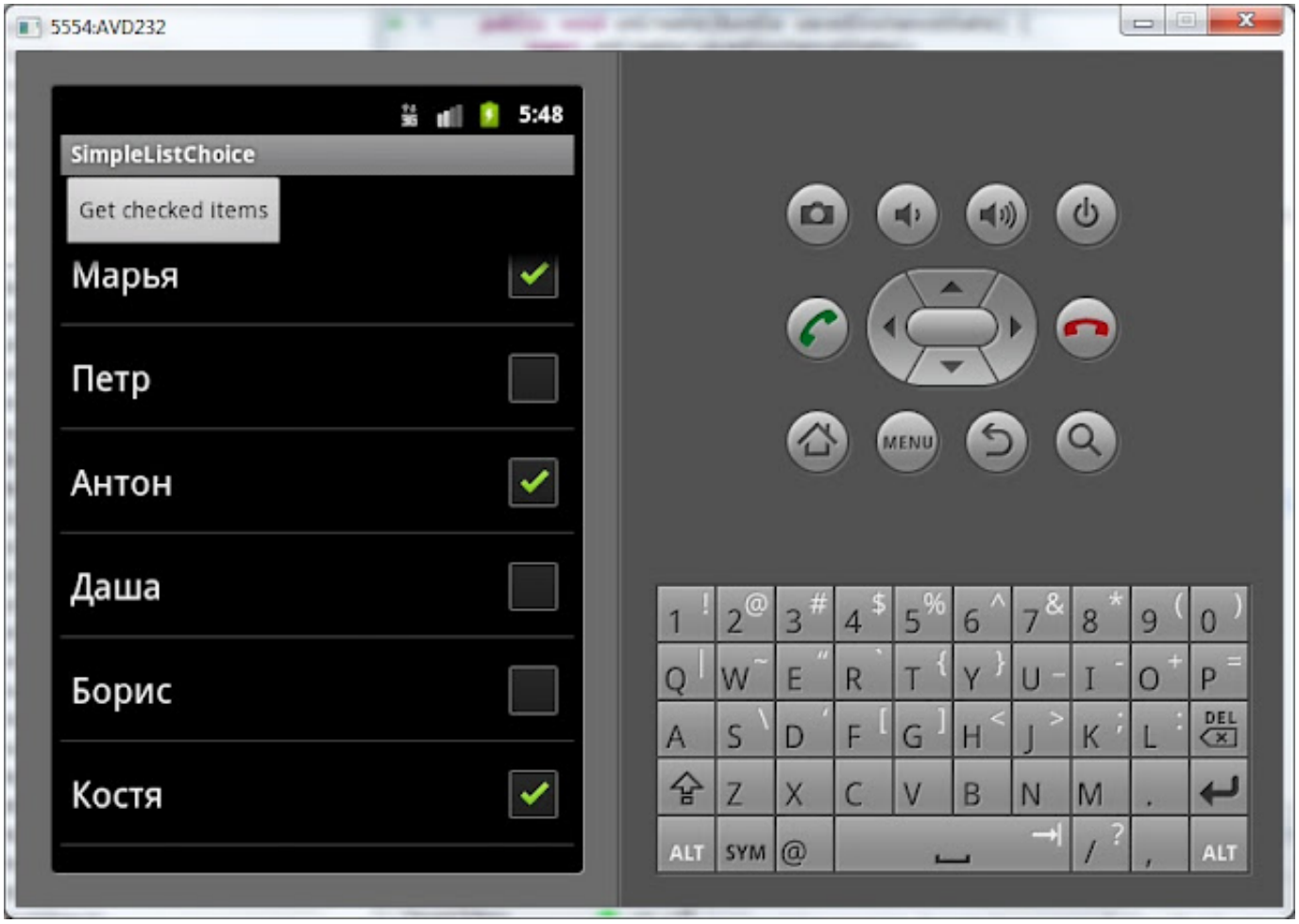
Мы заменили **CHOICE_MODE_SINGLE** на **CHOICE_MODE_MULTIPLE** и теперь список будет хранить позиции выделенных элементов. Также мы заменили **simple_list_item_single_choice** на **simple_list_item_multiple_choice** – пункты списка теперь будут позволять множественный выбор.

Метод **onClick** перепишем следующим образом:

```
  public void onClick(View arg0) {
    // пишем в лог выделенные элементы
    Log.d(LOG_TAG, "checked: ");
    SparseBooleanArray sbArray = lvMain.getCheckedItemPositions();
    for (int i = 0; i < sbArray.size(); i++) {
      int key = sbArray.keyAt(i);
      if (sbArray.get(key))
        Log.d(LOG_TAG, names[key]);
    }
  }
```

Мы получаем позиции выделенных элементов в виде объекта SparseBooleanArray. Он представляет собой **Map(int, boolean)**. **Ключ** (int) – это позиция элемента, а **значение** (boolean) – это выделен пункт списка или нет. Причем **SparseBooleanArray** хранит инфу **не о всех** пунктах, а только о тех, с которыми проводили **действие** (выделяли и снимали выделение). Мы перебираем его содержимое, получаем **позицию** пункта и, если пункт **выделен**, то выводим в лог **имя** из массива, соответствующее позиции пункта.

Все сохраняем, запускаем приложение. Выделяем несколько элементов:



Жмем кнопку **Get checked items** и смотрим лог:

checked:
Марья
Антон
Костя

Что мы выделили, то нам список и вернул.

Как видим, отличие обычного списка от списка с возможностью выделения пунктов заключается только в разных режимах **ChoiceMode** и в использовании разных **layout-ресурсов** для пунктов списка.

Кстати, здесь мы снова видим, зачем нужен **Context** при создании **адаптера**. Без него адаптер не смог бы добраться до **файла ресурсов**. Метода **getResources** у адаптера **нет**, и он использует указанный **контекст**, который содержит такой метод.

На следующем уроке:

- рассматриваем события ListView: нажатие - onItemClick, выделение - onItemSelect, прокрутка - onScroll

Урок 44. События в ListView

В этом уроке:

- рассматриваем события ListView: нажатие - onItemClick, выделение - onItemSelect, прокрутка - onScroll

При взаимодействии со списком может возникнуть необходимость обрабатывать события – нажатие на пункт и прокрутка. Попробуем это сделать.

Создадим проект:

Project name: P0441_SimpleListEvents
Build Target: Android 2.3.3
Application name: SimpleListEvents
Package name: ru.startandroid.develop.p0441simplelistevents
Create Activity: MainActivity

Нарисуем экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="@string/hello"
          android:id="@+id/tvText">
     </TextView>
     <ListView
          android:id="@+id/lvMain"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
     </ListView>
</LinearLayout>
```

На экране только **TextView** и **ListView**.

Так же, как и на прошлом уроке добавим список имен в ресурс **res/values/strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
     <string  name="hello">Hello World, MainActivity!</string>
     <string  name="app_name">SimpleListEvents</string>
     <string-array  name="names">
          <item>Иван</item>
```

```
        <item>Марья</item>
        <item>Петр</item>
        <item>Антон</item>
        <item>Даша</item>
        <item>Борис</item>
        <item>Костя</item>
        <item>Игорь</item>
        <item>Анна</item>
        <item>Денис</item>
        <item>Вадим</item>
        <item>Ольга</item>
        <item>Сергей</item>
    </string-array>
</resources>
```

Пишем код **MainActivity.java**:

```
package ru.startandroid.develop.p0441simplelistevents;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  ListView lvMain;
  TextView tvText;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    lvMain = (ListView) findViewById(R.id.lvMain);
    tvText = (TextView) findViewById(R.id.tvText);

    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
        this, R.array.names, android.R.layout.simple_list_item_1);
    lvMain.setAdapter(adapter);

    lvMain.setOnItemClickListener(new OnItemClickListener() {
      public void onItemClick(AdapterView<?> parent, View view,
          int position, long id) {
        Log.d(LOG_TAG, "itemClick: position = " + position + ", id = "
```

```
            + id);
      }
    });

    lvMain.setOnItemSelectedListener(new OnItemSelectedListener() {
      public void onItemSelected(AdapterView<?> parent, View view,
          int position, long id) {
        Log.d(LOG_TAG, "itemSelect: position = " + position + ", id = "
            + id);
      }

      public void onNothingSelected(AdapterView<?> parent) {
        Log.d(LOG_TAG, "itemSelect: nothing");
      }
    });

  }
}
```

Смотрим код. Мы находим экранные элементы, создаем и присваиваем списку адаптер. Далее списку мы присваиваем два обработчика событий:

1) **OnItemClickListener** – обрабатывает **нажатие** на пункт списка

Предоставляет нам метод onItemClick(AdapterView<?> parent, View view, int position, long id), где

parent – View-родитель для нажатого пункта, в нашем случае - ListView
view – это нажатый пункт, в нашем случае – TextView из android.R.layout.simple_list_item_1
position – порядковый номер пункта в списке
id – идентификатор элемента,

Мы в лог будем выводить **id** и **position** для элемента, на который нажали.

2) **OnItemSelectedListener** – обрабатывает **выделение** пунктов списка (не check, как на прошлом уроке)

Предоставляет нам метод onItemSelected полностью аналогичен по параметрам методу **onItemClick** описанному выше. Не буду повторяться.

Также есть метод onNothingSelected – когда список **теряет выделение** пункта и ни один пункт не выделен.

Все сохраним и запустим приложение.

Ткнем какой-нибудь элемент, например - Петр. Смотрим лог:

itemClick: position = 2, id = 2

Все верно. Т.к. позиция считается не с единицы, а с **нуля** – Петр имеет позицию 2. (В нашем случае id равен position. Я пока не встречал случаев id != position, но наверняка они есть)

Теперь покрутите колесо мышки или понажимайте клавиши вверх вниз на клавиатуре. Видно что идет **визуальное выделение** элементов списка.

А в логах мы видим такие записи:

itemSelect: position = 2, id = 2
itemSelect: position = 3, id = 3
itemSelect: position = 4, id = 4
itemSelect: position = 5, id = 5
itemSelect: position = 4, id = 4
itemSelect: position = 3, id = 3
itemSelect: position = 2, id = 2

Т.е. обработчик фиксирует какой пункт **выделен**. Честно говоря, я не очень понимаю как можно использовать такое выделение. Но обработчик для него есть и я решил про него рассказать. Пусть будет.

Снова нажмем теперь на любой пункт списка, мы видим, что выделение пропало. Логи:

itemSelect: nothing
itemClick: position = 3, id = 3

Ничего не выделено и нажат пункт с позицией 3.

Давайте добавим к списку еще один обработчик:

```
    lvMain.setOnScrollListener(new OnScrollListener() {
      public void onScrollStateChanged(AbsListView view, int scrollState) {
        // Log.d(LOG_TAG, "scrollState = " + scrollState);
      }

      public void onScroll(AbsListView view, int firstVisibleItem,
          int visibleItemCount, int totalItemCount) {
        Log.d(LOG_TAG, "scroll: firstVisibleItem = " + firstVisibleItem
            + ", visibleItemCount" + visibleItemCount
            + ", totalItemCount" + totalItemCount);
      }
    });
```

OnScrollListener – обрабатывает прокрутку списка.

Методы:

1) onScrollStateChanged(AbsListView view, int scrollState) - обработка состояний прокрутки

view – это прокручиваемый элемент, т.е. ListView
scrollState – состояние списка. Может принимать три значения:

SCROLL_STATE_IDLE = 0, список закончил прокрутку
SCROLL_STATE_TOUCH_SCROLL = 1, список начал прокрутку
SCROLL_STATE_FLING = 2, список «катнули», т.е. при прокрутке отпустили палец и прокрутка дальше идет «по инерции»

Вывод в лог я пока закаментил, чтобы не мешалось. Чуть позже раскаментим.

2) onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount, int totalItemCount) - обработка прокрутки

view – прокручиваемый элемент
firstVisibleItem – первый видимый на экране пункт списка
visibleItemCount – сколько пунктов видно на экране
totalItemCount – сколько всего пунктов в списке

Причем для параметров **firstVisibleItem** и **visibleItemCount** пункт считается видимым на экране даже если он виден не полностью.

Все сохраним и запустим.

Теперь потаскайте список туда-сюда курсором (как будто пальцем) и смотрите логи. Там слишком много всего выводится. Я не буду здесь выкладывать. Но принцип понятен – меняется первый видимый пункт (firstVisibleItem) и может на единицу меняться кол-во видимых пунктов (visibleItemCount).

Теперь закоментируем вывод в лог в методе **onScroll** (чтобы не спамил нам лог) и раскаментим в **onScrollStateChanged**.

```
    lvMain.setOnScrollListener(new OnScrollListener() {
      public void onScrollStateChanged(AbsListView view, int scrollState) {
        Log.d(LOG_TAG, "scrollState = " + scrollState);
      }

      public void onScroll(AbsListView view, int firstVisibleItem,
          int visibleItemCount, int totalItemCount) {
        //Log.d(LOG_TAG, "scroll: firstVisibleItem = " + firstVisibleItem
        //    + ", visibleItemCount" + visibleItemCount
        //    + ", totalItemCount" + totalItemCount);
      }
    });
```

Сохраняем, запускаем.

Схватим список, немного потягаем туда сюда и отпустим. Смотрим лог:

scrollState = 1
scrollState = 0

Отработали два события – список **начал** прокрутку, список **закончил** прокрутку.

Попробуем взять список, «катнуть» его и отпустить.

scrollState = 1
scrollState = 2
scrollState = 0

Видим три события – прокрутка **началась**, список «**катнули**», прокрутка **закончилась**.

Полный код урока:

```
package ru.startandroid.develop.p0441simplelistevents;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AbsListView;
import android.widget.AbsListView.OnScrollListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  ListView lvMain;
  TextView tvText;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    lvMain = (ListView) findViewById(R.id.lvMain);
    tvText = (TextView) findViewById(R.id.tvText);

    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
        this, R.array.names, android.R.layout.simple_list_item_1);
    lvMain.setAdapter(adapter);

    lvMain.setOnItemClickListener(new OnItemClickListener() {
      public void onItemClick(AdapterView<?> parent, View view,
          int position, long id) {
        Log.d(LOG_TAG, "itemClick: position = " + position + ", id = "
            + id);
      }
    });

    lvMain.setOnItemSelectedListener(new OnItemSelectedListener() {
      public void onItemSelected(AdapterView<?> parent, View view,
          int position, long id) {
        Log.d(LOG_TAG, "itemSelect: position = " + position + ", id = "
            + id);
      }

      public void onNothingSelected(AdapterView<?> parent) {
        Log.d(LOG_TAG, "itemSelect: nothing");
      }
    });

    lvMain.setOnScrollListener(new OnScrollListener() {
      public void onScrollStateChanged(AbsListView view, int scrollState) {
        Log.d(LOG_TAG, "scrollState = " + scrollState);
      }

      public void onScroll(AbsListView view, int firstVisibleItem,
```

```
            int visibleItemCount, int totalItemCount) {
          Log.d(LOG_TAG, "scroll: firstVisibleItem = " + firstVisibleItem
              + ", visibleItemCount" + visibleItemCount
              + ", totalItemCount" + totalItemCount);
        }
      });

    }
}
```

На следующем уроке:

- строим список-дерево ExpandableListView

Урок 45. Список-дерево ExpandableListView

В этом уроке:

- строим список-дерево ExpandableListView

Если список элементов получается большой, имеет смысл разбить его на **группы** для упрощения навигации. Для этих целей можно использовать ExpandableListView. Это список в виде двухуровневого дерева. Первый уровень – **группа**, а в ней второй – **элемент**.

Чтобы построить такой список нам нужно как-то передать адаптеру данные по группам и элементам.

Каждая группа представляет из себя **Map<String, ?>**. Этот Map содержит **атрибуты**, которые вам нужны для каждой группы. Потом все эти Map (группы) собираются в List-**коллекцию**, например ArrayList. В итоге мы получили упакованные в один объект группы.

Каждый элемент группы также представлен объектом **Map<String, ?>**. Мы собираем все Map (элементы) для каждой группы в отдельную **коллекцию**. Получается, каждой группе соответствует коллекция с элементами. Далее эти коллекции мы теперь помещаем в общую коллекцию. Т.е. получается подобие двумерного массива. И в итоге пункты упакованы в один объект.

Сейчас начнем кодить пример и там станет понятнее.

List-коллекции называются обычно «список». Но т.к. список в контексте последних уроков - это набор пунктов на экране (ListView), то чтобы не путаться я буду использовать слово «коллекция».

Создадим проект:

Project name: P0451_ExpandableList
Build Target: Android 2.3.3
Application name: ExpandableList
Package name: ru.startandroid.develop.p0451expandablelist
Create Activity: MainActivity

Нарисуем экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <ExpandableListView
          android:id="@+id/elvMain"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
     </ExpandableListView>
</LinearLayout>
```

Только **ExpandableList** на экране.

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0451expandablelist;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ExpandableListView;
import android.widget.SimpleExpandableListAdapter;

public class MainActivity extends Activity {

  // названия компаний (групп)
  String[] groups = new String[] {"HTC", "Samsung", "LG"};

  // названия телефонов (элементов)
  String[] phonesHTC = new String[] {"Sensation", "Desire", "Wildfire", "Hero"};
  String[] phonesSams = new String[] {"Galaxy S II", "Galaxy Nexus", "Wave"};
  String[] phonesLG = new String[] {"Optimus", "Optimus Link", "Optimus Black", "Optimus One"};

  // коллекция для групп
  ArrayList<Map<String, String>> groupData;

  // коллекция для элементов одной группы
  ArrayList<Map<String, String>> childDataItem;

  // общая коллекция для коллекций элементов
  ArrayList<ArrayList<Map<String, String>>> childData;
  // в итоге получится childData = ArrayList<childDataItem>

  // список аттрибутов группы или элемента
  Map<String, String> m;

  ExpandableListView elvMain;


    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // заполняем коллекцию групп из массива с названиями групп
        groupData = new ArrayList<Map<String, String>>();
        for (String group : groups) {
          // заполняем список аттрибутов для каждой группы
          m = new HashMap<String, String>();
            m.put("groupName", group); // имя компании
            groupData.add(m);
        }

        // список аттрибутов групп для чтения
        String groupFrom[] = new String[] {"groupName"};
        // список ID view-элементов, в которые будет помещены аттрибуты групп
        int groupTo[] = new int[] {android.R.id.text1};


        // создаем коллекцию для коллекций элементов
        childData = new ArrayList<ArrayList<Map<String, String>>>();

        // создаем коллекцию элементов для первой группы
```

```
        childDataItem = new ArrayList<Map<String, String>>();
        // заполняем список аттрибутов для каждого элемента
        for (String phone : phonesHTC) {
          m = new HashMap<String, String>();
            m.put("phoneName", phone); // название телефона
            childDataItem.add(m);
        }
        // добавляем в коллекцию коллекций
        childData.add(childDataItem);

        // создаем коллекцию элементов для второй группы
        childDataItem = new ArrayList<Map<String, String>>();
        for (String phone : phonesSams) {
          m = new HashMap<String, String>();
            m.put("phoneName", phone);
            childDataItem.add(m);
        }
        childData.add(childDataItem);

        // создаем коллекцию элементов для третьей группы
        childDataItem = new ArrayList<Map<String, String>>();
        for (String phone : phonesLG) {
          m = new HashMap<String, String>();
            m.put("phoneName", phone);
            childDataItem.add(m);
        }
        childData.add(childDataItem);

        // список аттрибутов элементов для чтения
        String childFrom[] = new String[] {"phoneName"};
        // список ID view-элементов, в которые будет помещены аттрибуты элементов
        int childTo[] = new int[] {android.R.id.text1};

        SimpleExpandableListAdapter adapter = new SimpleExpandableListAdapter(
            this,
            groupData,
            android.R.layout.simple_expandable_list_item_1,
            groupFrom,
            groupTo,
            childData,
            android.R.layout.simple_list_item_1,
            childFrom,
            childTo);

        elvMain = (ExpandableListView) findViewById(R.id.elvMain);
        elvMain.setAdapter(adapter);
    }
}
```

Код громоздкий и сложноватый, давайте разбираться.

Сначала мы в классе описываем массивы **данных** – это названия **групп** и названия **элементов** для них. Я решил в качестве данных выбрать смартфоны. **Группы** в нашем списке – это будут **компании**, а **элементы** – **смартфоны** этих компаний.

Затем описываем коллекцию для групп, коллекции для элементов и Map для атрибутов.

В методе onCreate заполняем **groupData**. Это коллекция групп. Каждая группа представляет собой **Map**. А в Map мы пишем необходимые нам **атрибуты** для каждой группы. В нашем случае, для каждой группы мы укажем всего один атрибут **groupName** - это **название** компании из массива **groups**.

Как мы помним, **адаптер** обычно использует **layout**-ресурс для отображения пункта списка. В нашем случае **пунктами** ListView являются и **группа** и **элемент**. В layout-ресурсе могут быть какие-либо **TextView**. Мы можем заполнить их значениями из **атрибутов** элементов или групп, которые собраны в Map. Для этого нам надо указать сначала **имена атрибутов**, которые хотим использовать, а затем **ID TextView**-элементов, в которые хотим поместить значения этих атрибутов. Речь сейчас идет о текстовых атрибутах. (Хотя вообще атрибут вовсе не обязан быть класса String)

Для связки **атрибутов** и **TextView**-элементов мы используем два массива:

groupFrom – список имен атрибутов, которые будут считаны. В нашем случае – это **groupName**, который мы добавили к группе с помощью Map чуть выше в коде, когда собирали группы в groupData.
groupTo – список ID View-элементов, в которые будут помещены считанные значения атрибутов. Наш используемый layout будет содержать TextView с ID = **android.R.id.text1**.

Два этих массива сопоставляются по порядку элементов. В итоге, в **layout**-ресурсе группы найдется элемент с ID = **android.R.id.text1** и в него запишется текст из атрибута **groupName**. Тем самым мы получим отображение имени группы (компании) в списке.

Далее формируем **коллекции элементов**. Создаем общую **коллекцию коллекций**. А затем создаем коллекции элементов каждой группы. Принцип тот же, что и с группами – создаем **Map** и в него пишем атрибут **phoneName** со значением равным **имени** элемента (телефона). Коллекцию элементов для каждой группы добавляем в общую коллекцию.

Формируем два массива для сопоставления **TextView** из layout и **атрибутов** элементов. Полностью аналогично, как выше мы уже проделали с группами. В итоге при отображении элемента, найдется TextView с ID = android.R.id.text1 и туда запишется текст из атрибута phoneName. И мы увидим текст нашего элемента (телефона) в списке.

В конце кода мы создаем адаптер SimpleExpandableListAdapter и присваиваем его списку.

На вход при создании адаптера идут элементы:

this – контекст
groupData – коллекция групп
android.R.layout.simple_expandable_list_item_1 – layout-ресурс, который будет использован для отображения группы в списке. Соответственно, запросто можно использовать свой layout-файл.
groupFrom – массив имен атрибутов групп
groupTo – массив ID TextView из layout для групп
childData – коллекция коллекций элементов по группам
android.R.layout.simple_list_item_1 - layout-ресурс, который будет использован для отображения элемента в списке. Можно использовать свой layout-файл
childFrom – массив имен атрибутов элементов
childTo - массив ID TextView из layout для элементов.

В общем непростая, на мой взгляд, реализация дерева получилась. Возможно, не сразу получится понять. Но я попытался расписать все досконально и как можно подробнее.

Layout **simple_expandable_list_item_1**, который мы использовали для отображения групп – это TextView с отступом от левого края, чтобы осталось место для кнопки раскрытия/сворачивания списка. Для эксперимента вы можете попробовать использовать для групп layout **simple_list_item_1**, который мы использовали для элементов. В этом случае текст будет пересекаться с кнопкой.

А вообще вы можете создать для элементов свой **layout**, например, с тремя **TextView**. И к каждому элементу списка (Map) добавить еще по два атрибута: цена и цвет. Далее указываете ваш layout в конструкторе, формируете соответственно массивы **childFrom** и **childTo** чтобы сопоставить атрибуты и TextView, и получится, что каждый элемент группы содержит более подробную информацию о смартфоне.

На следующем уроке:

- обрабатываем события дерева-списка

Урок 46. События ExpandableListView

В этом уроке:

- обрабатываем события дерева-списка

Дерево-список строить мы умеем, теперь посмотрим, как с ним можно взаимодействовать. Нам предоставлена возможность обрабатывать следующие события: нажатие на группу, нажатие на элемент, сворачивание группы, разворачивание группы.

Создадим проект:

Project name: P0461_ExpandableListEvents
Build Target: Android 2.3.3
Application name: ExpandableListEvents
Package name: ru.startandroid.develop.p0461expandablelistevents
Create Activity: MainActivity

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <LinearLayout
         android:id="@+id/linearLayout1"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:orientation="vertical">
        <TextView
             android:id="@+id/tvInfo"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content">
        </TextView>
        <ExpandableListView
             android:id="@+id/elvMain"
             android:layout_width="match_parent"
             android:layout_height="wrap_content">
        </ExpandableListView>
    </LinearLayout>
</LinearLayout>
```

TextView для вывода информации и **ExpandableListView**.

В проекте, рядом с классом **MainActivity** создадим (не Activity) класс **AdapterHelper**. В него поместим код для заполнения списка, чтобы разгрузить MainActivity.java.

Код **AdapterHelper.java**:

```
package ru.startandroid.develop.p0461expandablelistevents;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import android.content.Context;
import android.widget.SimpleExpandableListAdapter;

public class AdapterHelper {

  final String ATTR_GROUP_NAME= "groupName";
  final String ATTR_PHONE_NAME= "phoneName";


  // названия компаний (групп)
  String[] groups = new String[] {"HTC", "Samsung", "LG"};

  // названия телефонов (элементов)
  String[] phonesHTC = new String[] {"Sensation", "Desire", "Wildfire", "Hero"};
  String[] phonesSams = new String[] {"Galaxy S II", "Galaxy Nexus", "Wave"};
  String[] phonesLG = new String[] {"Optimus", "Optimus Link", "Optimus Black", "Optimus One"};

  // коллекция для групп
  ArrayList<Map<String, String>> groupData;

  // коллекция для элементов одной группы
  ArrayList<Map<String, String>> childDataItem;

  // общая коллекция для коллекций элементов
  ArrayList<ArrayList<Map<String, String>>> childData;
  // в итоге получится childData = ArrayList<childDataItem>

  // список аттрибутов группы или элемента
  Map<String, String> m;

  Context ctx;

  AdapterHelper(Context _ctx) {
    ctx = _ctx;
  }

  SimpleExpandableListAdapter adapter;


  SimpleExpandableListAdapter getAdapter() {

    // заполняем коллекцию групп из массива с названиями групп
        groupData = new ArrayList<Map<String, String>>();
        for (String group : groups) {
          // заполняем список аттрибутов для каждой группы
          m = new HashMap<String, String>();
            m.put(ATTR_GROUP_NAME, group); // имя компании
            groupData.add(m);
        }

        // список аттрибутов групп для чтения
        String groupFrom[] = new String[] {ATTR_GROUP_NAME};
        // список ID view-элементов, в которые будет помещены аттрибуты групп
        int groupTo[] = new int[] {android.R.id.text1};
```

```
      // создаем коллекцию для коллекций элементов
      childData = new ArrayList<ArrayList<Map<String, String>>>();

      // создаем коллекцию элементов для первой группы
      childDataItem = new ArrayList<Map<String, String>>();
      // заполняем список аттрибутов для каждого элемента
      for (String phone : phonesHTC) {
        m = new HashMap<String, String>();
          m.put(ATTR_PHONE_NAME, phone); // название телефона
          childDataItem.add(m);
      }
      // добавляем в коллекцию коллекций
      childData.add(childDataItem);

      // создаем коллекцию элементов для второй группы
      childDataItem = new ArrayList<Map<String, String>>();
      for (String phone : phonesSams) {
        m = new HashMap<String, String>();
          m.put(ATTR_PHONE_NAME, phone);
          childDataItem.add(m);
      }
      childData.add(childDataItem);

      // создаем коллекцию элементов для третьей группы
      childDataItem = new ArrayList<Map<String, String>>();
      for (String phone : phonesLG) {
        m = new HashMap<String, String>();
          m.put(ATTR_PHONE_NAME, phone);
          childDataItem.add(m);
      }
      childData.add(childDataItem);

      // список аттрибутов элементов для чтения
      String childFrom[] = new String[] {ATTR_PHONE_NAME};
      // список ID view-элементов, в которые будет помещены аттрибуты элементов
      int childTo[] = new int[] {android.R.id.text1};

      adapter = new SimpleExpandableListAdapter(
          ctx,
          groupData,
          android.R.layout.simple_expandable_list_item_1,
          groupFrom,
          groupTo,
          childData,
          android.R.layout.simple_list_item_1,
          childFrom,
          childTo);

    return adapter;
  }

  String getGroupText(int groupPos) {
    return ((Map<String,String>)(adapter.getGroup(groupPos))).get(ATTR_GROUP_NAME);
  }

  String getChildText(int groupPos, int childPos) {
    return ((Map<String,String>)(adapter.getChild(groupPos, childPos))).get(ATTR_PHONE_NAME);
  }

  String getGroupChildText(int groupPos, int childPos) {
    return getGroupText(groupPos) + " " +  getChildText(groupPos, childPos);
```

```
    }
}
```

Код создания адаптера полностью заимствован с прошлого урока. Чтобы получить адаптер нам надо будет просто вызвать метод **getAdapter**.

У класса есть **конструктор**, через который мы передаем объекту ссылку на **context**. Context нам понадобится, чтобы создать адаптер. Адаптеру же в свою очередь context нужен, например, для доступа к LayoutInflater.

В конце класса находятся методы, которые возвращают нам названия групп и элементов из коллекций по номеру группы или номеру элемента. Для этого используем методы адаптера **getGroup** и **getChild**, приводим их к **Map** и извлекаем значение **атрибута** с именем компании или телефона.

Пишем код в **MainActivity.java**:

```
package ru.startandroid.develop.p0461expandablelistevents;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ExpandableListView;
import android.widget.ExpandableListView.OnChildClickListener;
import android.widget.ExpandableListView.OnGroupClickListener;
import android.widget.ExpandableListView.OnGroupCollapseListener;
import android.widget.ExpandableListView.OnGroupExpandListener;
import android.widget.SimpleExpandableListAdapter;
import android.widget.TextView;


public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  ExpandableListView elvMain;
  AdapterHelper ah;
  SimpleExpandableListAdapter adapter;
  TextView tvInfo;

    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        tvInfo = (TextView) findViewById(R.id.tvInfo);

        // создаем адаптер
        ah = new AdapterHelper(this);
        adapter = ah.getAdapter();

        elvMain = (ExpandableListView) findViewById(R.id.elvMain);
        elvMain.setAdapter(adapter);

        // нажатие на элемент
        elvMain.setOnChildClickListener(new OnChildClickListener() {
      public boolean onChildClick(ExpandableListView parent, View v,
          int groupPosition,   int childPosition, long id) {
```

```
        Log.d(LOG_TAG, "onChildClick groupPosition = " + groupPosition +
                " childPosition = " + childPosition +
                " id = " + id);
        tvInfo.setText(ah.getGroupChildText(groupPosition, childPosition));
        return false;
      }
    });

        // нажатие на группу
        elvMain.setOnGroupClickListener(new OnGroupClickListener() {
      public boolean onGroupClick(ExpandableListView parent, View v,
          int groupPosition, long id) {
        Log.d(LOG_TAG, "onGroupClick groupPosition = " + groupPosition +
                " id = " + id);
        // блокируем дальнейшую обработку события для группы с позицией 1
        if (groupPosition == 1) return true;

        return false;
      }
    });

        // сворачивание группы
        elvMain.setOnGroupCollapseListener(new OnGroupCollapseListener() {
      public void onGroupCollapse(int groupPosition) {
        Log.d(LOG_TAG, "onGroupCollapse groupPosition = " + groupPosition);
        tvInfo.setText("Свернули " + ah.getGroupText(groupPosition));
      }
    });

        // разворачивание группы
        elvMain.setOnGroupExpandListener(new OnGroupExpandListener() {
      public void onGroupExpand(int groupPosition) {
        Log.d(LOG_TAG, "onGroupExpand groupPosition = " + groupPosition);
        tvInfo.setText("Развернули " + ah.getGroupText(groupPosition));
      }
    });

        // разворачиваем группу с позицией 2
        elvMain.expandGroup(2);
    }
}
```

Благодаря классу **AdapterHelper**, код создания адаптера занял всего две строчки: **создание** объекта и вызов метода **getAdapter**. Далее присваиваем адаптер списку и добавляем **обработчики**:

1) OnChildClickListener - нажатие на элемент

Метод
public boolean **onChildClick**(ExpandableListView parent, View v, int groupPosition, int childPosition, long id), где

parent – ExpandableListView с которым работаем
v – View элемента
groupPosition – позиция группы в списке
childPosition – позиция элемента в группе
id – id элемента

Мы выводим в лог позицию и id. А в **TextView** сверху от списка выводим **текст** нажатого **элемента** и его **группы**, который получаем с помощью методов AdapterHelper.

Метод должен вернуть **boolean**. Если мы возвращаем **true** – это значит, мы сообщаем, что сами полностью **обработали**

событие и оно **не пойдет в дальнейшие обработчики** (если они есть). Если возвращаем **false** – значит, мы позволяем **событию идти дальше**.

2) OnGroupClickListener – нажатие на группу

Метод
public boolean **onGroupClick**(ExpandableListView parent, View v, int groupPosition, long id), где

parent – ExpandableListView с которым работаем
v – View элемента
groupPosition – позиция группы в списке
id – id группы

Мы выводим в лог позицию и id группы.

Этот метод также должен вернуть **boolean**. Мы будет возвращать true, если позиция группы = 1, иначе - false. Т.е. для этой группы мы блокируем дальнейшую обработку события. Далее увидим, что нам это даст.

3) OnGroupCollapseListener – сворачивание группы

Метод
onGroupCollapse(int groupPosition), где groupPosition – позиция группы, которую свернули

4) OnGroupExpandListener – разворачивание группы

Метод
onGroupExpand(int groupPosition), где groupPosition – позиция группы, которую развернули

И в конце кода MainActivity мы разворачиваем группу с позицией 2, используя метод expandGroup.

Все сохраним и запускаем.

Как видим, группа LG сразу развернута. Это сработала команда **expandGroup** в конце кода.

Если посмотреть в лог, то видим

onGroupExpand groupPosition = 2

Т.е. отработало событие разворачивания группы с позицией 2.

Нажмем, например, на Optimus Link. Смотрим лог:

onChildClick groupPosition = 2 childPosition = 1 id = 1

Не забываем, что позиция считается с нуля. Группа с позицией 2 – LG, элемент с позицией 1 – Optimus Link, все верно.

Смотрим TextView сверху экрана, он считал из адаптера значение атрибута и отобразил его.

Теперь попробуем свернуть группу LG, нажмем на нее. Смотрим лог:

onGroupClick groupPosition = 2 id = 2
onGroupCollapse groupPosition = 2

Сначала отработал **onGroupClick** – нажатие на группу, а потом **onGroupCollapse** – сворачивание группы. TextView наверху экрана оповестил о том, что свернули группу LG.

Снова развернем группу LG. Лог:

onGroupClick groupPosition = 2 id = 2
onGroupExpand groupPosition = 2

Нажатие на группу и разворачивание. Обратите внимание, что при программном разворачивании, события нажатия не было, только разворот.

Теперь попробуем развернуть группу с позицией 1 – Samsung. Группа не разворачивается. Смотрим лог:

onGroupClick groupPosition = 1 id = 1

Событие нажатия есть, а вот обработчик разворачивания не вызывается. Это происходит из-за строчки

```
        // блокируем дальнейшую обработку события для группы с позицией 1
        if (groupPosition == 1) return true;
```

Мы для группы с позицией 1 блокируем дальнейшую обработку события и оно не уходит в обработчики разворачивания или сворачивания. Поэтому и не срабатывает **onGroupExpand**.

В итоге эти 4 обработчика позволяют вам определять, как пользователь взаимодействует с деревом.

Урок 47. Обзор адаптеров

В этом уроке:

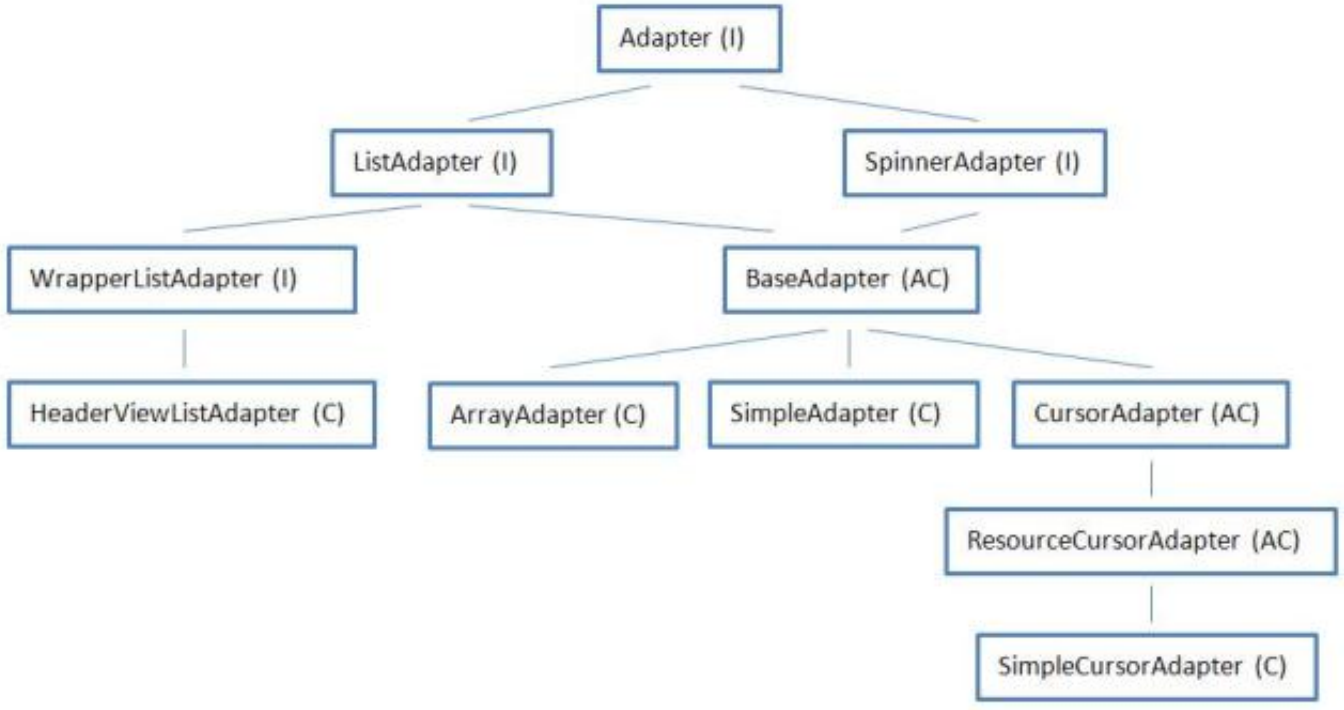
- разбираемся в адаптерах

На последних уроках мы использовали **адаптеры** и сейчас вполне можем сформулировать, что это такое. Адаптер – **мост** между набором **данных** и **объектом**, использующим эти данные. Также адаптер отвечает за создание **View**-компонента для каждой единицы данных из набора.

Я по себе помню, что новичку непросто разобраться в адаптерах. Кажется, что их куча и абсолютно непонятно какой и где используется. И рассказывать об этом без практики – бессмысленно. Поэтому в своих уроках я сначала рассмотрел пару примеров, чтобы в голове сложилась схема использования: набор данных – адаптер – объект. И теперь проще будет разобрать, какие еще адаптеры есть и чем они отличаются.

В рассмотренных нами примерах были:
- адаптер **ArrayAdapter** и объект для отображения данных **ListView**
- адаптер **SimpleExpandableListAdapter** и объект **ExpandableListView**.

Я попробую в этом материале сделать обзор адаптеров и показать, чем они отличаются друг от друга. Материал будет периодически обновляться и пополняться. А пока я накидал схему java-иерархии интерфейсов и классов адаптеров. В скобках указываю тип: I – это интерфейс, AC – абстрактный класс, C – класс. Линии – это наследование. Читать следует сверху вниз.



Давайте по порядку смотреть, зачем оно все нужно.

Интерфейс Adapter. Описывает базовые методы, которые должны содержать адаптеры: getCount, getItem, getView и пр.

Интерфейс ListAdapter. Этот интерфейс должен быть реализован адаптером, который будет использован в ListView (метод setAdapter). Содержит описание методов для работы с разделителями(separator) списка.

Интерфейс SpinnerAdapter. Адаптеры, реализующие этот интерфейс, используются для построения Spinner(выпадающий список или drop-down). Содержит метод getDropDownView, который возвращает элемент выпадающего списка. На офиц.сайте есть пример использования.

Интерфейс WrapperListAdapter. Адаптеры, наследующие этот интерфейс используются для работы с вложенными адаптерами. Содержит метод getWrappedAdapter, который позволяет вытащить из основного адаптера вложенный. Чуть дальше поясню.

Класс HeaderViewListAdapter. Готовый адаптер для работы с Header и Footer. Внутри себя содержит еще один адаптер (ListAdapter), который можно достать с помощью выше рассмотренного метода getWrappedAdapter из интерфейса WrapperListAdapter.

Абстрактный класс BaseAdapter. Содержит немного своих методов и реализует методы интерфейсов, которые наследует, но не все. Своим наследникам оставляет на обязательную реализацию методы: getView, getItemId, getItem, getCount из ListAdapter. Т.е. если хотите создать свой адаптер – это класс вам подходит.

Класс ArrayAdapter<T>. Готовый адаптер, который мы уже использовали. Принимает на вход список или массив объектов, перебирает его и вставляет строковое значение в указанный TextView. Кроме наследуемых методов содержит методы по работе с коллекцией данных – add, insert, remove, sort, clear и метод setDropDownViewResource для задания layout-ресурса для отображения пунктов выпадающего списка.

Класс SimpleAdapter. Также готовый к использованию адаптер. Принимает на вход список Map-объектов, где каждый Map-объект – это список атрибутов. Кроме этого на вход принимает два массива – from[] и to[]. В to указываем id экранных элементов, а в from имена(key) из объектов Map, значения которых будут вставлены в соответствующие (из from) экранные элементы.

Т.е. SimpleAdapter – это расширенный ArrayAdapter. Если вы делаете ListView и у вас каждый пункт списка содержит не один TextView, а несколько, то вы используете SimpleAdapter. Кроме наследуемых методов SimpleAdapter содержит методы по наполнению View-элементов значениями из Map – setViewImage, setViewText, setViewBinder. Т.е. видим, что он умеет работать не только с текстом, но и с изображениями. Метод setViewBinder – отличная штука, позволяет вам написать свой парсер значений из Map в View-элементы и адаптер будет использовать его. Это мы еще детально обсудим в дальнейших уроках.

Также содержит реализацию метода setDropDownViewResource.

Абстрактный класс CursorAdapter. Реализует абстрактные методы класса BaseAdapter, содержит свои методы по работе с курсором и оставляет наследникам методы по созданию и наполнению View: newView, bindView.

Абстрактный класс ResourceCursorAdapter. Содержит методы по настройке используемых адаптером layout-ресурсов. Реализует метод newView из CursorAdapter.

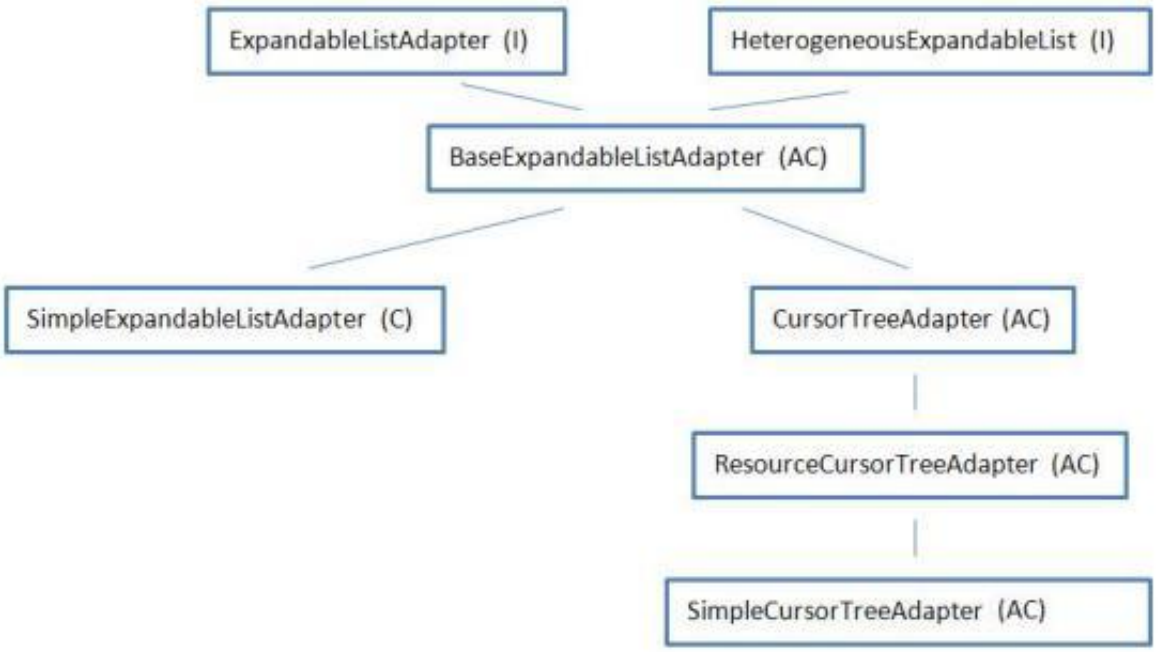
Класс SimpleCursorAdapter. Готовый адаптер, похож, на SimpleAdapter. Только использует не набор объектов Map, а Cursor, т.е. набор строк с полями. Соответственно в массив from[] вы заносите наименования полей, значения которых хотите вытащить в соответствующие View из массива to.

Содержит метод convertToString, который возвращает строковое значение столбца, который задается методом setStringConversionColumn. Либо можно задать свой конвертер методом setCursorToStringConverter и адаптер будет использовать его при вызове convertToString. В этом конвертере вы уже сами реализуете, что он будет возвращать.

Итого мы получили 4 готовых адаптера: HeaderViewListAdapter, ArrayAdapter<T>, SimpleAdapter, SimpleCursorAdapter. Какой из них использовать - решать вам. Если у вас есть массив строк - то, не раздумывая, берете ArrayAdapter. Если работаете с БД и есть курсор, данные из которого надо вывести в список - используете SimpleCursorAdapter.

Если же эти адаптеры вам не подходят, есть набор абстрактных классов и интерфейсов, которые можем наследовать и реализовать в своих классах для создания своего адаптера. Да и готовые адаптеры всегда можно наследовать и сделать свою реализацию методов.

Кроме этой иерархии есть почти аналогичная ей. В ней содержатся адаптеры для работы с деревом – **ExpandableListView**. Я не буду здесь их расписывать, в целом они похожи на, уже рассмотренные, нами объекты. Но есть разница в том, что здесь данные не одноуровневые, а делятся на группы и элементы.



Здесь мы видим один готовый адаптер. SimpleExpandableListAdapter - работает с коллекциями Map.

Если вам интересны примеры работы того или иного адаптера – пишите на форуме в ветке этого урока (ссылка внизу страницы). Ближайшие несколько уроков будут посвящены этой теме.

Урок 48. Используем SimpleAdapter.

В этом уроке:

- рассмотрим пример использования SimpleAdapter

SimpleAdapter работает с данными такого вида List<? extends Map<String, ?>> - т.е. коллекция Map-объектов (или - наследников). Если смотреть с точки зрения списка, то каждый **Map** содержит данные для соответствующего **пункта** списка. Чтобы адаптер знал, какие **данные** в какие **View**-компоненты каждого пункта списка ему вставлять, мы указываем два массива: String[] **from** и int[] **to**. В from перечисляем ключи из Map, а в to – ID View-компонентов. Адаптер перебирает **View** из массива **to** и сопоставляет им **значения** из **Map** по ключам из **from**.

Проверим это на примере. Создадим небольшой список, каждый пункт которого будет состоять из checkbox-а, текстового поля и картинки.

Создадим проект:

Project name: P0481_SimpleAdapter
Build Target: Android 2.3.3
Application name: SimpleAdapter
Package name: ru.startandroid.develop.p0481simpleadapter
Create Activity: MainActivity

Рисуем экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <ListView
          android:id="@+id/lvSimple"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
     </ListView>
</LinearLayout>
```

Здесь только LIstView.

Создаем **item.xml** для пункта списка:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="horizontal">
     <ImageView
          android:id="@+id/ivImg"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher">
        </ImageView>
        <LinearLayout
            android:id="@+id/linearLayout1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:orientation="vertical">
            <CheckBox
                android:id="@+id/cbChecked"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="CheckBox">
            </CheckBox>
            <TextView
                android:id="@+id/tvText"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="right"
                android:layout_marginRight="20dp"
                android:text="TextView">
            </TextView>
        </LinearLayout>
</LinearLayout>
```

Тройка ImageView, CheckBox и TextView будут у нас представлять каждый пункт списка.

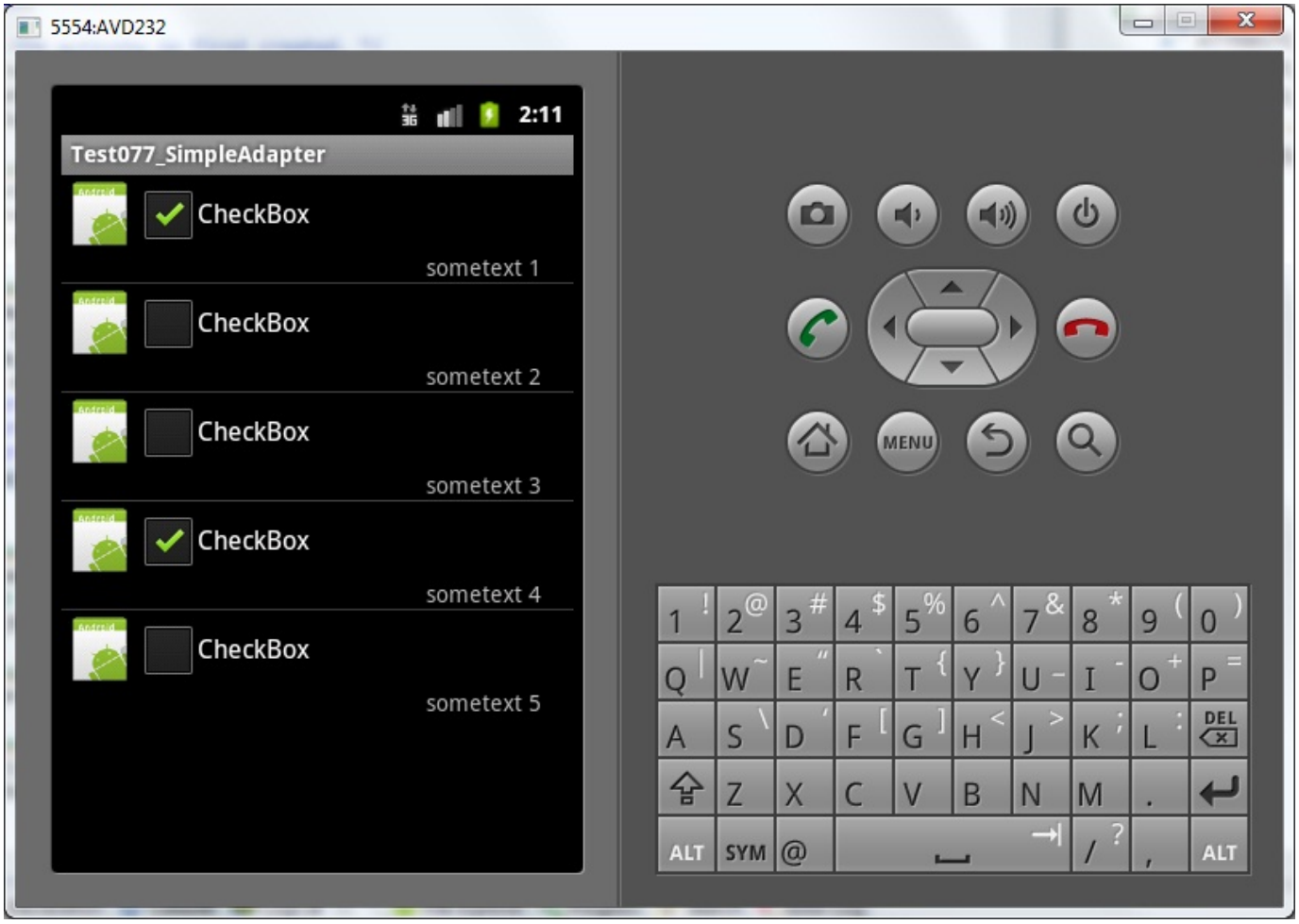
Пишем код **MainActivity.java**:

```
package ru.startandroid.develop.p0481simpleadapter;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;
import android.widget.SimpleAdapter;

public class MainActivity extends Activity {

  // имена атрибутов для Map
  final String ATTRIBUTE_NAME_TEXT = "text";
  final String ATTRIBUTE_NAME_CHECKED = "checked";
  final String ATTRIBUTE_NAME_IMAGE = "image";

  ListView lvSimple;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.main);

    // массивы данных
    String[] texts = { "sometext 1", "sometext 2", "sometext 3",
        "sometext 4", "sometext 5" };
    boolean[] checked = { true, false, false, true, false };
    int img = R.drawable.ic_launcher;

    // упаковываем данные в понятную для адаптера структуру
    ArrayList<Map<String, Object>> data = new ArrayList<Map<String, Object>>(
        texts.length);
    Map<String, Object> m;
    for (int i = 0; i < texts.length; i++) {
      m = new HashMap<String, Object>();
      m.put(ATTRIBUTE_NAME_TEXT, texts[i]);
      m.put(ATTRIBUTE_NAME_CHECKED, checked[i]);
      m.put(ATTRIBUTE_NAME_IMAGE, img);
      data.add(m);
    }

    // массив имен атрибутов, из которых будут читаться данные
    String[] from = { ATTRIBUTE_NAME_TEXT, ATTRIBUTE_NAME_CHECKED,
        ATTRIBUTE_NAME_IMAGE };
    // массив ID View-компонентов, в которые будут вставлять данные
    int[] to = { R.id.tvText, R.id.cbChecked, R.id.ivImg };

    // создаем адаптер
    SimpleAdapter sAdapter = new SimpleAdapter(this, data, R.layout.item,
        from, to);

    // определяем список и присваиваем ему адаптер
    lvSimple = (ListView) findViewById(R.id.lvSimple);
    lvSimple.setAdapter(sAdapter);
  }
}
```

Разбираем код. Мы создаем константы для имен атрибутов, описываем массивы данных и пакуем эти данные в ArrayList<Map<String,Object>> (img у нас будет одинаков для всех). Далее мы определяем два массива, которые будут использованы для сопоставления данных и View-компонентов. Массив **from** содержит **имена Map-ключей**, а массив **to** – **ID View**-компонентов. Т.е. в R.id.tvText будет вставлено значение из элемента Map с ключом ATTRIBUTE_NAME_TEXT и т.д. по порядку. После этого мы создаем адаптер и увязываем его со списком.

Все сохраним и запускаем. Видим, что все так, как мы и планировали.



Адаптер перебрал **View**-компоненты (массив **to**) для каждого пункта списка и сопоставил им значения из **Map** (массив **from**). Т.е. карта сопоставления примерно такая:

R.id.tvText – Map.get(ATTRIBUTE_NAME_TEXT)
R.id.cbChecked - Map.get(ATTRIBUTE_NAME_CHECKED)
R.id.ivImg - Map.get(ATTRIBUTE_NAME_IMAGE)

А как адаптер понимает, какие методы и для каких View-компонентов надо вызывать, чтобы передать им значения из Map? Он смотрит на **тип** View-компонента и в зависимости от него уже определяет как работать с ним. Для него существует три типа:

1) View, наследующие интерфейс Checkable. Примеры - CheckBox, CheckedTextView, CompoundButton, RadioButton, Switch, ToggleButton. В этом случае адаптер проверяет, является ли соответствующее значение из Map типа **boolean**. Если да, то все ок и вызывается метод Checkable.setChecked. Иначе получим ошибку.

С версии 2.2 алгоритм чуть поменялся – добавили еще одну ветку логики. Теперь, если значение из Map не **boolean**, то проверяется, что View является наследником **TextView**. Если да, то вставляется текстовое значение из Map.

2) View, являющиеся **TextView** или его наследниками. Тут просто вызывается метод SimpleAdapter.setViewText, а он уже вызывает TextView.setText и передает туда значение из Map.

3) View, являющиеся **ImageView** или его наследниками. Адаптер проверяет тип данных из Map:

- если их можно привести (instanceof) к типу Integer, то вызывается метод SimpleAdapter.setViewImage(ImageView v, int value), а он уже вызывает ImageView.setImageResource.

- иначе вызывается метод SimpleAdapter.setViewImage (ImageView v, String value), который снова пытается привести значение к int и вызвать метод ImageView.setImageResource, и, если не получается, то преобразует строку к **Uri** (Uri.parse) и вызвает метод ImageView.setImageURI(Uri).

Если View не подходит ни под один из трех вышеперечисленных типов, то получим ошибку.

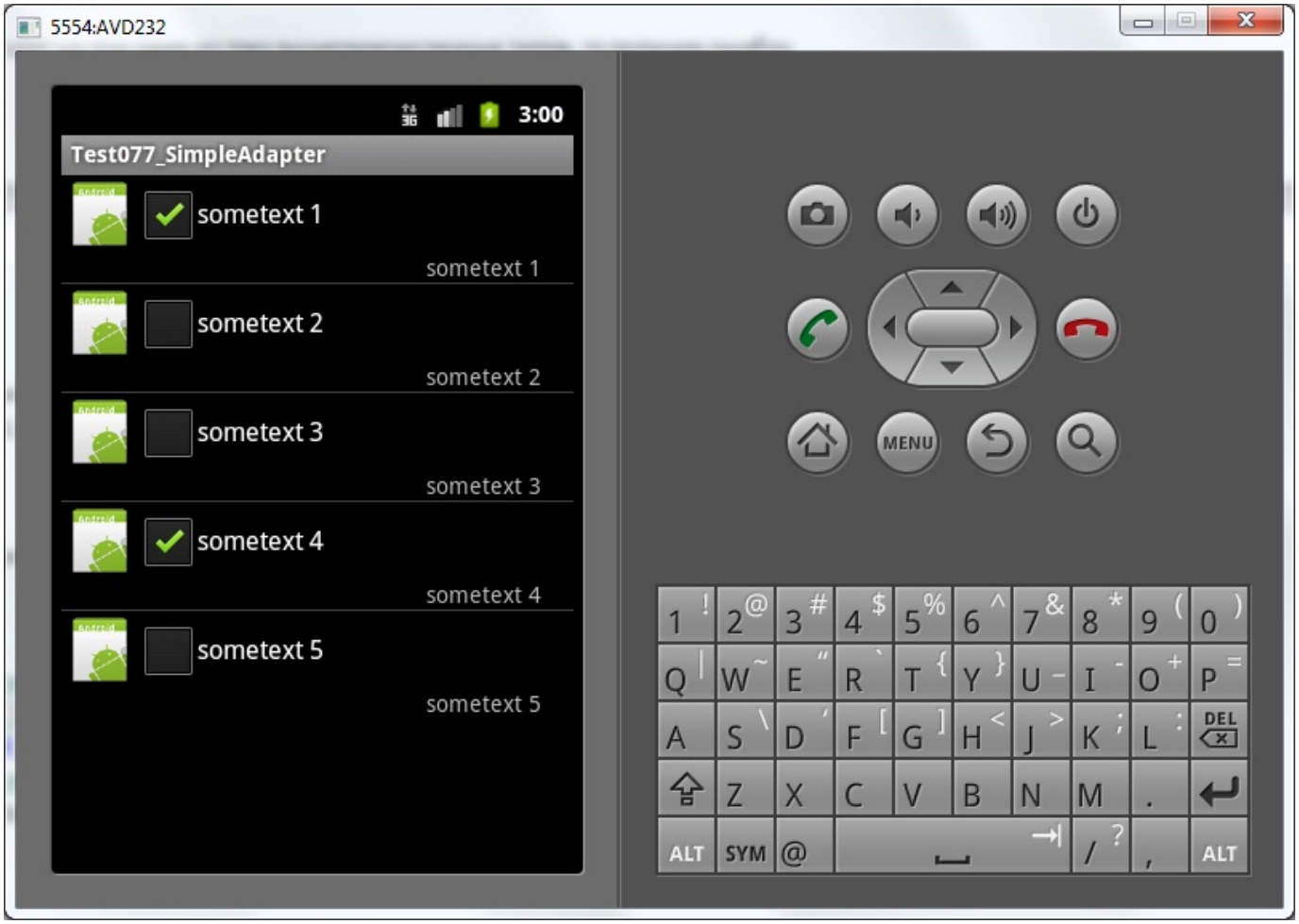
Немного поэкспериментируем, и попробуем в CheckBox вставлять тот же текст, что и в TextView. Адаптер сможет это сделать (если версия Android - 2.2 и выше), т.к. **CheckBox** является наследником **TextView**.

Для этого перепишем заполнение массивов:

```
    // массив имен атрибутов, из которых будут читаться данные
    String[] from = { ATTRIBUTE_NAME_TEXT, ATTRIBUTE_NAME_CHECKED,
        ATTRIBUTE_NAME_IMAGE, ATTRIBUTE_NAME_TEXT };

    // массив ID View-компонентов, в которые будут вставлять данные
    int[] to = { R.id.tvText, R.id.cbChecked, R.id.ivImg, R.id.cbChecked };
```

Мы добавили адаптеру указание, чтобы он в R.id.cbChecked (CheckBox), вставил текст из Map по ключу ATTRIBUTE_NAME_TEXT.

Сохраняем, запускаем.



Текст успешно вставился в CheckBox.

Теперь попробуем передать данные неверно. Например, вместо id изображения, передадим boolean в R.id.ivImg. Для этого изменим from:

```
    // массив имен атрибутов, из которых будут читаться данные
    String[] from = { ATTRIBUTE_NAME_TEXT, ATTRIBUTE_NAME_CHECKED,
        ATTRIBUTE_NAME_CHECKED, ATTRIBUTE_NAME_TEXT };
```

Сохраним, запустим.

Видим, что картинки нет. Адаптер определил, что передали не Integer и попытался показать картинку по Uri из текстовой интерпретации boolean, но у него не получилось.

В общем, штука неплохая, но функционал довольно ограничен - поставить галку, указать картинку и разместить текст. В следующий раз попробуем расширить функционал под свои потребности.

Думаю, имеет смысл сказать, что, если ставить и снимать галку в пунктах, то данные адаптера при этом не меняются.

На следующем уроке:

- используем методы SetViewText и SetViewImage

Урок 49. SimpleAdapter. Методы SetViewText и SetViewImage

В этом уроке:

- используем методы SetViewText и SetViewImage

Мы уже знаем, что **SimpleAdapter** умеет вставлять текст в TextView элементы и изображения в ImageView. Он использует для этого методы **SetViewText** и **SetViewImage**. Мы можем создать **свой адаптер** на основе SimpleAdapter и реализовать эти методы под наши цели.

Эти методы предоставляют нам **View** и **данные**, а значит мы можем менять View в зависимости от данных. В качестве примера, сделаем список, отражающий динамику некоего показателя в разрезе дней. Если динамика положительная – будем разукрашивать элементы в зеленый цвет, если отрицательная – в красный.

Создадим проект:

Project name: P0491_SimpleAdapterCustom1
Build Target: Android 2.3.3
Application name: SimpleAdapterCustom1
Package name: ru.startandroid.develop.p0491simpleadaptercustom1
Create Activity: MainActivity

Рисуем экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <ListView
          android:id="@+id/lvSimple"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
     </ListView>
</LinearLayout>
```

На экране только список.

И layout для пункта списка **item.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal">
     <LinearLayout
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
```

```
            android:layout_marginBottom="5dp"
            android:layout_marginTop="5dp"
            android:orientation="horizontal">
        <ImageView
            android:id="@+id/ivImg"
            android:layout_width="30dp"
            android:layout_height="22dp"
            android:scaleType="fitCenter">
        </ImageView>
        <TextView
            android:id="@+id/tvValue"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:text="TextView"
            android:textSize="20sp">
        </TextView>
        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content">
            <TextView
                android:id="@+id/tvText"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center_vertical|center_horizontal"
                android:text="TextView"
                android:textSize="20sp">
            </TextView>
        </FrameLayout>
    </LinearLayout>
</LinearLayout>
```

Компонент **ivImg** будет отображать стрелку вниз или вверх, **tvValue** – значение динамики, **tvText** – номер дня.

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0491simpleadaptercustom1;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import android.app.Activity;
import android.content.Context;
import android.graphics.Color;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;

public class MainActivity extends Activity {
```

```
// имена атрибутов для Map
final String ATTRIBUTE_NAME_TEXT = "text";
final String ATTRIBUTE_NAME_VALUE = "value";
final String ATTRIBUTE_NAME_IMAGE = "image";

// картинки для отображения динамики
final int positive = android.R.drawable.arrow_up_float;
final int negative = android.R.drawable.arrow_down_float;

ListView lvSimple;

/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  // массив данных
  int[] values = { 8, 4, -3, 2, -5, 0, 3, -6, 1, -1 };

  // упаковываем данные в понятную для адаптера структуру
  ArrayList<Map<String, Object>> data = new ArrayList<Map<String, Object>>(
      values.length);
  Map<String, Object> m;
  int img = 0;
  for (int i = 0; i < values.length; i++) {
    m = new HashMap<String, Object>();
    m.put(ATTRIBUTE_NAME_TEXT, "Day " + (i + 1));
    m.put(ATTRIBUTE_NAME_VALUE, values[i]);
    if (values[i] == 0) img = 0; else
      img = (values[i] > 0) ? positive : negative;
    m.put(ATTRIBUTE_NAME_IMAGE, img);
    data.add(m);
  }

  // массив имен атрибутов, из которых будут читаться данные
  String[] from = { ATTRIBUTE_NAME_TEXT, ATTRIBUTE_NAME_VALUE,
      ATTRIBUTE_NAME_IMAGE };
  // массив ID View-компонентов, в которые будут вставлять данные
  int[] to = { R.id.tvText, R.id.tvValue, R.id.ivImg };

  // создаем адаптер
  MySimpleAdapter sAdapter = new MySimpleAdapter(this, data,
      R.layout.item, from, to);

  // определяем список и присваиваем ему адаптер
  lvSimple = (ListView) findViewById(R.id.lvSimple);
  lvSimple.setAdapter(sAdapter);
}

class MySimpleAdapter extends SimpleAdapter {

  public MySimpleAdapter(Context context,
      List<? extends Map<String, ?>> data, int resource,
      String[] from, int[] to) {
    super(context, data, resource, from, to);
  }

  @Override
  public void setViewText(TextView v, String text) {
    // метод супер-класса, который вставляет текст
```

```
      super.setViewText(v, text);
      // если нужный нам TextView, то разрисовываем
      if (v.getId() == R.id.tvValue) {
        int i = Integer.parseInt(text);
        if (i < 0) v.setTextColor(Color.RED); else
          if (i > 0) v.setTextColor(Color.GREEN);
      }
    }

    @Override
    public void setViewImage(ImageView v, int value) {
      // метод супер-класса
      super.setViewImage(v, value);
      // разрисовываем ImageView
      if (value == negative) v.setBackgroundColor(Color.RED); else
        if (value == positive) v.setBackgroundColor(Color.GREEN);
    }
  }
}
```

Код создания адаптера обычен. Придумываем данные **values** и упаковываем их в коллекцию **Map**-объектов. Каждый Map будет состоять из трех атрибутов

ATTRIBUTE_NAME_TEXT - текст с номером дня
ATTRIBUTE_NAME_VALUE - значение динамики
ATTRIBUTE_NAME_IMAGE - id картинки для отображения, в зависимости от значения (положительное или отрицательное)

Затем заполняем массивы сопоставления данных (from) и View (to), создаем адаптер, используя свой класс **MySimpleAdapter**, и настраиваем список.

Смотрим реализацию **MySimpleAdapter**. Конструктор вызывает конструктор супер-класса, тут ничего не меняем. Менять будем методы:

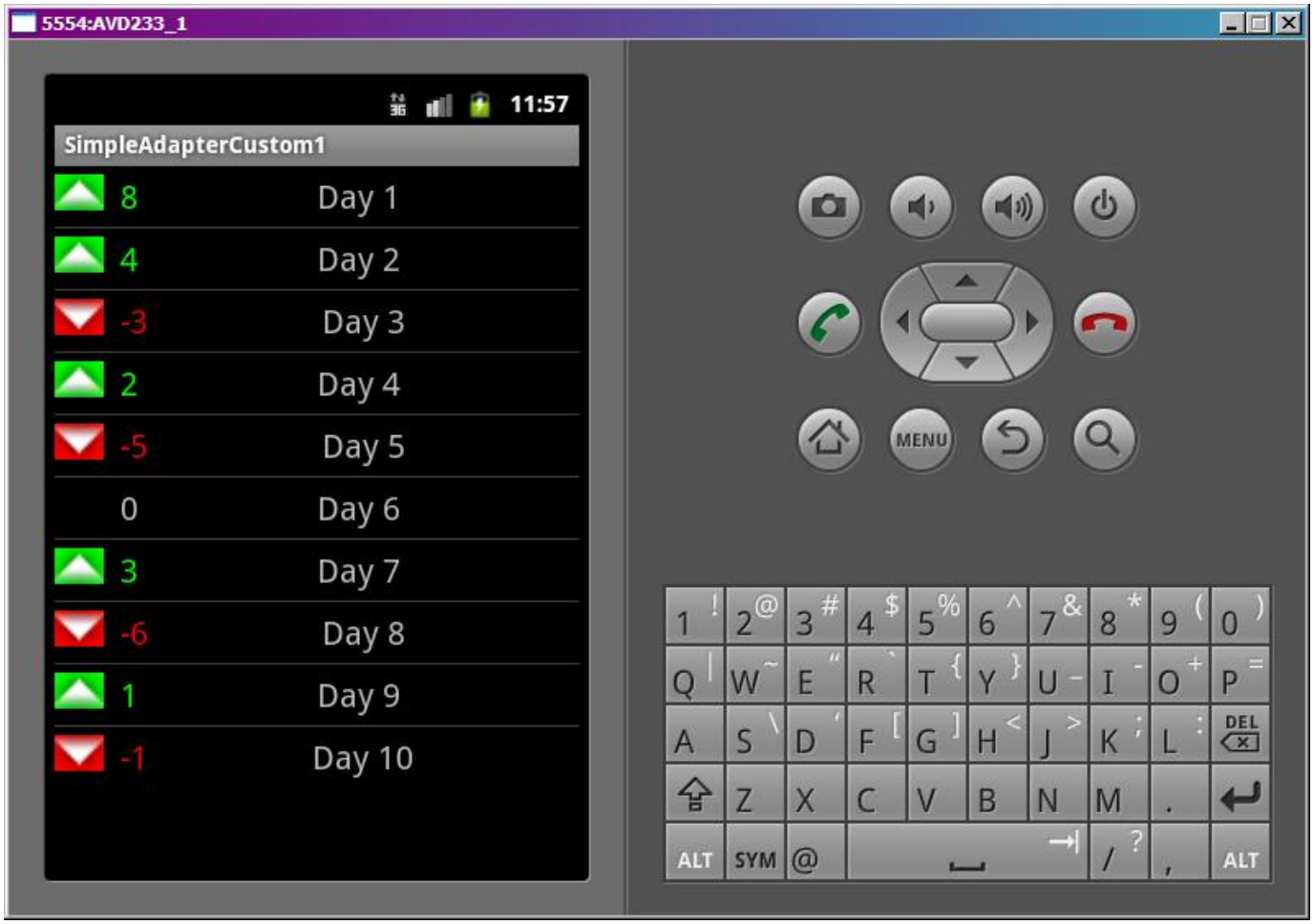
setViewText

Сначала выполняем метод супер-класса, который вставляет данные. А далее смотрим, если View, это тот TextView, который будет отображать значения, то меняем цвет текста на красный или зеленый в зависимости от значения, которое он будет отображать.

setViewImage

Выполняем метод супер-класса, чтобы ImageView получил изображение, а дальше меняем его фон в зависимости от значения. Картинка, которую мы вставляем , с альфа-слоем, поэтому фон будет виден. Проверку по id ImageView не выполняем, т.к. ImageView у нас только один.

Все сохраним и запускаем.



Отобразились картинка, значение и день. Фон картинки и цвет текста значения меняются в зависимости от значения – положительное или отрицательное.

Пример, конечно, не особо интересный, но надо было рассмотреть работу этих методов. Ничего лучше в голову не пришло)

На следующем уроке:

- используем свой SimpleAdapter.ViewBinder

Урок 50. SimpleAdapter. Используем ViewBinder

В этом уроке:

- используем свой SimpleAdapter.ViewBinder

Адаптер **SimpleAdapter** при своей работе сопоставляет **View**-компоненты и значения из **Map**-объектов. Как он это делает по умолчанию и с каким View-компонентами умеет работать, мы рассмотрели в предыдущих уроках. Но если нам не хватает этих возможностей, мы всегда можем **создать свой обработчик** и присвоить его адаптеру.

Для этого используется метод setViewBinder (SimpleAdapter.ViewBinder viewBinder), который на вход требует объект SimpleAdapter.ViewBinder. Мы создаем свой вариант этого биндера и реализуем в нем метод setViewValue(View view, Object data, String textRepresentation), в котором прописываем всю логику сопоставления данных и компонентов (биндинга). Метод возвращает значение boolean.

Алгоритм работы адаптера таков: он сначала проверяет, давали ли ему сторонний биндер.

Если находит, то выполняет его метод **setViewValue**. Если метод возвращает **true**, то адаптер считает, что обработка успешно **завершена**, если же **false** – то он выполняет биндинг в своем **стандартном** алгоритме, который мы рассмотрели на предыдущих уроках.

Если адаптер **не находит** сторонний биндер, он также выполняет **стандартный** биндинг.

Т.е. наша задача – заполнить метод **SimpleAdapter.ViewBinder.setViewValue**. И здесь уже нет ограничений на TextView или ImageView, может быть обработан и заполнен любой компонент. Создадим пример, в котором будем заполнять значение ProgressBar и менять цвет LinearLayout.

Это будет приложение-мониторинг, которое отображает уровень загрузки мощностей какой-то системы в разрезе дней. ProgressBar будет показывать уровень загрузки, а весь пункт списка будет подкрашиваться цветом в зависимости от уровня загрузки.

Создадим проект:

Project name: P0501_SimpleAdapterCustom2
Build Target: Android 2.3.3
Application name: SimpleAdapterCustom2
Package name: ru.startandroid.develop.p0501simpleadaptercustom2
Create Activity: MainActivity

Рисуем экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <ListView
          android:id="@+id/lvSimple"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
     </ListView>
```

```
</LinearLayout>
```

Только список.

Нам понадобится создать файл res/values/**colors.xml**, где мы перечислим требуемые нам цвета:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color  name="Bckgr">#9C9C9C</color>
    <color  name="Red">#33FF0000</color>
    <color  name="Orange">#33FFFF00</color>
    <color  name="Green">#3300FF00</color>
    <color  name="Black">#000000</color>
</resources>
```

Теперь создаем layout для пунктов списка res/layout/**item.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical"
     android:background="@color/Bckgr">
    <LinearLayout
         android:id="@+id/llLoad"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:orientation="vertical">
        <TextView
             android:id="@+id/tvLoad"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="TextView"
             android:layout_gravity="center_horizontal"
             android:textColor="@color/Black">
        </TextView>
        <ProgressBar
             android:id="@+id/pbLoad"
             style="?android:attr/progressBarStyleHorizontal"
             android:layout_width="match_parent"
             android:layout_height="wrap_content"
             android:layout_margin="5dp"
             android:max="100">
        </ProgressBar>
    </LinearLayout>
</LinearLayout>
```

llLoad – LinearLayout который занимает весь пункт списка, и который мы будем разукрашивать,
tvLoad – текст-инфа,
pbLoad – индикатор загрузки.

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0501simpleadaptercustom2;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ListView;
import android.widget.ProgressBar;
import android.widget.SimpleAdapter;

public class MainActivity extends Activity {

  // имена атрибутов для Map
  final String ATTRIBUTE_NAME_TEXT = "text";
  final String ATTRIBUTE_NAME_PB = "pb";
  final String ATTRIBUTE_NAME_LL = "ll";

  ListView lvSimple;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // массив данных
    int load[] = { 41, 48, 22, 35, 30, 67, 51, 88 };

    // упаковываем данные в понятную для адаптера структуру
    ArrayList<Map<String, Object>> data = new ArrayList<Map<String, Object>>(
        load.length);
    Map<String, Object> m;
    for (int i = 0; i < load.length; i++) {
      m = new HashMap<String, Object>();
      m.put(ATTRIBUTE_NAME_TEXT, "Day " + (i+1) + ". Load: " + load[i] + "%");
      m.put(ATTRIBUTE_NAME_PB, load[i]);
      m.put(ATTRIBUTE_NAME_LL, load[i]);
      data.add(m);
    }

    // массив имен атрибутов, из которых будут читаться данные
    String[] from = { ATTRIBUTE_NAME_TEXT, ATTRIBUTE_NAME_PB,
        ATTRIBUTE_NAME_LL };
    // массив ID View-компонентов, в которые будут вставлять данные
    int[] to = { R.id.tvLoad, R.id.pbLoad, R.id.llLoad };

    // создаем адаптер
    SimpleAdapter sAdapter = new SimpleAdapter(this, data, R.layout.item,
        from, to);
```

```
    // Указываем адаптеру свой биндер
    sAdapter.setViewBinder(new MyViewBinder());

    // определяем список и присваиваем ему адаптер
    lvSimple = (ListView) findViewById(R.id.lvSimple);
    lvSimple.setAdapter(sAdapter);
  }

  class MyViewBinder implements SimpleAdapter.ViewBinder {

    int red = getResources().getColor(R.color.Red);
    int orange = getResources().getColor(R.color.Orange);
    int green = getResources().getColor(R.color.Green);

    @Override
    public boolean setViewValue(View view, Object data,
        String textRepresentation) {
      int i = 0;
      switch (view.getId()) {
      // LinearLayout
      case R.id.llLoad:
        i = ((Integer) data).intValue();
        if (i < 40) view.setBackgroundColor(green); else
          if (i < 70) view.setBackgroundColor(orange); else
        view.setBackgroundColor(red);
        return true;
      // ProgressBar
      case R.id.pbLoad:
        i = ((Integer) data).intValue();
        ((ProgressBar)view).setProgress(i);
        return true;
      }
      return false;
    }
  }
}
```

Смотрим код. Заполняем массив данных загрузки по 100-бальной шкале. Формируем данные для адаптера: в **TextView** будем передавать краткую информацию (**String**), а в **PgrogressBar** и **LinearLayout** – значение загрузки (**int**). Заполняем массивы сопоставления, создаем адаптер, говорим ему, чтобы использовал наш биндер, и настраиваем список.

Вложенный класс **MyViewBinder** – это наша реализация биндера. Мы должны реализовать метод **setViewValue**, который будет использоваться адаптером для сопоставления данных из Map и View-компонентов. На вход ему идут:

view - View-компонент
data - данные для него
textRepresentation - текстовое представление данных (data.toString() или пустой String, но никогда не null)

Итак, смотрим реализацию. Сначала проверяется какой компонент нам дали на обработку.

В случае **llLoad** мы ожидаем данные типа int, поэтому выполняем приведение Object к Integer и получаем данные по загрузке из массива. А далее смотрим уровень этой загрузки. Если меньше 40, то будем считать, что все ок, цвет фона зеленый. От 40 до 70 – внимание, цвет желтый. Выше 70 – высокая нагрузка, красный цвет. Возвращаем **true**. Это важно! Тем самым мы говорим адаптеру, что мы успешно **выполнили** биндинг для данного компонента и выполнять для него стандартную обработку не надо.

В случае **pbLoad** мы также ожидаем int и выполняем приведение. А затем просто вызываем метод setProgress и передаем ему значение. В конце возвращаем true – мы сами успешно обработали компонент, стандартная обработка не нужна.

Для всех других View-компонентов метод **setViewValue** будет возвращать **false**. Это значит, компоненты пойдут на **стандартную** обработку адаптером. В нашем случае по этому пути отправится **tvLoad** и адаптер сам передаст ему текст из Map, наше вмешательство не нужно.

Все сохраняем и запускаем.

Получаем вот такую картинку. LinearLayout заполнился цветом, ProgressBar отобразил уровень загрузки, TextView показал информацию о дне и загрузке.

На следующем уроке:

- используем SimpleAdapter для построения списка
- добавляем и удаляем записи в списке

Урок 51. SimpleAdapter, добавление и удаление записей

В этом уроке:

- используем SimpleAdapter для построения списка
- добавляем и удаляем записи в списке

Как выводить данные в список с помощью **SimpleAdapter** мы знаем. Теперь попробуем эти данные менять. Сделаем список с возможностью **удаления** и **добавления** записей. Добавлять будем кнопкой, а удалять с помощью контекстного меню.

Создадим проект:

Project name: P0511_SimpleAdapterData
Build Target: Android 2.3.3
Application name: SimpleAdapterData
Package name: ru.startandroid.develop.p0511simpleadapterdata
Create Activity: MainActivity

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onButtonClick"
           android:text="Добавить запись">
     </Button>
     <ListView
           android:id="@+id/lvSimple"
           android:layout_width="match_parent"
           android:layout_height="wrap_content">
     </ListView>
</LinearLayout>
```

Кнопка для добавления и список. Из интересного можно отметить свойство **onClick** у кнопки. Дальше станет понятно, что это.

Layout для пункта списка **item.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
```

```
     android:layout_height="wrap_content"
     android:orientation="horizontal">
    <ImageView
         android:id="@+id/ivImg"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:src="@drawable/ic_launcher">
    </ImageView>
    <TextView
         android:id="@+id/tvText"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center_vertical"
         android:layout_marginLeft="10dp"
         android:text=""
         android:textSize="18sp">
    </TextView>
</LinearLayout>
```

Картинка и текст.

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0511simpleadapterdata;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView.AdapterContextMenuInfo;
import android.widget.ListView;
import android.widget.SimpleAdapter;

public class MainActivity extends Activity {

  private static final int CM_DELETE_ID = 1;

  // имена атрибутов для Map
  final String ATTRIBUTE_NAME_TEXT = "text";
  final String ATTRIBUTE_NAME_IMAGE = "image";

  ListView lvSimple;
  SimpleAdapter sAdapter;
  ArrayList<Map<String, Object>> data;
  Map<String, Object> m;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
```

```
    // упаковываем данные в понятную для адаптера структуру
    data = new ArrayList<Map<String, Object>>();
    for (int i = 1; i < 5; i++) {
      m = new HashMap<String, Object>();
      m.put(ATTRIBUTE_NAME_TEXT, "sometext " + i);
      m.put(ATTRIBUTE_NAME_IMAGE, R.drawable.ic_launcher);
      data.add(m);
    }

    // массив имен атрибутов, из которых будут читаться данные
    String[] from = { ATTRIBUTE_NAME_TEXT, ATTRIBUTE_NAME_IMAGE };
    // массив ID View-компонентов, в которые будут вставлять данные
    int[] to = { R.id.tvText, R.id.ivImg };

    // создаем адаптер
    sAdapter = new SimpleAdapter(this, data, R.layout.item, from, to);

    // определяем список и присваиваем ему адаптер
    lvSimple = (ListView) findViewById(R.id.lvSimple);
    lvSimple.setAdapter(sAdapter);
    registerForContextMenu(lvSimple);
  }

  public void onButtonClick(View v) {
    // создаем новый Map
    m = new HashMap<String, Object>();
    m.put(ATTRIBUTE_NAME_TEXT, "sometext " + (data.size() + 1));
    m.put(ATTRIBUTE_NAME_IMAGE, R.drawable.ic_launcher);
    // добавляем его в коллекцию
    data.add(m);
    // уведомляем, что данные изменились
    sAdapter.notifyDataSetChanged();
  }

  @Override
  public void onCreateContextMenu(ContextMenu menu, View v,
      ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.add(0, CM_DELETE_ID, 0, "Удалить запись");
  }

  @Override
  public boolean onContextItemSelected(MenuItem item) {
    if (item.getItemId() == CM_DELETE_ID) {
      // получаем инфу о пункте списка
      AdapterContextMenuInfo acmi = (AdapterContextMenuInfo) item.getMenuInfo();
      // удаляем Map из коллекции, используя позицию пункта в списке
      data.remove(acmi.position);
      // уведомляем, что данные изменились
      sAdapter.notifyDataSetChanged();
      return true;
    }
    return super.onContextItemSelected(item);
  }
}
```

В методе **onCreate** мы формируем коллекцию Map-объектов, массивы сопоставления, создаем адаптер и список, добавляем возможность контекстного меню для списка.

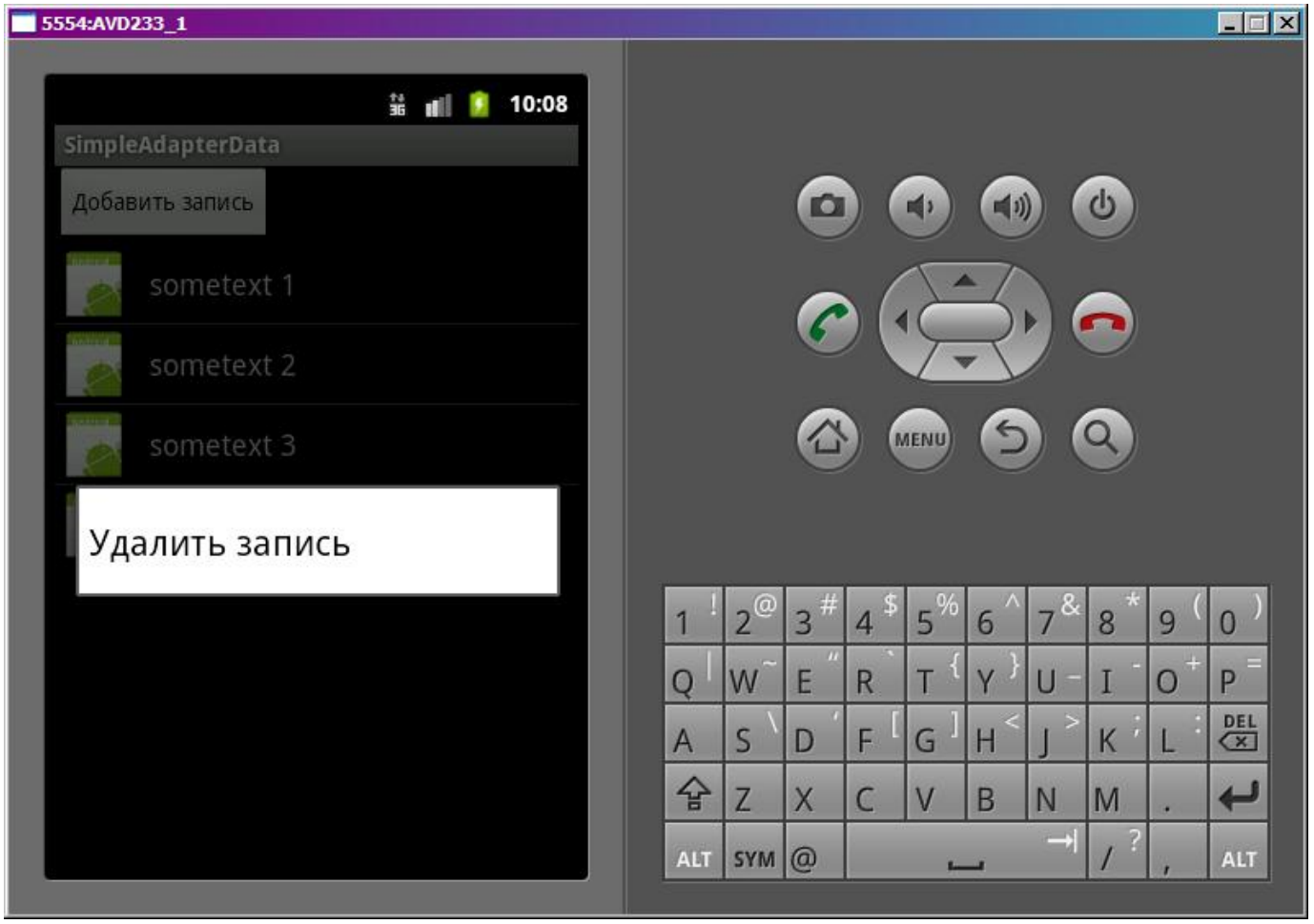
Метод **onButtonClick** – его мы указали в main.xml в свойстве **onClick** кнопки. И теперь при нажатии на кнопку выполнится этот метод. Отдельный обработчик нажатия не нужен.

В этом методе мы создаем новый Map, добавляем его к коллекции данных и сообщаем, что данные изменились и надо обновить список.

Метод **onCreateContextMenu** – создание контекстного меню. Создаем только один пункт - для удаления записи.

В **onContextItemSelected** обрабатываем нажатие на пункт контекстного меню. При вызове контекстного меню объект, для которого оно было вызвано, передает в меню информацию о себе. Чтобы получить данные по пункту списка, для которого был совершен вызов контекстного меню, мы используем метод getMenuInfo. Объект AdapterContextMenuInfo содержит данные о View, id и позиции пункта списка. Мы используем позицию для удаления соответствующего Map из коллекции. После этого сообщаем, что данные изменились.

Все сохраним и запустим.



На скрине показано контекстное меню, которое вызывается при долгом нажатии на пункт списка. За ним виден список и кнопка для добавления записей.

Записи добавляются и удаляются. Редактирование я не стал делать. Там принцип тот же. Получаете Map и меняете его атрибуты.

Из кода видно, что для обновления списка надо поменять данные, которые использует адаптер, и вызвать его метод-уведомление.

На следующем уроке:

- используем SimpleCursorAdapter для построения списка
- добавляем и удаляем записи в списке

Урок 52. SimpleCursorAdapter, пример использования

В этом уроке:

- используем SimpleCursorAdapter для построения списка
- добавляем и удаляем записи в списке

После нескольких уроков посвященных **SimpleAdapter** мы знаем про него достаточно и представляем схему его работы. И теперь нам будет нетрудно усвоить **SimpleCursorAdapter**. Он отличается тем, что в качестве данных используется не коллекция Map, а Cursor с данными из БД. И в массиве from, соответственно, мы указываем не ключи Map-атрибутов, а наименования полей (столбца) курсора. Значения из этих полей будут сопоставлены указанным View-компонентам из массива to.

Также немного отличается от SimpleAdapter стандартный биндинг и внешний ViewBinder. SimpleCursorAdapter умеет работать с TextView и ImageView компонентами и их производными, а Checkable-производные не воспримет . А при использовании ViewBinder, необходимо реализовать его метод boolean setViewValue (View view, Cursor cursor, int columnIndex). На вход он принимает **View**-компонент для биндинга, **cursor** с данными и **номер столбца**, из которого надо взять данные. Позиция курсора уже установлена в соответствии с позицией пункта списка. Не буду снова расписывать примеры использования, т.к. они будут очень похожи на примеры из предыдущих уроков по SimpleAdapter. Если там все было понятно, то и здесь проблем не должно возникнуть.

Итак, давайте накидаем пример использования **SimpleCursorAdapter**. Список будет отображать картинку и текст. Также реализуем возможность **добавления** и **удаления** данных из списка. Добавлять будем кнопкой, а удалять с помощью контекстного меню.

Создадим проект:

Project name: P0521_SimpleCursorAdapter
Build Target: Android 2.3.3
Application name: SimpleCursorAdapter
Package name: ru.startandroid.develop.p0521simplecursoradapter
Create Activity: MainActivity

Обычно в уроках я тексты для кнопок и прочего указывал напрямую. Делал я это не со зла, а чтобы не перегружать урок лишней информацией. Но с последними обновлениями Eclipse стал ругаться примерно так: ***[I18N] Hardcoded string "какой-то текст", should use @string resource***. Ошибка не критична и запуску приложения никак не помешает, но некоторых она смущает. Да и действительно, хардкод – это плохо. С этого урока постараюсь следовать правилам хорошего тона и использовать файлы **ресурсов**. На нашем текущем уровне знаний это не должно стать помехой в понимании и усвоении уроков.

Заполняем **res/values/string.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">SimpleCursorAdapter</string>
    <string name="add_record">Добавить запись</string>
    <string name="delete_record">Удалить запись</string>
</resources>
```

Тут кроме названия приложения я записал тексты для кнопки и контекстного меню

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onButtonClick"
           android:text="@string/add_record">
     </Button>
     <ListView
           android:id="@+id/lvData"
           android:layout_width="match_parent"
           android:layout_height="wrap_content">
     </ListView>
</LinearLayout>
```

Кнопка для добавления записи и список.

Layout для пункта списка **item.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal">
     <ImageView
           android:id="@+id/ivImg"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:src="@drawable/ic_launcher">
     </ImageView>
     <TextView
           android:id="@+id/tvText"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_gravity="center_vertical"
           android:layout_marginLeft="10dp"
           android:text=""
           android:textSize="18sp">
     </TextView>
</LinearLayout>
```

Картинка и текст.

Т.к. **SimpleCursorAdapter** – это адаптер для работы с данными из **БД**, то нам нужно эту БД организовать. Чтобы не загромождать **MainActivity.java**, я вынесу код по работе с БД в отдельный класс **DB**. Создаем класс **DB.java** в том же пакете, где и MainActivity.java

Код **DB.java**:

```
package ru.startandroid.develop.p0521simplecursoradapter;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class DB {

  private static final String DB_NAME = "mydb";
  private static final int DB_VERSION = 1;
  private static final String DB_TABLE = "mytab";

  public static final String COLUMN_ID = "_id";
  public static final String COLUMN_IMG = "img";
  public static final String COLUMN_TXT = "txt";

  private static final String DB_CREATE =
    "create table " + DB_TABLE + "(" +
      COLUMN_ID + " integer primary key autoincrement, " +
      COLUMN_IMG + " integer, " +
      COLUMN_TXT + " text" +
    ");";

  private final Context mCtx;


  private DBHelper mDBHelper;
  private SQLiteDatabase mDB;

  public DB(Context ctx) {
    mCtx = ctx;
  }

  // открыть подключение
  public void open() {
    mDBHelper = new DBHelper(mCtx, DB_NAME, null, DB_VERSION);
    mDB = mDBHelper.getWritableDatabase();
  }

  // закрыть подключение
  public void close() {
    if (mDBHelper!=null) mDBHelper.close();
  }

  // получить все данные из таблицы DB_TABLE
  public Cursor getAllData() {
    return mDB.query(DB_TABLE, null, null, null, null, null, null);
  }

  // добавить запись в DB_TABLE
  public void addRec(String txt, int img) {
```

```
    ContentValues cv = new ContentValues();
    cv.put(COLUMN_TXT, txt);
    cv.put(COLUMN_IMG, img);
    mDB.insert(DB_TABLE, null, cv);
  }

  // удалить запись из DB_TABLE
  public void delRec(long id) {
    mDB.delete(DB_TABLE, COLUMN_ID + " = " + id, null);
  }

  // класс по созданию и управлению БД
  private class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context, String name, CursorFactory factory,
        int version) {
      super(context, name, factory, version);
    }

    // создаем и заполняем БД
    @Override
    public void onCreate(SQLiteDatabase db) {
      db.execSQL(DB_CREATE);

      ContentValues cv = new ContentValues();
      for (int i = 1; i < 5; i++) {
        cv.put(COLUMN_TXT, "sometext " + i);
        cv.put(COLUMN_IMG, R.drawable.ic_launcher);
        db.insert(DB_TABLE, null, cv);
      }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
  }
}
```

Здесь все нам знакомо по прошлым урокам **SQLite**.

Мы создаем несколько public методов, чтобы Activity могла через них работать с данными:

open – установить соединение
close – закрыть соединение
getAllData – получить курсор со всеми данными из таблицы
addRec – добавить запись
delRec – удалить запись

Это методы-оболочки для работы с БД, которые предоставят MainActivity только те возможности, какие ей нужны.

Вложенный класс **DBHelper** – для создания и управления БД. В методе **onCreate** мы создаем таблицу и заполняем ее сгенерированными данными. Метод **onUpgrade** я оставил пустым, т.к. в этом примере не планирую обновлять версию БД.

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0521simplecursoradapter;
```

```
import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView.AdapterContextMenuInfo;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

public class MainActivity extends Activity {

  private static final int CM_DELETE_ID = 1;
  ListView lvData;
  DB db;
  SimpleCursorAdapter scAdapter;
  Cursor cursor;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // открываем подключение к БД
    db = new DB(this);
    db.open();

    // получаем курсор
    cursor = db.getAllData();
    startManagingCursor(cursor);

    // формируем столбцы сопоставления
    String[] from = new String[] { DB.COLUMN_IMG, DB.COLUMN_TXT };
    int[] to = new int[] { R.id.ivImg, R.id.tvText };

    // создааем адаптер и настраиваем список
    scAdapter = new SimpleCursorAdapter(this, R.layout.item, cursor, from, to);
    lvData = (ListView) findViewById(R.id.lvData);
    lvData.setAdapter(scAdapter);

    // добавляем контекстное меню к списку
    registerForContextMenu(lvData);
  }

  // обработка нажатия кнопки
  public void onButtonClick(View view) {
    // добавляем запись
    db.addRec("sometext " + (cursor.getCount() + 1), R.drawable.ic_launcher);
    // обновляем курсор
    cursor.requery();
  }

  public void onCreateContextMenu(ContextMenu menu, View v,
      ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.add(0, CM_DELETE_ID, 0, R.string.delete_record);
  }
```

```
  public boolean onContextItemSelected(MenuItem item) {
    if (item.getItemId() == CM_DELETE_ID) {
      // получаем из пункта контекстного меню данные по пункту списка
      AdapterContextMenuInfo acmi = (AdapterContextMenuInfo) item.getMenuInfo();
      // извлекаем id записи и удаляем соответствующую запись в БД
      db.delRec(acmi.id);
      // обновляем курсор
      cursor.requery();
      return true;
    }
    return super.onContextItemSelected(item);
  }

  protected void onDestroy() {
    super.onDestroy();
    // закрываем подключение при выходе
    db.close();
  }

}
```

Хорошо, что мы создали DB.java. Благодаря ему в MainActivity.java все красиво, прозрачно и удобно. Смотрим код.

В **onCreate** мы организуем подключение к БД, получаем курсор и просим Activity присмотреть за ним. Теперь при смене Lifecycle-состояний Activity, оно будет менять соответствующим образом состояния курсора. Затем настраиваем биндинг – формируем массивы, которые укажут адаптеру, как сопоставлять данные из курсора и View-компоненты. В R.id.ivImg пойдет значение из поля img, а в R.id.tvText – значение из поля txt. Имена полей мы здесь указываем public-константами класса DB. Далее мы создаем адаптер и настраиваем список на его использование. В конце добавляем контекстное меню к списку.

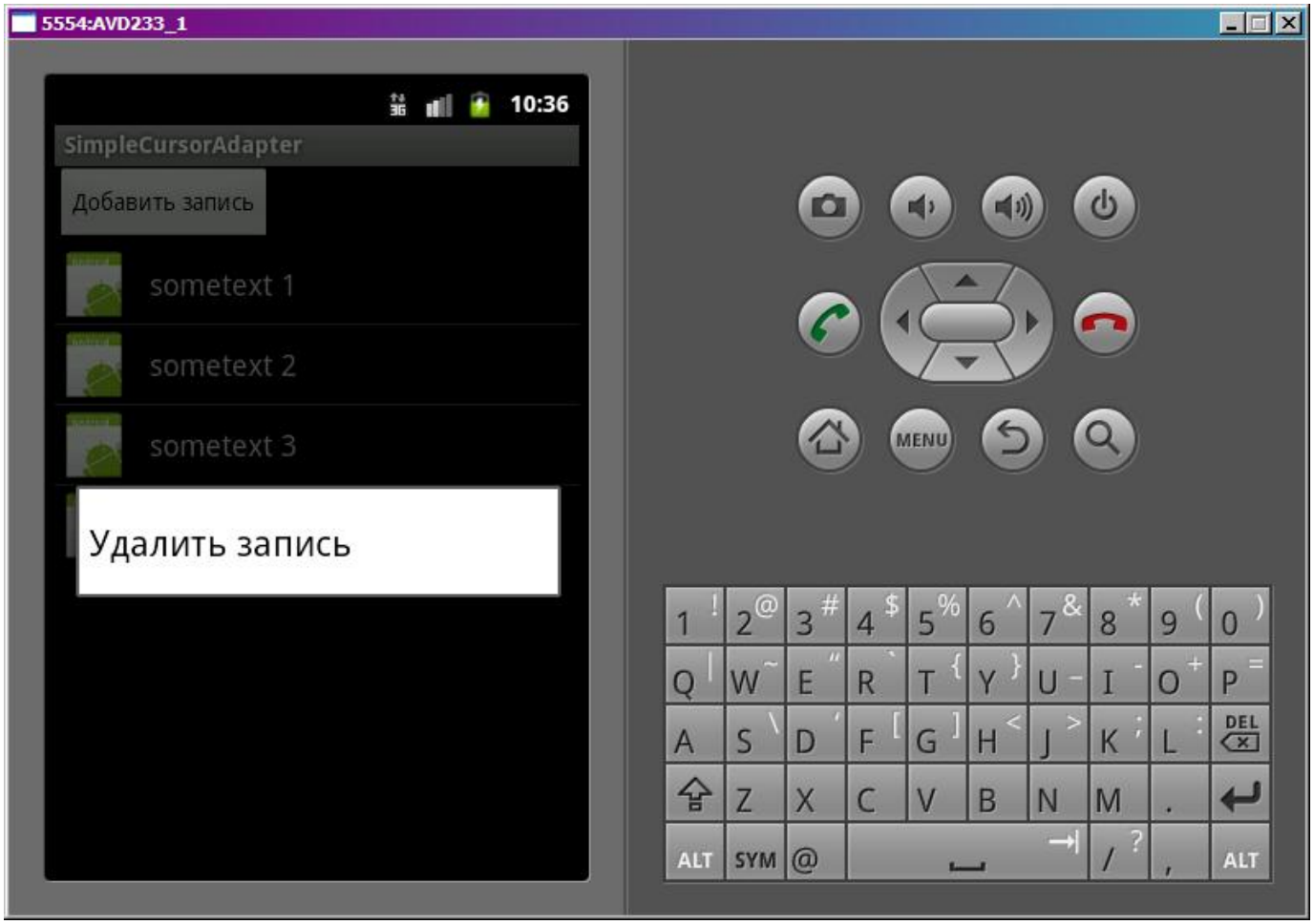
В методе **onButtonClick** мы генерируем и добавляем запись в БД и обновляем курсор методом requery, чтобы получить свежие данные из БД.

При создании контекстного меню, в методе **onCreateContextMenu**, мы добавляем пункт для удаления.

В методе **onContextItemSelected** мы обрабатываем нажатие пункта контекстного меню. Чтобы получить данные по пункту списка, для которого был совершен вызов контекстного меню, мы используем метод getMenuInfo. Объект **AdapterContextMenuInfo** содержит данные о View, id и позиции пункта списка. Нам нужно **id**. Этот **id** равен значению поля **_id** для соответствующей записи в курсоре. Мы вызываем метод удаления записи и обновляем курсор.

В методе **onDestroy** мы закрываем подключение к БД. Это будет происходить при закрытии Activity.

Все сохраняем и запускаем.



Нажав на кнопку, мы добавляем запись. А вызвав контекстное меню (долгое нажатие) для пункта списка можно его удалить.

Мы рассмотрели возможность добавления и удаления записей в списке при использовании SimpleCursorAdapter. Возможность редактирования я рассматривать не стал. Это не особо усложнило бы урок, но сделало бы его больше и размыло бы тему. А я стараюсь делать уроки максимально заточенными под конкретную тему. Для тех, кому интересно редактирование – гугл любезно создал такой пример на официальном сайте - http://developer.android.com/resources/tutorials/notepad/index.html. Мой пример похож на него, так что будет проще разобраться.

Кстати, в этом уроке мы встретили список, в котором id пункта может не совпадать с позицией. Для теста попробуйте повесить обработку нажатия на пункт списка и посмотреть, что позиция – это будет позиция пункта в списке, а id – это идентификатор записи из БД (поле _id). Чтобы это работало, необходимо поле-идентификатор в таблице называть _id, т.к. курсор будет использовать его, как id. Иначе получим ошибку.

На следующем уроке:

- используем SimpleCursorTreeAdapter для построения списка

Урок 53. SimpleCursorTreeAdapter, пример использования

В этом уроке:

- используем SimpleCursorTreeAdapter для построения списка

SimpleCursorTreeAdapter не является полноценным готовым адаптером. Это абстрактный класс, он требует реализацию метода Cursor getChildrenCursor (Cursor groupCursor).

Что это за метод и чего в нем нужно написать, чтобы все заработало? Чтобы понять это, надо разобраться в алгоритме работы этого адаптера, давайте рассмотрим один из конструкторов: SimpleCursorTreeAdapter (Context context, Cursor cursor, int groupLayout, String[] groupFrom, int[] groupTo, int childLayout, String[] childFrom, int[] childTo).

Мы указываем в конструкторе **cursor** – это курсор групп. В нем содержатся **данные** по **группам**, необходимые вам для вывода данных в список. Т.е. все как при использовании **SimpleCursorAdapter** (см. предыдущие уроки). И адаптер спокойно отображает свернутые группы, используя данные из этого курсора. При сопоставлении полей курсора и View-компонентов он использует массивы **groupFrom**[] и **groupTo**[].

Но если мы захотим группу развернуть и посмотреть **элементы**, то адаптеру неоткуда взять данные по элементам. Курсор **cursor** содержит данные только по **группам**. В этом случае адаптер ставит в **cursor** позицию, соответствующую раскрываемой группе, вызывает метод **getChildrenCursor**, передает ему этот курсор групп и ожидает в ответ **курсор элементов** этой группы.

Мы в реализации этого метода получаем id текущей раскрываемой группы из курсора групп, делаем по нему запрос в БД и получаем в виде курсора данные по **элементам** группы. Этот курсор элементов мы возвращаем, как **результат** метода **getChildrenCursor** и адаптер использует его, чтобы создать элементы группы. При сопоставлении полей курсора и View-компонентов используются массивы **childFrom** и **childTo**.

Ну и поглядим, что там еще осталось в конструкторе:
context – контекст
groupLayout – layout-ресурс для отображения группы
childLayout - layout-ресурс для отображения элемента

В целом все несложно, рассмотрим на примере. Сделаем список из компаний (групп) и их смартфонов (элементов).

Создадим проект:

Project name: P0531_SimpleCursorTreeAdapter
Build Target: Android 2.3.3
Application name: SimpleCursorTreeAdapter
Package name: ru.startandroid.develop.p0531simplecursortreeadapter
Create Activity: MainActivity

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <ExpandableListView
```

```
        android:id="@+id/elvMain"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </ExpandableListView>
</LinearLayout>
```

Т.к. нам нужна будет БД, выделим под работу с ней отдельный класс

DB.java:

```
package ru.startandroid.develop.p0531simplecursortreeadapter;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class DB {

  private static final String DB_NAME = "mydb";
  private static final int DB_VERSION = 1;

  // имя таблицы компаний, поля и запрос создания
  private static final String COMPANY_TABLE = "company";
  public static final String COMPANY_COLUMN_ID = "_id";
  public static final String COMPANY_COLUMN_NAME = "name";
  private static final String COMPANY_TABLE_CREATE = "create table "
      + COMPANY_TABLE + "(" + COMPANY_COLUMN_ID
      + " integer primary key, " + COMPANY_COLUMN_NAME + " text" + ");";

  // имя таблицы телефонов, поля и запрос создания
  private static final String PHONE_TABLE = "phone";
  public static final String PHONE_COLUMN_ID = "_id";
  public static final String PHONE_COLUMN_NAME = "name";
  public static final String PHONE_COLUMN_COMPANY = "company";
  private static final String PHONE_TABLE_CREATE = "create table "
      + PHONE_TABLE + "(" + PHONE_COLUMN_ID
      + " integer primary key autoincrement, " + PHONE_COLUMN_NAME
      + " text, " + PHONE_COLUMN_COMPANY + " integer" + ");";

  private final Context mCtx;

  private DBHelper mDBHelper;
  private SQLiteDatabase mDB;

  public DB(Context ctx) {
    mCtx = ctx;
  }

  // открываем подключение
  public void open() {
    mDBHelper = new DBHelper(mCtx, DB_NAME, null, DB_VERSION);
    mDB = mDBHelper.getWritableDatabase();
```

```
}

// закрываем подключение
public void close() {
  if (mDBHelper != null)
    mDBHelper.close();
}

// данные по компаниям
public Cursor getCompanyData() {
  return mDB.query(COMPANY_TABLE, null, null, null, null, null, null);
}

// данные по телефонам конкретной группы
public Cursor getPhoneData(long companyID) {
  return mDB.query(PHONE_TABLE, null, PHONE_COLUMN_COMPANY + " = "
      + companyID, null, null, null, null);
}

private class DBHelper extends SQLiteOpenHelper {

  public DBHelper(Context context, String name, CursorFactory factory,
      int version) {
    super(context, name, factory, version);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    ContentValues cv = new ContentValues();

    // названия компаний (групп)
    String[] companies = new String[] { "HTC", "Samsung", "LG" };

    // создаем и заполняем таблицу компаний
    db.execSQL(COMPANY_TABLE_CREATE);
    for (int i = 0; i < companies.length; i++) {
      cv.put(COMPANY_COLUMN_ID, i + 1);
      cv.put(COMPANY_COLUMN_NAME, companies[i]);
      db.insert(COMPANY_TABLE, null, cv);
    }

    // названия телефонов (элементов)
    String[] phonesHTC = new String[] { "Sensation", "Desire",
        "Wildfire", "Hero" };
    String[] phonesSams = new String[] { "Galaxy S II", "Galaxy Nexus",
        "Wave" };
    String[] phonesLG = new String[] { "Optimus", "Optimus Link",
        "Optimus Black", "Optimus One" };

    // создаем и заполняем таблицу телефонов
    db.execSQL(PHONE_TABLE_CREATE);
    cv.clear();
    for (int i = 0; i < phonesHTC.length; i++) {
      cv.put(PHONE_COLUMN_COMPANY, 1);
      cv.put(PHONE_COLUMN_NAME, phonesHTC[i]);
      db.insert(PHONE_TABLE, null, cv);
    }
    for (int i = 0; i < phonesSams.length; i++) {
      cv.put(PHONE_COLUMN_COMPANY, 2);
      cv.put(PHONE_COLUMN_NAME, phonesSams[i]);
```

```
        db.insert(PHONE_TABLE, null, cv);
      }
      for (int i = 0; i < phonesLG.length; i++) {
        cv.put(PHONE_COLUMN_COMPANY, 3);
        cv.put(PHONE_COLUMN_NAME, phonesLG[i]);
        db.insert(PHONE_TABLE, null, cv);
      }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
  }

}
```

Мы создаем две таблицы **company** (компании) и **phone** (телефоны). В таблице телефонов мы указываем для каждого телефона **id ero группы** из таблицы компаний. Также создаем методы для открытия и закрытия подключения, и методы для получения данных по группе и по элементам конкретной группы.

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0531simplecursortreeadapter;

import android.app.Activity;
import android.content.Context;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.ExpandableListView;
import android.widget.SimpleCursorTreeAdapter;

public class MainActivity extends Activity {

  ExpandableListView elvMain;
  DB db;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // подключаемся к БД
    db = new DB(this);
    db.open();

    // готовим данные по группам для адаптера
    Cursor cursor = db.getCompanyData();
    startManagingCursor(cursor);
    // сопоставление данных и View для групп
    String[] groupFrom = { DB.COMPANY_COLUMN_NAME };
    int[] groupTo = { android.R.id.text1 };
    // сопоставление данных и View для элементов
    String[] childFrom = { DB.PHONE_COLUMN_NAME };
    int[] childTo = { android.R.id.text1 };

    // создаем адаптер и настраиваем список
    SimpleCursorTreeAdapter sctAdapter = new MyAdapter(this, cursor,
```

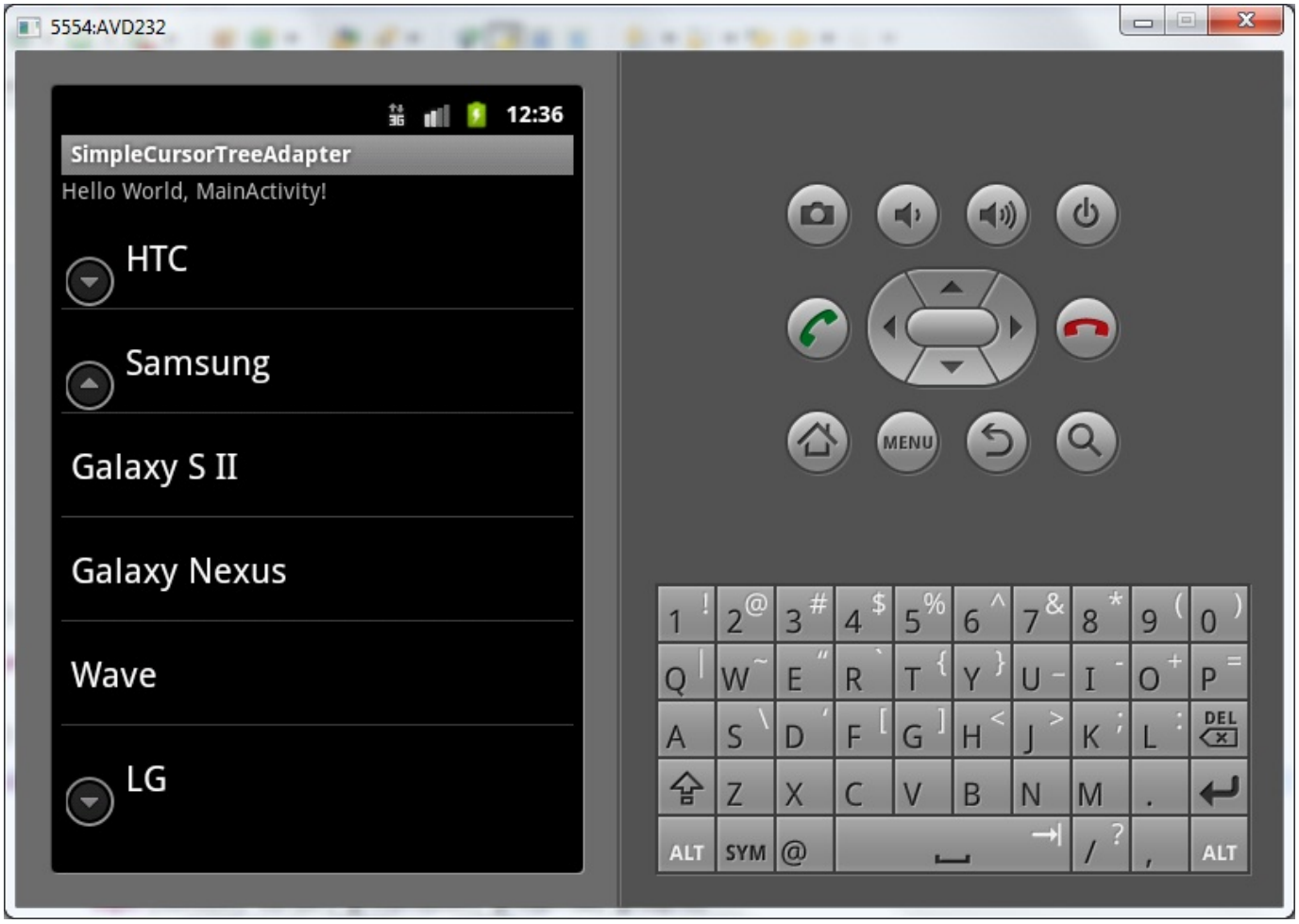
```
        android.R.layout.simple_expandable_list_item_1, groupFrom,
        groupTo, android.R.layout.simple_list_item_1, childFrom,
        childTo);
    elvMain = (ExpandableListView) findViewById(R.id.elvMain);
    elvMain.setAdapter(sctAdapter);
  }

  protected void onDestroy() {
    super.onDestroy();
    db.close();
  }

  class MyAdapter extends SimpleCursorTreeAdapter {

    public MyAdapter(Context context, Cursor cursor, int groupLayout,
        String[] groupFrom, int[] groupTo, int childLayout,
        String[] childFrom, int[] childTo) {
      super(context, cursor, groupLayout, groupFrom, groupTo,
          childLayout, childFrom, childTo);
    }

    protected Cursor getChildrenCursor(Cursor groupCursor) {
      // получаем курсор по элементам для конкретной группы
      int idColumn = groupCursor.getColumnIndex(DB.COMPANY_COLUMN_ID);
      return db.getPhoneData(groupCursor.getInt(idColumn));
    }
  }
}
```

В **onCreate** мы подключаемся к БД, создаем адаптер и список.

В **onDestroy** закрываем подключение.

Класс **MyAdapter** – реализация абстрактного **SimpleCursorTreeAdapter**. Конструктор используем по умолчанию. В методе **getChildrenCursor** мы получаем id текущей группы и по нему получаем курсор с данными об элементах группы.

Все сохраняем, запускаем.



Список-дерево работает.

Если вы повесите для списка обработчики нажатий на группы и элементы, они будут предоставлять вам не только позицию пункта в списке, но и соответствующее значение _id из БД.

Сопоставление **данных** и **View**-компонентов происходит так же, как и для **SimpleCursorAdapter**. Если есть внешний ViewBinder, вызывается его метод setViewValue. Если нет, то идет стандартная обработка для TextView и ImageView. Подробности биндинга можно найти в прошлых уроках.

На следующем уроке:

- создаем свой адаптер на основе BaseAdapter

Урок 54. Кастомизация списка. Создаем свой адаптер

В этом уроке:

- создаем свой адаптер на основе BaseAdapter

Предоставляемые нам адаптеры универсальны и полезны, но иногда их возможностей не хватает для реализации задуманного. Тогда возникает необходимость написать свой адаптер. Попробуем и мы. Создавать будем не с нуля, а используя BaseAdapter.

Сделаем подобие интернет магазина. Будем выводить список товаров. Каждый пункт списка будет содержать название товара, цену и изображение. Также будет возможность отметить пункт галкой, поместив его тем самым в корзину.

Внизу списка сделаем кнопку, которая будет отображать содержимое корзины. В настоящем интернет-магазине мы повесили бы на нее, например, переход к созданию заказа.

Создадим проект:

Project name: P0541_CustomAdapter
Build Target: Android 2.3.3
Application name: CustomAdapter
Package name: ru.startandroid.develop.p0541customadapter
Create Activity: MainActivity

В файл **strings.xml** добавим текстовый параметр для названия кнопки.

```
    <string  name="box">Корзина</string>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <ListView
         android:id="@+id/lvMain"
         android:layout_width="match_parent"
         android:layout_height="match_parent"
         android:layout_weight="1">
    </ListView>
    <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center_horizontal"
         android:layout_marginTop="5dp"
         android:onClick="showResult"
         android:text="@string/box">
```

```
    </Button>
</LinearLayout>
```

layout для пункта списка – **item.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal">
     <CheckBox
          android:id="@+id/cbBox"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content">
     </CheckBox>
     <LinearLayout
          android:id="@+id/linearLayout1"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:layout_marginLeft="5dp"
          android:layout_weight="1"
          android:orientation="vertical">
         <TextView
              android:id="@+id/tvDescr"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_marginTop="5dp"
              android:text=""
              android:textSize="20sp">
         </TextView>
         <TextView
              android:id="@+id/tvPrice"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:layout_gravity="right"
              android:layout_marginRight="10dp"
              android:text="">
         </TextView>
     </LinearLayout>
     <ImageView
          android:id="@+id/ivImage"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:src="@drawable/ic_launcher">
     </ImageView>
</LinearLayout>
```

Чекбокс, пара текстовых полей и картинка.

Теперь пишем код. Можно все написать в MainActivity.java, но тогда он получится достаточно большим и неудобным для чтения. Я раскидаю весь код по трем классам.

Product.java – класс, описывающий товар:

```
package ru.startandroid.develop.p0541customadapter;

public class Product {

  String name;
  int price;
  int image;
  boolean box;


  Product(String _describe, int _price, int _image, boolean _box) {
    name = _describe;
    price = _price;
    image = _image;
    box = _box;
  }
}
```

Тут все просто – только конструктор и элементы класса. Не заморачиваюсь с доступом и методами Set/Get, чтобы не усложнять код.

BoxAdapter.java – созданный адаптер, который будем использовать

```
package ru.startandroid.develop.p0541customadapter;

import java.util.ArrayList;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.ImageView;
import android.widget.TextView;

public class BoxAdapter extends BaseAdapter {
  Context ctx;
  LayoutInflater lInflater;
  ArrayList<Product> objects;

  BoxAdapter(Context context, ArrayList<Product> products) {
    ctx = context;
    objects = products;
    lInflater = (LayoutInflater) ctx
        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
  }

  // кол-во элементов
```

```
@Override
public int getCount() {
  return objects.size();
}

// элемент по позиции
@Override
public Object getItem(int position) {
  return objects.get(position);
}

// id по позиции
@Override
public long getItemId(int position) {
  return position;
}

// пункт списка
@Override
public View getView(int position, View convertView, ViewGroup parent) {
  // используем созданные, но не используемые view
  View view = convertView;
  if (view == null) {
    view = lInflater.inflate(R.layout.item, parent, false);
  }

  Product p = getProduct(position);

  // заполняем View в пункте списка данными из товаров: наименование, цена
  // и картинка
  ((TextView) view.findViewById(R.id.tvDescr)).setText(p.name);
  ((TextView) view.findViewById(R.id.tvPrice)).setText(p.price + "");
  ((ImageView) view.findViewById(R.id.ivImage)).setImageResource(p.image);

  CheckBox cbBuy = (CheckBox) view.findViewById(R.id.cbBox);
  // присваиваем чекбоксу обработчик
  cbBuy.setOnCheckedChangeListener(myCheckChangList);
  // пишем позицию
  cbBuy.setTag(position);
  // заполняем данными из товаров: в корзине или нет
  cbBuy.setChecked(p.box);
  return view;
}

// товар по позиции
Product getProduct(int position) {
  return ((Product) getItem(position));
}

// содержимое корзины
ArrayList<Product> getBox() {
  ArrayList<Product> box = new ArrayList<Product>();
  for (Product p : objects) {
    // если в корзине
    if (p.box)
      box.add(p);
  }
  return box;
}
```

```
    // обработчик для чекбоксов
    OnCheckedChangeListener myCheckChangList = new OnCheckedChangeListener() {
      public void onCheckedChanged(CompoundButton buttonView,
          boolean isChecked) {
        // меняем данные товара (в корзине или нет)
        getProduct((Integer) buttonView.getTag()).box = isChecked;
      }
    };
  }
```

На всякий случай напомню общий принцип действия **адаптера**: он получает **данные** и выдает **View** для отображения пункта списка.

Смотрим код. В **конструкторе** мы заполняем наши внутренние переменные и получаем **LayoutInflater** для работы с layout-ресурсами. В **objects** у нас теперь хранится список товаров, которые надо отобразить в списке.

Методы, отмеченные аннотацией **@Override**, мы обязаны реализовать при наследовании **BaseAdapter**. Эти методы используются списком и должны работать корректно.

Метод **getCount** должен возвращать кол-во элементов. Мы возвращаем кол-во товаров.

Метод **getItem** должен возвращать элемент по указанной позиции. Используя позицию, получаем конкретный элемент из objects.

Метод **getItemId** должен возвращать id элемента. Здесь не заморачиваемся и возвращаем позицию. Кстати, также сделано в некоторых адаптерах. Поэтому мы и видели в обработчиках, что id = position.

Метод **getView** должен возвращать **View** пункта списка. Для этого мы создавали layout-ресурс **R.layout.item**. В этом методе мы должны из R.layout.item создать View, заполнить его данными и отдать списку. Но перед тем как создавать, мы пробуем использовать **convertView**, который идет на вход метода. Это уже созданное ранее View, но неиспользуемое в данный момент. Например, при прокрутке списка, часть пунктов уходит за экран и их уже не надо прорисовывать. View из этих «невидимых» пунктов используются для новых пунктов. Нам остается только заполнить их данными. Это значительно ускоряет работу приложения, т.к. не надо прогонять inflate лишний раз.

Если же convertView в этот раз нам не дали (null), то создаем сами view. Далее заполняем наименования, цену и картинку из данных по товарам. Для чекбокса мы присваиваем обработчик, сохраняем в Tag позицию элемента и ставим галку, если товар уже в корзине.

Tag – это некое Object-хранилище у каждого **View**, куда вы можете поместить нужные вам данные. В нашем случае я для каждого чекбокса помещаю в его Tag номер позиции пункта списка. Далее в обработчике чекбокса я смогу этот номер позиции извлечь и определить, в каком пункте списка был нажат чекбокс.

В итоге, метод **getView** возвращает списку полностью заполненное **view**, и список его отобразит как очередной пункт.

Далее идет пара методов, которые не обязательно было создавать при наследовании BaseAdapter. Я их создал для удобства.

Метод **getProduct** – это аналог getItem, но он сразу конвертирует Object в Product. Он используется всего пару раз. И в принципе, можно было бы и без него обойтись.

Метод **getBox** проверяет, какие товары отмечены галками и формирует из них коллекцию-корзину.

myCheckChangList – обработчик для чекбоксов. Когда мы нажимаем на чекбокс в списке, он срабатывает, читает из

Tag позицию пункта списка и помечает соответствующий товар, как положенный в корзину.

Тут важно понимать, что без этого обработчика не работало бы помещение товаров в корзину. Да и на экране - значения чекбоксов в списке терялись бы при прокрутке. Потому что пункты списка пересоздаются, если они уйдут «за экран» и снова появятся. Это пересоздание обеспечивает метод getView, а он для заполнения View берет данные из товаров. Значит при нажатии на чекбокс, обязательно надо сохранить в данных о товаре то, что он теперь в корзине.

Остается накодить **MainActivity.java**:

```
package ru.startandroid.develop.p0541customadapter;

import java.util.ArrayList;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ListView;
import android.widget.Toast;

public class MainActivity extends Activity {

  ArrayList<Product> products = new ArrayList<Product>();
  BoxAdapter boxAdapter;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // создаем адаптер
    fillData();
    boxAdapter = new BoxAdapter(this, products);

    // настраиваем список
    ListView lvMain = (ListView) findViewById(R.id.lvMain);
    lvMain.setAdapter(boxAdapter);
  }

  // генерируем данные для адаптера
  void fillData() {
    for (int i = 1; i <= 20; i++) {
      products.add(new Product("Product " + i, i * 1000,
          R.drawable.ic_launcher, false));
    }
  }

  // выводим информацию о корзине
  public void showResult(View v) {
    String result = "Товары в корзине:";
    for (Product p : boxAdapter.getBox()) {
      if (p.box)
        result += "\n" + p.name;
    }
    Toast.makeText(this, result, Toast.LENGTH_LONG).show();
  }
}
```

Тут кода совсем мало.

В **onCreate** создаем адаптер и список.

В методе **fillData** генерируем данные для адаптера. В качестве картинки используем стандартную для всех пунктов. В идеале, для каждого товара своя картинка.

Метод **showResult** получает из адаптера список товаров корзины и выводит их наименования. Этот метод вызывается по нажатию кнопки на экране, т.к. прописан в ее свойстве **onClick**.

Все сохраняем и запускаем. Отмечаем товары и жмем кнопку для просмотра содержимого корзины.

Достаточно непростой получился пример из-за чекбокса.

Вполне может быть, что есть другой способ реализации этого примера. Но смысл был в том, чтобы показать создание своего адаптера. Для закрепления темы посмотрите еще этот гугловский пример.

На следующем уроке:

- используем Header и Footer в списках
- разбираемся, как и где используется HeaderViewListAdapter

Урок 55. Header и Footer в списках. HeaderViewListAdapter

В этом уроке:

- используем Header и Footer в списках
- разбираемся, как и где используется HeaderViewListAdapter

Header и **Footer** (далее по тексту HF) – это View, которые могут быть добавлены к списку сверху и снизу. За создание соответствующих View для пунктов списка отвечает уже не адаптер, а программист. Он должен сам создать View и предоставить его списку в методы **addHeader** или **addFooter**.

У этих методов есть **две реализации**. Рассмотрим на примере Header.

1) addHeaderView (View v, Object data, boolean isSelectable)

v – View, которое отобразится, как пункт списка
data – объект, связанный с этим пунктом списка
isSelectable – можно ли будет кликать на пункт или выделять его

2) addHeaderView (View v)

Тут просто идет вызов первого метода с параметрами: **addHeaderView(v, null, true)**;

При использовании HF и адаптера есть нюанс. При присвоении списку **адаптера** (метод setAdapter), список проверяет, были ли уже добавлены **Header** или **Footer**.

Если **нет**, то список использует данный ему адаптер и **запрещает** в дальнейшем добавлять себе HF. Это же написано в хелпе методов addHeader и addFooter – «Call this before calling setAdapter». Т.е. вы должны добавить HF до того, как присвоите адаптер списку.

Если **да**, то список оборачивает полученный адаптер в HeaderViewListAdapter, используя конструктор: HeaderViewListAdapter (ArrayList<ListView.FixedViewInfo> headerViewInfos, ArrayList<ListView.FixedViewInfo> footerViewInfos, ListAdapter adapter), где **headerViewInfos** и **footerViewInfos** – это ранее добавленные к списку HF, а **adapter** – тот адаптер, который мы даем списку. И теперь при работе списка будут использоваться методы адаптера **HeaderViewListAdapter**, т.е. будут использоваться как HF, так и данные из адаптера, который присвоили списку.

Посмотрим на практике.

Создадим проект:

Project name: P0551_HeaderFooter
Build Target: Android 2.3.3
Application name: HeaderFooter
Package name: ru.startandroid.develop.p0551headerfooter
Create Activity: MainActivity

В файл **strings.xml** добавим строки:

```
    <string  name="header_title">Header</string>
    <string  name="footer_title">Footer</string>
    <string  name="button_text">Test</string>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
           android:id="@+id/button1"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onclick"
           android:text="@string/button_text">
     </Button>
     <ListView
           android:id="@+id/lvMain"
           android:layout_width="match_parent"
           android:layout_height="wrap_content">
     </ListView>
</LinearLayout>
```

Кнопка и список.

Создадим Layout-файлы для Header и Footer.

header.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
     <TextView
           android:id="@+id/tvTitle"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="@string/header_title">
     </TextView>
     <TextView
           android:id="@+id/tvText"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text=""
           android:textSize="20sp">
     </TextView>
</LinearLayout>
```

footer.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
     <TextView
          android:id="@+id/tvTitle"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/footer_title">
     </TextView>
     <TextView
          android:id="@+id/tvText"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text=""
          android:textSize="20sp">
     </TextView>
</LinearLayout>
```

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0551headerfooter;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  String[] data = {"one", "two", "three", "four", "five"};
  ListView lvMain;
  ArrayAdapter<String> adapter;

  View header1;
  View header2;

  View footer1;
  View footer2;


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        lvMain = (ListView) findViewById(R.id.lvMain);
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, data);
```

```
        // создаем Header и Footer
        header1 = createHeader("header 1");
        header2 = createHeader("header 2");
        footer1 = createFooter("footer 1");
        footer2 = createFooter("footer 2");

        fillList();

    }

  // формирование списка
    void fillList() {
    }

    // нажатие кнопки
    public void onclick(View v) {
    }

    // создание Header
    View createHeader(String text) {
      View v = getLayoutInflater().inflate(R.layout.header, null);
      ((TextView)v.findViewById(R.id.tvText)).setText(text);
      return v;
    }

    // создание Footer
    View createFooter(String text) {
      View v = getLayoutInflater().inflate(R.layout.footer, null);
      ((TextView)v.findViewById(R.id.tvText)).setText(text);
      return v;
    }
}
```

В **onCreate** подготавливаем адаптер и HF-элементы.

В **fillList** будем заполнять список.

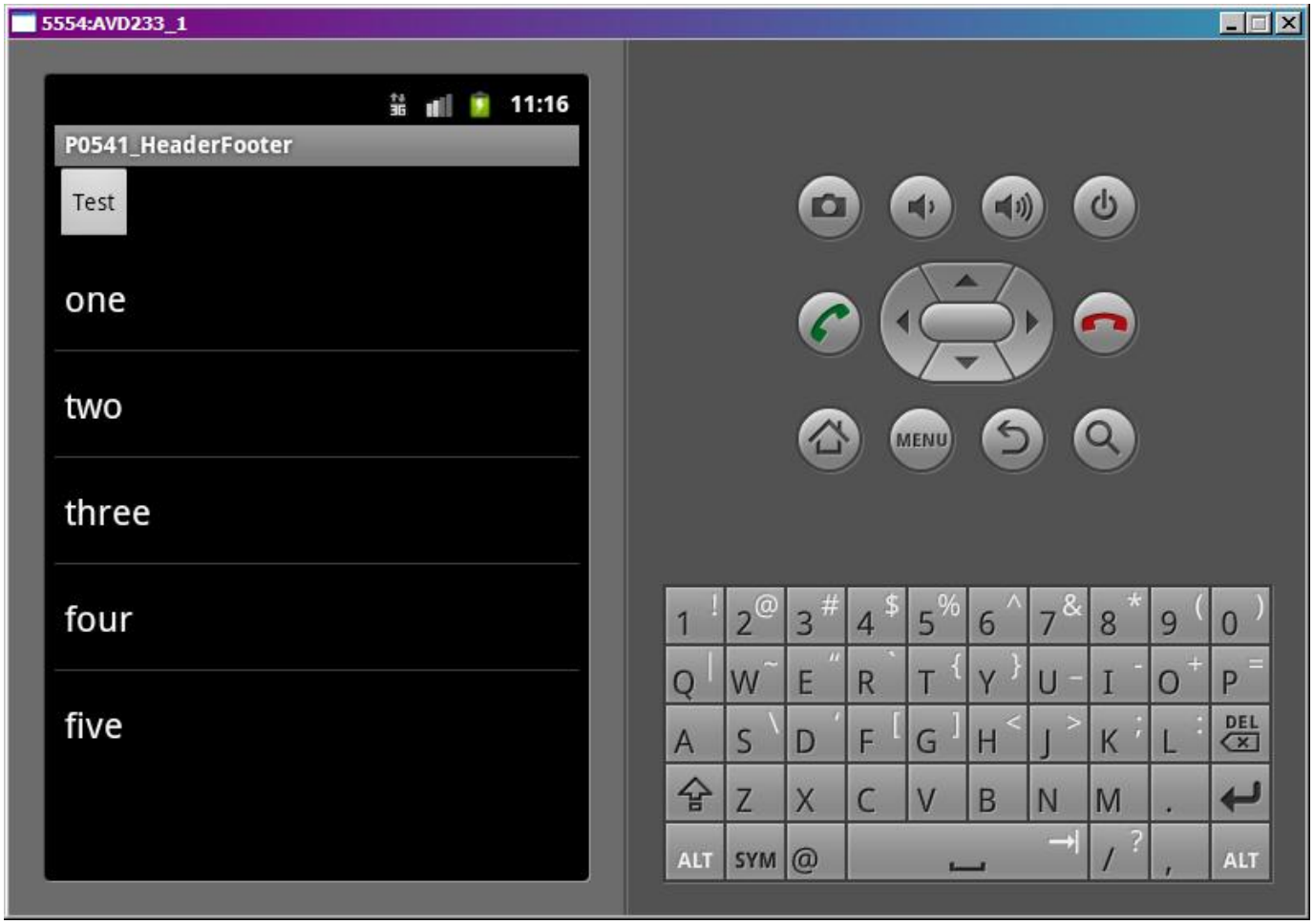
onclick – обработка нажатия кнопки

createHeader и **createFooter** – создание View для HF и заполнение их данными.

Попробуем проверить, что Header реально нельзя добавить после установки адаптера. Реализуем метод **fillList** так:

```
  // формирование списка
    void fillList() {
          try {
          lvMain.setAdapter(adapter);
          lvMain.addHeaderView(header1);
          } catch (Exception ex) {
            Log.e(LOG_TAG, ex.getMessage());
          }
    }
```

Сохраним и запустим.



Пункты списка есть. Header-а нет.

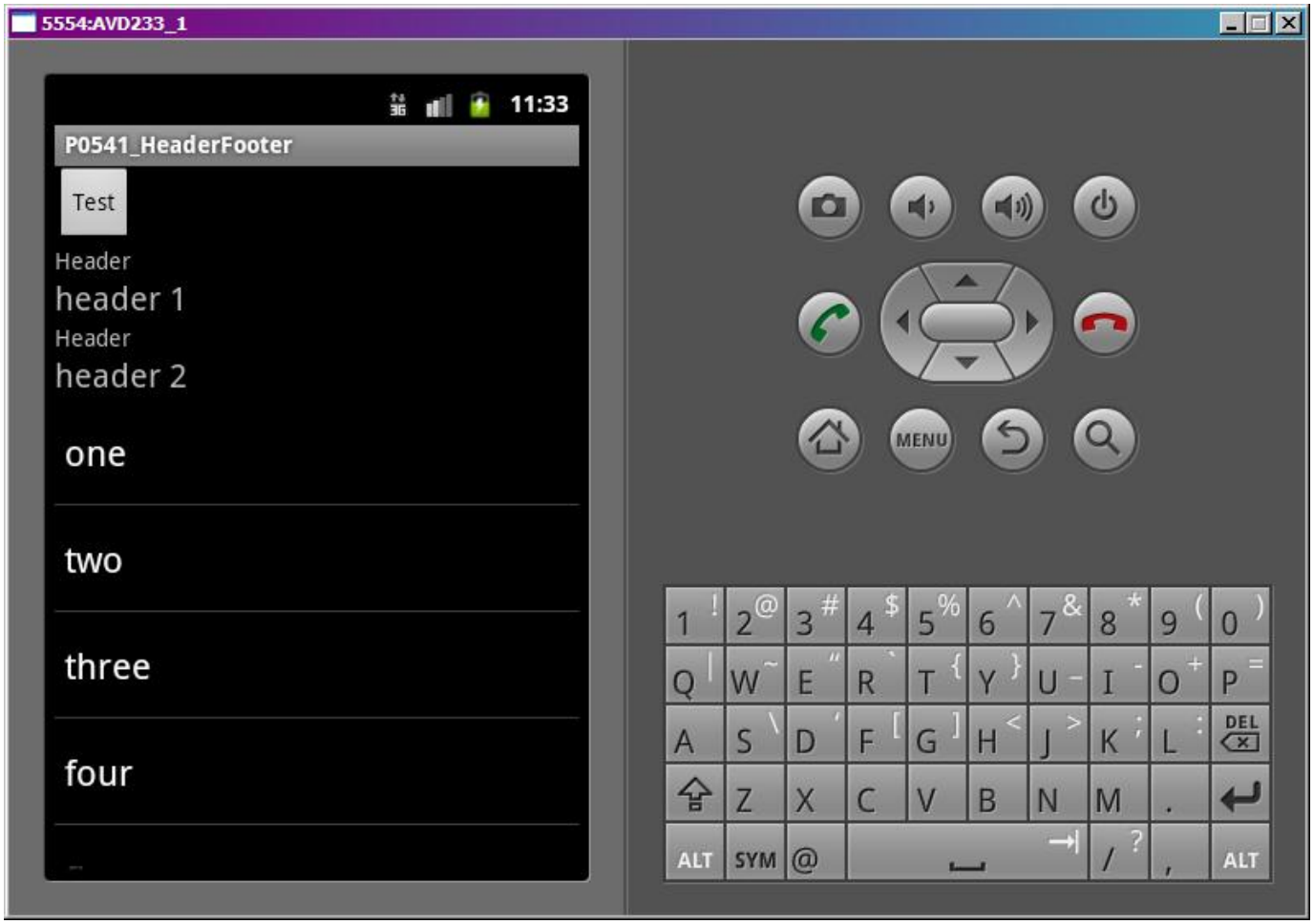
Смотрим лог, видим там ошибку: «*Cannot add header view to list -- setAdapter has already been called*». По русски: *нельзя добавить Header, адаптер уже установлен*.

Давайте сначала добавим HF, а потом присвоим адаптер. Поменяем код **fillList**:

```
// формирование списка
   void fillList() {
     lvMain.addHeaderView(header1);
       lvMain.addHeaderView(header2, "some text for header 2", false);
       lvMain.addFooterView(footer1);
       lvMain.addFooterView(footer2, "some text for footer 2", false);
       lvMain.setAdapter(adapter);
   }
```

Обратите внимание на то, с какими параметрами добавляем **header2** и **footer2**. Мы связываем с ними **String** объекты и с помощью **false** указываем, что выделение на этих пунктах не будет работать.

Все сохраним, запустим приложение.



HF появились. Убедитесь, что клик на header2 и footer2 ни к чему не приводит. И если покрутить колесо мыши, то выделение на этих пунктах не останавливается.

Кстати, если вы делаете свой адаптер, то этого эффекта можно добиться, выполнив свою реализацию метода isEnabled.

Попробуем удалить HF. Метод **onclick** перепишем так:

```
// нажатие кнопки
public void onclick(View v) {
  lvMain.removeHeaderView(header2);
  lvMain.removeFooterView(footer2);
}
```

Запустим, нажмем кнопку **Test** и видим, что исчезли те HF, которые мы удалили - **header2** и **footer2**.

Теперь посмотрим, как работает **HeaderViewListAdapter**, разберемся, что там за вложенный адаптер у него и как до него добраться.

Перепишем метод **onclick**:

```
// нажатие кнопки
public void onclick(View v) {
  Object obj;
  HeaderViewListAdapter hvlAdapter = (HeaderViewListAdapter) lvMain.getAdapter();
```

```
    obj = hvlAdapter.getItem(1);
    Log.d(LOG_TAG, "hvlAdapter.getItem(1) = " + obj.toString());
    obj = hvlAdapter.getItem(4);
    Log.d(LOG_TAG, "hvlAdapter.getItem(4) = " + obj.toString());

    ArrayAdapter<String> alAdapter = (ArrayAdapter<String>) hvlAdapter.getWrappedAdapter();
    obj = alAdapter.getItem(1);
    Log.d(LOG_TAG, "alAdapter.getItem(1) = " + obj.toString());
    obj = alAdapter.getItem(4);
    Log.d(LOG_TAG, "alAdapter.getItem(4) = " + obj.toString());
  }
```

Сначала мы с помощью метода **getAdapter** получаем адаптер, который использует **ListView**. Т.к. мы **добавляли HF** к списку, то он использует адаптер **HeaderViewListAdapter**. Попробуем получить данные по второму и пятому пункту этого адаптера, вызвав **getItem** и выведем значение в лог.

Далее мы от **HeaderViewListAdapter** получаем его **вложенный** адаптер. Это тот адаптер, который мы давали на вход списку в методе setAdapter. Но т.к. были использованы HF, то список создал HeaderViewListAdapter в качестве основного адаптера и отдал ему наш в качестве вложенного и теперь мы его получаем методом **getWrappedAdapter**. Попробуем от нашего адаптера получить данные также по второму и пятому пункту и вывести в лог.

Тут важно понимать, что если бы мы не добавляли HF к списку, то список не стал бы заморачиваться с созданием HeaderViewListAdapter и метод lvMain.getAdapter сразу вернул бы **ArrayAdapter**.

Сохраняем, запускаем, жмем кнопку **Test** и смотрим лог.

hvlAdapter.getItem(1) = some text for header 2
hvlAdapter.getItem(4) = three
alAdapter.getItem(1) = two
alAdapter.getItem(4) = five

Адаптеры показали разные данные. Это произошло потому, что **hvlAdapter объединяет HF** с данными из **alAdapter** и работает с ними, как с **одним набором** данных. А **alAdapter** учитывает только **свои** данные, которые мы ему давали (массив строк data[]).

Для **hvlAdapter** второй пункт – это header2. При добавлении Header-а мы присвоили ему объект String, его он сейчас и вывел в лог. Пятый пункт – three.

Для **alAdapter** второй пункт – two, а пятый - five. Он вообще не в курсе ни про какие HF, он работает только со своими данными.

Вот как-то так организована работа с Header и Footer элементами. А про HeaderViewListAdapter в хелпе написано, что разрабам скорее всего не придется использовать этот класс напрямую в своем коде. Наверно так оно и есть. Этот класс используется списком, чтобы работать с данными адаптера и добавленными Header и Footer, как с одним набором данных.

Для тех, кто расстроился, что нельзя добавлять HF после присвоения адаптера списку, внесу сразу важную поправку. В хелпе действительно так написано. Но на деле это относится только к Header. А Footer добавляется без проблем. И я не знаю, баг это или фича)

На следующем уроке:

- используем Spinner

Урок 56. Spinner – выпадающий список

В этом уроке:

- используем Spinner

Многовато уроков получилось про адаптеры и списки. Я изначально хотел буквально пару примеров показать, но чето затянуло и расписал чуть ли не все существующие адаптеры) Напоследок сделаем пару полезных примеров, где еще используются адаптеры. В этом небольшом уроке рассмотрим Spinner.

Spinner – это выпадающий список, позволяющий выбрать одно значение. Он позволяет сэкономить место на экране. Я думаю, все встречали его не только в Android, но и в других языках программирования.

Сделаем простой пример.

Создадим проект:

Project name: P0561_Spinner
Build Target: Android 2.3.3
Application name: Spinner
Package name: ru.startandroid.develop.p0561spinner
Create Activity: MainActivity

В экран **main.xml** поместим Spinner:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
    <Spinner
          android:id="@+id/spinner"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
    </Spinner>
</LinearLayout>
```

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0561spinner;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

public class MainActivity extends Activity {

  String[] data = {"one", "two", "three", "four", "five"};

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // адаптер
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, data);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
```

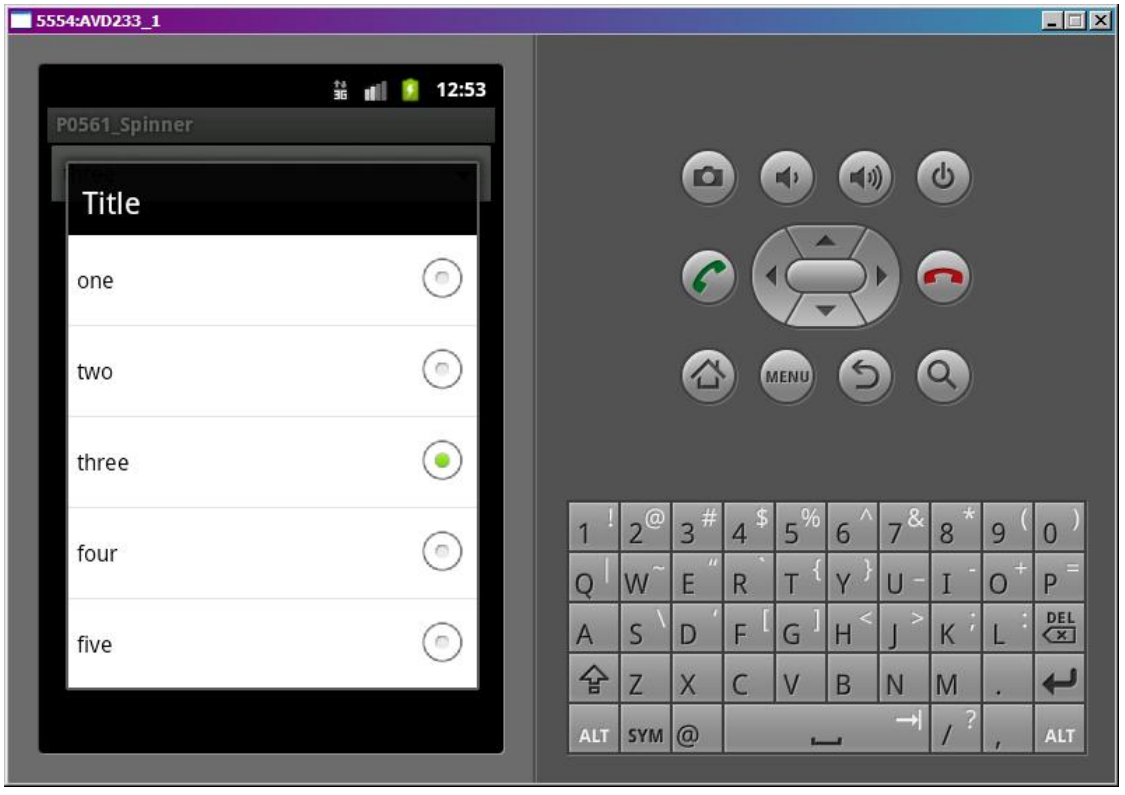
```
        Spinner spinner = (Spinner) findViewById(R.id.spinner);
        spinner.setAdapter(adapter);
        // заголовок
        spinner.setPrompt("Title");
        // выделяем элемент
        spinner.setSelection(2);
        // устанавливаем обработчик нажатия
        spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
      @Override
      public void onItemSelected(AdapterView<?> parent, View view,
          int position, long id) {
        // показываем позиция нажатого элемента
        Toast.makeText(getBaseContext(), "Position = " + position, Toast.LENGTH_SHORT).show();
      }
      @Override
      public void onNothingSelected(AdapterView<?> arg0) {
      }
    });
    }
}
```

Код несложен. Создаем адаптер, используем **simple_spinner_item** в качестве layout для отображения Spinner на экране. А методом setDropDownViewResource указываем какой layout использовать для прорисовки пунктов выпадающего списка.

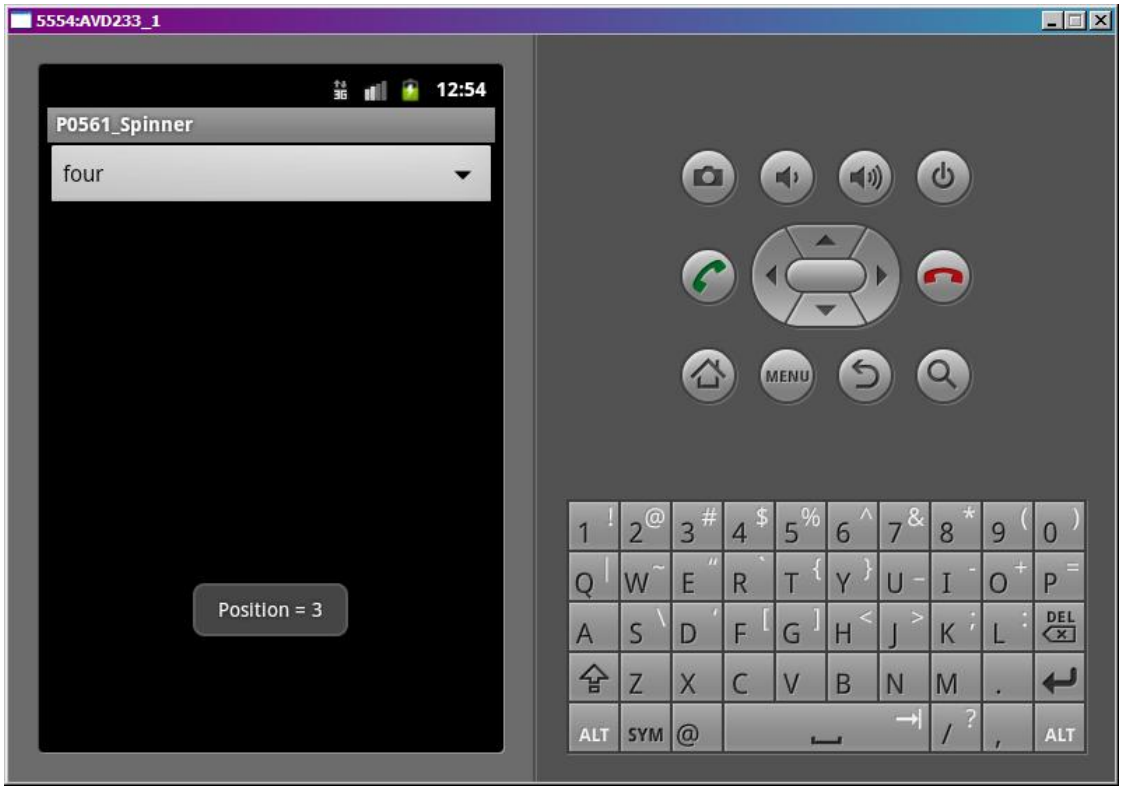
Метод setPrompt устанавливает текст заголовка выпадающего списка, а setSelection – элемент, который мы хотим выделить. Оба метода, разумеется, необязательны. Я их использовал для демонстрации.

Обработчик выбора элемента из списка присваивается методом setOnItemSelectedListener. Будем выводить на экран позицию выбранного элемента.

Все сохраним и запустим. Нажмем на Spinner.



Виден заголовок Title и выделен элемент с позицией 2. Выберем пункт four.



Обработчик показал, что выделили пункт с позицией 3.

Чтобы определить, какой элемент выделен сейчас, используйте spinner.getSelectedItemPosition(). Чтобы получить текст этого элемента можно сразу выполнять spinner.getSelectedItem().toString().

На следующем уроке:

- используем GridView

Урок 57. GridView и его атрибуты

В этом уроке:

- используем GridView

GridView – еще один из компонентов, использующих адаптеры. Он выводит элементы в виде сетки/матрицы/таблицы, нужное подчеркнуть)

Сделаем простой пример. И рассмотрим интересные атрибуты этого компонента.

Создадим проект:

Project name: P0571_GridView
Build Target: Android 2.3.3
Application name: GridView
Package name: ru.startandroid.develop.p0571gridview
Create Activity: MainActivity

В экран **main.xml** поместим GridView:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
     <GridView
          android:id="@+id/gvMain"
          android:layout_width="match_parent"
          android:layout_height="wrap_content">
     </GridView>
</LinearLayout>
```

Создадим в любой папке res/drawable-* файл **rect.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<shape  xmlns:android="http://schemas.android.com/apk/res/android"  android:shape="rectangle">
    <solid  android:color="#99000099"></solid>
</shape>
```

Это просто прямоугольник, залитый синим цветом. Я буду использовать его как фон. Я эту тему еще не рассматривал в уроках, тут можно почитать подробнее .

Создадим свой layout для адаптера – **item.xml**

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:background="@drawable/rect"
      android:orientation="vertical">
     <TextView
          android:id="@+id/tvText"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:gravity="center_vertical"
          android:minHeight="40dp"
          android:textSize="20sp"
          android:text="">
     </TextView>
</LinearLayout>
```

LinearLayout с фоном drawable/rect, который мы создали ранее. И TextView.

Код **MainActivity.java**:

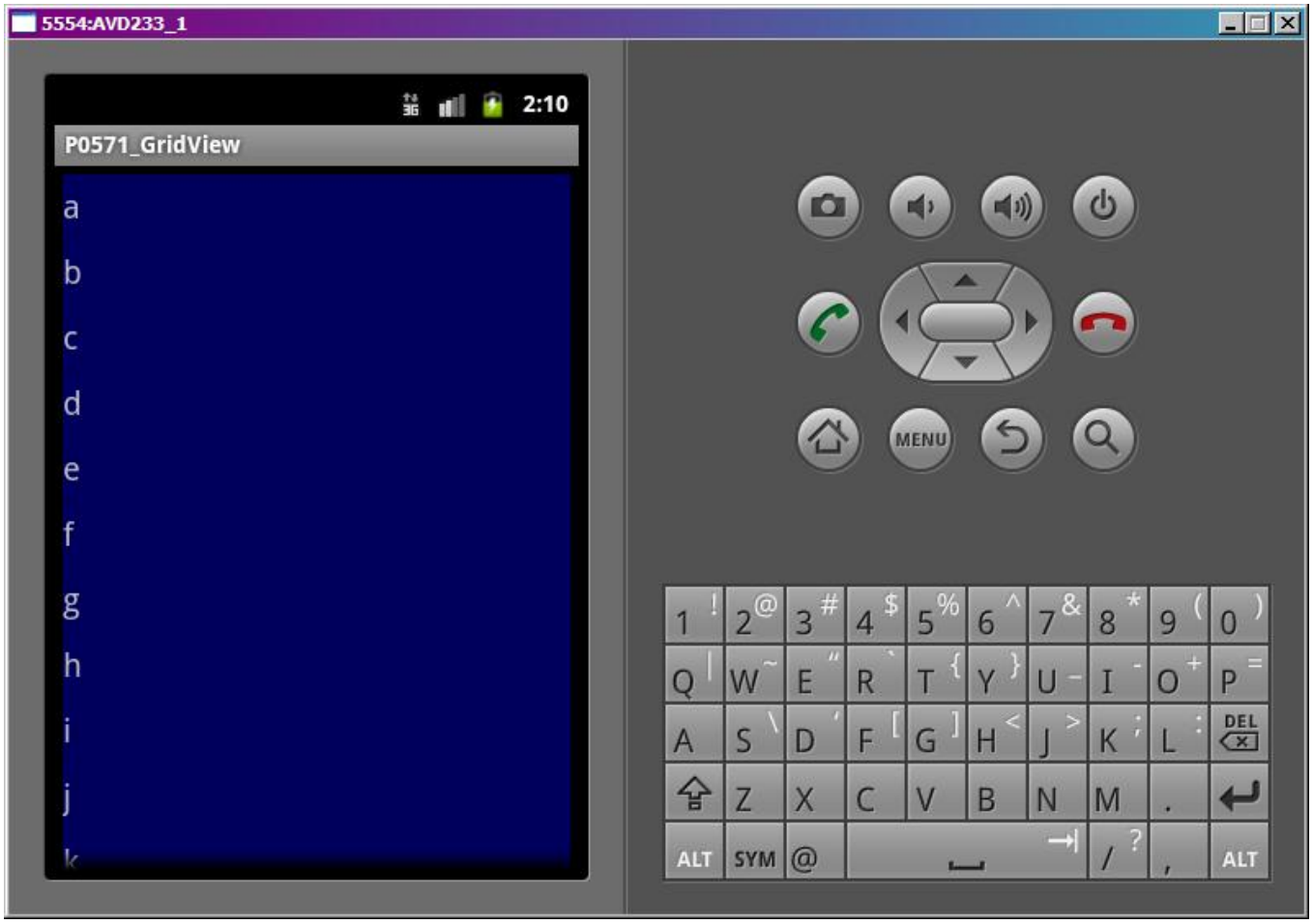
```
package ru.startandroid.develop.p0571gridview;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.GridView;

public class MainActivity extends Activity {

  String[] data = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k"};

  GridView gvMain;
  ArrayAdapter<String> adapter;


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        adapter = new ArrayAdapter<String>(this, R.layout.item, R.id.tvText, data);
        gvMain = (GridView) findViewById(R.id.gvMain);
        gvMain.setAdapter(adapter);
        adjustGridView();
    }


  private void adjustGridView() {
  }
}
```

Кода немного. Определяем GridView и создаем адаптер. В качестве layout для адаптера используем созданный item.xml, а tvText – это элемент, в который адаптер будет вставлять текст. Метод adjustGridView пока пустой, в нем будем кодить настройки Grid-а.

Давайте смотреть, какие для GridView есть атрибуты.

numColumns и columnWidth

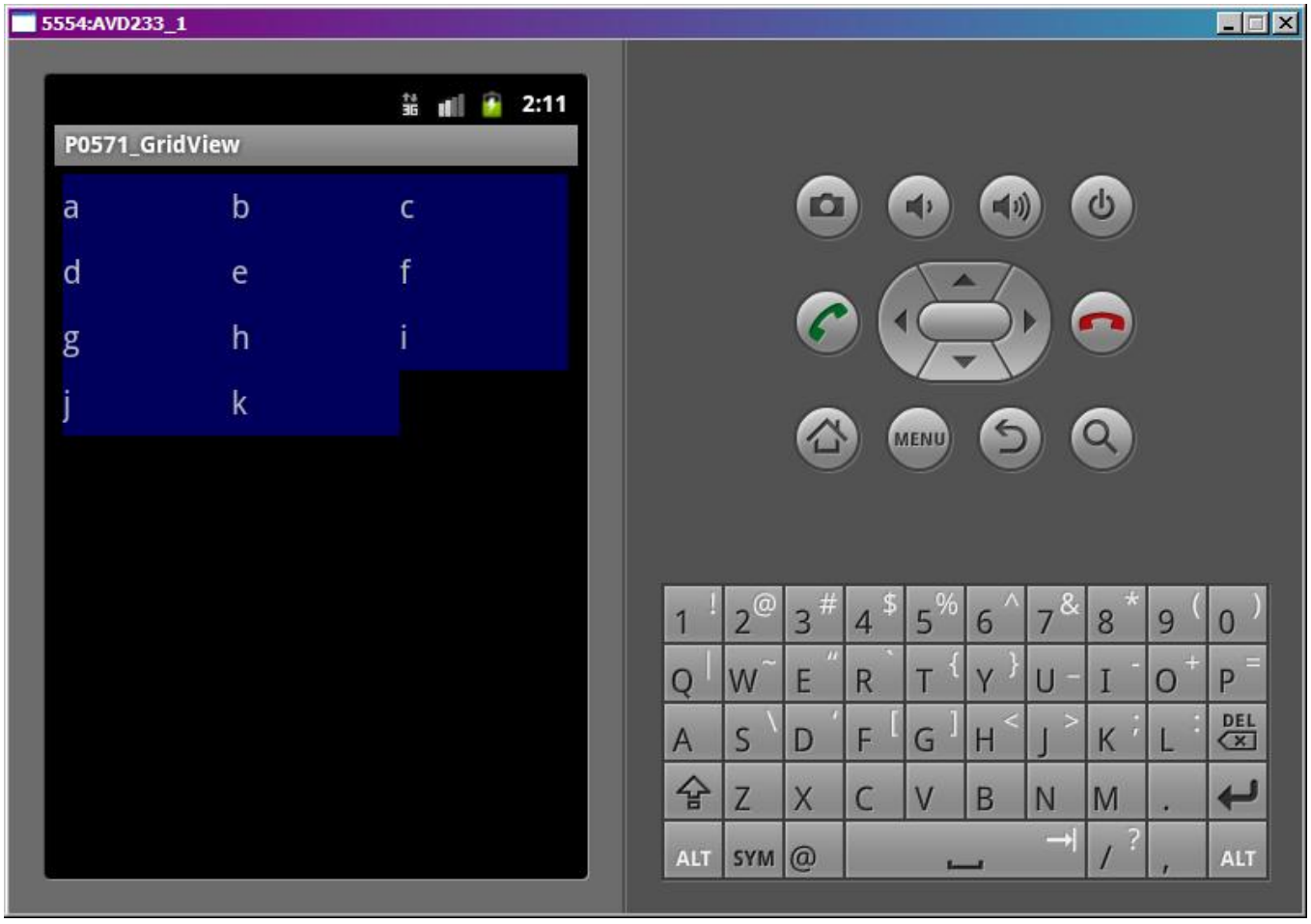
numColumns – кол-во столбцов в сетке. Если его не задавать, то столбец будет по умолчанию один. Запустим приложение и убедимся.



Давайте поменяем это свойство - укажем, например 3. Сделаем это в пустом пока что методе adjustGridView

```
private void adjustGridView() {
  gvMain.setNumColumns(3);
}
```

Сохраним и запустим.



Все верно, получилось три столбца.

Это свойство также может иметь значение AUTO_FIT. В этом случае проверяется значение поля атрибута **columnWidth** (ширина столбца).

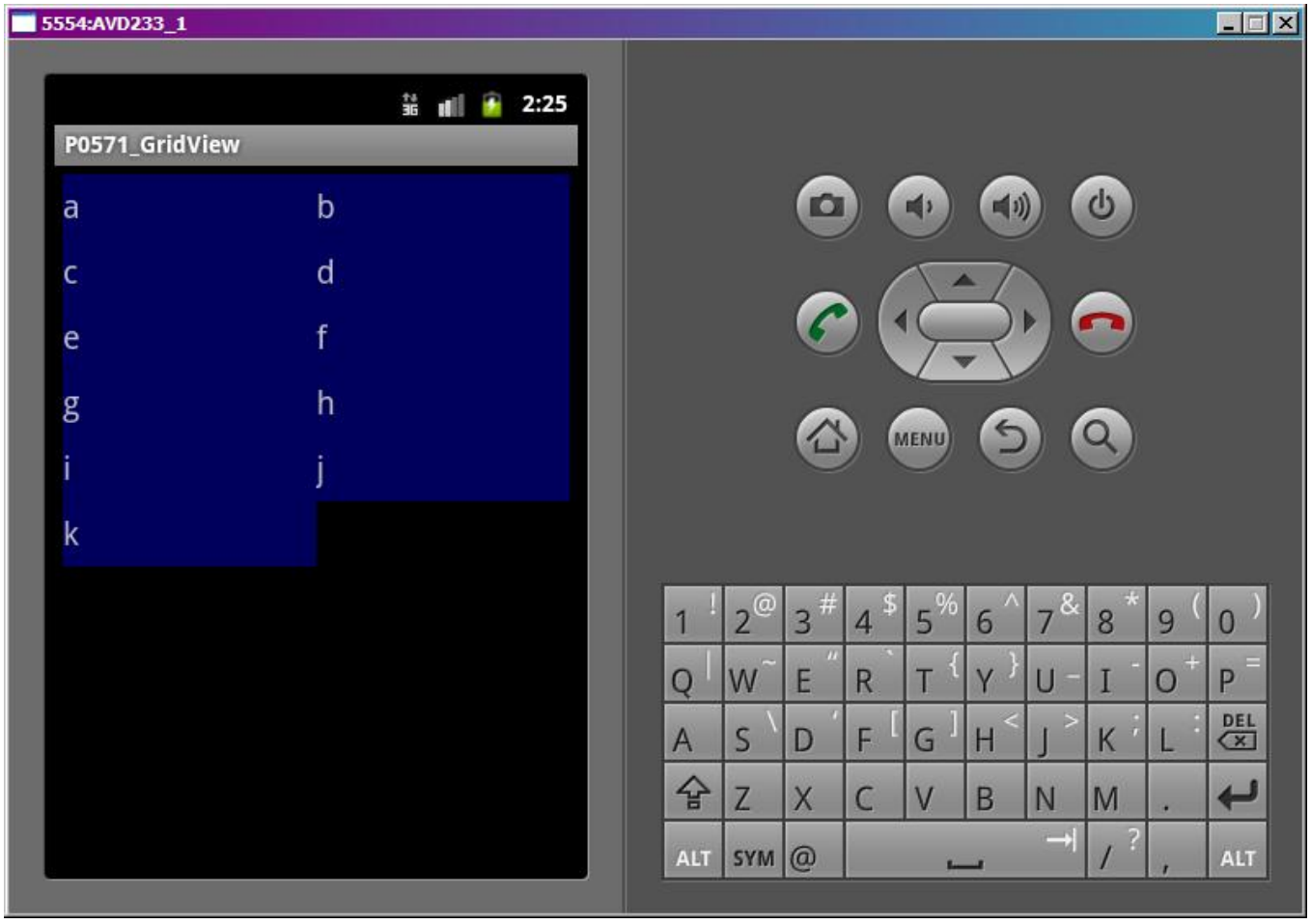
- если ширина столбца явно указана, то кол-во столбцов рассчитывается исходя из ширины, доступной GridView, и ширины столбцов.

- иначе, кол-во столбцов считается равным 2

Проверим. Укажем кол-во столбцов = **AUTO_FIT**, а ширину столбцов задавать пока не будем.

```
private void adjustGridView() {
  gvMain.setNumColumns(GridView.AUTO_FIT);
}
```

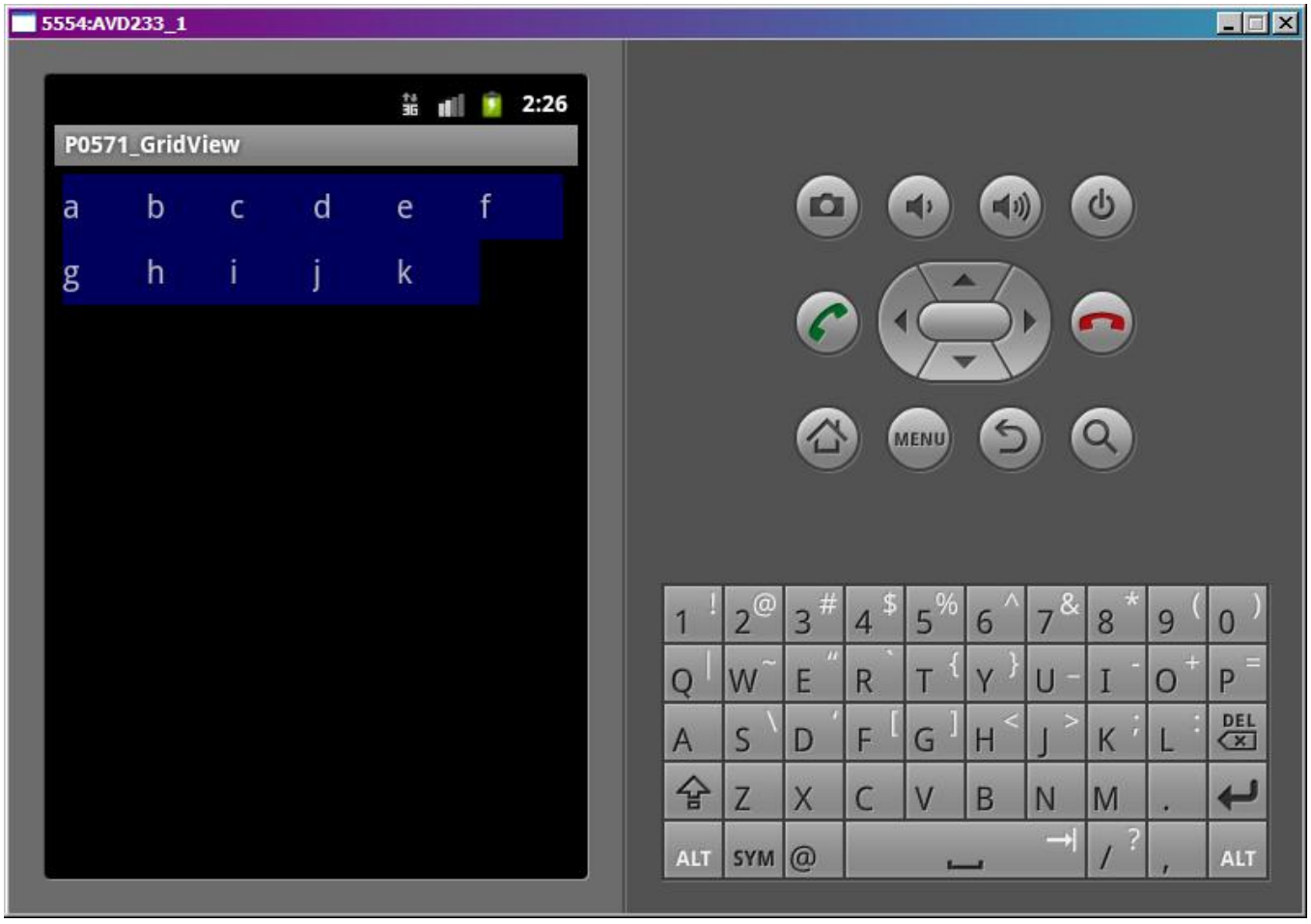
Запускаем, видим два столбца



Теперь укажем явно ширину столбцов, пусть будет 50.

```
private void adjustGridView() {
  gvMain.setNumColumns(GridView.AUTO_FIT);
  gvMain.setColumnWidth(50);
}
```

Теперь кол-во столбцов рассчитывается исходя из их ширины.



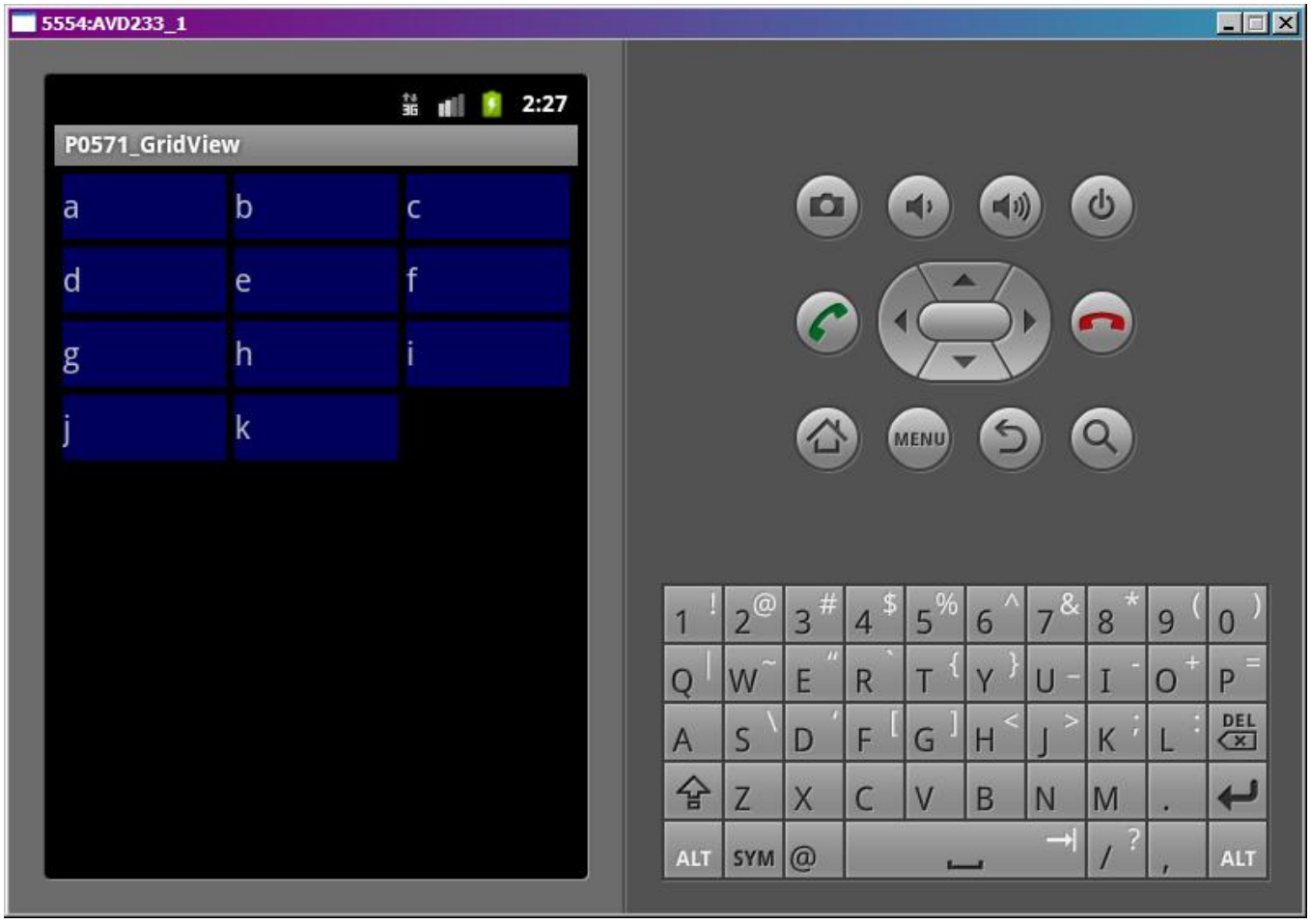
Видно, что в экран влезло 6 столбцов. Вы можете поизменять параметр ширины столбцов и убедиться, что их кол-во будет меняться.

horizontalSpacing, verticalSpacing

Это горизонтальный и вертикальный отступы между ячейками. Пусть будет 5.

```
  private void adjustGridView() {
    gvMain.setNumColumns(GridView.AUTO_FIT);
    gvMain.setColumnWidth(80);
    gvMain.setVerticalSpacing(5);
    gvMain.setHorizontalSpacing(5);
  }
```

Запустим приложение.



Между ячейками появилось расстояние.

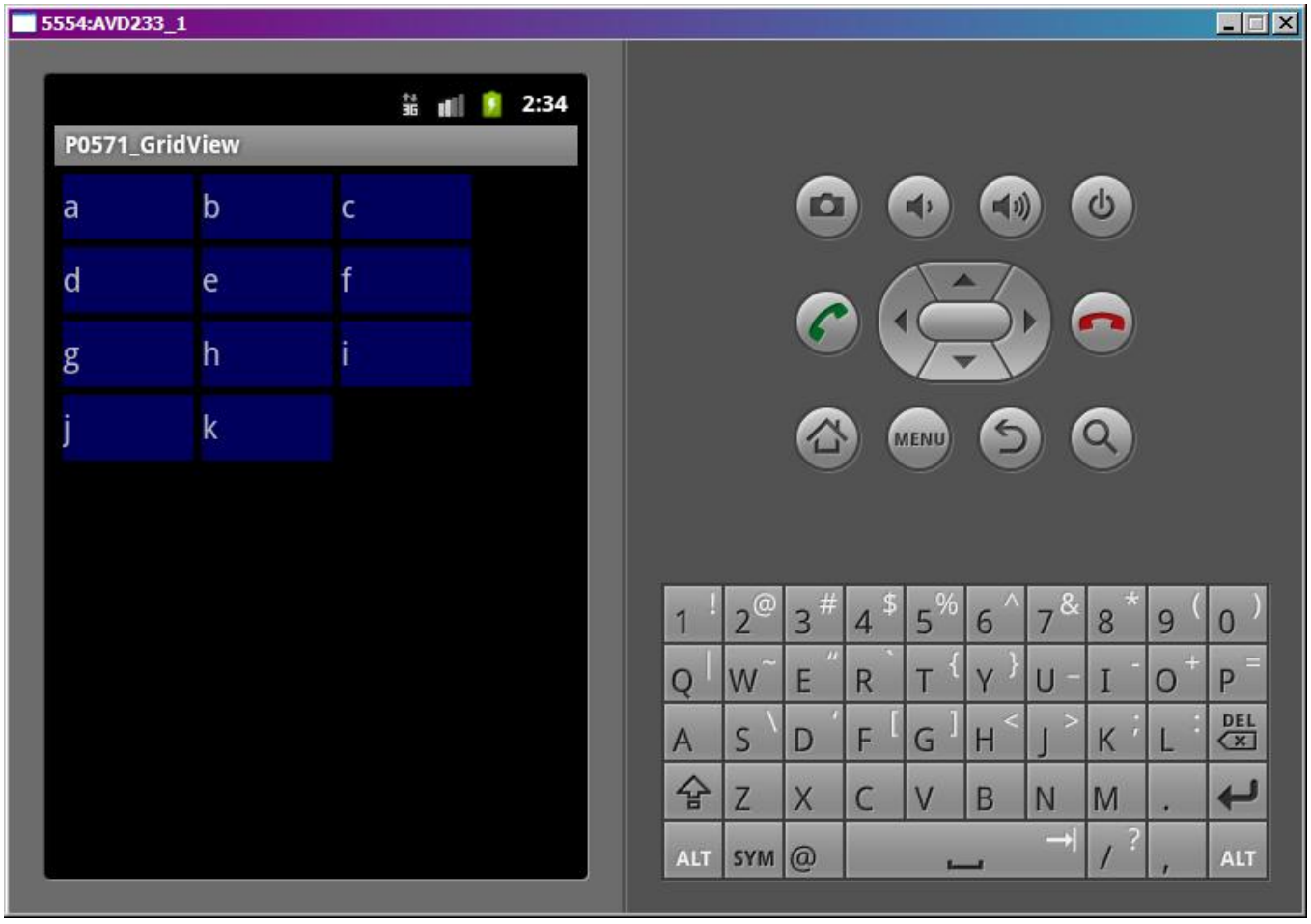
stretchMode

Этот параметр определяет, как будет использовано свободное пространство, если оно есть. Используется в случае, когда вы указываете ширину столбца и кол-во ставите в режим AUTO_FIT. Изменим наш метод, добавим туда настройку stretch-параметра.

```
  private void adjustGridView() {
    gvMain.setNumColumns(GridView.AUTO_FIT);
    gvMain.setColumnWidth(80);
    gvMain.setVerticalSpacing(5);
    gvMain.setHorizontalSpacing(5);
    gvMain.setStretchMode(GridView.NO_STRETCH);
  }
```

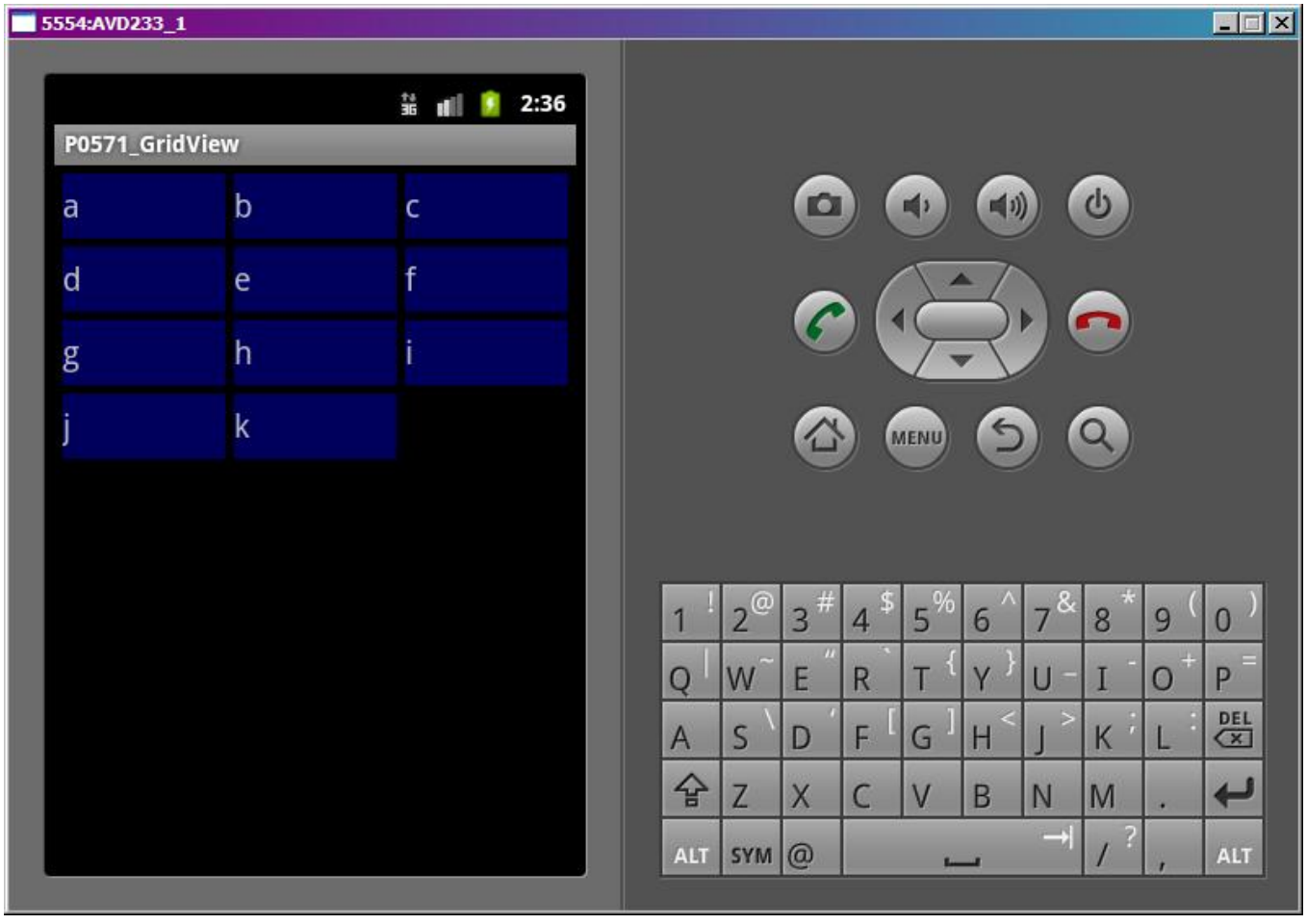
stretchMode может принимать **4** значения:

NO_STRETCH – свободное пространство не используется



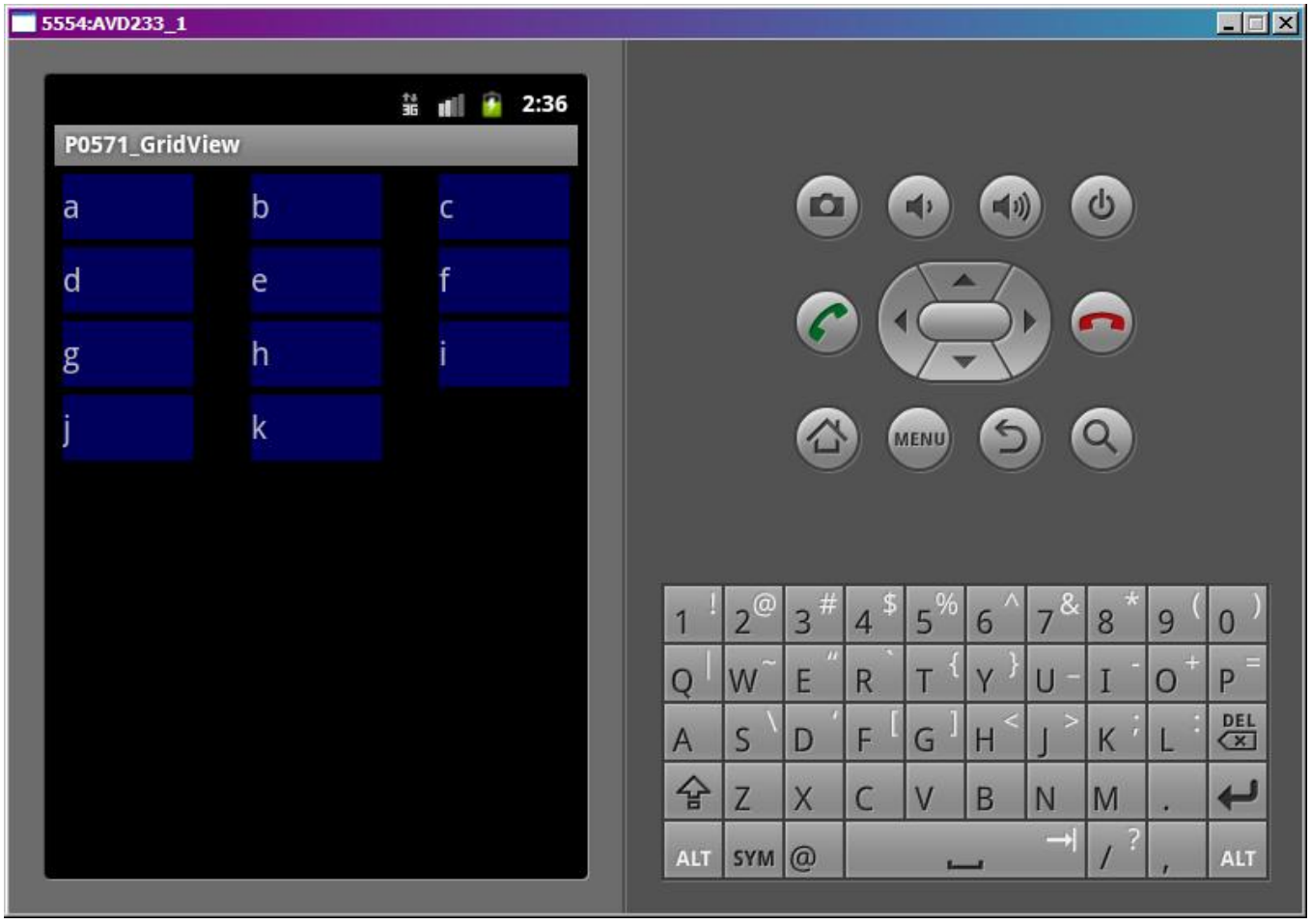
Столбцы выровнены по левому краю. Все свободное пространство справа.

STRETCH_COLUMN_WIDTH – свободное пространство используется столбцами, это режим по умолчанию



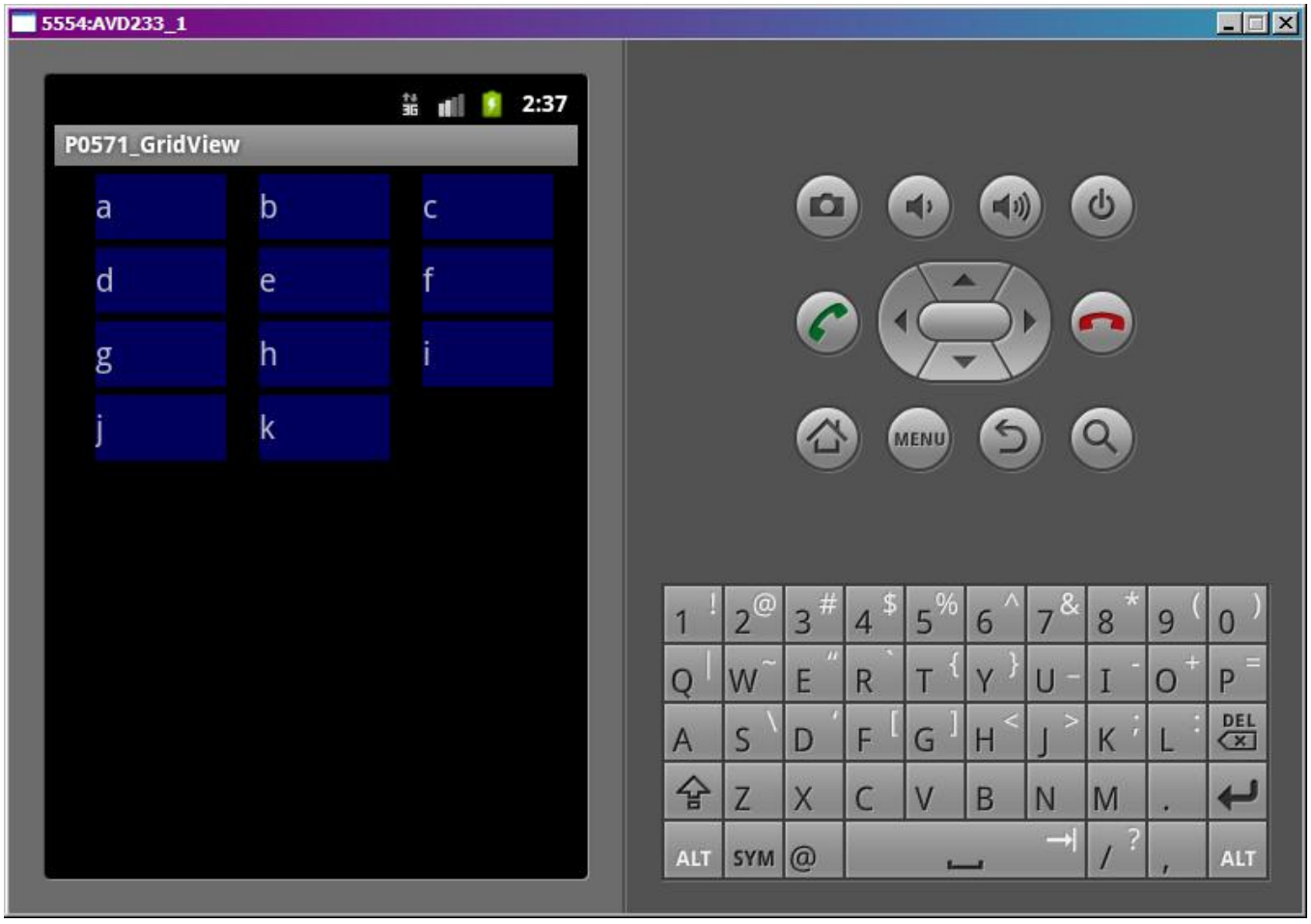
Столбцы растянуты по ширине. Она уже может не соответствовать той, что указана в setColumnWidth.

STRETCH_SPACING – свободное пространство равномерно распределяется между столбцами



Ширина столбцов неизменна. Увеличены интервалы между ними.

STRETCH_SPACING_UNIFORM – свободное пространство равномерно распределяется не только между столбцами, но и справа и слева



Ширина столбцов неизменна. Увеличены интервалы между ними и с боков.

Разумеется, все эти параметры можно задавать не только программно, но и через атрибуты в layout-файлах. Вместо ArrayAdapter можно использовать любой другой. Можно прикрутить обработчик setOnItemClickListener и получать позицию или id нажатого элемента. Все как в обычных списках.

Есть также хороший гугловский пример по этой теме:

http://developer.android.com/resources/tutorials/views/hello-gridview.html

На следующем уроке:

- используем TimePickerDialog

Урок 58. Диалоги. TimePickerDialog

В этом уроке:

- используем TimePickerDialog

Начнем рассматривать новую тему – **диалоги**. Диалог – небольшое всплывающее окно, которое чаще всего используется для подтверждения каких-либо операций или ввода небольшого количества данных.

Мы рассмотрим стандартные диалоги, предоставляемые системой, и научимся делать свои. На этом уроке рассмотрим TimePickerDialog. Это диалог, который позволяет указать время.

Сделаем простое приложение. На экране будет TextView, на него можно будет нажать и появится диалог для выбора времени. После выбора оно отобразится в TextView.

Создадим проект:

Project name: P0581_TimePickerDialog
Build Target: Android 2.3.3
Application name: TimePickerDialog
Package name: ru.startandroid.develop.p0581timepickerdialog
Create Activity: MainActivity

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:id="@+id/tvTime"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:clickable="true"
          android:onClick="onclick"
          android:text="@string/hello"
          android:textSize="22sp">
     </TextView>
</LinearLayout>
```

На экране **TextView**, для которого включим **clickable** и назначим метод обработки **onclick**.

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0581timepickerdialog;

import android.app.Activity;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.app.TimePickerDialog.OnTimeSetListener;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.TimePicker;

public class MainActivity extends Activity {

  int DIALOG_TIME = 1;
  int myHour = 14;
  int myMinute = 35;
  TextView tvTime;

    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvTime = (TextView) findViewById(R.id.tvTime);
    }

    public void onclick(View view) {
      showDialog(DIALOG_TIME);
    }

    protected Dialog onCreateDialog(int id) {
      if (id == DIALOG_TIME) {
        TimePickerDialog tpd = new TimePickerDialog(this, myCallBack, myHour, myMinute, true);
        return tpd;
      }
      return super.onCreateDialog(id);
    }

    OnTimeSetListener myCallBack = new OnTimeSetListener() {
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
      myHour = hourOfDay;
      myMinute = minute;
      tvTime.setText("Time is " + myHour + " hours " + myMinute + " minutes");
    }
  };
}
```

В **onCreate** находим TextView.

В **onClick** вызываем метод showDialog и передаем ему ID диалога. Этот метод (showDialog) создает диалог с помощью отдельного метода и показывает его. ID используется для указания, какой именно диалог создавать и показывать.

Метод **onCreateDialog** – это и есть отдельный метод, который вызывается в showDialog для создания диалога. В этом методе мы смотрим, какой ID пришел на вход, создаем соответствующий диалог и возвращаем его.

В нашем случае мы создаем TimePickerDialog, используя конструктор:

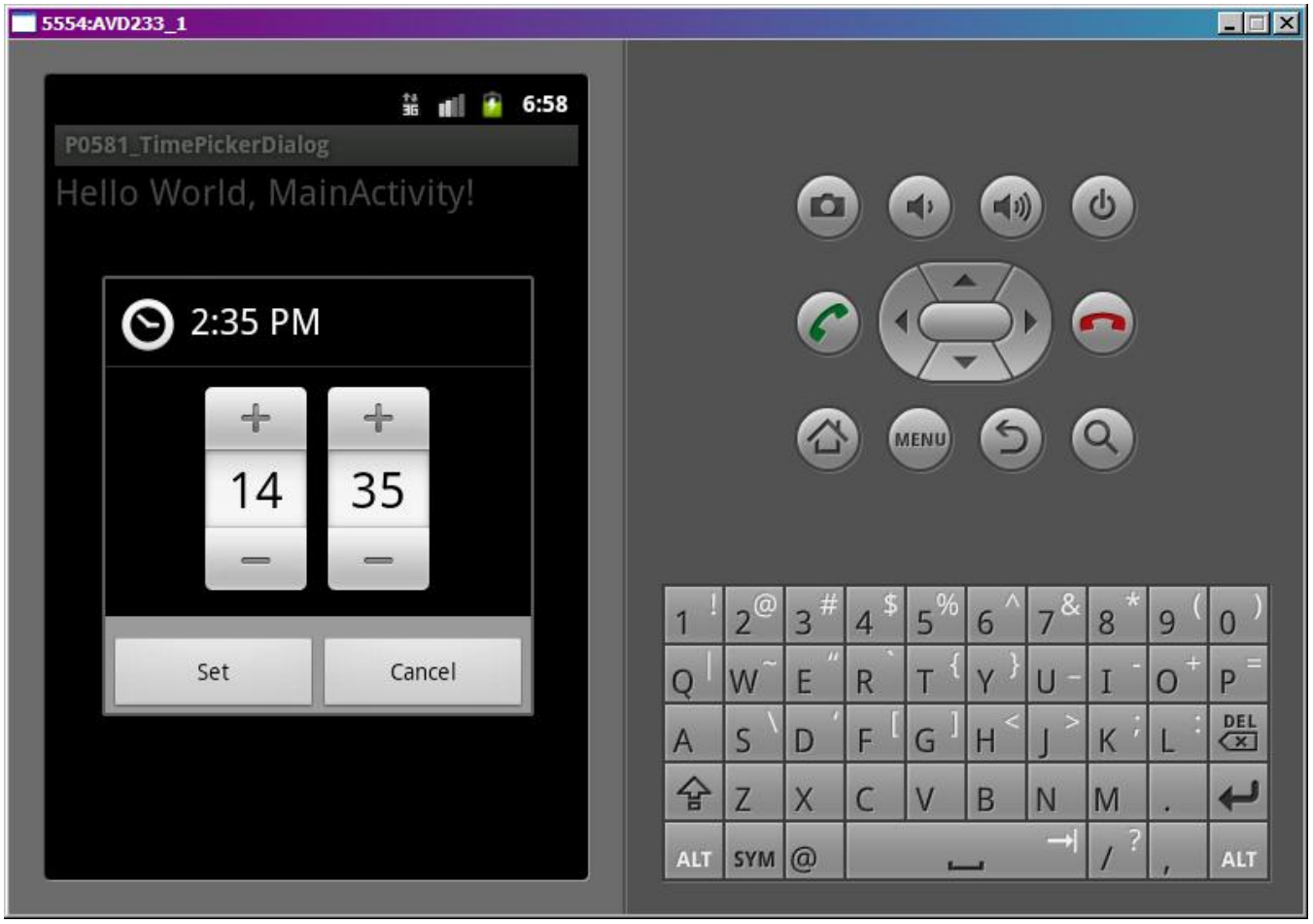
TimePickerDialog (Context context, TimePickerDialog.OnTimeSetListener callBack, int hourOfDay, int minute, boolean is24HourView), где

context – контекст
callBack – это обработчик с интерфйесом TimePickerDialog.OnTimeSetListener, метод которого срабатывает при нажатии кнопки OK на диалоге
hourOfDay – час, который покажет диалог
minute – минута, которую покажет диалог
is24HourView – формат времени 24 часа (иначе AM/PM)

myCallBack – объект, реализующий интерфейс TimePickerDialog.OnTimeSetListener. У него только один метод – onTimeSet, который предоставляет нам TimePicker из диалога, и час и минуту, которые он показывает. Т.е. то, что мы ввели в диалоге.

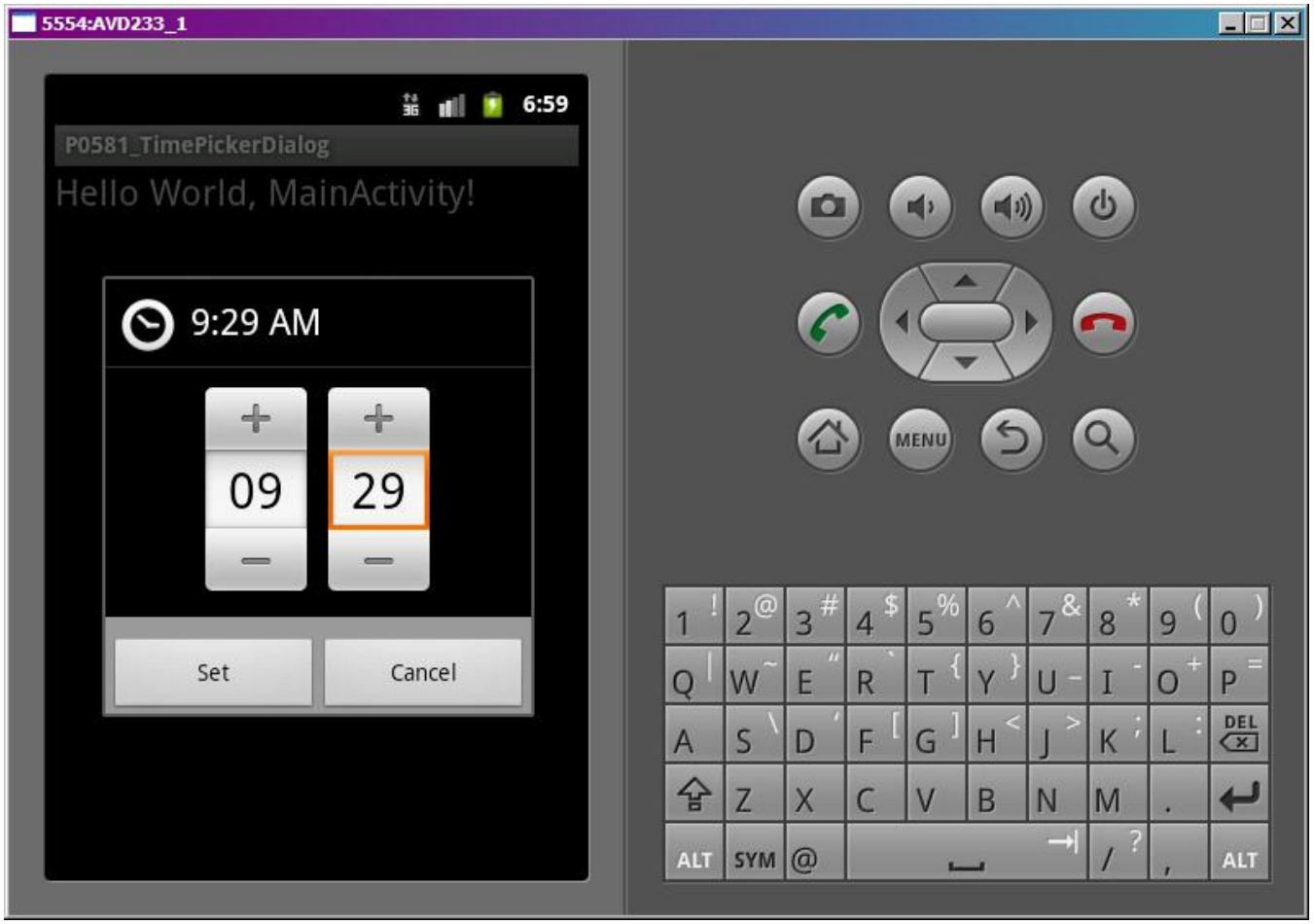
Эти данные мы пишем в tvTime.

Все сохраним и запустим. Нажмем на TextView. Появился диалог для ввода времени.

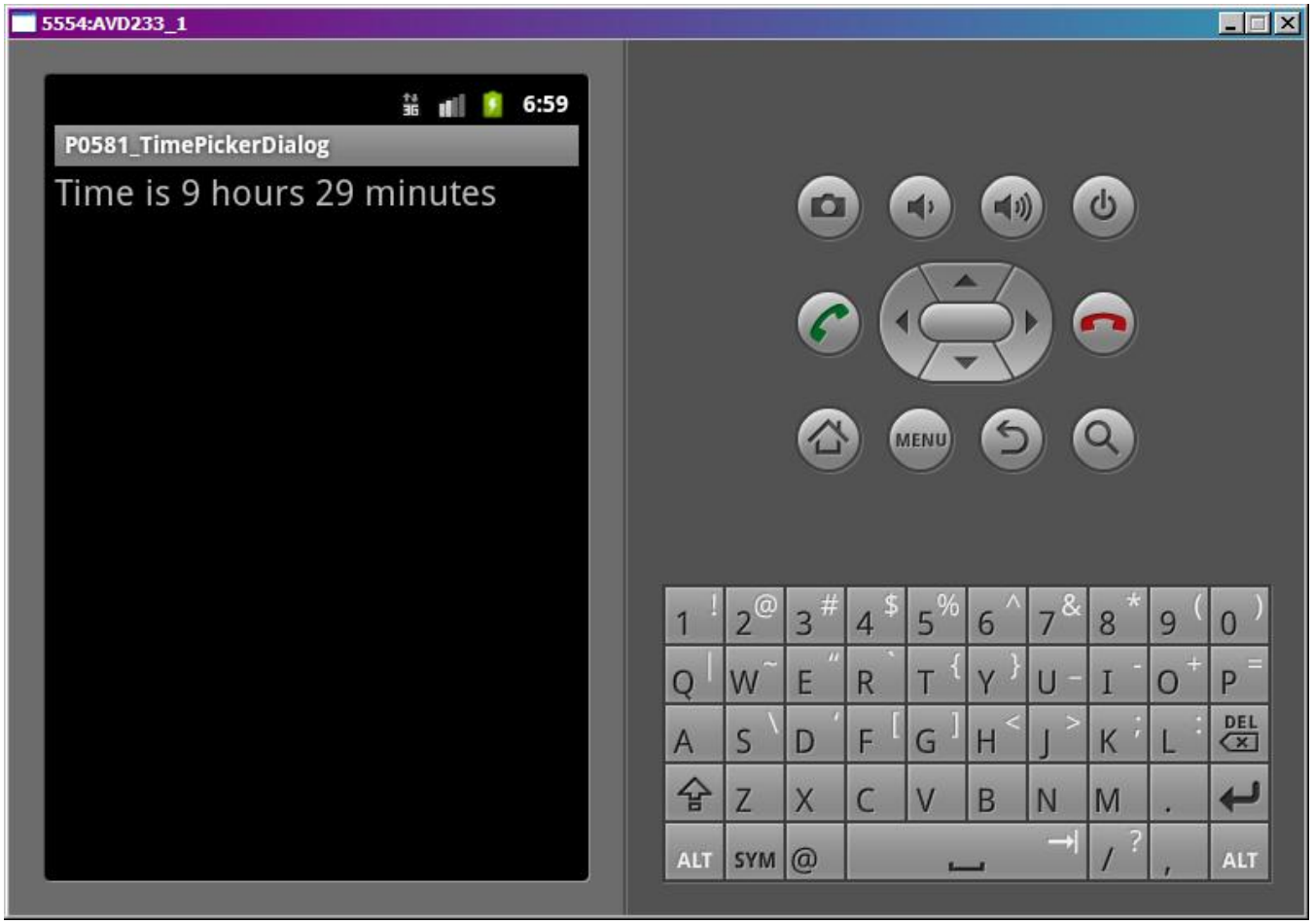


Сейчас он показывает 14:35, т.к. мы при создании передали ему значения myHour и myMinute.

Поменяем значения


5554:AVD233_1
6:59
P0581_TimePickerDialog
Hello World, MainActivity!
9:29 AM
09
29
Set
Cancel
MENU

и нажмем Set.


5554:AVD233_1
6:59
P0581_TimePickerDialog
Time is 9 hours 29 minutes
MENU

Текст показал новые значения.

Диалог также можно закрыть кнопкой **Cancel** или кнопкой **Back** (назад) на эмуляторе. В этом случае метод обработчика не сработает и текст не обновится.

Это самый простейший диалог. И тут, разумеется, рассмотрены далеко не все возможности. Здесь пока важно понять схему взаимодействия showDialog и onCreateDialog, и наличие обработчика результатов диалога.

Также здесь использовался новый для нас компонент TimePicker. Он несложен и просто позволяет удобно задавать или отображать время.

На следующем уроке сделаем похожий пример, но с датой, для закрепления.

На следующем уроке:

- используем DatePickerDialog

Урок 59. Диалоги. DatePickerDialog

В этом уроке:

- используем DatePickerDialog

Урок будет аналогичен прошлому, но теперь в диалоге мы будем задавать дату, а не время. Такой диалог называется DatePickerDialog.

Сделаем простое приложение. На экране будет TextView, на него можно будет нажать и появится диалог для выбора даты. После выбора она отобразится в TextView.

Создадим проект:

Project name: P0591_DatePickerDialog
Build Target: Android 2.3.3
Application name: DatePickerDialog
Package name: ru.startandroid.develop.p0591datepickerdialog
Create Activity: MainActivity

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:id="@+id/tvDate"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:clickable="true"
          android:onClick="onclick"
          android:text="@string/hello"
          android:textSize="22sp">
     </TextView>
</LinearLayout>
```

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0591datepickerdialog;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.DatePickerDialog.OnDateSetListener;
import android.app.Dialog;
import android.os.Bundle;
import android.view.View;
import android.widget.DatePicker;
```

```
import android.widget.TextView;

public class MainActivity extends Activity {

  int DIALOG_DATE = 1;
  int myYear = 2011;
  int myMonth = 02;
  int myDay = 03;
  TextView tvDate;

    /** Called when the activity is first created. */

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvDate = (TextView) findViewById(R.id.tvDate);
    }

    public void onclick(View view) {
      showDialog(DIALOG_DATE);
    }


    protected Dialog onCreateDialog(int id) {
      if (id == DIALOG_DATE) {
        DatePickerDialog tpd = new DatePickerDialog(this, myCallBack, myYear, myMonth, myDay);
        return tpd;
      }
      return super.onCreateDialog(id);
    }

    OnDateSetListener myCallBack = new OnDateSetListener() {

    public void onDateSet(DatePicker view, int year, int monthOfYear,
        int dayOfMonth) {
      myYear = year;
      myMonth = monthOfYear;
      myDay = dayOfMonth;
      tvDate.setText("Today is " + myDay + "/" + myMonth + "/" + myYear);
    }
    };
}
```

В **onCreate** находим TextView.

В **onClick** вызываем метод showDialog и передаем ему ID диалога. Этот метод (showDialog) создает диалог с помощью отдельного метода и показывает его. ID используется для указания, какой именно диалог создавать и показывать.

Метод **onCreateDialog** – это и есть отдельный метод, который вызывается в showDialog для создания диалога. В этом методе мы смотрим, какой ID пришел на вход, создаем соответствующий диалог и возвращаем его.

В нашем случае мы создаем DatePickerDialog, используя конструктор:

DatePickerDialog (Context context, DatePickerDialog.OnDateSetListener callBack, int year, int monthOfYear, int dayOfMonth), где

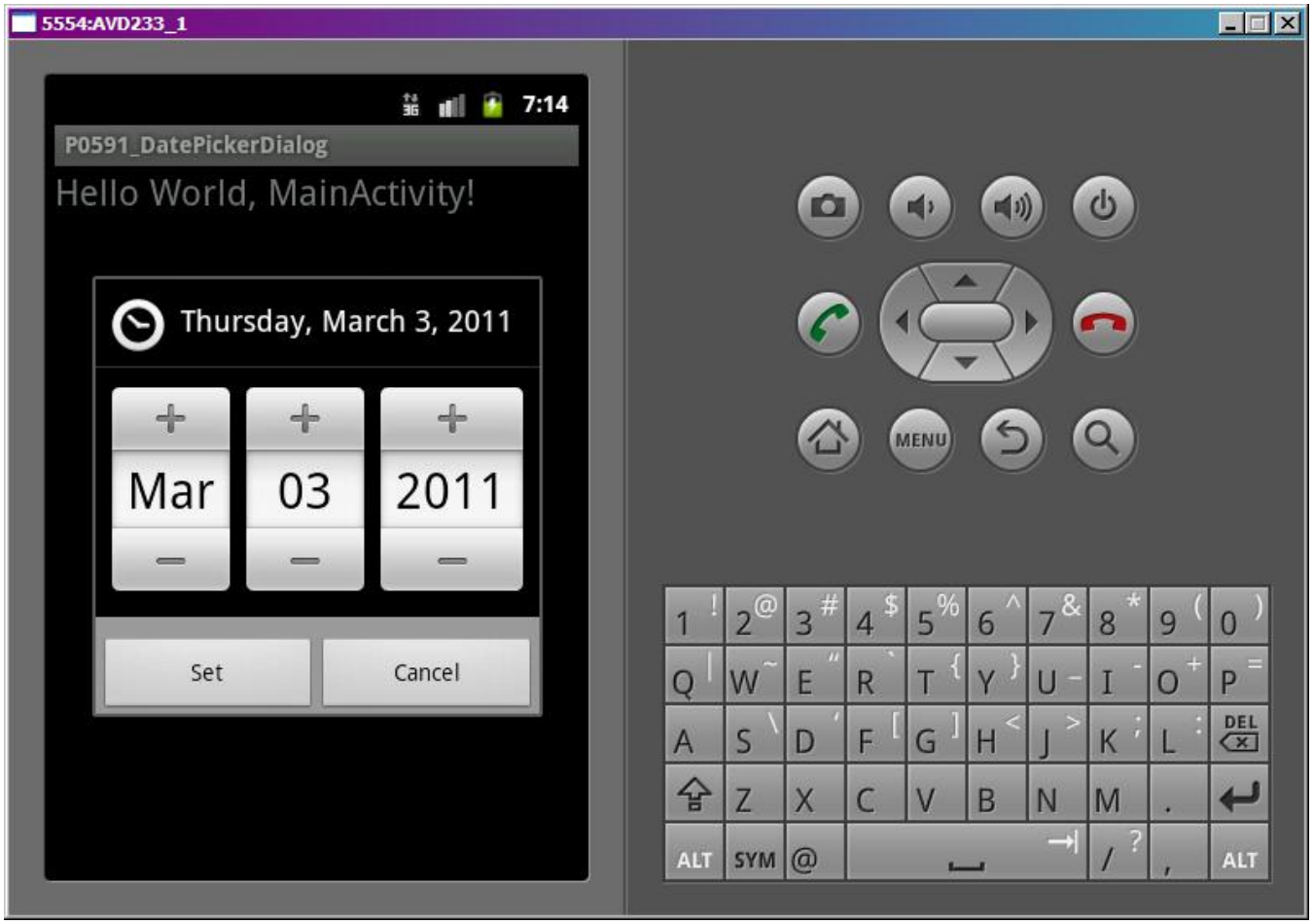
context – контекст
callBack – это обработчик с интерфейсом DatePickerDialog.OnDateSetListener, метод которого срабатывает при нажатии кнопки OK на диалоге

year – год, который покажет диалог
monthOfYear – месяц, который покажет диалог
dayOfMonth – день, который покажет диалог

myCallBack – объект, реализующий интерфейс DatePickerDialog.OnDateSetListener. У него только один метод – onDateSet, который предоставляет нам DatePicker из диалога, и год, месяц и день, которые он показывает. Т.е. то, что мы ввели в диалоге.

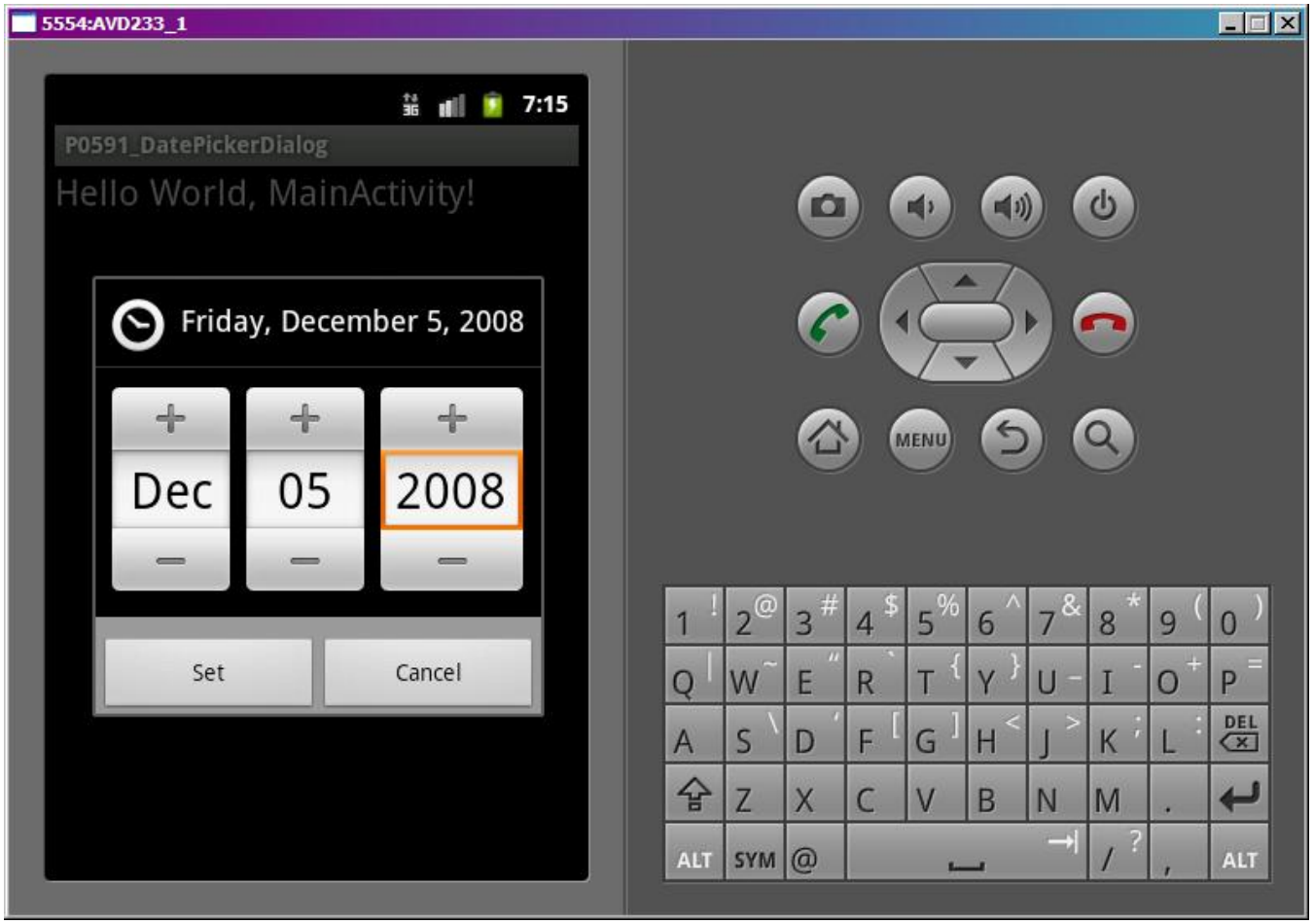
Эти данные мы пишем в tvDate.

Все сохраним и запустим. Нажмем на TextView. Появился диалог для ввода даты.

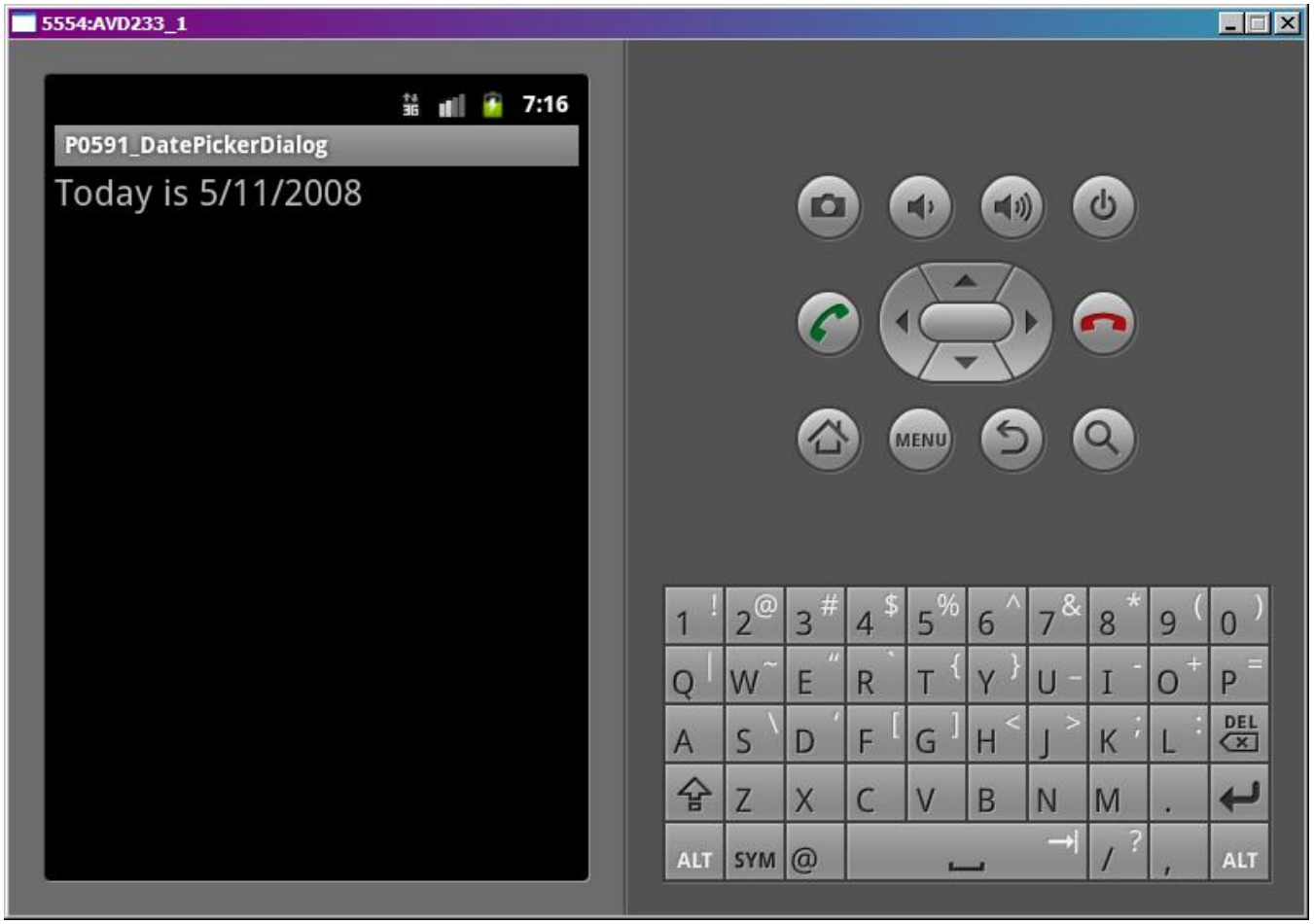


Сейчас он показывает 3 марта 2011, т.к. мы при создании передали ему значения myYear, myMonth и myDay. Месяцы он считает, начиная с нуля.

Поменяем значения


5554:AVD233_1
7:15
P0591_DatePickerDialog
Hello World, MainActivity!
Friday, December 5, 2008
Dec
05
2008
Set
Cancel
MENU

и нажмем Set.


5554:AVD233_1
7:16
P0591_DatePickerDialog
Today is 5/11/2008
MENU

Текст показал новые значения. Обратите внимание, что месяц декабрь он выдал как число 11. Месяцы нумеруются с нуля.

Диалог также можно закрыть кнопкой Cancel или кнопкой Back на эмуляторе. В этом случае обработчик не сработает и текст не обновится.

На следующем уроке:

- создаем AlertDialog
- настраиваем заголовок, сообщение, картинку и кнопки

Урок 60. Диалоги. AlertDialog: Title, Message, Icon, Buttons

В этом уроке:

- создаем AlertDialog
- настраиваем заголовок, сообщение, картинку и кнопки

Начнем знакомство с AlertDialog. Этот **диалог** используется, если вы хотите сообщить о чем-то пользователю или попросить его сделать выбор типа Да/Нет/Отмена.

Напишем приложение, которое при закрытии будет вызывать диалог о сохранении данных, аналогичный диалогу из программ MS Office . Если мы ответим Да, то данные сохранятся, если Нет – то не сохранятся, если Отмена – приложение не закроется.

Создадим проект:

Project name: P0601_AlertDialogSimple
Build Target: Android 2.3.3
Application name: AlertDialogSimple
Package name: ru.startandroid.develop.p0601alertdialogsimple
Create Activity: MainActivity

Добавим в **res/values/strings.xml** строки с текстами:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AlertDialogSimple</string>
    <string  name="exit">Выход</string>
    <string  name="save_data">Сохранить данные?</string>
    <string  name="yes">Да</string>
    <string  name="no">Нет</string>
    <string  name="cancel">Отмена</string>
    <string  name="saved">Сохранено</string>
</resources>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnExit"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/exit"
```

```
        android:onClick="onclick">
    </Button>
</LinearLayout>
```

Код **MainActivity.java**:

```
package ru.startandroid.develop.p0601alertdialogsimple;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends Activity {

  final int DIALOG_EXIT = 1;

    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onclick(View v) {
      // вызываем диалог
      showDialog(DIALOG_EXIT);
    }

    protected Dialog onCreateDialog(int id) {
      if (id == DIALOG_EXIT) {
        AlertDialog.Builder adb = new AlertDialog.Builder(this);
        // заголовок
        adb.setTitle(R.string.exit);
        // сообщение
        adb.setMessage(R.string.save_data);
        // иконка
        adb.setIcon(android.R.drawable.ic_dialog_info);
        // кнопка положительного ответа
        adb.setPositiveButton(R.string.yes, myClickListener);
        // кнопка отрицательного ответа
        adb.setNegativeButton(R.string.no, myClickListener);
        // кнопка нейтрального ответа
        adb.setNeutralButton(R.string.cancel, myClickListener);
        // создаем диалог
        return adb.create();
      }
      return super.onCreateDialog(id);
    }

    OnClickListener myClickListener = new OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
      switch (which) {
      // положительная кнопка
```

```
        case Dialog.BUTTON_POSITIVE:
          saveData();
          finish();
          break;
        // негаитвная кнопка
        case Dialog.BUTTON_NEGATIVE:
          finish();
          break;
        // нейтральная кнопка
        case Dialog.BUTTON_NEUTRAL:
          break;
        }
      }
    };

    void saveData() {
      Toast.makeText(this, R.string.saved, Toast.LENGTH_SHORT).show();
    }
  }
```

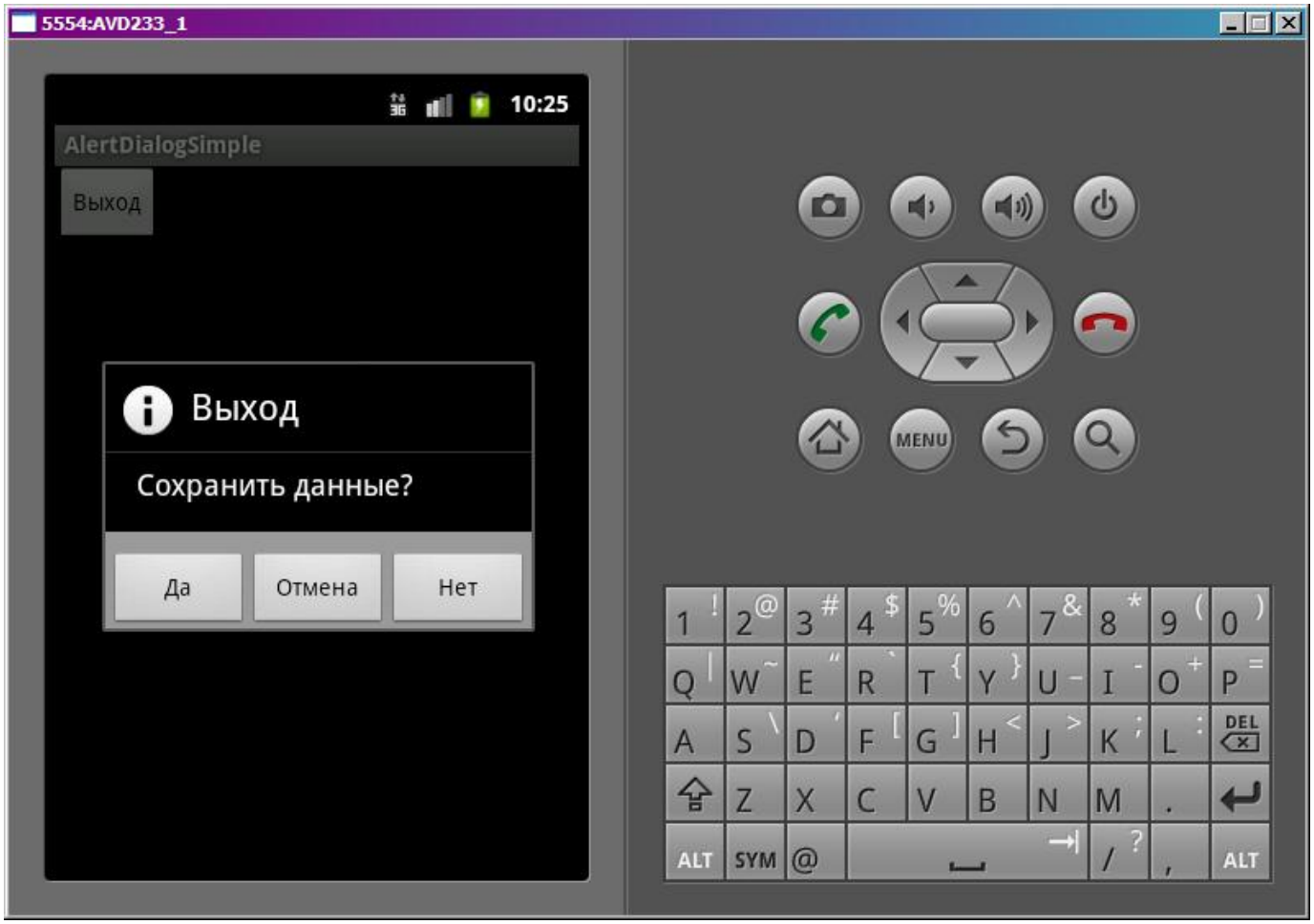
В обработчике кнопки **onclick** вызываем диалог.

В **onCreateDialog** мы создаем диалог. Для этого используется класс AlertDialog.Builder. Мы указываем заголовок, текст сообщения, иконку и кнопки. Диалог может содержать максимум три кнопки ответа: положительная, отрицательная и нейтральная. Для каждой указываем текст и обработчик. Метод create создает диалог и мы его возвращаем (return).

Обработчик кнопок **myClickListener** реализует интерфейс DialogInterface.OnClickListener и в нашем случае является общим для всех кнопок. В нем мы проверяем, какая кнопка была нажата:
если положительная (BUTTON_POSITIVE), то сохраняем данные и закрываем приложение
если отрицательная (BUTTON_NEGATIVE), то закрываем приложение без сохранения
если нейтральная (BUTTON_NEUTRAL), то не делаем ничего

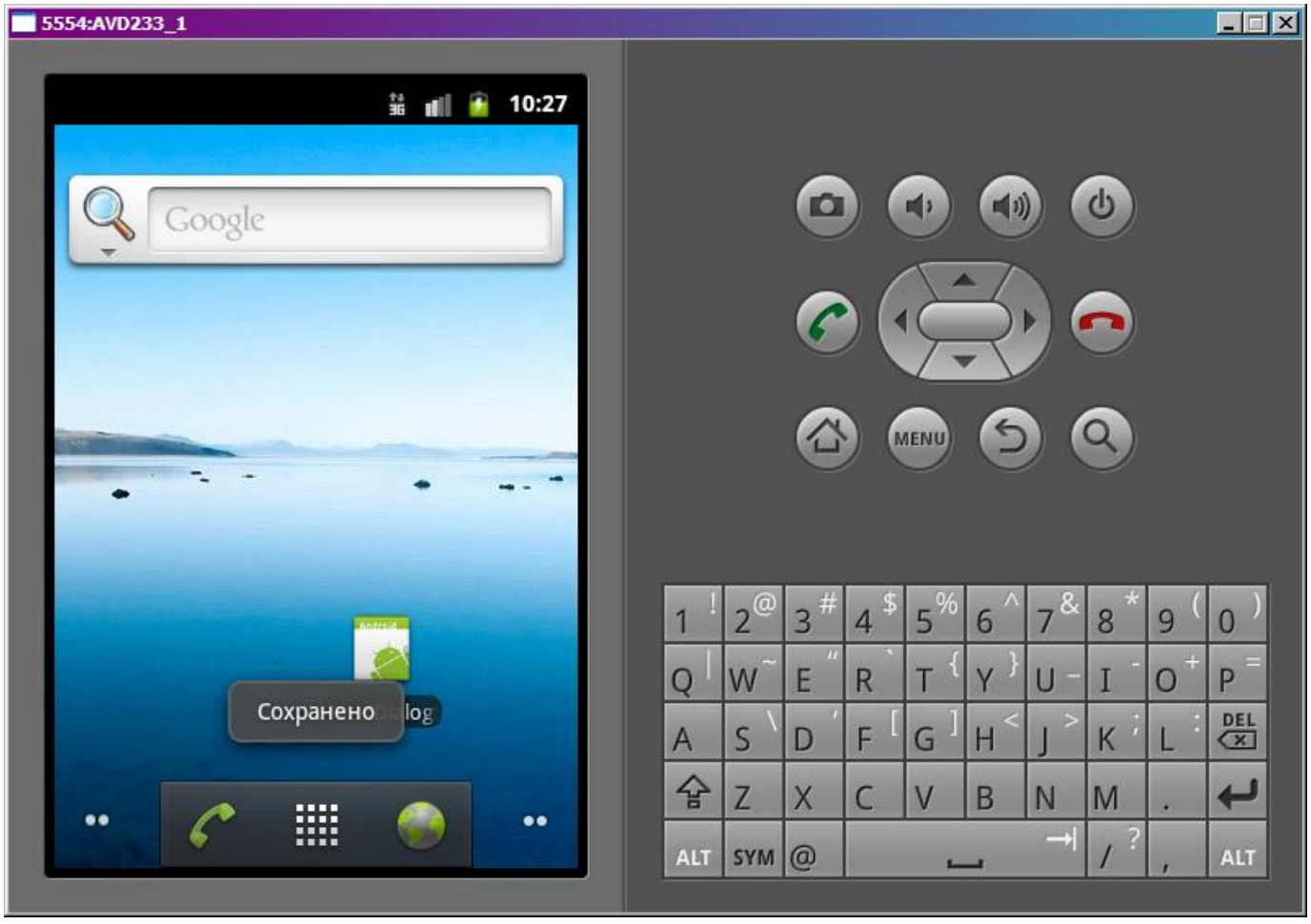
В своем методе saveData выводим текст, что данные как-будто сохранены. Просто, чтобы убедиться, что метод выполняется.

Все сохраним и запустим приложение. Нажмем кнопку Выход:



если жмем **Да**,

то приложение закроется и метод **saveData будет выполнен**.



Если жмем **Отмена**, то диалог закроется и с приложением ничего не произойдет.
А если жмем **Нет**, то приложение закроется **без вызова** нашего метода **saveData**.

Вот так несложно и недолго создать диалог для взаимодействия с пользователем. Использованные нами атрибуты диалога **не являются обязательными**. Вы можете, например, не указывать заголовок, или сделать только одну кнопку, а не три.

Для указания заголовка, сообщения и текстов кнопок необязательно использовать переменные **R.string**. Есть аналогичные реализации методов, которые принимают на вход просто текст.

И еще пара советов.

1) Чтобы диалог вызывался не только по кнопке выход, но и при нажатии на кнопку **Назад** в приложении, добавьте вызов диалога в реализацию метода onBackPressed

```
    public void onBackPressed() {
      // вызываем диалог
      showDialog(DIALOG_EXIT);
    }
```

2) А если хотите, чтобы вызванный диалог не закрывался по нажатию кнопки Назад, то используйте метод setCancelable:

```
adb.setCancelable(false);
```

На следующем уроке:

- используем метод подготовки диалога

Урок 61. Диалоги. AlertDialog.Метод onPrepareDialog

В этом уроке:

- используем метод подготовки диалога

Когда мы создаем диалог в методе onCreateDialog, **Activity** складывает его в кучу созданных диалогов. И когда надо отобразить, достает его и показывает. Т.е. метод **onCreateDialog** выполняется только один раз для диалога. И если вам надо перед отображением что-то изменить, надо использовать метод onPrepareDialog. Этот метод вызывается каждый раз перед показом диалога.

Напишем приложение, в нем будет AlertDialog, который будет показывать текущее время.

Создадим проект:

Project name: P0611_AlertDialogPrepare
Build Target: Android 2.3.3
Application name: AlertDialogPrepare
Package name: ru.startandroid.develop.p0611alertdialogprepare
Create Activity: MainActivity

На экране **main.xml** только кнопка:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/hello"
          android:onClick="onclick">
     </Button>
</LinearLayout>
```

MainActivity.java:

```
package ru.startandroid.develop.p0611alertdialogprepare;

import java.sql.Date;
import java.text.SimpleDateFormat;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.os.Bundle;
import android.util.Log;
```

```
import android.view.View;

public class MainActivity extends Activity {

  final static String LOG_TAG = "myLogs";
  final int DIALOG = 1;
  SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onclick(View v) {
      showDialog(DIALOG);
    }

    protected Dialog onCreateDialog(int id) {
      Log.d(LOG_TAG, "onCreateDialog");
      if (id == DIALOG) {
        AlertDialog.Builder adb = new AlertDialog.Builder(this);
        adb.setTitle("Текущее время");
        adb.setMessage(sdf.format(new Date(System.currentTimeMillis())));
        return adb.create();
      }
      return super.onCreateDialog(id);
    }

    protected void onPrepareDialog(int id, Dialog dialog) {
      super.onPrepareDialog(id, dialog);
      Log.d(LOG_TAG, "onPrepareDialog");
      if (id == DIALOG) {
        ((AlertDialog)dialog).setMessage(sdf.format(new Date(System.currentTimeMillis())));
      }
    }
}
```

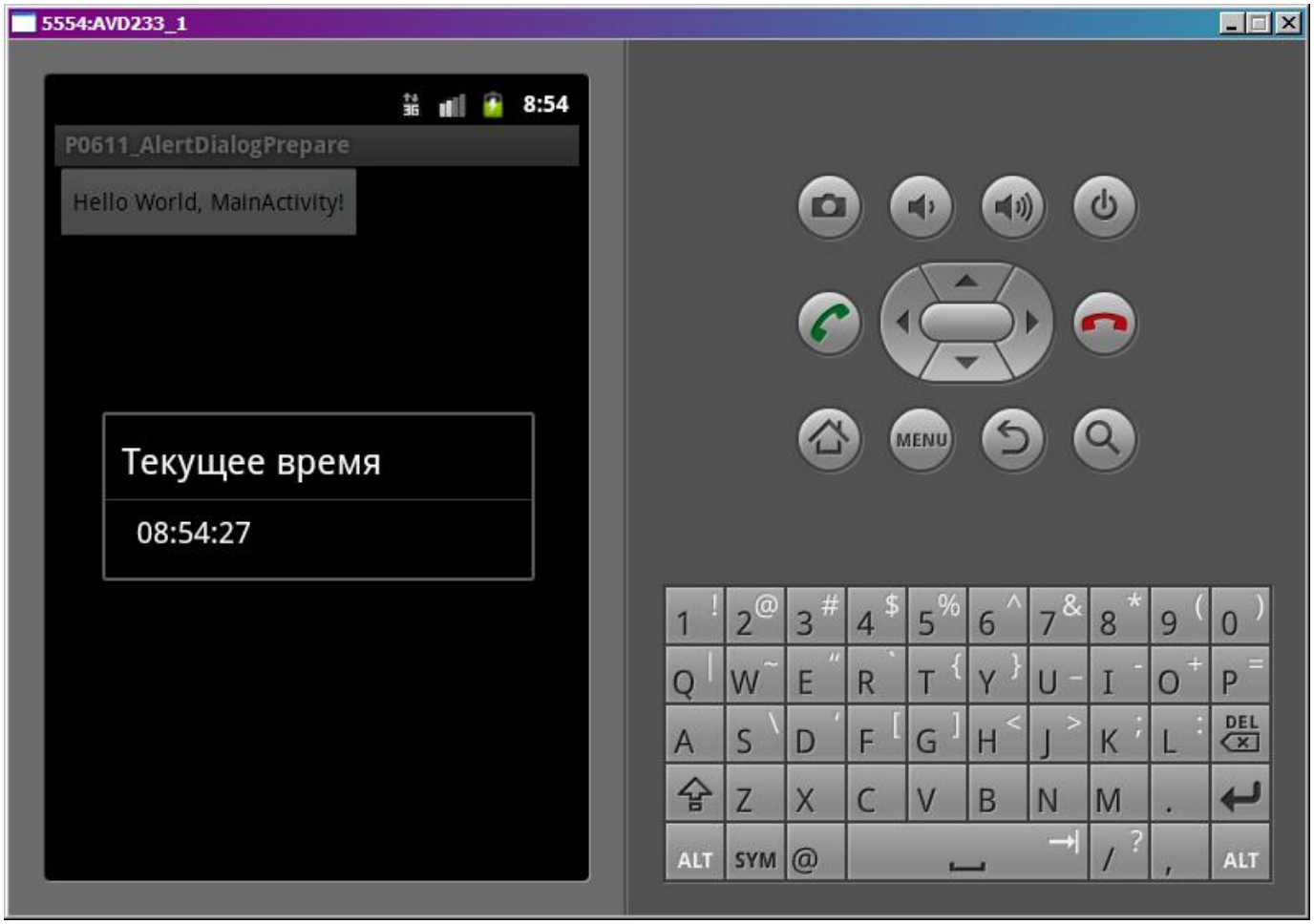
SimpleDateFormat – это класс, позволяющий выводить дату-время в нужном формате. Задаем ему формат **HH:mm:ss** и он покажет время в классическом виде **часы:минуты:секунды**.

В **onCreateDialog** создаем диалог и устанавливаем заголовок и текст. Кнопки не добавляем, они не нужны. Диалог можно будет закрыть кнопкой Назад.

В **onPrepareDialog** мы на вход получаем ID вызываемого диалога и сам диалог (Dialog). Мы преобразуем его к AlertDialog и пишем в текст диалога текущее время.

В методах создания и подготовки диалога мы пишем лог, чтобы убедиться, что создание происходит один раз, а подготовка выполняется перед каждым показом.

Все сохраним и запустим приложение. Нажмем кнопку, появился диалог:

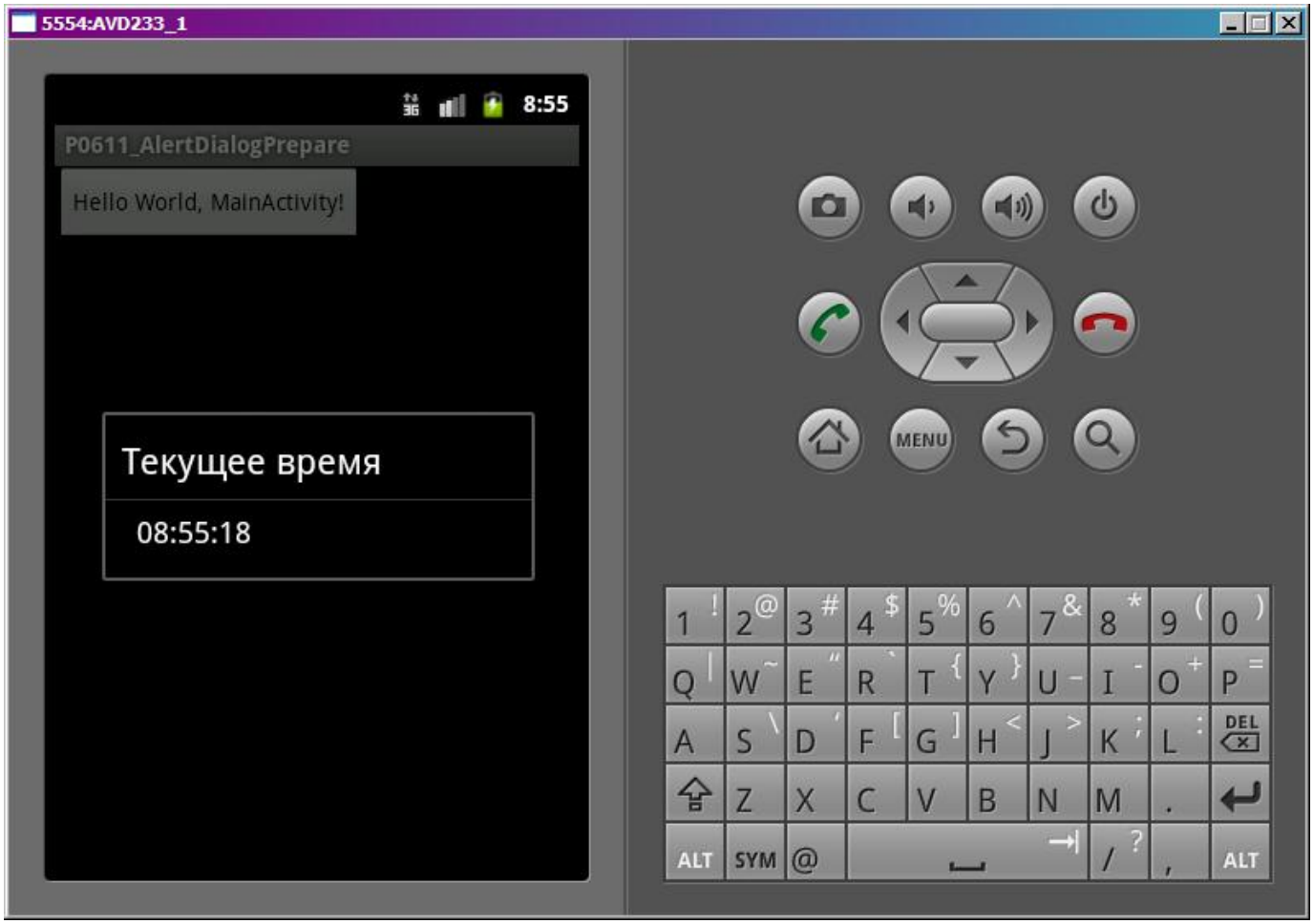


Показывает время. Смотрим лог:

onCreateDialog
onPrepareDialog

Выполнились оба метода – создание и подготовка.

Закроем диалог (но не программу) и вызовем снова



Время обновилось, а в логах добавилась запись.

onPrepareDialog

На этот раз Activity просто достало созданный ранее диалог и выполнило метод его подготовки.

Т.е. когда мы показываем диалог **первый** раз, он проходит через методы **создания** и **подготовки**. Далее мы его закрываем, но при этом объект не уничтожается, а **Activity сохраняет** его у себя. И когда мы **снова** хотим **отобразить** диалог, Activity **достает** его, прогоняет через **метод подготовки** и показывает.

В этом механизме есть небольшой изъян. Диалог, как и все экраны, состоит из набора View-компонентов. В зависимости от используемых при создании параметров диалог задает видимость этих View. Т.е. если вы при создании не задали, например, Message, то в созданном диалоге будет скрыта View (*setVisibility(View.GONE)*), которая отвечает за отображение текста Message. И если в методе подготовки диалога вы решите-таки Message указать, то диалог его просто не отобразит, т.к. структура задается при создании.

Я не нашел способ, как можно заставить диалог заново сформировать себя без удаления и создания заново. Если кто знает – пишите, добавлю в урок. А о том, как удалить диалог, чтобы при показе снова вызвался метод onCreateDialog, мы еще поговорим в следующих уроках.

На следующем уроке:

- формируем список в диалоге

Урок 62. Диалоги. AlertDialog. Список

В этом уроке:

- формируем список в диалоге

В диалог можно выводить не только **текст**, но и **список** значений. Диалоговый список может быть **трех** видов:

- без выбора элементов
- с одиночным выбором
- с множественным выбором

В этом уроке рассмотрим первый вид.

Научимся создавать диалог со списком, используя **массив** данных, **адаптер** или **курсор**. Кроме создания попробуем менять данные и обновлять список перед каждым показом. Для этого введем счетчик показов и будем показывать его в последней строке списка. Каждый вызов списка будет увеличивать счетчик на единицу и это должно отразиться в списке. Так мы убедимся, что он обновляется.

Создадим проект:

Project name: P0621_AlertDialogItems
Build Target: Android 2.3.3
Application name: AlertDialogItems
Package name: ru.startandroid.develop.p0621alertdialogitems
Create Activity: MainActivity

В **strings.xml** пропишем тексты:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AlertDialogItems</string>
    <string  name="items">Items</string>
    <string  name="adapter">Adapter</string>
    <string  name="cursor">Cursor</string>
</resources>
```

На экране **main.xml** три кнопки:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="horizontal">
    <Button
         android:id="@+id/btnItems"
         android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="@string/items"
            android:onClick="onclick">
        </Button>
        <Button
            android:id="@+id/btnAdapter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/adapter"
            android:onClick="onclick">
        </Button>
        <Button
            android:id="@+id/btnCursor"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/cursor"
            android:onClick="onclick">
        </Button>
    </LinearLayout>
```

Для работы с БД выделим отдельный класс **DB.java**:

```
package ru.startandroid.develop.p0621alertdialogitems;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class DB {

  private static final String DB_NAME = "mydb";
  private static final int DB_VERSION = 1;
  private static final String DB_TABLE = "mytab";

  public static final String COLUMN_ID = "_id";
  public static final String COLUMN_TXT = "txt";

  private static final String DB_CREATE =
    "create table " + DB_TABLE + "(" +
      COLUMN_ID + " integer primary key, " +
      COLUMN_TXT + " text" +
    ");";

  private final Context mCtx;


  private DBHelper mDBHelper;
  private SQLiteDatabase mDB;

  public DB(Context ctx) {
```

```
    mCtx = ctx;
  }

  // открыть подключение
  public void open() {
    mDBHelper = new DBHelper(mCtx, DB_NAME, null, DB_VERSION);
    mDB = mDBHelper.getWritableDatabase();
  }

  // закрыть подключение
  public void close() {
    if (mDBHelper!=null) mDBHelper.close();
  }

  // получить все данные из таблицы DB_TABLE
  public Cursor getAllData() {
    return mDB.query(DB_TABLE, null, null, null, null, null, null);
  }

  // изменить запись в DB_TABLE
  public void changeRec(int id, String txt) {
    ContentValues cv = new ContentValues();
    cv.put(COLUMN_TXT, txt);
    mDB.update(DB_TABLE, cv, COLUMN_ID + " = " + id, null);
  }

  // класс по созданию и управлению БД
  private class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context, String name, CursorFactory factory,
        int version) {
      super(context, name, factory, version);
    }

    // создаем и заполняем БД
    @Override
    public void onCreate(SQLiteDatabase db) {
      db.execSQL(DB_CREATE);

      ContentValues cv = new ContentValues();
      for (int i = 1; i < 5; i++) {
        cv.put(COLUMN_ID, i);
        cv.put(COLUMN_TXT, "sometext " + i);
        db.insert(DB_TABLE, null, cv);
      }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
  }
}
```

Тут все как обычно. Методы для открытия и закрытия подключения, получение курсора с данными, и изменение записи по ID. Таблица всего из двух полей – _id и txt. При создании вставляем в таблицу 4 записи.

MainActivity.java:

```
package ru.startandroid.develop.p0621alertdialogitems;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.database.Cursor;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.BaseAdapter;
import android.widget.CursorAdapter;
import android.widget.ListAdapter;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final int DIALOG_ITEMS = 1;
  final int DIALOG_ADAPTER = 2;
  final int DIALOG_CURSOR = 3;
  int cnt = 0;
  DB db;
  Cursor cursor;

  String data[] = { "one", "two", "three", "four" };

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // открываем подключение к БД
    db = new DB(this);
    db.open();
    cursor = db.getAllData();
    startManagingCursor(cursor);
  }

  public void onclick(View v) {
    changeCount();
    switch (v.getId()) {
    case R.id.btnItems:
      showDialog(DIALOG_ITEMS);
      break;
    case R.id.btnAdapter:
      showDialog(DIALOG_ADAPTER);
      break;
    case R.id.btnCursor:
      showDialog(DIALOG_CURSOR);
      break;
    default:
      break;
    }
  }

  protected Dialog onCreateDialog(int id) {
    AlertDialog.Builder adb = new AlertDialog.Builder(this);
```

```
    switch (id) {
    // массив
    case DIALOG_ITEMS:
      adb.setTitle(R.string.items);
      adb.setItems(data, myClickListener);
      break;
    // адаптер
    case DIALOG_ADAPTER:
      adb.setTitle(R.string.adapter);
      ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
          android.R.layout.select_dialog_item, data);
      adb.setAdapter(adapter, myClickListener);
      break;
    // курсор
    case DIALOG_CURSOR:
      adb.setTitle(R.string.cursor);
      adb.setCursor(cursor, myClickListener, DB.COLUMN_TXT);
      break;
    }
    return adb.create();
  }

  protected void onPrepareDialog(int id, Dialog dialog) {
    // получаем доступ к адаптеру списка диалога
    AlertDialog aDialog = (AlertDialog) dialog;
    ListAdapter lAdapter = aDialog.getListView().getAdapter();

    switch (id) {
    case DIALOG_ITEMS:
    case DIALOG_ADAPTER:
      // проверка возможности преобразования
      if (lAdapter instanceof BaseAdapter) {
        // преобразование и вызов метода-уведомления о новых данных
        BaseAdapter bAdapter = (BaseAdapter) lAdapter;
        bAdapter.notifyDataSetChanged();
      }
      break;
    case DIALOG_CURSOR:
      break;
    default:
      break;
    }
  };

  // обработчик нажатия на пункт списка диалога
  OnClickListener myClickListener = new OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
      // выводим в лог позицию нажатого элемента
      Log.d(LOG_TAG, "which = " + which);
    }
  };

  // меняем значение счетчика
  void changeCount() {
    cnt++;
    // обновляем массив
    data[3] = String.valueOf(cnt);
    // обновляем БД
    db.changeRec(4, String.valueOf(cnt));
    cursor.requery();
```

```
    }

    @Override
    protected void onDestroy() {
      super.onDestroy();
      db.close();
    }
  }
```

data – массив из 4 элементов. В таблице из **DB.java** у нас тоже 4 записи. Мы решили, что будем менять последний (четвертый) элемент/запись в данных и помещать туда кол-во показов.

В **onCreate** подключаемся к базе.

onclick – меняем значение счетчика и корректируем данные массива и БД, и в зависимости от нажатой кнопки вызываем соответствующий диалог.

onCreateDialog – создаем вызываемый диалог, используя AlertDialog.Builder. Диалог может построить **список**, используя один из следующих объектов:

1) **Массив** строк. Используется метод setItems. На вход подается массив и обработчик нажатия.

2) **Адаптер**. Мы создаем ArrayAdapter, используя массив **data** и стандартный **layout select_dialog_item**, и передаем его в метод setAdapter. Также передаем туда и обработчик.

3) **Курсор** БД. Вызываем метод setCursor. Передаем туда курсор, обработчик нажатия и имя поля, значение которого будет показано в списке.

Кроме списка указываем только заголовок. Кнопки и иконку не добавляем. В конце создаем и возвращаем Dialog.

Метод создания диалога (onCreateDialog) выполняется один раз, чтобы создать диалог. При последующих показах диалога выполняется метод **onPrepareDialog**. В нем мы будем обновлять данные списка. С помощью преобразований и методов getListView и getAdapter получим список из диалога, а потом адаптер из списка.

Далее для диалогов, использующих массив и адаптер мы выполняем преобразование до BaseAdapter, чтобы иметь возможность вызвать метод notifyDataSetChanged. Этот метод обновит список в соответствии с новыми данными.

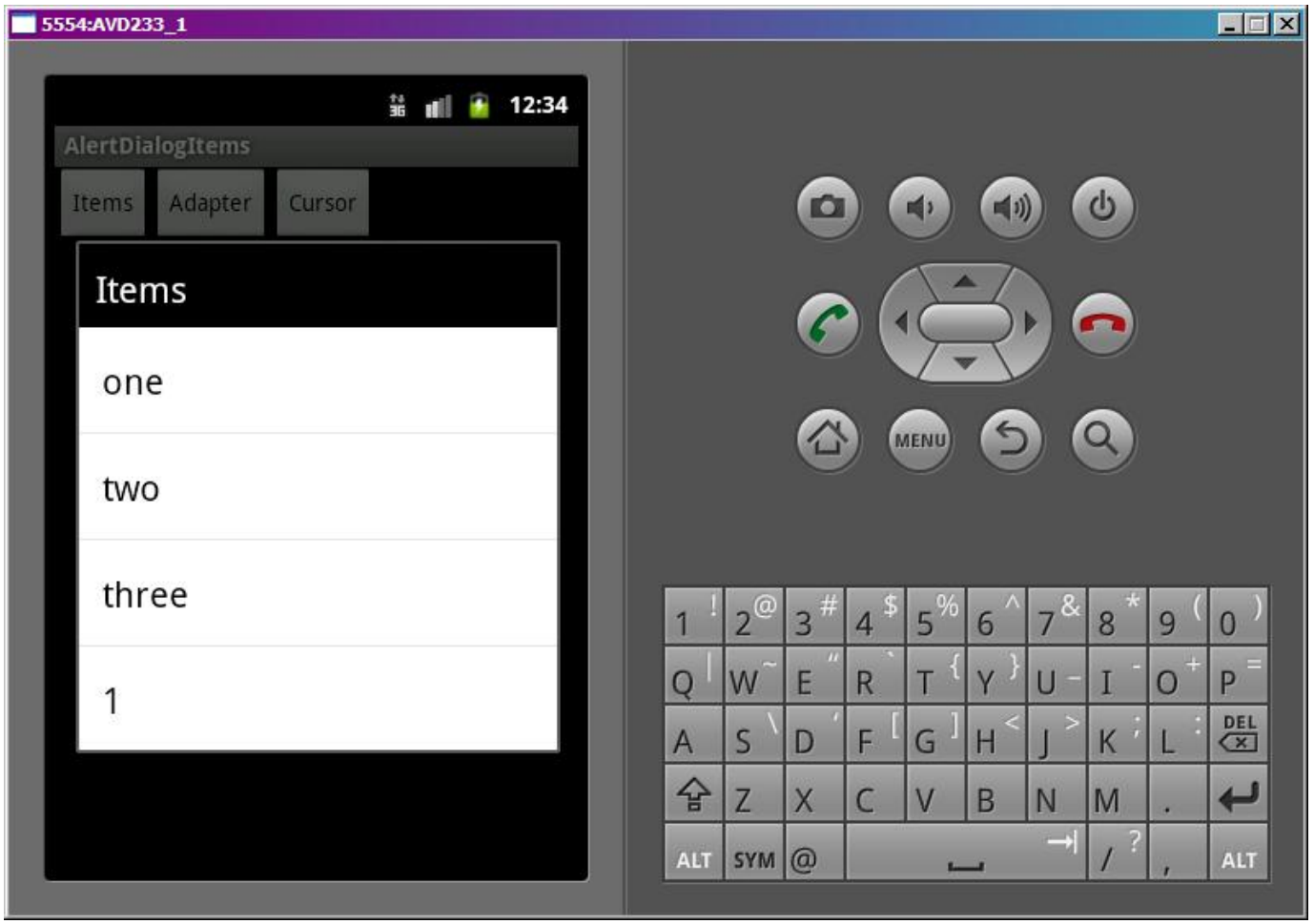
Для диалога с курсором нет необходимости уведомлять адаптер о новых данных. Курсор это делает сам.

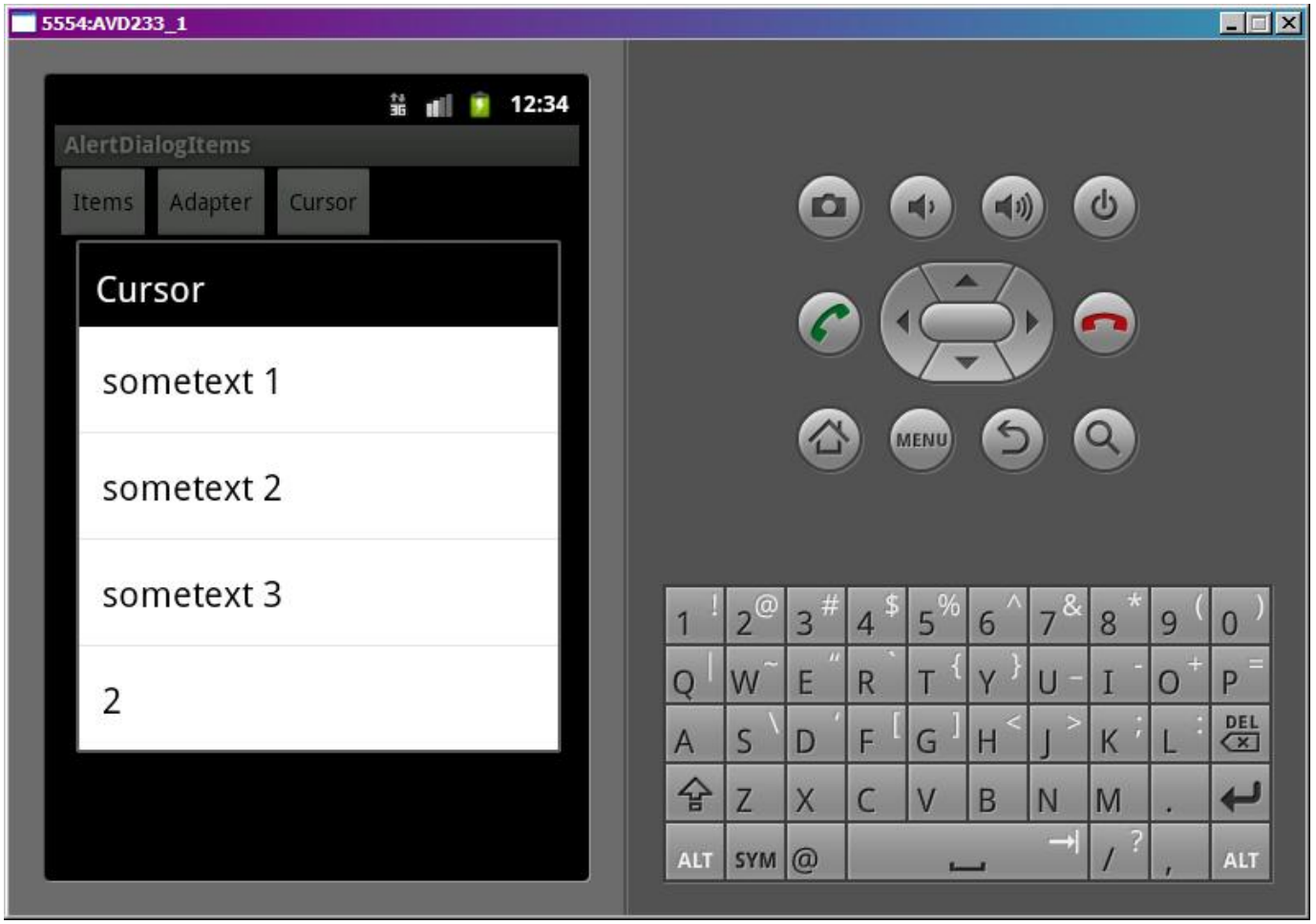
myClickListener – обработчик нажатия на пункты списка. Он у нас общий для всех диалогов и просто выводит в лог позицию нажатого пункта.

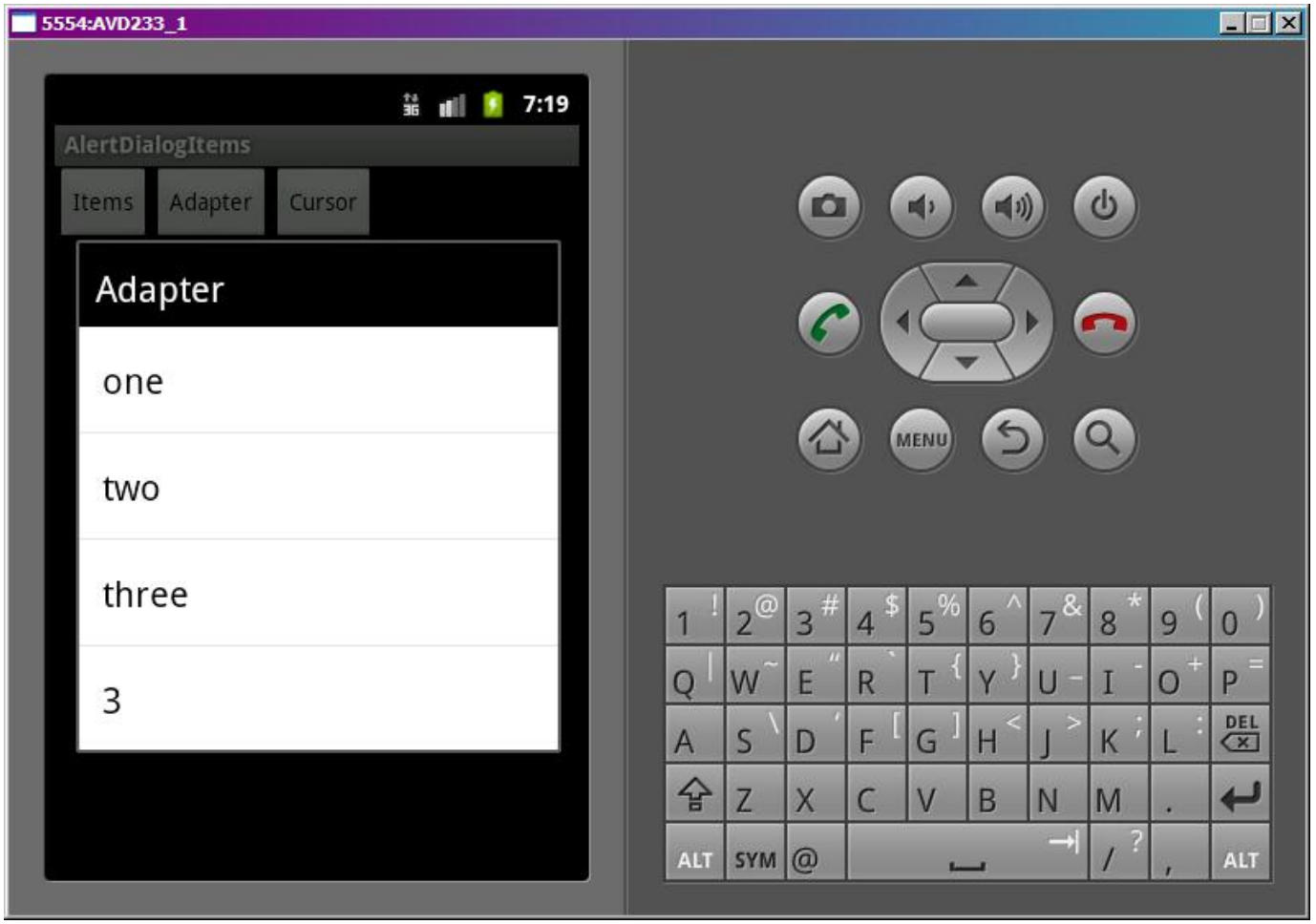
В **changeCount** увеличиваем счетчик на единицу и пишем это значение в четвертый элемент массива (нумерация с нуля) и в строку с _id = 4 в БД. Обновляем курсор.

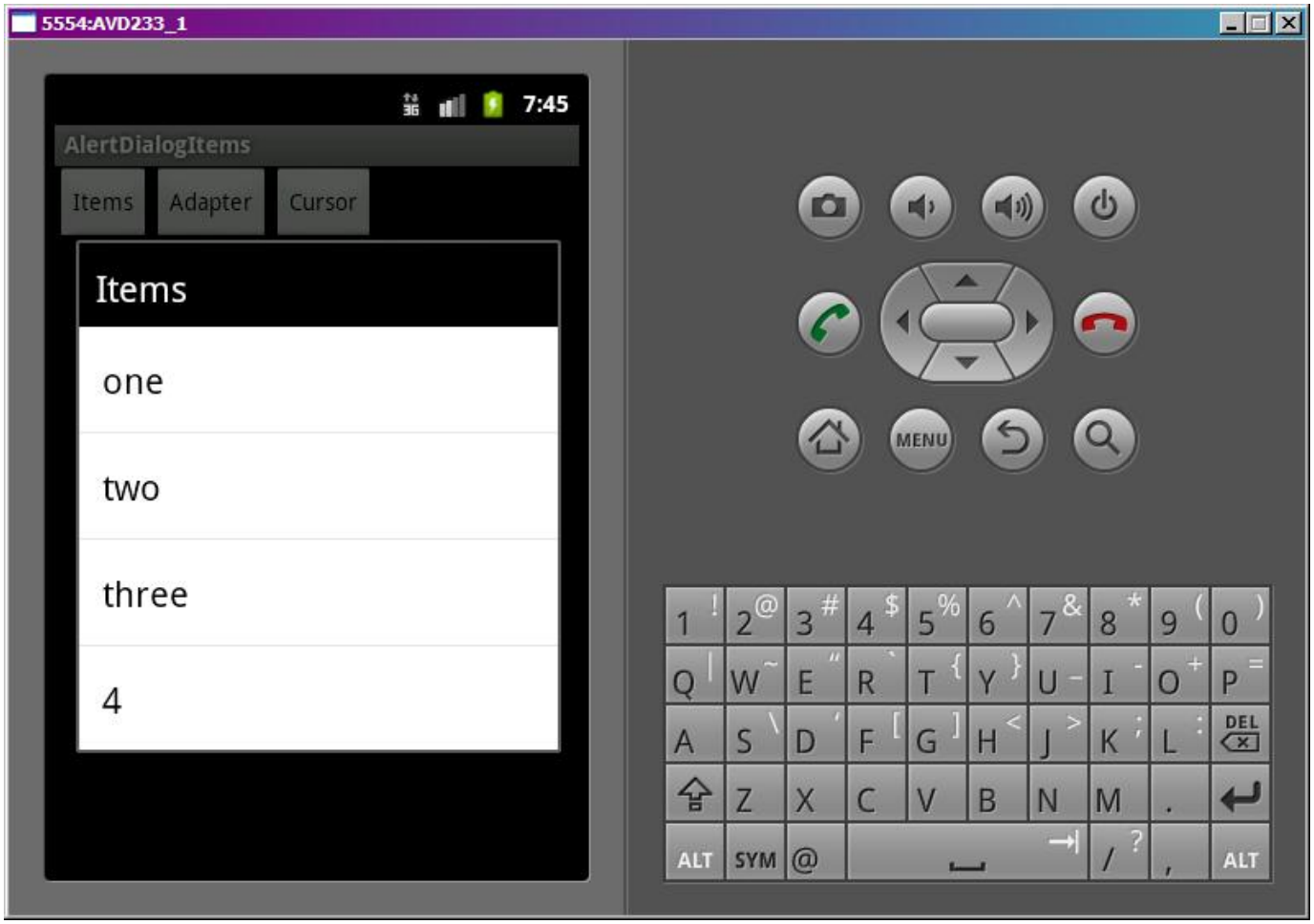
В **onDestroy** закрываем подключение к БД.

Все сохраним и запустим. Повызываем диалоги и убедимся, что данные в них обновляются при каждом показе.




5554:AVD233_1
12:34
AlertDialogItems
Items
Adapter
Cursor
Cursor
sometext 1
sometext 2
sometext 3
2
MENU


5554:AVD233_1
7:19
AlertDialogItems
Items
Adapter
Cursor
Adapter
one
two
three
3
MENU



Клик на пункте списка задействует обработчик, выведет в лог его позицию и закроет диалог.

Если бы мы не реализовали метод **onPrepareDialog**, то списки диалогов (кроме курсорного) не обновлялись и выводили в последней строке те значения счетчика, при которых они создавались в методе **onCreateDialog**. Попробуйте закоментить содержимое onPrepareDialog и убедиться в этом.

UPD от 11.07.2012. Протестил этот урок на Android 3.2 - данные обновляются из без onPrepareDialog.

На следующем уроке:

- формируем список с одиночным выбором в диалоге

Урок 63. Диалоги. AlertDialog. Список с одиночным выбором

В этом уроке:

- формируем список с одиночным выбором в диалоге

Урок будет аналогичен прошлому, только теперь мы не будем обновлять данные списка диалога.

Мы будем формировать в диалоге список с одиночным выбором и определять какой элемент был выбран (чекнут).

Создадим проект:

Project name: P0631_AlertDialogItemsSingle
Build Target: Android 2.3.3
Application name: AlertDialogItemsSingle
Package name: ru.startandroid.develop.p0631alertdialogitemssingle
Create Activity: MainActivity

В **strings.xml** пропишем тексты:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AlertDialogItemsSingle</string>
    <string  name="items">Items</string>
    <string  name="adapter">Adapter</string>
    <string  name="cursor">Cursor</string>
    <string  name="ok">OK</string>
</resources>
```

main.xml, три кнопки:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="horizontal">
    <Button
         android:id="@+id/btnItems"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/items"
         android:onClick="onclick">
    </Button>
    <Button
         android:id="@+id/btnAdapter"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/adapter"
            android:onClick="onclick">
        </Button>
        <Button
            android:id="@+id/btnCursor"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/cursor"
            android:onClick="onclick">
        </Button>
    </LinearLayout>
```

DB.java аналогичен с прошлого урока, только уберем метод изменения записей

```
package ru.startandroid.develop.p0631alertdialogitemssingle;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class DB {

  private static final String DB_NAME = "mydb";
  private static final int DB_VERSION = 1;
  private static final String DB_TABLE = "mytab";

  public static final String COLUMN_ID = "_id";
  public static final String COLUMN_TXT = "txt";

  private static final String DB_CREATE =
    "create table " + DB_TABLE + "(" +
      COLUMN_ID + " integer primary key, " +
      COLUMN_TXT + " text" +
    ");";

  private final Context mCtx;


  private DBHelper mDBHelper;
  private SQLiteDatabase mDB;

  public DB(Context ctx) {
    mCtx = ctx;
  }

  // открыть подключение
  public void open() {
    mDBHelper = new DBHelper(mCtx, DB_NAME, null, DB_VERSION);
    mDB = mDBHelper.getWritableDatabase();
```

```
  }

  // закрыть подключение
  public void close() {
    if (mDBHelper!=null) mDBHelper.close();
  }

  // получить все данные из таблицы DB_TABLE
  public Cursor getAllData() {
    return mDB.query(DB_TABLE, null, null, null, null, null, null);
  }

  // класс по созданию и управлению БД
  private class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context, String name, CursorFactory factory,
        int version) {
      super(context, name, factory, version);
    }

    // создаем и заполняем БД
    @Override
    public void onCreate(SQLiteDatabase db) {
      db.execSQL(DB_CREATE);

      ContentValues cv = new ContentValues();
      for (int i = 1; i < 5; i++) {
        cv.put(COLUMN_ID, i);
        cv.put(COLUMN_TXT, "sometext " + i);
        db.insert(DB_TABLE, null, cv);
      }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
  }
}
```

MainActivity.java немного изменился:

```
package ru.startandroid.develop.p0631alertdialogitemssingle;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.database.Cursor;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {
```

```
final String LOG_TAG = "myLogs";

final int DIALOG_ITEMS = 1;
final int DIALOG_ADAPTER = 2;
final int DIALOG_CURSOR = 3;
DB db;
Cursor cursor;

String data[] = { "one", "two", "three", "four" };

/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  // открываем подключение к БД
  db = new DB(this);
  db.open();
  cursor = db.getAllData();
  startManagingCursor(cursor);
}

public void onclick(View v) {
  switch (v.getId()) {
  case R.id.btnItems:
    showDialog(DIALOG_ITEMS);
    break;
  case R.id.btnAdapter:
    showDialog(DIALOG_ADAPTER);
    break;
  case R.id.btnCursor:
    showDialog(DIALOG_CURSOR);
    break;
  default:
    break;
  }
}

protected Dialog onCreateDialog(int id) {
  AlertDialog.Builder adb = new AlertDialog.Builder(this);
  switch (id) {
  // массив
  case DIALOG_ITEMS:
    adb.setTitle(R.string.items);
    adb.setSingleChoiceItems(data, -1, myClickListener);
    break;
  // адаптер
  case DIALOG_ADAPTER:
    adb.setTitle(R.string.adapter);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.select_dialog_singlechoice, data);
    adb.setSingleChoiceItems(adapter, -1, myClickListener);
    break;
  // курсор
  case DIALOG_CURSOR:
    adb.setTitle(R.string.cursor);
    adb.setSingleChoiceItems(cursor, -1, DB.COLUMN_TXT, myClickListener);
    break;
  }
  adb.setPositiveButton(R.string.ok, myClickListener);
```

```
    return adb.create();
  }

  // обработчик нажатия на пункт списка диалога или кнопку
  OnClickListener myClickListener = new OnClickListener() {
    public void onClick(DialogInterface dialog, int which) {
      ListView lv = ((AlertDialog) dialog).getListView();
      if (which == Dialog.BUTTON_POSITIVE)
        // выводим в лог позицию выбранного элемента
        Log.d(LOG_TAG, "pos = " + lv.getCheckedItemPosition());
      else
        // выводим в лог позицию нажатого элемента
        Log.d(LOG_TAG, "which = " + which);
    }
  };

  @Override
  protected void onDestroy() {
    super.onDestroy();
    db.close();
  }

}
```

В **onCreate** подключаемся к базе.

onclick – в зависимости от нажатой кнопки вызываем соответствующий диалог.

onCreateDialog – создаем вызываемый диалог, используя AlertDialog.Builder. Диалог может построить список, используя один из следующих объектов:

1) **Массив** строк. Используется метод setSingleChoiceItems. На вход подается массив, позиция выбранного элемента и обработчик нажатия. Если мы в значение выбранного элемента передаем -1, то в списке не будет выбранного элемента.

2) **Адаптер**. Мы создаем ArrayAdapter, используя массив data и стандартный layout select_dialog_singlechoice, и передаем его в метод setSingleChoiceItems. Также передаем туда позицию выбранного элемента и обработчик.

3) **Курсор** БД. Вызываем метод setSingleChoiceItems. Передаем туда курсор, позицию выбранного элемента, имя поля (значение которого будет показано в списке) и обработчик нажатия.

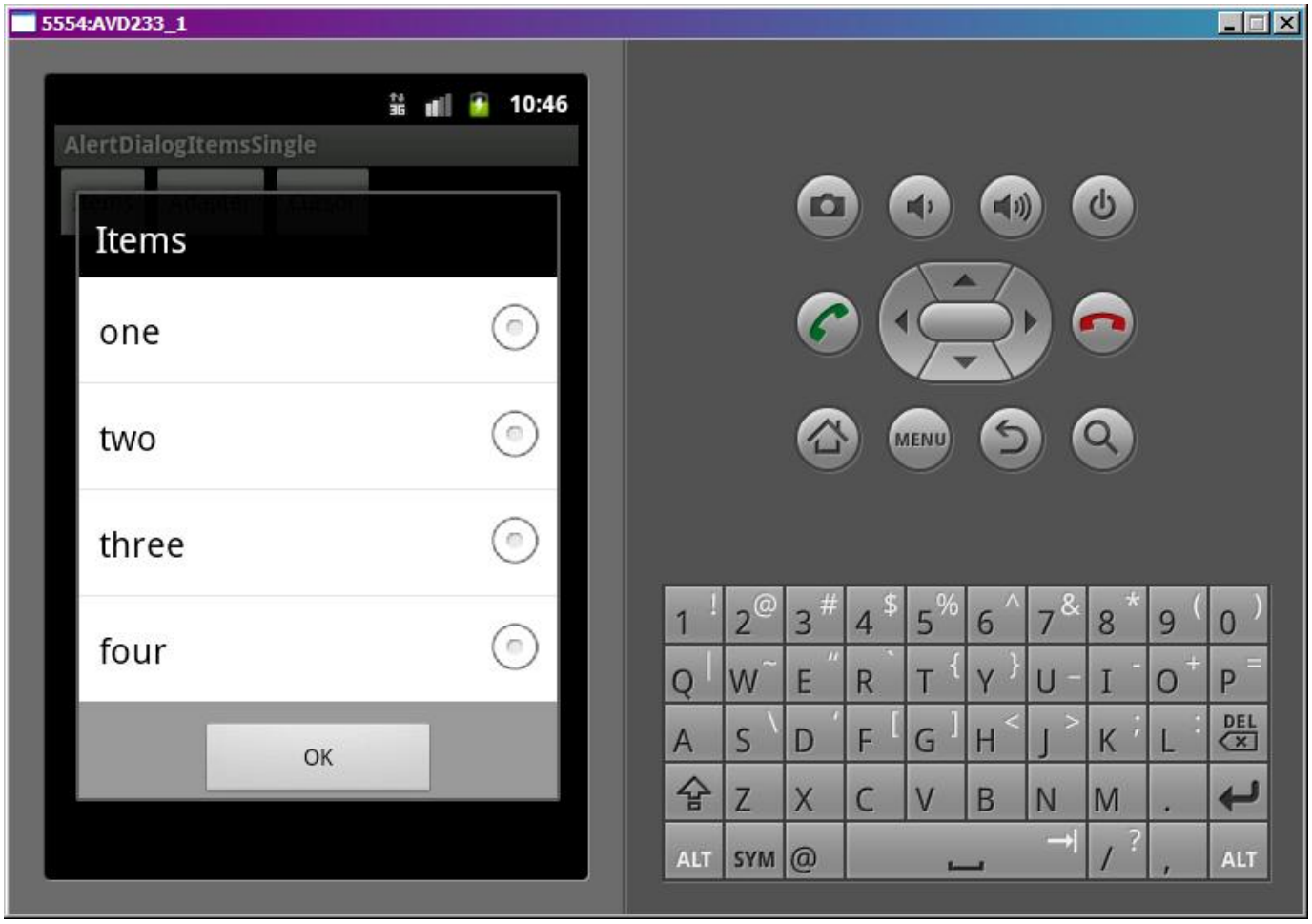
Кроме списка указываем только заголовок. В конце добавим кнопку OK, создаем и возвращаем Dialog.

Как видим для всех трех способов создания используются методы с одинаковым названием setSingleChoiceItems, но с разными аргументами на вход.

myClickListener – обработчик нажатия на пункты списка и на кнопку. Если было нажатие на кнопку (а значит - закрытие диалога), то в лог выводим информацию о выбранном элементе. Иначе (нажатие на пункт списка) – выводим позицию нажатого элемента. Если диалог содержит список с **одиночным** или **множественным** выбором, то нажатие на пункт списка **не приводит к закрытию** диалога. Но это можно без проблем реализовать программно.

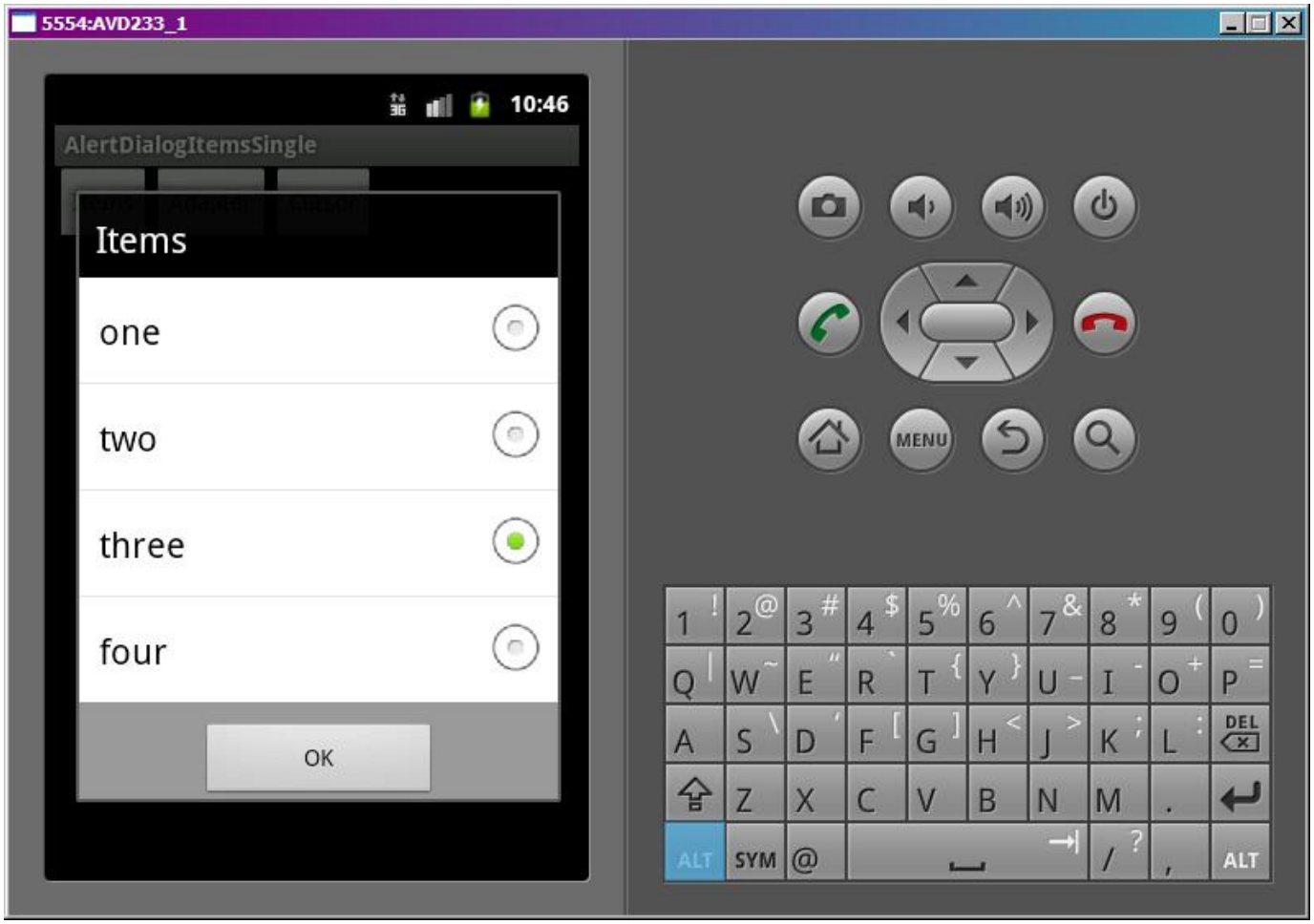
В **onDestroy** закрываем подключение к БД.

Все сохраним и запустим. Вызовем какой-нибудь диалог.



Ничего не выбрано, т.к. мы указали -1 в методе setSingleChoiceItems.

Нажмем на пункт, он выделился



А в логе появился текст

which = 2

Если нажмем на OK, то диалог закроется. А лог отобразит позицию выбранного элемента.

pos = 2

Если теперь снова откроете диалог, то выбранный пункт так и останется выбранным. Т.к. диалог не уничтожается при закрытии. Если хотите перед каждым вызовом менять выбранный элемент, реализуйте метод **onPrepareDialog**. В нем надо добраться до списка и вызвать метод **setItemChecked**.

Пример кода, в котором выбирается третий элемент (нумерация с нуля):

```
protected void onPrepareDialog(int id, Dialog dialog) {
  ((AlertDialog) dialog).getListView().setItemChecked(2, true);
};
```

На следующем уроке:

- формируем список с множественным выбором в диалоге

Урок 64. Диалоги. AlertDialog. Список с множественным выбором

В этом уроке:

- формируем список с множественным выбором в диалоге

Урок будет аналогичен прошлому. Мы будем формировать в диалоге список с множественным выбором и определять какие элементы были выбраны (чекнуты). На этот раз мы сможем работать только с массивом и курсором. Дать диалогу адаптер не получится, нет соответствующего метода. Это особенности реализации.

Создадим проект:

Project name: P0641_AlertDialogItemsMulti
Build Target: Android 2.3.3
Application name: AlertDialogItemsMulti
Package name: ru.startandroid.develop.p0641alertdialogitemsmulti
Create Activity: MainActivity

В **strings.xml** пропишем тексты:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AlertDialogItemsMulti</string>
    <string  name="items">Items</string>
    <string  name="cursor">Cursor</string>
    <string  name="ok">OK</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="horizontal">
    <Button
         android:id="@+id/btnItems"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/items"
         android:onClick="onclick">
    </Button>
    <Button
         android:id="@+id/btnCursor"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/cursor"
         android:onClick="onclick">
    </Button>
</LinearLayout>
```

DB.java:

```
package ru.startandroid.develop.p0641alertdialogitemsmulti;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class DB {

  private static final String DB_NAME = "mydb";
  private static final int DB_VERSION = 1;
  private static final String DB_TABLE = "mytab";

  public static final String COLUMN_ID = "_id";
  public static final String COLUMN_CHK = "checked";
  public static final String COLUMN_TXT = "txt";

  private static final String DB_CREATE =
    "create table " + DB_TABLE + "(" +
      COLUMN_ID + " integer primary key, " +
      COLUMN_CHK + " integer, " +
      COLUMN_TXT + " text" +
    ");";

  private final Context mCtx;


  private DBHelper mDBHelper;
  private SQLiteDatabase mDB;

  public DB(Context ctx) {
    mCtx = ctx;
  }

  // открыть подключение
  public void open() {
    mDBHelper = new DBHelper(mCtx, DB_NAME, null, DB_VERSION);
    mDB = mDBHelper.getWritableDatabase();
  }

  // закрыть подключение
  public void close() {
    if (mDBHelper!=null) mDBHelper.close();
  }

  // получить все данные из таблицы DB_TABLE
  public Cursor getAllData() {
    return mDB.query(DB_TABLE, null, null, null, null, null, null);
  }

  //изменить запись в DB_TABLE
  public void changeRec(int pos, boolean isChecked) {
      ContentValues cv = new ContentValues();
      cv.put(COLUMN_CHK, (isChecked) ? 1 : 0);
      mDB.update(DB_TABLE, cv, COLUMN_ID + " = " + (pos + 1), null);
  }

  // класс по созданию и управлению БД
```

```
  private class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context, String name, CursorFactory factory,
        int version) {
      super(context, name, factory, version);
    }

    // создаем и заполняем БД
    @Override
    public void onCreate(SQLiteDatabase db) {
      db.execSQL(DB_CREATE);

      ContentValues cv = new ContentValues();
      for (int i = 1; i < 5; i++) {
      cv.put(COLUMN_ID, i);
      cv.put(COLUMN_TXT, "sometext " + i);
      cv.put(COLUMN_CHK, 0);
      db.insert(DB_TABLE, null, cv);
      }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
  }
}
```

Мы в таблице создаем числовое поле **checked**. По этому полю список диалога будет определять выделен элемент (значение = 1) или нет (0). Метод **changeRec** берет на вход позицию элемента в списке и boolean-значение, выделен элемент или нет, и меняет соответствующую запись в таблице.

MainActivity.java:

```
package ru.startandroid.develop.p0641alertdialogitemsmulti;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.content.DialogInterface.OnMultiChoiceClickListener;
import android.database.Cursor;
import android.os.Bundle;
import android.util.Log;
import android.util.SparseBooleanArray;
import android.view.View;
import android.widget.CursorAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final int DIALOG_ITEMS = 1;
  final int DIALOG_CURSOR = 3;
  DB db;
  Cursor cursor;

  String data[] = { "one", "two", "three", "four" };
```

```
boolean chkd[] = { false, true, true, false };

/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  // открываем подключение к БД
  db = new DB(this);
  db.open();
  stopManagingCursor(cursor);
  cursor = db.getAllData();
}

public void onclick(View v) {
  switch (v.getId()) {
  case R.id.btnItems:
    showDialog(DIALOG_ITEMS);
    break;
  case R.id.btnCursor:
    showDialog(DIALOG_CURSOR);
    break;
  default:
    break;
  }
}

protected Dialog onCreateDialog(int id) {
  AlertDialog.Builder adb = new AlertDialog.Builder(this);
  switch (id) {
  // массив
  case DIALOG_ITEMS:
    adb.setTitle(R.string.items);
    adb.setMultiChoiceItems(data, chkd, myItemsMultiClickListener);
    break;
  // курсор
  case DIALOG_CURSOR:
    adb.setTitle(R.string.cursor);
    adb.setMultiChoiceItems(cursor, DB.COLUMN_CHK, DB.COLUMN_TXT, myCursorMultiClickListener);
    break;
  }
  adb.setPositiveButton(R.string.ok, myBtnClickListener);
  return adb.create();
}

// обработчик для списка массива
OnMultiChoiceClickListener myItemsMultiClickListener = new OnMultiChoiceClickListener() {
  @Override
  public void onClick(DialogInterface dialog, int which, boolean isChecked) {
    Log.d(LOG_TAG, "which = " + which + ", isChecked = " + isChecked);
  }
};

// обработчик для списка курсора
OnMultiChoiceClickListener myCursorMultiClickListener = new OnMultiChoiceClickListener() {
  @Override
  public void onClick(DialogInterface dialog, int which, boolean isChecked) {
    ListView lv = ((AlertDialog) dialog).getListView();
    Log.d(LOG_TAG, "which = " + which + ", isChecked = " + isChecked);
    db.changeRec(which, isChecked);
    cursor.requery();
  }
};
```

```
    // обработчик нажатия на кнопку
    OnClickListener myBtnClickListener = new OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
          SparseBooleanArray sbArray = ((AlertDialog)dialog).getListView().getCheckedItemPositions();
          for (int i = 0; i < sbArray.size(); i++) {
            int key = sbArray.keyAt(i);
            if (sbArray.get(key))
              Log.d("qwe", "checked: " + key);
          }
      }
    };

    protected void onDestroy() {
      super.onDestroy();
      db.close();
    }

  }
```

Разбираем код. Кроме массива строк **data**, нам нужен массив boolean[] **chkd**, который укажет диалогу, какие элементы списка надо будет сразу сделать выделенными. Будем выделять второй и третий.

В **onCreate** подключаемся к базе.

onclick – в зависимости от нажатой кнопки вызываем соответствующий диалог.

onCreateDialog – создаем вызываемый диалог, используя AlertDialog.Builder. Диалог может построить список, используя один из следующих объектов:

1) **Массив** строк. Используется метод setMultiChoiceItems. На вход подается массив строк, boolean-массив, определяющий выделенные элементов, и обработчик нажатия.

2) **Курсор** БД. Вызываем метод setMultiChoiceItems. Передаем туда курсор, имя поля выделения (данные о выделении элементов списка), имя поля с текстом (текст, который будет отображен в списке) и обработчик нажатия.

Кроме списка указываем только заголовок. В конце добавим кнопку OK, создаем и возвращаем Dialog.

Для обоих способов создания используется методы с одинаковым названием setMultiChoiceItems, но с разными аргументами на вход.

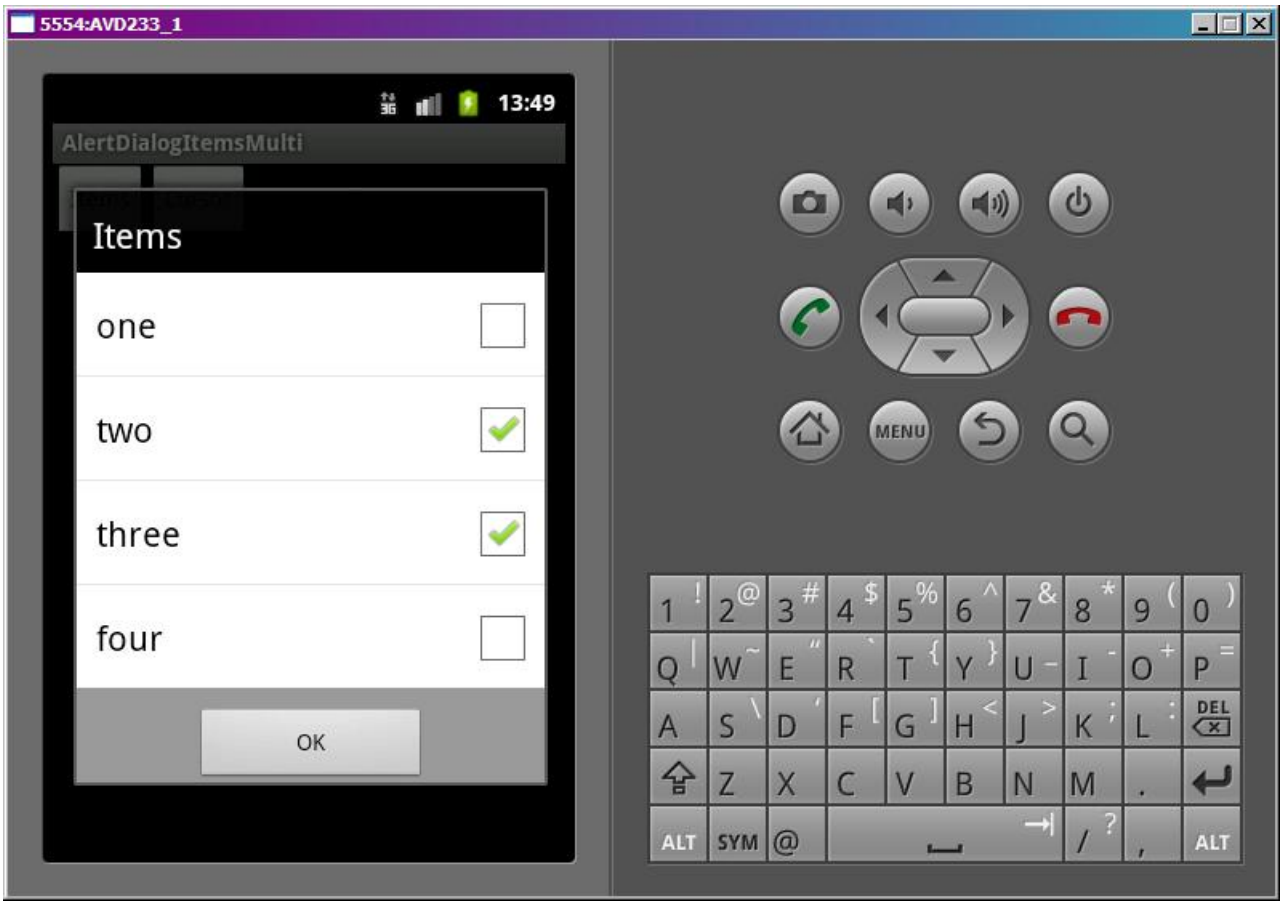
myItemsMultiClickListener – обработчик нажатий на список, построенный из массива. Выводит в лог какой элемент был нажат и стал он выделенным или не выделенным. Реализует интерфейс OnMultiChoiceClickListener.

myCursorMultiClickListener - обработчик нажатий на список, построенный из курсора. Выводит в лог какой элемент был нажат и стал он выделенным или не выделенным. Также он соответствующим образом меняет данные в БД и обновляет курсор списка диалога. Т.к. если какой-то элемент выделили, мы должны в обработчике сбегать в БД, обновить соответствующую запись (поле checked) и обновить курсор. Ради интереса закомментируйте код этого обработчика – вы увидите, что галки в списке просто не ставятся.

myBtnClickListener – обработчик нажатия на кнопку. Получает из списка информацию о выделенных элементах и выводит ее в лог.

В **onDestroy** закрываем подключение к БД.

Все сохраним и запустим. Открываем диалог Items.

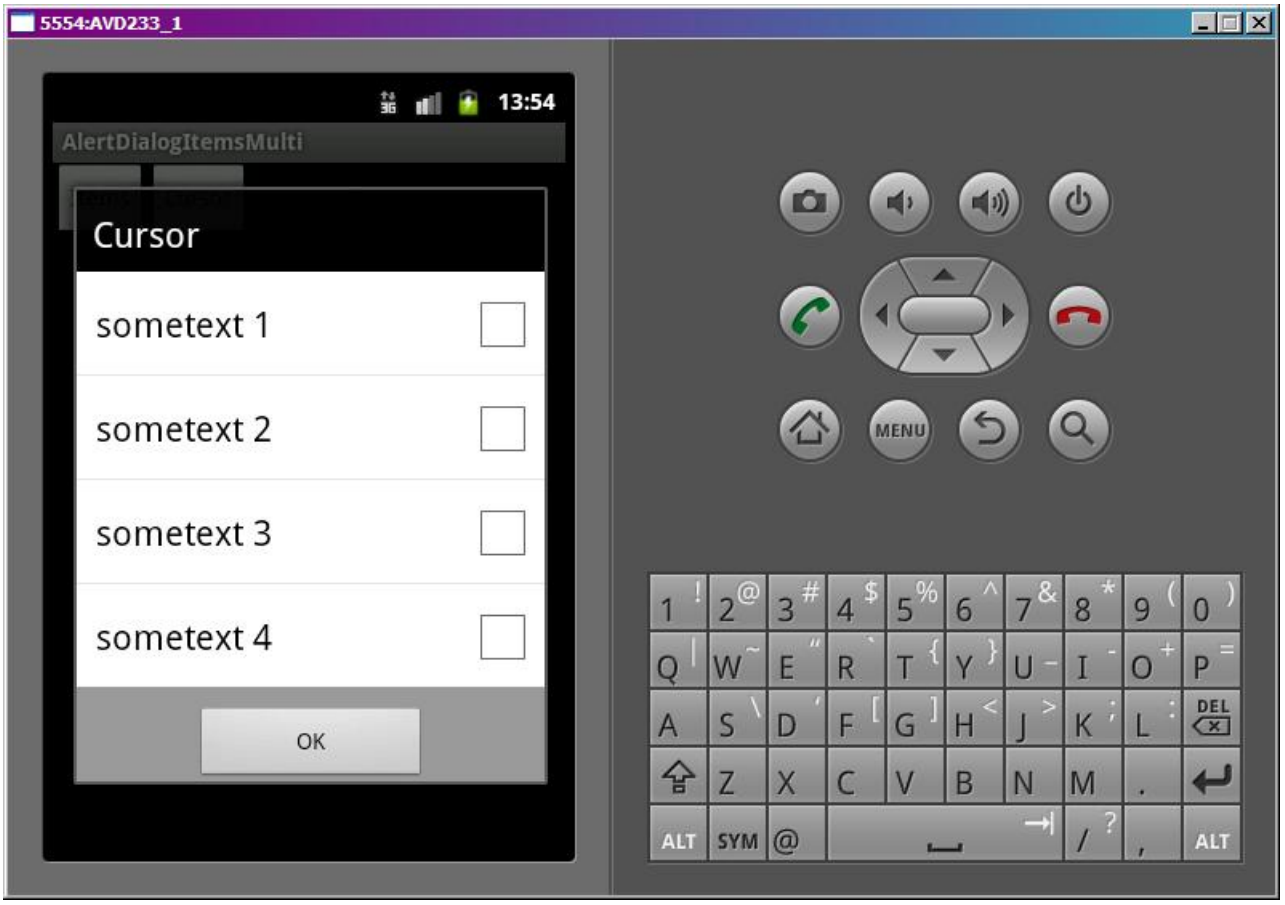


Видим, что галки проставились так, как мы указывали в массиве chkd. Если понажимать на пункты списка, лог показывает, какие изменения происходят.

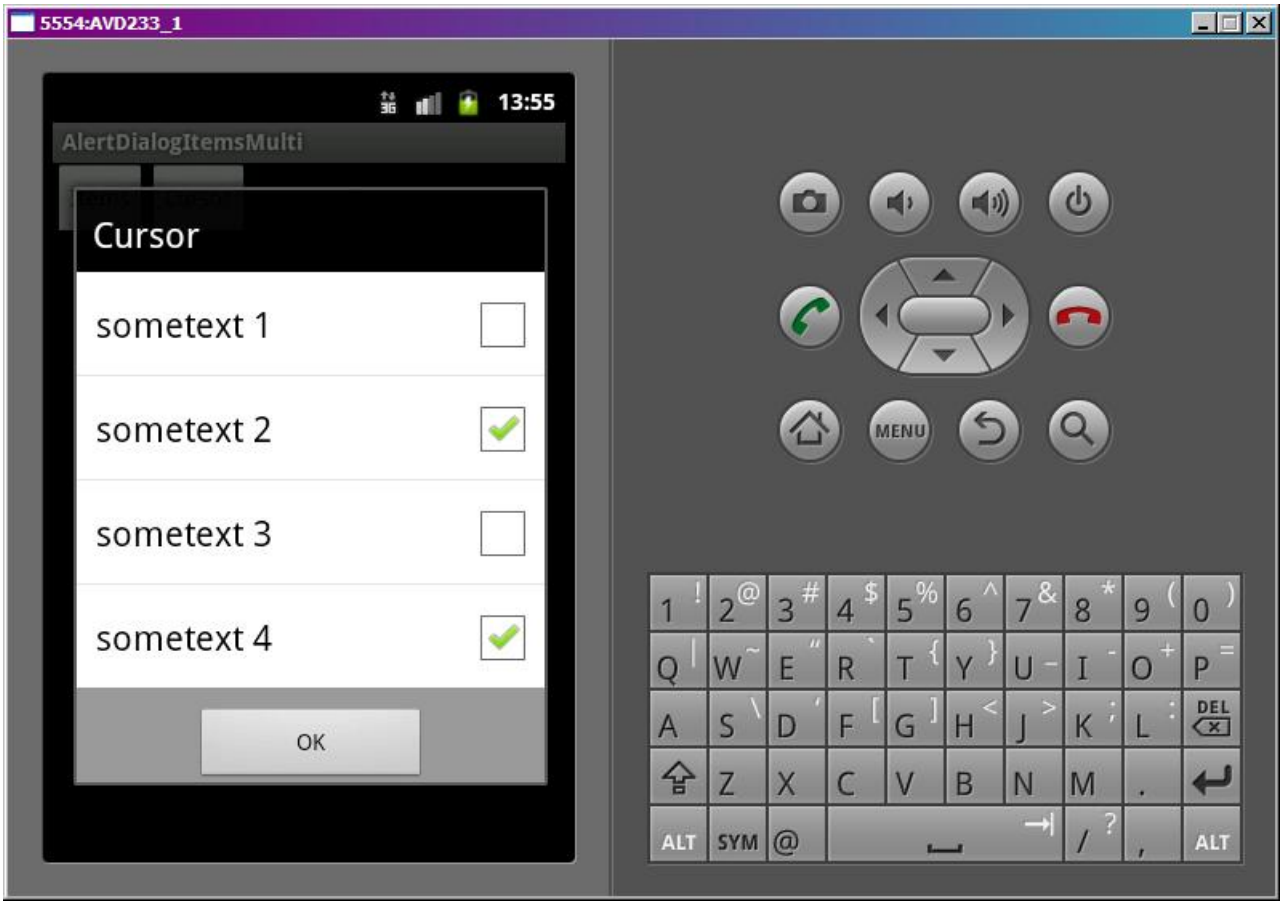
which = 1, isChecked = false
which = 0, isChecked = true
which = 3, isChecked = true

Если есть необходимость, можно добавить в обработчик код, который обновлял бы массив chkd в соответствии с нажатиями. Индекс элемента и значение у нас есть.

Откроем диалог Cursor. Здесь ничего не выделено, т.к. в поле checked мы поместили нули.



Проставим галки,



лог это отобразит

which = 1, isChecked = true
which = 3, isChecked = true

Это работает только потому, что мы вручную обновляем БД и курсор. Повторюсь, попробуйте закоментить код обработчика myCursorMultiClickListener и элементы перестанут выделяться.

Я не стал реализовывать метод **onPrepareDialog**. В случае Items, там все просто, аналогично прошлому уроку используйте такой код:

*((AlertDialog)dialog).getListView().setItemChecked(2, **true**);*

В этом примере: 2 – это позиция элемента списка, а true - означает, что галка должна стоять. Если хотите снять галку, передавайте false.

В случае с курсором код будет аналогичен коду из обработчика **myCursorMultiClickListener**. Меняете запись в БД методом **db.changeRec**, обновляете курсор и передаете его адаптеру.

Нетривиальная такая получилась штука, не знаю пригодится кому или нет, но раз уж взялся за диалоги - решил расписать и это.

Также наверно имеет смысл сказать, что у методов setItems, setSingleChoiceItems, setMultiChoiceItems есть также реализация, использующая не напрямую массив, а ID массива строк из файла ресурсов.

На следующем уроке:

- используем свои View для построения диалога

Урок 65. Диалоги. AlertDialog. Кастомизация

В этом уроке:

- используем свои View для построения диалога

Кроме сообщения или списка мы можем помещать в диалог свои View-компоненты. Для этого доступно как основное тело диалога, так и заголовок. Для этого нам необходимо создать View и с помощью методов setCustomTitle или setView вставить этот View соответственно в заголовок или тело диалога. Я буду использовать только setView и работать с телом диалога. Использование метода setCustomTitle и работа с заголовком диалога полностью аналогична.

Нарисуем приложение, которое будет использовать свой View-компонент в теле диалога и рассмотрим, как можно модифицировать содержимое этого компонента. Будем добавлять и удалять TextView в теле диалога.

Создадим проект:

Project name: P0651_AlertDialogCustom
Build Target: Android 2.3.3
Application name: AlertDialogCustom
Package name: ru.startandroid.develop.p0651alertdialogcustom
Create Activity: MainActivity

В **strings.xml** пропишем тексты:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AlertDialogCustom</string>
    <string  name="add">Добавить</string>
    <string  name="remove">Удалить</string>
</resources>
```

main.xml – экран с двумя кнопками для удаления и добавления элементов в диалог

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnAdd"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/add"
         android:onClick="onclick">
    </Button>
    <Button
         android:id="@+id/btnRemove"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/remove"
            android:onClick="onclick">
        </Button>
    </LinearLayout>
```

dialog.xml – наше будущее кастом-тело диалога.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView
          android:id="@+id/tvTime"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="">
    </TextView>
    <TextView
          android:id="@+id/tvCount"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="">
    </TextView>
</LinearLayout>
```

В **tvTime**- будем отображать текущее время, а в **tvCount** – кол-во добавленных TextView.

MainActivity.java:

```
package ru.startandroid.develop.p0651alertdialogcustom;

import java.sql.Date;
import java.text.SimpleDateFormat;
import java.util.ArrayList;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends Activity {

  final int DIALOG = 1;
```

```
int btn;
LinearLayout view;
SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
TextView tvCount;
ArrayList<TextView> textViews;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  textViews = new ArrayList<TextView>(10);
}

public void onclick(View v) {
  btn = v.getId();
  showDialog(DIALOG);
}

@Override
protected Dialog onCreateDialog(int id) {
  AlertDialog.Builder adb = new AlertDialog.Builder(this);
  adb.setTitle("Custom dialog");
  // создаем view из dialog.xml
  view = (LinearLayout) getLayoutInflater()
      .inflate(R.layout.dialog, null);
  // устанавливаем ее, как содержимое тела диалога
  adb.setView(view);
  // находим TexView для отображения кол-ва
  tvCount = (TextView) view.findViewById(R.id.tvCount);
  return adb.create();
}

@Override
protected void onPrepareDialog(int id, Dialog dialog) {
  super.onPrepareDialog(id, dialog);
  if (id == DIALOG) {
    // Находим TextView для отображения времени и показываем текущее
    // время
    TextView tvTime = (TextView) dialog.getWindow().findViewById(
        R.id.tvTime);
    tvTime.setText(sdf.format(new Date(System.currentTimeMillis())));
    // если была нажата кнопка Добавить
    if (btn == R.id.btnAdd) {
      // создаем новое TextView, добавляем в диалог, указываем текст
      TextView tv = new TextView(this);
      view.addView(tv, new LayoutParams(LayoutParams.MATCH_PARENT,
          LayoutParams.WRAP_CONTENT));
      tv.setText("TextView " + (textViews.size() + 1));
      // добавляем новое TextView в коллекцию
      textViews.add(tv);
      // иначе
    } else {
      // если коллекция созданных TextView непуста
      if (textViews.size() > 0) {
        // находим в коллекции последний TextView
        TextView tv = textViews.get(textViews.size() - 1);
        // удаляем из диалога
        view.removeView(tv);
```

```
            // удаляем из коллекции
            textViews.remove(tv);
          }
        }
        // обновляем счетчик
        tvCount.setText("Кол-во TextView = " + textViews.size());
      }
    }
}
```

Рассмотрим код. В методе **onCreate** выполняем стандартные процедуры и создаем коллекцию textViews для хранения добавляемых TextView.

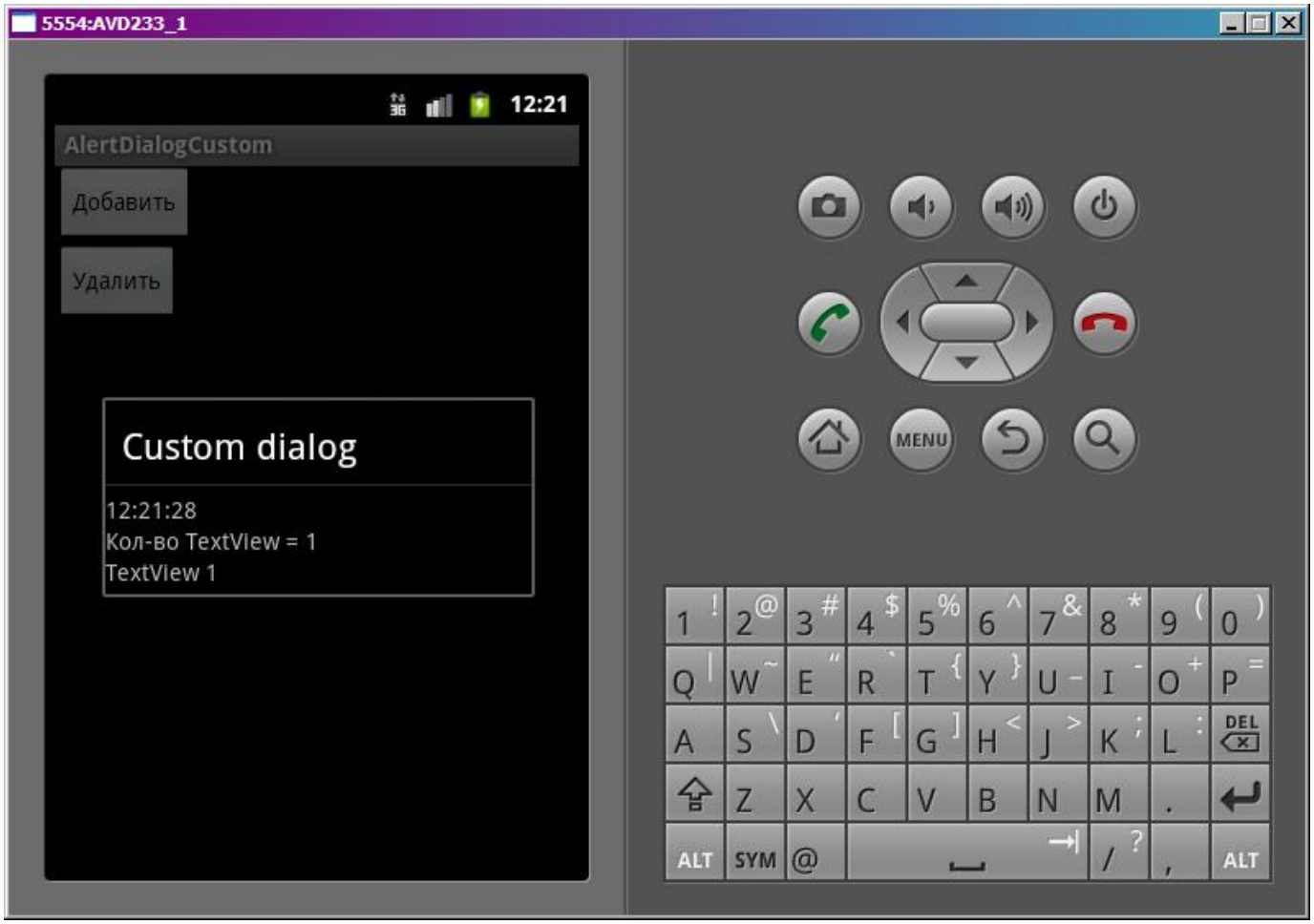
Метод **onclick** – обработчик нажатий на кнопки, сохраняет в btn идентификатор нажатой кнопки и показывает диалог.

В методе создания диалога **onCreateDialog** мы задаем текст заголовка диалога, создаем **view** из layout-файла dialog.xml и с помощью метода setView говорим диалогу, что надо использовать наше **view** в качестве тела диалога. И теперь, работая с **view**, мы будем формировать тело диалога. В только что созданном **view** сразу находим tvCount – для отображения кол-ва добавленных TextView.

Метод **onPrepareDialog** – здесь мы с помощью метода getWindow получаем доступ к View-компонентам диалога, находим среди них tvTime и показываем в нем время. Далее определяем, какая кнопка была нажата. Если кнопка добавления, то создаем TextView им помещаем его в **view** (который определили в методе onCreateDialog) и добавляем в коллекцию textViews. Таким образом TextView добавится в тело диалога. Если же хотим удалить TextView, то находим в коллекции последний добавленный и удаляем его из компонента view и из коллекции textViews. В конце обновляем счетчик кол-ва добавленных TextView в диалоге.

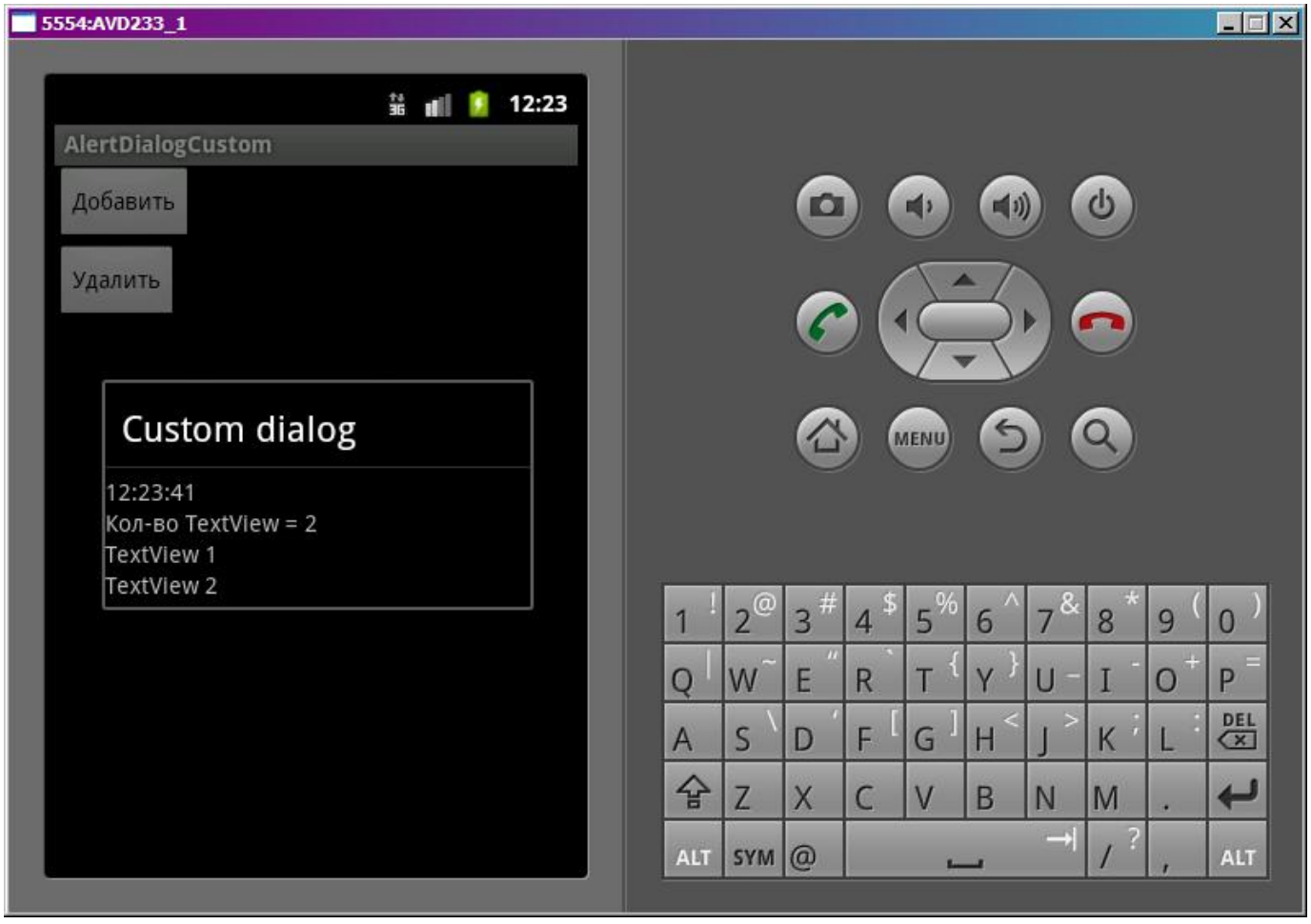
Обратите внимание, что я использую два разных способа для получения доступа к tvCount и tvTime. tvCount я нашел сразу после создания **view** в методе onCreateDialog. А в случае с tvTime я показываю, как найти View-компонент в диалоге без использования объекта **view**. Какой вам удобнее по ситуации, тот и используйте.

Все сохраним и запустим приложение. Нажмем кнопку Добавить



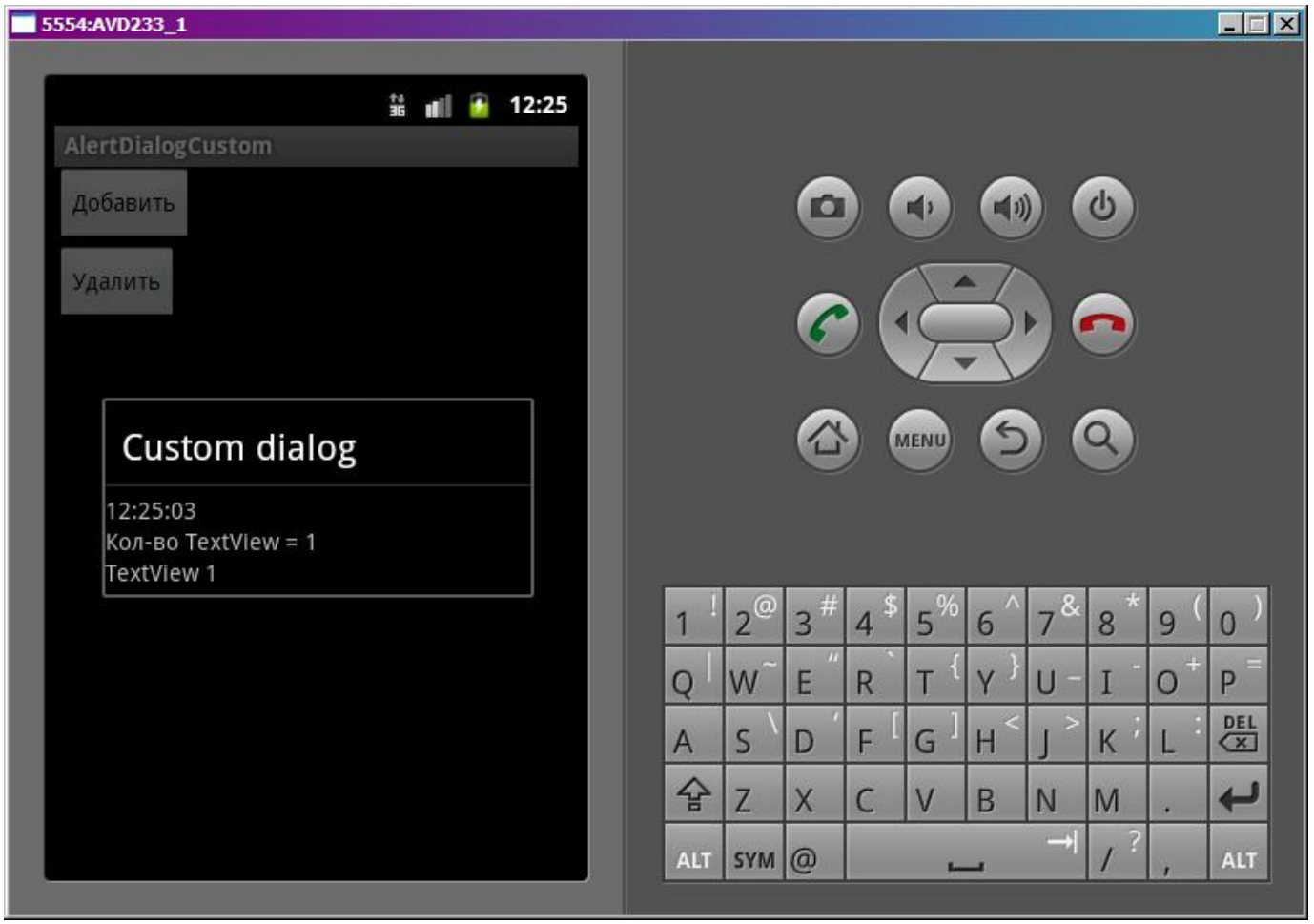
Появился диалог. Отображает время, кол-во добавленных TextView и собственно сами добавленные TextView.

Закрываем диалог кнопкой Назад, жмем еще раз добавить



Добавилось еще одно TextView.

Закрываем диалог, жмем кнопку Удалить.



Последнее TextView удалилось.

На следующем уроке:

- рассматриваем обработчики событий диалога
- программно закрываем и показываем диалог

Урок 66. Диалоги. Обработчики и операции

В этом уроке:

- рассматриваем обработчики событий диалога
- программно закрываем и показываем диалог

Мы закрываем диалог нажатием на кнопку, на пункт списка или кнопкой Назад. Давайте рассмотрим, какие есть программные способы закрытия. Также узнаем, какие обработчики диалога можно использовать, чтобы отследить закрытие.

Создадим проект:

Project name: P0661_AlertDialogOperations
Build Target: Android 2.3.3
Application name: AlertDialogOperations
Package name: ru.startandroid.develop.p0661alertdialogoperations
Create Activity: MainActivity

В **strings.xml** пропишем тексты:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="dialog">Диалог</string>
    <string  name="app_name">P0661_AlertDialogOperations</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/button1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/dialog"
         android:onClick="onclick">
    </Button>
</LinearLayout>
```

MainActivity.java:

```
package ru.startandroid.develop.p0661alertdialogoperations;

import android.app.Activity;
```

```
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnCancelListener;
import android.content.DialogInterface.OnDismissListener;
import android.content.DialogInterface.OnShowListener;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";
  final int DIALOG = 1;

  Dialog dialog;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  @Override
  protected Dialog onCreateDialog(int id) {
    if (id == DIALOG) {
      Log.d(LOG_TAG, "Create");
      AlertDialog.Builder adb = new AlertDialog.Builder(this);
      adb.setTitle("Title");
      adb.setMessage("Message");
      adb.setPositiveButton("OK", null);
      dialog = adb.create();

      // обработчик отображения
      dialog.setOnShowListener(new OnShowListener() {
        public void onShow(DialogInterface dialog) {
          Log.d(LOG_TAG, "Show");
        }
      });

      // обработчик отмены
      dialog.setOnCancelListener(new OnCancelListener() {
        public void onCancel(DialogInterface dialog) {
          Log.d(LOG_TAG, "Cancel");
        }
      });

      // обработчик закрытия
      dialog.setOnDismissListener(new OnDismissListener() {
        public void onDismiss(DialogInterface dialog) {
          Log.d(LOG_TAG, "Dismiss");
        }
      });
      return dialog;
    }
    return super.onCreateDialog(id);
  }

  public void onclick(View v) {
```

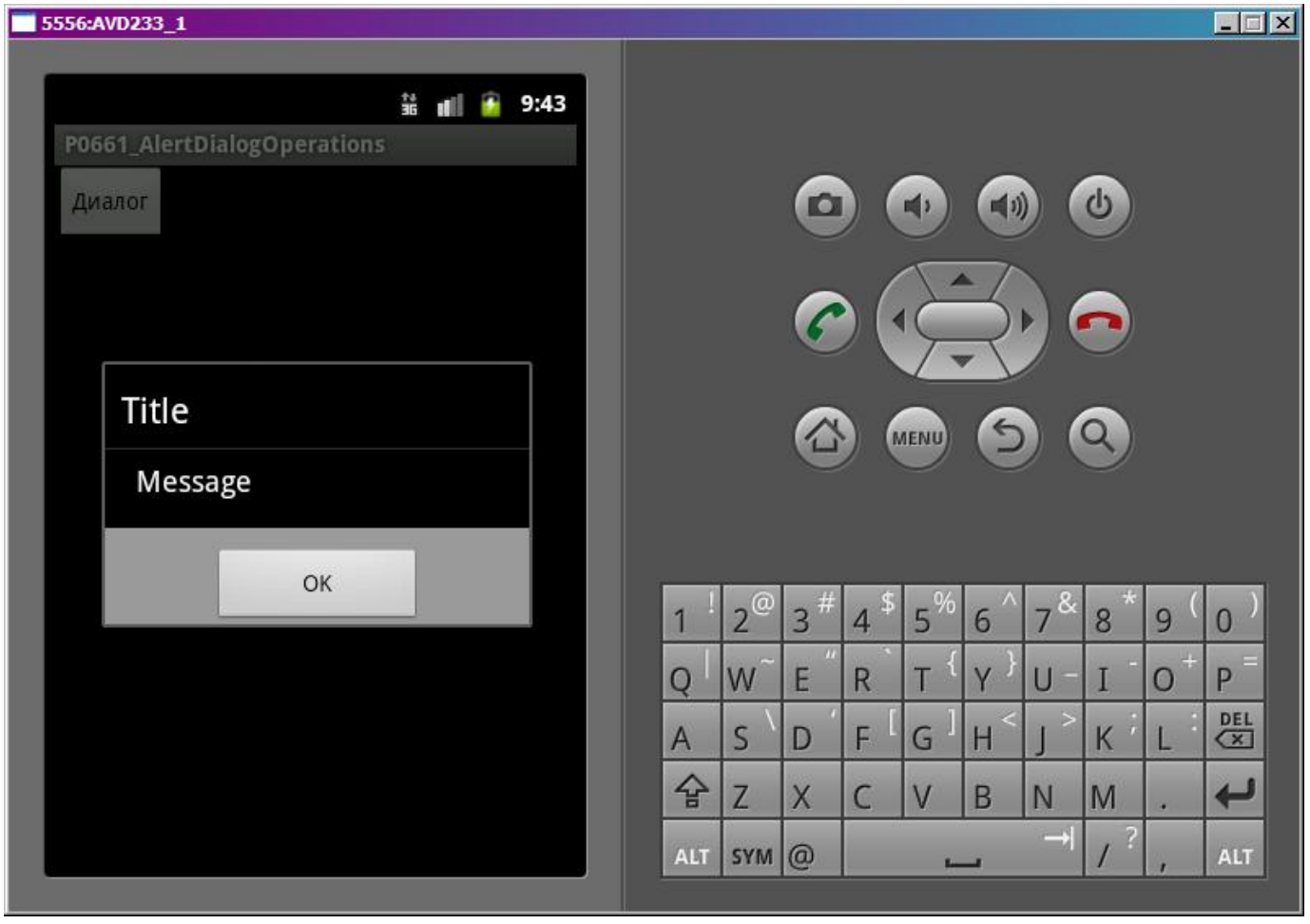
```
      showDialog(DIALOG);
    }
  }
```

Код в основном должен быть понятен по прошлым урокам. Создаем диалог, настраиваем заголовок, сообщение и одну кнопку без обработчика (нам он сейчас не нужен). Далее для диалога указываем три обработчика: отображения, отмены и закрытия диалога. Все они пишут о себе в лог.

onclick – обработчик кнопки из main.xml. Здесь мы просто запускаем диалог.

Обработчики

Давайте смотреть, когда и какие обработчики событий диалога будут срабатывать. Все сохраним и запустим. Жмем кнопку Диалог, появляется диалог.



В логе видим:

Create
Show

Диалог **создался** и сработал **обработчик отображения** диалога. Нажмем кнопку OK. Диалог закрылся, а лог показал следующее:

Dismiss

Сработал **обработчик закрытия** диалога.

Теперь еще раз запустим диалог кнопкой Диалог. В логе видим:

Show

Метод onCreateDialog не отработал, т.к. диалог уже создан. Это мы подробно рассматривали в прошлых уроках. Сработал обработчик отображения.

Для закрытия диалога нажмем кнопку Back (Назад) на эмуляторе. В логе появились следующие строки:

Cancel
Dismiss

Перед обработчиком закрытия (Dismiss) сработал **обработчик отмены** (Cancel), т.к. диалог был отменен.

Операции

Разберем программные методы управления диалогом. Для этого немного изменим код MainActivity.java. Добавим два пустых пока метода **method1** и **method2**, и перепишем **onclick**:

```
void method1() {
}

void method2() {
}

public void onclick(View v) {
  showDialog(DIALOG);

  Handler h = new Handler();

  h.postDelayed(new Runnable() {
    public void run() {
      method1();
    }
  }, 2000);

  h.postDelayed(new Runnable() {
    public void run() {
      method2();
    }
  }, 4000);
}
```

Handler мы пока не проходили, его понимать необязательно. Сейчас просто надо принять, что вся эта конструкция в **onclick** покажет диалог, затем через 2 секунды выполнит метод method1 и еще через 2 секунды выполнит метод method2. Т.е. получится такая последовательность:

отображение диалога
2 сек
выполнение method1
2 сек
выполнение method2

DISMISS

Мы будем работать с method1 и method2. Начнем с метода dismiss – он закрывает диалог. Перепишем метод method1:

```
void method1() {
  dialog.dismiss();
}
```

method2 пока не трогаем. Запустим приложение и нажмем кнопку Диалог. Диалог появился, повисел две секунды и закрылся. Это нам обеспечил **dismiss**, вызванный через 2 секунды после отображения диалога. Смотрим лог:

Create
Show
Dismiss

Все верно. Диалог создался, отобразился и закрылся. Обратите внимание на время записей в логе. Между *Show* и *Dismiss* должно быть примерно 2 секунды.

CANCEL

Теперь используем метод cancel. Перепишем method1:

```
void method1() {
  dialog.cancel();
}
```

Все сохраним, запустим приложение и вызовем диалог. Снова диалог появился и закрылся через две секунды. Сработал метод **cancel**. Логи:

Create
Show
Cancel
Dismiss

Все так же, как при закрытии диалога кнопкой Back.

HIDE

Снова перепишем method1, используя метод hide:

```
void method1() {
  dialog.hide();
}
```

Запустим приложение, вызовем диалог. Он отобразился и закрылся. Смотрим лог:

Create
Show

На этот раз **обработчик закрытия** не сработал. Диалог просто скрылся. Зачем это нужно, я не знаю, но метод такой есть, поэтому я рассказал о нем.

Управление из Activity

Мы работали напрямую с объектом Dialog и вызывали его методы. Есть еще другой способ. Сначала немного теории о механизме взаимодействия Activity и диалога. Когда мы первый выполняем метод showDialog, мы передаем туда ID. Это ID далее передается в onCreateDialog. В итоге onCreateDialog возвращает созданный диалог, и Activity для себя увязывает его с ID. И если мы захотим обратиться к этому диалогу, нам нужен будет только ID, Activity сама по нему определит, какой диалог нам нужен.

Когда мы, например, следующие разы вызываем showDialog, мы передаем туда ID, но диалог не создается. Activity по ID находит ранее созданный диалог и показывает его. У Activity также есть методы по закрытию диалога – это dismissDialog и removeDialog. Первый просто закрывает диалог, а второй закрывает и заставляет Activity забыть про него. Т.е. когда мы в след.раз захотим показать этот диалог, Activity будет заново создавать его, а не брать уже готовый. Проверим это.

DISMISSDIALOG

Перепишем методы:

```
void method1() {
  dismissDialog(DIALOG);
}

void method2() {
  showDialog(DIALOG);
}
```

Все сохраним и запустим. Вызовем диалог и ждем. Диалог отобразился, через 2 секунды закрылся, и еще через 2 снова открылся. Смотрим лог:

Create
Show
Dismiss
Show

Когда диалог отобразился второй раз, не сработал метод его создания, т.к. Activity использовало созданный при первом вызове объект.

REMOVEDIALOG

Перепишем метод method1:

```
void method1() {
  removeDialog(DIALOG);
}
```

Будем не только закрывать диалог, но и «забывать» его. method2 оставляем без изменений, он будет показывать диалог.

Запустим приложение, запустим диалог и ждем. Диалог открылся, закрылся и открылся снова. Смотрим лог:

Create
Show
Dismiss

Create
Show

Но на этот раз при втором показе он снова создавался, т.к. Activity его забыло благодаря методу removeDialog.

У объекта Dialog есть еще метод show. Чем он отличается от метода Activity showDialog? show просто покажет созданный диалог, а showDialog, начинает проверять был ли уже создан диалог, создает его, если необходимо, и вызывает для него метод onPrepareDialog.

На следующем уроке:

- работаем с ProgressDialog

Урок 67. Диалоги. ProgressDialog

В этом уроке:

- работаем с ProgressDialog

ProgressDialog позволяет показать пользователю, что идет какая-либо операция и надо подождать. Он бывает двух видов: просто вращающийся круг и полоса-индикатор, которая показывает процент выполнения. Сделаем приложение, которое будет показывать оба вида.

Создадим проект:

Project name: P0671_ProgressDialog
Build Target: Android 2.3.3
Application name: ProgressDialog
Package name: ru.startandroid.develop.p0671progressdialog
Create Activity: MainActivity

В **strings.xml** пропишем тексты:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="dflt">Обычный</string>
    <string  name="horiz">Горизонтальный</string>
    <string  name="app_name">ProgressDialog</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnDefault"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/dflt"
         android:onClick="onclick">
    </Button>
    <Button
         android:id="@+id/btnHoriz"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/horiz"
         android:onClick="onclick">
    </Button>
```

```
</LinearLayout>
```

Две кнопки: одна покажет диалог с крутящимся кругом (ProgressDialog по умолчанию), другая - с горизонтальной полосой индикатором

MainActivity.java:

```
package ru.startandroid.develop.p0671progressdialog;

import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;

public class MainActivity extends Activity {

  ProgressDialog pd;
  Handler h;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

  }

  public void onclick(View v) {
    switch (v.getId()) {
    case R.id.btnDefault:
      pd = new ProgressDialog(this);
      pd.setTitle("Title");
      pd.setMessage("Message");
      // добавляем кнопку
      pd.setButton(Dialog.BUTTON_POSITIVE, "OK", new OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
        }
      });
      pd.show();
      break;
    case R.id.btnHoriz:
      pd = new ProgressDialog(this);
      pd.setTitle("Title");
      pd.setMessage("Message");
      // меняем стиль на индикатор
      pd.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
      // устанавливаем максимум
      pd.setMax(2148);
      // включаем анимацию ожидания
      pd.setIndeterminate(true);
      pd.show();
      h = new Handler() {
```

```
        public void handleMessage(Message msg) {
          // выключаем анимацию ожидания
          pd.setIndeterminate(false);
          if (pd.getProgress() < pd.getMax()) {
            // увеличиваем значения индикаторов
            pd.incrementProgressBy(50);
            pd.incrementSecondaryProgressBy(75);
            h.sendEmptyMessageDelayed(0, 100);
          } else {
            pd.dismiss();
          }
        }
      };
      h.sendEmptyMessageDelayed(0, 2000);
      break;
    default:
      break;
    }
  }
}
```

Для первого диалога все почти как обычно. Сначала настраиваем заголовок и сообщение. Затем добавляем кнопку. Здесь немного по-другому, чем с AlertDialog.Builder. Мы используем метод setButton. На вход передаем тип кнопки, текст и обработчик. Тип кнопки определяется константами: BUTTON_POSITIVE, BUTTON_NEGATIVE, BUTTON_NEUTRAL. В обработчике я ничего не пишу, мне он не нужен сейчас. Далее показываем диалог методом show.

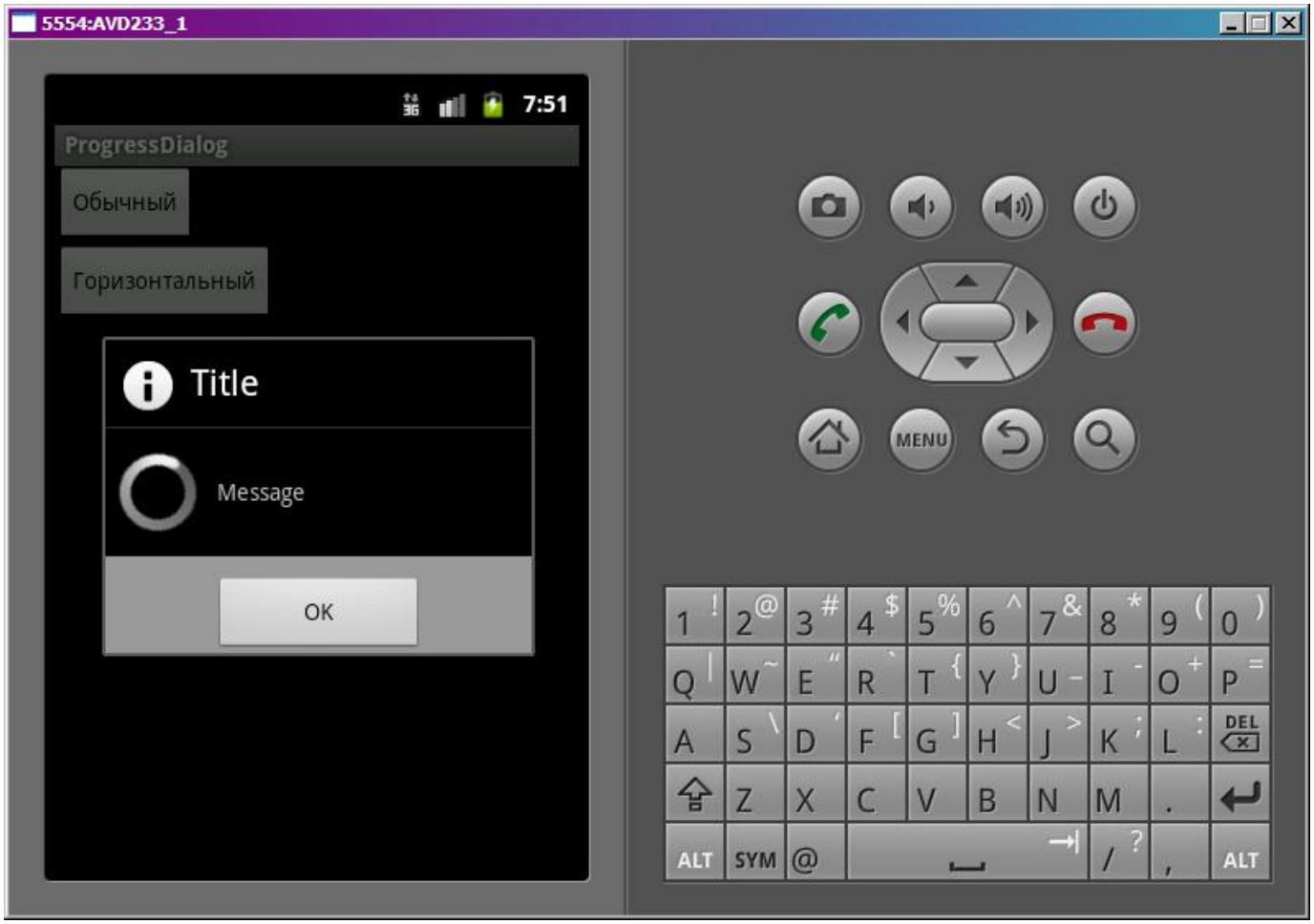
Для второго диалога указываем, что он будет индикатором. Это делается с помощью метода setProgressStyle. Далее задаем заголовок и сообщение. Кнопку не добавляем, но если вдруг нужна будет, то это делается полностью аналогично первому диалогу. Метод setMax устанавливает максимальное значение заполнения индикатора. Значение 2148 я выбрал случайно. Метод setIndeterminate включает анимацию индикатора, имитирующую ожидание. Метод show показывает диалог.

Далее снова приходится использовать незнакомый нам Handler. Понимать его действие сейчас необязательно, позже мы его еще будем проходить. Сейчас я просто распишу на словах, что он делает. Handler ждет 2 секунды (просто имитация, например, подключения к серверу), выключает анимацию ожидания (setIndeterminate), затем каджые 100 миллисекунд увеличивает значение основного (incrementProgressBy) и дополнительного (incrementSecondaryProgressBy) индикатора, пока основной индикатор (getProgress) не достигнет максимума (getMax, в нашем случае = 2148). После этого диалог закрываем (dismiss).

Пример дополнительного индикатора вы могли видеть на YouTube. Когда смотрите видео, там снизу по мере воспроизведения заполняется основной индикатор, а правее его заполняется еще один, затемненный, который показывает, сколько видео закэшировалось. Разумеется, если он вам не нужен, вы просто его не заполняете. Будет отображаться только основной.

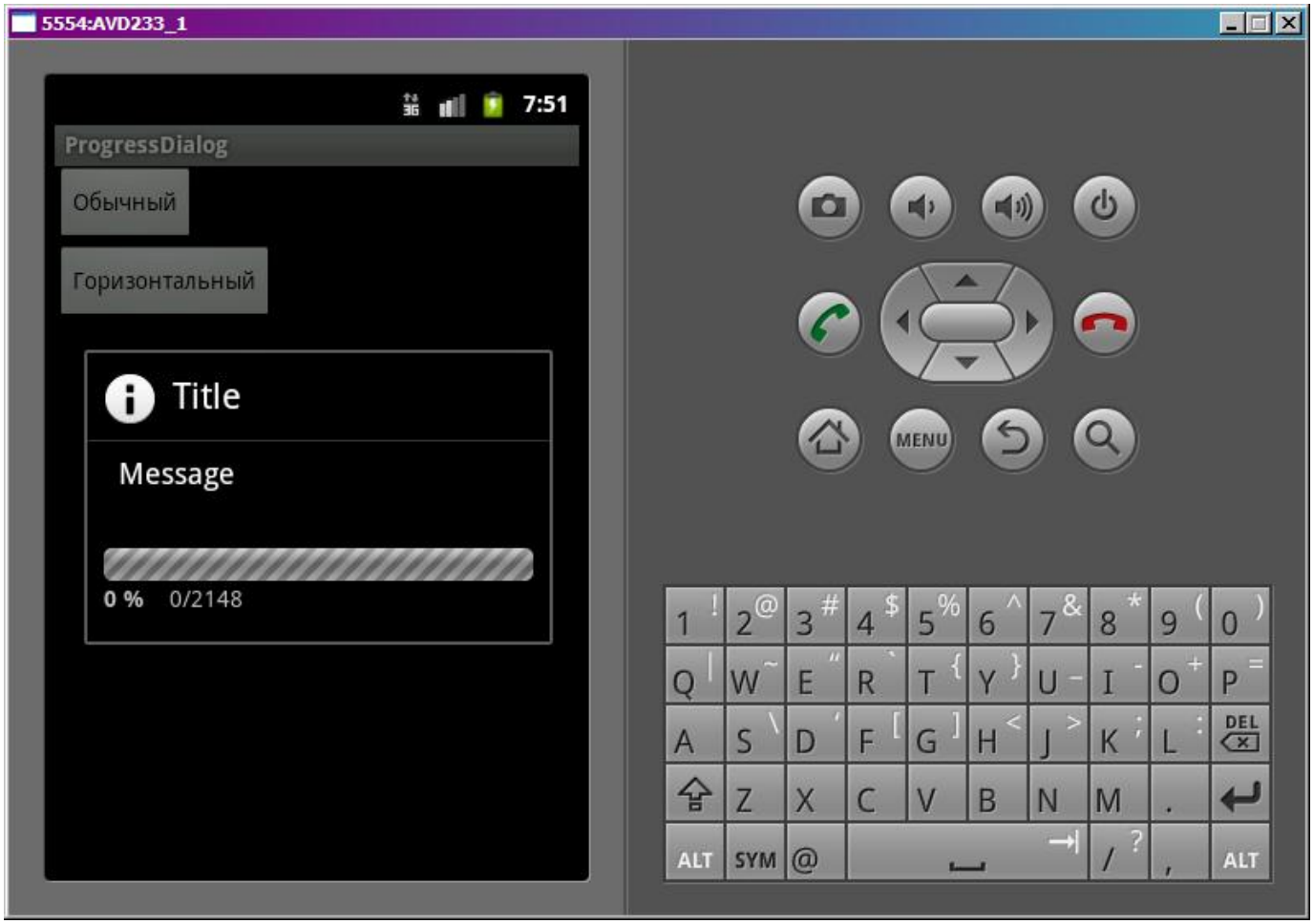
Все сохраним и запустим.

Вызовем первый диалог

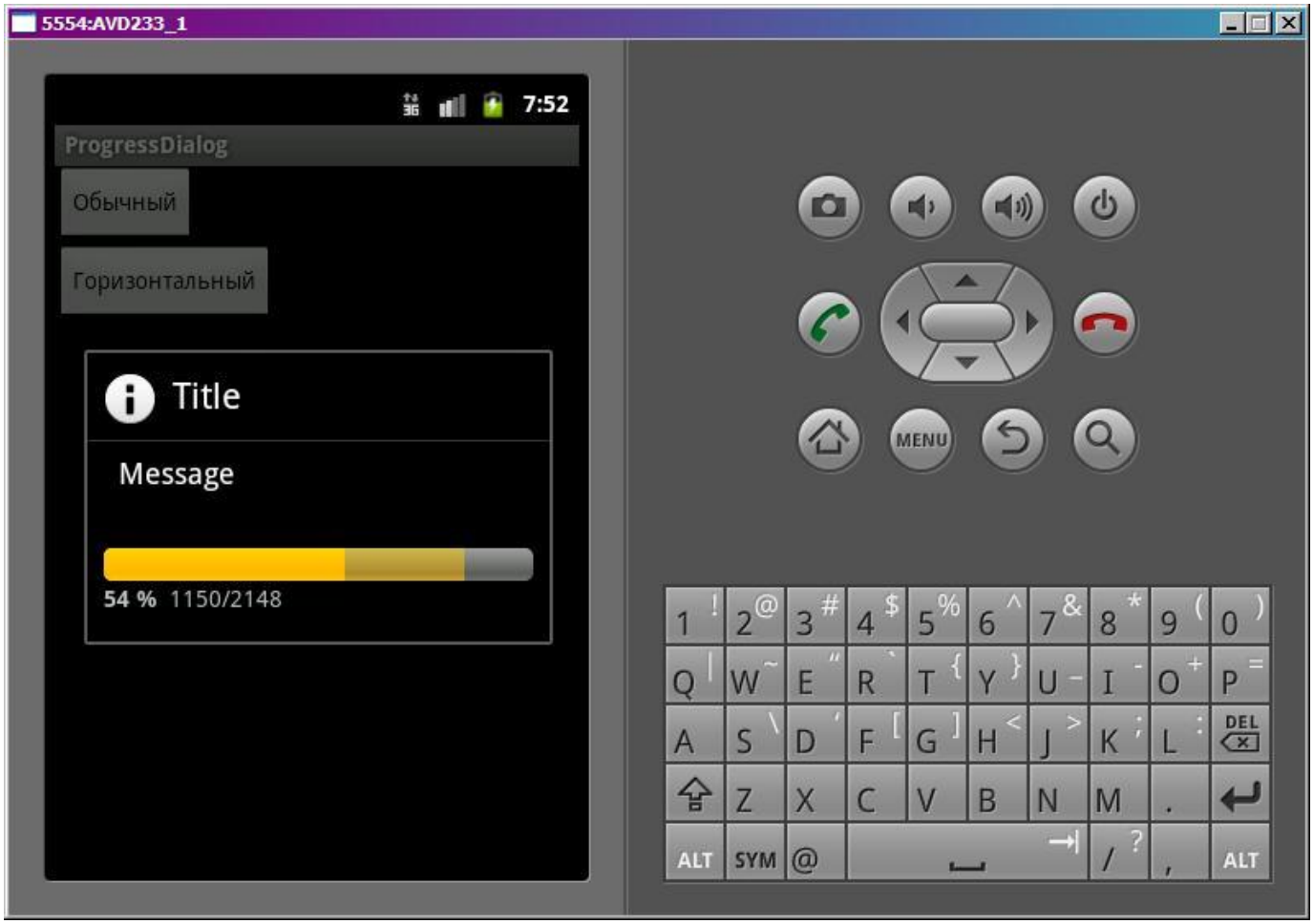


Бесконечно крутящийся круг показывает, что процесс идет. Но сколько осталось – непонятно. Не всегда можно спрогнозировать оставшееся время или показать процент выполнения задачи.

Закроме первый диалог и откроем второй. Он две секунды повисит с анимацией ожидания



А затем начнет заполнять индикатор (основной и дополнительный).



Когда основной индикатор заполнится, диалог закроется.

Думаю, по диалогам тему можно закрывать. Мы рассмотрели их достаточно подробно. То, что пока непонятен Handler – это ничего страшного, через несколько уроков возьмемся и за него. Если вдруг чего осталось непонятным, велкам на форум, будем разбираться)

На следующем уроке:

- знакомимся с Parcel

Урок 68. Немного о Parcel

В этом уроке:

- знакомимся с Parcel

Сам по себе Parcel мне никогда еще использовать не приходилось и не знаю, придется ли. Меня он заинтересовал, когда я начал разбираться с интерфейсом Parcelable. Этот интерфейс используется при передаче объектов через Intent и мне стало интересно, как создавать свои объекты с поддержкой такой передачи. В итоге я немного разобрался в Parcel и Parcelable, хотя понял далеко не все. Попробую теперь рассказать об этом.

Parcel – это контейнер для передачи данных. У него есть куча методов для помещения и извлечения данных. В этом уроке рассмотрим самые простейшие из них.

Создадим проект:

Project name: P0681_Parcel
Build Target: Android 2.3.3
Application name: Parcel
Package name: ru.startandroid.develop.p0681parcel
Create Activity: MainActivity

В этом уроке экран нам не понадобится, main.xml оставляем без изменений. Работать будем с логом.

Кодим в **MainActivity.java**:

```
package ru.startandroid.develop.p0681parcel;

import android.app.Activity;
import android.os.Bundle;
import android.os.Parcel;
import android.util.Log;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";
  Parcel p;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    writeParcel();
    readParcel();
  }

  void writeParcel() {
    p = Parcel.obtain();

    byte b = 1;
    int i = 2;
    long l = 3;
    float f = 4;
    double d = 5;
    String s = "abcdefgh";
```

```
        logWriteInfo("before writing");
        p.writeByte(b);
        logWriteInfo("byte");
        p.writeInt(i);
        logWriteInfo("int");
        p.writeLong(l);
        logWriteInfo("long");
        p.writeFloat(f);
        logWriteInfo("float");
        p.writeDouble(d);
        logWriteInfo("double");
        p.writeString(s);
        logWriteInfo("String");
    }

    void logWriteInfo(String txt) {
        Log.d(LOG_TAG, txt + ": " + "dataSize = " + p.dataSize());
    }

    void readParcel() {
    }

    void logReadInfo(String txt) {
    }

}
```

Метод **writeParcel** – получаем экземпляр Parcel, описываем набор переменных и пишем их в Parcel, используя для этого соответствующие методы. После каждой записи выводим в лог информацию о Parcel, используя метод logWriteInfo.

Метод **logWriteInfo** пишет в лог данные о Parcel. dataSize – это объем записанных данных.

Методы **readParcel** и **logReadInfo** – пока пустые. Позже заполним.

Все сохраняем и запускаем приложение. Смотрим лог.

before writing: dataSize = 0
byte: dataSize = 4
int: dataSize = 8
long: dataSize = 16
float: dataSize = 20
double: dataSize = 28
String: dataSize = 52

Разбираем по порядку.

before writing: перед записью у нас размер данных равен 0. Записали ***byte***: dataSize = 4 (для записи данных типа byte использовались 4 байта). Записали ***int***: dataSize = 8 (для записи данных типа int использовались еще 4 байта в дополнение к ранее заполненным 4 байтам для byte). Записали ***long***: dataSize = 16 (для записи long использовались еще 8 байтов в дополнение к ранее заполненным 8 байтам для byte и int). И т.д. В итоге видим, что **dataSize** показывает, сколько всего занято байт.

Обратите внимание, что типы **int**, **long**, **float** и **double** заняли столько байт, сколько они действительно занимают в Java – соответственно 4, 8, 4 и 8. **byte** – вместо одного байта почему-то занял целых 4. А **String** под каждый символ использует два байта, но пишет еще служебную информацию, поэтому получается больше.

Теперь попробуем прочесть то, что записали. Заполним пустые методы чтения:

```
void readParcel() {
  logReadInfo("before reading");
  p.setDataPosition(0);
  logReadInfo("byte = " + p.readByte());
  logReadInfo("int = " + p.readInt());
  logReadInfo("long = " + p.readLong());
  logReadInfo("float = " + p.readFloat());
  logReadInfo("double = " + p.readDouble());
  logReadInfo("string = " + p.readString());
}

void logReadInfo(String txt) {
  Log.d(LOG_TAG, txt + ": " + "dataPosition = " + p.dataPosition());
}
```

В методе **readParcel** мы устанавливаем (метод setDataPosition) позицию в 0, т.к. нам нужно читать с начала. Читаем данные в том же порядке, как и записывали: byte, int, long, float, double, String. В лог выводим результат чтения и текущую позицию (dataPosition).

Все сохраним, запустим приложение и смотрим лог.

Первые строки лога про запись нам уже знакомы. Нас интересуют строки чтения.

before reading: dataPosition = 52
byte = 1: dataPosition = 4
int = 2: dataPosition = 8
long = 3: dataPosition = 16
float = 4.0: dataPosition = 20
double = 5.0: dataPosition = 28
string = abcdefgh: dataPosition = 52

Перед тем, как мы установим позицию в 0 (***before reading***), видим, что она равна 52. Там она находится после записи. Каждая запись данных перемещает позицию на кол-во, равное размеру записываемых данных. Размер всех последовательно записанных данных у нас составил 52, и позиция соответственно переместилась в 52. Вы можете в качестве эксперимента выводить в лог позицию после каждой записи данных. Я же вывожу только для процедур чтения.

Итак, мы устанавливаем позицию в 0 и начинаем читать данные. Прочли значение ***byte***, оно равно 1, как мы и записывали. Позиция сместилась на размер прочтенного значения, и теперь мы будем читать с позиции 4. Читаем ***int***, оно равно 2, позиция сместилась и равна 8. И т.д.

Все значения, которые мы последовательно записывали, мы в том же порядке считали. Здесь надо понимать, что если вы записали **int**, а читать потом будете **double**, то результат получится не тот, что нужен. Т.к. **int** пишет 4 байта, а **double** считывает 8. Тем самым он залезет на следующий записанный тип и возьмет из него недостающие 4 байта. Получится каша. Поэтому тут надо быть аккуратным.

Вы всегда можете установить нужную вам позицию и считать хранимое значение. Главное – знать, какой тип там хранится. Например, у нас сейчас при записи double пишется с позиции 20. Поэтому мы можем перевести позицию в 20 и выполнить readDouble. Мы успешно получим записанный туда double, а позиция станет равна 28.

Если вы хотите глянуть содержимое Parcel можно использовать его метод marshall(), он вернет массив записанных в Parcel байтов.

Вот такой краткий экскурс. Эти знания понадобятся для понимания следующего урока.

На следующем уроке:

- добавляем своему объекту поддержку Parcelable
- передаем объект с помощью Intent

Урок 69. Передаем Parcelable объекты с помощью Intent

В этом уроке:

- добавляем объекту поддержку Parcelable
- передаем с помощью Intent

С Parcel мы немного поработали на прошлом уроке. Этих знаний хватит, чтобы понять, как реализовать в своем объекте интерфейс Parcelable. Создадим свой объект, реализуем в нем интерфейс Parcelable и попробуем передать в другое Activity через Intent.

Создадим проект:

Project name: P0691_Parcelable
Build Target: Android 2.3.3
Application name: Parcelable
Package name: ru.startandroid.develop.p0691parcelable
Create Activity: MainActivity

В **strings.xml** пропишем тексты:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="main">MainActivity</string>
    <string  name="second">SecondActivity</string>
    <string  name="send">Send</string>
    <string  name="app_name">Parcelable</string>
</resources>
```

В **main.xml** нарисуем кнопку:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <TextView
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="@string/main">
    </TextView>
    <Button
         android:id="@+id/button1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:onClick="onclick"
         android:text="@string/send">
    </Button>
</LinearLayout>
```

Перед тем как кодить MainActivity.java, создадим свой объект для передачи **MyObject.java**:

```
package ru.startandroid.develop.p0691parcelable;

import android.os.Parcel;
import android.os.Parcelable;
import android.util.Log;

public class MyObject implements Parcelable {

  final static String LOG_TAG = "myLogs";

  public String s;
  public int i;

  // обычный конструктор
  public MyObject(String _s, int _i) {
    Log.d(LOG_TAG, "MyObject(String _s, int _i)");
    s = _s;
    i = _i;
  }

  public int describeContents() {
    return 0;
  }

  // упаковываем объект в Parcel
  public void writeToParcel(Parcel parcel, int flags) {
    Log.d(LOG_TAG, "writeToParcel");
    parcel.writeString(s);
    parcel.writeInt(i);
  }

  public static final Parcelable.Creator<MyObject> CREATOR = new Parcelable.Creator<MyObject>() {
    // распаковываем объект из Parcel
    public MyObject createFromParcel(Parcel in) {
      Log.d(LOG_TAG, "createFromParcel");
      return new MyObject(in);
    }

    public MyObject[] newArray(int size) {
      return new MyObject[size];
    }
  };

  // конструктор, считывающий данные из Parcel
  private MyObject(Parcel parcel) {
    Log.d(LOG_TAG, "MyObject(Parcel parcel)");
    s = parcel.readString();
    i = parcel.readInt();
  }

}
```

Объект сам по себе несложный: пара переменных **s** и **i**, и конструктор. Все остальное используется для реализации Parcelable. Давайте смотреть.

Про метод describeContents ничего сказать не могу. Я не понял, зачем он нужен.

В методе writeToParcel мы получаем на вход Parcel и упаковываем в него наш объект. Т.е., в нашем случае, помещаем туда переменные s и i. flags не используем.

CREATOR типа Parcelable.Creator<MyObject> используется для создания экземпляра нашего MyObject и заполнения его данными из Parcer.

Для этого используется его метод createFromParcel, который мы должны реализовать. На вход нам дается Parcel, а вернуть мы должны готовый MyObject. В нашем примере мы используем здесь конструктор MyObject(Parcel parcel), который реализован чуть дальше.

Смысл метода newArray остался для меня непонятен.

Конструктор MyObject(Parcel parcel) принимает на вход Parcel и заполняет объект данными из него. Этот метод использовался нами чуть ранее в CREATOR.createFromParcel.

Создадим второе Activity, в которое будем передавать объект.

Сначала создаем экран **second.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical">
     <TextView
          android:id="@+id/textView1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/second">
     </TextView>
</LinearLayout>
```

Кодим **SecondActivity.java**:

```
package ru.startandroid.develop.p0691parcelable;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class SecondActivity extends Activity {

  final String LOG_TAG = "myLogs";

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.second);
    Log.d(LOG_TAG, "getParcelableExtra");
    MyObject myObj = (MyObject) getIntent().getParcelableExtra(
        MyObject.class.getCanonicalName());
    Log.d(LOG_TAG, "myObj: " + myObj.s + ", " + myObj.i);
  }

}
```

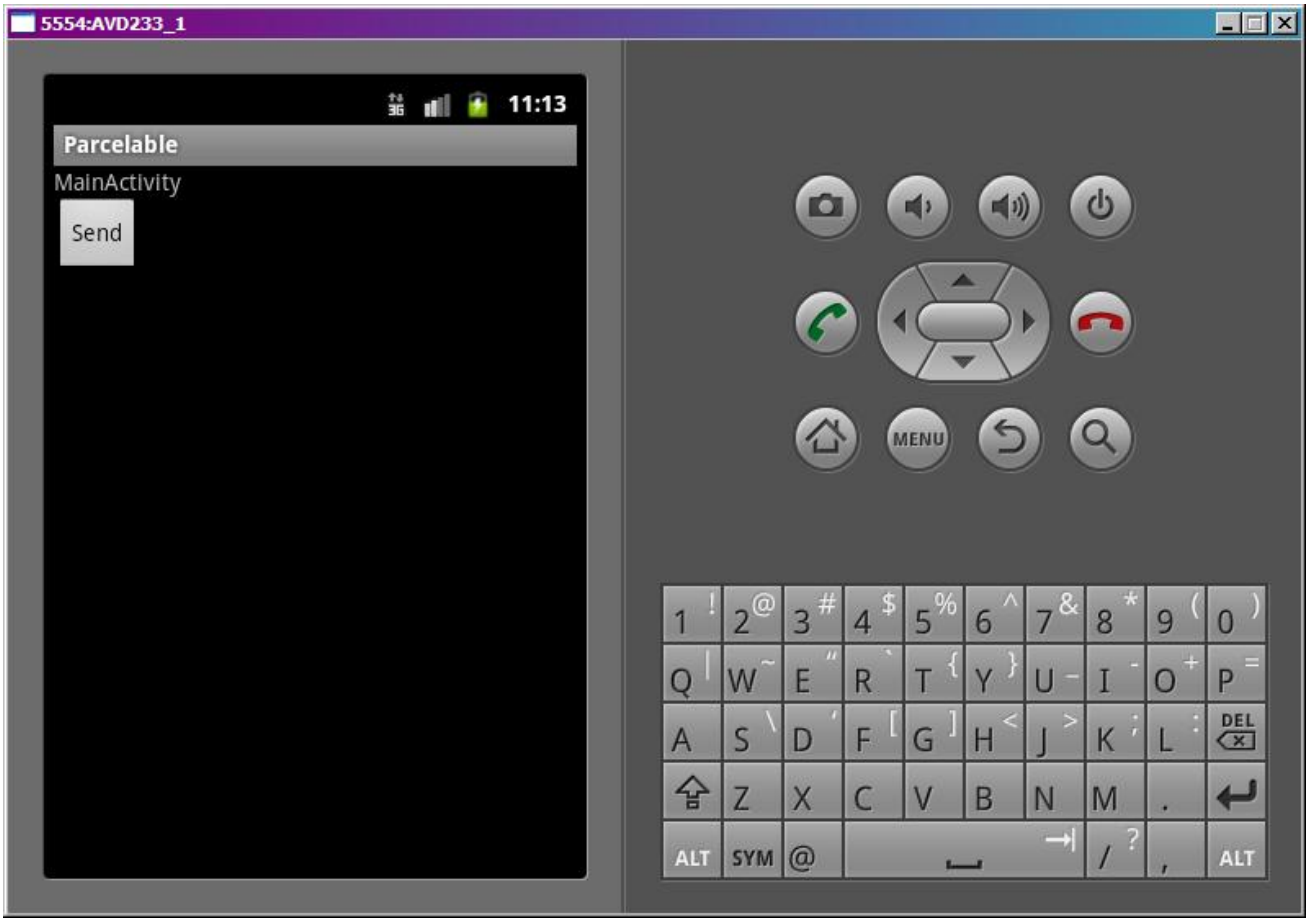
Мы вытаскиваем наш MyObject-объект из Intent и в лог выводим значения s и i.

Кодим **MainActivity.java**:

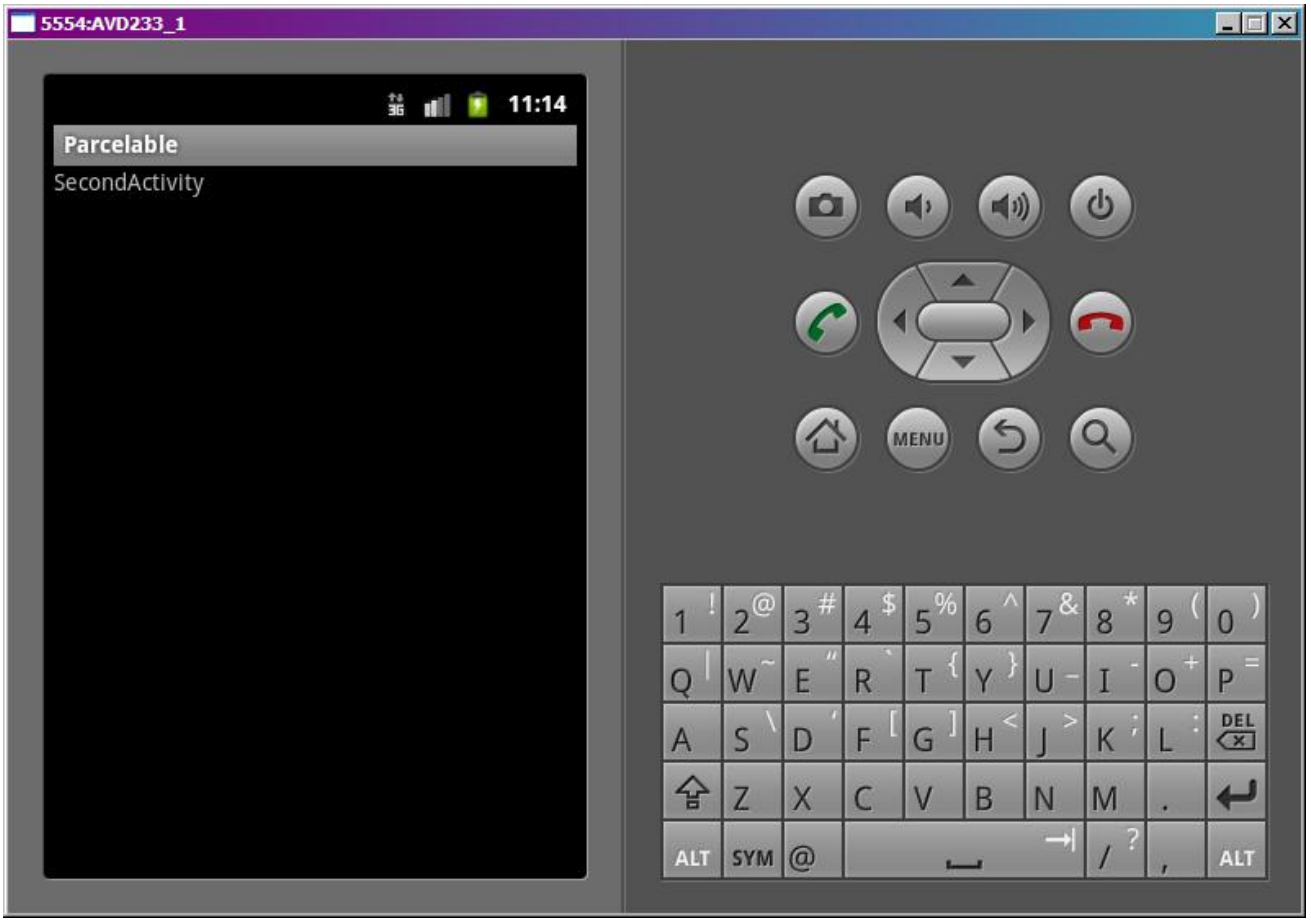
```
package ru.startandroid.develop.p0691parcelable;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  public void onclick(View v) {
    MyObject myObj = new MyObject("text", 1);
    Intent intent = new Intent(this, SecondActivity.class);
    intent.putExtra(MyObject.class.getCanonicalName(), myObj);
    Log.d(LOG_TAG, "startActivity");
    startActivity(intent);
  }
}
```

Создаем Intent, помещаем туда объект MyObject. В качестве ключа используем его имя класса (разумеется, это необязательно, вы можете свое имя использовать). И отправляем Intent с вызовом SecondActivity.

Все сохраним и запустим приложение.



Жмем Send, Intent уходит в SecondActivity



Смотрим лог:

MyObject(String _s, int _i)
startActivity
writeToParcel
getParcelableExtra
createFromParcel
MyObject(Parcel parcel)
myObj: text, 1

Сначала вызвался конструктор ***MyObject(String _s, int _i)*** – это мы создали myObj.

startActivity – начинаем вызов Activity

writeToParcel - мы поместили объект в Intent, и похоже, что при отправке он упаковался в Parcer. Т.к. сам Parcer не знает, как именно упаковать объект, он вызвал метод writeToParcel, где мы реализовали упаковку.

getParcelableExtra – извлекаем объект из Intent

createFromParcel – это был вызван метод CREATOR.createFromParcel, которому на вход дали Parcer, а он должен вернуть MyObject. Этот метод в свою очередь для создания MyObject использует конструктор ***MyObject(Parcel parcel)***, в котором мы расписали, как надо читать Parcer и заполнить объект.

myObj: text, 1 – вывели в лог значения объекта.

Итак. Чтобы нам **передать объект через Intent**, нам надо реализовать в нем интерфейс **Parcelable**. В этом случае Intent без проблем запакует, передаст и распакует наш объект. И я так подозреваю, что делает он это с помощью Parcel. Т.е. в реализации интерфейса **Parcelable** мы полностью описываем **алгоритм упаковки и распаковки** объекта, а Parcel эти алгоритмы потом использует. Т.к. сам он не может знать, как правильно распаковать и создать передаваемый объект.

Если кто разберется, зачем нужны непонятые мною методы – пишите на форуме в ветке этого урока. Я добавлю вашу инфу в урок.

На следующем уроке:

- сохраняем данные при повороте экрана

Урок 70. onSaveInstanceState. Сохранение данных Activity при повороте экрана

В этом уроке:

- сохраняем данные при повороте экрана

Теорию по этому вопросу можно почитать тут. Я здесь вкратце дам вольный перевод.

Когда работа Activity приостанавливается(**onPause** или **onStop**), она остается в памяти и хранит все свои объекты и их значения. И при возврате в Activity, все остается, как было. Но если приостановленное Activity уничтожается, например, при нехватке памяти, то соответственно удаляются и все его объекты. И если к нему снова вернуться, то системе надо заново его создавать и восстанавливать данные, которые были утеряны при уничтожении. Для этих целей Activity предоставляет нам для реализации пару методов: первый позволяет сохранить данные – onSaveInstanceState, а второй – восстановить - onRestoreInstanceState.

Эти методы используются в случаях, когда Activity уничтожается, но есть вероятность, что оно еще будет востребовано в своем текущем состоянии. Т.е. при нехватке памяти или при повороте экрана. Если же вы просто нажали кнопку Back (назад) и тем самым явно сами закрыли Activity, то эти методы не будут выполнены.

Но даже если не реализовать эти методы, у них есть реализация по умолчанию, которая сохранит и восстановит данные в экранных компонентах. Это выполняется для всех экранных компонентов, у которых есть ID.

Создадим простое приложение, чтобы протестить все эти тезисы. Посмотрим, в какой момент вызываются эти методы, попробуем в них что-нить сохранить. Также убедимся, что необходимо вызывать соответствующие методы супер-класса, чтобы сохранялись данные экранных компонентов.

Т.к. нам надо будет поворачивать экран, используйте при разработке **Android 2.2**. В AVD с версией 2.3 поворот глючит.

Создадим проект:

Project name: P0701_SaveInstanceState
Build Target: Android 2.2
Application name: SaveInstanceState
Package name: ru.startandroid.develop.p0701saveinstancestate
Create Activity: MainActivity

В **strings.xml** пропишем тексты:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">SaveInstanceState</string>
    <string  name="count">Count</string>
</resources>
```

В **main.xml** нарисуем кнопку и пару полей для ввода текста:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
```

```
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
          android:id="@+id/button1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onclick"
          android:text="@string/count">
    </Button>
    <EditText
          android:id="@+id/editText1"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:ems="10">
        <requestFocus>
        </requestFocus>
    </EditText>
    <EditText
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:ems="10">
    </EditText>
</LinearLayout>
```

Обратите внимание, что второй EditText без ID.

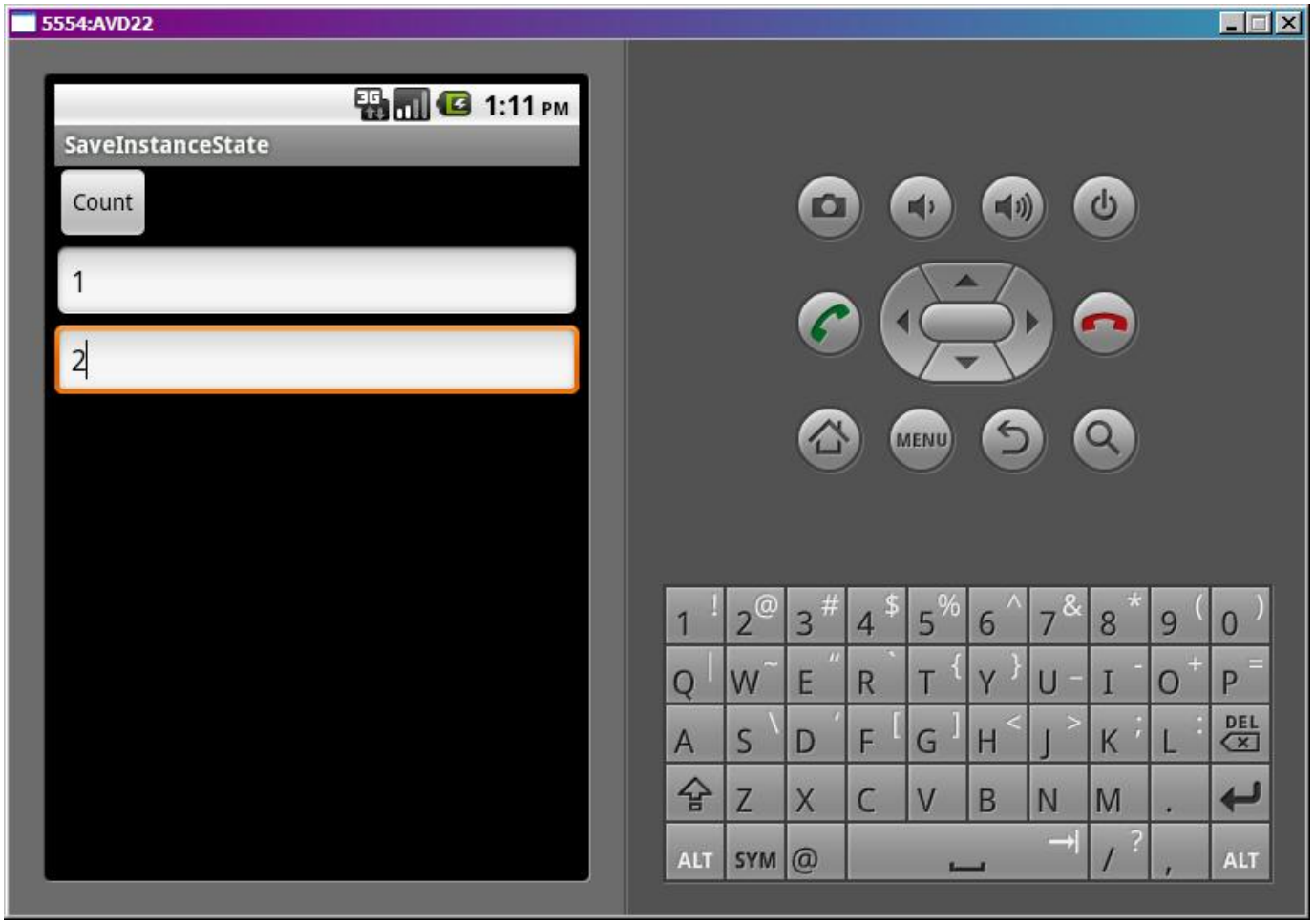
В **MainActivity** будем вызывать все методы Lifecycle и два вышеописанных:

```
package ru.startandroid.develop.p0701saveinstancestate;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";
  int cnt = 0;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.d(LOG_TAG, "onCreate");
  }

  protected void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "onDestroy");
  }

  protected void onPause() {
```

```
    super.onPause();
    Log.d(LOG_TAG, "onPause");
  }

  protected void onRestart() {
    super.onRestart();
    Log.d(LOG_TAG, "onRestart");
  }

  protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    Log.d(LOG_TAG, "onRestoreInstanceState");
  }

  protected void onResume() {
    super.onResume();
    Log.d(LOG_TAG, "onResume ");
  }

  protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    Log.d(LOG_TAG, "onSaveInstanceState");
  }

  protected void onStart() {
    super.onStart();
    Log.d(LOG_TAG, "onStart");
  }

  protected void onStop() {
    super.onStop();
    Log.d(LOG_TAG, "onStop");
  }

  public void onclick(View v) {
  }
}
```

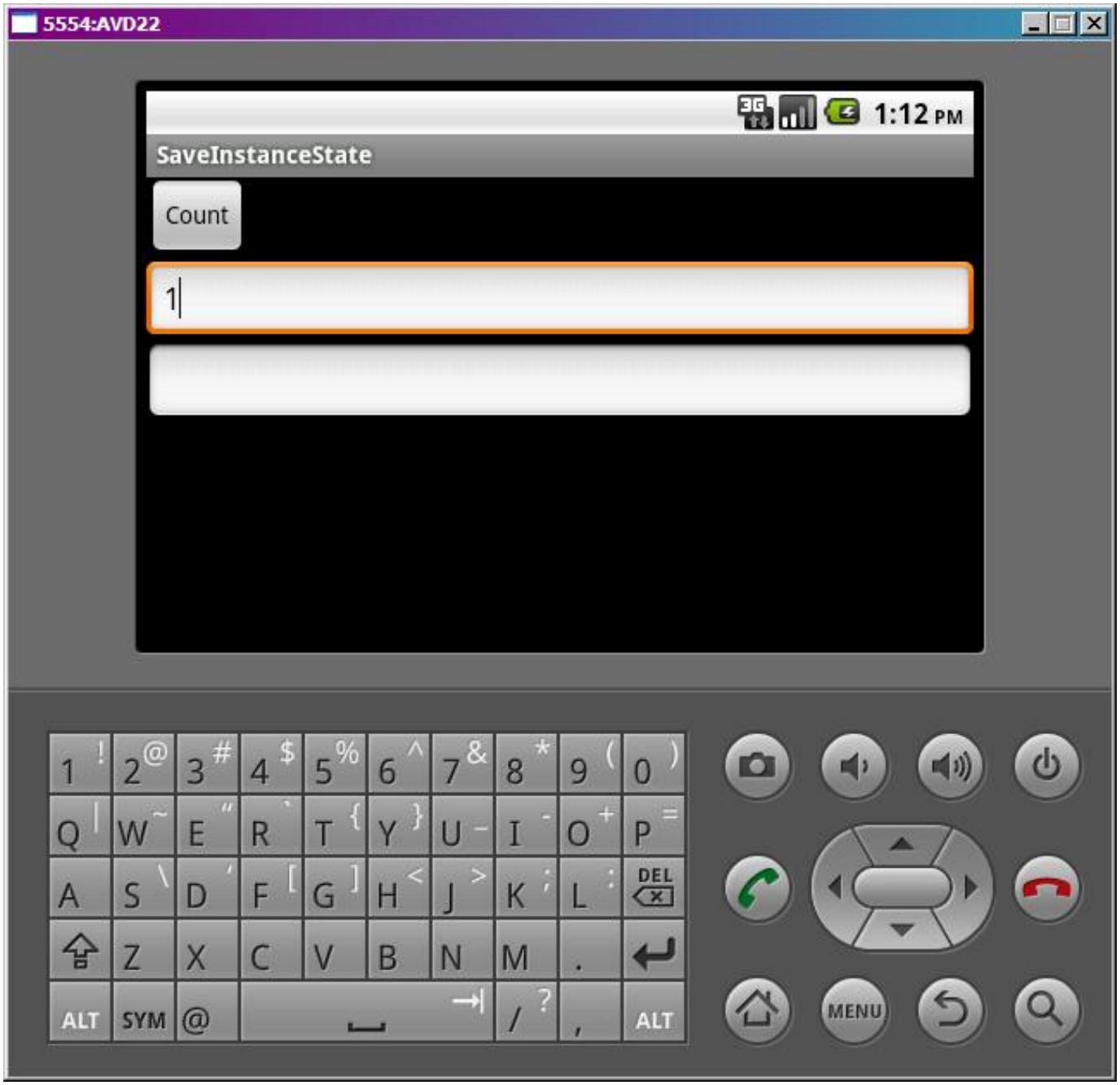
В каждом из них пишем лог, чтобы отследить последовательность вызовов. Метод **onclick** пока не реализуем.

Все сохраним и запустим. Введем в текстовые поля какие-нить данные:



и повернем экран CTRL+F12.

В итоге видим:



Данные в первом поле сохранились при повороте, а во втором пропали. Это произошло потому, что дефолтовые методы сохранения/восстановления умеют работать только с компонентами, которые имеют ID. Посмотрим лог.

onCreate
onStart
onResume

Эти три метода выполнились при запуске.

Затем мы повернули экран:

onSaveInstanceState
onPause
onStop
onDestroy
onCreate
onStart
onRestoreInstanceState
onResume

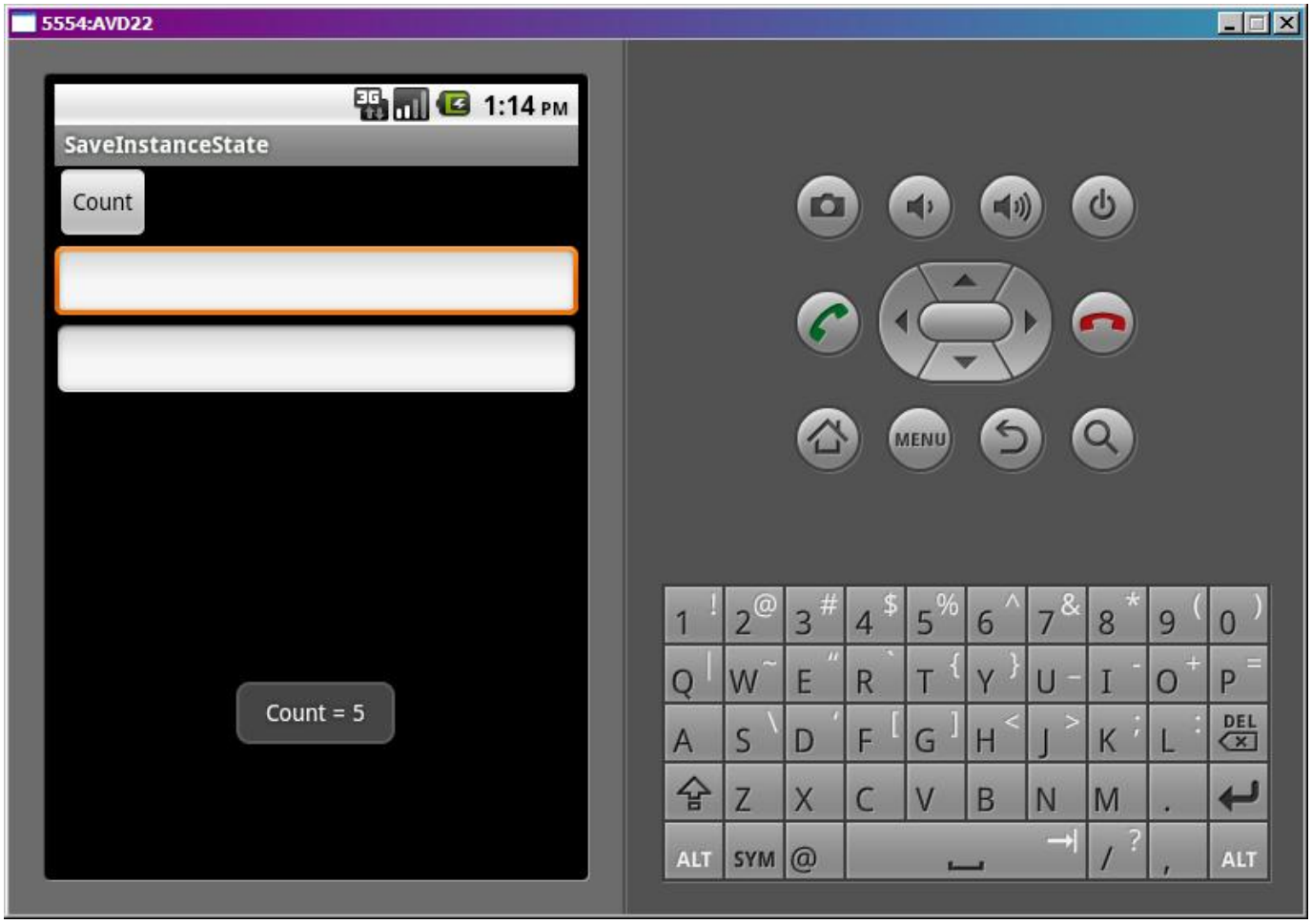
Первым делом вызывается ***onSaveInstanceState***, здесь нам надо будет реализовывать сохранение своих данных.

Далее идет уничтожение Activity (***onPause***, ***onStop***, ***onDestroy***) и создание нового ***onCreate***, ***onStart***. И перед ***onResume*** вызывается метод восстановления данных – ***onRestoreInstanceState***.

Последовательность мы рассмотрели - сохраняются данные перед onPause, а восстанавливаются перед onResume. Попробуем теперь что-нибудь сохранить и восстановить. У нас на экране есть кнопка, будем по ее нажатию увеличивать счетчик нажатий на единицу и выводить всплывающее сообщение с итоговым кол-вом нажатий. Переменная cnt у нас уже есть. Реализуем **onclick**:

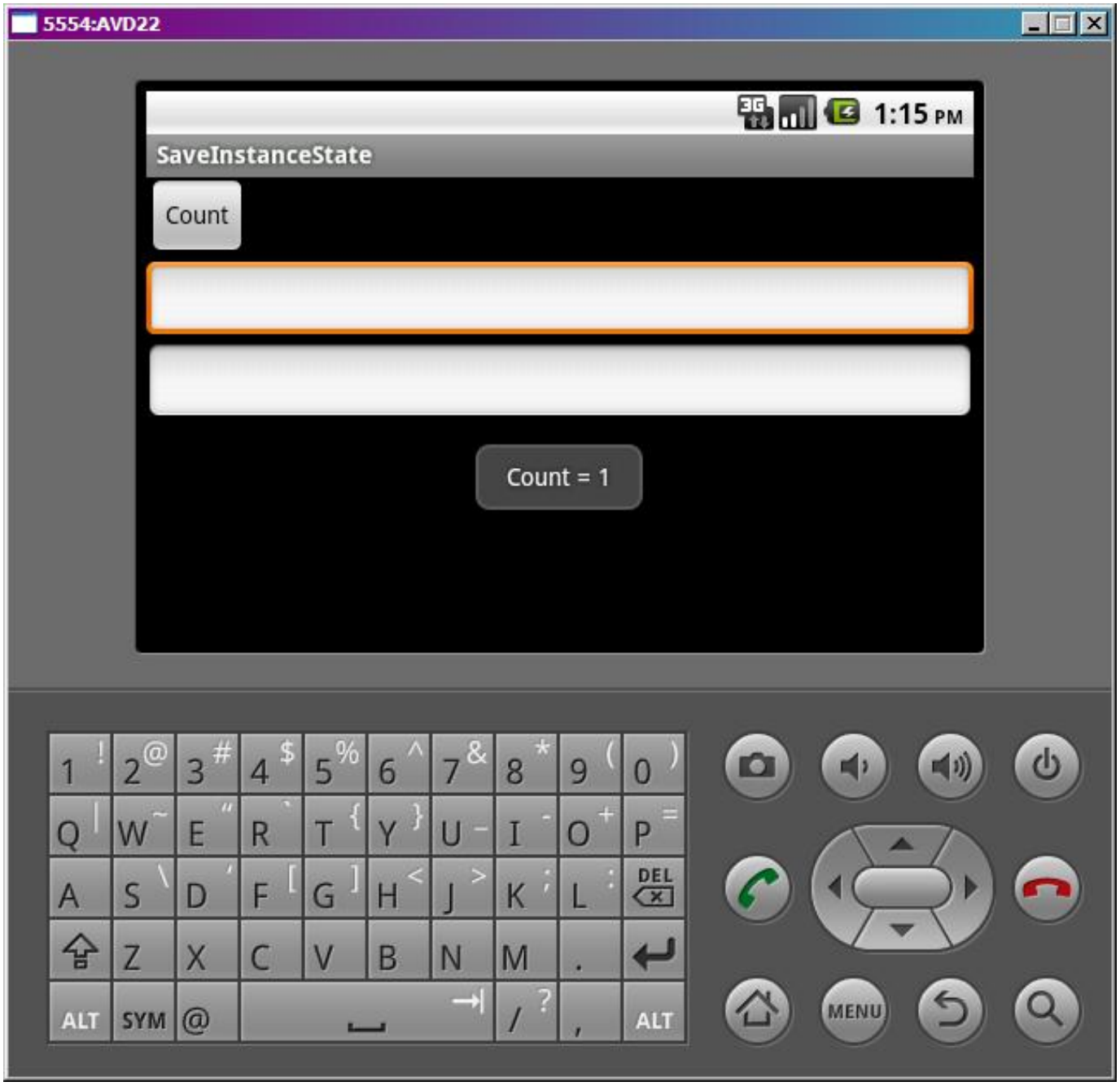
```
public void onclick(View v) {
  Toast.makeText(this, "Count = " + ++cnt, Toast.LENGTH_SHORT).show();
}
```

Повернем эмулятор обратно в вертикальную ориентацию. Запустим приложение, и жмем на кнопку **Count**. Видим сообщение с кол-вом нажатий. Нажмем еще несколько раз, получим, например 5.



Теперь повернем экран и снова нажмем кнопку.

Мы видим, что счетчик сбросился.



Это произошло потому, что текущий объект **Activity** был уничтожен и потерял значения всех переменных, в том числе и **cnt**. При создании нового Activity значение cnt равно 0 и отсчет пошел заново. Давайте это пофиксим. Реализуем метод сохранения **onSaveInstanceState**:

```
protected void onSaveInstanceState(Bundle outState) {
  super.onSaveInstanceState(outState);
  outState.putInt("count", cnt);
  Log.d(LOG_TAG, "onSaveInstanceState");
}
```

В объект **outState** мы пишем значение переменной cnt. Механизм аналогичен помещению данных в Intent.

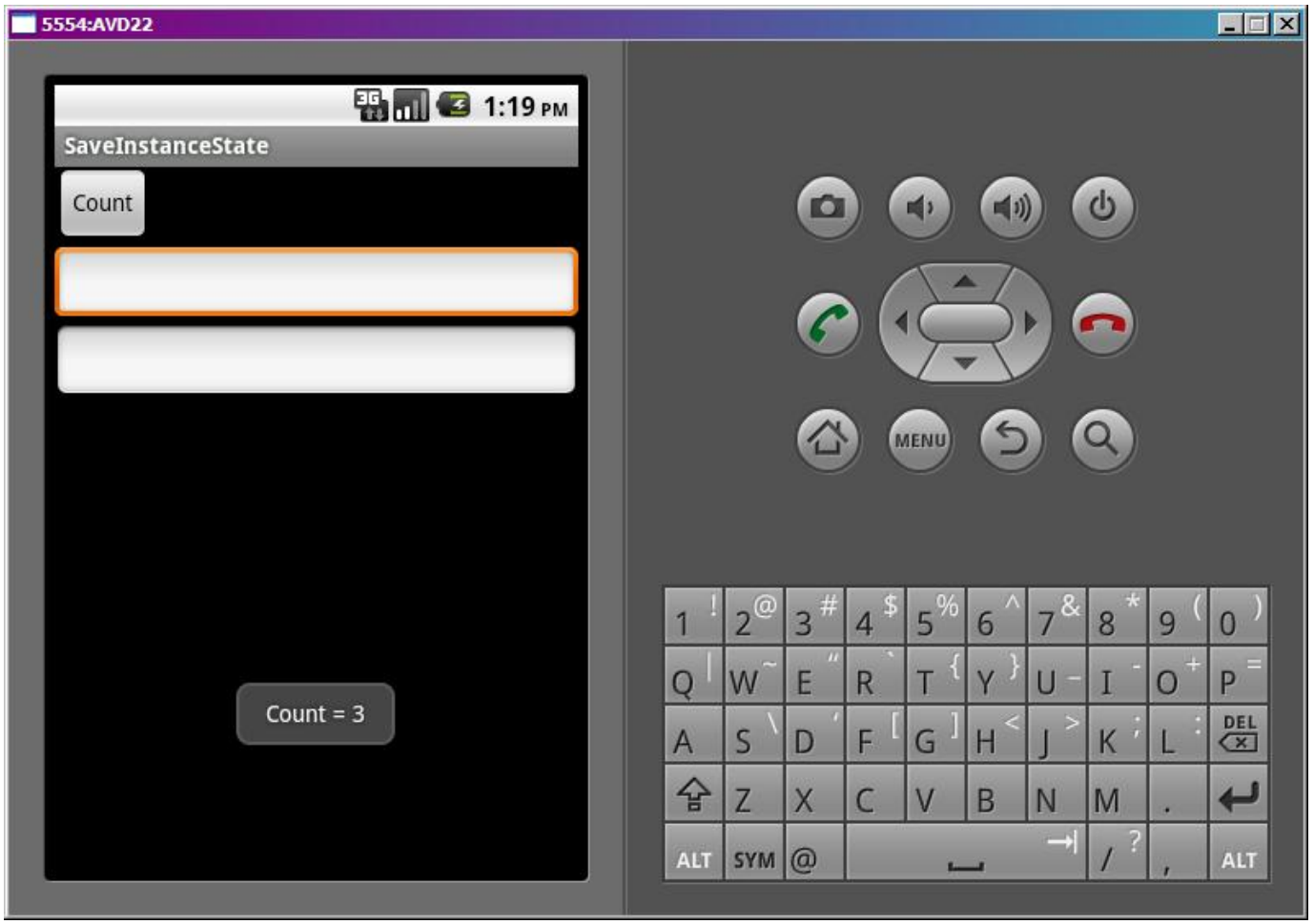
Метод восстановления **onRestoreInstanceState**:

```
protected void onRestoreInstanceState(Bundle savedInstanceState) {
  super.onRestoreInstanceState(savedInstanceState);
  cnt = savedInstanceState.getInt("count");
  Log.d(LOG_TAG, "onRestoreInstanceState");
```

```
    }
```

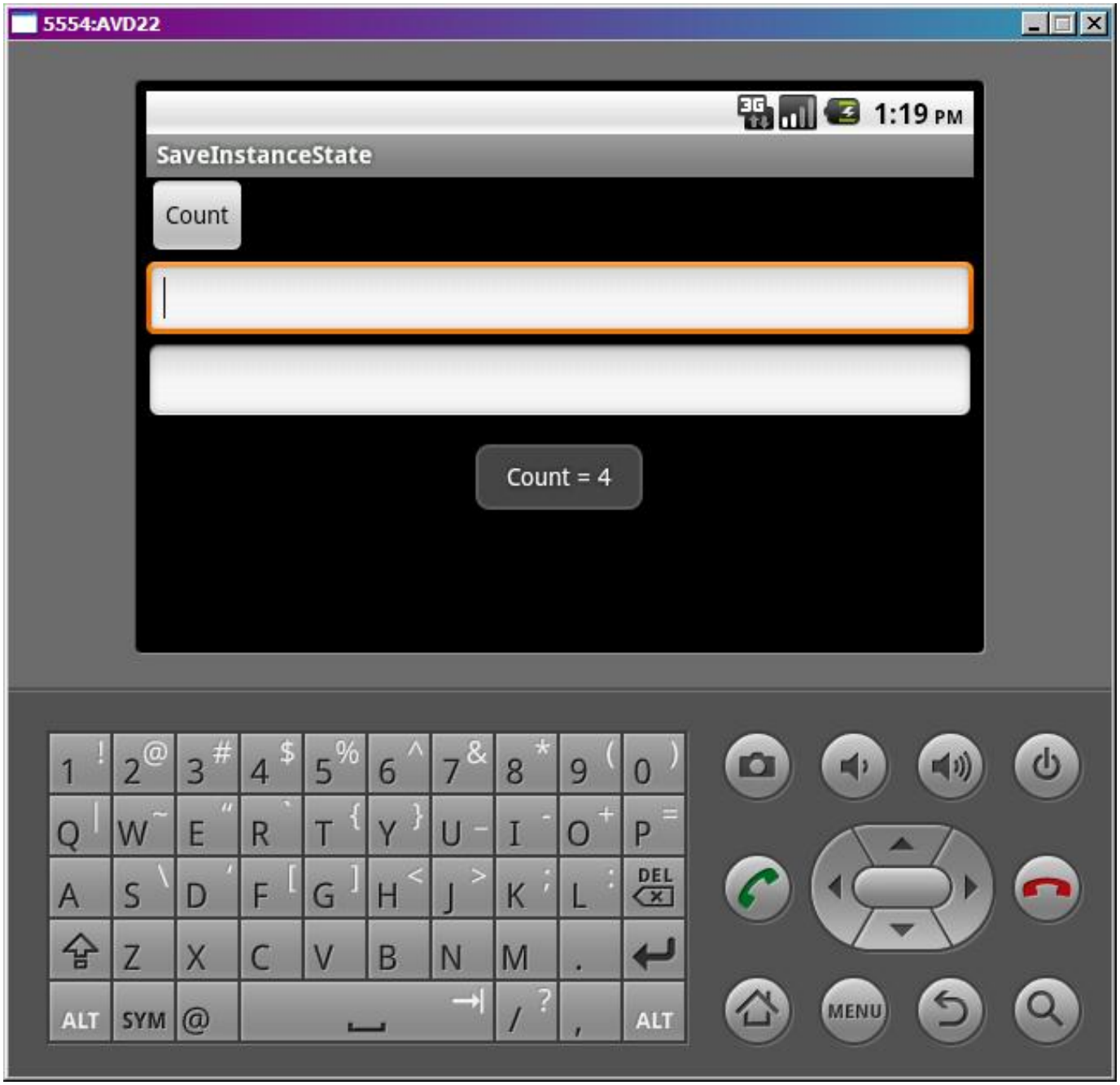
Из **savedInstanceState** вытаскиваем значение и помещаем в переменную **cnt**. Теперь при уничтожении и воссоздании Activity переменная cnt сохранит свое значение, и наш счетчик продолжит работать.

Проверим. Вернем AVD в вертикальную ориентацию. Все сохраним, запустим приложение. Понажимаем на кнопку, немного накрутим счетчик



и поворачиваем экран.

Жмем снова кнопку



счетчик не сбросился, а продолжил увеличиваться с последней позиции.

Итак, методы **onSaveInstanceState** и **onRestoreInstanceState** по дефолту сохраняют данные в экранных компонентах. Если мы реализуем их самостоятельно, то вызываем методы супер-класса и пишем свой код для своих переменных. Ради интереса, можете попробовать убрать вызовы методов суперкласса из onSaveInstanceState и onRestoreInstanceState. Данные в текстовом поле перестанут сохраняться при повороте экрана.

Кроме метода onRestoreInstanceState, доступ к сохраненным данным также можно получить в методе onCreate. На вход ему подается тот же самый Bundle. Если восстанавливать ничего не нужно, он будет = null.

Есть еще один полезный механизм сохранения данных. Android дает нам возможность сохранить ссылку на какой-либо объект и вернуть ее в новый созданный Activity. Для этого существуют методы:

onRetainNonConfigurationInstance – в нем мы сохраняем ссылку, передавая ее на выход (return) метода

getLastNonConfigurationInstance – этот метод ссылку нам возвращает

Т.е., например, у нас есть какой то объект myObj (класс MyObject) и нам надо сохранить ссылку на него при повороте экрана.

Мы реализуем в Activity метод **onRetainNonConfigurationInstance**:

```
public Object onRetainNonConfigurationInstance() {
  return myObj;
}
```

Этот метод будет вызван перед уничтожением Activity. От нас требуется дать на выход этому методу наш объект, который надо сохранить.

А, при создании нового Activity, в onCreate (например) мы используем метод **getLastNonConfigurationInstance**:

```
myObj = (MyObject) getLastNonConfigurationInstance();
```

Мы получили обратно объект класса Object и привели его к нашему классу MyObject.

На следующем уроке:

- используем Preferences для работы с настройками приложения

Урок 71. Preferences как настройки приложения. PreferenceActivity

В этом уроке:

- используем Preferences для работы с настройками приложения

Мы уже проходили Preferences, использовали их для хранения своих данных, знаем механизм. Теперь посмотрим, как они используются для хранения настроек приложения. Android предоставляет специальное Activity для работы с настройками – PreferenceActivity. Оно умеет читать определенные xml-файлы и создавать из них экран с настройками.

Создадим простое приложение. На первом экране будем читать и отображать настройки приложения, а на втором будем их задавать с помощью PreferenceActivity.

Надо придумать какие-нибудь настройки. Пусть это будут настройки неких уведомлений, а именно возможность включить/выключить уведомления (чекбокс) и возможность прописать адрес получателя (поле для ввода текста).

Создадим проект:

Project name: P0711_PreferencesSimple
Build Target: Android 2.3.3
Application name: PreferencesSimple
Package name: ru.startandroid.develop.p0711preferencessimple
Create Activity: MainActivity

Сначала создадим xml-файл с описанием настроек. В папке **res** создаем папку **xml**, а в ней создаем файл **pref.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
          android:key="notif"
          android:summary="Enable notifications"
          android:title="Notifications">
    </CheckBoxPreference>
    <EditTextPreference
          android:key="address"
          android:summary="Address for notifications"
          android:title="Address">
    </EditTextPreference>
</PreferenceScreen>
```

Здесь мы указали, что наш экран настроек (**PreferenceScreen**) будет содержать чекбокс (**CheckBoxPreference**) и поле (**EditTextPreference**) для ввода значения. Параметры: **key** – ключ для сохранения/чтения, **summary** – текст-пояснение, **title** – заголовок. Далее увидим, что и где используется.

Создаем Activity для настроек. **PrefActivity.java**:

```
package ru.startandroid.develop.p0711preferencessimple;
```

```
import android.os.Bundle;
import android.preference.PreferenceActivity;

public class PrefActivity extends PreferenceActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.pref);
  }
}
```

Вместо setContentView используется метод addPreferencesFromResource, который берет файл pref.xml и по нему создает экран настроек.

Не забудьте добавить это Activity в манифест. Тут никаких особенностей, все как обычно.

Рисуем экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <TextView
         android:id="@+id/tvInfo"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="">
    </TextView>
</LinearLayout>
```

Одно TextView, которое будет читать и отображать настройки приложения.

MainActivity.java:

```
package ru.startandroid.develop.p0711preferencessimple;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class MainActivity extends Activity {

  TextView tvInfo;
  SharedPreferences sp;
```

```
  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvInfo = (TextView) findViewById(R.id.tvInfo);

    // получаем SharedPreferences, которое работает с файлом настроек
    sp = PreferenceManager.getDefaultSharedPreferences(this);
    // полная очистка настроек
    // sp.edit().clear().commit();
  }

  protected void onResume() {
    Boolean notif = sp.getBoolean("notif", false);
    String address = sp.getString("address", "");
    String text = "Notifications are "
        + ((notif) ? "enabled, address = " + address : "disabled");
    tvInfo.setText(text);
    super.onResume();
  }

  public boolean onCreateOptionsMenu(Menu menu) {
    MenuItem mi = menu.add(0, 1, 0, "Preferences");
    mi.setIntent(new Intent(this, PrefActivity.class));
    return super.onCreateOptionsMenu(menu);
  }
}
```

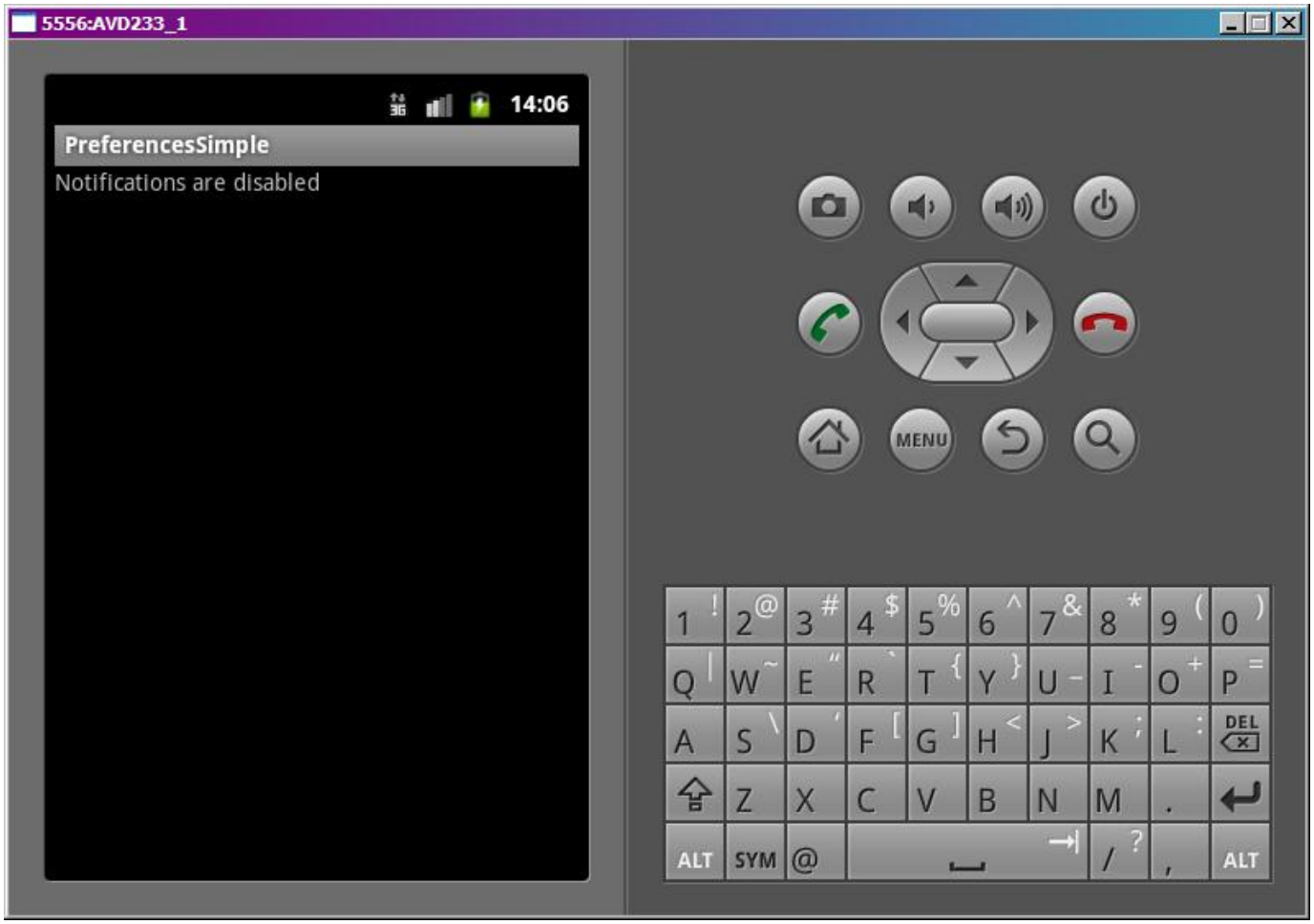
В **onCreate** мы находим TextView и получаем объект для работы с настройками - SharedPreferences. Он нам знаком, мы с ним работали ранее в уроке про Preferences. Далее идет закоментированный код полной очистки настроек приложения. Мы его не используем, я на всякий случай указал, может кому понадобится.

В **onResume** мы читаем из SharedPreferences настройки и выводим их в TextView. При чтении используем те самые ключи, которые прописывали в xml-файле в атрибутах **key**.

В **onCreateOptionsMenu** просто настраиваем меню для вызова окна настроек. Мы создаем пункт меню и вешаем на него Intent - в итоге при нажатии вызовется Activity.

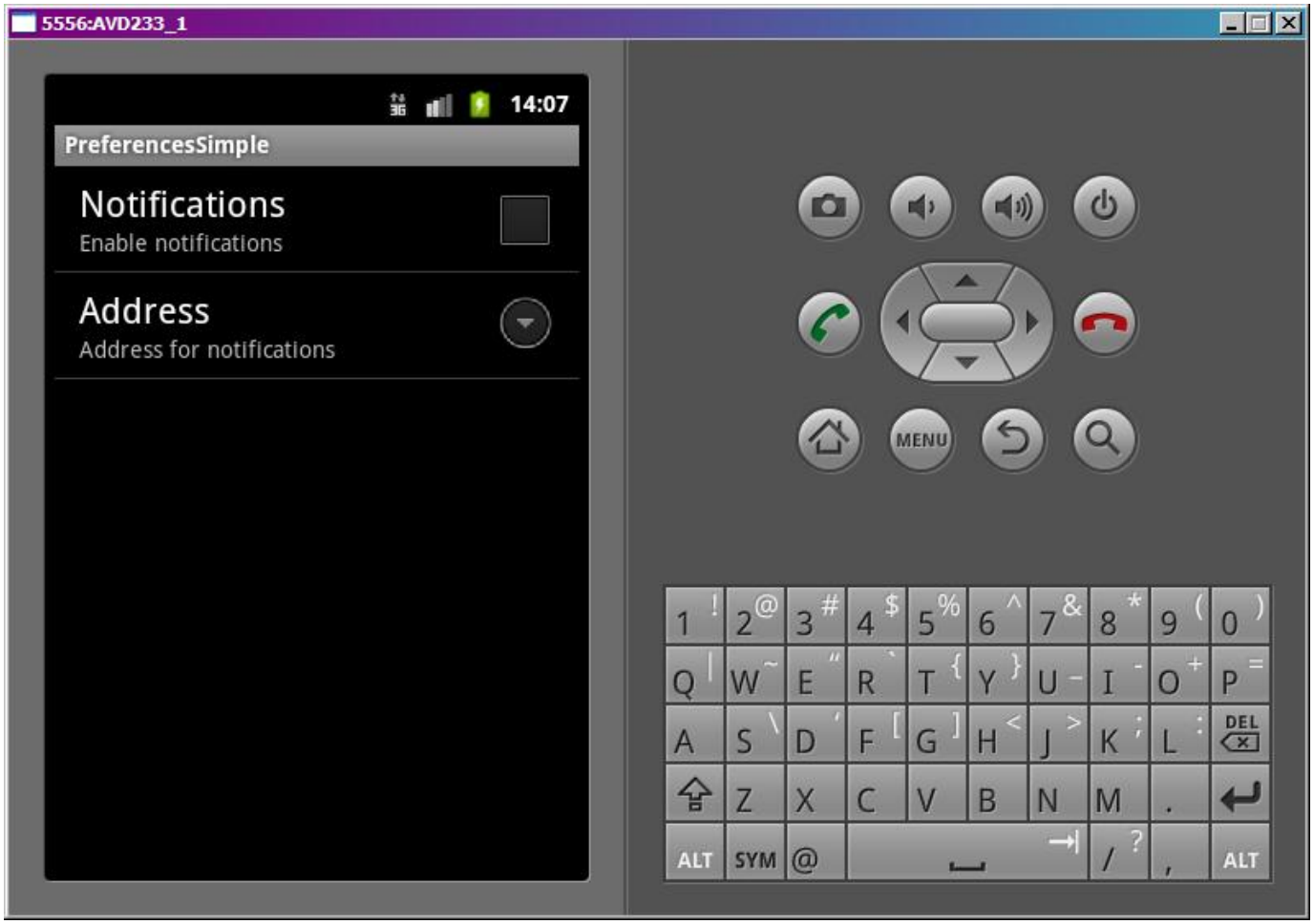
На всякий случай уточню, почему для вывода инфы на экран здесь использую **onResume**, а не **onCreate**. Потому что, когда мы будем возвращаться с экрана настроек в главное окно, то onCreate главного окна не сработает (т.к. окно уже создано и висит в памяти) и изменения мы не увидим. А onResume точно сработает.

Все сохраняем и запускаем приложение. Видим такой экран:



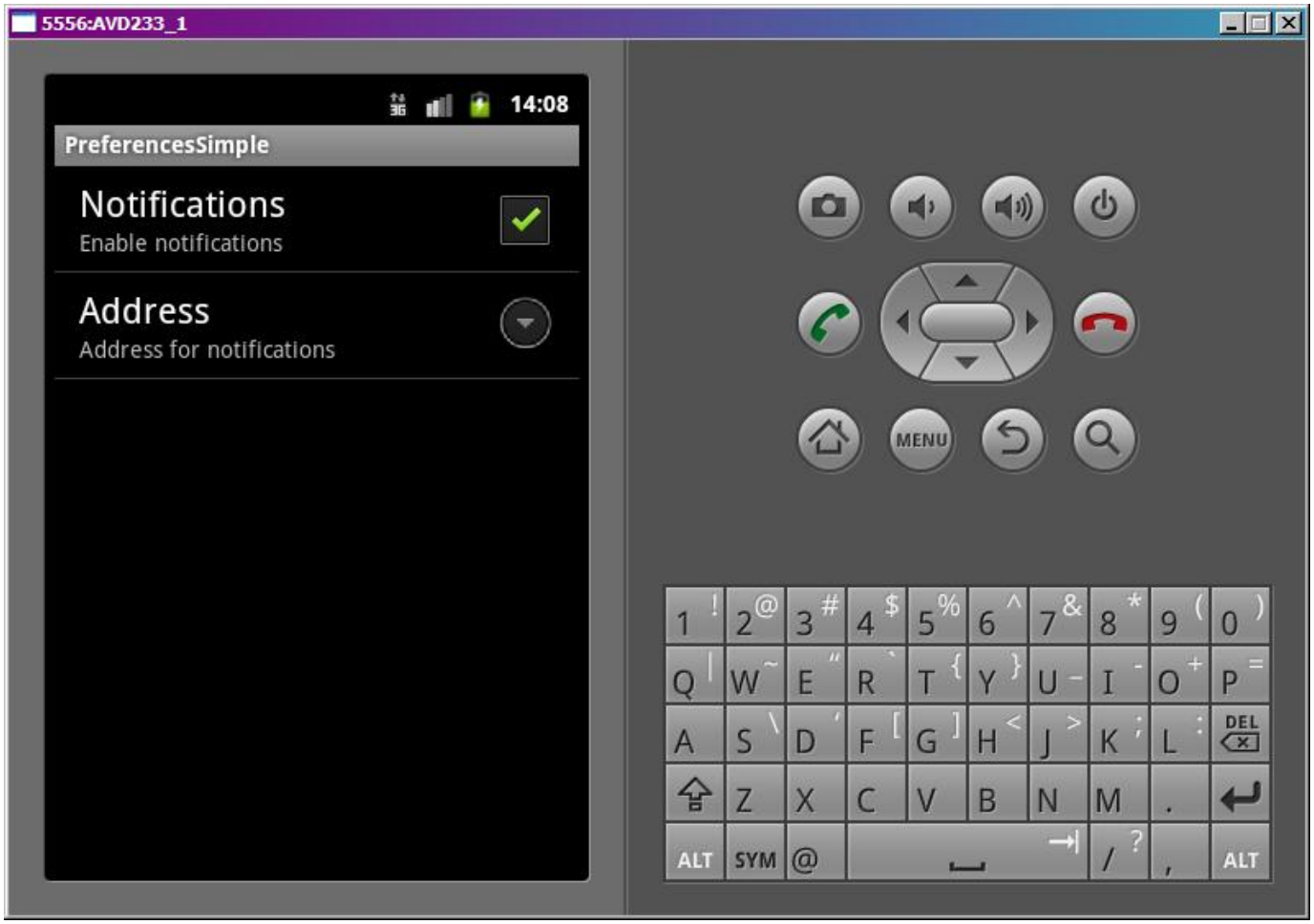
По умолчанию, если настройки еще не заданы, мы получаем false и пустую строку, как и указывали в методах getBoolean и getString в onResume.

Жмем menu и переходим к настройкам.

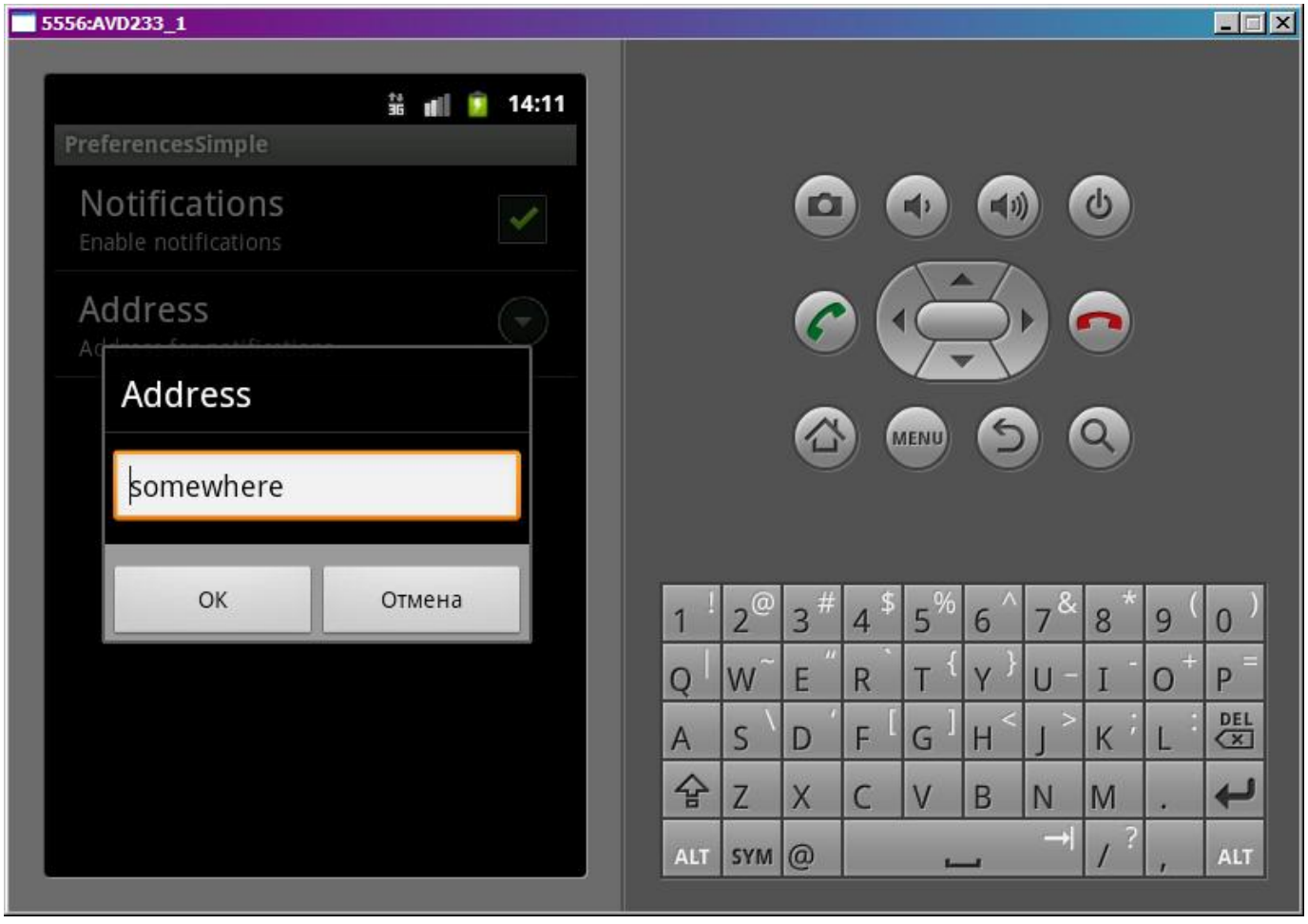


То, что мы указывали в **pref.xml** система прочитала и сама создала экран настроек. Теперь мы можем здесь указывать нужные нам значения и они сразу же будут сохраняться без всяких кнопок Save. Впрочем, это стандартное поведение Android-настроек. Думаю, каждый из вас когда-нить копался в настройках и знает это.

Давайте включим наши уведомления

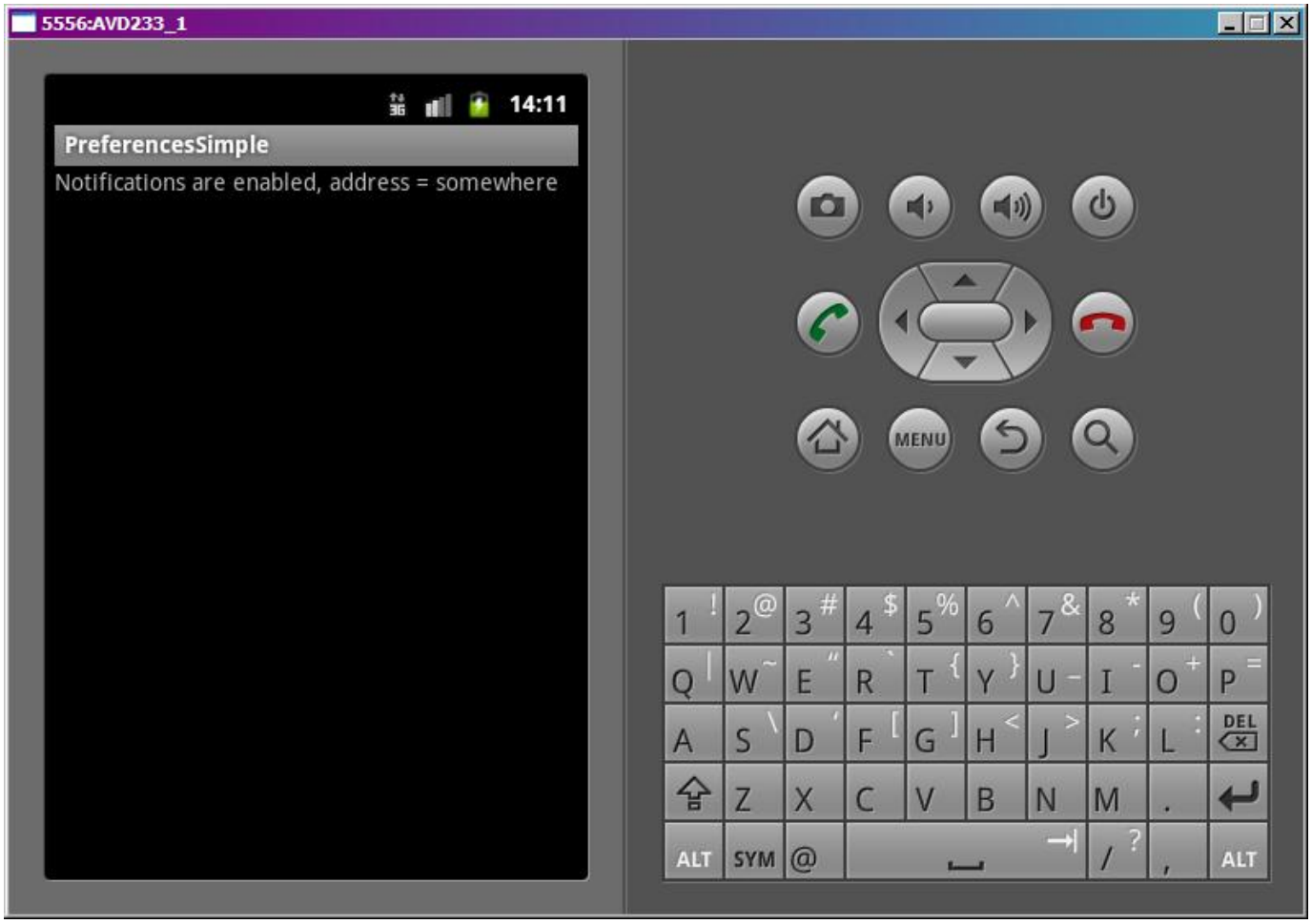


и пропишем адрес



Жмем OK.

Жмем кнопку назад, попадаем на главный экран



который в onResume прочел свежие настройки и отобразил их.

На следующем уроке:

- используем в настройках список
- группируем настройки по экранам и категориям

Урок 72. Preferences. Список, экраны и категории

В этом уроке:

- используем в настройках список
- группируем настройки по экранам и категориям

На прошлом уроке мы узнали, как несложно создается экран настроек, и использовали там чекбокс и поле для ввода. Немного расширим наши знания и используем новые компоненты:

список (ListPreference) – позволяет выбрать значение из нескольких возможных

экран (PreferenceScreen) – с его помощью можно делать вложенные экраны с настройками

категория (PreferenceCategory) – группировка нескольких настроек

Создадим простое приложение. На первом экране будем читать и отображать значение из настройки-списка. Второй экран будет отображать все настройки.

Создадим проект:

Project name: P0721_PreferencesSimple2
Build Target: Android 2.3.3
Application name: PreferencesSimple2
Package name: ru.startandroid.develop.p0721preferencessimple2
Create Activity: MainActivity

Заполним **strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">PreferencesSimple2</string>
    <string-array  name="entries">
        <item>one</item>
        <item>two</item>
        <item>three</item>
    </string-array>
    <string-array  name="entry_values">
        <item>1</item>
        <item>2</item>
        <item>3</item>
    </string-array>
</resources>
```

Мы создали два массива, чуть позже их используем для списка значений.

Рисуем основной экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:id="@+id/tvInfo"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="">
     </TextView>
</LinearLayout>
```

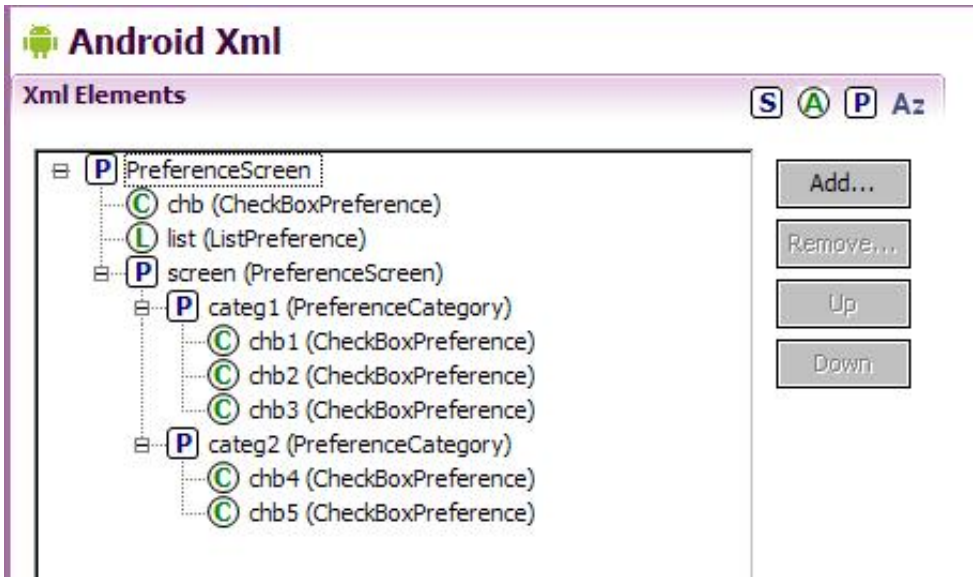
Здесь только текстовое поле.

Создадим xml-файл с настройками **res/xml/pref.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android">
     <CheckBoxPreference
          android:key="chb"
          android:summary="Description of checkbox"
          android:title="Checkbox">
     </CheckBoxPreference>
     <ListPreference
          android:entries="@array/entries"
          android:entryValues="@array/entry_values"
          android:key="list"
          android:summary="Description of list"
          android:title="List">
     </ListPreference>
     <PreferenceScreen
          android:key="screen"
          android:summary="Description of screen"
          android:title="Screen">
         <PreferenceCategory
              android:key="categ1"
              android:summary="Description of category 1"
              android:title="Category 1">
             <CheckBoxPreference
                  android:key="chb1"
                  android:summary="Description of checkbox 1"
                  android:title="CheckBox 1">
             </CheckBoxPreference>
             <CheckBoxPreference
                  android:key="chb2"
                  android:summary="Description of checkbox 2"
                  android:title="CheckBox 2">
             </CheckBoxPreference>
             <CheckBoxPreference
                  android:key="chb3"
```

```
                android:summary="Description of checkbox 3"
                android:title="CheckBox 3">
            </CheckBoxPreference>
        </PreferenceCategory>
        <PreferenceCategory
             android:key="categ2"
             android:summary="Description of category 2"
             android:title="Category 2">
            <CheckBoxPreference
                android:key="chb4"
                android:summary="Description of checkbox 4"
                android:title="CheckBox 4">
            </CheckBoxPreference>
            <CheckBoxPreference
                android:key="chb5"
                android:summary="Description of checkbox 5"
                android:title="CheckBox 5">
            </CheckBoxPreference>
        </PreferenceCategory>
    </PreferenceScreen>
</PreferenceScreen>
```

Создавать его, конечно, можно не только через xml, но и конструктором. Вот такая получается картинка:



У нас в настройках будет чекбокс (chb), список (list) и вложенный экран (screen). Внутри screen две категории (categ 1 и categ2), а в них просто чекбоксы. Все просто и понятно.

Из интересного стоит отметить аттрибуты entries и entryValues у ListPreference. entries – это то, что отобразит список пользователю, а entryValues – это то, что сохранит система после выбора значения в списке. Мы используем тут два разных массива. В итоге список покажет значения ***one***, ***two***, ***three***, а сохранит соответственно ***1***,***2*** или ***3***. Можно использовать один массив в обоих атрибутах, тогда отображаемые и сохраняемые значения будут равны.

Создаем Activity для настроек – **PrefActivity.java**:

```
package ru.startandroid.develop.p0721preferencessimple2;
```

```
import android.os.Bundle;
import android.preference.PreferenceActivity;

public class PrefActivity extends PreferenceActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.pref);
  }
}
```

И **MainActivity.java**:

```
package ru.startandroid.develop.p0721preferencessimple2;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class MainActivity extends Activity {

  TextView tvInfo;
  SharedPreferences sp;

    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        tvInfo = (TextView) findViewById(R.id.tvInfo);
        sp = PreferenceManager.getDefaultSharedPreferences(this);
    }

    protected void onResume() {
      String listValue = sp.getString("list", "не выбрано");
      tvInfo.setText("Значение списка - " + listValue);
      super.onResume();
    }

    public boolean onCreateOptionsMenu(Menu menu) {
      MenuItem mi = menu.add(0, 1, 0, "Preferences");
      mi.setIntent(new Intent(this, PrefActivity.class));
      return super.onCreateOptionsMenu(menu);
    }
}
```

В **onCreate** получаем доступ к настройкам.

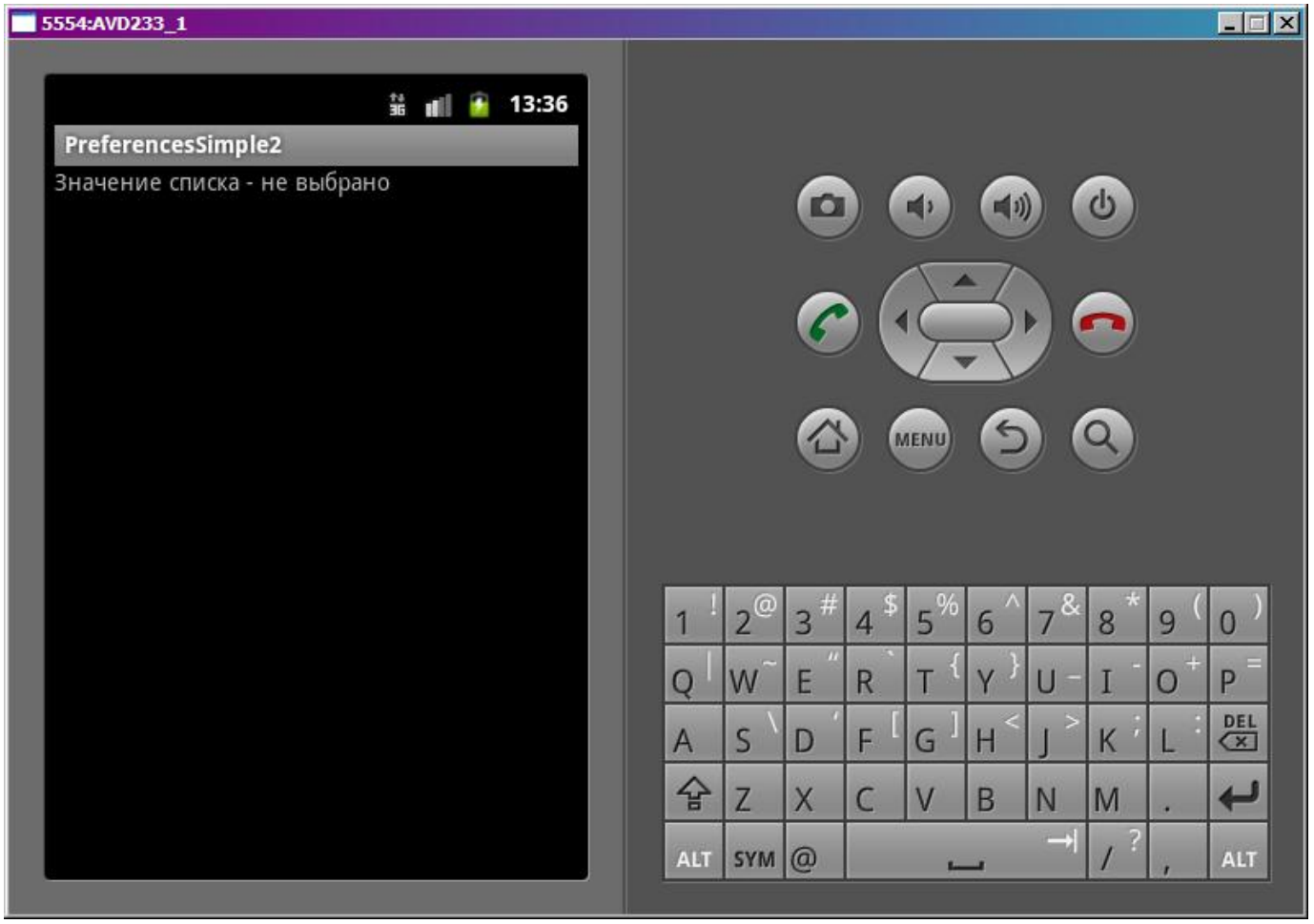
В **onResume** выводим в tvInfo значение из настроек, которое записал туда список. Если значений нет, то пишем текст

«не выбрано»

В **onCreateOptionsMenu** создаем пункт меню и вешаем на него Intent, который запустит нам экран настроек.

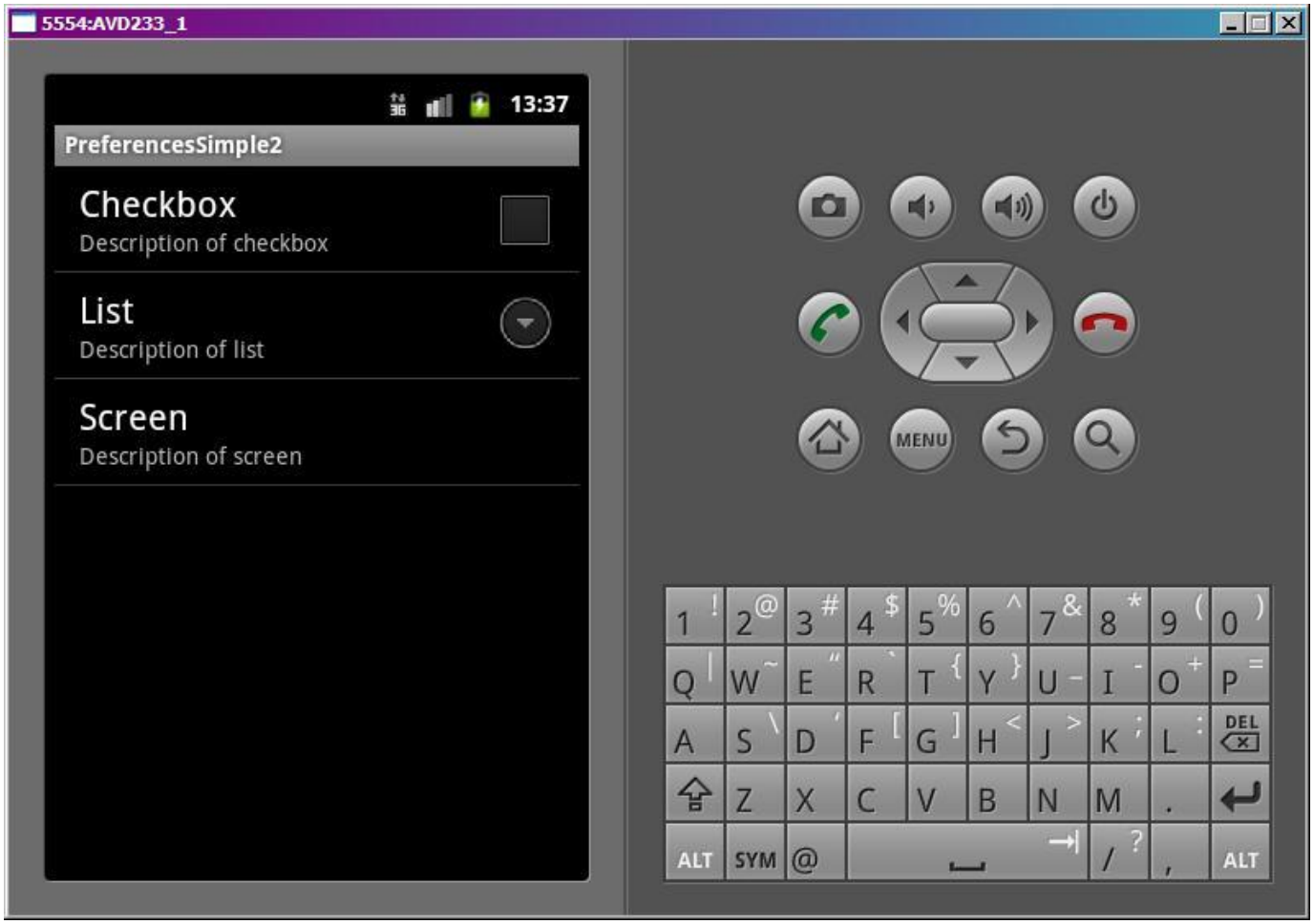
Остальные параметры читать не буду, это мы на прошлом уроке рассмотрели, там все просто.

Все сохраним и запустим.



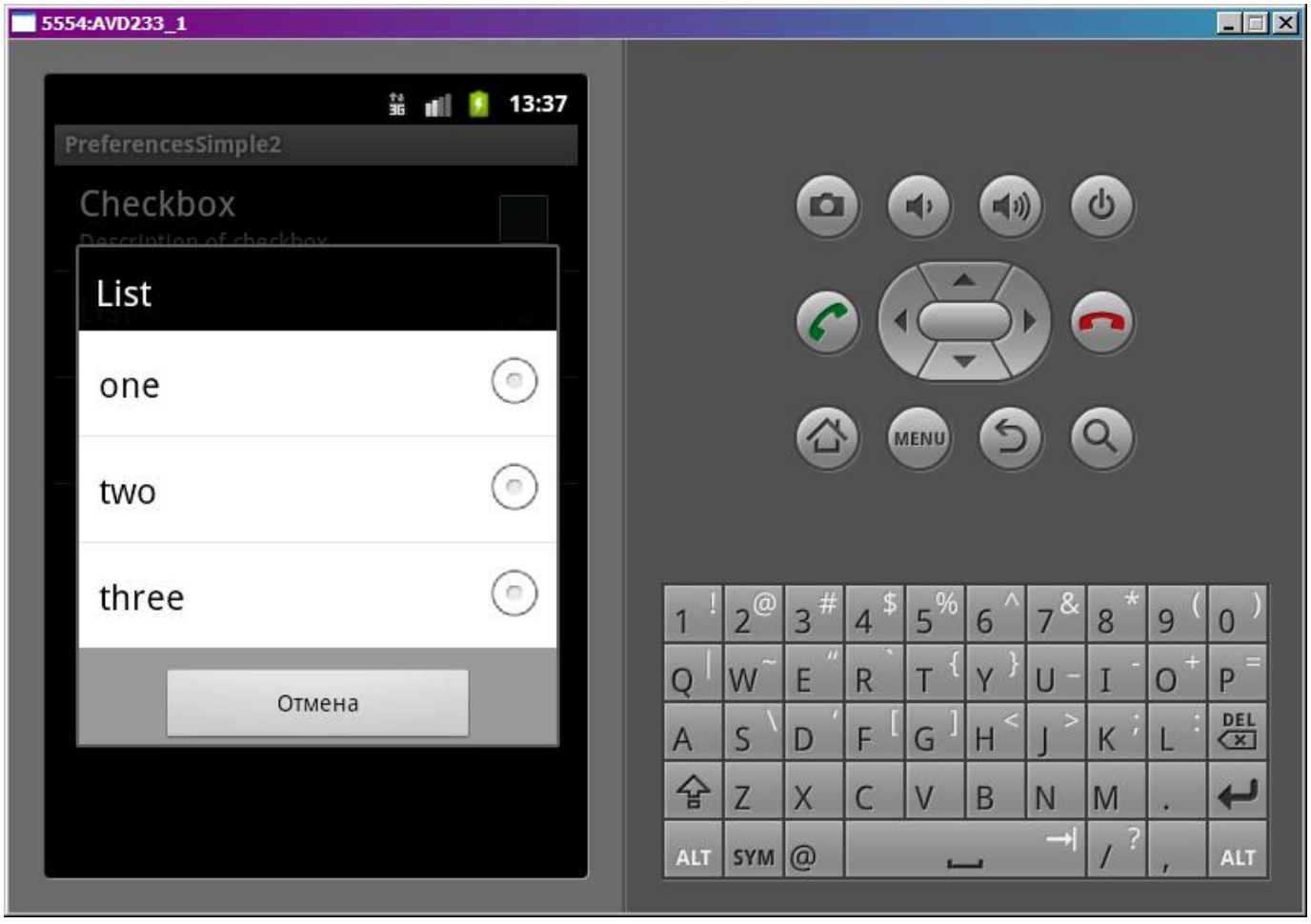
Изначально настройки пусты, в списке ничего не было выбрано.

Переходим через меню на экран настроек.

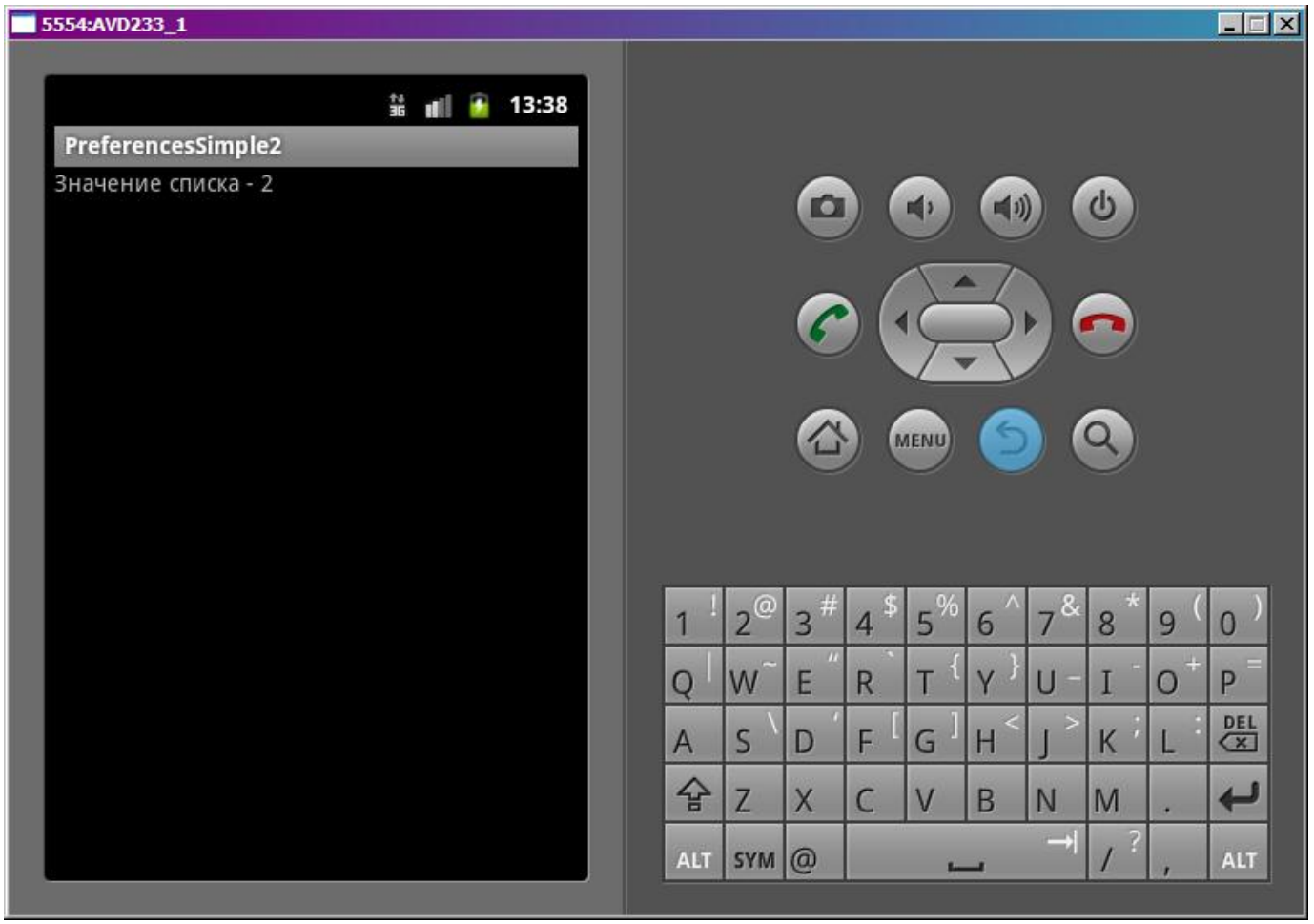


Первый пункт – чекбокс, второй – список, третий – вложенный экран.

Нажмем на список:

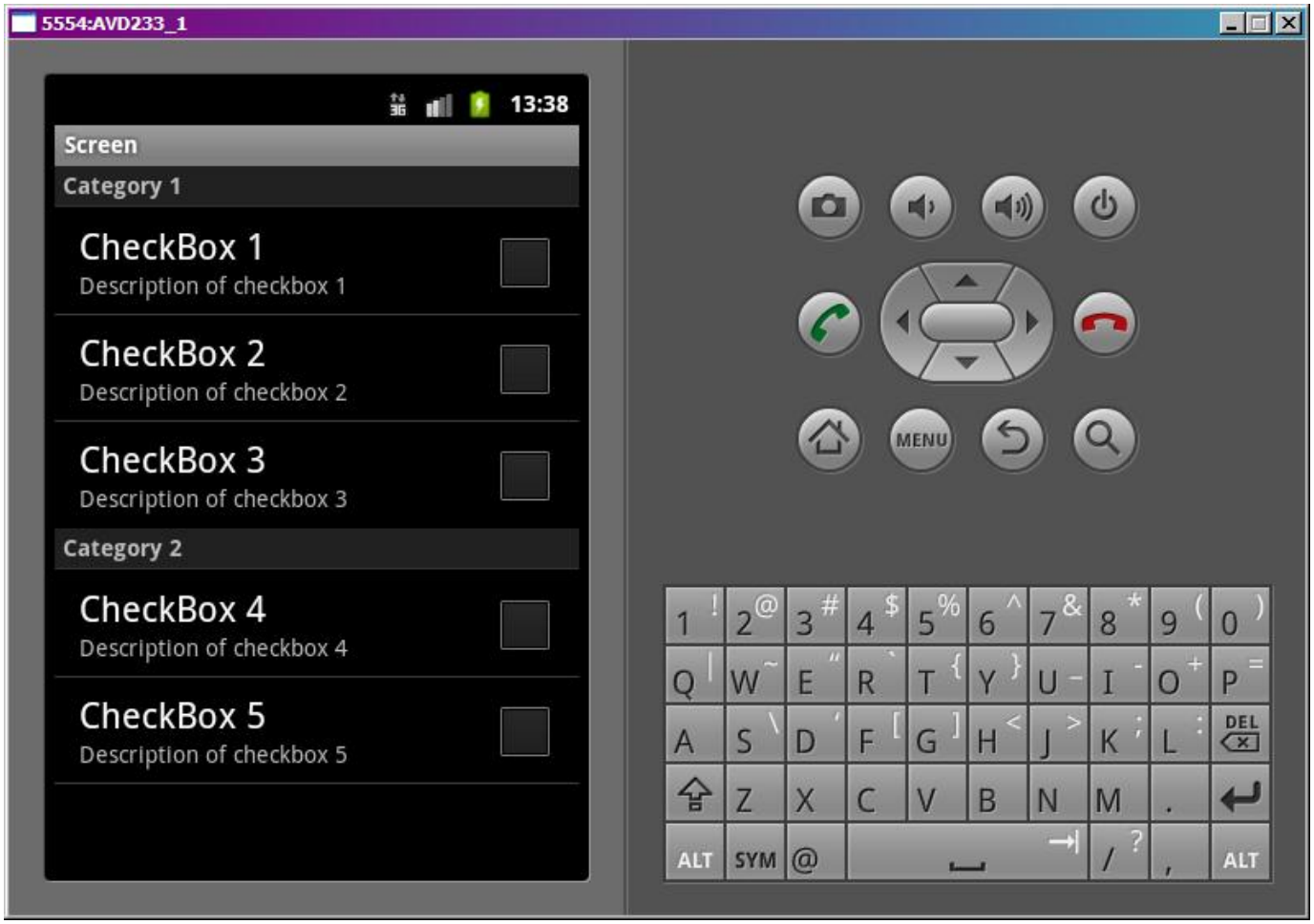


Показаны значения из списка entries. Выбираем устраивающее нас значение, список закрывается и сохраняется соответствующее значение из списка entry_values. Можно вернуться на первый экран кнопкой Назад (Back) и убедиться в этом.



Я выбрал two, сохранилось 2.

Снова идем в экран настроек и жмем на вложенный экран Screen. Провалились внутрь.



Тут куча чекбоксов, которые сгруппированы по двум категориям.

Теперь мы умеем создавать экраны настроек с группировкой элементов по категориям и подэкранам. Это значительно удобнее для пользователя, чем куча разных настроек на одном экране.

На следующем уроке:

- управляем активностью настроек (setEnabled)

Урок 73. Preferences. Управляем активностью настроек (setEnabled)

В этом уроке:

- управляем активностью настроек (setEnabled)

Иногда в настройках можно наблюдать, как некоторые из них неактивны - с ними нельзя взаимодействовать. Причем это зависит от смежных настроек. Т.е. выключили одну галку - стала неактивна другая. Разберемся, как это делается.

Создадим проект:

Project name: P0731_PreferencesEnable
Build Target: Android 2.3.3
Application name: PreferencesEnable
Package name: ru.startandroid.develop.p0731preferencesenable
Create Activity: MainActivity

Заполним **strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="hello">Hello World, MainActivity!</string>
    <string  name="app_name">PreferencesEnable</string>
    <string-array  name="entries">
        <item>one</item>
        <item>two</item>
        <item>three</item>
    </string-array>
    <string-array  name="entry_values">
        <item>1</item>
        <item>2</item>
        <item>3</item>
    </string-array>
</resources>
```

Это два массива для списка ListPreference.

Создадим xml файл настроек, **res/xml/pref.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
     xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
         android:key="chb1"
         android:summaryOff="Description of checkbox 1 off"
         android:summaryOn="Description of checkbox 1 on"
         android:title="CheckBox 1">
    </CheckBoxPreference>
```

```
    <ListPreference
        android:dependency="chb1"
        android:entries="@array/entries"
        android:entryValues="@array/entry_values"
        android:key="list"
        android:summary="Description of list"
        android:title="List">
    </ListPreference>
    <CheckBoxPreference
        android:key="chb2"
        android:summary="Description of checkbox 2"
        android:title="CheckBox 2">
    </CheckBoxPreference>
    <PreferenceScreen
        android:dependency="chb2"
        android:key="screen"
        android:summary="Description of screen"
        android:title="Screen">
        <CheckBoxPreference
            android:key="chb3"
            android:summary="Description of checkbox 3"
            android:title="CheckBox 3">
        </CheckBoxPreference>
        <PreferenceCategory
            android:key="categ1"
            android:summary="Description of category 1"
            android:title="Category 1">
            <CheckBoxPreference
                android:key="chb4"
                android:summary="Description of checkbox 4"
                android:title="CheckBox 4">
            </CheckBoxPreference>
        </PreferenceCategory>
        <PreferenceCategory
            android:key="categ2"
            android:summary="Description of category 2"
            android:title="Category 2">
            <CheckBoxPreference
                android:key="chb5"
                android:summary="Description of checkbox 5"
                android:title="CheckBox 5">
            </CheckBoxPreference>
            <CheckBoxPreference
                android:key="chb6"
                android:summary="Description of checkbox 6"
                android:title="CheckBox 6">
            </CheckBoxPreference>
        </PreferenceCategory>
    </PreferenceScreen>
</PreferenceScreen>
```

Нас интересует атрибут dependency. В нем можно указать key какого-либо CheckBoxPreference - и по включению/выключения чекбокса будет активна/неактивна настройка. Т.е. например мы для ListPreference с key = ***list*** указали dependency = ***chb1***. И теперь включая/выключая ***chb1*** будет меняться активность ***list***. Этот механизм работает с

обычными настройками и с PreferenceScreen, а с PreferenceCategory – нет. Для категорий придется то же самое накодить.

Еще обратите внимание на атрибуты summaryOn и summaryOff у ***chb1***. Это тексты-описания настройки, аналогично summary. Они отображаются в зависимости от того включен (summaryOn) чекбокс или выключен (summaryOff).

Создаем экран настроек, **PrefActivity.java**:

```
package ru.startandroid.develop.p0731preferencesenable;

import android.os.Bundle;
import android.preference.CheckBoxPreference;
import android.preference.Preference;
import android.preference.Preference.OnPreferenceClickListener;
import android.preference.PreferenceActivity;
import android.preference.PreferenceCategory;

public class PrefActivity extends PreferenceActivity {

  CheckBoxPreference chb3;
  PreferenceCategory categ2;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.pref);

    chb3 = (CheckBoxPreference) findPreference("chb3");
    categ2  = (PreferenceCategory) findPreference("categ2");
    categ2.setEnabled(chb3.isChecked());

    chb3.setOnPreferenceClickListener(new OnPreferenceClickListener() {
      public boolean onPreferenceClick(Preference preference) {
        categ2.setEnabled(chb3.isChecked());
        return false;
      }
    });
  }
}
```

С помощью метода findPreference мы находим на экране чекбокс (***chb3***) и категорию (***categ2***). И методом setEnabled устанавливаем, что активность категории равна значению чекбокса (вкл/выкл). Это, чтобы при старте экрана все было верно.

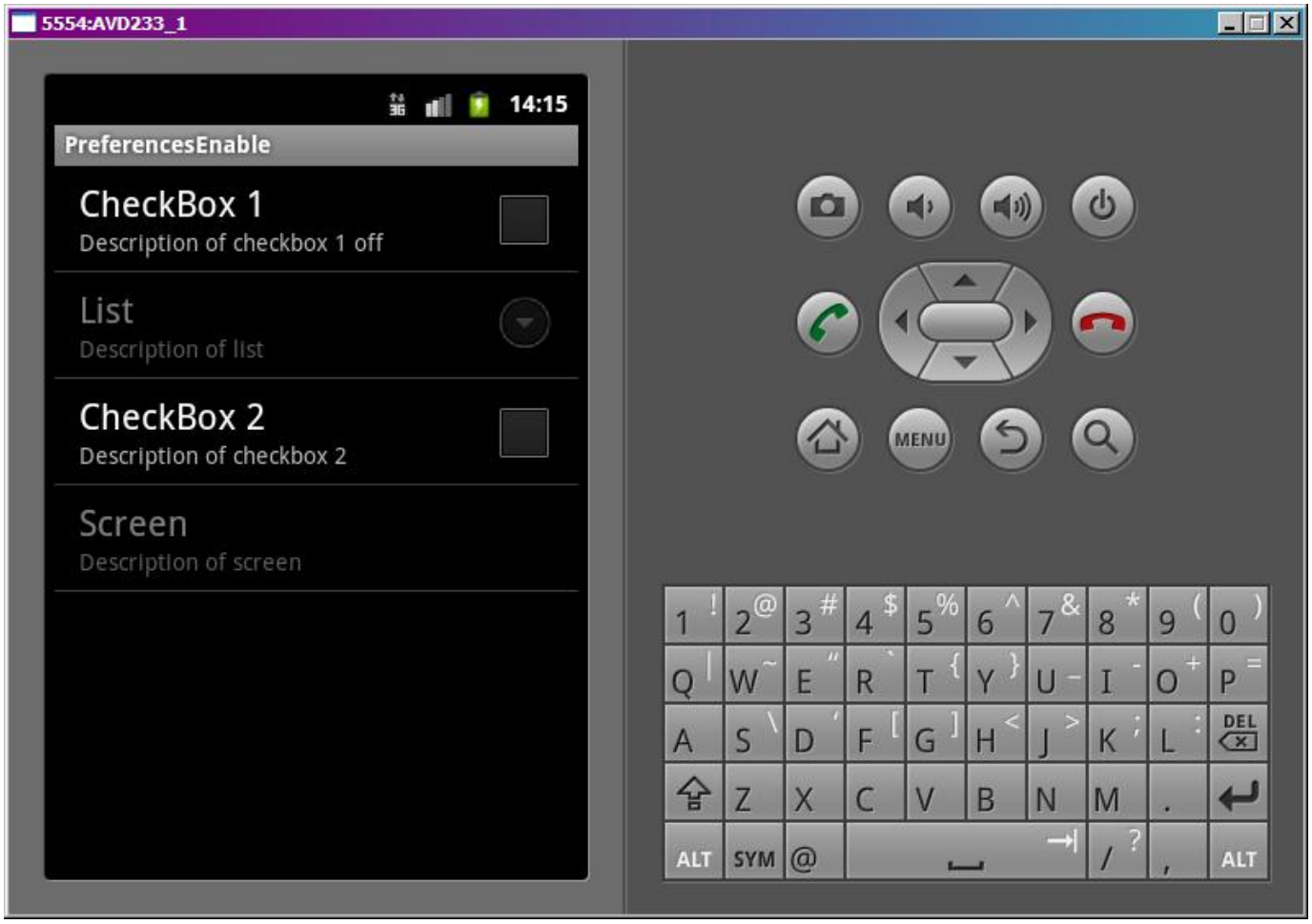
Далее для чекбокса прописываем обработчик и в нем по нажатию устанавливаем связь - активность категории равна значению чекбокса.

В **MainActivity.java** только создаем пункт меню для перехода к настройкам:

```
package ru.startandroid.develop.p0731preferencesenable;

import android.app.Activity;
import android.content.Intent;
```

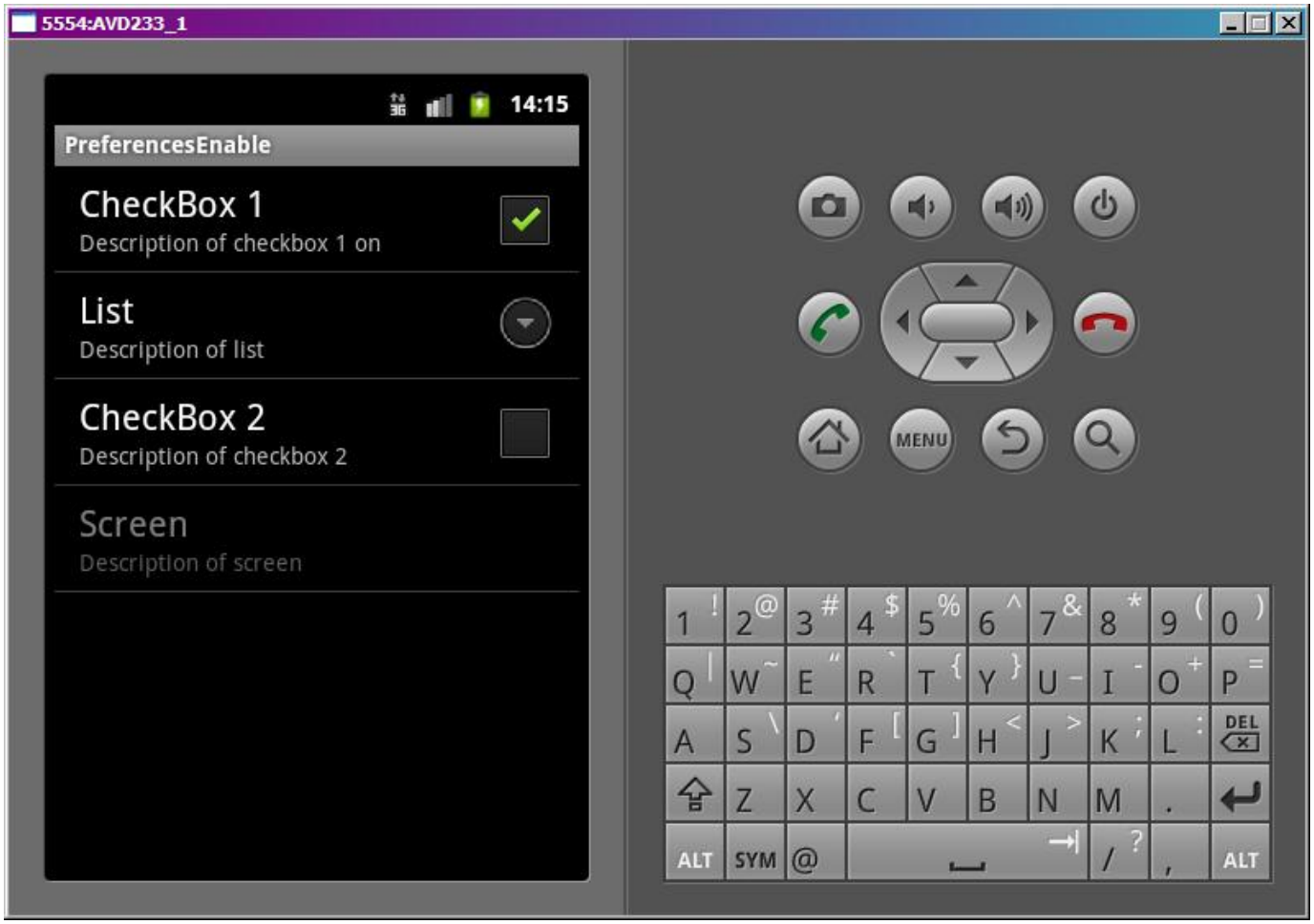
```
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends Activity {

    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public boolean onCreateOptionsMenu(Menu menu) {
      MenuItem mi = menu.add(0, 1, 0, "Preferences");
      mi.setIntent(new Intent(this, PrefActivity.class));
      return super.onCreateOptionsMenu(menu);
    }
}
```

Все сохраняем и запускаем приложение. Переходим к настройкам.



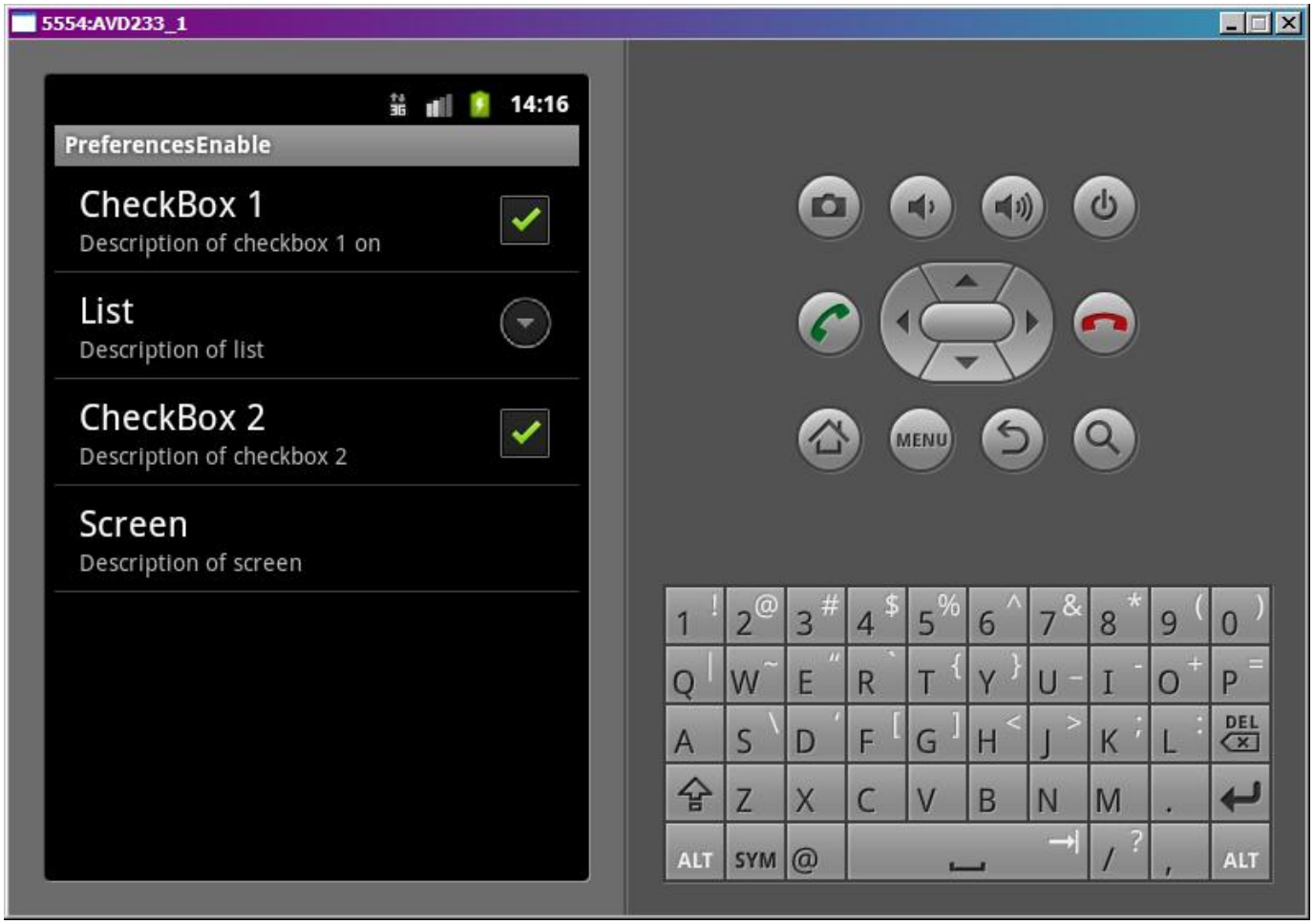
CheckBox 1 выключен, соответственно неактивен и ***List***. Также ***CheckBox 2*** выключен, неактивен ***Screen***. Это мы прописывали в атрибутах **dependency** в **pref.xml**.

Включим ***CheckBox 1***:



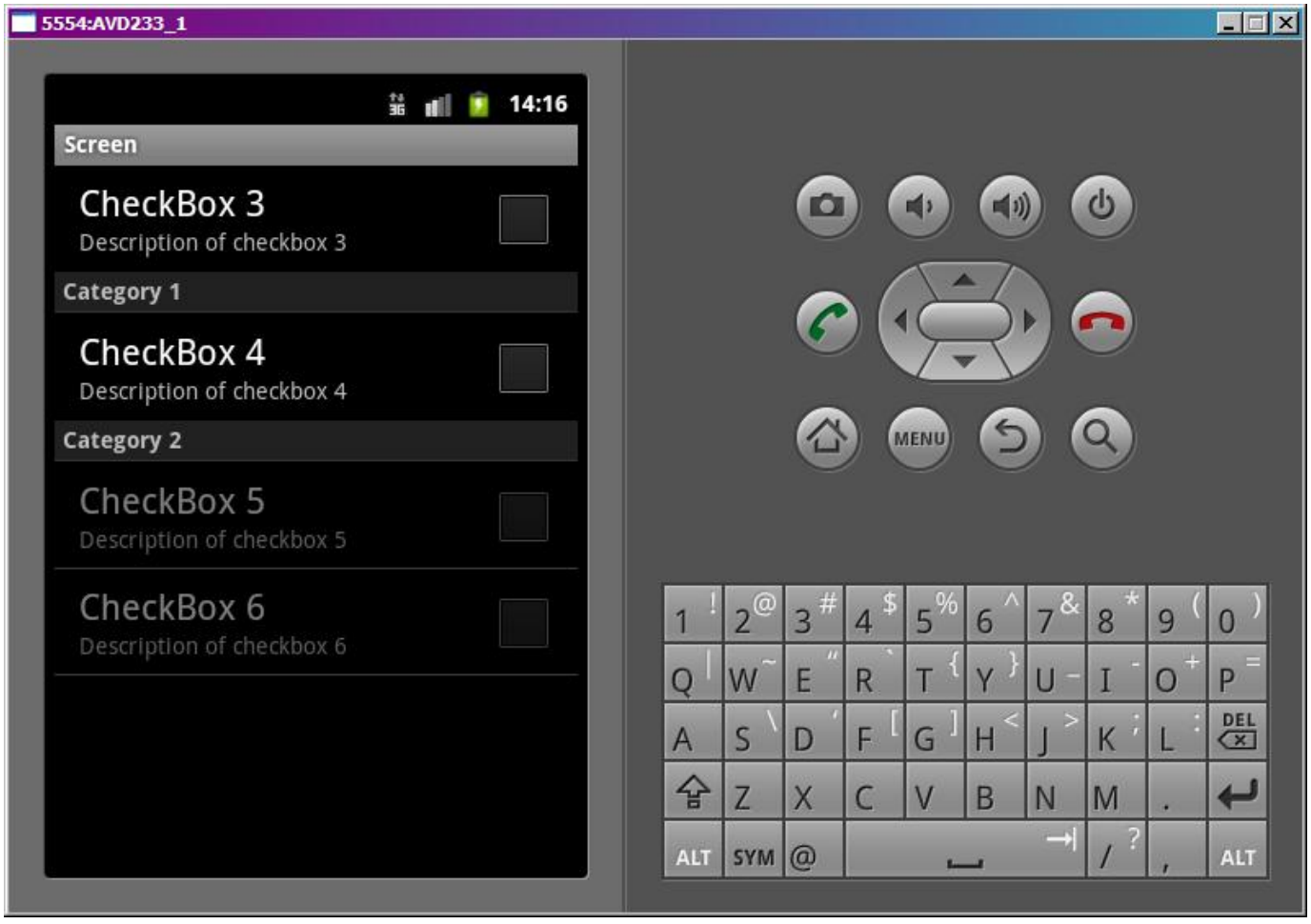
List теперь активен. И обратите внимание, что поменялось описание ***CheckBox 1***. Пока чекбокс был выключен, отображался текст из атрибута **summaryOff**, а теперь он включен и мы видим текст из **summaryOn**.

Давайте включим ***CheckBox2.***



Screen теперь активен.

Нажимаем на него и проваливаемся внутрь.



Напомню в коде мы прописывали, что ***Category 2*** зависела от ***CheckBox 3***. И действительно – чекбокс выключен и категория не активна.

Включим чекбокс

Категория стала активна. Все ок.

На этом уроке немного посмотрели, как через код работать с экраном настроек. Думаю, имеет смысл рассмотреть это более подробно.

На следующем уроке:

- создаем экран настроек программно

Урок 74. Preferences. Программное создание экрана настроек

В этом уроке:

- создаем экран настроек программно

На прошлых уроках мы создавали экран настроек из xml файлов. Теперь попробуем сделать это в коде. Попробуем создать экран с прошлого урока.

Создадим проект:

Project name: P0741_PreferencesCode
Build Target: Android 2.3.3
Application name: PreferencesCode
Package name: ru.startandroid.develop.p0741preferencescode
Create Activity: MainActivity

Заполним **strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="hello">Hello World, MainActivity!</string>
    <string  name="app_name">PreferencesCode</string>
    <string-array  name="entries">
        <item>one</item>
        <item>two</item>
        <item>three</item>
    </string-array>
    <string-array  name="entry_values">
        <item>1</item>
        <item>2</item>
        <item>3</item>
    </string-array>
</resources>
```

Напомню xml-код настроек из прошлого урока:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
     xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
         android:key="chb1"
         android:summaryOff="Description of checkbox 1 off"
         android:summaryOn="Description of checkbox 1 on"
         android:title="CheckBox 1">
    </CheckBoxPreference>
    <ListPreference
         android:dependency="chb1"
         android:entries="@array/entries"
```

```
            android:entryValues="@array/entry_values"
            android:key="list"
            android:summary="Description of list"
            android:title="List">
        </ListPreference>
        <CheckBoxPreference
            android:key="chb2"
            android:summary="Description of checkbox 2"
            android:title="CheckBox 2">
        </CheckBoxPreference>
        <PreferenceScreen
            android:dependency="chb2"
            android:key="screen"
            android:summary="Description of screen"
            android:title="Screen">
            <CheckBoxPreference
                android:key="chb3"
                android:summary="Description of checkbox 3"
                android:title="CheckBox 3">
            </CheckBoxPreference>
            <PreferenceCategory
                android:key="categ1"
                android:summary="Description of category 1"
                android:title="Category 1">
                <CheckBoxPreference
                    android:key="chb4"
                    android:summary="Description of checkbox 4"
                    android:title="CheckBox 4">
                </CheckBoxPreference>
            </PreferenceCategory>
            <PreferenceCategory
                android:key="categ2"
                android:summary="Description of category 2"
                android:title="Category 2">
                <CheckBoxPreference
                    android:key="chb5"
                    android:summary="Description of checkbox 5"
                    android:title="CheckBox 5">
                </CheckBoxPreference>
                <CheckBoxPreference
                    android:key="chb6"
                    android:summary="Description of checkbox 6"
                    android:title="CheckBox 6">
                </CheckBoxPreference>
            </PreferenceCategory>
        </PreferenceScreen>
    </PreferenceScreen>
```

Попытаемся создать все эти элементы программно.

Создаем **PrefActivity.java**:

```
package ru.startandroid.develop.p0741preferencescode;

import android.os.Bundle;
import android.preference.CheckBoxPreference;
import android.preference.ListPreference;
import android.preference.Preference;
import android.preference.Preference.OnPreferenceClickListener;
import android.preference.PreferenceActivity;
import android.preference.PreferenceCategory;
import android.preference.PreferenceScreen;

public class PrefActivity extends PreferenceActivity {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // создаем экран
    PreferenceScreen rootScreen = getPreferenceManager().createPreferenceScreen(this);
    // говорим Activity, что rootScreen - корневой
    setPreferenceScreen(rootScreen);

    // даллее создаем элементы, присваиваем атрибуты и формируем иерархию

    CheckBoxPreference chb1 = new CheckBoxPreference(this);
    chb1.setKey("chb1");
    chb1.setTitle("CheckBox 1");
    chb1.setSummaryOn("Description of checkbox 1 on");
    chb1.setSummaryOff("Description of checkbox 1 off");

    rootScreen.addPreference(chb1);


    ListPreference list = new ListPreference(this);
    list.setKey("list");
    list.setTitle("List");
    list.setSummary("Description of list");
    list.setEntries(R.array.entries);
    list.setEntryValues(R.array.entry_values);

    rootScreen.addPreference(list);


    CheckBoxPreference chb2 = new CheckBoxPreference(this);
    chb2.setKey("chb2");
    chb2.setTitle("CheckBox 2");
    chb2.setSummary("Description of checkbox 2");

    rootScreen.addPreference(chb2);


    PreferenceScreen screen = getPreferenceManager().createPreferenceScreen(this);
    screen.setKey("screen");
    screen.setTitle("Screen");
    screen.setSummary("Description of screen");


    final CheckBoxPreference chb3 = new CheckBoxPreference(this);
    chb3.setKey("chb3");
    chb3.setTitle("CheckBox 3");
    chb3.setSummary("Description of checkbox 3");
```

```
    screen.addPreference(chb3);


    PreferenceCategory categ1 = new PreferenceCategory(this);
    categ1.setKey("categ1");
    categ1.setTitle("Category 1");
    categ1.setSummary("Description of category 1");

    screen.addPreference(categ1);

    CheckBoxPreference chb4 = new CheckBoxPreference(this);
    chb4.setKey("chb4");
    chb4.setTitle("CheckBox 4");
    chb4.setSummary("Description of checkbox 4");

    categ1.addPreference(chb4);


    final PreferenceCategory categ2 = new PreferenceCategory(this);
    categ2.setKey("categ2");
    categ2.setTitle("Category 2");
    categ2.setSummary("Description of category 2");

    screen.addPreference(categ2);


    CheckBoxPreference chb5 = new CheckBoxPreference(this);
    chb5.setKey("chb5");
    chb5.setTitle("CheckBox 5");
    chb5.setSummary("Description of checkbox 5");

    categ2.addPreference(chb5);


    CheckBoxPreference chb6 = new CheckBoxPreference(this);
    chb6.setKey("chb6");
    chb6.setTitle("CheckBox 6");
    chb6.setSummary("Description of checkbox 6");

    categ2.addPreference(chb6);

    rootScreen.addPreference(screen);

    list.setDependency("chb1");
    screen.setDependency("chb2");

    // код из прошлого урока для связи активности categ2 и значения chb3
    categ2.setEnabled(chb3.isChecked());
    chb3.setOnPreferenceClickListener(new OnPreferenceClickListener() {
      public boolean onPreferenceClick(Preference preference) {
        categ2.setEnabled(chb3.isChecked());
        return false;
      }
    });
  }
}
```

Сначала мы создаем PreferenceScreen и сообщаем Activity, что этот экран будет корневым.

Далее мы последовательно создаем Preference-элементы, присваиваем им атрибуты и добавляем детей к родителям.

В конце мы реализуем зависимости (dependency) и, как и на прошлом уроке, привязываем состояние активности категории ***categ2*** к значению чекбокса ***chb3***.

Метод setDependency лучше вызывать в самом конце. Иначе может быть неверное поведение элементов.

MainActivity.java:

```
package ru.startandroid.develop.p0741preferencescode;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  public boolean onCreateOptionsMenu(Menu menu) {
    MenuItem mi = menu.add(0, 1, 0, "Preferences");
    mi.setIntent(new Intent(this, PrefActivity.class));
    return super.onCreateOptionsMenu(menu);
  }

}
```

Запустив приложение, вы получите экран, аналогичный экрану с предыдущего урока. Только сделан программно, а не из xml.

Кстати, вы можете не только добавлять, но и удалять Preferences-элементы. Для этого у категорий и экранов есть методы removePreference и removeAll.

На следующем уроке:

- работаем с файлами

Урок 75. Хранение данных. Работа с файлами.

В этом уроке:

- работаем с файлами

Работа с файлами в Android не сильно отличается от таковой в Java. В этом уроке рассмотрим, как записать/прочесть файл во внутреннюю память и на флэшку.

Создадим проект:

Project name: P0751_Files
Build Target: Android 2.3.3
Application name: Files
Package name: ru.startandroid.develop.p0751files
Create Activity: MainActivity

Заполним **strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">P0751_Files</string>
    <string  name="write_file">Записать файл</string>
    <string  name="read_file">Прочесть файл</string>
    <string  name="write_file_sd">Записать файл на SD</string>
    <string  name="read_file_sd">Прочесть файл с SD</string>
</resources>
```

Рисуем экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <LinearLayout
         android:layout_width="match_parent"
         android:layout_height="wrap_content">
        <Button
             android:id="@+id/btnWrite"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="@string/write_file"
             android:onClick="onclick">
        </Button>
        <Button
             android:id="@+id/btnRead"
             android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="@string/read_file"
            android:onClick="onclick">
        </Button>
    </LinearLayout>
    <LinearLayout
         android:layout_width="match_parent"
         android:layout_height="wrap_content">
        <Button
            android:id="@+id/btnWriteSD"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/write_file_sd"
            android:onClick="onclick">
        </Button>
        <Button
            android:id="@+id/btnReadSD"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/read_file_sd"
            android:onClick="onclick">
        </Button>
    </LinearLayout>
</LinearLayout>
```

4 кнопки, смысл которых понятен по тексту на них.

MainActivity.java:

```
package ru.startandroid.develop.p0751files;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final String FILENAME = "file";
```

```
final String DIR_SD = "MyFiles";
final String FILENAME_SD = "fileSD";

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
}

public void onclick(View v) {
  switch (v.getId()) {
  case R.id.btnWrite:
    writeFile();
    break;
  case R.id.btnRead:
    readFile();
    break;
  case R.id.btnWriteSD:
    writeFileSD();
    break;
  case R.id.btnReadSD:
    readFileSD();
    break;
  }
}

void writeFile() {
  try {
    // отрываем поток для записи
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(
        openFileOutput(FILENAME, MODE_PRIVATE)));
    // пишем данные
    bw.write("Содержимое файла");
    // закрываем поток
    bw.close();
    Log.d(LOG_TAG, "Файл записан");
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }
}

void readFile() {
  try {
    // открываем поток для чтения
    BufferedReader br = new BufferedReader(new InputStreamReader(
        openFileInput(FILENAME)));
    String str = "";
    // читаем содержимое
    while ((str = br.readLine()) != null) {
      Log.d(LOG_TAG, str);
    }
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }
```

```
  }

  void writeFileSD() {
    // проверяем доступность SD
    if (!Environment.getExternalStorageState().equals(
        Environment.MEDIA_MOUNTED)) {
      Log.d(LOG_TAG, "SD-карта не доступна: " + Environment.getExternalStorageState());
      return;
    }
    // получаем путь к SD
    File sdPath = Environment.getExternalStorageDirectory();
    // добавляем свой каталог к пути
    sdPath = new File(sdPath.getAbsolutePath() + "/" + DIR_SD);
    // создаем каталог
    sdPath.mkdirs();
    // формируем объект File, который содержит путь к файлу
    File sdFile = new File(sdPath, FILENAME_SD);
    try {
      // открываем поток для записи
      BufferedWriter bw = new BufferedWriter(new FileWriter(sdFile));
      // пишем данные
      bw.write("Содержимое файла на SD");
      // закрываем поток
      bw.close();
      Log.d(LOG_TAG, "Файл записан на SD: " + sdFile.getAbsolutePath());
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  void readFileSD() {
    // проверяем доступность SD
    if (!Environment.getExternalStorageState().equals(
        Environment.MEDIA_MOUNTED)) {
      Log.d(LOG_TAG, "SD-карта не доступна: " + Environment.getExternalStorageState());
      return;
    }
    // получаем путь к SD
    File sdPath = Environment.getExternalStorageDirectory();
    // добавляем свой каталог к пути
    sdPath = new File(sdPath.getAbsolutePath() + "/" + DIR_SD);
    // формируем объект File, который содержит путь к файлу
    File sdFile = new File(sdPath, FILENAME_SD);
    try {
      // открываем поток для чтения
      BufferedReader br = new BufferedReader(new FileReader(sdFile));
      String str = "";
      // читаем содержимое
      while ((str = br.readLine()) != null) {
        Log.d(LOG_TAG, str);
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

В **onclick** обрабатываем нажатия 4-х кнопок и вызываем соответствующие методы.

writeFile – запись файла во внутреннюю память. Используется метод openFileOutput, который на вход берет имя файла и режим записи: MODE_PRIVATE – файл доступен только этому приложению, MODE_WORLD_READABLE – файл доступен для чтения всем, MODE_WORLD_WRITEABLE - файл доступен для записи всем, MODE_APPEND – файл будет дописан, а не начат заново.

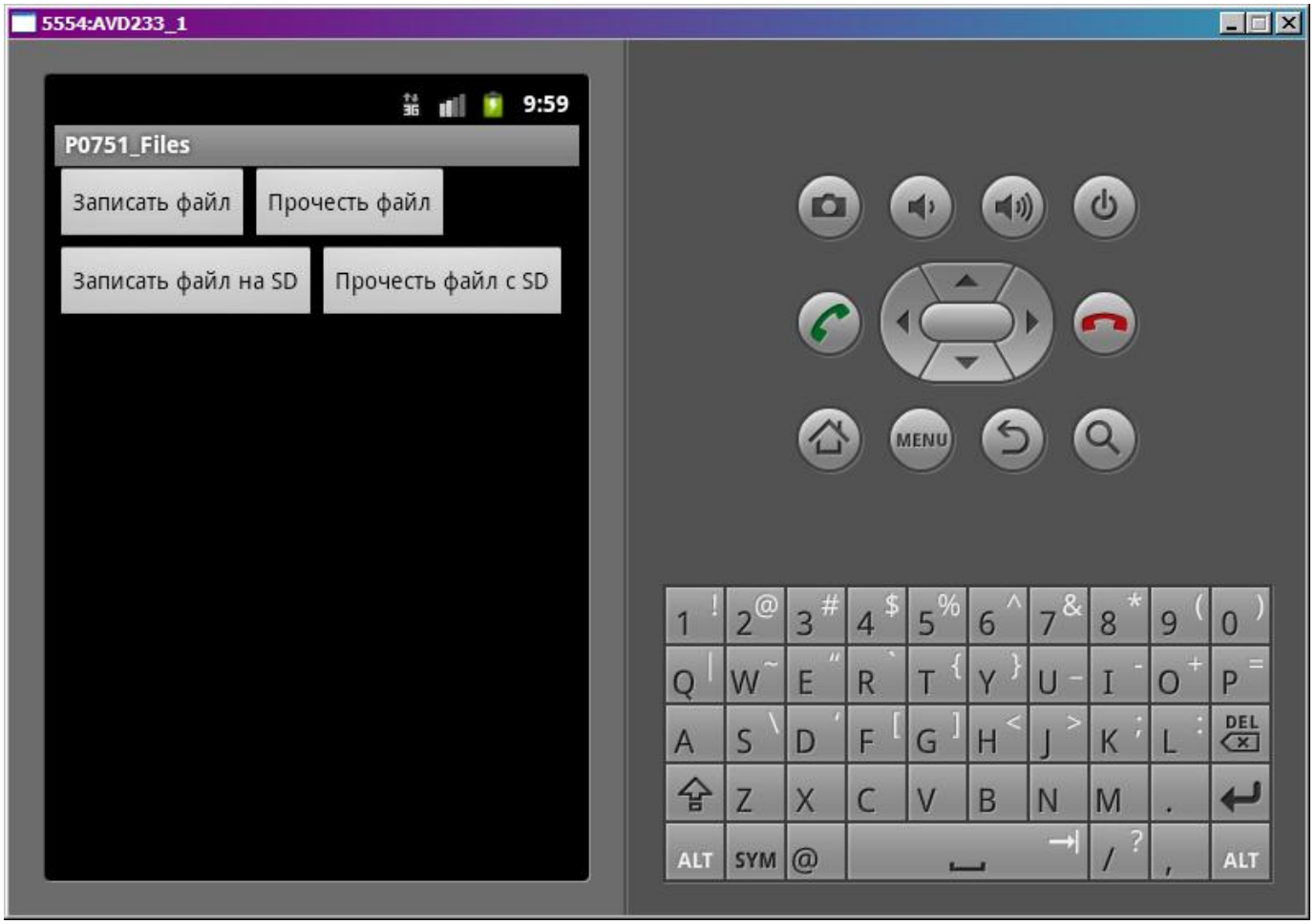
readFile – чтение файла из внутренней памяти. Используем метод openFileInput, принимающий на вход имя файла. Здесь и в методе записи внутреннего файла вы можете задать только имя файла, а каталог для ваших файлов вам уже выделен.

writeFileSD – запись файла на SD. Используем метод getExternalStorageState для получения состояния SD-карты. Здесь можно посмотреть какие бывают состояния. Нам нужно MEDIA_MOUNTED – когда SD-карта вставлена и готова к работе. Далее мы получаем путь к SD-карте (метод getExternalStorageDirectory), добавляем свой каталог и имя файла, создаем каталог и пишем данные в файл.

readFileSD – чтение файла с SD. Все аналогично предыдущему методу, только файл не пишем, а читаем.

Осталось в манифест добавить разрешение на работу с файлами на SD - android.permission.WRITE_EXTERNAL_STORAGE.

Все сохраним и запустим. Видим экран с 4-мя кнопками:



Внутренняя память

Жмем кнопку **Записать файл**. Видим в логе:

Файл записан

Проверим. Идем в **File Explorer** (Window > Show View > Other > Android > File Explorer) и открываем там папку **data/data/ru.startandroid.develop.p0751files/files** и видим там наш файл **file**.

Возвращаемся в эмулятор. Жмем **Прочесть файл** и в логе видим:

Содержимое файла

Это тот текст, который мы записывали в файл.

SD карта

Теперь жмем **Записать файл на SD**.

В логе видим:

Файл записан на SD: /mnt/sdcard/MyFiles/fileSD

Проверяем. Идем в **FileExplorer** и открываем там папку **mnt/sdcard/MyFiles/** а в ней файл **fileSD**.

Возвращаемся в эмулятор и жмем кнопку Прочесть файл с SD. В логе видим:

Содержимое файла на SD

Этот текст мы и записывали.

mnt/sdcard - обычно этот путь ведет к содержимому SD-карты. Возможно у вас он будет другой.

В общем, при работе с файлами на SD вы используете стандартные java механизмы. А при работе с внутренним хранилищем для удобства можно использовать методы-оболочки от Activity:

openFileOutput – открыть файл на запись

openFileInput – открыть файл на чтение

deleteFile – удалить файл

И есть метод getFilesDir – возвращает объект File, соответствующий каталогу для файлов вашей программы. Используйте его, чтобы работать напрямую, без методов-оболочек.

Подробности работы в java с файловой системой я здесь описывать не буду. На нашем форуме пользователь SKR сделал отличную памятку по работе с файлами. Скорее всего, вы найдете там все что нужно.

Если у вас проверка SD-карты показывает, что карта недоступна (см. лог), то убедитесь в свойствах AVD, что у вас для SDCard указан Size или File. Если указаны, то попробуйте перезапустить AVD.

На следующем уроке:

- создаем экран с вкладками
- используем иконку в названии вкладки
- используем обработчик перехода между вкладками

Урок 76. Tab - вкладки. Общий обзор

В этом уроке:

- создаем экран с вкладками
- используем иконку в названии вкладки
- используем обработчик перехода между вкладками

Вкладки помогают логически разделить содержимое экрана. Вместо того, чтобы бегать по разным экранам, вы можете сделать вкладки и переключаться между ними. В этом уроке создадим приложение с вкладками и посмотрим их основные возможности.

Создадим проект:

Project name: P0761_Tab
Build Target: Android 2.3.3
Application name: Tab
Package name: ru.startandroid.develop.p0761tab
Create Activity: MainActivity

Пропишем тексты в **strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">Tab</string>
    <string  name="text_tab1">Это первая вкладка</string>
    <string  name="text_tab2">Это вторая вкладка</string>
    <string  name="text_tab3">Это третья вкладка</string>
    <string  name="text_tab_header">Свой заголовок</string>
</resources>
```

Удалим все дефолтное с экрана **main.xml** и добавим туда компонент **TabHost** из вкладки **Composite**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <TabHost
         android:id="@android:id/tabhost"
         android:layout_width="match_parent"
         android:layout_height="match_parent">
        <LinearLayout
             android:layout_width="match_parent"
             android:layout_height="match_parent"
             android:orientation="vertical">
            <TabWidget
                 android:id="@android:id/tabs"
                 android:layout_width="match_parent"
                 android:layout_height="wrap_content">
            </TabWidget>
            <FrameLayout
                 android:id="@android:id/tabcontent"
```

```
                android:layout_width="match_parent"
                android:layout_height="match_parent">
                <LinearLayout
                    android:id="@+id/tab1"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent">
                </LinearLayout>
                <LinearLayout
                    android:id="@+id/tab2"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent">
                </LinearLayout>
                <LinearLayout
                    android:id="@+id/tab3"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent">
                </LinearLayout>
            </FrameLayout>
        </LinearLayout>
    </TabHost>
</LinearLayout>
```

Компонент добавился и притащил с собой еще кучу всего. Давайте смотреть. TabHost – корневой элемент вкладок. В нем вертикальный LinearLayout, в котором расположены TabWidget и FrameLayout. TabWidget будет отображать заголовки вкладок, а FrameLayout – содержимое вкладок. В этом FrameLayout мы размещаем все View-компоненты, которые хотим отображать на вкладках. Позже мы (в коде) сообщим вкладке, какой именно компонент она должна показать (явно укажем id), вкладка выберет из этой общей кучи нужный ей компонент и отобразит его, как свое содержимое.

По дефолту во FrameLayout созданы три LinearLayout – они могут быть использованы, как контейнеры для содержимого вкладок. Т.е. вы их заполняете компонентами, как вам необходимо, а потом в коде просто указываете id нужного LinearLayout-а и он со всем содержимым отобразится на вкладке.

Нам сейчас не нужны LInearLayout, мы не будем делать вкладки с сложным содержимым, разместим во FrameLayout просто несколько TextView.

В итоге **main.xml** получился такой:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <TabHost
         android:id="@android:id/tabhost"
         android:layout_width="match_parent"
         android:layout_height="match_parent">
        <LinearLayout
             android:layout_width="match_parent"
             android:layout_height="match_parent"
             android:orientation="vertical">
            <TabWidget
                 android:id="@android:id/tabs"
                 android:layout_width="match_parent"
                 android:layout_height="wrap_content">
            </TabWidget>
            <FrameLayout
                 android:id="@android:id/tabcontent"
                 android:layout_width="match_parent"
```

```
                android:layout_height="match_parent">
                <TextView
                    android:id="@+id/tvTab1"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/text_tab1">
                </TextView>
                <TextView
                    android:id="@+id/tvTab2"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/text_tab2">
                </TextView>
                <TextView
                    android:id="@+id/tvTab3"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/text_tab3">
                </TextView>
            </FrameLayout>
        </LinearLayout>
    </TabHost>
</LinearLayout>
```

Создадим еще один layout-файл - **tab_header.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/text_tab_header">
    </TextView>
</LinearLayout>
```

Этот layout мы используем как свой экран для заголовка вкладки. Тут просто TextView.

Создайте в папке **res** папку **drawable**, если ее нет. В ней создайте файл **tab_icon_selector.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<selector  xmlns:android="http://schemas.android.com/apk/res/android">
    <item  android:drawable="@android:drawable/star_on"  android:state_selected="true"></item>
    <item  android:drawable="@android:drawable/star_off"></item>
</selector>
```

Подробно об этом можно почитать тут. Этот xml-файл мы укажем как картинку для заголовка вкладки. И когда система будет прорисовывать заголовок вкладки, она обратится к этому файлу, чтобы понять какую картинку ей отображать. Этот код будет возвращать стандартную Android картинку star_on, если вкладка выбрана (state_selected="true"). Иначе вернет star_off. Далее увидим это в приложении, и станет понятней.

Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p0761tab;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TabHost;
import android.widget.TabHost.OnTabChangeListener;
import android.widget.Toast;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        TabHost tabHost = (TabHost) findViewById(android.R.id.tabhost);
        // инициализация
        tabHost.setup();

        TabHost.TabSpec tabSpec;

        // создаем вкладку и указываем тег
        tabSpec = tabHost.newTabSpec("tag1");
        // название вкладки
        tabSpec.setIndicator("Вкладка 1");
        // указываем id компонента из FrameLayout, он и станет содержимым
        tabSpec.setContent(R.id.tvTab1);
        // добавляем в корневой элемент
        tabHost.addTab(tabSpec);

        tabSpec = tabHost.newTabSpec("tag2");
        // указываем название и картинку
        // в нашем случае вместо картинки идет xml-файл,
        // который определяет картинку по состоянию вкладки
        tabSpec.setIndicator("Вкладка 2", getResources().getDrawable(R.drawable.tab_icon_selector));
        tabSpec.setContent(R.id.tvTab2);
        tabHost.addTab(tabSpec);

        tabSpec = tabHost.newTabSpec("tag3");
        // создаем View из layout-файла
        View v = getLayoutInflater().inflate(R.layout.tab_header, null);
        // и устанавливаем его, как заголовок
        tabSpec.setIndicator(v);
        tabSpec.setContent(R.id.tvTab3);
        tabHost.addTab(tabSpec);

        // вторая вкладка будет выбрана по умолчанию
        tabHost.setCurrentTabByTag("tag2");

        // обработчик переключения вкладок
        tabHost.setOnTabChangedListener(new OnTabChangeListener() {
      public void onTabChanged(String tabId) {
        Toast.makeText(getBaseContext(), "tabId = " + tabId, Toast.LENGTH_SHORT).show();
      }
    });
    }
}
```

Находим компонент **TabHost**. Обратите внимание, используется андроидный id. Он был таким по умолчанию при добавлении компонента в **main.xml**. В принципе, в нашем случае, этот id можно сменить на свой. Далее вызываем обязательный метод

setup. Это первичная инициализация. В этом методе **TabHost** находит в себе **TabWidget** и **FrameLayout**. Вот их id в main.xml менять нельзя. Иначе TabHost будет ругаться, что не может их найти.

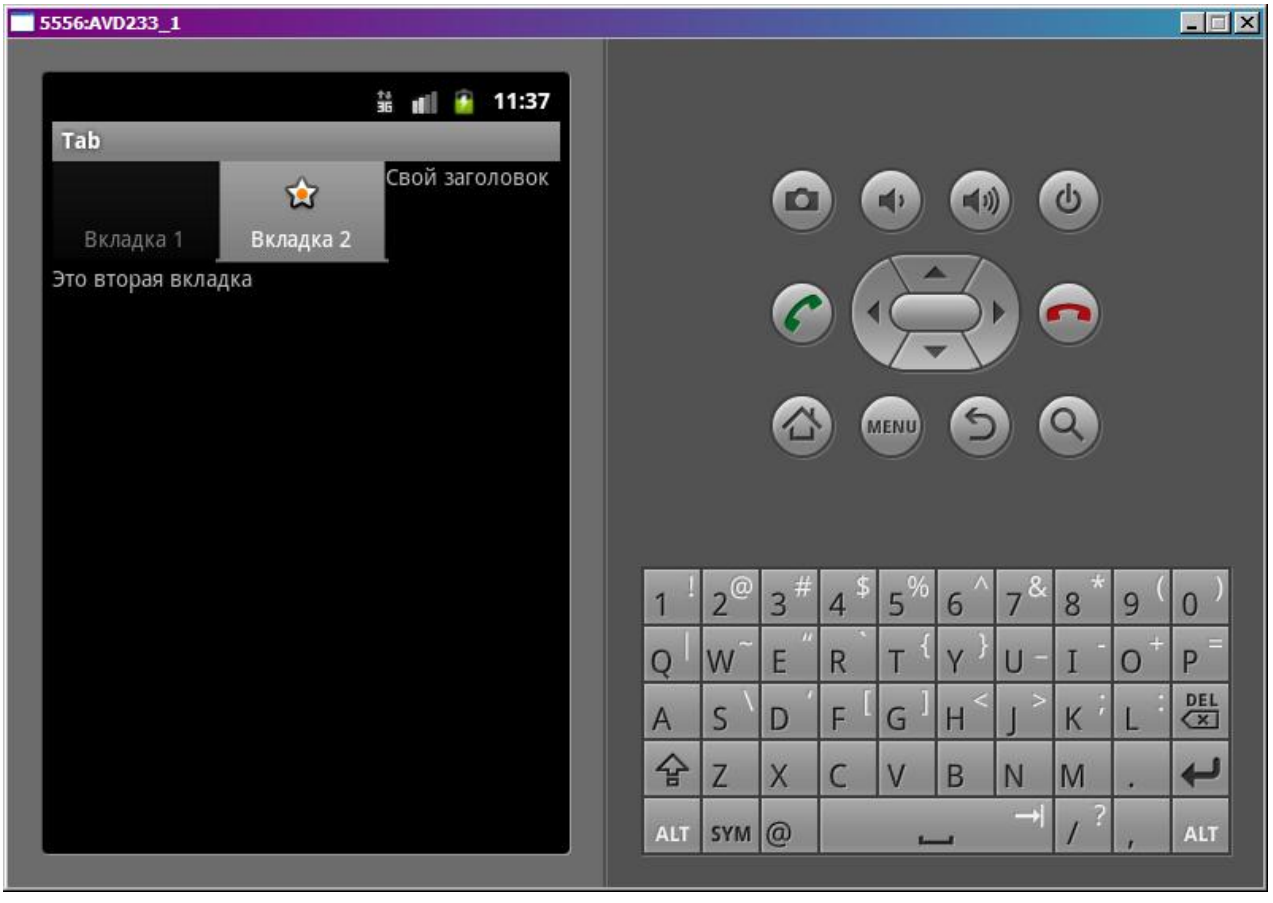
Далее создаем три вкладки. Для создания используется метод newTabSpec, на вход он берет тэг. Тэг – это просто некий строковый идентификатор вкладки. Позже увидим, где он используется. Для первой вкладки задаем название методом setIndicator. В метод setContent передаем id компонента (из FrameLayout), который мы хотели бы видеть в качестве содержимого вкладки. В нашем случае это TextView. Метод addTab присоединяет готовую вкладку к TabHost.

Вторая вкладка создается аналогично, только используем другую реализацию метода setIndicator. Заголовок вкладки может содержать не только текст, но и картинку. И здесь мы это используем – передаем в метод текст и xml вместо картинки. Тот самый xml, который определяет картинку по состоянию вкладки. Разумеется, если вам нужна статичная картинка, вы можете указать ее и не использовать xml вообще.

При создании третьей вкладки используем еще одну реализацию метода setIndicator, которая берет на вход View и его ставит как заголовок. Используем тут наш layout-файл tab_header.

Вкладки созданы. Устанавливаем (setCurrentTabByTag) вторую в качестве выбранной по умолчанию. И пропишем (setOnTabChangedListener) для TabHost обработчик, который срабатывает при переключении вкладок. Будем выводить сообщение с тэгом вкладки.

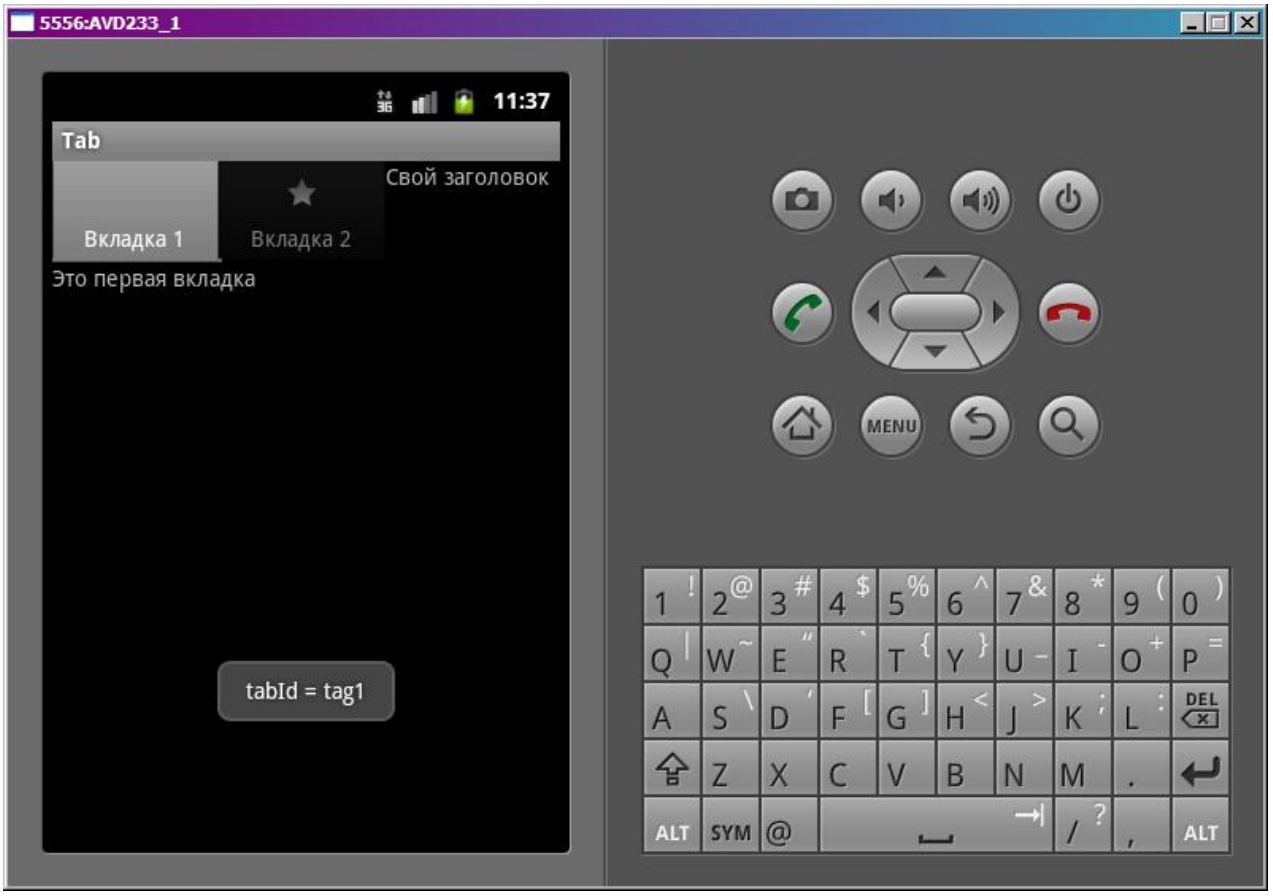
Все сохраним и запустим приложение.



Выбрана вторая вкладка, т.к. мы это определили методом **setCurrentTabByTag**. Ее содержимое – это TextView с id = tvTab2, как мы и указывали в коде в методе setContent при создании вкладки.

У третьей вкладки заголовок соответствует содержимому **tab_header**, т.к. мы использовали **setIndicator**, который принимает на вход **View**.

Выберем первую вкладку.



Сработал обработчик и появилось сообщение с тэгом выбранной вкладки. Содержимое первой вкладки – это TextView с id = tvTab1 из FrameLayout.

Обратите внимание, что сменилась картинка на заголовке второй вкладки. Это обеспечил **selector** из res/drawable/tab_icon_selector. В зависимости от состояния вкладки он выдает разные картинки.

На следующем уроке:

- используем Activity в качестве содержимого вкладки
- используем TabActivity

Урок 77. Tab - вкладки. TabActivity. Activity, как содержимое вкладки

В этом уроке:

- используем Activity в качестве содержимого вкладки
- используем TabActivity

В качестве вкладки можно использовать Activity. Для этого существует реализация метода setContent, которая принимает на вход Intent. А в Intent мы прописываем, какое Activity нам нужно.

При использовании Intent и вкладок есть нюанс. Наше основное Activity, которое содержит TabHost должно наследовать не android.app.Activity как обычно, а android.app.TabActivity. В этом случае нам не надо заморачиваться с дополнительной инициализацией для работы с Intent.

Создадим проект:

Project name: P0771_TabIntent
Build Target: Android 2.3.3
Application name: TabIntent
Package name: ru.startandroid.develop.p0771tabintent
Create Activity: MainActivity

Пропишем тексты в **strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">TabIntent</string>
    <string  name="text_activity1">Это первое Activity</string>
    <string  name="text_activity2">Это второе Activity</string>
</resources>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <TabHost
         android:id="@android:id/tabhost"
         android:layout_width="match_parent"
         android:layout_height="match_parent">
        <LinearLayout
             android:layout_width="match_parent"
             android:layout_height="match_parent"
             android:orientation="vertical">
            <TabWidget
                 android:id="@android:id/tabs"
```

```
                android:layout_width="match_parent"
                android:layout_height="wrap_content">
            </TabWidget>
            <FrameLayout
                android:id="@android:id/tabcontent"
                android:layout_width="match_parent"
                android:layout_height="match_parent">
            </FrameLayout>
        </LinearLayout>
    </TabHost>
</LinearLayout>
```

Менять id у TabHost в случае использования android.app.TabActivity нельзя. Иначе система просто не найдет TabHost.

FrameLayout не заполняем, т.к. мы не будем использовать его компоненты для содержимого вкладок. Мы будем туда целые Activity грузить.

Создадим пару Activity.

layout-файлы

one.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/text_activity1">
    </TextView>
</LinearLayout>
```

two.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/text_activity2">
    </TextView>
</LinearLayout>
```

Классы

OneActivity.java:

```
package ru.startandroid.develop.p0771tabintent;

import android.app.Activity;
import android.os.Bundle;

public class OneActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.one);
  }
}
```

TwoActivity.java:

```
package ru.startandroid.develop.p0771tabintent;

import android.app.Activity;
import android.os.Bundle;

public class TwoActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.two);
  }
}
```

Не забываем прописать их в манифесте.

Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p0771tabintent;

import android.app.TabActivity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TabHost;

public class MainActivity extends TabActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

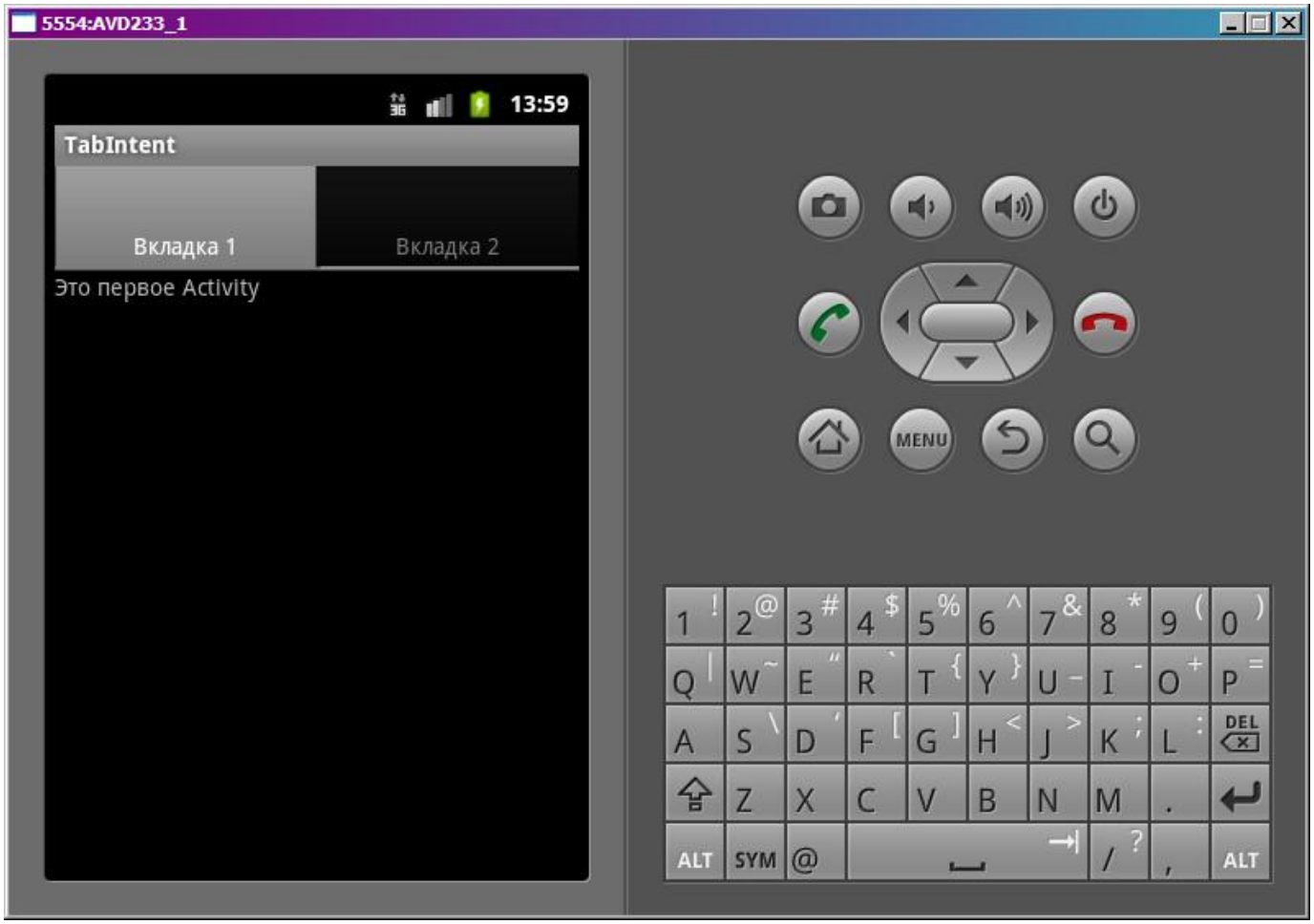
```
        // получаем TabHost
        TabHost tabHost = getTabHost();

        // инициализация была выполнена в getTabHost
        // метод setup вызывать не нужно

        TabHost.TabSpec tabSpec;

        tabSpec = tabHost.newTabSpec("tag1");
        tabSpec.setIndicator("Вкладка 1");
        tabSpec.setContent(new Intent(this, OneActivity.class));
        tabHost.addTab(tabSpec);

        tabSpec = tabHost.newTabSpec("tag2");
        tabSpec.setIndicator("Вкладка 2");
        tabSpec.setContent(new Intent(this, TwoActivity.class));
        tabHost.addTab(tabSpec);
    }
}
```

Наше Activity наследует **TabActivity**. Это дает нам возможность получить TabHost методом getTabHost. Нам не нужно самим искать его на экране. Также этот замечательный метод выполняет за нас обычную инициализацию, нам не надо вызывать метод setup, как на прошлом уроке. И кроме обычной инициализации, этот метод выполняет подготовку для работы с Activity, как содержимым вкладок.

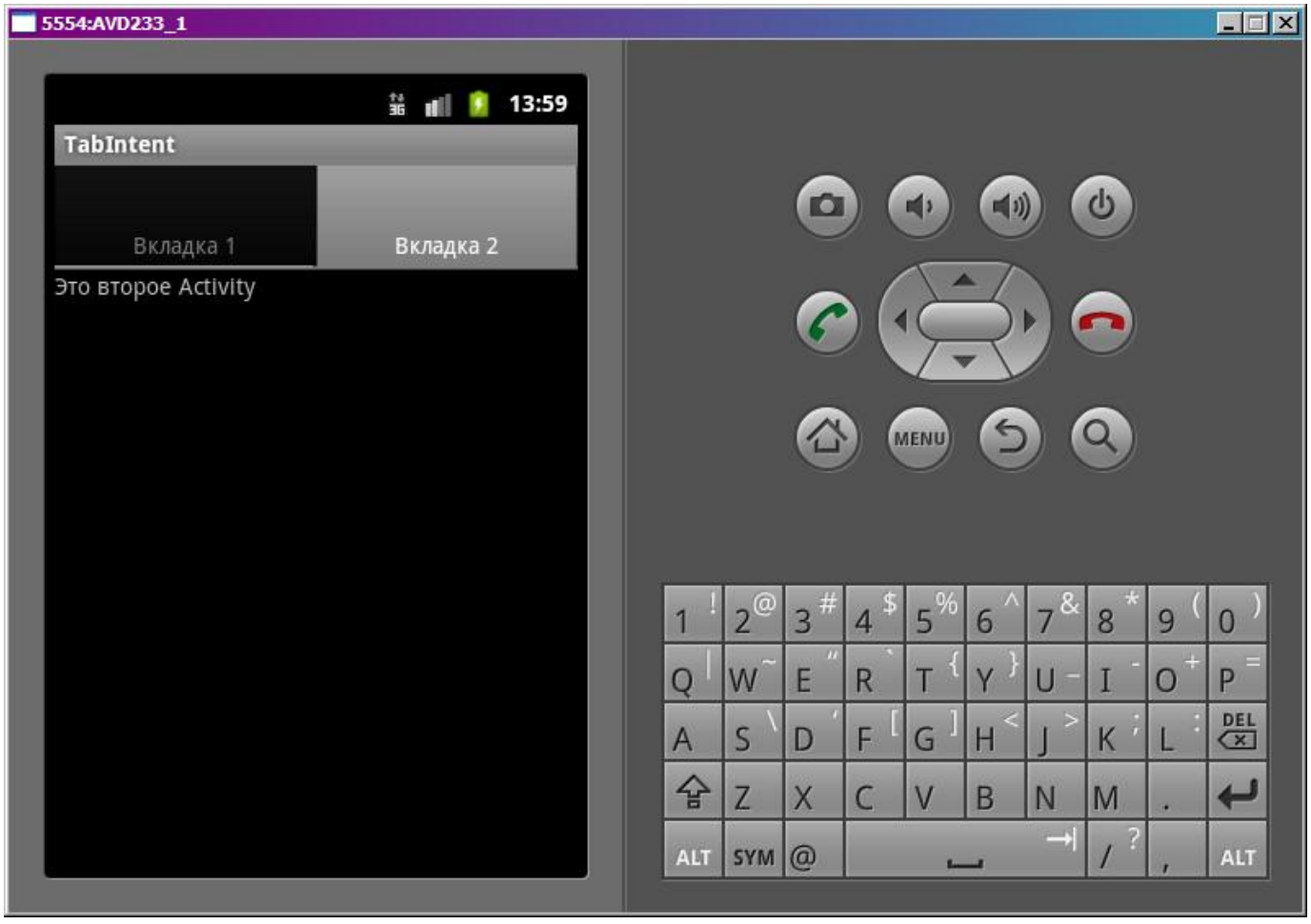
Ну а далее вам все знакомо с прошлого урока. Создаем вкладки, указываем имена. В методе setContent вместо содержимого из FrameLayout мы даем Intent, указывающий на необходимое нам Activity.

Все сохраняем и запускаем приложение.

Первая вкладка с **OneActivity**



Вторая вкладка с **TwoActivity**



А как отрабатывают события Activity LifeCycle? При первом показе первой вкладки срабатывают три метода **OneActivity**: **onCreate**, **onStart**, **onResume**. Переключаемся на вторую вкладку – срабатывает **onPause** в **OneActivity**, а потом три метода **TwoActivity**: **onCreate**, **onStart**, **onResume**. И далее при переключениях между вкладками одна уходит в **onPause**, другая возвращается в **onResume**.

На следующем уроке:

- вручную создаем содержимое вкладки

Урок 78. Tab - вкладки. TabContentFactory, ручное создание содержимого вкладки

В этом уроке:

- вручную создаем содержимое вкладки

При работе с вкладками система предоставляет нам возможность самим создать View, которое будет использовано в качестве содержимого вкладки. Нам надо просто создать объект, реализующий интерфейс TabContentFactory, и накодить метод createTabContent. Этот метод на вход берет тэг, а выдать должен View.

View мы умеем создавать двумя способами: 1) непосредственно ручное **создание объекта**; 2) с помощью **LayoutInflater**.

Создадим проект:

Project name: P0781_TabContentFactory
Build Target: Android 2.3.3
Application name: TabContentFactory
Package name: ru.startandroid.develop.p0781tabcontentfactory
Create Activity: MainActivity

Пропишем текст в **strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">TabContentFactory</string>
    <string  name="text_tab">Это создано с помощью LayoutInflater </string>
</resources>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <TabHost
         android:id="@android:id/tabhost"
         android:layout_width="match_parent"
         android:layout_height="match_parent">
        <LinearLayout
             android:layout_width="match_parent"
             android:layout_height="match_parent"
             android:orientation="vertical">
            <TabWidget
                 android:id="@android:id/tabs"
                 android:layout_width="match_parent"
```

```
                android:layout_height="wrap_content">
            </TabWidget>
            <FrameLayout
                android:id="@android:id/tabcontent"
                android:layout_width="match_parent"
                android:layout_height="match_parent">
            </FrameLayout>
        </LinearLayout>
    </TabHost>
</LinearLayout>
```

FrameLayout не заполняем, т.к. сами будем View создавать.

Создадим layout-файл **tab.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:orientation="vertical">
    <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/text_tab">
    </TextView>
</LinearLayout>
```

Из него мы с помощью LayoutInflater будем получать View и устанавливать в качестве содержимого вкладки.

Кодим **MainActivity.java**:

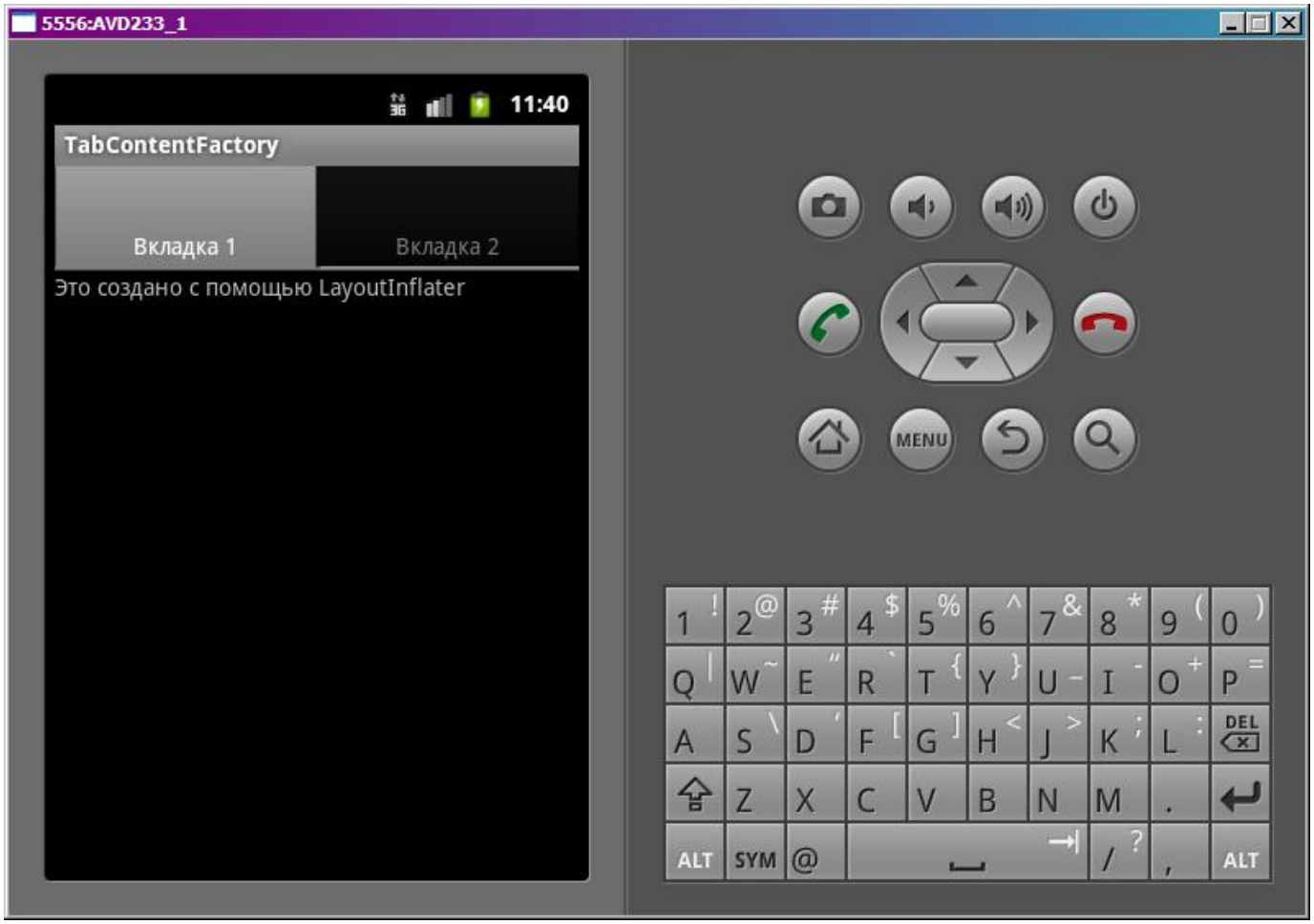
```
package ru.startandroid.develop.p0781tabcontentfactory;

import android.app.TabActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TabHost;
import android.widget.TextView;

public class MainActivity extends TabActivity {

  final String TABS_TAG_1 = "Tag 1";
  final String TABS_TAG_2 = "Tag 2";


    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        TabHost tabHost = getTabHost();
```

```
        TabHost.TabSpec tabSpec;

        tabSpec = tabHost.newTabSpec(TABS_TAG_1);
        tabSpec.setContent(TabFactory);
        tabSpec.setIndicator("Вкладка 1");
        tabHost.addTab(tabSpec);

        tabSpec = tabHost.newTabSpec(TABS_TAG_2);
        tabSpec.setContent(TabFactory);
        tabSpec.setIndicator("Вкладка 2");
        tabHost.addTab(tabSpec);

    }

    TabHost.TabContentFactory TabFactory = new TabHost.TabContentFactory() {

    @Override
    public View createTabContent(String tag) {
      if (tag == TABS_TAG_1) {
        return getLayoutInflater().inflate(R.layout.tab, null);
      } else if (tag == TABS_TAG_2) {
        TextView tv = new TextView(MainActivity.this);
        tv.setText("Это создано вручную");
        return tv;
      }
      return null;
    }
  };
}
```

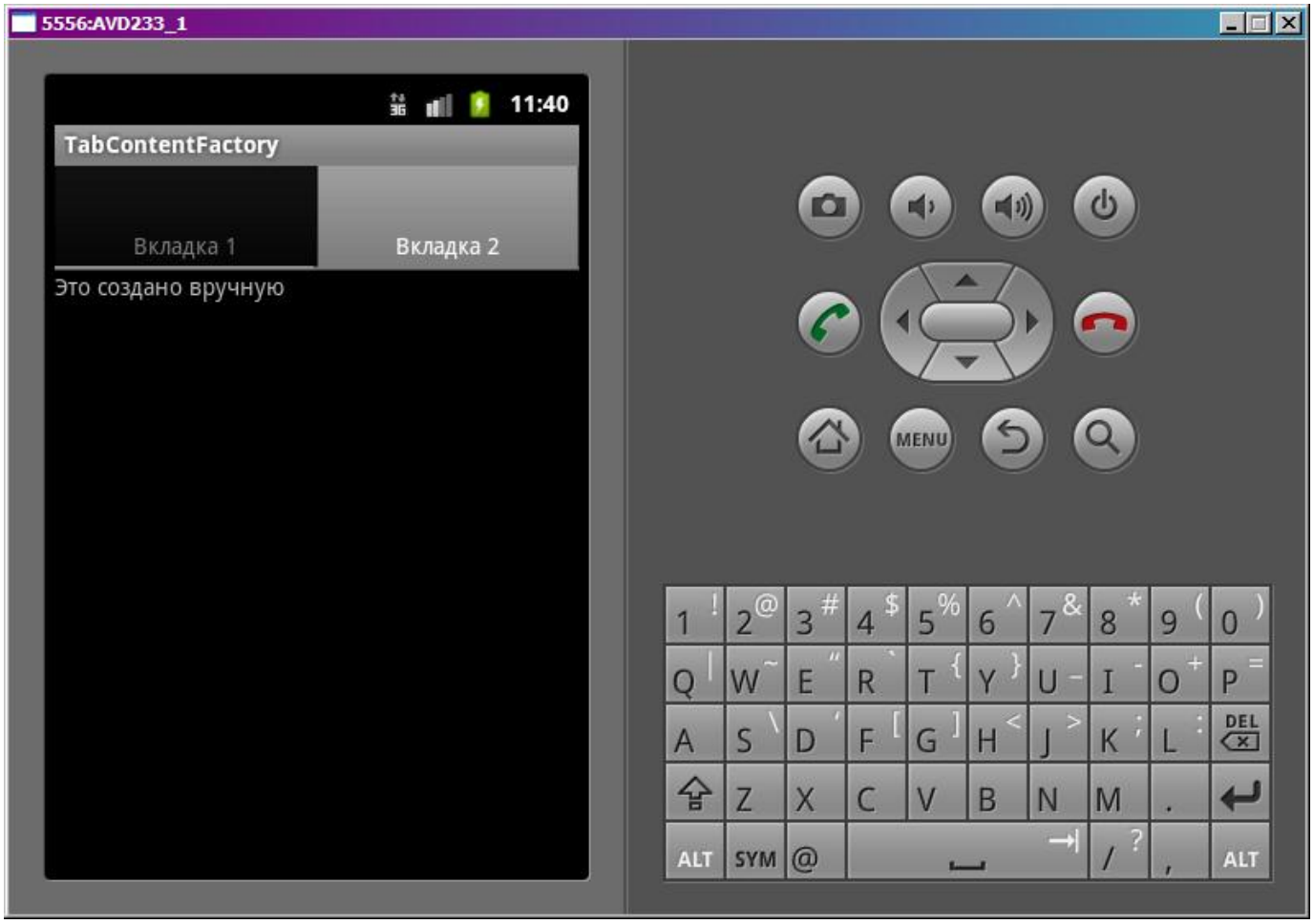
Создание вкладок идет как обычно, только методу setContent на вход передаем объект, реализующий **TabHost.TabContentFactory**. Когда системе необходимо будет построить содержимое вкладки, она воспользуется этим объектом - вызовет его метод **createTabContent**. На вход ему пойдет тэг вкладки, на выходе должно получиться View. И наша задача – реализовать это.

TabFactory – объект, реализующий интерфейс TabHost.TabContentFactory. В его методе createTabContent мы проверяем тэг. Если это тэг первой вкладки, то прогоняем **tab.xml** через **LayoutInflater** и возвращаем получившееся **View**. Если тэг второй вкладки, то вручную создаем **TextView** и возвращаем его.

Все сохраним и запустим приложение. Первая вкладка:



Вторая вкладка:



В итоге мы создали содержимое **первой** вкладки из **layout**-файла, а содержимое **второй** – сами, через **создание объектов**.

На следующем уроке:

- парсим XML с помощью XmlPullParser

Урок 79. XmlPullParser. Парсим XML

В этом уроке:

- парсим XML с помощью XmlPullParser

XmlPullParser – XML-парсер, который можно использовать для разбора XML документа. Принцип его работы заключается в том, что он пробегает весь документ, останавливаясь на его элементах. Но пробегает он не сам, а с помощью метода next. Мы постоянно вызываем метод next и с помощью метода getEventType проверяем, на каком элементе парсер остановился.

Основные элементы документа, которые ловит парсер:

START_DOCUMENT – начало документа

START_TAG – начало тэга

TEXT – содержимое элемента

END_TAG – конец тэга

END_DOCUMENT – конец документа

Напишем приложение, которое возьмет xml-файл и разберет его на тэги и аттрибуты.

Создадим проект:

Project name: P0791_ XmlPullParser
Build Target: Android 2.3.3
Application name: XmlPullParser
Package name: ru.startandroid.develop.p0791xmlpullparser
Create Activity: MainActivity

В папке **res** создайте папку **xml**, и в ней создайте файл **data.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<data>
    <phone>
        <company>Samsung</company>
        <model>Galaxy</model>
        <price>18000</price>
        <screen  multitouch="yes"  resolution="320x480">3</screen>
        <colors>
            <color>black</color>
            <color>white</color>
        </colors>
    </phone>
</data>
```

Это файл с описанием телефона Samsung Galaxy. Указаны его цена, характеристики экрана и возможные цвета

корпуса. Данные выдуманы и могут не совпадать с реальностью :)

MainActivity.java:

```
package ru.startandroid.develop.p0791xmlpullparser;

import java.io.IOException;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;

import android.app.Activity;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    String tmp = "";

    try {
      XmlPullParser xpp = prepareXpp();

      while (xpp.getEventType() != XmlPullParser.END_DOCUMENT) {
        switch (xpp.getEventType()) {
        // начало документа
        case XmlPullParser.START_DOCUMENT:
          Log.d(LOG_TAG, "START_DOCUMENT");
          break;
        // начало тэга
        case XmlPullParser.START_TAG:
          Log.d(LOG_TAG, "START_TAG: name = " + xpp.getName()
              + ", depth = " + xpp.getDepth() + ", attrCount = "
              + xpp.getAttributeCount());
          tmp = "";
          for (int i = 0; i < xpp.getAttributeCount(); i++) {
            tmp = tmp + xpp.getAttributeName(i) + " = "
                + xpp.getAttributeValue(i) + ", ";
          }
          if (!TextUtils.isEmpty(tmp))
            Log.d(LOG_TAG, "Attributes: " + tmp);
          break;
        // конец тэга
        case XmlPullParser.END_TAG:
          Log.d(LOG_TAG, "END_TAG: name = " + xpp.getName());
          break;
        // содержимое тэга
        case XmlPullParser.TEXT:
          Log.d(LOG_TAG, "text = " + xpp.getText());
          break;
```

```
          default:
            break;
          }
          // следующий элемент
          xpp.next();
        }
        Log.d(LOG_TAG, "END_DOCUMENT");

      } catch (XmlPullParserException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }

    XmlPullParser prepareXpp() {
      return getResources().getXml(R.xml.data);
    }
}
```

В **onCreate** мы получаем **XmlPullParser** с помощью метода prepareXpp и начинаем его разбирать. Затем в цикле while мы запускаем прогон документа, пока не достигнем конца - **END_DOCUMENT**. Прогон обеспечивается методом **next** в конце цикла while. В switch мы проверяем на каком элементе остановился парсер.

START_DOCUMENT – начало документа

START_TAG – начало тега. Выводим в лог имя тэга, его уровень в дереве тэгов (глубину) и количество атрибутов. Следующей строкой выводим имена и значения атрибутов, если они есть.

END_TAG – конец тэга. Выводим только имя.

TEXT – содержимое тэга

В методе **prepareXpp** мы подготавливаем XmlPullParser. Для этого вытаскиваем данные из папки **res/xml**. Это аналогично вытаскиванию строк или картинок – сначала получаем доступ к ресурсам (getResources), затем вызываем метод, соответствующий ресурсу. В нашем случае это - метод getXml. Но возвращает он не xml-строку , а готовый XmlPullParser.

Все сохраним и запустим приложение.

Смотрим лог:

START_DOCUMENT
START_DOCUMENT
START_TAG: name = data, depth = 1, attrCount = 0
START_TAG: name = phone, depth = 2, attrCount = 0
START_TAG: name = company, depth = 3, attrCount = 0
text = Samsung
END_TAG: name = company
START_TAG: name = model, depth = 3, attrCount = 0
text = Galaxy
END_TAG: name = model

START_TAG: name = price, depth = 3, attrCount = 0
text = 18000
END_TAG: name = price
START_TAG: name = screen, depth = 3, attrCount = 2
Attributes: multitouch = yes, resolution = 320x480,
text = 3
END_TAG: name = screen
START_TAG: name = colors, depth = 3, attrCount = 0
START_TAG: name = color, depth = 4, attrCount = 0
text = black
END_TAG: name = color
START_TAG: name = color, depth = 4, attrCount = 0
text = white
END_TAG: name = color
END_TAG: name = colors
END_TAG: name = phone
END_TAG: name = data
END_DOCUMENT
START_DOCUMENT
START_DOCUMENT
START_TAG: name = data, depth = 1, attrCount = 0
START_TAG: name = phone, depth = 2, attrCount = 0
START_TAG: name = company, depth = 3, attrCount = 0
text = Samsung
END_TAG: name = company
START_TAG: name = model, depth = 3, attrCount = 0
text = Galaxy
END_TAG: name = model
START_TAG: name = price, depth = 3, attrCount = 0
text = 18000
END_TAG: name = price
START_TAG: name = screen, depth = 3, attrCount = 2
Attributes: multitouch = yes, resolution = 320x480,
text = 3
END_TAG: name = screen
START_TAG: name = colors, depth = 3, attrCount = 0
START_TAG: name = color, depth = 4, attrCount = 0
text = black
END_TAG: name = color
START_TAG: name = color, depth = 4, attrCount = 0
text = white
END_TAG: name = color
END_TAG: name = colors
END_TAG: name = phone
END_TAG: name = data
END_DOCUMENT

START_DOCUMENT срабатывает два раза по неведомым мне причинам. Далее можно наблюдать, как парсер останавливается в начале каждого тега и дает нам информацию о нем: имя, уровень (глубина), количество атрибутов, имена и названия атрибутов, текст. Также он останавливается в конце тега и мы выводим имя. В конце парсер говорит, что документ закончен **END_DOCUMENT**.

Если xml у вас не в файле, а получен откуда-либо, то XmlPullParser надо создавать другим способом. Перепишем метод prepareXpp:

```
XmlPullParser prepareXpp() throws XmlPullParserException {
  // получаем фабрику
  XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
  // включаем поддержку namespace (по умолчанию выключена)
  factory.setNamespaceAware(true);
  // создаем парсер
  XmlPullParser xpp = factory.newPullParser();
  // даем парсеру на вход Reader
  xpp.setInput(new StringReader(
      "<data><phone><company>Samsung</company></phone></data>"));
  return xpp;
}
```

Здесь мы сами создаем парсер с помощью фабрики, включаем поддержку namespace (в нашем случае это не нужно, на всякий случай показываю) и даем парсеру на вход поток из xml-строки (укороченный вариант data.xml).

Все сохраним и запустим. Смотрим лог:

START_DOCUMENT
START_TAG: name = data, depth = 1, attrCount = 0
START_TAG: name = phone, depth = 2, attrCount = 0
START_TAG: name = company, depth = 3, attrCount = 0
text = Samsung
END_TAG: name = company
END_TAG: name = phone
END_TAG: name = data
END_DOCUMENT

Здесь уже START_DOCUMENT сработал один раз, как и должно быть. Ну и далее идут данные элементов документа.

Урок 80. Handler. Немного теории. Наглядный пример использования

В этом уроке:

- разбираемся, что такое Handler и зачем он нужен

Для полного понимания урока желательно иметь представление о **потоках** (threads) в Java.

Так просто ведь и не объяснишь, что такое **Handler**. Можете попробовать почитать официальное описание, но там достаточно нетривиально и мало написано. Я попробую здесь в двух словах рассказать.

В Android к **потоку** (thread) может быть привязана **очередь сообщений**. Мы можем помещать туда **сообщения**, а система будет за очередью следить и отправлять сообщения на **обработку**. При этом мы можем указать, чтобы сообщение ушло на обработку не сразу, а спустя определенное кол-во времени.

Handler - это механизм, который позволяет **работать** с очередью сообщений. Он привязан к конкретному потоку (thread) и работает с его очередью. Handler умеет **помещать** сообщения в очередь. При этом он ставит самого себя в качестве **получателя** этого сообщения. И когда приходит время, система достает сообщение из очереди и **отправляет** его адресату (т.е. в Handler) на обработку.

Handler дает нам две интересные и полезные возможности:

1) реализовать **отложенное по времени** выполнение кода

2) выполнение кода **не в своем потоке**

Подозреваю, что стало не сильно понятнее, что такое Handler, а главное – зачем он вообще нужен :) . В ближайшие несколько уроков будем с этим разбираться, и все станет понятно.

В этом уроке сделаем небольшое приложение. Оно будет эмулировать какое-либо долгое действие, например закачку файлов и в TextView выводить кол-во закачанных файлов. С помощью этого примера мы увидим, зачем может быть нужен Handler.

Создадим проект:

Project name: P0801_Handler
Build Target: Android 2.3.3
Application name: Handler
Package name: ru.startandroid.develop.p0801handler
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Handler</string>
    <string name="start">Start</string>
    <string name="test">Test</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <ProgressBar
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:indeterminate="true">
     </ProgressBar>
     <TextView
           android:id="@+id/tvInfo"
           android:layout_width="fill_parent"
           android:layout_height="wrap_content"
           android:text="">
     </TextView>
     <Button
           android:id="@+id/btnStart"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onclick"
           android:text="@string/start">
     </Button>
     <Button
           android:id="@+id/btnTest"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onclick"
           android:text="@string/test">
     </Button>
</LinearLayout>
```

ProgressBar у нас будет крутиться всегда. Позже станет понятно, зачем. **TextView** – для вывода информации о закачке файлов. Кнопка **Start** будет стартовать закачку. Кнопка **Test** будет просто выводить в лог слово test.

Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p0801handler;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

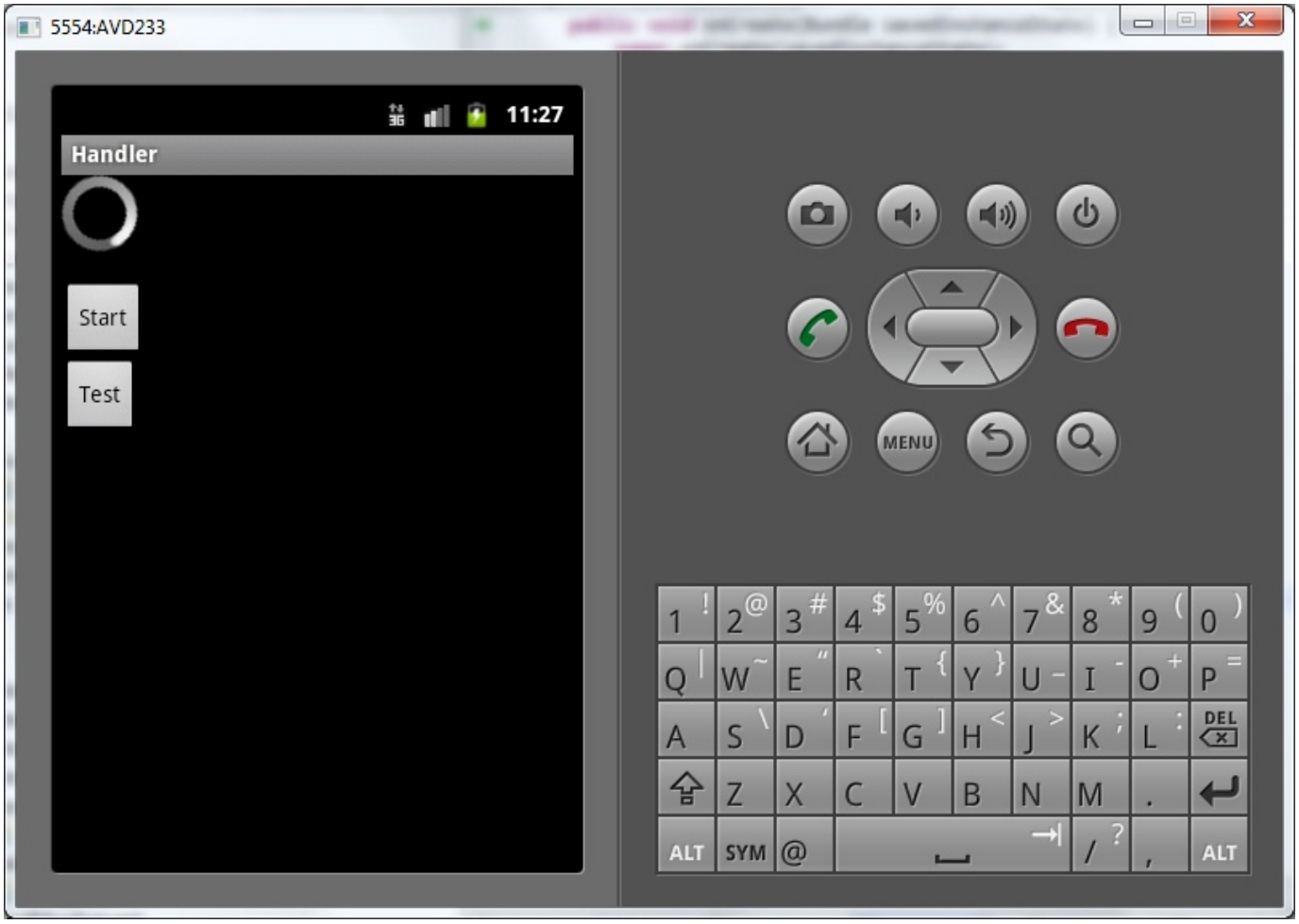
```
public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  Handler h;
  TextView tvInfo;
  Button btnStart;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tvInfo = (TextView) findViewById(R.id.tvInfo);
  }

  public void onclick(View v) {
    switch (v.getId()) {
    case R.id.btnStart:
      for (int i = 1; i <= 10; i++) {
        // долгий процесс
        downloadFile();
        // обновляем TextView
        tvInfo.setText("Закачано файлов: " + i);
        // пишем лог
        Log.d(LOG_TAG, "Закачано файлов: " + i);
      }
      break;
    case R.id.btnTest:
      Log.d(LOG_TAG, "test");
      break;
    default:
      break;
    }
  }

  void downloadFile() {
    // пауза - 1 секунда
    try {
      TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

В обработчике кнопки **Start** мы организуем цикл для закачки файлов. В каждой итерации цикла выполняем метод **downloadFile** (который эмулирует закачку файла), обновляем **TextView** и пишем в лог информацию о том, что кол-во закачанных файлов изменилось. Итого у нас должны закачаться 10 файлов и после закачки каждого из них лог и экран должны показывать, сколько файлов уже закачано.

По нажатию кнопки **Test** – просто выводим в лог сообщение.

downloadFile – эмулирует закачку файла, это просто пауза в одну секунду.

Все сохраним и запустим приложение.



Мы видим, что **ProgressBar** крутится. Понажимаем на кнопку **Test**, в логах появляется test. Все в порядке, приложение отзывается на наши действия.

Теперь расположите AVD на экране монитора так, чтобы он не перекрывал вкладку **логов** в Eclipse (LogCat). Нам надо будет видеть их одновременно.

Если мы нажмем кнопку **Start**, то мы должны наблюдать, как обновляется **TextView** и пишется **лог** после закачки очередного файла. Но на деле будет немного не так. Наше приложение просто «зависнет» и перестанет реагировать на нажатия. Остановится **ProgressBar**, не будет обновляться **TextView**, и не будет нажиматься кнопка **Test**. Т.е. **UI** (экран) для нас станет недоступным. И только по логам будет понятно, что приложение на самом деле работает и файлы закачиваются. Нажмите **Start** и убедитесь.

Экран «висит», а логи идут. Как только все 10 файлов будут закачаны, приложение оживет и снова станет реагировать на ваши нажатия.

А все почему? Потому что работа экрана обеспечивается **основным** потоком приложения. А мы **заняли** весь этот основной поток под свои нужды. В нашем случае, как будто под закачку файлов. И как только мы закончили закачивать файлы – поток **освободился**, и экран стал снова обновляться и реагировать на нажатия.

Для тех, кто имеет опыт кодинга на Java, я ничего нового не открыл. Для остальных же, надеюсь, у меня получилось доступно объяснить. Тут надо понять одну вещь - **основной поток приложения отвечает за экран**. Этот поток ни в коем случае **нельзя грузить** чем-то тяжелым – экран просто перестает обновляться и реагировать на нажатия. Если у вас есть долгоиграющие задачи – их надо вынести в **отдельный поток**. Попробуем это сделать.

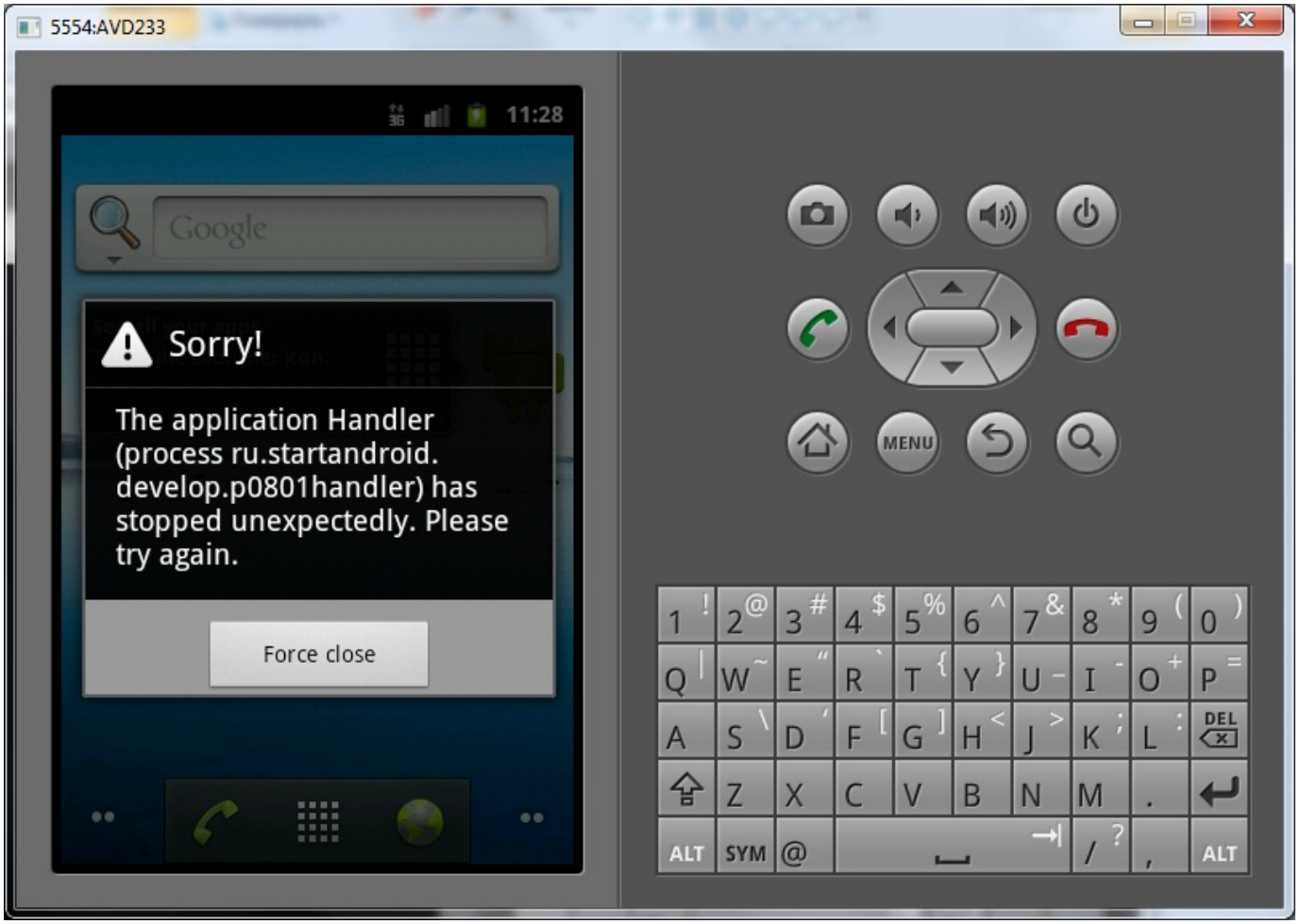
Перепишем **onclick**:

```
public void onclick(View v) {
  switch (v.getId()) {
  case R.id.btnStart:
    Thread t = new Thread(new Runnable() {
```

```
        public void run() {
          for (int i = 1; i <= 10; i++) {
            // долгий процесс
            downloadFile();
            // обновляем TextView
            tvInfo.setText("Закачано файлов: " + i);
            // пишем лог
            Log.d(LOG_TAG, "i = " + i);
          }
        }
      });
      t.start();
      break;
    case R.id.btnTest:
      Log.d(LOG_TAG, "test");
      break;
    default:
      break;
    }
  }
```

Т.е. мы просто помещаем весь цикл **в новый поток** и запускаем его. Теперь закачка файлов пойдет в этом **новом** потоке. А основной поток будет не занят и сможет без проблем прорисовывать экран и реагировать на нажатия. А значит, мы будем видеть изменение TextView после каждого закачанного файла и крутящийся ProgressBar. И, вообще, сможем полноценно взаимодействовать с приложением. Казалось бы, вот оно счастье :)

Все сохраним и запустим приложение. Жмем **Start**.



Приложение вылетело с ошибкой. Смотрим лог ошибок в LogCat. Там есть строки:

android.view.ViewRoot$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.

и

at ru.startandroid.develop.p0801handler.MainActivity$1.run(MainActivity.java:37)

Смотрим, что за код у нас в **MainActivity.java** в **37**-й строке:

```
tvInfo.setText("Закачано файлов: " + i);
```

При попытке выполнить этот код (не в основном потоке) мы получили ошибку «**Only the original thread that created a view hierarchy can touch its views**». Если по-русски, то «**Только оригинальный поток, создавший view-компоненты, может взаимодействовать с ними**». Т.е. работа с view-компонентами доступна только из **основного** потока. А новые потоки, которые мы создаем, не имеют доступа к элементам экрана.

Т.е. с одной стороны **нельзя загружать основной поток** тяжелыми задачами, чтобы не «вешался» экран. С другой стороны – **новые потоки**, созданные для выполнения тяжелых задач, **не имеют доступа к экрану**, и мы не сможем из них показать пользователю, что наша тяжелая задача как-то движется.

Тут нам поможет Handler. План такой:

- мы создаем в основном потоке Handler
- в потоке закачки файлов обращаемся к Handler и с его помощью помещаем в очередь сообщение для него же самого
- система берет это сообщение, видит, что адресат – Handler, и отправляет сообщение на обработку в Handler
- Handler, получив сообщение, обновит TextView

Чем это отличается от нашей предыдущей попытки обновить TextView из другого потока? Тем, что **Handler** был **создан** в **основном** потоке, и обрабатывать поступающие ему сообщения он будет в основном потоке, а значит, будет иметь доступ к экранным компонентам и сможет поменять текст в TextView. Получить доступ к Handler из какого-либо другого потока мы сможем без проблем, т.к. основной поток монополизирует только доступ к UI. А элементы классов (в нашем случае это Handler в MainActivity.java) доступны в любых потоках. Таким образом Handler выступит в качестве «**моста**» между потоками.

Перепишем метод **onCreate**:

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  tvInfo = (TextView) findViewById(R.id.tvInfo);
  btnStart = (Button) findViewById(R.id.btnStart);
  h = new Handler() {
    public void handleMessage(android.os.Message msg) {
      // обновляем TextView
      tvInfo.setText("Закачано файлов: " + msg.what);
      if (msg.what == 10) btnStart.setEnabled(true);
    };
  };
}
```

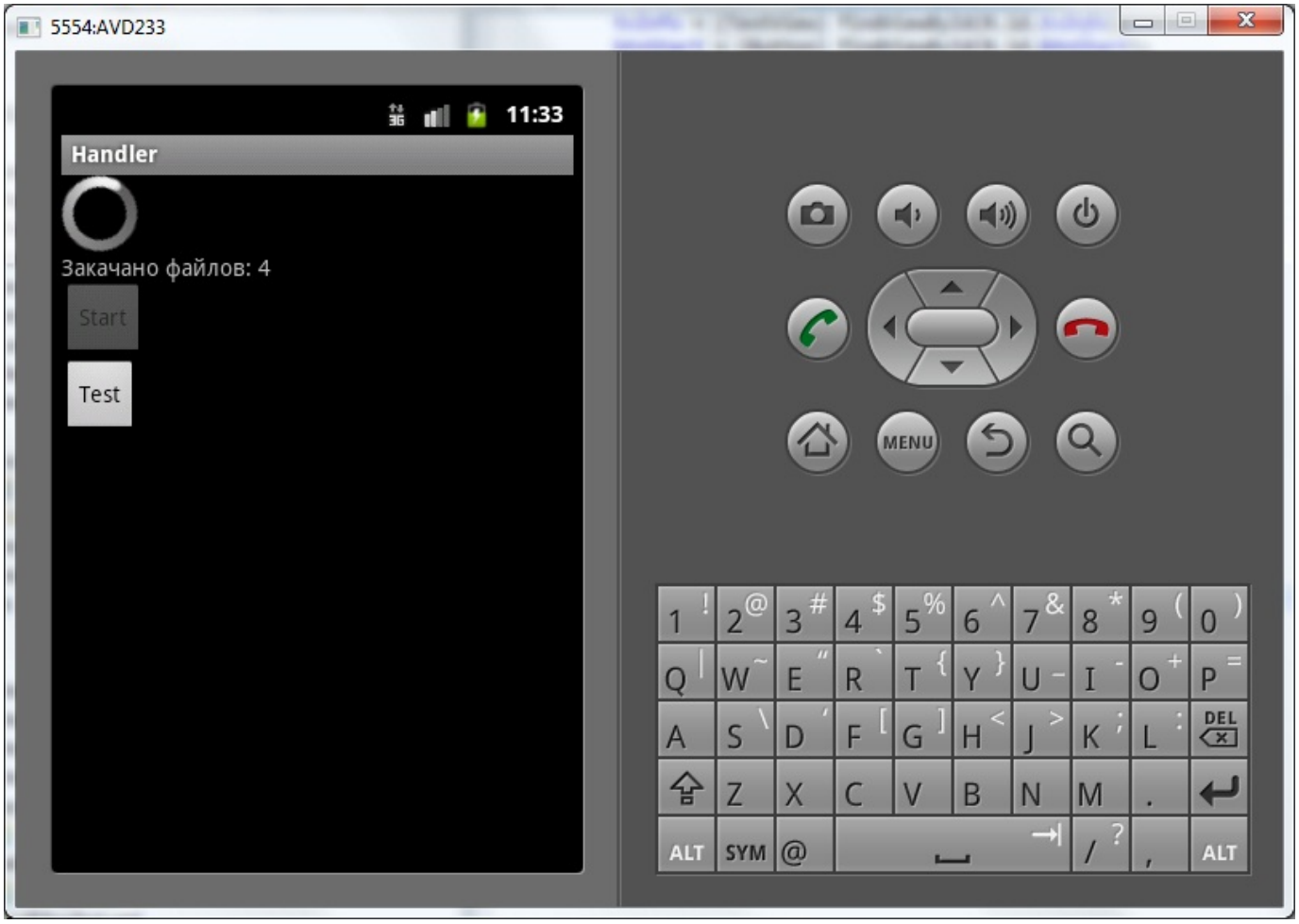
Здесь мы создаем Handler и в нем реализуем метод обработки сообщений handleMessage. Мы извлекаем из сообщения атрибут what – это кол-во закачанных файлов. Если оно равно 10, т.е. все файлы закачаны, мы активируем кнопку Start. (кол-во закачанных файлов мы сами кладем в сообщение - сейчас увидите, как)

Метод **onclick** перепишем так:

```
public void onclick(View v) {
  switch (v.getId()) {
  case R.id.btnStart:
    btnStart.setEnabled(false);
    Thread t = new Thread(new Runnable() {
      public void run() {
        for (int i = 1; i <= 10; i++) {
          // долгий процесс
          downloadFile();
          h.sendEmptyMessage(i);
          // пишем лог
          Log.d(LOG_TAG, "i = " + i);
        }
      }
    });
    t.start();
    break;
  case R.id.btnTest:
    Log.d(LOG_TAG, "test");
    break;
  default:
    break;
  }
}
```

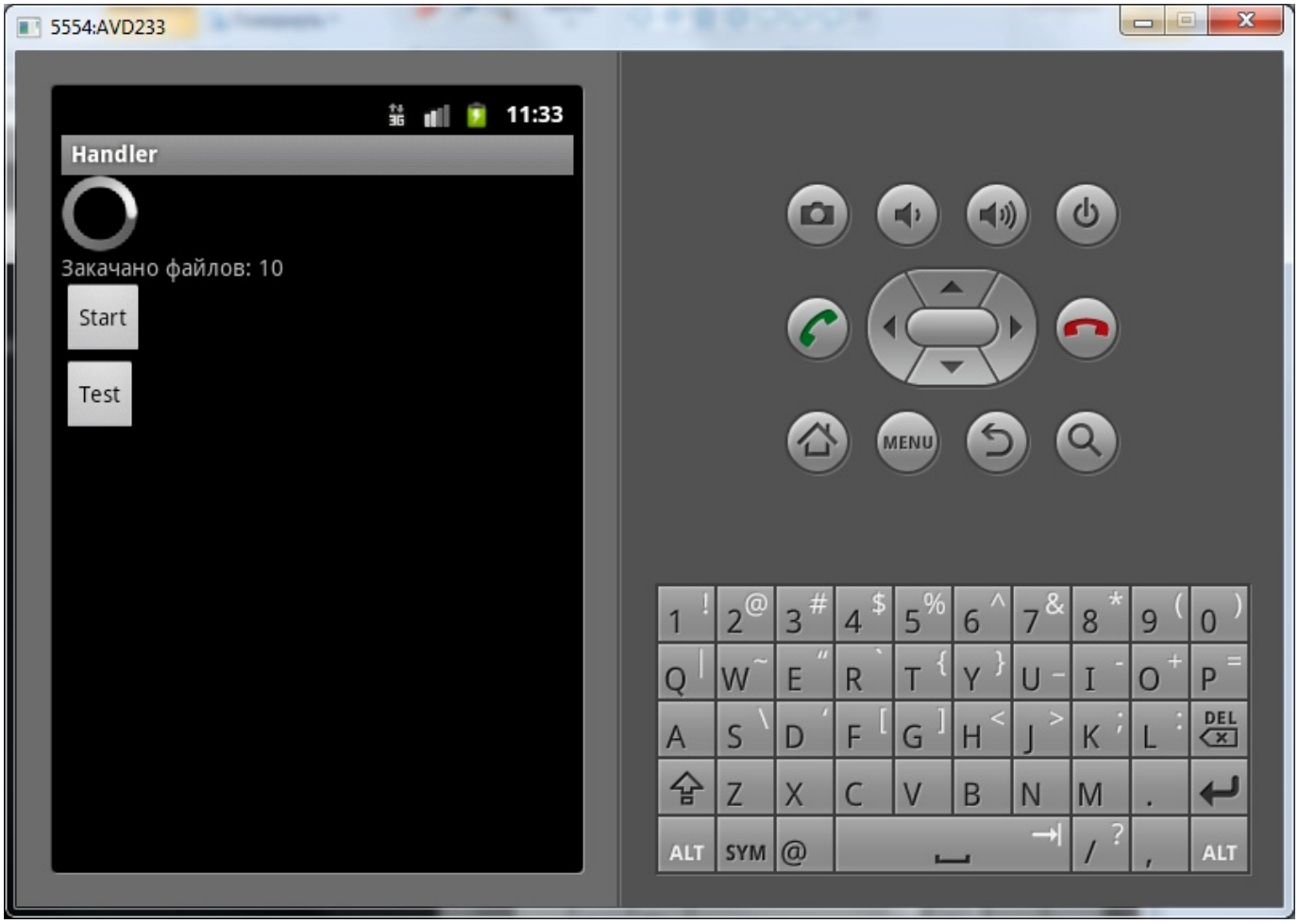
Мы деактивируем кнопку **Start** перед запуском закачки файлов. Это просто защита, чтобы нельзя было запустить несколько закачек одновременно. А в процессе закачки, после каждого закачанного файла, отправляем (sendEmptyMessage) для Handler сообщение с кол-вом уже закачанных файлов. Handler это сообщение примет, извлечет из него кол-во файлов и обновит TextView.

Все сохраняем и запускаем приложение. Жмем кнопку **Start**.



Кнопка **Start** стала неактивной, т.к. мы ее сами выключили. А TextView обновляется, ProgressBar крутится и кнопка Test нажимается. Т.е. и закачка файлов идет, и приложение продолжает работать без проблем, отображая статус закачки.

Когда все файлы закачаются, кнопка Start снова станет активной.



Подытожим все вышесказанное.

1) Сначала мы попытались грузить приложение тяжелой задачей в основном потоке. Это привело к тому, что мы потеряли экран – он перестал обновляться и отвечать на нажатия. Случилось это потому, что за экран отвечает основной поток приложения, а он был сильно загружен.

2) Мы создали отдельный поток и выполнили весь тяжелый код там. И это бы сработало, но нам надо было обновлять экран в процессе работы. А из не основного потока доступа к экрану нет. Экран доступен только из основного потока.

3) Мы создали Handler в основном потоке. А из нового потока отправляли для Handler сообщения, чтобы он нам обновлял экран. В итоге Handler помог нам обновлять экран не из основного потока.

Достаточно сложный урок получился. Наверняка, мало, что понятно. Не волнуйтесь, в этом уроке я просто показал, в какой ситуации Handler может быть полезен. А методы работы с ним мы рассмотрим подробно в следующих уроках.

На следующем уроке:

- посылаем простейшее сообщение для Handler

Урок 81. Handler. Посылаем простое сообщение

В этом уроке:

- посылаем простейшее сообщение для Handler

Надеюсь, вы прониклись предыдущим уроком и осознали, какая полезная штука Handler. Мы там отправляли ему сообщение. Сегодня сделаем это еще раз, но уже без кучи лишнего кода, зависающих экранов и ошибок приложения. Этакий чистый пример, чтобы закрепить.

Как мы помним, **Handler** позволяет **класть** в очередь сообщения и сам же умеет их **обрабатывать**. Фишка тут в том, что положить сообщение он может из **одного** потока, а прочесть из **другого**.

Сообщение может содержать в себе атрибуты. Сегодня рассмотрим самый простой вариант, атрибут **what**.

Напишем простое приложение-клиент. Оно, как-будто, будет подключаться к серверу, выполнять какую-то работу и отключаться. На экране мы будем наблюдать, как меняется статус подключения и как крутится ProgressBar при подключении.

При сменах состояния подключения мы будем отправлять сообщение для Handler. А в атрибут what будем класть текущий статус. Handler при обработке сообщения прочтет из него what и выполнит какие-либо действия.

Создадим проект:

Project name: P0811_ HandlerSimpleMessage
Build Target: Android 2.3.3
Application name: HandlerSimpleMessage
Package name: ru.startandroid.develop.p0811handlersimplemessage
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">HandlerSimpleMessage</string>
    <string  name="connect">Connect</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnConnect"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onclick"
            android:text="@string/connect">
        </Button>
        <TextView
            android:id="@+id/tvStatus"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="">
        </TextView>
        <ProgressBar
            android:id="@+id/pbConnect"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:indeterminate="true"
            android:visibility="gone">
        </ProgressBar>
    </LinearLayout>
```

Кнопка для старта подключения, TextView для вывода информации о статусе подключения и ProgressBar, работающий в процессе подключения.

MainActivity.java:

```
package ru.startandroid.develop.p0811handlersimplemessage;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final int STATUS_NONE = 0; // нет подключения
  final int STATUS_CONNECTING = 1; // подключаемся
  final int STATUS_CONNECTED = 2; // подключено

  Handler h;

  TextView tvStatus;
  ProgressBar pbConnect;
  Button btnConnect;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
```

```
    tvStatus = (TextView) findViewById(R.id.tvStatus);
    pbConnect = (ProgressBar) findViewById(R.id.pbConnect);
    btnConnect = (Button) findViewById(R.id.btnConnect);

    h = new Handler() {
      public void handleMessage(android.os.Message msg) {
        switch (msg.what) {
        case STATUS_NONE:
          btnConnect.setEnabled(true);
          tvStatus.setText("Not connected");
          break;
        case STATUS_CONNECTING:
          btnConnect.setEnabled(false);
          pbConnect.setVisibility(View.VISIBLE);
          tvStatus.setText("Connecting");
          break;
        case STATUS_CONNECTED:
          pbConnect.setVisibility(View.GONE);
          tvStatus.setText("Connected");
          break;
        }
      };
    };
    h.sendEmptyMessage(STATUS_NONE);
  }

  public void onclick(View v) {
    Thread t = new Thread(new Runnable() {
      public void run() {
        try {
          // устанавливаем подключение
          h.sendEmptyMessage(STATUS_CONNECTING);
          TimeUnit.SECONDS.sleep(2);

          // установлено
          h.sendEmptyMessage(STATUS_CONNECTED);

          // выполняется какая-то работа
          TimeUnit.SECONDS.sleep(3);

          // разрываем подключение
          h.sendEmptyMessage(STATUS_NONE);

        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    });
    t.start();
  }

}
```

STATUS_NONE, **STATUS_CONNECTING**, **STATUS_CONNECTED** – это константы статуса. Их будем передавать в сообщении, в атрибуте what. Разумеется, названия и значения этих констант произвольны и взяты из головы. Вы можете придумать и использовать свои.

В **onCreate** мы создаем Handler и реализуем его метод handleMessage. Этот метод отвечает за обработку сообщений,

которые предназначены для этого Handler. Соответственно на вход метода идет сообщение – Message. Мы читаем атрибут what и в зависимости от статуса подключения меняем экран:

STATUS_NONE – нет подключения. Кнопка подключения активна, TextView отражает статус подключения.

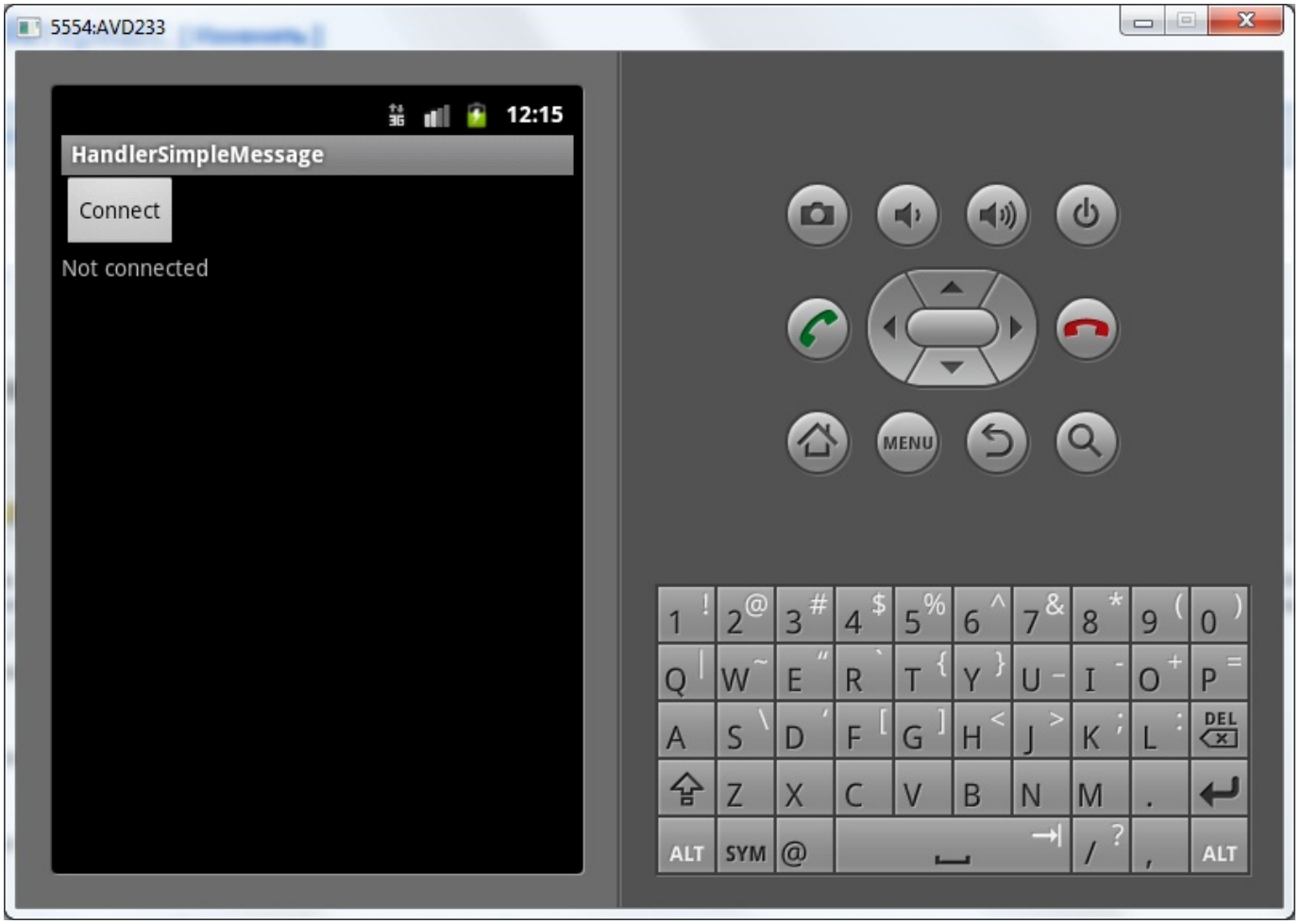
STATUS_CONNECTING – в процессе подключения. Кнопка подключения неактивна, показываем ProgressBar, TextView отражает статус подключения.

STATUS_CONNECTED – подключено. Скрываем ProgressBar, TextView отражает статус подключения.

В onCreate после создания Handler мы сразу отправляем ему сообщение со статусом STATUS_NONE. Для этого мы используем метод sendEmptyMessage. В этом методе создается сообщение, заполняется его атрибут what (значением, которое мы передаем в sendEmptyMessage), устанавливается Handler в качестве адресата и сообщение отправляется в очередь.

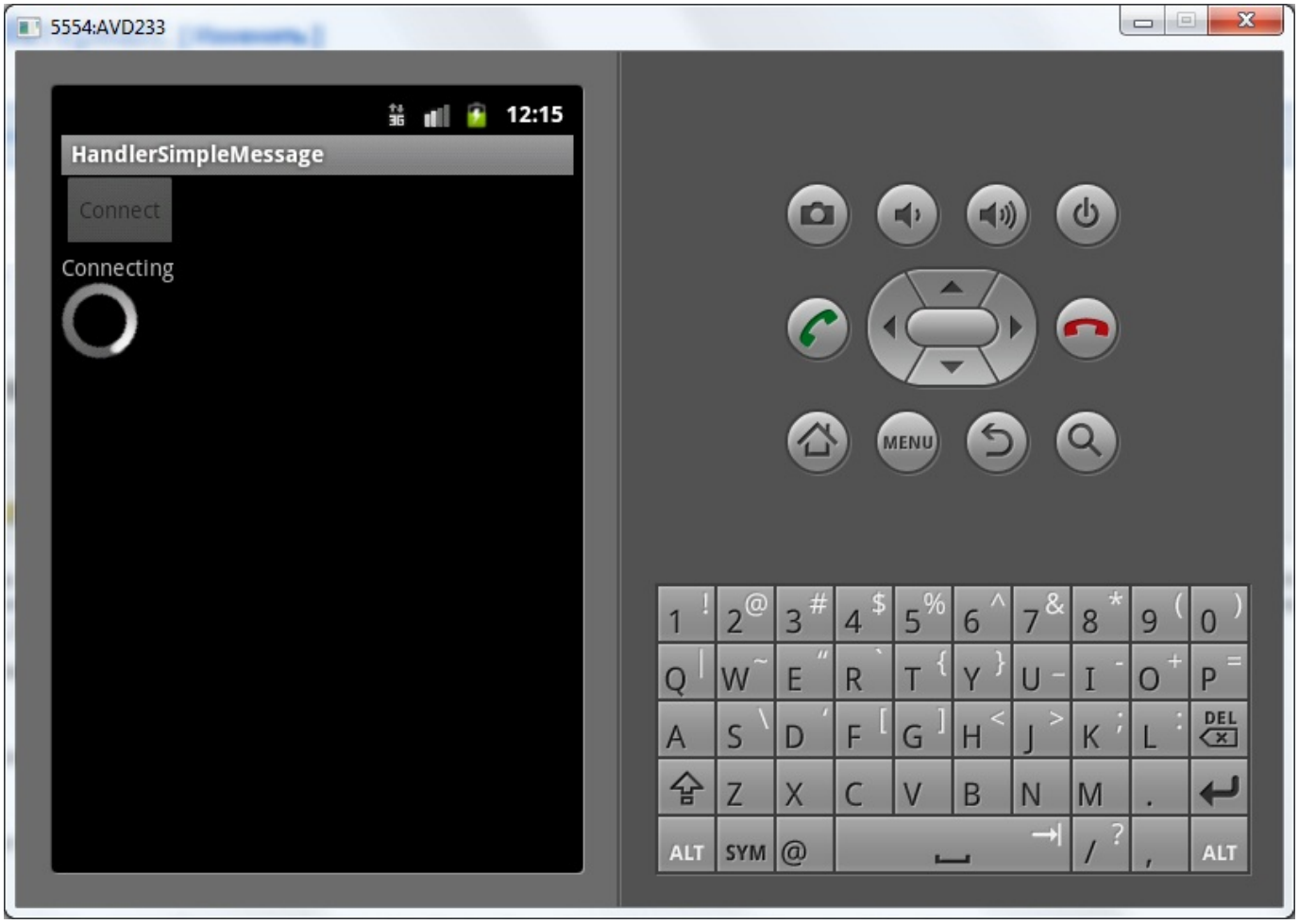
В методе **onclick** мы создаем и запускаем новый поток. В нем мы, с помощью **sleep**, эмулируем процесс подключения к серверу, выполнение работы и отключение. И, по мере выполнения действий, отправляем сообщения со статусами для Handler. Т.е. получается, что после нажатия на кнопку **Connect** статус меняется на STATUS_CONNECTING, две секунды идет подключение, статус меняется на STATUS_CONNECTED, 3 секунды выполняются действия и статус меняется на STATUS_NONE. Давайте проверим.

Все сохраним и запустим приложение.



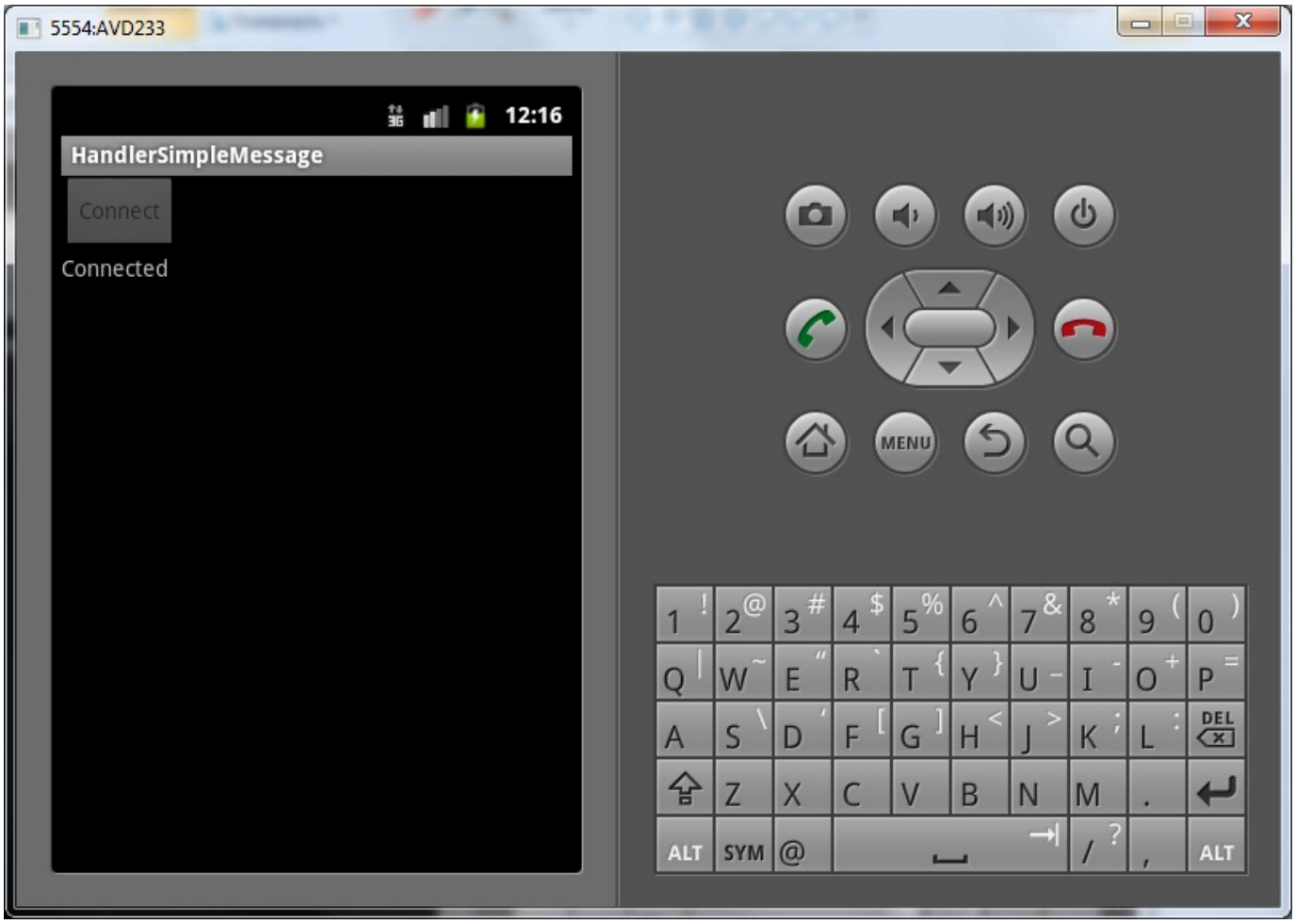
Жмем **Connect**.

Подключение пошло. Появляется ProgressBar, меняется текст и неактивна кнопка Connect.



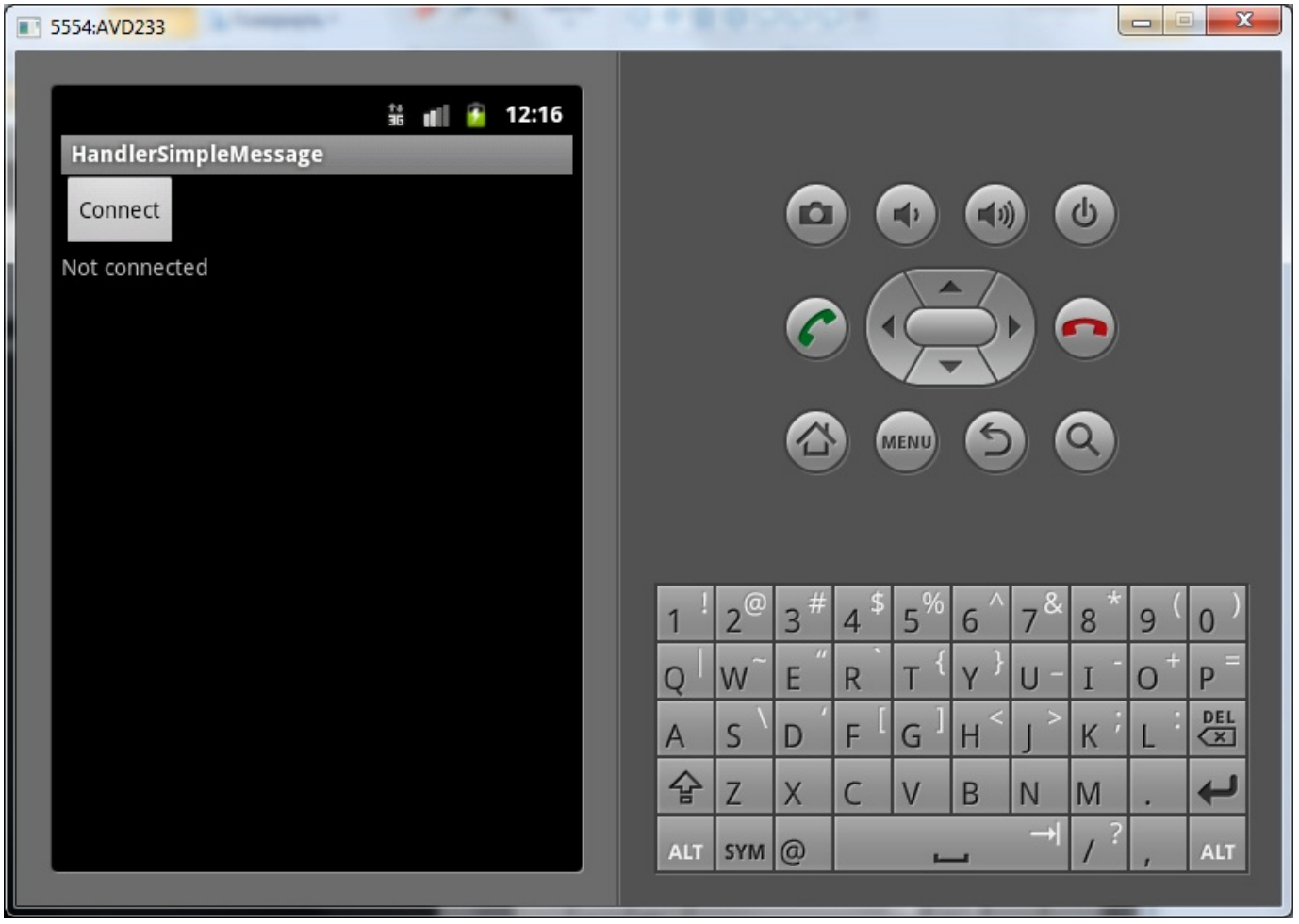
Проходит две секунды.

Подключение установлено. ProgressBar исчезает и меняется текст.



Проходит еще 3 секунды.

Подключение завершается. Кнопка Connect снова активна, а текст показывает, что мы не подключены.



Т.е. для простого обновления статуса из нового потока нам хватило атрибута what. Но кроме what сообщение может иметь еще несколько атрибутов. Рассмотрим их на следующем уроке.

На следующем уроке:

- создаем более содержательные сообщения для Handler

Урок 82. Handler. Пример с более содержательными сообщениями

В этом уроке:

- создаем более содержательные сообщения для Handler

В прошлых уроках мы использовали метод sendEmptyMessage. Этот метод сам создавал сообщение Message, заполнял его атрибут what и отправлял в очередь. Кроме **what** у сообщения есть еще атрибуты **arg1** и **arg2** типа int, и **obj** типа Object. В этом уроке мы сами будем создавать сообщение, заполнять атрибуты и отправлять.

Создадим приложение, которое будет подключаться к серверу, запрашивать кол-во файлов готовых для загрузки, эмулировать загрузку и отображать на экране ход действий, используя горизонтальный ProgressBar и TextView.

Создадим проект:

Project name: P0821_HandlerAdvMessage
Build Target: Android 2.3.3
Application name: HandlerAdvMessage
Package name: ru.startandroid.develop.p0821handleradvmessage
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">HandlerAdvMessage</string>
    <string  name="connect">Connect</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnConnect"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:onClick="onclick"
         android:text="@string/connect">
    </Button>
    <TextView
         android:id="@+id/tvStatus"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
            android:text="">
        </TextView>
        <ProgressBar
            android:id="@+id/pbDownload"
            style="?android:attr/progressBarStyleHorizontal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:visibility="gone">
        </ProgressBar>
    </LinearLayout>
```

MainActivity.java:

```
package ru.startandroid.develop.p0821handleradvmessage;

import java.util.Random;
import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final int STATUS_NONE = 0; // нет подключения
  final int STATUS_CONNECTING = 1; // подключаемся
  final int STATUS_CONNECTED = 2; // подключено
  final int STATUS_DOWNLOAD_START = 3; // загрузка началась
  final int STATUS_DOWNLOAD_FILE = 4; // файл загружен
  final int STATUS_DOWNLOAD_END = 5; // загрузка закончена
  final int STATUS_DOWNLOAD_NONE = 6; // загрузка закончена

  Handler h;

  TextView tvStatus;
  ProgressBar pbDownload;
  Button btnConnect;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvStatus = (TextView) findViewById(R.id.tvStatus);
    pbDownload = (ProgressBar) findViewById(R.id.pbDownload);
    btnConnect = (Button) findViewById(R.id.btnConnect);

    h = new Handler() {
```

```
    public void handleMessage(android.os.Message msg) {
      switch (msg.what) {
      case STATUS_NONE:
        btnConnect.setEnabled(true);
        tvStatus.setText("Not connected");
        pbDownload.setVisibility(View.GONE);
        break;
      case STATUS_CONNECTING:
        btnConnect.setEnabled(false);
        tvStatus.setText("Connecting");
        break;
      case STATUS_CONNECTED:
        tvStatus.setText("Connected");
        break;
      case STATUS_DOWNLOAD_START:
        tvStatus.setText("Start download " + msg.arg1 + " files");
        pbDownload.setMax(msg.arg1);
        pbDownload.setProgress(0);
        pbDownload.setVisibility(View.VISIBLE);
        break;
      case STATUS_DOWNLOAD_FILE:
        tvStatus.setText("Downloading. Left " + msg.arg2 + " files");
        pbDownload.setProgress(msg.arg1);
        saveFile((byte[]) msg.obj);
        break;
      case STATUS_DOWNLOAD_END:
        tvStatus.setText("Download complete!");
        break;
      case STATUS_DOWNLOAD_NONE:
        tvStatus.setText("No files for download");
        break;
      }
    };
  };
  h.sendEmptyMessage(STATUS_NONE);
}

public void onclick(View v) {

  Thread t = new Thread(new Runnable() {
    Message msg;
    byte[] file;
    Random rand = new Random();

    public void run() {
      try {
        // устанавливаем подключение
        h.sendEmptyMessage(STATUS_CONNECTING);
        TimeUnit.SECONDS.sleep(1);

        // подключение установлено
        h.sendEmptyMessage(STATUS_CONNECTED);

        // определяем кол-во файлов
        TimeUnit.SECONDS.sleep(1);
        int filesCount = rand.nextInt(5);

        if (filesCount == 0) {
          // сообщаем, что файлов для загрузки нет
          h.sendEmptyMessage(STATUS_DOWNLOAD_NONE);
```

```
            // и отключаемся
            TimeUnit.MILLISECONDS.sleep(1500);
            h.sendEmptyMessage(STATUS_NONE);
            return;
          }

          // загрузка начинается
          // создаем сообщение, с информацией о количестве файлов
          msg = h.obtainMessage(STATUS_DOWNLOAD_START, filesCount, 0);
          // отправляем
          h.sendMessage(msg);

          for (int i = 1; i <= filesCount; i++) {
            // загружается файл
            file = downloadFile();
            // создаем сообщение с информацией о порядковом номере
            // файла,
            // кол-вом оставшихся и самим файлом
            msg = h.obtainMessage(STATUS_DOWNLOAD_FILE, i,
                filesCount - i, file);
            // отправляем
            h.sendMessage(msg);
          }

          // загрузка завершена
          h.sendEmptyMessage(STATUS_DOWNLOAD_END);

          // отключаемся
          TimeUnit.MILLISECONDS.sleep(1500);
          h.sendEmptyMessage(STATUS_NONE);

        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    });
    t.start();
  }

  byte[] downloadFile() throws InterruptedException {
    TimeUnit.SECONDS.sleep(2);
    return new byte[1024];
  }

  void saveFile(byte[] file) {

  }

}
```

В **onCreate** мы создаем Handler и в его методе обработки (handleMessage) прописываем всю логику изменения экрана в зависимости от приходящих сообщений. Не буду подробно это расписывать, там все просто – меняем текст, включаем/выключаем кнопку, показываем/скрываем ProgressBar, меняем значение ProgressBar. Из интересного здесь стоит отметить, что читаем мы на этот раз не только **what**, но и остальные атрибуты сообщения – **arg1**, **arg2**, **obj**. А как они заполняются, увидим далее.

В **onclick** создаем новый поток для загрузки файлов. Устанавливаем подключение, получаем кол-во готовых для загрузки файлов. Если файлов для загрузки нет, посылаем соответствующее сообщение в Handler и отключаемся.

Если же файлы есть, мы создаем сообщение Message с помощью метода obtainMessage (int what, int arg1, int arg2). Он принимает на вход атрибуты what, arg1 и arg2. В **what** мы кладем статус, в **arg1** - кол-во файлов, **arg2** – не нужен, там просто ноль.

Далее начинаем загрузку. После загрузки каждого файла мы создаем сообщение Message с помощью метода obtainMessage (int what, int arg1, int arg2, Object obj), заполняем его атрибуты: **what** – статус, **arg1** – порядковый номер файла, **arg2** – кол-во оставшихся файлов, **obj** – файл. И отправляем.

По завершению загрузки отправляем соответствующее сообщение и отключаемся.

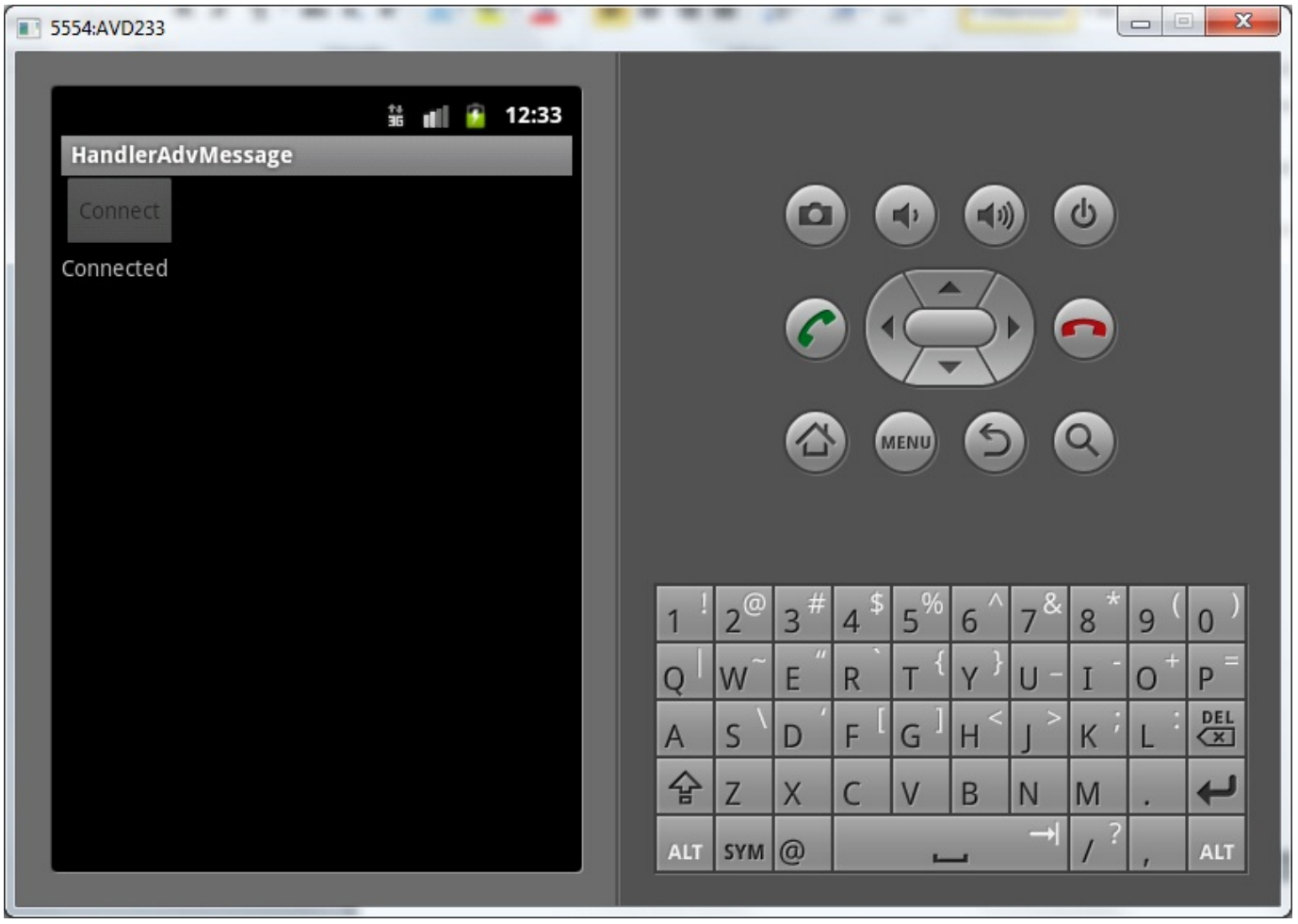
Метод **getFilesCount** – получение кол-ва файлов. Здесь просто генератор случайных чисел от нуля включительно до 5 не включительно.

downloadFile – эмулирует загрузку файла. ждет две секунды и возвращает массив из 1024 байтов.

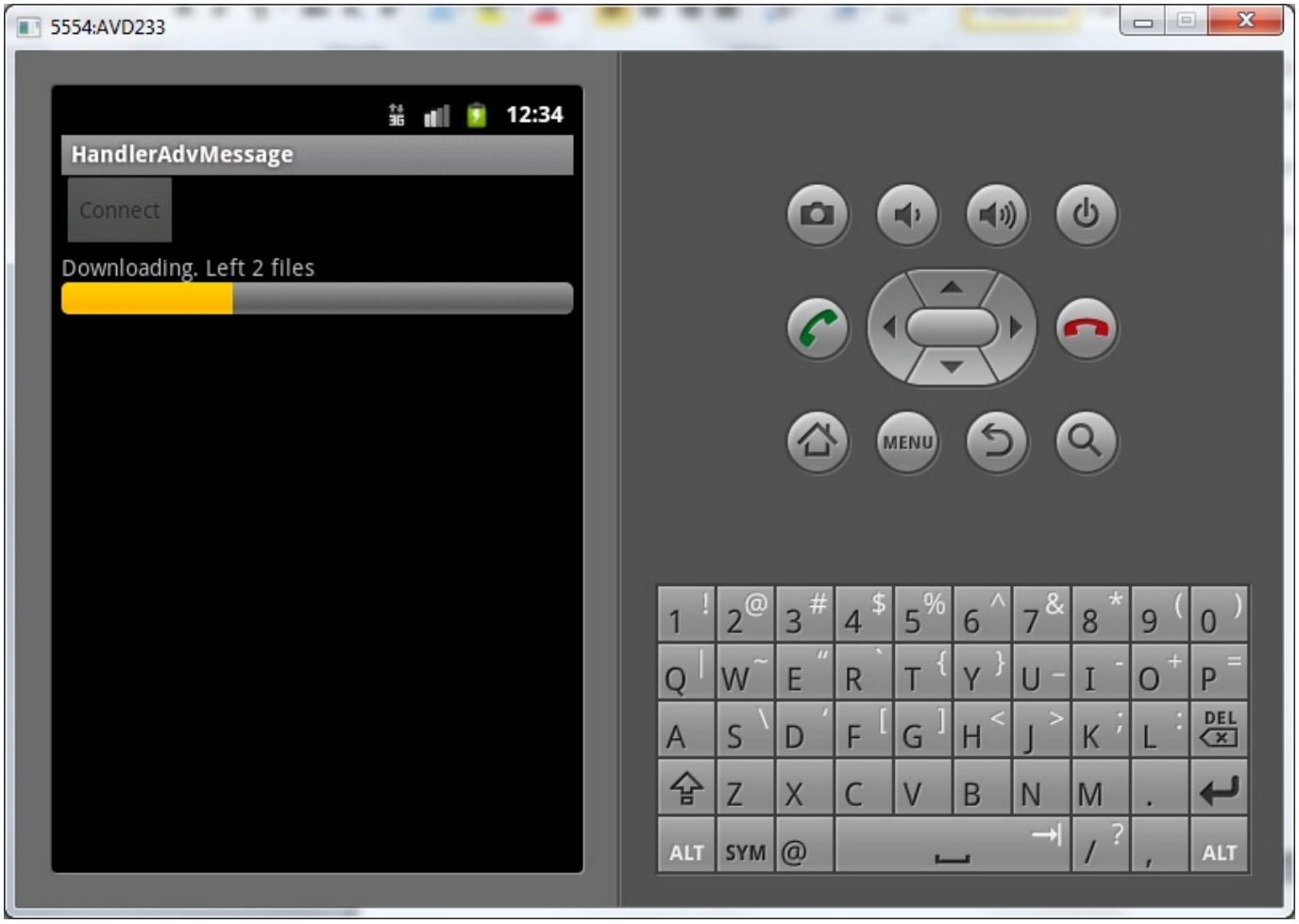
saveFile – метод сохранения файла на диск. Просто заглушка. Ничего не делает.

Все сохраняем и запускаем. Жмем **Connect**.

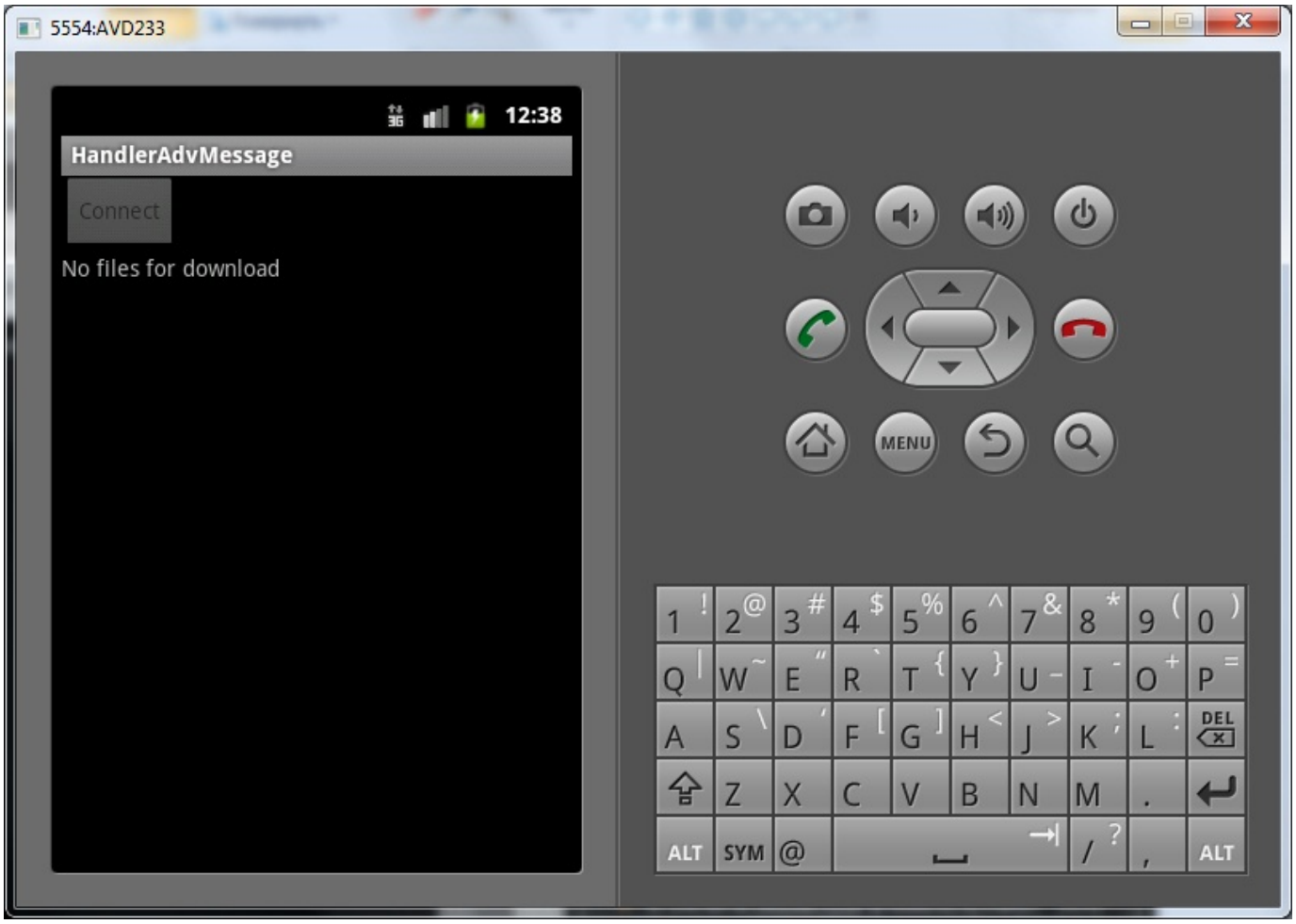
Устанавливается подключение



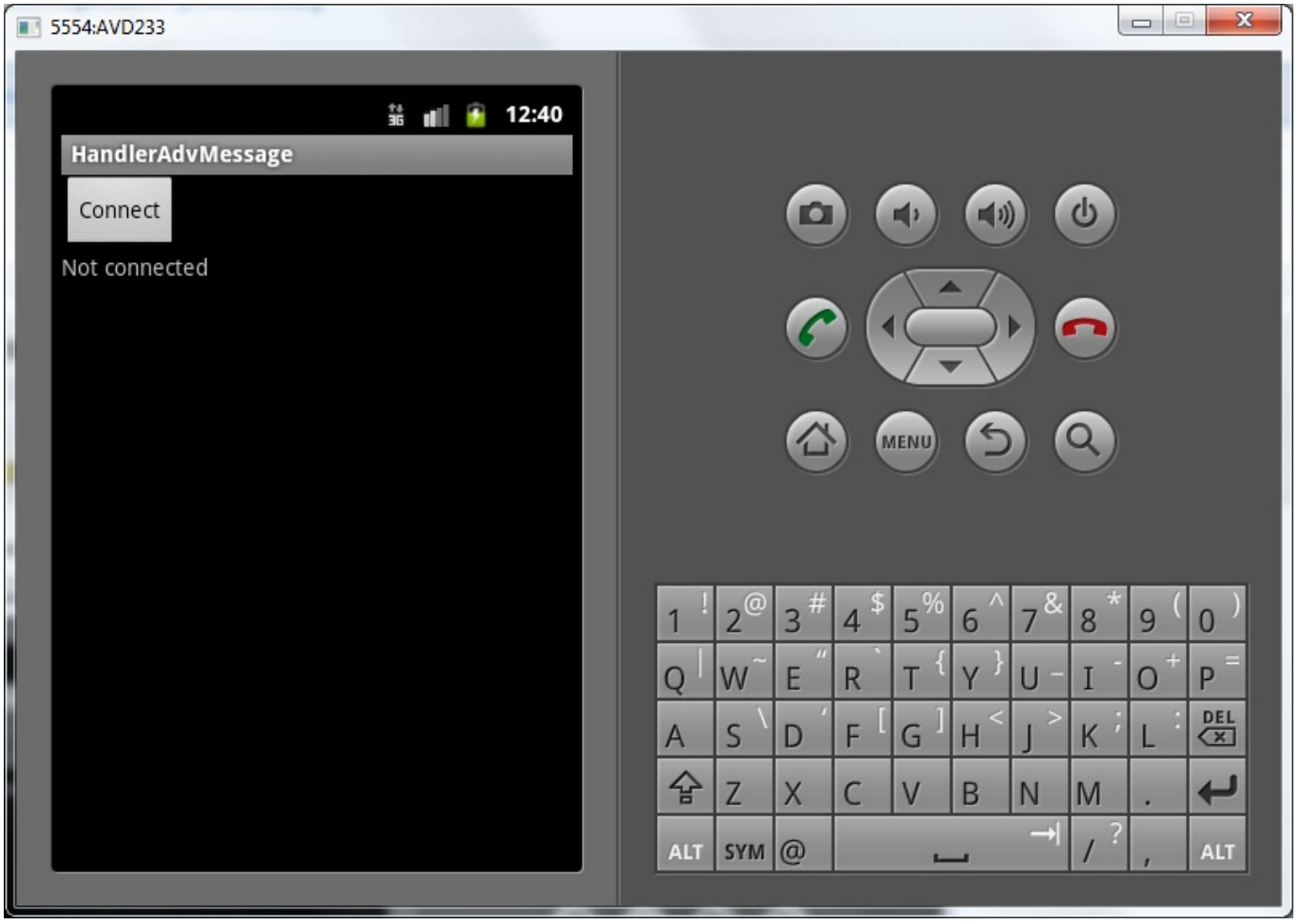
Далее, либо начинается загрузка



либо появляется сообщение, что файлов нет



Отключаемся



Используя разные атрибуты кроме what, мы смогли передать в основной поток и использовать там более разнообразные данные.

Мы создаем сообщения с помощью разных реализаций метода obtainMessage. А почему бы не создавать напрямую объект Message с помощью его конструкторов? В принципе можно, но официальный хелп рекомендует пользоваться методами obtainMessage, потому что это эффективней и быстрее. В этом случае сообщение достается из глобального пула сообщений, а не создается с нуля.

Здесь вы можете посмотреть все реализации метода obtainMessage для формирования сообщений и использовать тот, который подходит для ситуации. Они различаются разными комбинациями входных параметров.

На следующем уроке:

- посылаем отложенные сообщения
- удаляем сообщения из очереди
- используем Handler.Callback для обработки сообщений

Урок 83. Handler. Отложенные сообщения, удаление из очереди, Handler.Callback

В этом уроке:

- посылаем отложенные сообщения
- удаляем сообщения из очереди
- используем Handler.Callback для обработки сообщений

В прошлых уроках мы отправляли сообщения в очередь, а система сразу же доставала их и перенаправляла в Handler на обработку. Но мы можем настроить сообщение так, чтобы система отправило его на обработку не сразу, а с **задержкой**. Для этого используются методы sendEmptyMessageDelayed (если используете только what) и sendMessageDelayed (полное сообщение). В них мы можем указать паузу в миллисекундах. Система выждет эту паузу и только потом отправит сообщение в Handler.

Если вдруг поместили такое отложенное сообщение в очередь, а потом решили, что оно не должно уйти на обработку, то его можно из очереди удалить. Для этого используется метод removeMessages.

В прошлых уроках мы создавали свой Handler, и в его методе handleMessage кодили свой алгоритм обработки сообщений. Кроме этого способа Handler также может использовать для обработки сообщений объект, реализующий интерфейс Handler.Callback. У интерфейса всего один метод handleMessage – в нем и прописываем всю логику обработки сообщений. Я пока не встречал практической пользы от этой штуки, но все же разберемся, как ее можно использовать. Может когда и пригодится.

Создадим проект:

Project name: P0831_HandlerMessageManage
Build Target: Android 2.3.3
Application name: HandlerMessageManage
Package name: ru.startandroid.develop.p0831handlermessagemanage
Create Activity: MainActivity

strings.xml и **main.xml** не трогаем, они нам не нужны. Будем работать с логами.

Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p0831handlermessagemanage;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  Handler h;

  Handler.Callback hc = new Handler.Callback() {
    public boolean handleMessage(Message msg) {
```

```
      Log.d(LOG_TAG, "what = " + msg.what);
      return false;
    }
  };

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    h = new Handler(hc);
    sendMessages();
  }

  void sendMessages() {
    Log.d(LOG_TAG, "send messages");
    h.sendEmptyMessageDelayed(1, 1000);
    h.sendEmptyMessageDelayed(2, 2000);
    h.sendEmptyMessageDelayed(3, 3000);
  }
}
```

Мы создаем объект **hc** типа Handler.Callback. У него есть метод handleMessage, в котором мы будем обрабатывать сообщения. В нашем случае просто читаем атрибут **what** и выводим значение в лог.

В **onCreate** создаем handler, используя конструктор Handler (Handler.Callback callback). На вход передаем созданный ранее hc. И теперь Handler будет обрабатывать сообщения не сам, а перепоручит это объекту hc. Далее мы выполняем метод **sendMessages** , который кладет три сообщения в очередь сообщений. Для этого используется метод **sendEmptyMessageDelayed**. Это аналог знакомого нам метода sendEmptyMessage с прошлого урока. Он тоже заполняет в сообщении только атрибут **what**, но при этом он позволяет указать задержку в обработке сообщения. Т.е. сообщение будет извлечено из очереди и отправлено на обработку через указанное количество миллисекунд.

Итак, мы помещаем три сообщения:

1) what = 1, обработка через 1000 мс.
2) what = 2, обработка через 2000 мс.
3) what = 3, обработка через 3000 мс.

Замечу, что отсчет задержки начинается после помещения в очередь, а не после обработки предыдущего сообщения. Т.е. эти сообщения по отношению друг к другу сработают с интервалом в одну секунду.

Все сохраним и запустим приложение. В логе одна за другой будут появляться записи:

10:21:07.759: D/myLogs(332): send messages
10:21:08.786: D/myLogs(332): what = 1
10:21:09.765: D/myLogs(332): what = 2
10:21:10.776: D/myLogs(332): what = 3

Обратите внимание на время этих записей. Первое срабатывает через 1000 мс после помещения в очередь (***send messages***), второе - через две секунды, третье – через три.

Теперь попробуем удалить сообщение из очереди. Перепишем метод **sendMessages**:

```
  void sendMessages() {
    h.sendEmptyMessageDelayed(1, 1000);
    h.sendEmptyMessageDelayed(2, 2000);
    h.sendEmptyMessageDelayed(3, 3000);
```

```
    h.removeMessages(2);
  }
```

Используем метод removeMessages, в котором указываем значение атрибута **what**. Этот метод находит в очереди сообщение с атрибутом what, равным 2, и удаляет его из очереди.

Все сохраняем, запускаем приложение. Смотрим лог:

10:24:49.916: D/myLogs(434): send messages
10:24:50.927: D/myLogs(434): what = 1
10:24:52.948: D/myLogs(434): what = 3

Как видим, сообщение с what = 2 не сработало.

А если будет несколько сообщений с одинаковым значением what? Система удалит первое попавшееся или все?

Проверим. Перепишем **sendMessages**:

```
  void sendMessages() {
    Log.d(LOG_TAG, "send messages");
    h.sendEmptyMessageDelayed(1, 1000);
    h.sendEmptyMessageDelayed(2, 2000);
    h.sendEmptyMessageDelayed(3, 3000);
    h.sendEmptyMessageDelayed(2, 4000);
    h.sendEmptyMessageDelayed(5, 5000);
    h.sendEmptyMessageDelayed(2, 6000);
    h.sendEmptyMessageDelayed(7, 7000);
    h.removeMessages(2);
  }
```

Будем помещать в очередь кучу сообщений. Из них несколько с what = 2. Проверим, какие удалит система.

Запускаем приложение и смотрим лог:

10:29:23.297: D/myLogs(467): send messages
10:29:24.372: D/myLogs(467): what = 1
10:29:26.307: D/myLogs(467): what = 3
10:29:28.364: D/myLogs(467): what = 5
10:29:30.332: D/myLogs(467): what = 7

Все сообщения с what = 2 были удалены. Не забывайте это. А то захотите удалить одно последнее сообщение, а система найдет все подходящие, ожидающие обработки, и снесет их.

У метода removeMessages есть еще реализация с использованием obj. Тут все также, только система ищет для удаления из очереди сообщения с указанными атрибутами **what** и **obj**.

Если хотите запланировать полноценное сообщение, а не просто what, то используйте метод sendMessageDelayed – на вход даете сообщение и указываете задержку обработки.

Есть еще методы sendEmptyMessageAtTime и sendMessageAtTime. Они тоже позволяют указать задержку обработки. Но эта задержка будет отсчитана от времени последнего старта системы, а не от времени помещения в очередь. Если сообщение окажется просроченным на момент помещения в очередь, оно выполняется сразу.

На следующем уроке:

- работаем с Handler и Runnable

Урок 84. Handler. Обработка Runnable

В этом уроке:

- работаем с Handler и Runnable

Кроме обработки сообщений, мы можем попросить Handler выполнить кусок кода – **Runnable**. В прошлых уроках мы работали с сообщениями, которые содержали атрибуты. Мы их обрабатывали в Handler и в зависимости от значений атрибутов выполняли те или иные действия. Runnable же – это кусок кода, который мы пошлем вместо атрибутов сообщения, и он будет выполнен в потоке, с которым работает Handler. Нам уже ничего не надо обрабатывать.

Для отправки кода в работу используется метод post. Как и сообщения, Runnable может быть выполнен с задержкой (postDelayed), и может быть удален из очереди (removeCallbacks). Напишем приложение, которое продемонстрирует все эти возможности.

Создадим проект:

Project name: P0841_HandlerRunnable
Build Target: Android 2.3.3
Application name: HandlerRunnable
Package name: ru.startandroid.develop.p0841handlerrunnable
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">HandlerRunnable</string>
    <string  name="info">Подробно</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <ProgressBar
          android:id="@+id/pbCount"
          style="?android:attr/progressBarStyleHorizontal"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_marginTop="20dp">
    </ProgressBar>
    <CheckBox
          android:id="@+id/chbInfo"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/info">
        </CheckBox>
        <TextView
            android:id="@+id/tvInfo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text=""
            android:visibility="gone">
        </TextView>
    </LinearLayout>
```

ProgressBar, отображающий текущий прогресс. **CheckBox**, который будет включать отображение доп.информации в **TextView**.

MainActivity.java:

```
package ru.startandroid.develop.p0841handlerrunnable;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends Activity {

  ProgressBar pbCount;
  TextView tvInfo;
  CheckBox chbInfo;
  int cnt;

  final String LOG_TAG = "myLogs";
  final int max = 100;

  Handler h;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    h = new Handler();

    pbCount = (ProgressBar) findViewById(R.id.pbCount);
    pbCount.setMax(max);
    pbCount.setProgress(0);
```

```
    tvInfo = (TextView) findViewById(R.id.tvInfo);

    chbInfo = (CheckBox) findViewById(R.id.chbInfo);
    chbInfo.setOnCheckedChangeListener(new OnCheckedChangeListener() {
      public void onCheckedChanged(CompoundButton buttonView,
          boolean isChecked) {
        if (isChecked) {
          tvInfo.setVisibility(View.VISIBLE);
          // показываем информацию
          h.post(showInfo);
        } else {
          tvInfo.setVisibility(View.GONE);
          // отменяем показ информации
          h.removeCallbacks(showInfo);
        }
      }
    });

    Thread t = new Thread(new Runnable() {
      public void run() {
        try {
          for (cnt = 1; cnt < max; cnt++) {
            TimeUnit.MILLISECONDS.sleep(100);
            // обновляем ProgressBar
            h.post(updateProgress);
          }
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    });
    t.start();

  }

  // обновление ProgressBar
  Runnable updateProgress = new Runnable() {
    public void run() {
      pbCount.setProgress(cnt);
    }
  };

  // показ информации
  Runnable showInfo = new Runnable() {
    public void run() {
      Log.d(LOG_TAG, "showInfo");
      tvInfo.setText("Count = " + cnt);
      // планирует сам себя через 1000 мсек
      h.postDelayed(showInfo, 1000);
    }
  };
}
```

В **onCreate** мы прописываем обработчик для CheckBox. При включении галки отображается TextView и в работу отправляется задание **showInfo**. При выключении галки – задание **showInfo** удаляется из очереди.

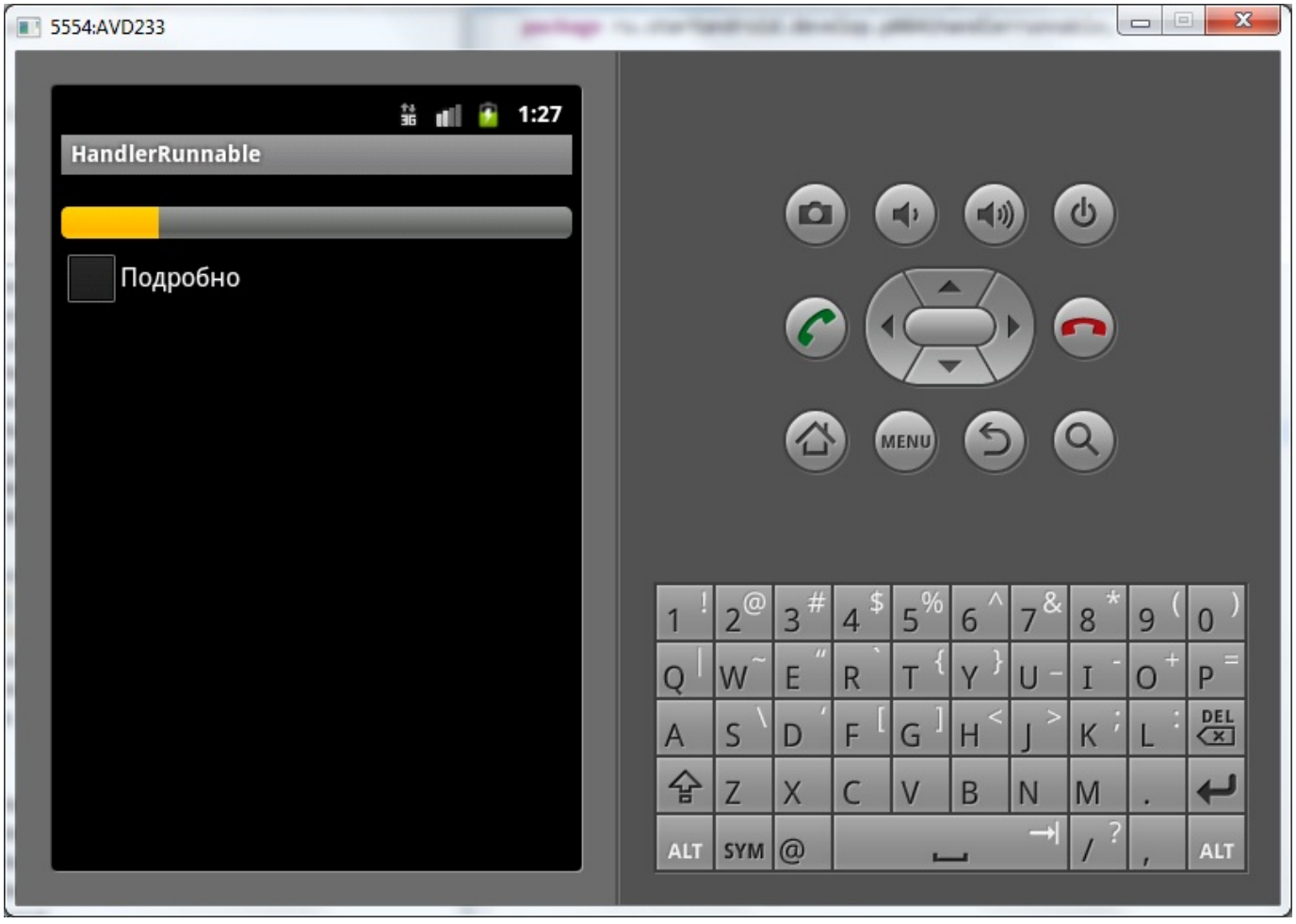
Далее в новом потоке эмулируем какое-либо действие - запускаем счетчик с паузами. В каждой итерации цикла отправляем в работу задание **updateProgress**, которое обновляет ProgressBar.

updateProgress – код, который обновляет значение ProgressBar.

showInfo – код, который обновляет TextView и сам себя планирует на выполнение через 1000 мсек. Т.е мы включаем CheckBox, showInfo срабатывает первый раз и само себя планирует на следующий раз. Т.е. этот код лежит в очереди сообщений, обрабатывается и снова кладет себя туда. Так продолжается, пока мы явно его не удалим из очереди (removeCallbacks), выключив CheckBox.

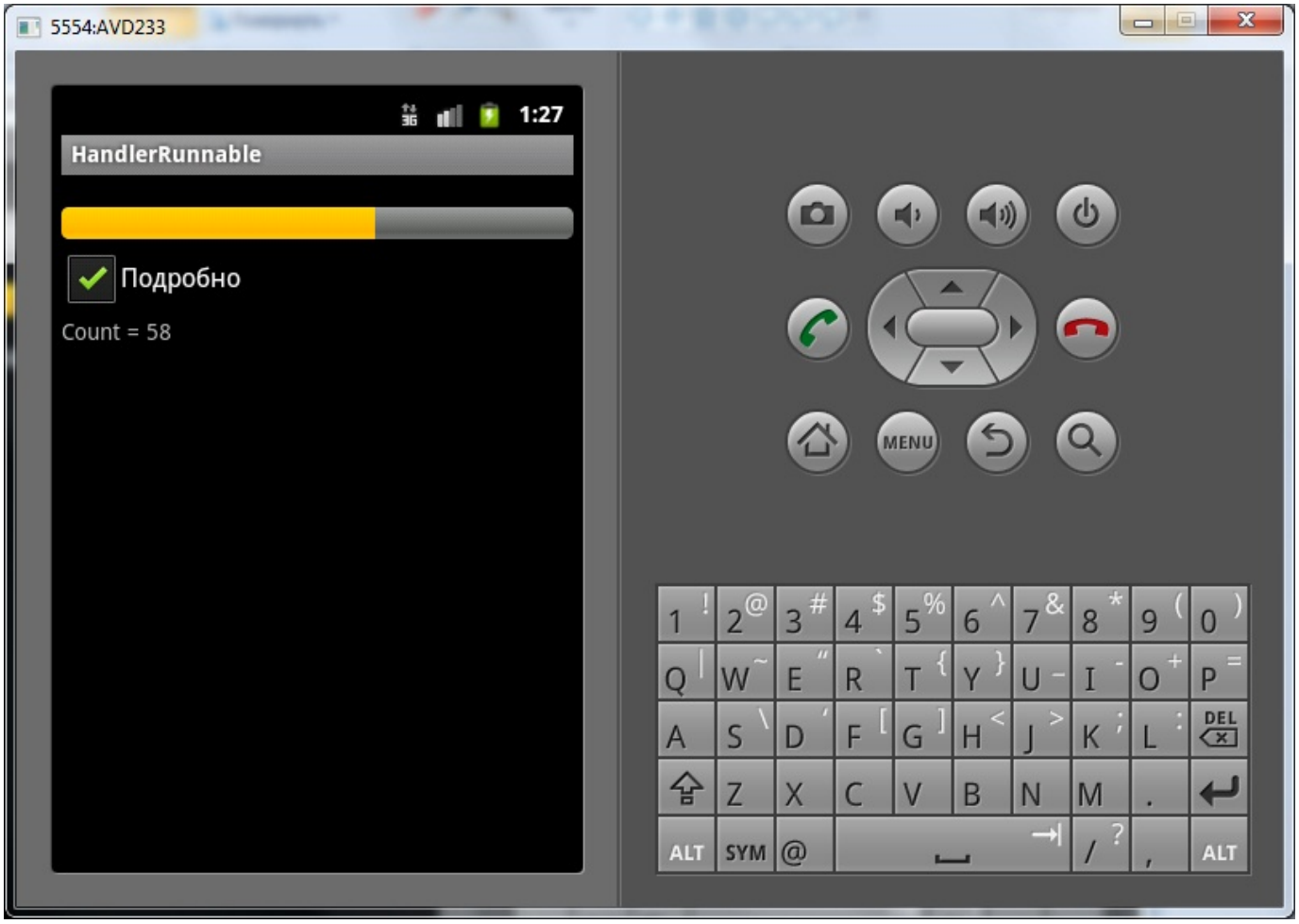
Будем выводить что-нибудь в лог из **showInfo**, чтобы увидеть, когда он работает, а когда нет.

Все сохраним и запустим приложение. Побежал ProgressBar.



Включим CheckBox.

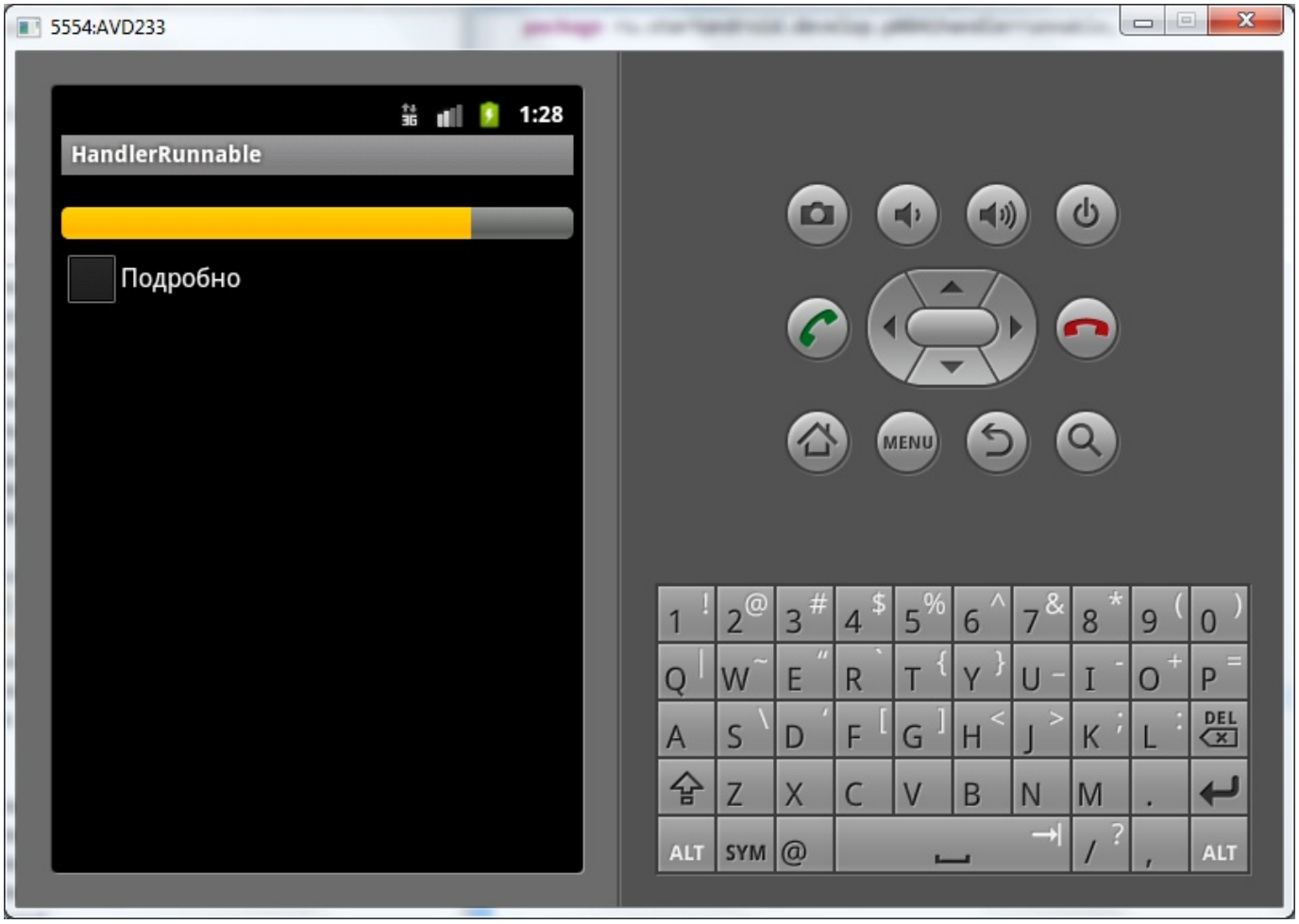
Появился TextView, который отображает текущее значение счетчика.



В логи при этом добавляется раз в секунду запись:

showInfo

Выключим CheckBox. Текст исчез.



И логи перестали идти. Значит, задание **showInfo** успешно удалилось из очереди и больше не работает.

Если снова включим CheckBox – оно снова начнет срабатывать и само себя помещать в очередь с задержкой исполнения. Выключаем CheckBox – удаляем его из очереди.

На следующем уроке:

- рассмотрим еще пару способов запуска Runnbale в UI-потоке

Урок 85. Еще несколько способов выполнения кода в UI-потоке

В этом уроке:

- рассмотрим еще пару способов запуска Runnbale в UI-потоке

Мы подробно рассмотрели Handler и увидели, что он умеет. Главное его достоинство – это умение выполнять код в UI-потоке. Существует еще пара способов выполнять Runnable в UI-потоке. Это методы:

Activity.runOnUiThread(Runnable)
View.post(Runnable)
View.postDelayed(Runnable, long)

Первые два похожи и отправляют Runnable на немедленную обработку. Я не знаю в чем их принципиальное отличие. Если у вас есть соображения на этот счет, пишите на форуме в ветке этого урока. А третий метод позволяет указать задержку выполнения Runnable.

Создадим приложение и опробуем эти методы.

Создадим проект:

Project name: P0851_RunnableUIThread
Build Target: Android 2.3.3
Application name: RunnableUIThread
Package name: ru.startandroid.develop.p0851runnableuithread
Create Activity: MainActivity

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/llMain"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:id="@+id/tvInfo"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="">
     </TextView>
</LinearLayout>
```

TextView, которое будем обновлять из нового потока.

MainActivity.java:

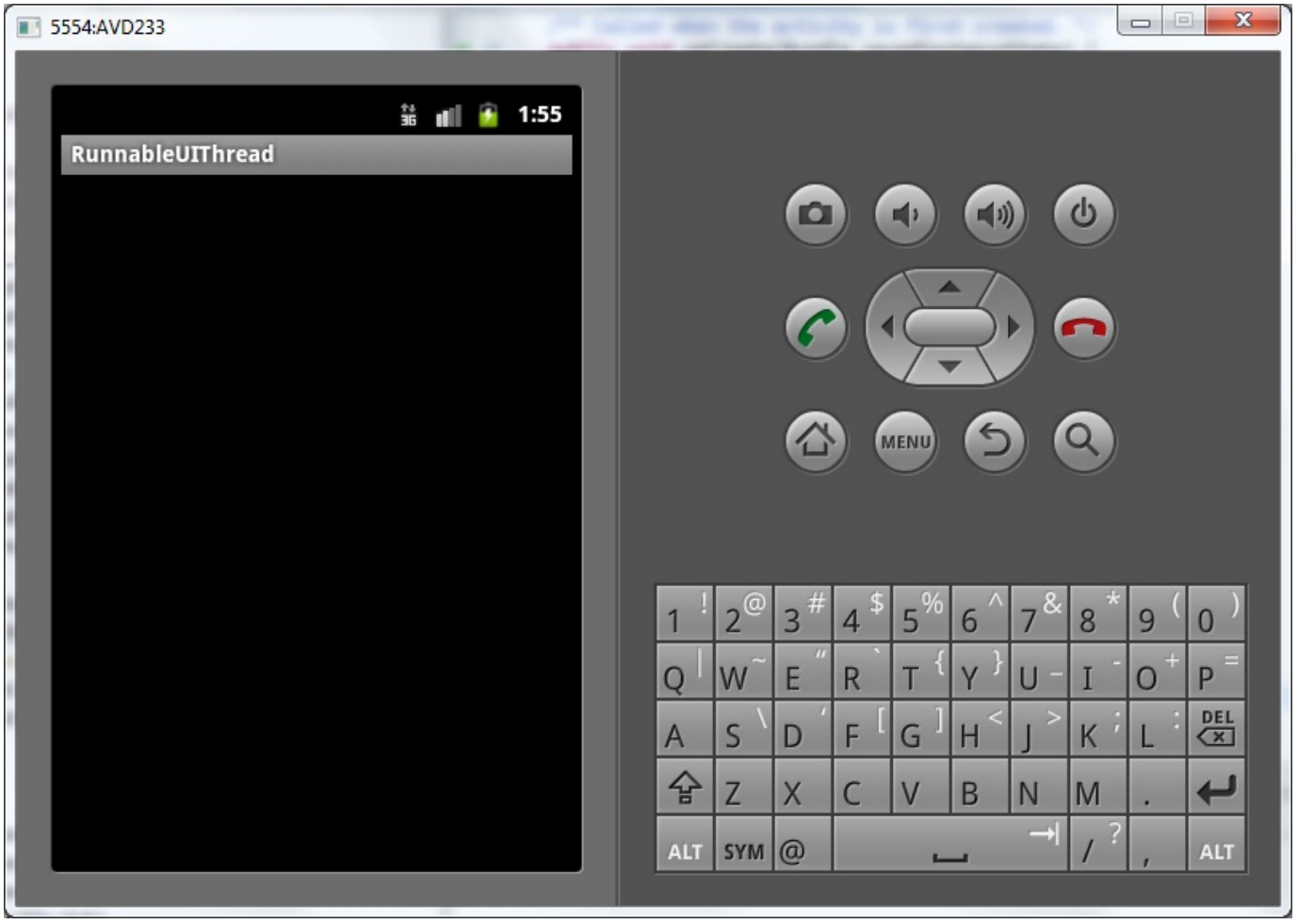
```
package ru.startandroid.develop.p0851runnableuithread;
```

```
import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  TextView tvInfo;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvInfo = (TextView) findViewById(R.id.tvInfo);

    Thread t = new Thread(new Runnable() {
      public void run() {
        try {
          TimeUnit.SECONDS.sleep(2);
          runOnUiThread(runn1);
          TimeUnit.SECONDS.sleep(1);
          tvInfo.postDelayed(runn3, 2000);
          tvInfo.post(runn2);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    });
    t.start();
  }

  Runnable runn1 = new Runnable() {
    public void run() {
      tvInfo.setText("runn1");
    }
  };

  Runnable runn2 = new Runnable() {
    public void run() {
      tvInfo.setText("runn2");
    }
  };

  Runnable runn3 = new Runnable() {
    public void run() {
      tvInfo.setText("runn3");
    }
  };
}
```

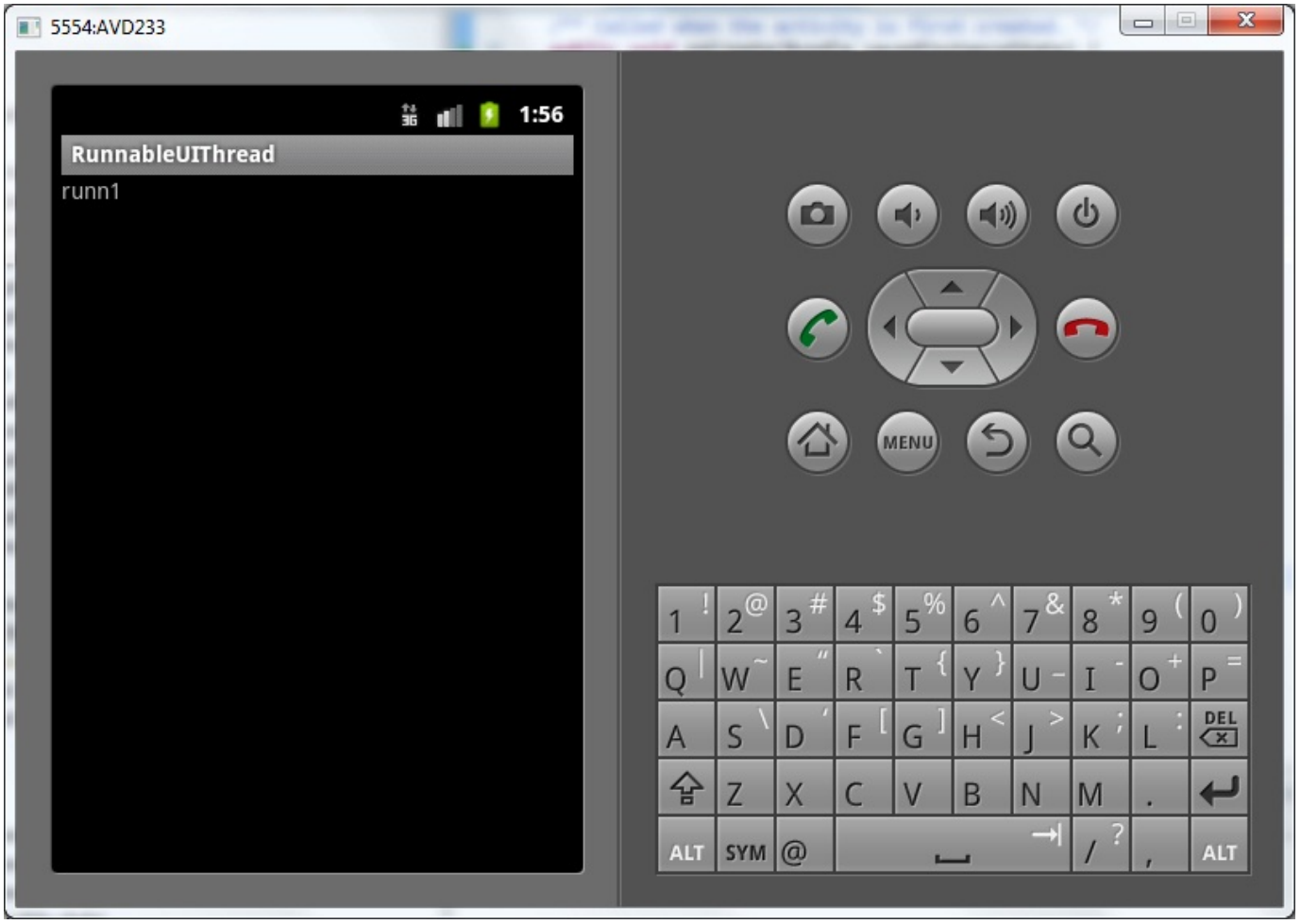
В **onCreate** создаем новый поток. В нем мы через паузы выполняем **runn1** и **runn2**, и планируем **runn3** с задержкой в 2000 мсек, используя выше упомянутые методы.

runn1, **runn2** и **runn3** – это просто **Runnable**, которые обновляют текст в **TextView**. Они должны быть выполнены в **UI**-потоке.

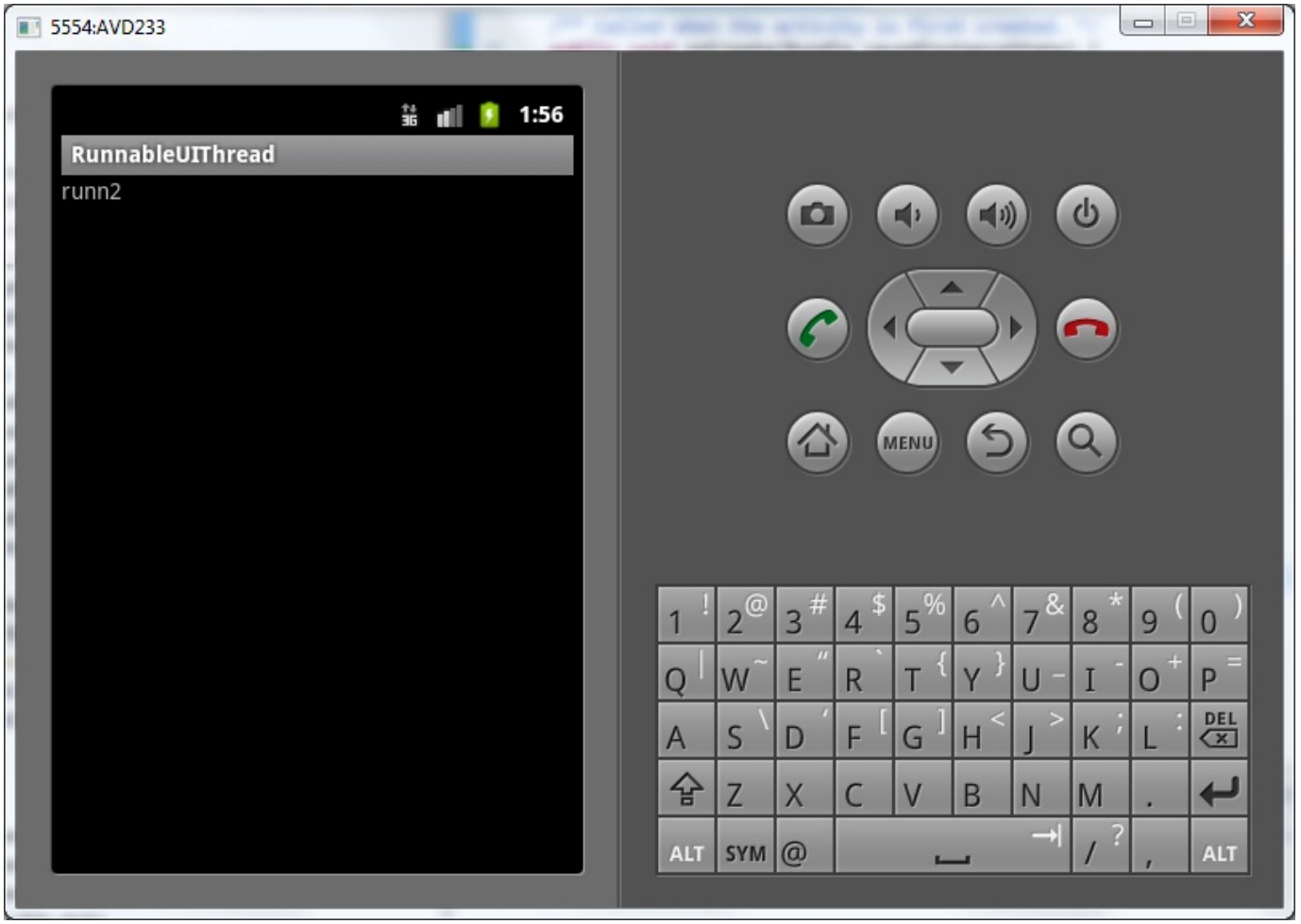
Все сохраним и запустим. Экран пустой.



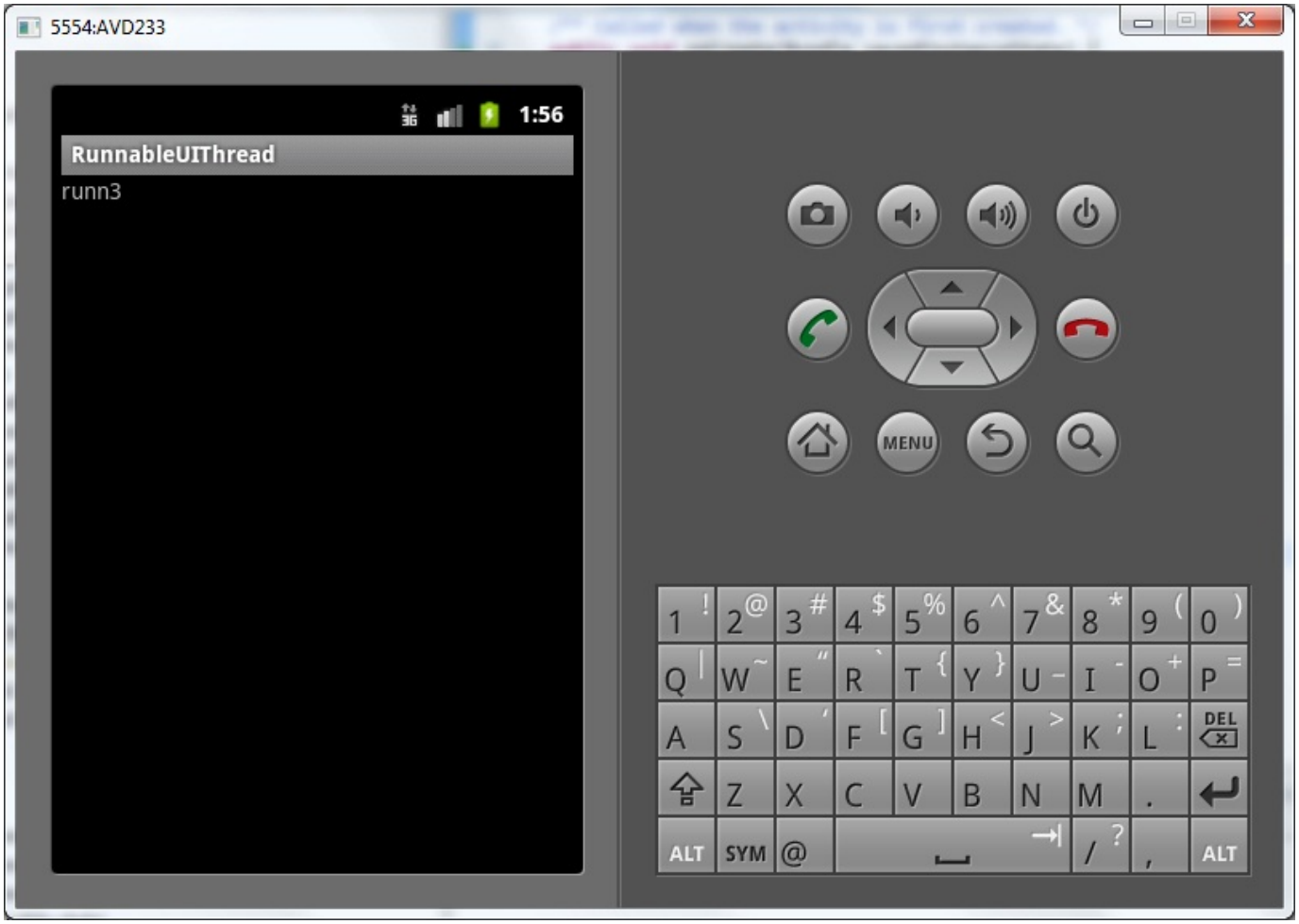
Через 2 сек выполняется **runn1**



Еще через секунду **runn2**



И еще через 2 секунды срабатывает **runn3**, который был отложен на 2 сек.



Тем самым, если ваши алгоритмы не особо сложны, можно использовать эти методы для выполнения кода в UI-потоке. Если же нужны навороты и алгоритм достаточно сложен, то используем Handler.

На следующем уроке:

- создаем несложный пример с AsyncTask

Урок 86. AsyncTask. Знакомство, несложный пример

В этом уроке:

- создаем несложный пример с AsyncTask

На прошлых уроках мы рассматривали, как в приложении можно выполнять тяжелые задачи. Мы вводили их в отдельный **поток** и использовали **Handler** для обратной связи и обновления экрана. Создатели Android решили, что эти механизмы стоит выделить в отдельный класс – **AsyncTask**. Т.е. его цель – это выполнение тяжелых задач и передача в UI-поток результатов работы. Но при этом нам не надо задумываться о создании Handler и нового потока.

Чтобы работать с AsyncTask, надо создать класс-наследник и в нем прописать свою реализацию необходимых нам методов. На этом уроке рассмотрим три метода:

doInBackground – будет выполнен в новом потоке, здесь решаем все свои тяжелые задачи. Т.к. поток не основной - не имеет доступа к UI.

onPreExecute – выполняется перед doInBackground, имеет доступ к UI

onPostExecute – выполняется после doInBackground (не срабатывает в случае, если AsyncTask был отменен - об этом в следующих уроках), имеет доступ к UI

Сделаем простое приложение, в котором используем вышеуказанные методы. Выведем на экран слово Begin в методе onPreExecute, эмулируем тяжелый код в методе doInBackground, выведем на экран слово End в методе onPostExecute.

Создадим проект:

Project name: P0861_AsyncTask
Build Target: Android 2.3.3
Application name: AsyncTask
Package name: ru.startandroid.develop.p0861asynctask
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AsyncTask</string>
    <string  name="start">Start</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onclick"
            android:text="@string/start">
        </Button>
        <ProgressBar
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
        </ProgressBar>
        <TextView
            android:id="@+id/tvInfo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="">
        </TextView>
    </LinearLayout>
```

По нажатию кнопки будем стартовать задачу, в TextView выводить информацию. ProgressBar покажет, что приложение не "висит" во время выполнения.

MainActivity.java:

```
package ru.startandroid.develop.p0861asynctask;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

  MyTask mt;
  TextView tvInfo;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvInfo = (TextView) findViewById(R.id.tvInfo);
  }

  public void onclick(View v) {
    mt = new MyTask();
    mt.execute();
  }

  class MyTask extends AsyncTask<Void, Void, Void> {

    @Override
    protected void onPreExecute() {
      super.onPreExecute();
      tvInfo.setText("Begin");
```

```
    }

    @Override
    protected Void doInBackground(Void... params) {
      try {
        TimeUnit.SECONDS.sleep(2);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      return null;
    }

    @Override
    protected void onPostExecute(Void result) {
      super.onPostExecute(result);
      tvInfo.setText("End");
    }
  }
}
```

В методе **onclick** мы создаем объект MyTask и запускаем его методом execute.

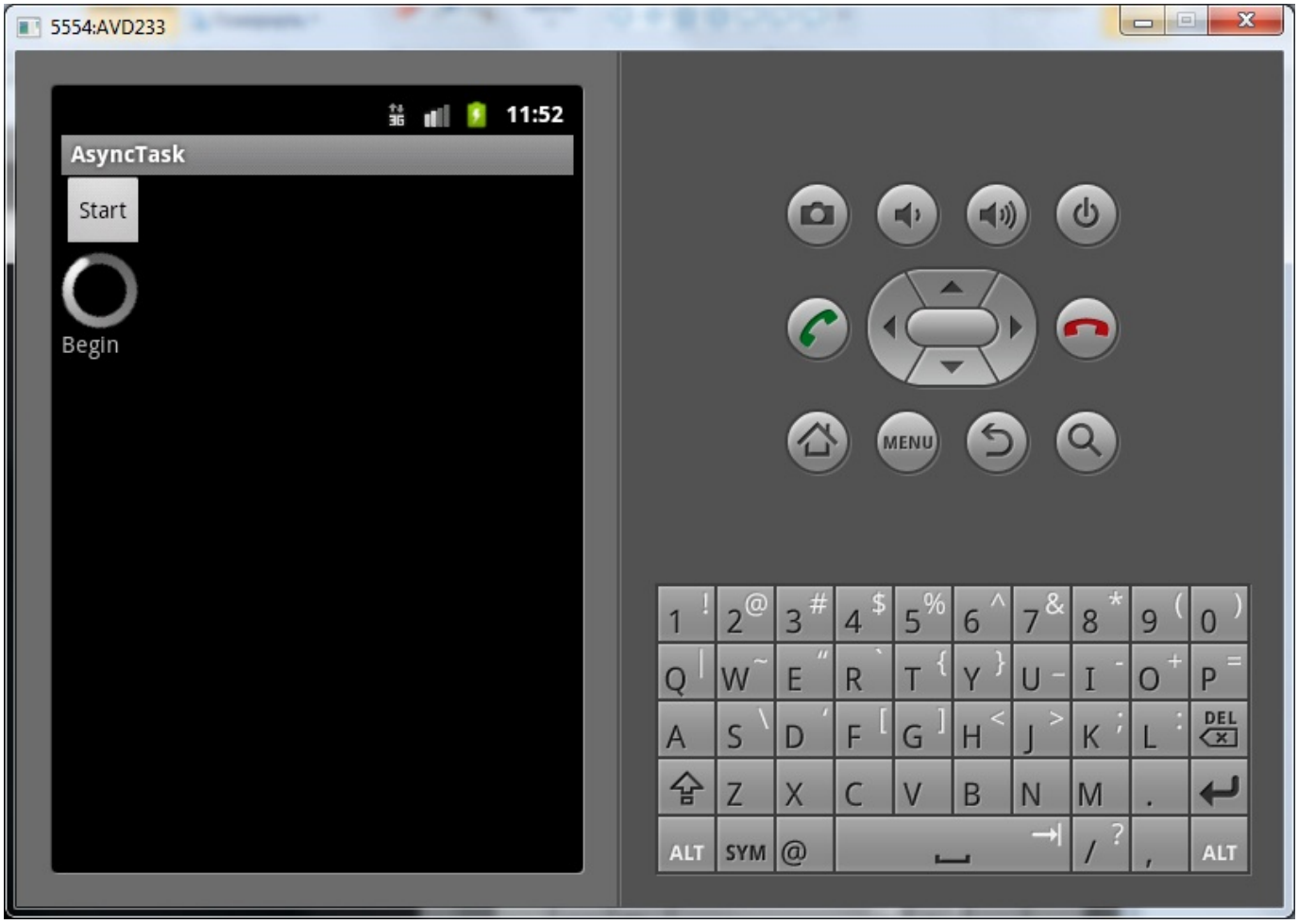
MyTask – это наш класс, наследующий AsyncTask. В нем мы прописываем свой код.

В методе **onPreExecute** выводим текст Begin.

В методе **doInBackground** эмулируем тяжелый код - делаем паузу на две секунды.

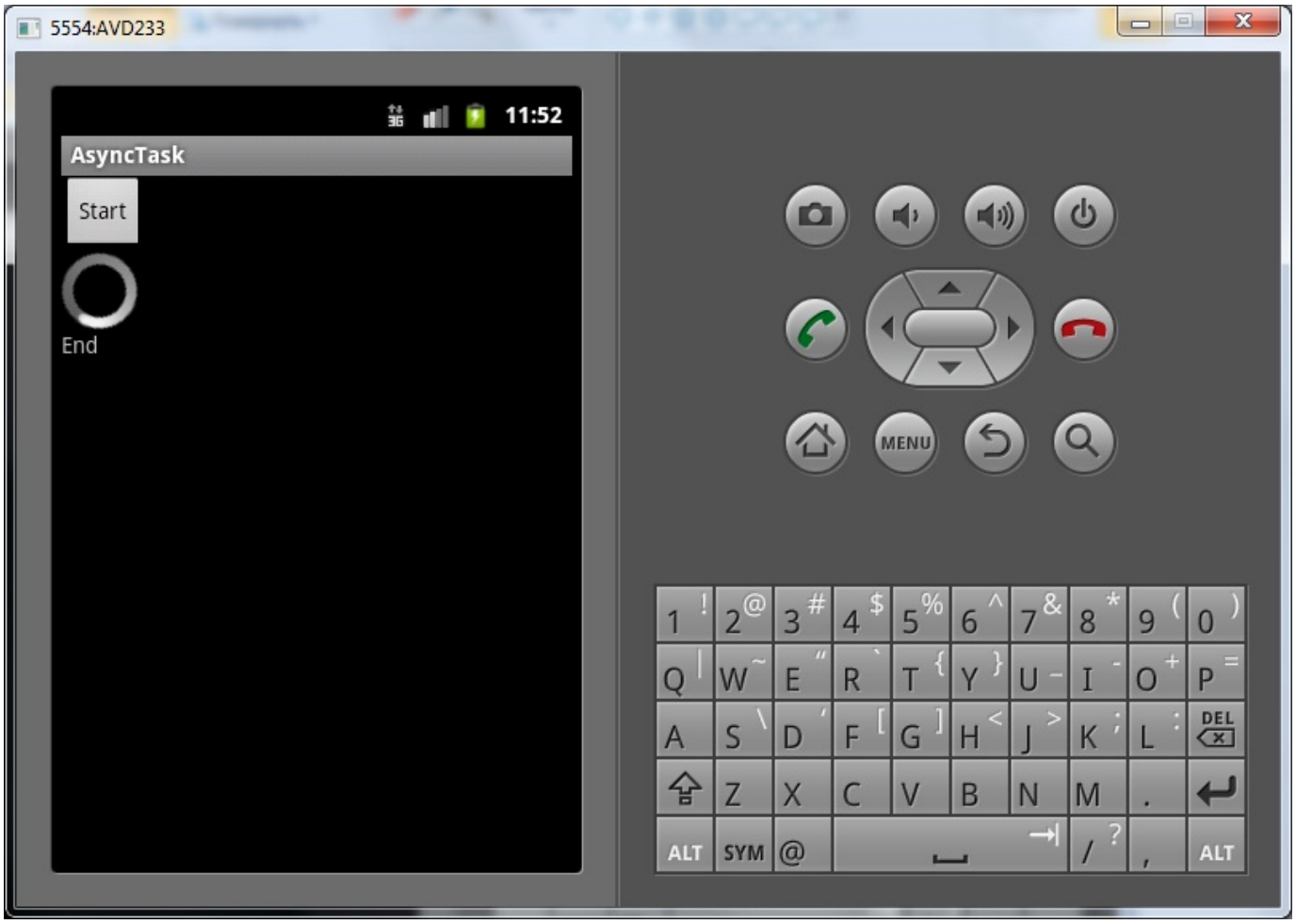
В методе **onPostExecute** выводим текст End.

Все сохраним и запустим приложение. Жмем **Start**, появляется текст Begin



ProgressBar работает, значит, основной поток (отвечающий за UI) - свободен.

Через две секунды появляется текст End



Мы получили работающую в новом потоке задачу и доступ к UI. При этом сами не создавали **потоков** и **Handler**, за нас все сделал **AsyncTask**.

Из рассмотренных методов обязательно надо реализовать только **doInBackground**. Остальные два – по желанию.

Этот пример показывает самые простые возможности. Мы везде использовали Void, чтобы пока не возиться с параметрами. Их и прочие возможности мы рассмотрим подробнее в следующих уроках. Пока что необходимо усвоить, в какой последовательности, и в каких потоках выполняются рассмотренные нами методы AsyncTask.

Обратите внимание, что если мы нажмем на Start, пока работает AsyncTask, то создастся и запустится новый AsyncTask. Но старый при этом никуда не денется, и продолжит работу в своем потоке. Т.е. они оба будут работать и обновлять один и тот же экран. Позже мы узнаем, как с этим можно справиться.

Официальный хелп дает **4 правила** использования AsyncTask, я также укажу их здесь:

- объект AsyncTask должен быть создан в UI-потоке

- метод execute должен быть вызван в UI-потоке

- не вызывайте напрямую методы onPreExecute, doInBackground, onPostExecute и onProgressUpdate (последний мы пока не проходили)

- AsyncTask может быть запущен (execute) только один раз, иначе будет exception

На следующем уроке:

- используем параметры
- выводим промежуточные результаты

Урок 87. AsyncTask. Параметры. Промежуточные результаты

В этом уроке:

- используем параметры
- выводим промежуточные результаты

На прошлом уроке мы увидели, как AsyncTask может взаимодействовать с экраном **до** и **после** выполнения задачи. Но часто бывает, что нужно **в процессе работы** выводить промежуточный результат. На этом уроке посмотрим, как это можно реализовать. А также разберемся, как можно передавать **данные** в AsyncTask.

При описании класса-наследника AsyncTask мы в угловых скобках указываем три типа данных:

1) Тип **входных** данных. Это данные, которые пойдут на вход AsyncTask

2) Тип **промежуточных** данных. Данные, которые используются для вывода промежуточных результатов

3) Тип **возвращаемых** данных. То, что вернет AsyncTask после работы.

В прошлом уроке мы указали <Void, Void, Void>. Это означало, что мы не используем параметры. Теперь попробуем воспользоваться ими. Нам надо вместо Void указать типы данных, которые будем использовать. В этом уроке используем первые два, третий пока оставим Void. Первые два – это у нас входные данные и промежуточные результаты.

Мы будем писать приложение, эмулирующее загрузку файлов. На вход в AsyncTask будем давать адреса файлов - для этих данных используем тип String. В процессе загрузки файлов будем выдавать кол-во уже загруженных файлов, как промежуточные результаты. Для этих данных можно использовать тип Integer. Т.е. в угловых скобках первый тип у нас будет String, а второй – Integer. Третий мы не используем, там снова укажем Void.

Создадим проект:

Project name: P0871_AsyncTaskParams
Build Target: Android 2.3.3
Application name: AsyncTaskParams
Package name: ru.startandroid.develop.p0871asynctaskparams
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AsyncTaskParams</string>
    <string  name="start">Start</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onclick"
        android:text="@string/start">
    </Button>
    <TextView
        android:id="@+id/tvInfo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="">
    </TextView>
</LinearLayout>
```

Нажав на кнопку будем стартовать задачу, в TextView будем выводить информацию о работе.

MainActivity.java:

```
package ru.startandroid.develop.p0871asynctaskparams;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

  MyTask mt;
  TextView tvInfo;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvInfo = (TextView) findViewById(R.id.tvInfo);
  }

  public void onclick(View v) {
    mt = new MyTask();
    mt.execute("file_path_1", "file_path_2", "file_path_3", "file_path_4");
  }

  class MyTask extends AsyncTask<String, Integer, Void> {

    @Override
    protected void onPreExecute() {
```

```
      super.onPreExecute();
      tvInfo.setText("Begin");
    }

    @Override
    protected Void doInBackground(String... urls) {
      try {
        int cnt = 0;
        for (String url : urls) {
          // загружаем файл
          downloadFile(url);
          // выводим промежуточные результаты
          publishProgress(++cnt);
        }
        // разъединяемся
        TimeUnit.SECONDS.sleep(1);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      return null;
    }


    @Override
    protected void onProgressUpdate(Integer... values) {
      super.onProgressUpdate(values);
      tvInfo.setText("Downloaded " + values[0] + " files");
    }

    @Override
    protected void onPostExecute(Void result) {
      super.onPostExecute(result);
      tvInfo.setText("End");
    }


    private void downloadFile(String url) throws InterruptedException {
      TimeUnit.SECONDS.sleep(2);
    }
  }
}
```

При описании класса-наследника AsyncTask мы использовали типы данных <String, Integer, Void>. Напомню, что **первый** (String) используется для **входных** данных, **второй** (Integer) – для **промежуточных** данных, **третий** (Void) – для **выходных** данных (пока не используем).

В методе **onPreExecute**, который выполняется перед началом работы AsyncTask, мы пишем в TextView текст Begin.

Обратите внимание на тип входных данных в методе **doInBackground**. Это String. Т.е. здесь используется первый тип из угловых скобок, тип входных данных. В нашей задаче, мы подаем на вход AsyncTask набор адресов файлов для загрузки. Метод doInBackground принимает эти данные и в цикле по одному загружает эти файлы. После загрузки каждого файла он вызывает метод publishProgress и передает туда кол-во загруженных файлов. После загрузки всех файлов отключаемся.

Когда мы в doInBackground вызываем метод publishProgress и передаем туда данные, срабатывает метод

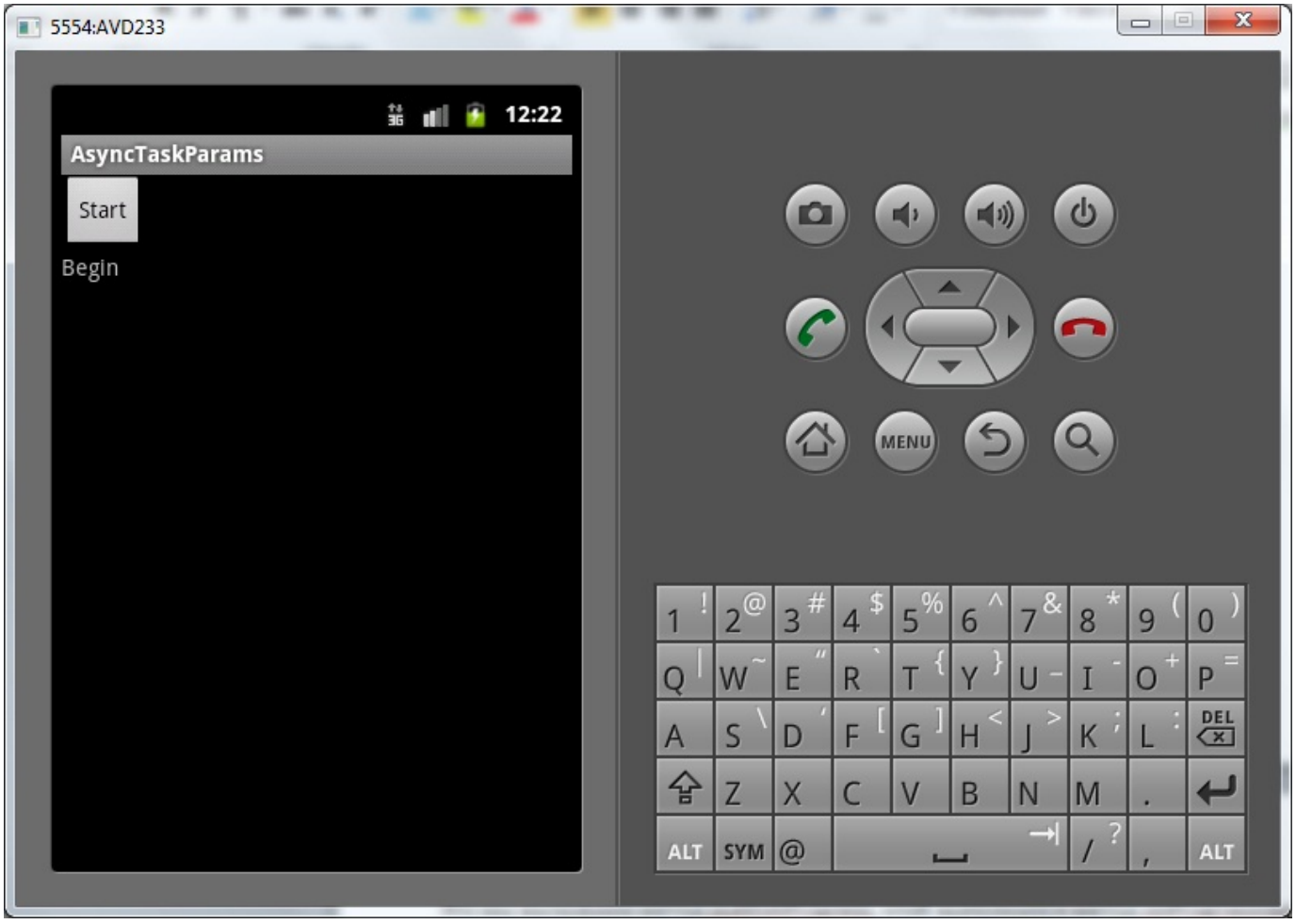
onProgressUpdate и получает эти данные. Тип этих данных равен второму типу из угловых скобок, т.е. Integer. Метод onProgressUpdate используется для вывода промежуточных результатов. Он выполняется в основном потоке и имеет доступ к UI. Из doInBackground мы (с помощью publishProgress) передали в onProgressUpdate кол-во загруженных файлов – это кол-во и выводим в TextView.

Метод **onPostExecute** выполняется по завершению задачи, выводим текст End в TextView.

Метод **downloadFile** – эмуляция загрузки файлов, просто пауза в 2 секунды.

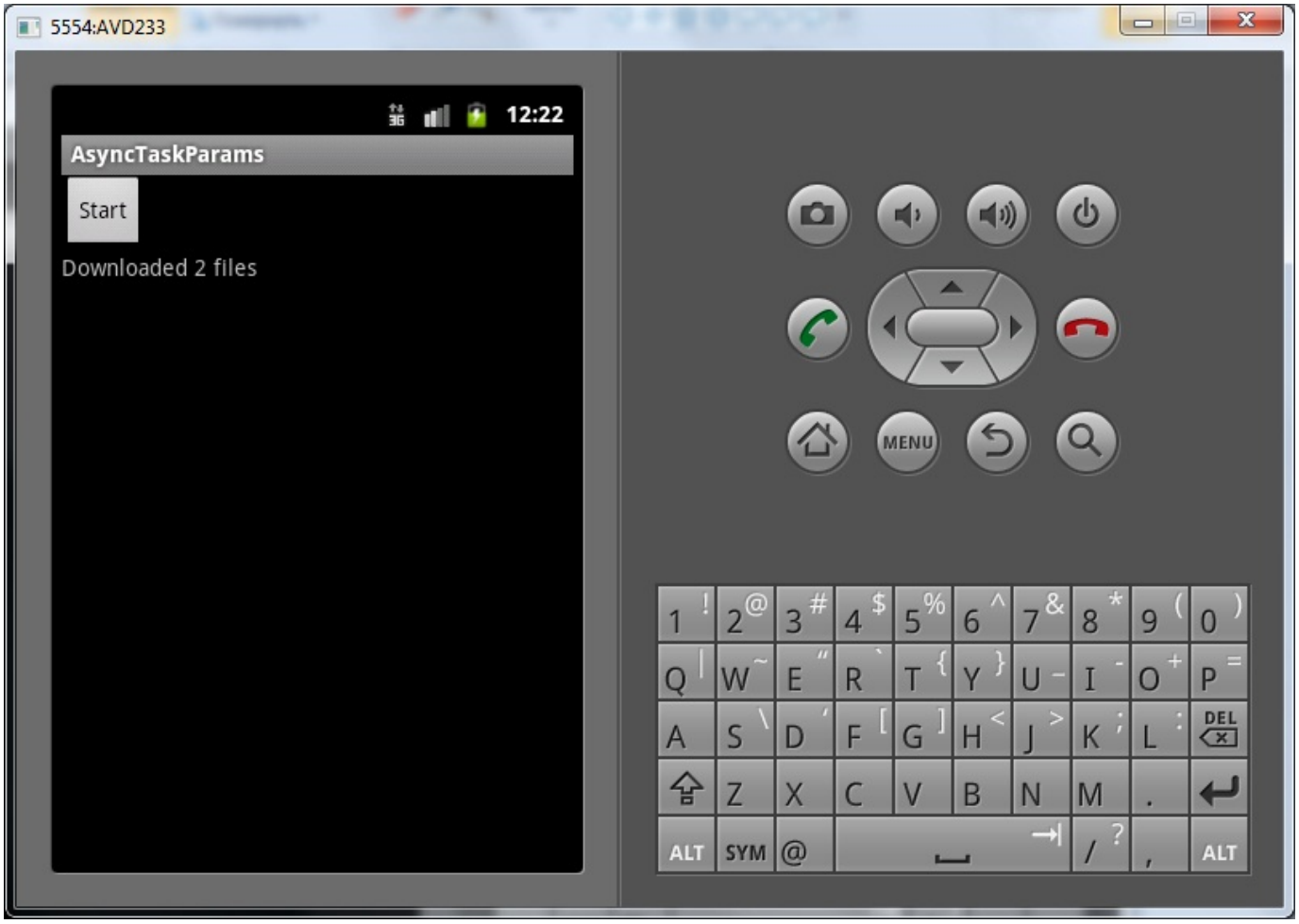
В методе **onclick** мы создаем AsyncTask и запускаем его методом execute. На вход этому методу передаем набор переменных типа String. String используется, потому что мы его указали в угловых скобках на первом месте – это тип входных данных. Я передаю данные просто перечислив их через запятую, но можно использовать и массив.

Все сохраним и запустим. Нажмем **Start**. Появился текст Begin



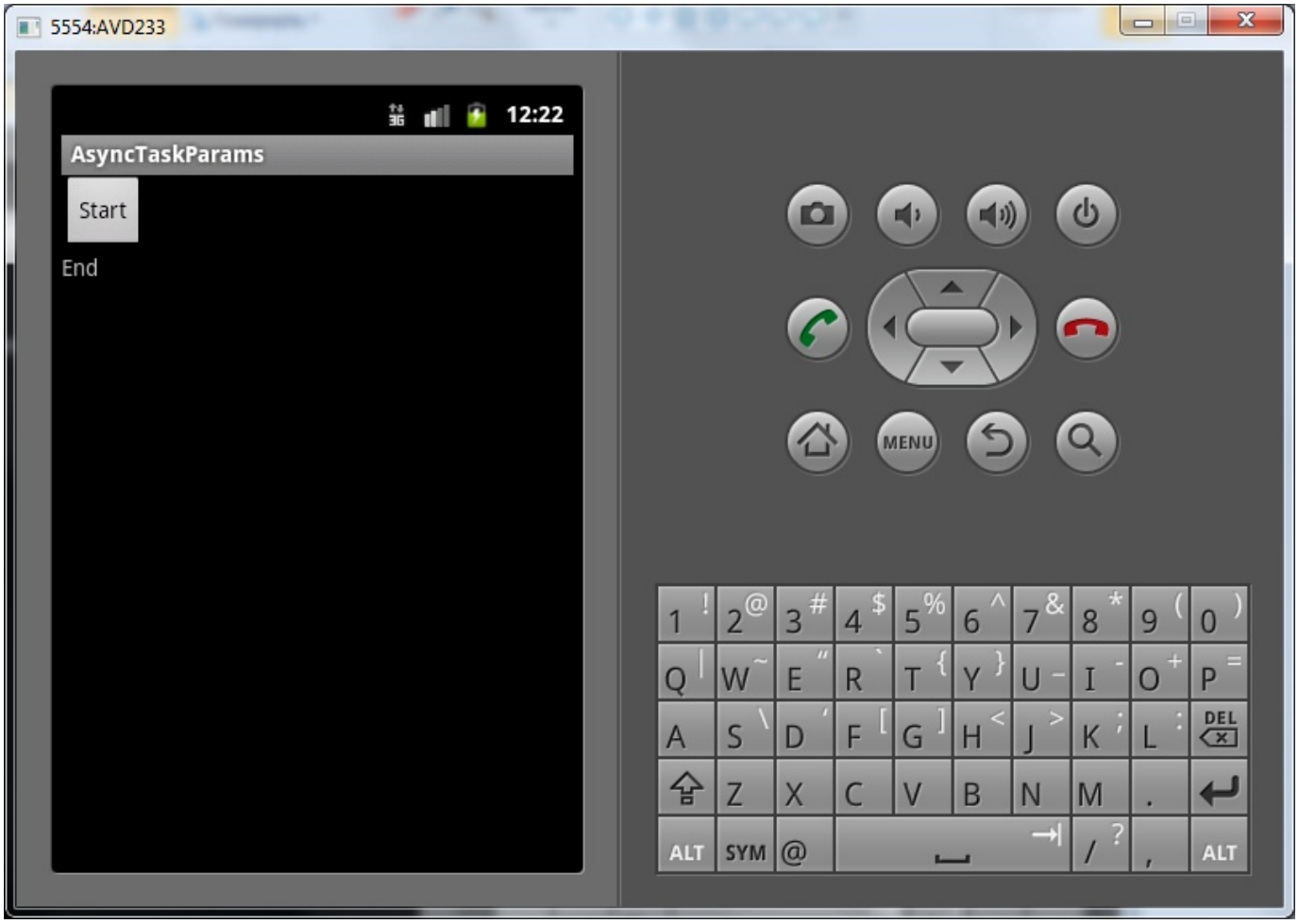
Т.е. сработал метод onPreExecute

Далее пошел отсчет загруженных файлов



Это мы вызываем метод publishProgress, чтоб выполнился метод onProgressUpdate и обновил экран

Появился текст End



Это выполнился onPostExecute

Сейчас, наверно, кажется, что методов целая куча и в них можно запутаться, попробуем резюмировать:

execute – этот метод мы явно вызываем, чтобы начать выполнение задачи. В него мы передаем набор данных определенного типа. Этот тип указан первым в угловых скобках при описании AsyncTask (в нашем примере это String).

onPreExecute и **onPostExecute** – их мы сами явно не вызываем, они вызываются системой в начале и конце выполнения задачи.

doInBackground – в нем мы указываем, что нам надо сделать в новом потоке. На вход поступают данные, которые мы передали в execute. Явно не вызываем.

publishProgress – явно вызываем в методах doInBackground, onPreExecute или onPostExecute. На вход передаем промежуточные данные определенного типа. Этот тип указан вторым в угловых скобках при описании AsyncTask (в нашем примере это Integer).

onProgressUpdate – метод получает на вход промежуточные результаты. Сами не вызываем, вместо этого используем метод publishProgress. То, что передаем в publishProgress, попадает в onProgressUpdate.

Еще имеет смысл пояснить такой момент. Метод **onProgressUpdate** принимает на вход набор параметров. Мы же в этом уроке передаем ему (через **publishProgress**) всего одно значение. Поэтому чтобы прочесть это значение мы берем первый элемент массива ([0]).

Ну и на всякий случай напишу, что в этом примере мы использовали String и Integer, но вы в решении ваших задач можете использовать какие угодно типы-классы, хоть свои собственные.

На следующем уроке:

- используем третий параметр
- используем метод get для получения результата

Урок 88. AsyncTask. Итоговый результат. Метод get

В этом уроке:

- используем третий параметр
- используем метод get для получения результата

На прошлом уроке мы рассмотрели типы данных, которые указываем в угловых скобках при описании класса-наследника AsyncTask. Их три:

1) Тип **входных** данных. Это данные, которые пойдут на вход AsyncTask

2) Тип **промежуточных** данных. Данные, которые используются для вывода промежуточных результатов

3) Тип **возвращаемых** данных. То, что вернет AsyncTask после работы.

Мы уже поработали с первыми двумя. Сейчас будем использовать **третий**. Это тип (класс) объекта, который должен нам **вернуться** из AsyncTask. Получить этот объект мы можем двумя способами:

1) Он передается на вход метода **onPostExecute**, который срабатывает по окончании задачи

2) Метод get возвращает нам этот объект

Мы вызываем метод get, чтобы получить результат работы AsyncTask. Но что будет, если задача еще не завершена, а мы вызвали get? Метод get будет ждать. Т.е. просто блокирует поток, в котором он выполняется, и не отпустит, пока не получит какой-то результат или не выскочит exception.

Есть еще реализация метода get с таймаутом. В этом случае get будет ждать указанное время, а потом сгенерирует Exception. Если же задача уже была завершена, то метод выполнится сразу и ждать ничего не будет.

Посмотрим на примере. Создадим приложение, будем запускать задачу, которая будет висеть 5 секунд, и попробуем повызывать метод get.

Создадим проект:

Project name: P0881_AsyncTaskResult
Build Target: Android 2.3.3
Application name: AsyncTaskResult
Package name: ru.startandroid.develop.p0881asynctaskresult
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AsyncTaskResult</string>
    <string  name="start">Start</string>
    <string  name="get">Result</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
          android:id="@+id/btnStart"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onclick"
          android:text="@string/start">
     </Button>
     <Button
          android:id="@+id/btnGet"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/get"
          android:onClick="onclick">
     </Button>
     <ProgressBar
          android:layout_width="wrap_content"
          android:layout_height="wrap_content">
     </ProgressBar>
     <TextView
          android:id="@+id/tvInfo"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="">
     </TextView>
</LinearLayout>
```

Кнопка Start будет начинать выполнение задачи, а кнопка Result – выводить результат. ProgressBar, который покажет, что приложение висит. TextView для вывода информации.

MainActivity.java:

```
package ru.startandroid.develop.p0881asynctaskresult;

import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {
```

```
final String LOG_TAG = "myLogs";

MyTask mt;
TextView tvInfo;

public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  tvInfo = (TextView) findViewById(R.id.tvInfo);
}

public void onclick(View v) {
  switch (v.getId()) {
  case R.id.btnStart:
    mt = new MyTask();
    mt.execute();
    break;
  case R.id.btnGet:
    showResult();
    break;
  default:
    break;
  }
}

private void showResult() {
  if (mt == null) return;
  int result = -1;
  try {
    Log.d(LOG_TAG, "Try to get result");
    result = mt.get();
    Log.d(LOG_TAG, "get returns " + result);
    Toast.makeText(this, "get returns " + result, Toast.LENGTH_LONG).show();
  } catch (InterruptedException e) {
    e.printStackTrace();
  } catch (ExecutionException e) {
    e.printStackTrace();
  }
}

class MyTask extends AsyncTask<Void, Void, Integer> {
  @Override
  protected void onPreExecute() {
    super.onPreExecute();
    tvInfo.setText("Begin");
    Log.d(LOG_TAG, "Begin");
  }


  @Override
  protected Integer doInBackground(Void... params) {
    try {
      TimeUnit.SECONDS.sleep(5);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    return 100500;
  }
```

```
    @Override
    protected void onPostExecute(Integer result) {
      super.onPostExecute(result);
      tvInfo.setText("End. Result = " + result);
      Log.d(LOG_TAG, "End. Result = " + result);

    }
  }
}
```

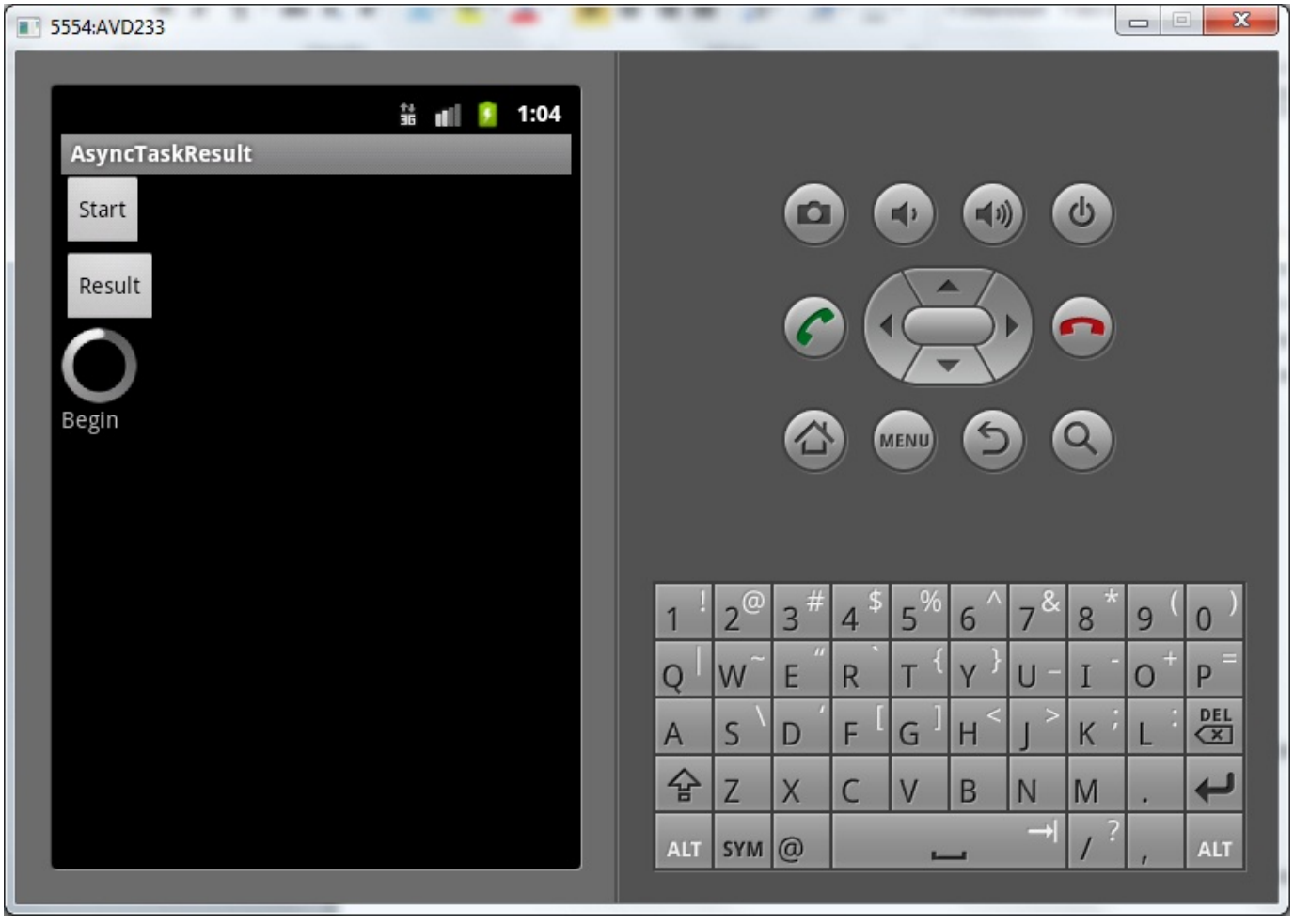
В **onclick** мы определяем, какая кнопка была нажата. Если Start, то создаем и запускаем задачу. Если Result, то вызываем метод showResult.

В **showResult** мы пытаемся получить результат из текущей задачи с помощью метода get. Результат выводим в лог и на экран.

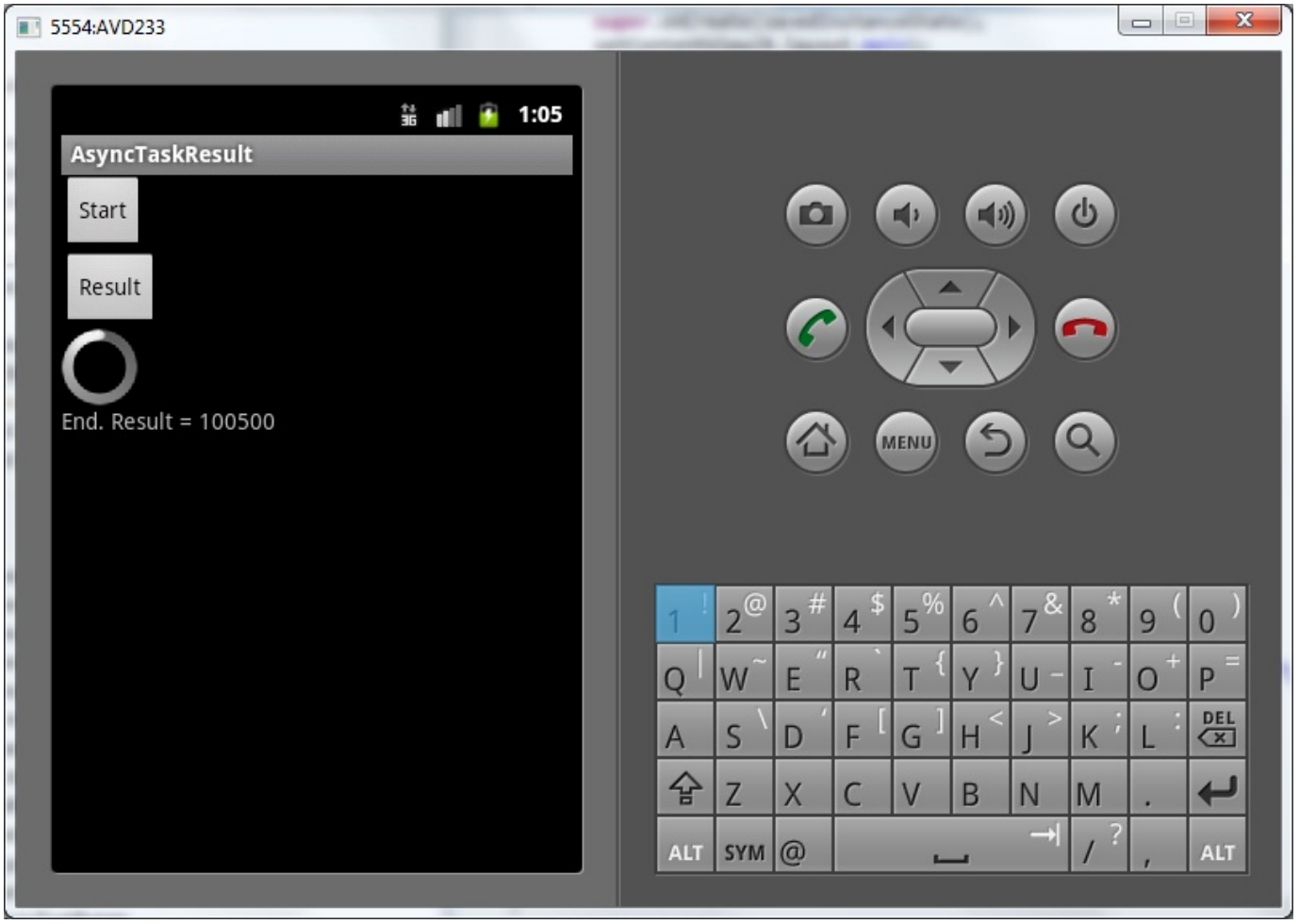
В описании AsyncTask мы указали Void для первых двух типов данных и Integer для третьего. Этот третий Integer и есть тип (класс) объекта, который должен нам вернуться из AsyncTask. Обратите внимание, что этот тип указан как тип **возвращаемых** данных для метода **doInBackground** и как **входящий** для метода **onPostExecute**. Т.е. мы выполняем задачу в **doInBackground**, формируем **результат** типа Integer и **возвращаем** его как результат работы этого метода. И далее этот результат попадает и может быть использован в **onPostExecute**, который выполняется после завершения задачи. Мы выводим его в лог и на экран.

В **doInBackground** мы ставим паузу в 5 секунд и возвращаем число 100500.

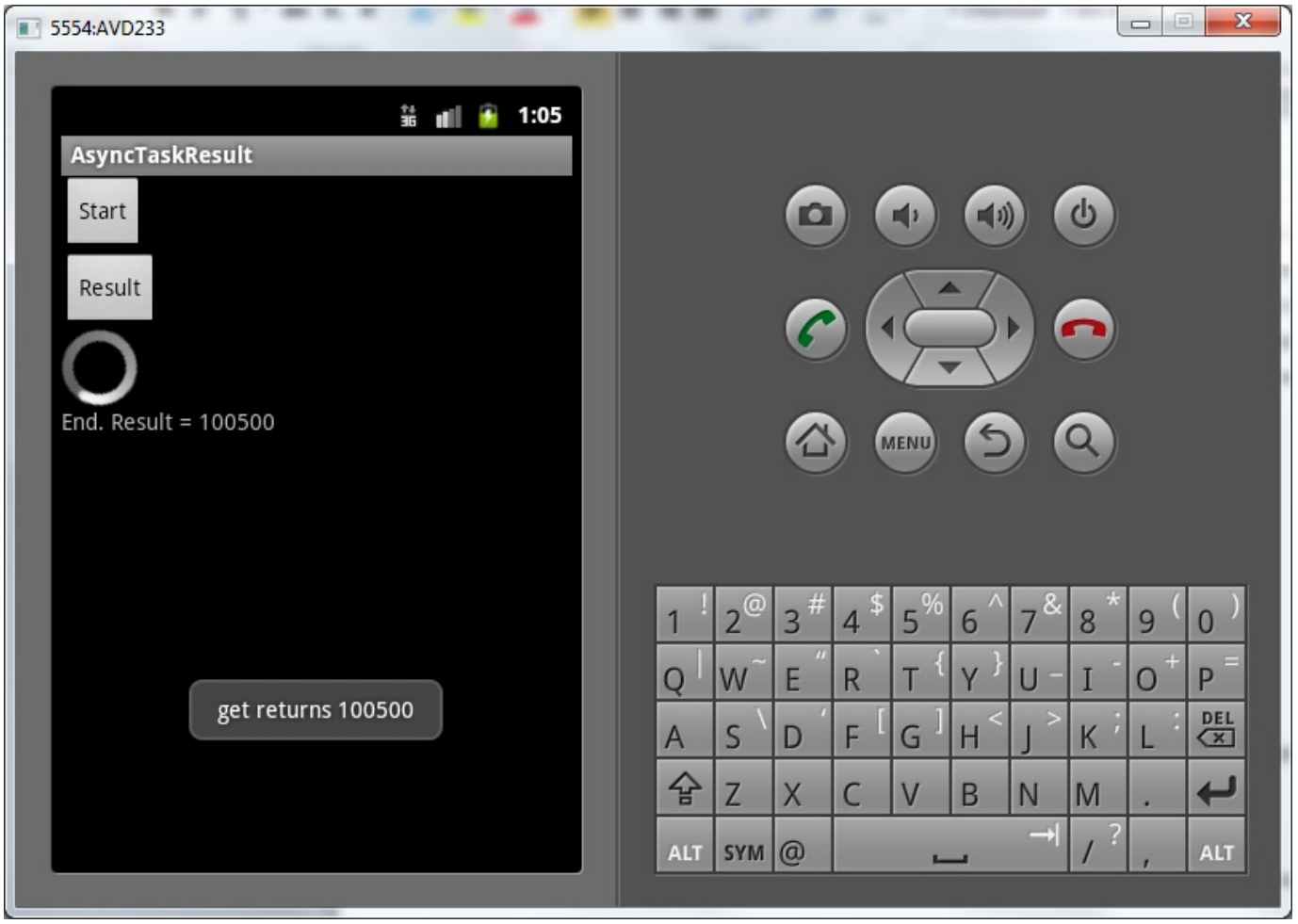
Все сохраним и запустим приложение. Жмем **Start**



и ждем 5 секунд, пока отработает задача. Появляется текст с результатом.



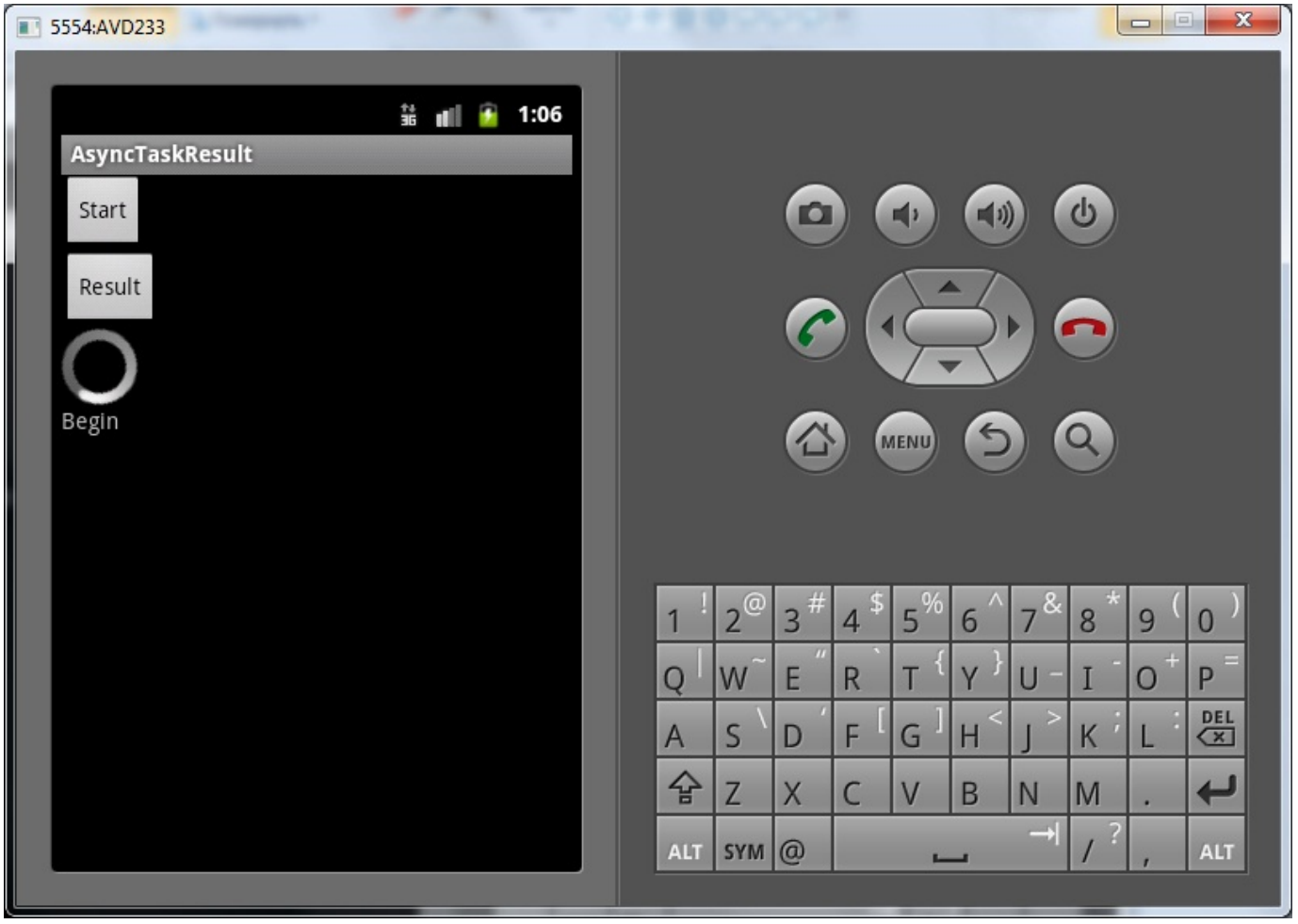
Теперь нажмем **Result**



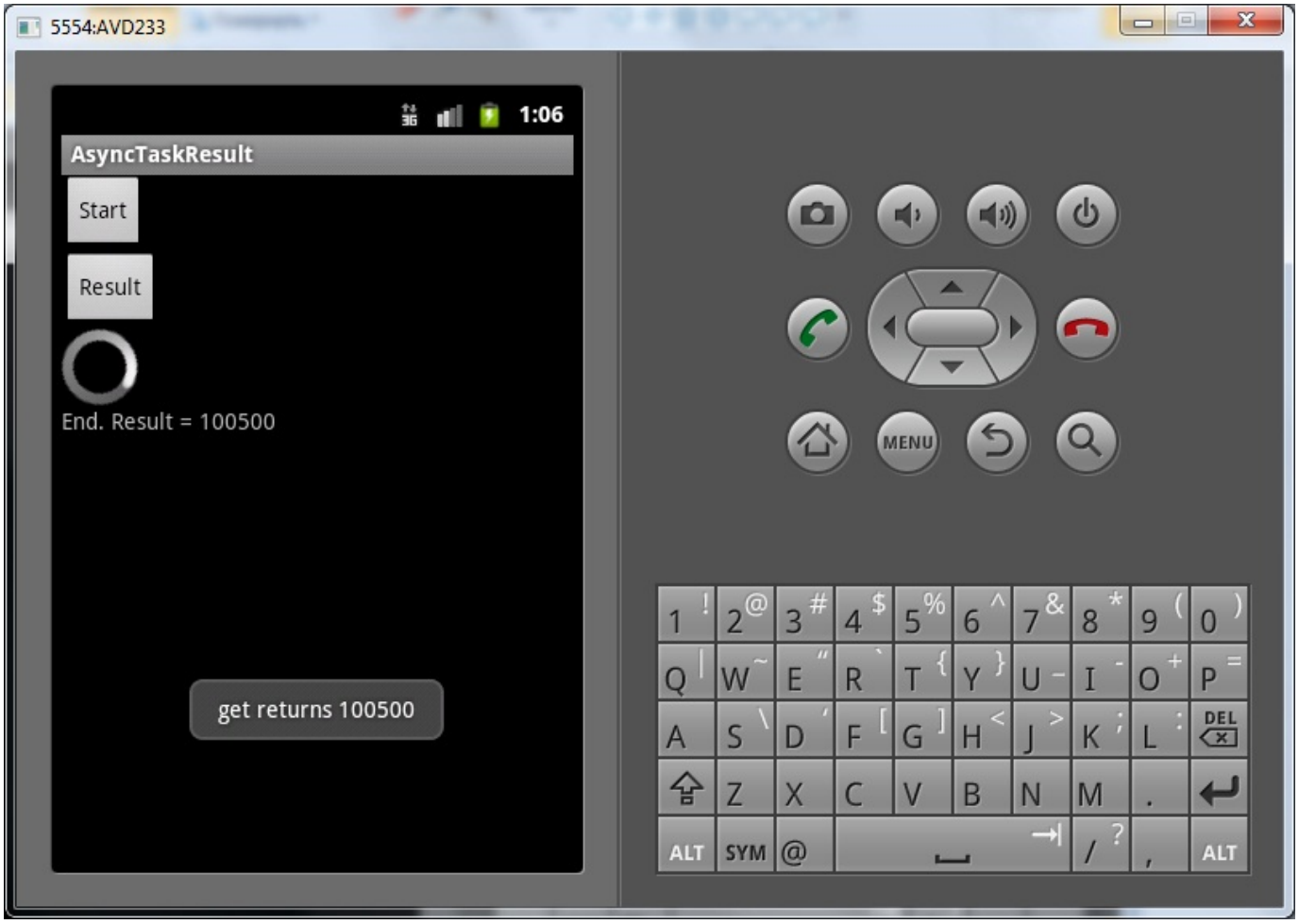
AsyncTask дал нам тот же самый результат, который пришел в onPostExecute. Все логично.

Теперь попробуем вызвать метод get пока не завершилась задача. Для этого надо нажать **Start** и сразу же нажать **Result**

Жмем **Start**, и сразу за ним **Result**



остановился **ProgressBar**, т.к. основной поток **блокирован** методом get. Метод get ждет завершения AsyncTask. Это продолжается пока не выполнится то, что мы написали в **doInBackground**. После этого метод get возвращает нам результат и освобождает поток.



Смотрим логи:

13:06:27.224: D/myLogs(656): Begin
13:06:27.479: D/myLogs(656): Try to get result
13:06:32.316: D/myLogs(656): get returns 100500
13:06:32.364: D/myLogs(656): End. Result = 100500

Видно, как началась задача (***Begin***). Почти сразу же нажали на Result и запустили метод get (***Try to get result***). А вот отработал (***get returns 100500***) метод get только спустя 5 секунд, т.е. пока не завершилась задача. Все это время он ждал и, тем самым, блокировал поток, в котором был запущен. Примерно в то же время отработал и onPostExecute (***End. Result = 100500***).

Попробуем добавить таймаут. Перепишем метод **showResult**:

```
private void showResult() {
  if (mt == null) return;
  int result = -1;
  try {
    Log.d(LOG_TAG, "Try to get result");
    result = mt.get(1, TimeUnit.SECONDS);
    Log.d(LOG_TAG, "get returns " + result);
    Toast.makeText(this, "get returns " + result, Toast.LENGTH_LONG).show();
  } catch (InterruptedException e) {
    e.printStackTrace();
  } catch (ExecutionException e) {
```

```
            e.printStackTrace();
        } catch (TimeoutException e) {
            Log.d(LOG_TAG, "get timeout, result = " + result);
            e.printStackTrace();
        }
    }
```

Изменился метод get. Теперь он ждет одну секунду, и если не получает результат, то генерирует **TimeoutException**. Мы это исключение ловим и выводим в лог соответствующее сообщение.

Сохраняем, запускаем. Жмем **Start** и сразу же **Result**. Приложение снова подвисает, но через секунду оживает. Смотрим лог:

13:10:42.574: D/myLogs(724): Begin
13:10:42.875: D/myLogs(724): Try to get result
13:10:43.883: D/myLogs(724): get timeout, result = -1
13:10:47.615: D/myLogs(724): End. Result = 100500

Видно, как началась задача (***Begin***). Почти сразу же нажали на Result и запустили метод get (***Try to get result***). Через секунду сработал TimeoutException, и видим, что get не вернул нам никакой результат (***get timeout, result = -1***). Ну а через 5 секунд после начала успешно сработал onPostExecute (***End. Result = 100500***).

На следующем уроке:

- отменяем задачу в процессе выполнения

Урок 89. AsyncTask. Cancel – отменяем задачу в процессе выполнения

В этом уроке:

- отменяем задачу в процессе выполнения

Иногда возникает необходимость отменить уже выполняющуюся задачу. Для этого в AsyncTask есть метод cancel. Он на вход принимает boolean-параметр, который указывает, может ли система прервать выполнение потока.

Но вообще, рекомендуется не ждать, пока система завершит поток, а действовать самим. В doInBackground мы должны периодически вызывать метод isCancelled. Как только мы выполним метод cancel для AsyncTask, isCancelled будет возвращать true. А это значит, что мы должны завершить метод doInBackground.

Т.е. метод **cancel** – это мы **ставим метку**, что задачу надо отменить. Метод **isCancelled** – мы же сами эту **метку читаем** и предпринимаем действия, для завершения работы задачи.

Метод **cancel** возвращает boolean. Мы получим **false**, если задача уже **завершена** или **отменена**.

Рассмотрим на примере.

Создадим проект:

Project name: P0891_AsyncTaskCancel
Build Target: Android 2.3.3
Application name: AsyncTaskCancel
Package name: ru.startandroid.develop.p0891asynctaskcancel
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AsyncTaskCancel</string>
    <string  name="start">Start</string>
    <string  name="cancel">Cancel</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnStart"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:onClick="onclick"
```

```
            android:text="@string/start">
        </Button>
        <Button
            android:id="@+id/btnCancel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onclick"
            android:text="@string/cancel">
        </Button>
        <TextView
            android:id="@+id/tvInfo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="">
        </TextView>
    </LinearLayout>
```

Кнопки старта и отмены задачи, и TextView для вывода текста.

MainActivity.java:

```
package ru.startandroid.develop.p0891asynctaskcancel;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  MyTask mt;
  TextView tvInfo;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvInfo = (TextView) findViewById(R.id.tvInfo);
  }

  public void onclick(View v) {
    switch (v.getId()) {
    case R.id.btnStart:
      mt = new MyTask();
      mt.execute();
      break;
    case R.id.btnCancel:
      cancelTask();
      break;
```

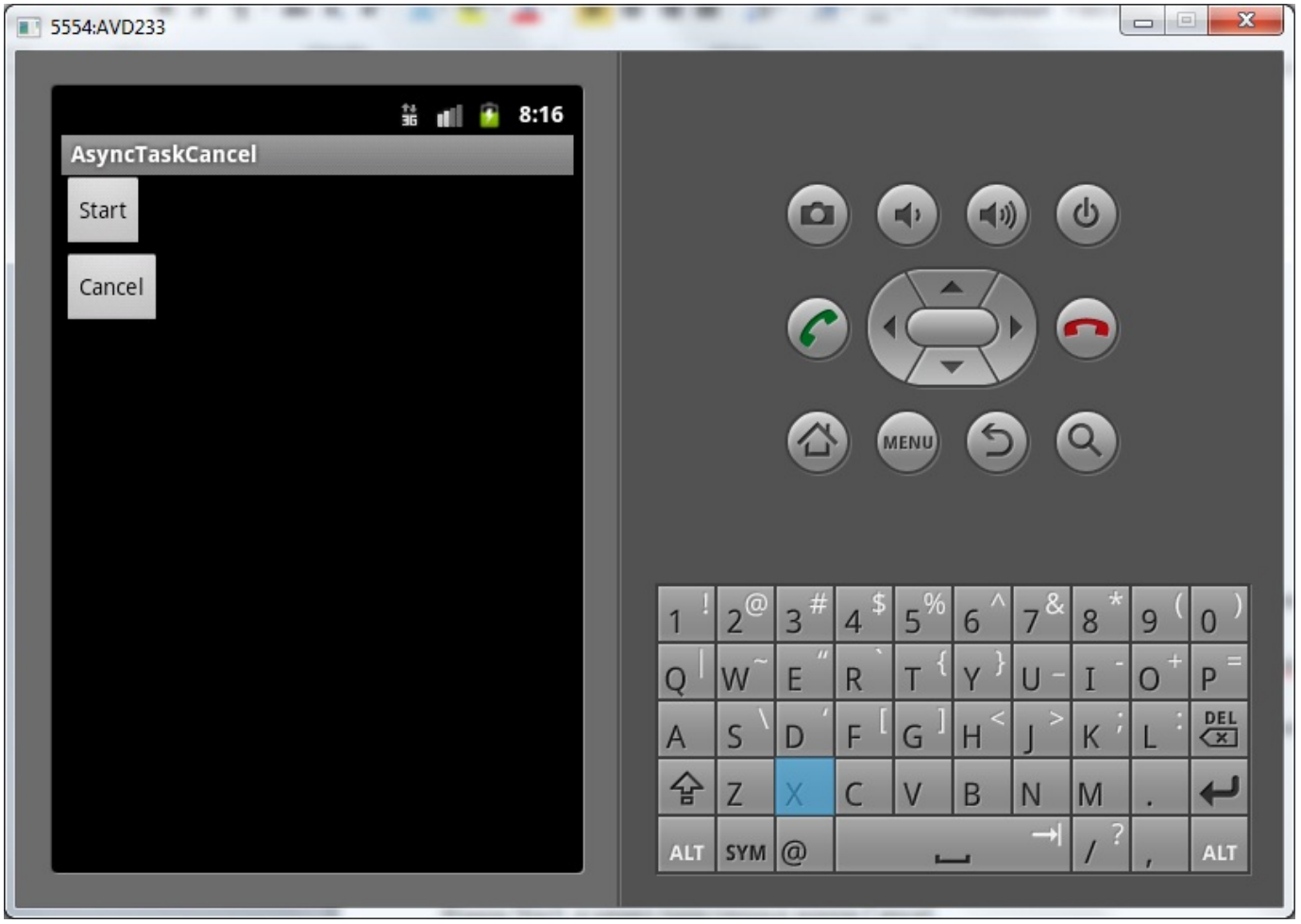
```
      default:
        break;
      }
    }

    private void cancelTask() {
      if (mt == null) return;
      Log.d(LOG_TAG, "cancel result: " + mt.cancel(false));
    }

    class MyTask extends AsyncTask<Void, Void, Void> {
      @Override
      protected void onPreExecute() {
        super.onPreExecute();
        tvInfo.setText("Begin");
        Log.d(LOG_TAG, "Begin");
      }

      @Override
      protected Void doInBackground(Void... params) {
        try {
          for (int i = 0; i < 5; i++) {
            TimeUnit.SECONDS.sleep(1);
            Log.d(LOG_TAG, "isCancelled: " + isCancelled());
          }
        } catch (InterruptedException e) {
          Log.d(LOG_TAG, "Interrupted");
          e.printStackTrace();
        }
        return null;
      }

      @Override
      protected void onPostExecute(Void result) {
        super.onPostExecute(result);
        tvInfo.setText("End");
        Log.d(LOG_TAG, "End");
      }

      @Override
      protected void onCancelled() {
        super.onCancelled();
        tvInfo.setText("Cancel");
        Log.d(LOG_TAG, "Cancel");

      }
    }
  }
```

По нажатию кнопки **Cancel** выполняется метод cancelTask, в котором выполняем cancel (с параметром false) для AsyncTask.

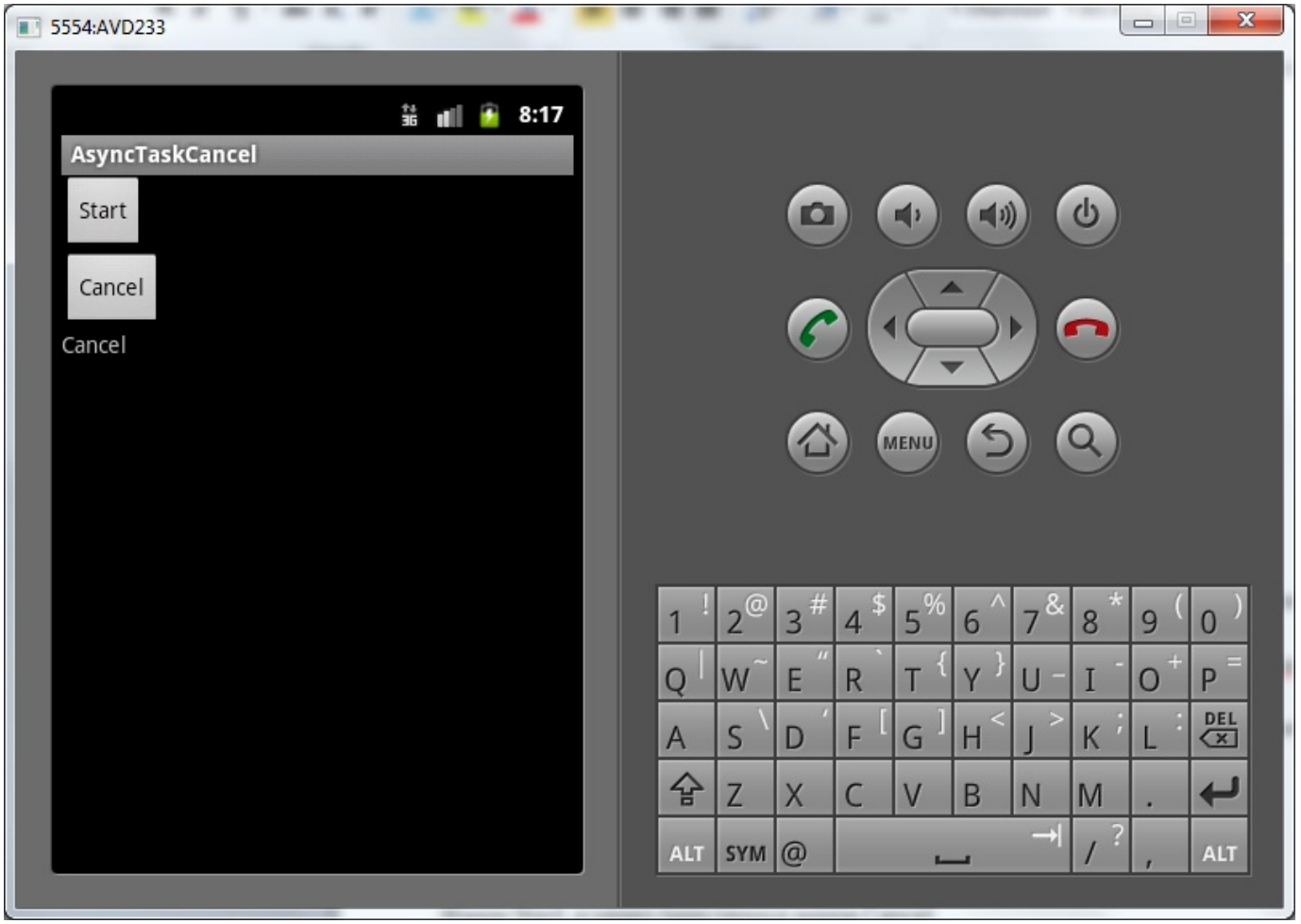
В **doInBackground** в цикле гоняем паузы и выводим в лог результат метода isCancelled.

Метод onCancelled вызывается системой вместо onPostExecute, если задача была отменена.

Все сохраним и запустим приложение.



Жмем **Start**, а через пару секунд жмем **Cancel**.



Смотрим логи:

08:17:51.956: D/myLogs(487): Begin
08:17:52.993: D/myLogs(487): isCancelled: false
08:17:53.998: D/myLogs(487): isCancelled: false
08:17:54.543: D/myLogs(487): cancel result: true
08:17:54.552: D/myLogs(487): Cancel
08:17:55.042: D/myLogs(487): isCancelled: true
08:17:56.061: D/myLogs(487): isCancelled: true
08:17:57.111: D/myLogs(487): isCancelled: true

Мы видим, что в первых двух циклах задачи метод isCancelled возвращал false. Затем мы нажали Cancel (***cancel result: true***). Сразу же сработал метод onCancelled (***Cancel***). А метод doInBackground продолжил свою работу и докрутил цикл до конца. Но при этом метод onPostExecute, который обычно вызывается в конце задачи, не был вызван вообще, потому что мы отменили задачу (методом cancel).

Т.е. мы хоть cancel и выполнили, но задача продолжила работать. Завершать задачу надо нам самим. Для этого мы читаем **isCancelled** и, если он **true**, то **завершаем** метод doInBackground. Т.е. в нашем случае надо переписать метод doInBackground:

```
    protected Void doInBackground(Void... params) {
      try {
        for (int i = 0; i < 5; i++) {
          TimeUnit.SECONDS.sleep(1);
          if (isCancelled()) return null;
          Log.d(LOG_TAG, "isCancelled: " + isCancelled());
        }
      } catch (InterruptedException e) {
        Log.d(LOG_TAG, "Interrupted");
        e.printStackTrace();
      }
      return null;
    }
```

Мы просто добавили проверку **isCancelled**. Если он возвращает true, то выходим (return). Разумеется, в более сложных задачах может потребоваться более продуманная логика выхода.

Теперь если мы нажмем **Cancel** в процессе выполнения задачи, doInBackground остановит свою работу, как только сможет:

08:40:12.439: D/myLogs(440): Begin
08:40:13.498: D/myLogs(440): isCancelled: false
08:40:14.558: D/myLogs(440): isCancelled: false
08:40:15.118: D/myLogs(440): cancel result: true
08:40:15.138: D/myLogs(440): Cancel

Удалите или закомментируйте только что добавленную строку:

```
if (isCancelled()) return null;
```

в методе doInBackground. Нам сейчас будет не нужна явная проверка отмены задачи. Мы проверим, что сделает метод cancel, если передать в него true.

Перепишем cancelTask():

```
  private void cancelTask() {
```

```
    if (mt == null) return;
    Log.d(LOG_TAG, "cancel result: " + mt.cancel(true));
  }
```

В метод cancel передаем true. Т.е. он попытается сам остановить поток.

Сохраняем, запускаем приложение. Жмем **Start**, а через пару секунд жмем **Cancel**. Смотрим логи:

08:58:35.949: D/myLogs(545): Begin
08:58:37.023: D/myLogs(545): isCancelled: false
08:58:38.052: D/myLogs(545): isCancelled: false
08:58:38.688: D/myLogs(545): cancel result: true
08:58:38.698: D/myLogs(545): Interrupted
08:58:38.710: D/myLogs(545): Cancel

Мы видим, что метод doInBackground завершил работу, т.к. метод sleep сгенерировал InterruptedException (***Interrupted***). Т.е. когда используем sleep, остановка работы потока работает. Но не факт, что сработает в других случаях. Поэтому повторюсь: не надейтесь особо на cancel(true), а используйте проверку **isCancelled** или метод **onCancelled** для завершения своей задачи. Либо проверьте и убедитесь, что cancel(true) работает в ваших условиях.

Ну и для теста попробуйте нажать Cancel, когда задача уже завершена или отменена. В этом случае метод cancel вернет false.

P.S. Я тестировал это все на версии Android 2.3.3. На форуме, в ветке этого урока, было замечено, что поведение отмены задачи немного отличается в 4-й версии Android.

На следующем уроке:

- читаем статусы задачи

Урок 90. AsyncTask. Status – статусы задачи

В этом уроке:

- читаем статусы задачи

Мы всегда можем определить, в каком состоянии сейчас находится задача. Для этого используются статусы. Их всего три:

PENDING – задача еще не запущена

RUNNING – задача в работе

FINISHED – метод onPostExecute отработал, т.е. задача успешно завершена

В принципе по названиям все понятно, но посмотрим и убедимся сами. К тому же, не очень понятно, в каком статусе будет задача, если мы ее отменим методом cancel.

Рассмотрим на примере.

Создадим проект:

Project name: P0901_AsyncTaskStatus
Build Target: Android 2.3.3
Application name: AsyncTaskStatus
Package name: ru.startandroid.develop.p0901asynctaskstatus
Create Activity: MainActivity

strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="app_name">AsyncTaskStatus</string>
    <string  name="start">Start</string>
    <string  name="status">Status</string>
</resources>
```

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnStart"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
            android:onClick="onclick"
            android:text="@string/start">
        </Button>
        <Button
            android:id="@+id/btnStatus"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onclick"
            android:text="@string/status">
        </Button>
        <TextView
            android:id="@+id/tvInfo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="">
        </TextView>
    </LinearLayout>
```

По кнопке **Start** запускаем задачу, по кнопке **Status** будем выводить ее статус.

MainActivity.java:

```
package ru.startandroid.develop.p0901asynctaskstatus;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {

  MyTask mt;
  TextView tvInfo;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    tvInfo = (TextView) findViewById(R.id.tvInfo);
  }

  public void onclick(View v) {
    switch (v.getId()) {
    case R.id.btnStart:
      startTask();
      break;
    case R.id.btnStatus:
      showStatus();
      break;
    default:
      break;
```

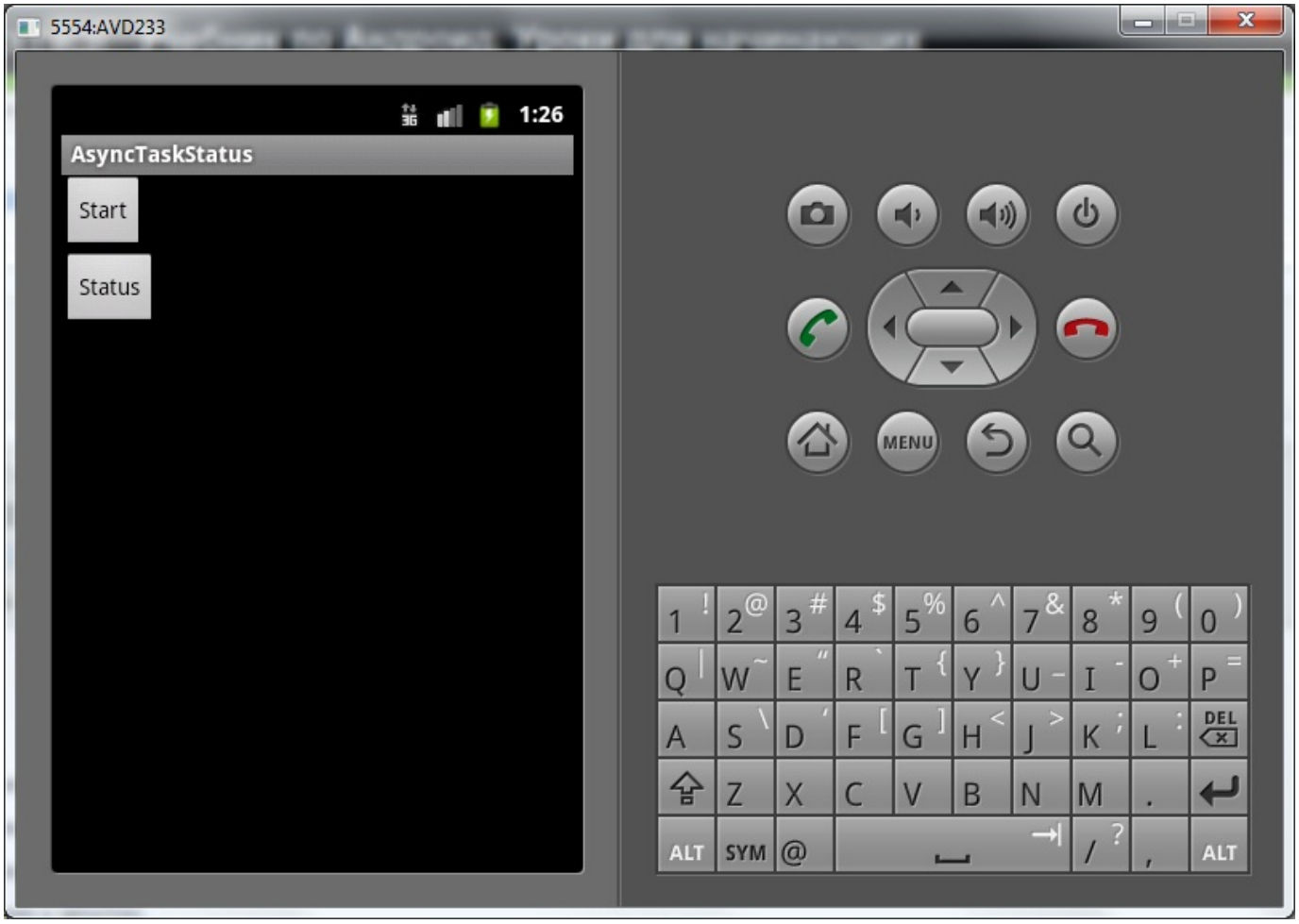
```
    }
  }

  private void startTask() {
    mt = new MyTask();
  }

  private void showStatus() {
    if (mt != null)
      Toast.makeText(this, mt.getStatus().toString(), Toast.LENGTH_SHORT).show();
  }


  class MyTask extends AsyncTask<Void, Void, Void> {
    @Override
    protected void onPreExecute() {
      super.onPreExecute();
      tvInfo.setText("Begin");
    }

    @Override
    protected Void doInBackground(Void... params) {
      try {
        for (int i = 0; i < 5; i++) {
if (isCancelled()) return null;
          TimeUnit.SECONDS.sleep(1);
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      return null;
    }

    @Override
    protected void onPostExecute(Void result) {
      super.onPostExecute(result);
      tvInfo.setText("End");
    }

    @Override
    protected void onCancelled() {
      super.onCancelled();
      tvInfo.setText("Cancel");
    }
  }
}
```

По нажатию кнопки **Start** срабатывает метод startTask. В нем мы просто создаем новую задачу, но пока что не запускаем ее.

По нажатию кнопки **Status** читаем и выводим на экран статус задачи. Используем метод getStatus.

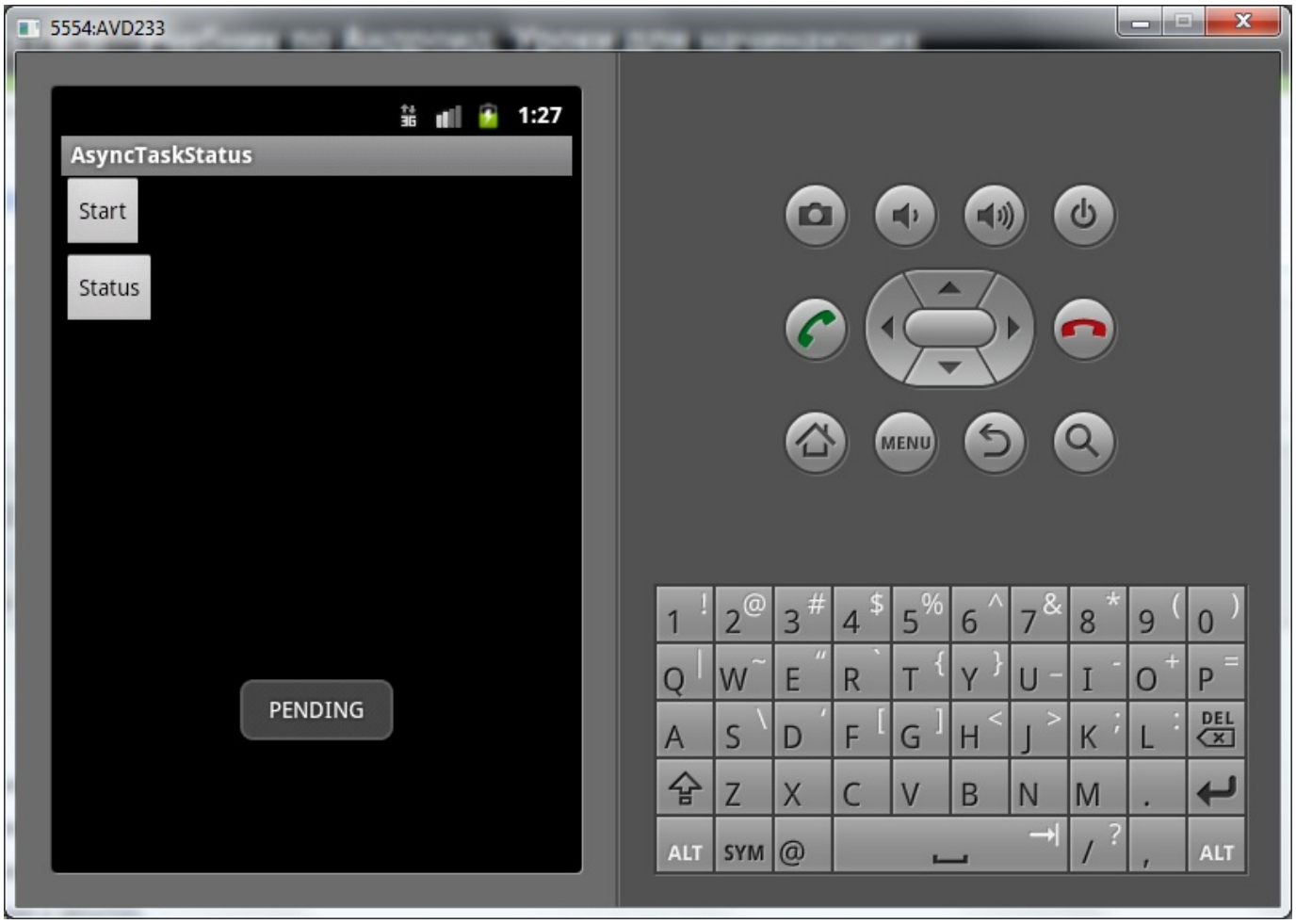
Сама задача проста, гоняем в цикле паузы.

Все сохраним и запустим.



Жмем **Start** – задача создалась.

Жмем **Status**



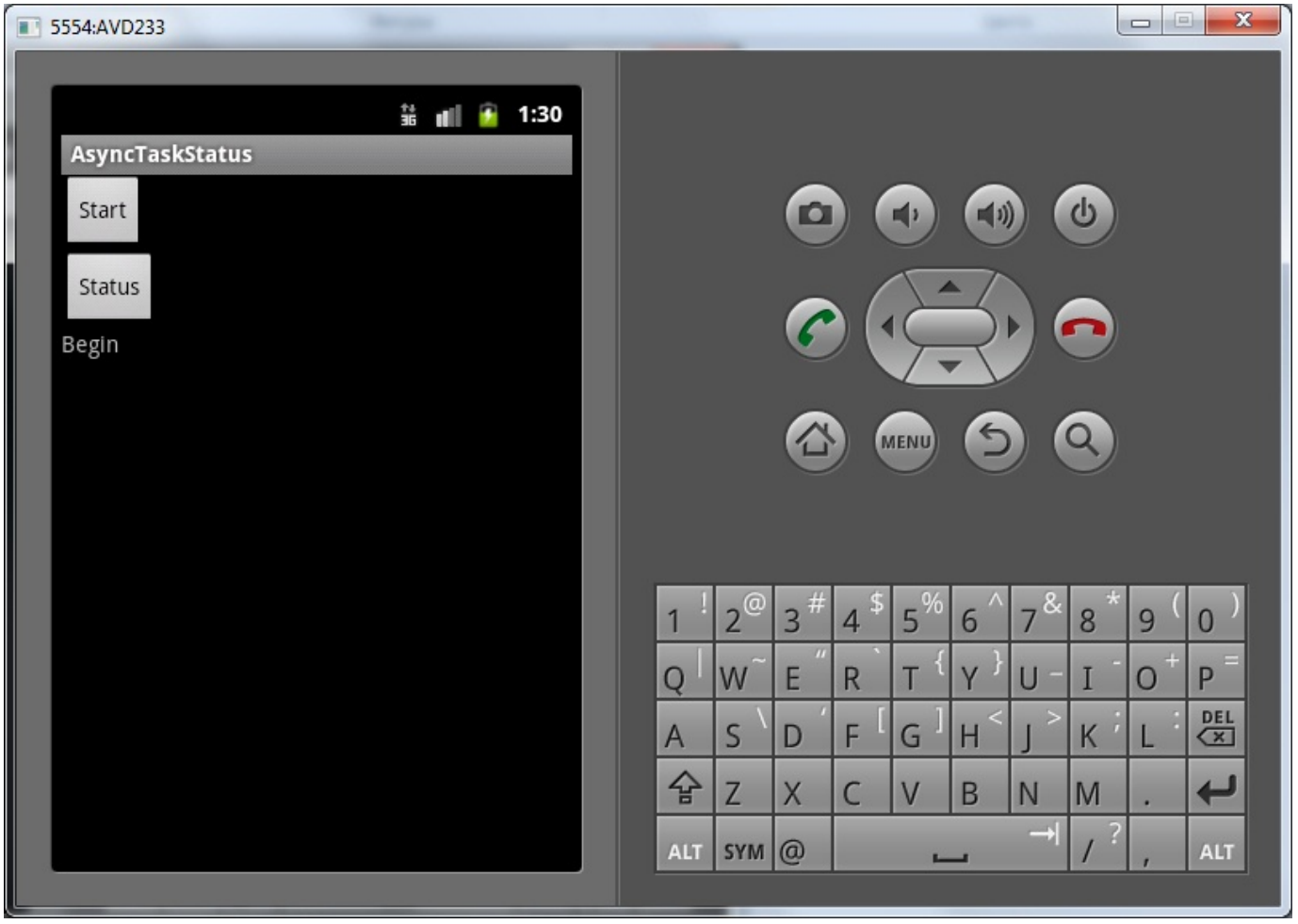
Статус = **PENDING**. Все верно, задача еще не запущена.

Перепишем метод **startTask**:

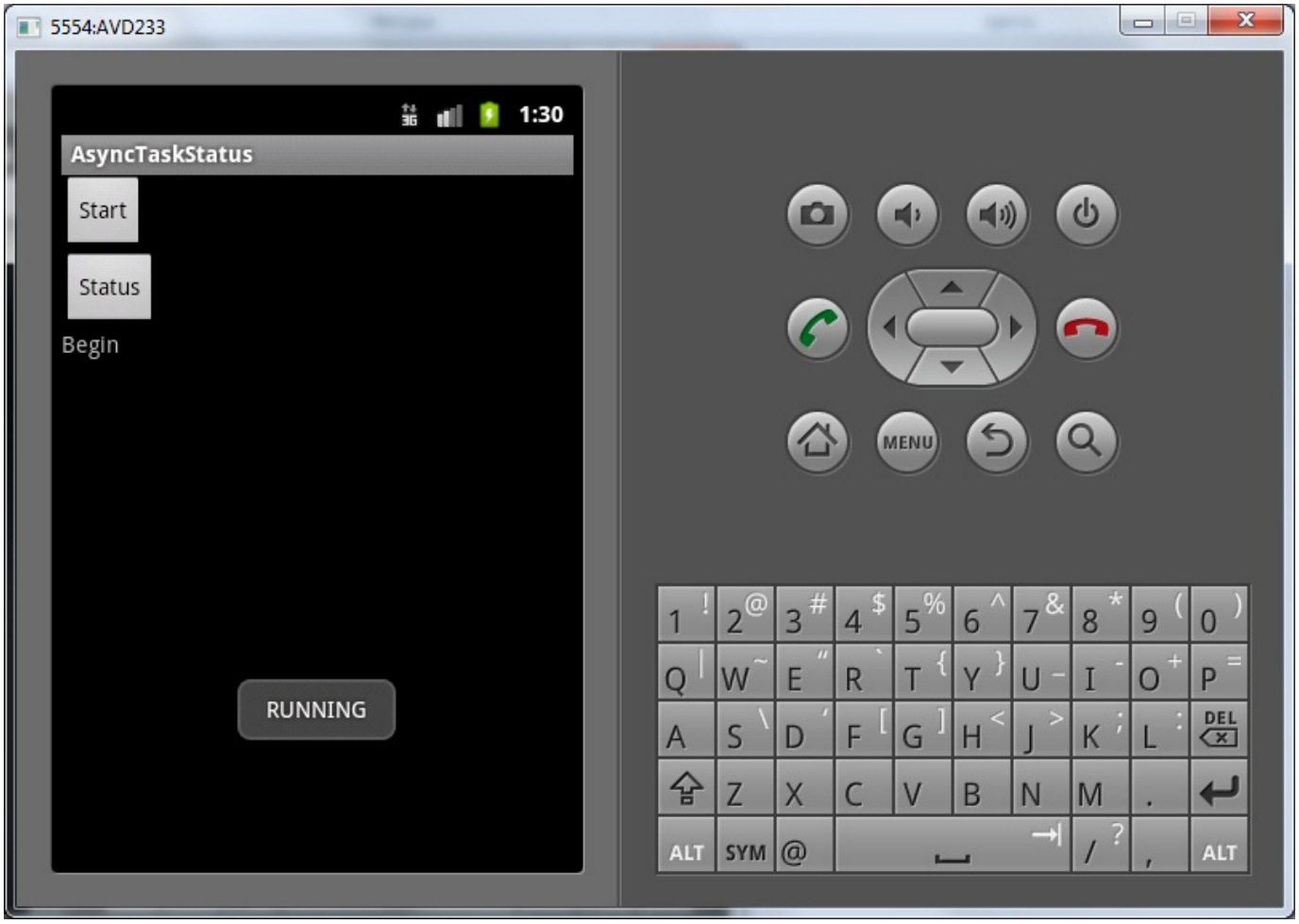
```
private void startTask() {
  mt = new MyTask();
  mt.execute();
}
```

Теперь будем создавать задачу и запускать ее.

Сохраняем, запускаем. Жмем **Start**. Задача начала работать.

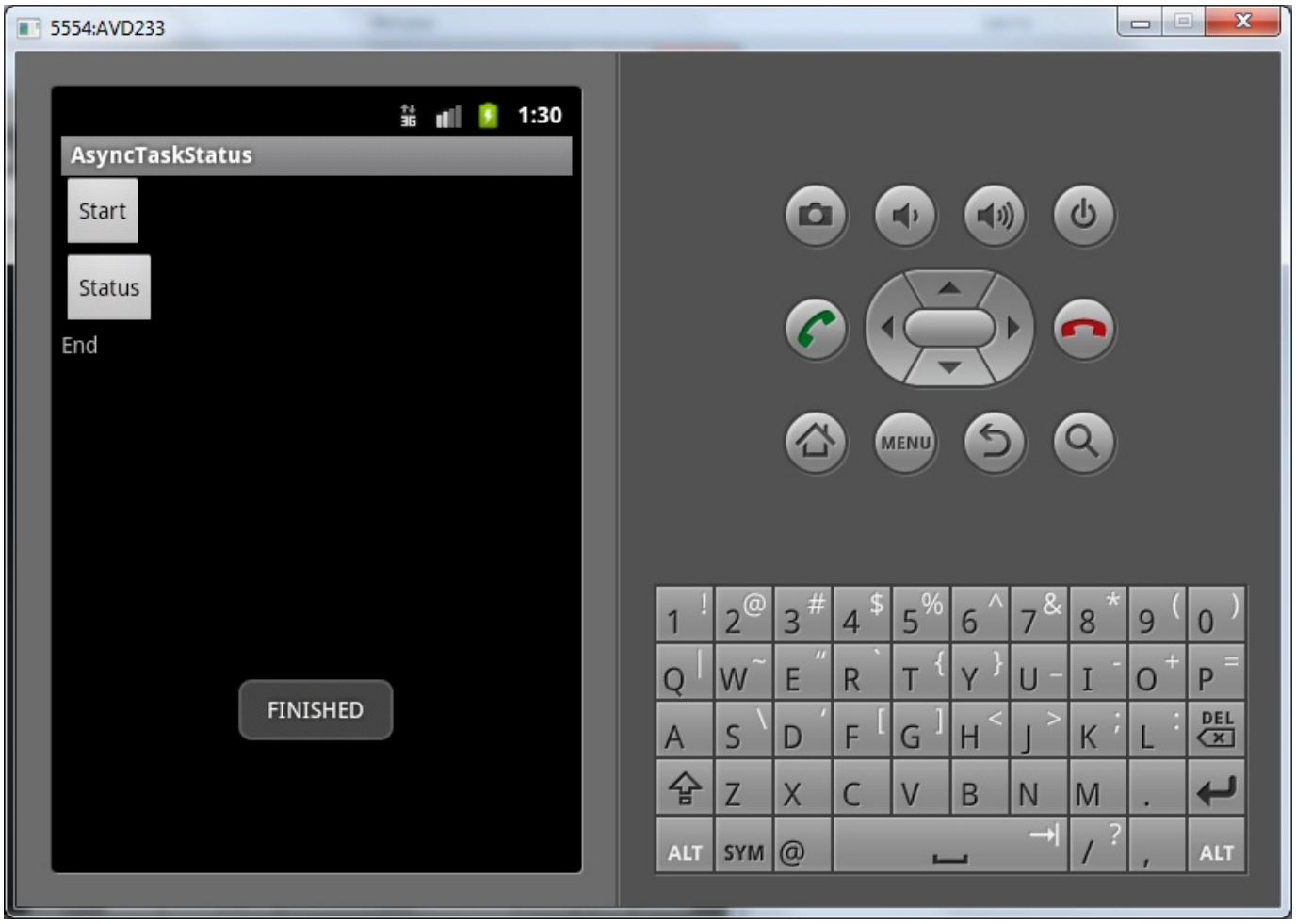


Жмем **Status**, пока задача работает



Статус = **RUNNING**. Задача в работе.

После того, как задача завершилась (End на экране), жмем **Status**



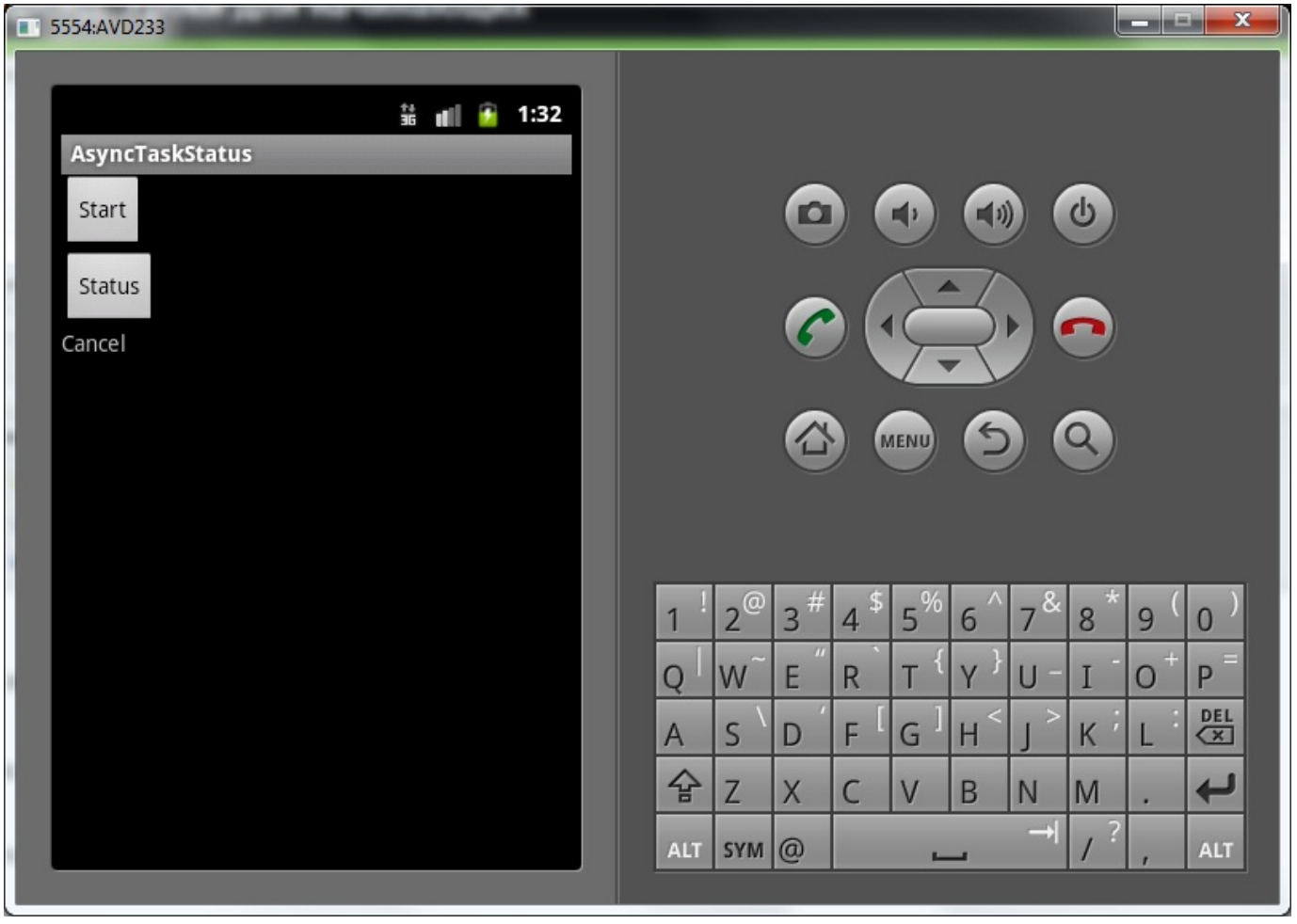
Статус = **FINISHED**. Задача завершена успешно, метод onPostExecute был выполнен.

Теперь посмотрим, какой статус будет если отменить задачу. Перепишем **startTask**:

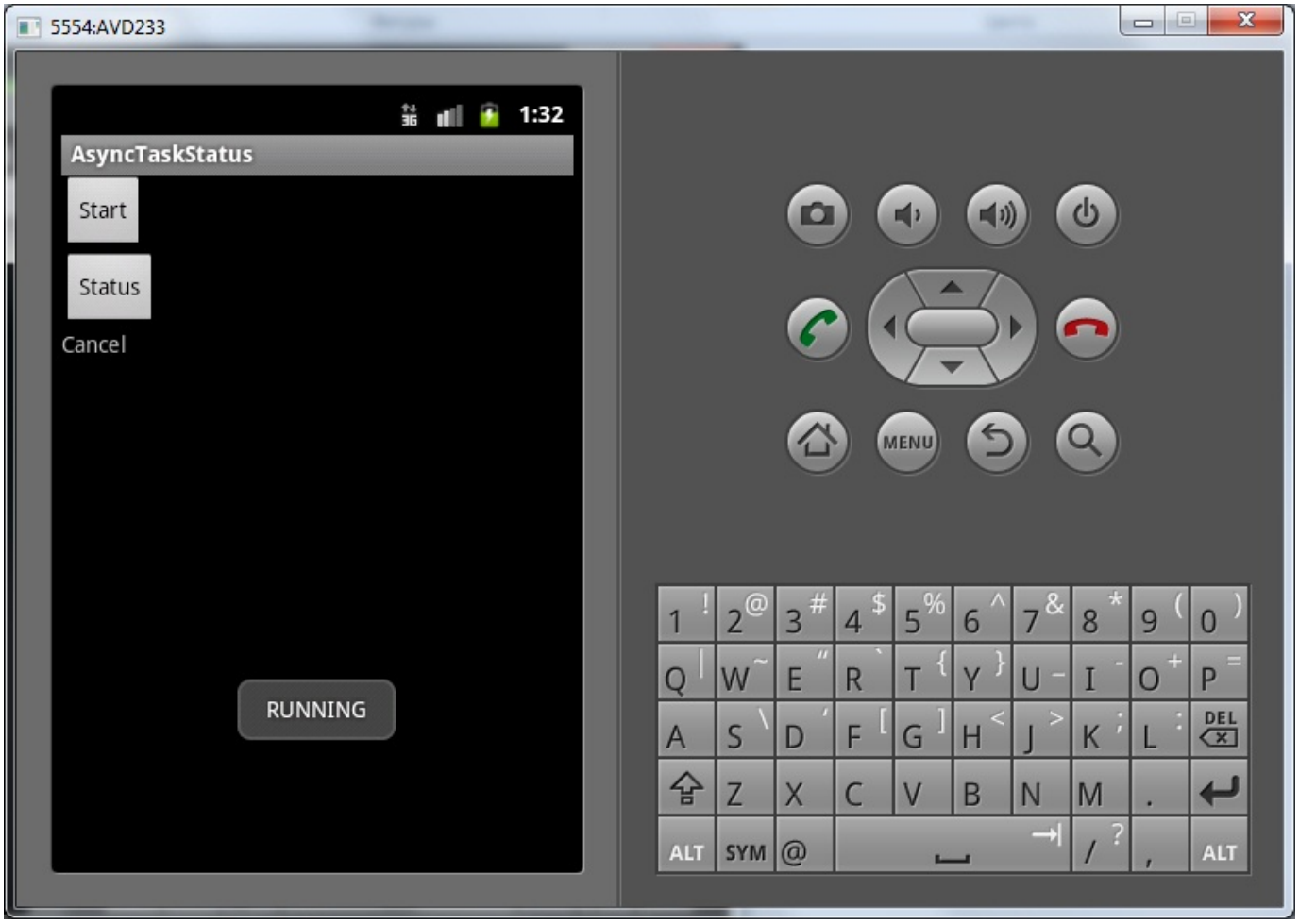
```
private void startTask() {
  mt = new MyTask();
  mt.execute();
  mt.cancel(false);
}
```

Недолго думая отменяем задачу сразу после запуска.

Сохраняем, запускаем приложение. Жмем **Start**.



Задача отменилась. Жмем **Status**

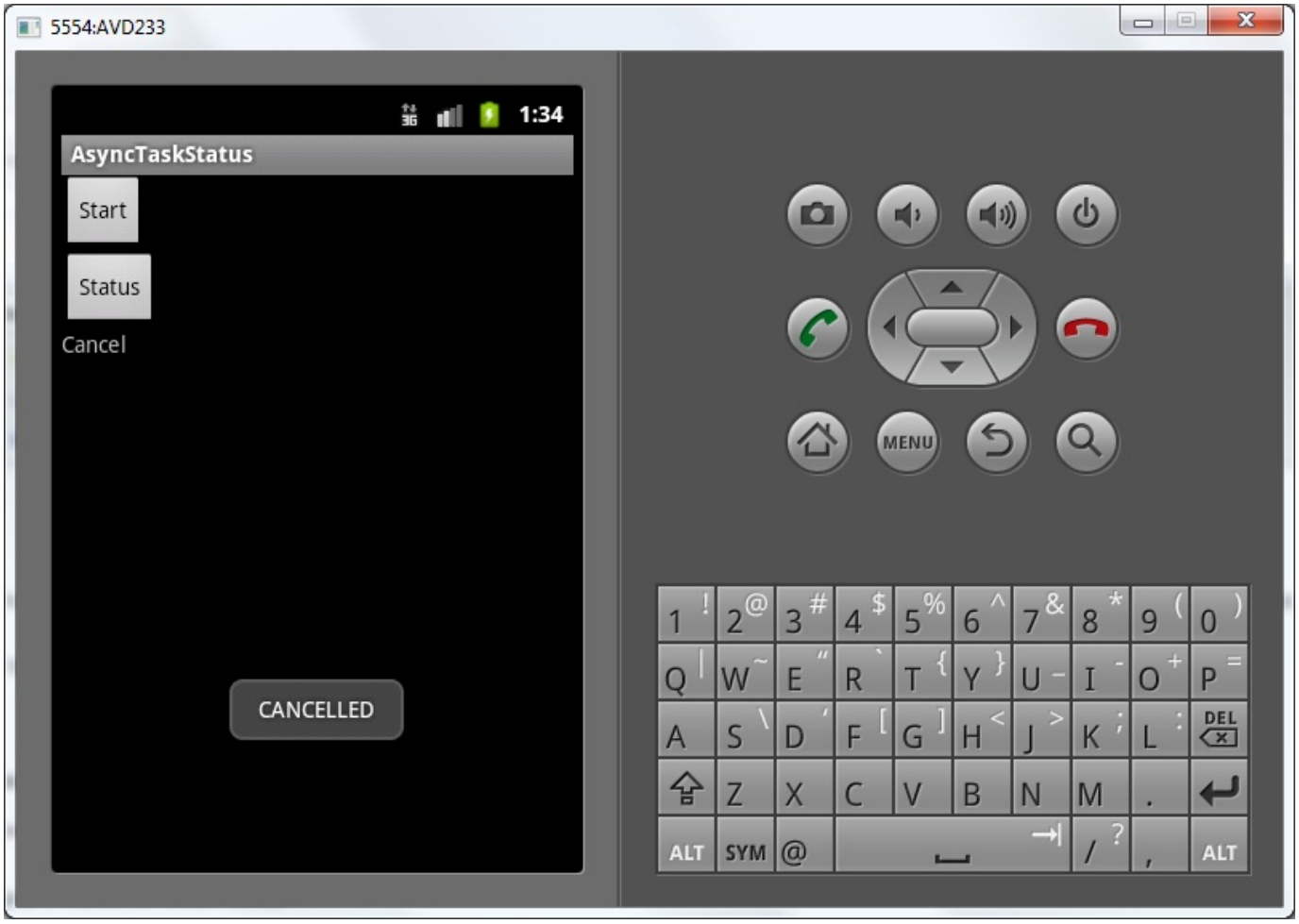


Статус почему-то **RUNNING**, как будто задача в работе. Не особо логично, конечно, но вот такая особенность реализации. Почему нельзя было ввести еще один статус CANCELED – я не знаю. Разработчикам Android виднее.

Как тогда отличить, задача запущена или отменена? Можно использовать метод isCancelled. Перепишем **showStatus** например так:

```
private void showStatus() {
  if (mt != null)
    if (mt.isCancelled())
      Toast.makeText(this, "CANCELLED", Toast.LENGTH_SHORT).show();
    else
      Toast.makeText(this, mt.getStatus().toString(), Toast.LENGTH_SHORT).show();
}
```

Сохраняем, запускаем приложение. Жмем **Start**, задача отменяется. Жмем **Status**



Задача отменена, значит isCancelled вернул true, и мы вывели соответствующее сообщение.

На следующем уроке:

- сохраняем связь с AsyncTask при повороте экрана

Урок 91. AsyncTask. Поворот экрана

В этом уроке:

- сохраняем связь с AsyncTask при повороте экрана

Для полного понимания урока желательно знать, что такое внутренние классы и static объекты.

В прошлых уроках мы в Activity создавали внутренний класс, наследующий AsyncTask. Далее мы по нажатию кнопки создавали экземпляр этого класса и работали с ним. Все бы хорошо ... но, если мы повернем экран, Activity будет создано заново, все прошлые объекты будут потеряны. В том числе мы потеряем и ссылку на наш созданный AsyncTask. А сам AsyncTask будет работать со старым Activity и держать его в памяти, т.к. объект внутреннего класса (AsyncTask) содержит скрытую ссылку на объект внешнего класса (Activity).

Давайте в этом убедимся и разберемся, как это пофиксить.

Т.к. будем работать с поворотом экрана, создавайте проект для Android 2.2 и используйте AVD на базе Android 2.2, потому что 2.3 криво поворачивается.

Создадим проект:

Project name: P0911_AsyncTaskRotate
Build Target: Android 2.2
Application name: AsyncTaskRotate
Package name: ru.startandroid.develop.p0911asynctaskrotate
Create Activity: MainActivity

main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:id="@+id/tv"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="">
     </TextView>
</LinearLayout>
```

MainActivity.java:

```
import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
```

```
public class MainActivity extends Activity {

  MyTask mt;
  TextView tv;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.d("qwe", "create MainActivity: " + this.hashCode());

    tv = (TextView) findViewById(R.id.tv);

    mt = new MyTask();
    Log.d("qwe", "create MyTask: " + mt.hashCode());
    mt.execute();
  }

  class MyTask extends AsyncTask<String, Integer, Void> {

    @Override
    protected Void doInBackground(String... params) {
      try {
        for (int i = 1; i <= 10; i++) {
          TimeUnit.SECONDS.sleep(1);
          publishProgress(i);
          Log.d("qwe", "i = " + i
              + ", MyTask: " + this.hashCode()
              + ", MainActivity: " + MainActivity.this.hashCode());
        }

      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      return null;
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
      super.onProgressUpdate(values);
      tv.setText("i = " + values[0]);
    }
  }
}
```

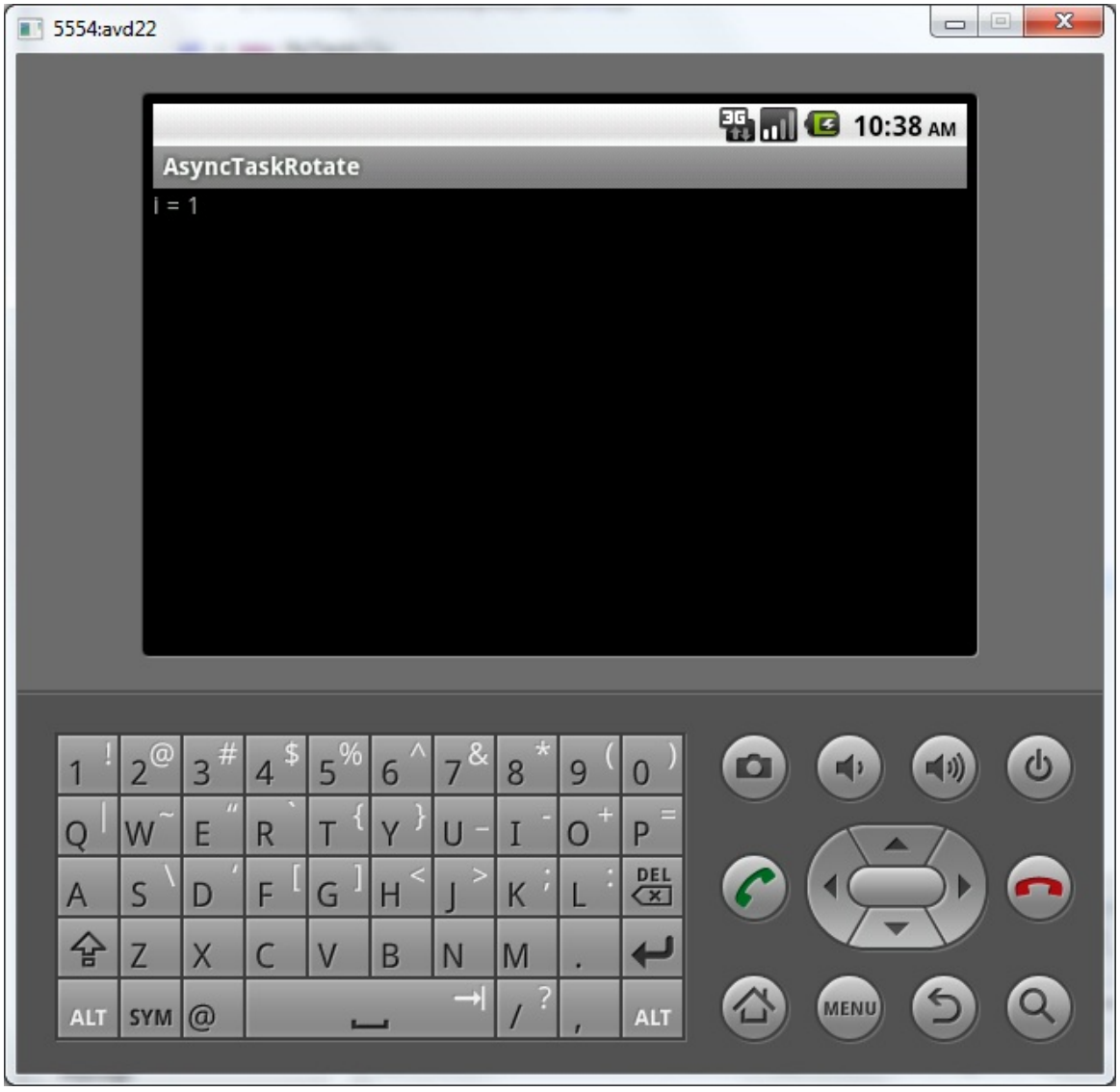
Обычный AsyncTask, который в цикле выполняет паузы (1 сек.) и в TextView на экране пишет номер (i) итерации цикла.

Из необычного здесь можно отметить то, что мы используем метод hashCode. Этот метод возвращает хэш-код объекта. Сейчас не будем вникать что это и зачем нужно. Главное, надо знать, что разным объектам соответствует разный хэш-код. Т.е. по хэш-коду мы будем отличать объекты друг от друга (можно воспринимать хэш-код, как некий ID).

Мы при создании MainActivity и MyTask пишем в лог их хэш-коды. Затем при выполнении задачи, мы также будем писать в лог хэш-коды этих объектов. Сейчас станет понятно, зачем это нужно.

Все сохраним и запускаем приложение. Оно сразу запускает задачу, которая раз в секунду выдает на экран номер итерации цикла. Дождемся пока на экране появится, например, 5

и повернем экран (CTRL+F12 или CTRL+F11)



Отсчет снова пошел с единицы.

Дождемся конца отсчета и посмотрим в логи. Буду выдавать их частями и каментить:

create MainActivity: 1156854488
create MyTask: 1156875480

Создались объекты и мы видим их хэш-коды.

Далее начинает работу MyTask

i = 1, MyTask: 1156875480, MainActivity: 1156854488
i = 2, MyTask: 1156875480, MainActivity: 1156854488
i = 3, MyTask: 1156875480, MainActivity: 1156854488
i = 4, MyTask: 1156875480, MainActivity: 1156854488
i = 5, MyTask: 1156875480, MainActivity: 1156854488

Выводит в лог номер итерации и хэш-коды – свой и MainActivity, с которым он работает. Хэш-коды совпадают с теми, что ранее вывелись в лог при создании. Тут все ясно.

Теперь мы поворачиваем экран.

create MainActivity: 1156904328

create MyTask: 1156916144

Создается новое MainActivity и в нем создается новый MyTask. Их хэш-коды (1156904328 и 1156916144) отличаются от хэш-кодов старых MainActivity и MyTask (1156854488 и 1156875480). Т.е. это совершенно другие, новые объекты.

i = 6, MyTask: 1156875480, MainActivity: 1156854488
i = 7, MyTask: 1156875480, MainActivity: 1156854488
i = 1, MyTask: 1156916144, MainActivity: 1156904328
i = 8, MyTask: 1156875480, MainActivity: 1156854488
i = 2, MyTask: 1156916144, MainActivity: 1156904328
i = 9, MyTask: 1156875480, MainActivity: 1156854488
i = 3, MyTask: 1156916144, MainActivity: 1156904328
i = 10, MyTask: 1156875480, MainActivity: 1156854488
i = 4, MyTask: 1156916144, MainActivity: 1156904328
i = 5, MyTask: 1156916144, MainActivity: 1156904328
i = 6, MyTask: 1156916144, MainActivity: 1156904328
i = 7, MyTask: 1156916144, MainActivity: 1156904328
i = 8, MyTask: 1156916144, MainActivity: 1156904328
i = 9, MyTask: 1156916144, MainActivity: 1156904328
i = 10, MyTask: 1156916144, MainActivity: 1156904328

Мы видим, как продолжает работать старый MyTask (1156875480), и работает он со старым MainActivity (1156854488), продолжая отсчет от 6 до 10.

А параллельно с ним работает новый MyTask (1156916144) с новым MainActivity (1156904328), он начал с 1. На экране мы видим именно работу этих новых объектов. Поэтому цифры в TextView снова пошли с единицы. А старые объекты продолжают существовать где-то в памяти и работать. Но главное то, что мы потеряли связь со старым MyTask, создалась новая задача и работа пошла сначала.

Каждый раз начинать задачу заново при повороте экрана – это получится кривое приложение. Будем фиксить. Нам надо при создании нового Activity как-то получать ссылку на старый MyTask и не создавать новый, чтобы не начинать работу с начала, а продолжать ее. В этом нам помогут методы onRetainNonConfigurationInstance и getLastNonConfigurationInstance. О них можно прочесть в уроке 70.

Добавим в класс MainActivity реализацию метода **onRetainNonConfigurationInstance**:

```
public Object onRetainNonConfigurationInstance() {
  return mt;
}
```

При повороте экрана, система сохранит для нас ссылку на объект mt.

И перепишем **onCreate**:

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  Log.d("qwe", "create MainActivity: " + this.hashCode());

  tv = (TextView) findViewById(R.id.tv);

  mt = (MyTask) getLastNonConfigurationInstance();
  if (mt == null) {
```

```
            mt = new MyTask();
            mt.execute();
        }
        Log.d("qwe", "create MyTask: " + mt.hashCode());
    }
```

При создании Activity мы просим систему вернуть (getLastNonConfigurationInstance) нам сохраненный в методе onRetainNonConfigurationInstance объект и приводим его к MyTask. Если Activity создается не после поворота экрана, то мы получим null, а значит, создаем сами MyTask.

Таким образом, при повороте экрана мы возвращаем себе старый MyTask. Посмотрим, что получится.

Верните эмулятор в вертикальную ориентацию. Сохраняем и запускаем приложение.

Ждем до 5

и поворачиваем экран:

А на экране ничего не происходит, хотя логи продолжают идти. Давайте посмотрим, что в логах:

create MainActivity: 1156854504
create MyTask: 1156875408
i = 1, MyTask: 1156875408, MainActivity: 1156854504
i = 2, MyTask: 1156875408, MainActivity: 1156854504
i = 3, MyTask: 1156875408, MainActivity: 1156854504
i = 4, MyTask: 1156875408, MainActivity: 1156854504
i = 5, MyTask: 1156875408, MainActivity: 1156854504

Тут все понятно, создались объекты, начала работать задача

create MainActivity: 1156904256
create MyTask: 1156875408

Создается новое MainActivity с новым хэш-кодом (1156904256). А вот MyTask мы добыли старый (хэш-код тот же - 1156875408), у нас получилось вернуть доступ к старому MyTask и не создавать новый. А значит работа продолжится и не будет начинаться заново. Это хорошо. Но есть и плохая новость.

i = 6, MyTask: 1156875408, MainActivity: 1156854504
i = 7, MyTask: 1156875408, MainActivity: 1156854504
i = 8, MyTask: 1156875408, MainActivity: 1156854504
i = 9, MyTask: 1156875408, MainActivity: 1156854504
i = 10, MyTask: 1156875408, MainActivity: 1156854504

Старый MyTask продолжает работать со старым MainActivity (1156854504), а новое (1156904256) в упор не видит.

Так происходит, потому что объект внутреннего класса (MyTask) содержит **скрытую ссылку** на объект внешнего класса (MainActivity). Обратите внимание, что мы в методах MyTask работаем с объектом tv. А ведь такого объекта в MyTask нет, он есть только в MainActivity. Тут используется скрытая ссылка – это позволяет MyTask работать с объектами MainActivity.

Поэтому наш старый MyTask связан со своим объектом внешнего класса MainActivity и видит только его. И меняет текст в TextView старого MainActivity, которое висит где-то в памяти. А на экране мы видим новое MainActivity. И оно не меняется.

То, что MyTask содержит сслыку на старое MainActivity плохо еще тем, что MainActivity не может быть уничтожено и висит в памяти.

Значит, нам надо избавиться от связки MainActivity и MyTask. Для этого применим **static** к внутреннему классу MyTask. Внутренний static класс никак **не связан** с объектом внешнего класса и не содержит скрытую ссылку на него. Но нам надо получить доступ к объектам (tv) MainActivity. Если не будет ссылки, не будет и доступа. Значит, сами создадим такую ссылку. В MyTask опишем объект, он и будет ссылаться на MainActivity. А мы будем этой ссылкой управлять – когда создается новое MainActivity, мы будем давать ссылку на него в MyTask.

Перепишем **MainActivity.java**:

```
package ru.startandroid.develop.p0911asynctaskrotate;

import java.util.concurrent.TimeUnit;

import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class MainActivity extends Activity {

  MyTask mt;
  TextView tv;

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Log.d("qwe", "create MainActivity: " + this.hashCode());

    tv = (TextView) findViewById(R.id.tv);

    mt = (MyTask) getLastNonConfigurationInstance();
    if (mt == null) {
      mt = new MyTask();
      mt.execute();
    }
    // передаем в MyTask ссылку на текущее MainActivity
    mt.link(this);

    Log.d("qwe", "create MyTask: " + mt.hashCode());
  }

  public Object onRetainNonConfigurationInstance() {
    // удаляем из MyTask ссылку на старое MainActivity
    mt.unLink();
    return mt;
```

```
  }


  static class MyTask extends AsyncTask<String, Integer, Void> {

    MainActivity activity;

    // получаем ссылку на MainActivity
    void link(MainActivity act) {
      activity = act;
    }

    // обнуляем ссылку
    void unLink() {
      activity = null;
    }

    @Override
    protected Void doInBackground(String... params) {
      try {
        for (int i = 1; i <= 10; i++) {
          TimeUnit.SECONDS.sleep(1);
          publishProgress(i);
          Log.d("qwe", "i = " + i + ", MyTask: " + this.hashCode()
              + ", MainActivity: " + activity.hashCode());
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      return null;
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
      super.onProgressUpdate(values);
      activity.tv.setText("i = " + values[0]);
    }
  }
}
```

Мы добавили **static** к описанию класса **MyTask**. Также описали в нем объект **activity** класса MainActivity и два метода:

link – с его помощью MyTask будет получать ссылку на MainActivity, с которой будет работать

unlink – обнуление ссылки

И теперь в классе MyTask мы уже не можем просто так работать с объектами MainActivity, т.к. MyTask у нас static, и не содержит скрытой ссылки на MainActivity. Мы должны явно указывать, что обращаемся к MainActivity, например *activity.tv*.

В методе **onRetainNonConfigurationInstance** перед тем, как сохранить MyTask для передачи новому Activity, мы обнуляем ссылку на старое MainActivity. MyTask больше не будет держать старое MainActivity и система сможет его (MainActivity) уничтожить.

А в **onCreate** мы после создания/получения объекта MyTask вызываем метод link и передаем туда ссылку на текущее новое MainActivity. С ним и продолжит работу MyTask.

Давайте проверим. Верните эмулятор в вертикальную ориентацию. Все сохраним и запустим приложение. Пошел отсчет. Дожидаемся 5

и поворачиваем экран

Отсчет продолжился, что и требовалось получить. Смотрим логи:

create MainActivity: 1156967624
create MyTask: 1156978504
i = 1, MyTask: 1156978504, MainActivity: 1156967624
i = 2, MyTask: 1156978504, MainActivity: 1156967624
i = 3, MyTask: 1156978504, MainActivity: 1156967624
i = 4, MyTask: 1156978504, MainActivity: 1156967624
i = 5, MyTask: 1156978504, MainActivity: 1156967624

Объекты создались, работа пошла

Поворот экрана

create MainActivity: 1156991528
create MyTask: 1156978504

MainActivity новое (1156991528), MyTask старый (1156978504).

i = 6, MyTask: 1156978504, MainActivity: 1156991528
i = 7, MyTask: 1156978504, MainActivity: 1156991528

i = 8, MyTask: 1156978504, MainActivity: 1156991528
i = 9, MyTask: 1156978504, MainActivity: 1156991528
i = 10, MyTask: 1156978504, MainActivity: 1156991528

Старый MyTask получил ссылку на новый MainActivity и продолжает работу уже с ним. А старое MainActivity кануло в небытие.

Уф! Хотел поверхностно показать механизм, но полез в объяснения «что да как» и получился достаточно непростой для понимания урок. Если остались непонятные моменты – велкам в форум, в ветку этого урока. Будем разбираться :)

Есть еще один способ (кроме static) избежать связки AsyncTask и Activity - просто сделайте ваш класс, наследующий AsyncTask, не внутренним, а отдельно от MainActivity.

Просьба к вам - откройте снова урок 86 и прочтите 4 правила использования AsyncTask. Я думаю, теперь они будут для вас гораздо информативнее, чем при первом прочтении.

P.S. В форуме верно заметили, что есть небольшой изъян в этом всем. Будет плохо, если onProgressUpdate выполнится между моментом, когда старое Activity выполнит метод unLink, и моментом, когда новое Activity выполнит метод link. В этом случае у нас activity будет равен null и мы получим NullPointerException. Вероятность это всего, конечно, мала, но решать проблему как-то надо.

Напишу здесь свой вариант решения. В методе onProgressUpdate мы ставим проверку *activity == null*. Если activity - не null, то без проблем меняем textView. Если же activity - null, то текст, который мы хотели прописать в TextView, мы сохраняем в какую-нить свою переменную класса MyTask. А новое Activity, когда получает MyTask, достает данные из этой переменной и сама помещает их в TextView.

Ваши предложения по решению проблемы пишите на в ветке этого урока.

На следующем уроке:

- создаем, запускаем и останавливаем простой сервис

Урок 92. Service. Простой пример

В этом уроке:

- создаем, запускаем и останавливаем простой сервис

Для начала надо определиться, как по-русски называть **Service**. Общепринятый перевод – служба. Наиболее популярный пример – службы в Windows. Но для системы Android мне привычнее использовать слово **сервис**. Его я и буду использовать в своих уроках для обозначения Service.

И еще один момент надо сразу прояснить. Service прописывается в манифесте в рядом с Activity, и получается, что приложение (Application) содержит в себе и Activity и сервис. Предлагаю для упрощения изложения материала под словом приложение понимать все таки только Activity, которые можно запустить и увидеть на экране. Т.е. то, что мы раньше и называли приложением. А сервис считать отдельной от приложения вещью. А если надо будет обозначить приложение, как контейнер для Activity и сервиса, буду использовать слово Application.

Т.е. приложение – это набор Activity, сервис – Service. Приложение + сервис = Application. Как то так.

Итак, сервис – это некая задача, которая работает в фоне и не использует UI. Запускать и останавливать сервис можно из приложений и других сервисов. Также можно подключиться к уже работающему сервису и взаимодействовать с ним.

В качестве примера можно рассмотреть алгоритм почтовой программы. Она состоит из приложения и сервиса. Сервис работает в фоне и периодически проверяет наличие новой почты, скачивает ее и выводит уведомления. А когда вы запускаете приложение, оно отображает вам эти загруженные сервисом письма. Также приложение может подключиться к сервису и поменять в нем, например, период проверки почты или совсем закрыть сервис, если постоянная проверка почты больше не нужна.

Т.е. сервис нужен, чтобы ваша задача продолжала работать, даже когда приложение закрыто. В ближайших уроках мы разберемся, какие способы взаимодействия существуют между приложением и сервисом. В этом же уроке создадим простейший сервис, который будет выводить что-нибудь в лог. А приложение будет запускать и останавливать сервис.

Создадим проект:

Project name: P0921_ServiceSimple
Build Target: Android 2.3.3
Application name: ServiceSimple
Package name: ru.startandroid.develop.p0921servicesimple
Create Activity: MainActivity

Добавим в **strings.xml** строки:

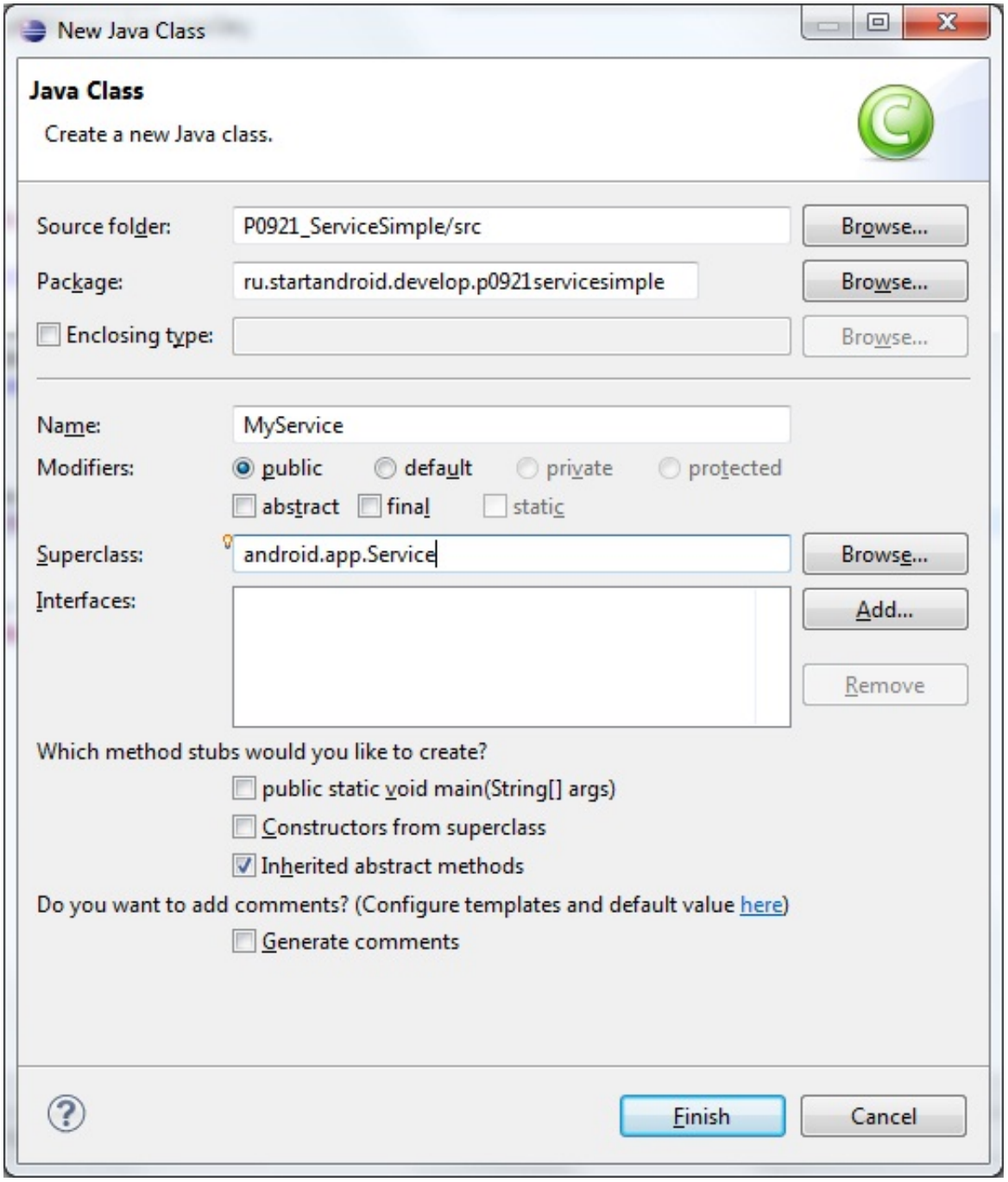
```
    <string  name="start">Start service</string>
    <string  name="stop">Stop service</string>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
           android:id="@+id/btnStart"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onClickStart"
           android:text="@string/start">
     </Button>
     <Button
           android:id="@+id/btnStop"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:onClick="onClickStop"
           android:text="@string/stop">
     </Button>
     <ProgressBar
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:indeterminate="true">
     </ProgressBar>
</LinearLayout>
```

Две кнопки – для запуска и остановки сервиса. И ProgressBar.

Рядом с **MainActivity** создайте класс **MyService**, наследующий **android.app.Service**



Содержимое **MyService.java**:

```
package ru.startandroid.develop.p0921servicesimple;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {

  final String LOG_TAG = "myLogs";

  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "onCreate");
  }
```

```
  public int onStartCommand(Intent intent, int flags, int startId) {
    Log.d(LOG_TAG, "onStartCommand");
    someTask();
    return super.onStartCommand(intent, flags, startId);
  }

  public void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "onDestroy");
  }

  public IBinder onBind(Intent intent) {
    Log.d(LOG_TAG, "onBind");
    return null;
  }

  void someTask() {
  }
}
```

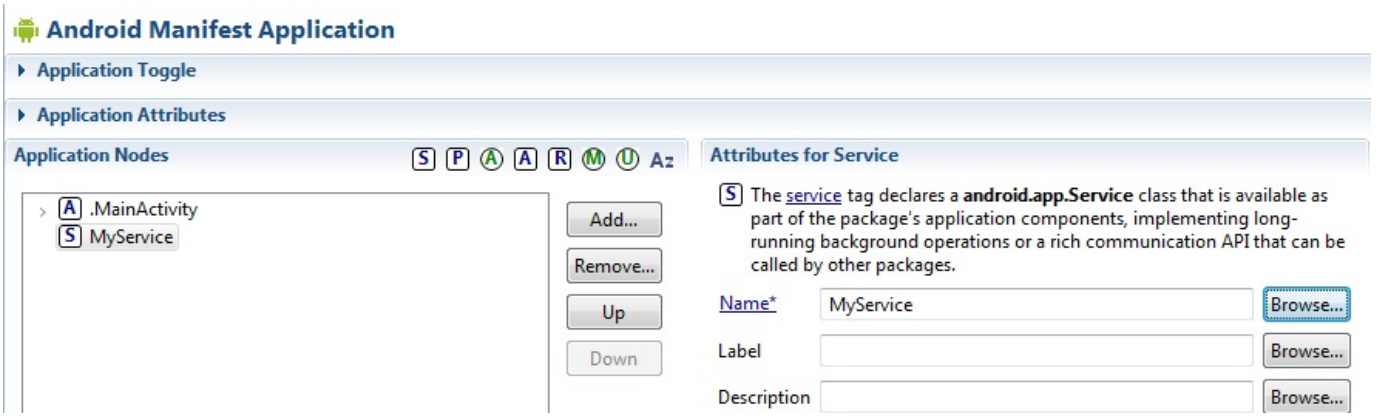
У сервиса так же, как и у Activity есть методы **onCreate** и **onDestroy** – которые срабатывают при создании и уничтожении сервиса.

Метод onStartCommand – срабатывает, когда сервис запущен методом startService. В нем мы запускаем наш метод **someTask**, который пока пуст. У **onStartCommand** на вход и на выход идут параметры, мы их пока не используем.

Метод **onBind** нам пока не интересен. Но реализовать его мы обязаны, возвращаем null.

someTask – здесь будем кодить работу для сервиса

Сервис, как и новые, создаваемые Activity необходимо прописать в манифесте. Делается это полностью аналогично. Жмете **Add**, выбираете **Service**. И в поле **Name** выбираете **MyService**.



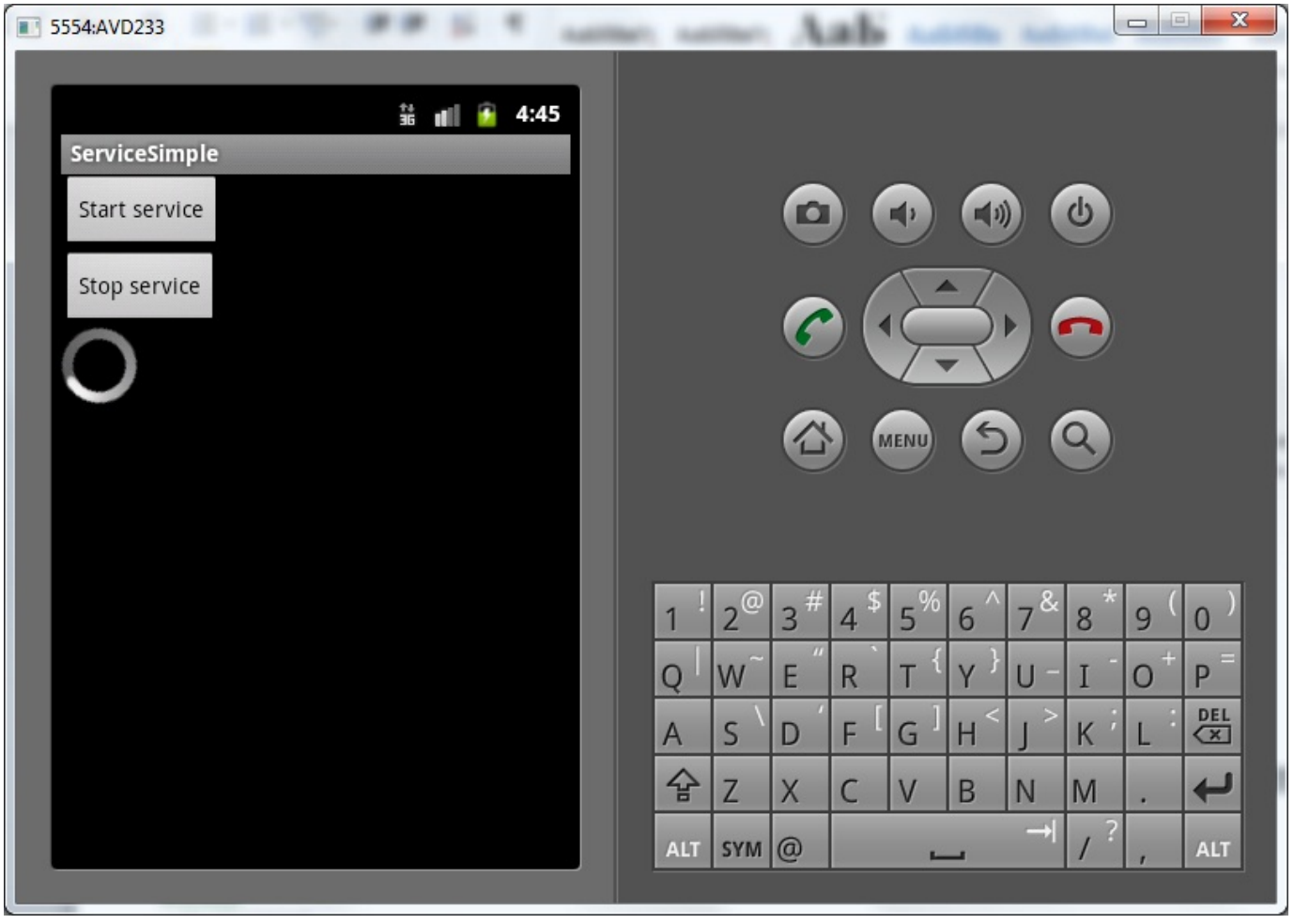
MainActivity.java:

```
package ru.startandroid.develop.p0921servicesimple;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

    }

    public void onClickStart(View v) {
      startService(new Intent(this, MyService.class));
    }

    public void onClickStop(View v) {
      stopService(new Intent(this, MyService.class));
    }
}
```

Здесь у нас два метода, которые срабатывают при нажатии на кнопки **Start service** и **Stop service**. В них мы соответственно запускаем или останавливаем сервис методами **startService** и stopService. На вход передаем Intent, указывающий на сервис. Это очень похоже на то, как мы вызываем Activity методом startActivity.

Давайте все сохраним и запустим приложение.



Нажмем **Start service** и смотрим лог:

onCreate
onStartCommand

Выполнился метод **onCreate** – сервис создался, и **onStartCommand** – сервис выполняет содержимое метода onStartCommand.

Если мы теперь еще раз нажмем **Start service**:

onStartCommand

Сервис уже создан, **onCreate** не вызывается, выполняется только метод **onStartCommand**.

Жмем **Stop service**

onDestroy

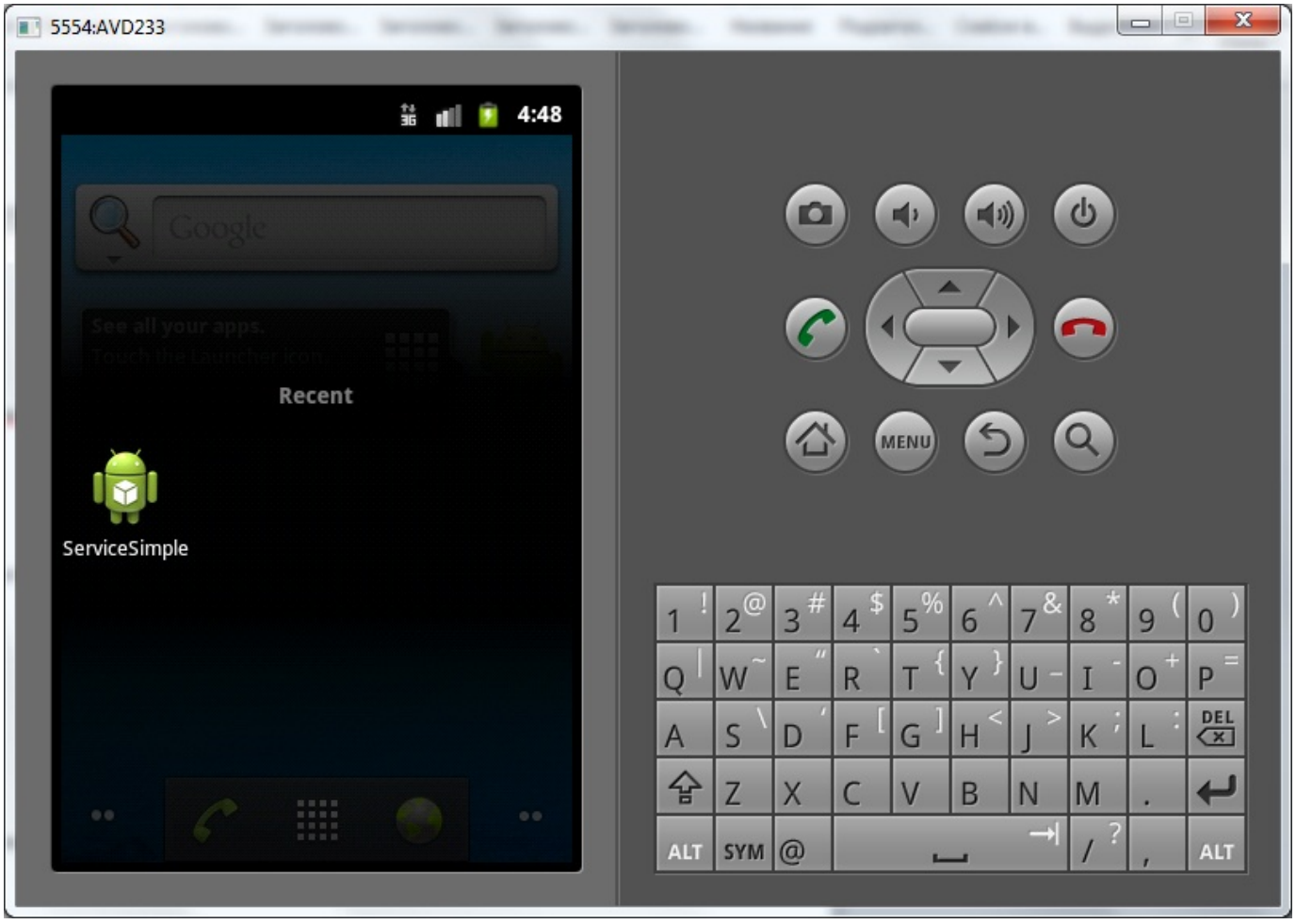
Сервис уничтожился.

Убедимся, что сервис не зависит от приложения. Жмем **Start service**.

onCreate
onStartCommand

Сервис запущен. Закрываем приложение кнопкой Назад. В логах тишина, **onDestroy** в сервисе не выполнился, сервис продолжает жить. Ему все равно, работает приложение, его запустившее или не работает.

Долгим удержанием клавиши Домой выведем список последних приложений



снова откроем наше приложение и нажмем **Stop service**. В логах:

onDestroy

Сервис уничтожен.

Теперь попробуем выполнять что-нибудь осмысленное в **onStartCommand**. Перепишем метод **someTask** в MyService.java:

```
void someTask() {
  for (int i = 1; i<=5; i++) {
    Log.d(LOG_TAG, "i = " + i);
    try {
      TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

Будем с интервалом в 1 секунду выводить в лог записи из сервиса.

Все сохраним, запустим и нажмем **Start service**:

ProgressBar замер, экран стал недоступен. А логи идут:

onCreate
onStartCommand
i = 1
i = 2
i = 3
i = 4
i = 5

Вывод – сервис работает в основном потоке и блокирует экран.

Вынесем цикл с паузами в отдельный поток. И чтобы два раза не бегать, давайте заодно узнаем, что делает метод stopSelf.

Перепишем метод **someTask**:

```
void someTask() {
  new Thread(new Runnable() {
    public void run() {
      for (int i = 1; i<=5; i++) {
        Log.d(LOG_TAG, "i = " + i);
        try {
          TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
      stopSelf();
    }
  }).start();
}
```

Мы вынесли код в отдельный поток и добавили вызов метода **stopSelf** – этот метод аналогичен методу **stopService**, он останавливает сервис, в котором был вызван.

Все сохраняем, запускаем и жмем **Start service**. ProgressBar крутится, экран не блокирован, логи идут:

onCreate
onStartCommand
i = 1
i = 2
i = 3
i = 4
i = 5
onDestroy

Сервис создался, выполнил работу и сам остановился методом **stopSelf**.

На следующем уроке:

- передаем данные в сервис
- рассматриваем методы остановки сервиса stopSelf и stopSelfResult

Урок 93. Service. Передача данных в сервис. Методы остановки сервиса

В этом уроке:

- передаем данные в сервис
- рассматриваем методы остановки сервиса stopSelf и stopSelfResult

В прошлом уроке мы использовали в коде метод stopSelf. Он вызывается внутри сервиса и останавливает этот сервис. У этого метода есть еще пара вариаций: stopSelf(int startId) и stopSelfResult(int startId). Как видим, на вход они требуют некий **startId**. Что это и откуда его взять? Для ответа на вопрос давайте взглянем на входящие параметры метода onStartCommand:

Первый – это **Intent**. Тот самый, который отправляется в путь, когда мы стартуем сервис с помощью метода startService. Соответственно вы можете использовать его для передачи данных в ваш сервис. Тут все аналогично, как при вызове другого Activity – там вы тоже можете передать данные с помощью intent.

Второй параметр – **флаги** запуска. Он нам пока не нужен, пропускаем его.

Третий параметр – **startId**. Простыми словами – это **счетчик** вызовов startService пока сервис запущен. Т.е. вы запустили сервис методом startService, сработал метод onStartCommand и получил на вход startId = 1. Вызываем еще раз метод startService, сработал метод onStartCommand и получил на вход startId = 2. И так далее. Счетчик сбрасывается, когда сервис будет остановлен методами stopService, stopSelf и пр. После этого вызовы снова идут с единицы.

Именно этот **startId** и нужен на вход методу stopSelf(startId). Т.е. этот метод дает системе понять, что конкретный вызов сервиса был успешно **обработан**. И мне кажется логичным, что, когда все вызовы сервиса выполнили метод stopSelf(startId) и система видит, что не осталось необработанных вызовов – сервис можно останавливать. Т.е. поступила куча вызовов, они все обработались и когда последний обработанный (но при этом не обязательно последний поступивший) выполняет stopSelf(startId) – сервис останавливается.

Но на самом деле алгоритм чуть другой. Сервис останавливается, когда последний полученный (а не последний обработанный) вызов выполняет метод stopSelf(startId). А при этом могут продолжать работать ранее полученные вызовы. Почему так сделано – я не знаю.

В общем, наверно сумбурно объяснил, сейчас на примерах все станет понятнее.

А метод **stopSelfResult(startId)** аналогичен методу **stopSelf(startId)**, но при этом еще возвращает boolean значение – остановил он сервис или нет.

Создадим проект:

Project name: P0931_ServiceStop
Build Target: Android 2.3.3
Application name: ServiceStop
Package name: ru.startandroid.develop.p0931servicestop
Create Activity: MainActivity

Добавим в **strings.xml** строки:

```
    <string  name="start">Start</string>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnStart"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:onClick="onClickStart"
         android:text="@string/start">
    </Button>
</LinearLayout>
```

Только кнопка для запуска сервиса.

Создаем сервис **MyService.java**:

```
package ru.startandroid.develop.p0931servicestop;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {

  final String LOG_TAG = "myLogs";
  ExecutorService es;
  Object someRes;

  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "MyService onCreate");
    es = Executors.newFixedThreadPool(1);
    someRes = new Object();
  }

  public void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "MyService onDestroy");
    someRes = null;
  }

  public int onStartCommand(Intent intent, int flags, int startId) {
    Log.d(LOG_TAG, "MyService onStartCommand");
```

```
    int time = intent.getIntExtra("time", 1);
    MyRun mr = new MyRun(time, startId);
    es.execute(mr);
    return super.onStartCommand(intent, flags, startId);
  }

  public IBinder onBind(Intent arg0) {
    return null;
  }

  class MyRun implements Runnable {

    int time;
    int startId;

    public MyRun(int time, int startId) {
      this.time = time;
      this.startId = startId;
      Log.d(LOG_TAG, "MyRun#" + startId + " create");
    }

    public void run() {
      Log.d(LOG_TAG, "MyRun#" + startId + " start, time = " + time);
      try {
        TimeUnit.SECONDS.sleep(time);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      try {
        Log.d(LOG_TAG, "MyRun#" + startId + " someRes = " + someRes.getClass() );
      } catch (NullPointerException e) {
        Log.d(LOG_TAG, "MyRun#" + startId + " error, null pointer");
      }
      stop();
    }

    void stop() {
      Log.d(LOG_TAG, "MyRun#" + startId + " end, stopSelf(" + startId + ")");
      stopSelf(startId);
    }
  }
}
```

Скажем прямо, тут немного нетривиально получилось :) , особенно если опыт работы в Java небольшой. Но попробуем разобраться.

В **onCreate** создаем некий объект someRes. Это моя выдумка, на примере этого объекта я попробую показать, какую нехорошую ситуацию с ресурсами можно получить с применением метода stopSelf(startId). Этот объект будет использоваться сервисом в обработках вызовов. Пока что на него можно особо не смотреть.

Executors.newFixedThreadPool(1) – эта строка дает нам объект (я буду называть его - **экзекутор**), который будет получать от нас **задачи** (Runnable) и запускать их по очереди в **одном** потоке (на вход ему мы передаем значение **1**). Он сделает за нас всю работу по управлению потоками.

В **onDestroy** обнуляем someRes.

В **onStartCommand** мы читаем из intent параметр time. Создаем Runnable-объект **MyRun**, передаем ему time и startId и отдаем этот объект **экзекутору**, который его запустит в отдельном потоке.

MyRun – Runnable-объект. Он и будет обрабатывать входящие вызовы сервиса. В конструкторе он получает **time** и

startId. Параметр **time** будет использован для кол-ва секунд **паузы** (т.е. эмуляции работы). А **startId** будет использован в методе **stopSelf(startId)**, который даст сервису понять, что вызов под номером **strartId** обработан. В лог выводим инфу о создании, старте и завершении работы. Также здесь используем объект someRes, в лог просто выводим его класс. Если же объект = null, то ловим эту ошибку и выводим ее в лог.

В общем, получилась, на первый взгляд, непонятная мегамонстроконструкция, но я реально не смог придумать ничего проще, чтобы наглядно показать тему урока. Сейчас все это взлетит и станет понятнее)

Почему вообще я использую потоки, я показал в прошлом уроке. Т.к. сервис работает в том же потоке, что и приложение и соответственно будет тормозить экран своей работой.

Да, и не забудьте прописать сервис в манифесте!

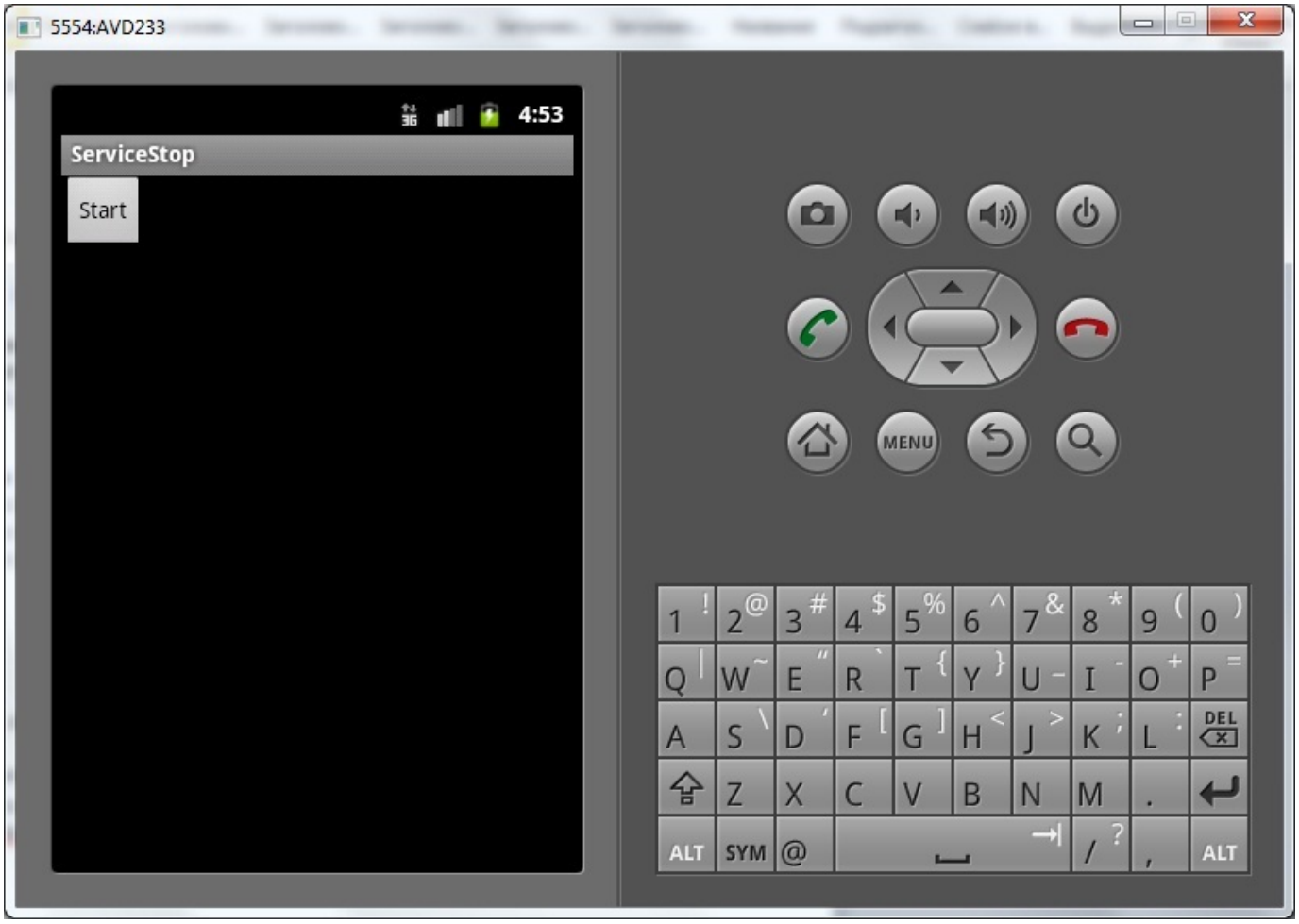
Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p0931servicestop;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  public void onClickStart(View v) {
    startService(new Intent(this, MyService.class).putExtra("time", 7));
    startService(new Intent(this, MyService.class).putExtra("time", 2));
    startService(new Intent(this, MyService.class).putExtra("time", 4));
  }
}
```

По нажатию на кнопку мы отправляем **вызов** в сервис **три** раза. И в **intent** помещаем параметр **time**.

Соответственно в сервисе будет **три** раза выполнен метод **onStartCommand**. Будет создано и передано экзекутору три **MyRun**-объекта. Он их по очереди начнет выполнять. Это займет у него соответственно 7,2 и 4 секунд (время паузы мы передаем в intent-е). В конце обработки каждого MyRun будет выполняться stopSelf(startId).

Все сохраняем и запускаем приложение.



Жмем **Start**, открываем логи и ждем секунд 15. Все должно закончиться строкой ***MyService onDestroy***

Давайте разбирать, чего получилось.

MyService onCreate

Сервис создан

MyService onStartCommand
MyRun#1 create
MyService onStartCommand
MyRun#2 create
MyService onStartCommand
MyRun#3 create

Три раза выполнился метод **onStartCommand**, т.к. мы в приложении три раза вызвали **startService**. В каждом вызове создалось по одному объекту MyRun. После символа # идет значение **startId**. Т.е. видим, что счетчик вызовов работает и startId увеличивается на единицу при каждом вызове.

Все MyRun объекты были переданы экзекутору, и он начинает их обрабатывать по очереди в одном потоке.

MyRun#1 start, time = 7
MyRun#1 someRes = class java.lang.Object
MyRun#1 end, stopSelf(1)

Первый готов. Метод stopSelf(1) вызван, сервис продолжает жить.

MyRun#2 start, time = 2
MyRun#2 someRes = class java.lang.Object
MyRun#2 end, stopSelf(2)

Второй готов. Метод stopSelf(2) вызван, сервис продолжает жить.

MyRun#3 start, time = 4
MyRun#3 someRes = class java.lang.Object
MyRun#3 end, stopSelf(3)
MyService onDestroy

Третий готов. Метод stopSelf(3) вызван и сервис после этого остановлен (***MyService onDestroy***), т.к. для последнего поступившего вызова (номер 3 в нашем случае) был вызван stopSelf(startId).

Но, как я написал в начале урока, мне кажется странным останавливать сервис, если закончена обработка последнего поступившего вызова. В этом примере у нас вызовы обрабатывались по очереди и проблем не возникло. Т.е. последний поступивший вызов обрабатывался последним и его stopSelf(startId) останавливал сервис. Но, что если вызовы будут обрабатываться параллельно? И последний поступивший вызов выполнит stopSelf(startId), а первый поступивший еще будет работать? Проверим.

В **MyService.java** в методе **onCreate** в строке создания экзекутора поменяйте 1 на 3. Должно получиться так:

```
public void onCreate() {
  super.onCreate();
  Log.d(LOG_TAG, "MyService onCreate");
  es = Executors.newFixedThreadPool(3);
  someRes = new Object();
}
```

Теперь наш экзекутор будет выполнять поступающие ему задачи не в одном, а в **трех** разных потоках. Как говорил один полосатый кошак из Простоквашино: «я и так счастливый был, а теперь в три раза счастливее стал, потому что у меня теперь три потока есть» :) Соответственно, три наших вызова будут теперь обработаны параллельно, а не по очереди. И пришло время обратить внимание на **someRes**.

Все сохраним и запустим, жмем **Start**. На этот раз все заняло не 15 сек, а 7. Последняя строка логов должна быть такой: ***MyRun#1 end, stopSelf(1)***

Разбираемся в логах.

MyService onCreate

Сервис создан

MyService onStartCommand
MyRun#1 create
MyService onStartCommand
MyRun#2 create

MyRun#1 start, time = 7
MyService onStartCommand
MyRun#3 create
MyRun#2 start, time = 2
MyRun#3 start, time = 4
(у вас может быть немного иной порядок строк)

Три раза выполнился метод **onStartCommand**, т.к. мы в приложении три раза вызвали startService. В каждом вызове создалось по одному объекту MyRun и все они сразу начали работать, потому что экзекутор теперь раскидал их по трем потокам и им не надо ждать друг друга.

Судя по параметру time, который мы используем для паузы, MyRun#2 закончит работать первым (через 2 сек.), MyRun#3 – вторым (через 4 сек.), MyRun#1 – третьим (через 7 сек.). Так и происходит.

MyRun#2 someRes = class java.lang.Object
MyRun#2 end, stopSelf(2)

Объект **someRes доступен**. Вызов с startId = 2 обработан. stopSelf(2) вызван – сервис продолжает жить.

MyRun#3 someRes = class java.lang.Object
MyRun#3 end, stopSelf(3)
MyService onDestroy

someRes доступен. Вызов с startId = 3 обработан. stopSelf(3) вызван – сервис остановлен (MyService onDestroy). Тут все верно - последний поступивший вызов имеет startId = 3, следовательно, вызов метода stopSelf(3) останавливает сервис. Следовательно, выполнился метод onDestroy, а в нем **обнулился someRes**.

MyRun#1 error, null pointer
MyRun#1 end, stopSelf(1)

someRes недоступен, т.к. он был обнулен чуть раньше в **onDestroy**. Ну и вызов метода stopSelf(1) уже бесполезен. Сервис был официально остановлен ранее.

Т.е. получилась неприятная ситуация, когда мы в onDestroy **освободили** объект, а он был еще **нужен**. А ведь вместо someRes мог быть объект по работе с БД. Здесь надо быть аккуратным, понимать механизм работы stopSelf(startId) и, когда именно этот метод остановит сервис.

Ну и напоследок рассмотрим метод stopSelfResult(startId). Напомню, что он полностью аналогичен методу stopSelf(startId), и при этом возвращает boolean-значение, остановлен сервис или нет после вызова этого метода.

В **MyService.java** в классе **MyRun** перепишем метод **stop**:

```
    void stop() {
      Log.d(LOG_TAG, "MyRun#" + startId + " end, stopSelfResult("
          + startId + ") = " + stopSelfResult(startId));
    }
```

Мы выводим в лог результат вызова метода **stopSelfResult**.

Сохраняем, запускаем и жмем **Start**.

Смотрим логи:

MyService onCreate
MyService onStartCommand
MyRun#1 create
MyService onStartCommand
MyRun#2 create
MyRun#1 start, time = 7
MyService onStartCommand
MyRun#3 create
MyRun#2 start, time = 2
MyRun#3 start, time = 4
MyRun#2 someRes = class java.lang.Object
MyRun#2 end, stopSelfResult(2) = false
MyRun#3 someRes = class java.lang.Object
MyService onDestroy
MyRun#3 end, stopSelfResult(3) = true
MyRun#1 error, null pointer
MyRun#1 end, stopSelfResult(1) = false

Видим, что stopSelfResult(3) вернул true, и тем самым сообщил нам, что именно он остановил сервис. А stopSelfResult(1) и stopSelfResult(2) вернули false – их вызов не привел к остановке сервиса.

Понимаю, что сложно это все, но надеюсь, что у меня получилось тему раскрыть. Если остаются сомнения, попробуйте поиграться экзекутором, кол-вом потоков и вызовов, и параметром time – должно стать понятнее.

А вообще, до тех пор, пока вам это все не понадобится в боевых условиях, вполне можно понимать это не до конца.

На следующем уроке:

- рассмотрим подробнее метод onStartCommand

Урок 94. Service. Подробно про onStartCommand

В этом уроке:

- рассмотрим подробнее метод onStartCommand

Метод onStartCommand должен возвращать **int**. В прошлых примерах мы использовали ответ метода супер-класса. В этом уроке разберемся, что мы можем возвращать, и чем нам это грозит. Также посмотрим, что за флаги (второй параметр) идут на вход этому методу.

Система может убить сервис, если ей будет не хватать памяти. Но в наших силах сделать так, чтобы наш сервис ожил, когда проблема с памятью будет устранена. И более того, не просто ожил, а еще и снова начал выполнять незавершенные вызовы **startService**.

Т.е. мы вызываем startService, срабатывает **onStartCommand** и возвращает одну из следующих констант:

START_NOT_STICKY – сервис не будет перезапущен после того, как был убит системой

START_STICKY – сервис будет перезапущен после того, как был убит системой

START_REDELIVER_INTENT – сервис будет перезапущен после того, как был убит системой. Кроме этого, сервис снова получит все вызовы startService, которые не были завершены методом **stopSelf(startId)**.

А второй параметр **flags** метода onStartCommand дает нам понять, что это повторная попытка вызова onStartCommand.

В хелпе написано, что flags может принимать значения 0, START_FLAG_RETRY или START_FLAG_REDELIVERY.

На практике я ни разу не встречал значения 0. Постоянно приходит START_FLAG_RETRY. По этому поводу я видел в инете мнение, что это реальный баг системы.

А вот на флаг **START_FLAG_REDELIVERY** можно положиться. Если он пришел вам в методе onStartCommand, значит, прошлый вызов этого метода вернул **START_REDELIVER_INTENT**, но не был завершен успешно методом stopSelf(startId).

В общем, на словах трудно все это объяснять, сейчас нарисуем приложение, и все станет понятнее.

Будем создавать два проекта - соответственно, два Application-а у нас получится. В **первом** будет только одно **Activity**, которое будет вызывать сервис. А во **втором** собственно и будет этот **сервис**. Т.е. в прошлых уроках у нас приложение и сервис были в **одном** Application, а в этом уроке мы их раскидаем по **двум разным** Application. И сейчас даже объясню, зачем мы так сделаем.

В начале урока я написал «Система может убить сервис». Этим мы и будем сейчас заниматься, чтобы на практике проверить поведение сервиса в таких случаях. Мы будем убивать процесс, который отвечает за сервис. Если бы у нас сервис и приложение были бы в одном Application, то они выполнялись бы в одном процессе. И убив процесс, мы грохнули бы и сервис и приложение. А нам надо убить только сервис, поэтому выносим его в отдельное Application, и он в своем процессе будет работать.

Я раскидал сервис и приложение по разным проектам для большей наглядности. А вообще, организовать работу сервиса и приложения в разных процессах можно было и в одном проекте (в одном Application). Для этого в манифесте для сервиса надо прописать атрибут: *android:process=":newproc"*. Вместо newproc можно использовать свое слово.

Итак, создадим первый проект:

Project name: P0941_ServiceKillClient
Build Target: Android 2.3.3
Application name: ServiceKillClient
Package name: ru.startandroid.develop.p0941servicekillclient
Create Activity: MainActivity

Это у нас будет приложение-клиент, оно будет вызывать сервис.

Добавим в **strings.xml** строки:

```
    <string  name="start">Start</string>
```

Экран **main.xml**:

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <Button
          android:id="@+id/btnStart"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onClickStart"
          android:text="@string/start">
     </Button>
</LinearLayout>
```

MainActivity.java:

```java
package ru.startandroid.develop.p0941servicekillclient;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  public void onClickStart(View v) {
    startService(new Intent("ru.startandroid.develop.p0942servicekillserver.MyService").putExtra("name", "value"));
  }
}
```

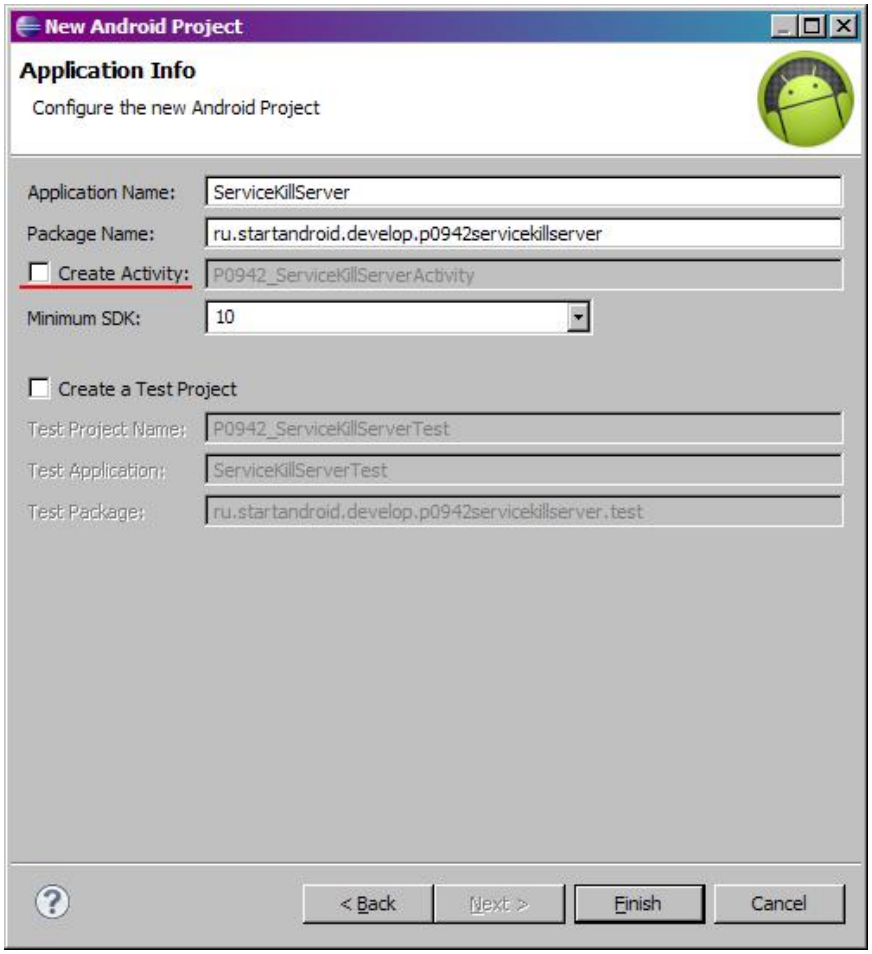
Просто вызов сервиса. Вспоминаем способы вызова Activity, которые мы знаем, и понимаем, что здесь идет аналогичный вызов сервиса через Action. А значит, когда будем создавать сервис, надо будет настроить Intent Filter с этим Action. На прошлых уроках мы сервис вызывали через класс, т.к. там приложение и сервис в одном Application у нас были.

В Intent помещаем параметр с именем name и значением value. Это нам понадобится, чтобы убедиться, что система при восстановлении сервиса и повторном вызове не потеряет наши данные.

Создадим второй проект, без Activity:

Project name: P0942_ServiceKillServer
Build Target: Android 2.3.3
Application name: ServiceKillServer
Package name: ru.startandroid.develop.p0942servicekillserver

Уберите эту галку



Нам в этом Application не понадобится Activity.

В втором проекте создаем сервис, класс **MyService.java**:

```
package ru.startandroid.develop.p0942servicekillserver;

import java.util.concurrent.TimeUnit;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {

  final String LOG_TAG = "myLogs";

  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "MyService onCreate");
  }

  public void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "MyService onDestroy");
  }

  public int onStartCommand(Intent intent, int flags, int startId) {
    Log.d(LOG_TAG, "MyService onStartCommand");
    readFlags(flags);
    MyRun mr = new MyRun(startId);
    new Thread(mr).start();
    return START_NOT_STICKY;
  }

  public IBinder onBind(Intent arg0) {
    return null;
```

```
  }

  void readFlags(int flags) {
    if ((flags&START_FLAG_REDELIVERY) == START_FLAG_REDELIVERY)
      Log.d(LOG_TAG, "START_FLAG_REDELIVERY");
    if ((flags&START_FLAG_RETRY) == START_FLAG_RETRY)
      Log.d(LOG_TAG, "START_FLAG_RETRY");
  }

  class MyRun implements Runnable {

    int startId;

    public MyRun(int startId) {
      this.startId = startId;
      Log.d(LOG_TAG, "MyRun#" + startId + " create");
    }

    public void run() {
      Log.d(LOG_TAG, "MyRun#" + startId + " start");
      try {
        TimeUnit.SECONDS.sleep(15);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      stop();
    }

    void stop() {
      Log.d(LOG_TAG, "MyRun#" + startId + " end, stopSelfResult("
          + startId + ") = " + stopSelfResult(startId));
    }
  }
}
```

Он похож на сервис с прошлого урока.

MyRun – Runnable класс, который эмулирует долгую работу. Ставим в нем паузу 15 секунд. После этого вызываем stopSelfResult.

В **onStartCommand** пишем в лог значение из Intent, создаем экземпляр MyRun и отправляем его работать в новый поток. Также здесь вызываем метод readFlags и передаем ему flags. И возвращаем константу START_NOT_STICKY. Напомню, что START_NOT_STICKY означает, что сервис не будет перезапущен после того, как будет убит.

В методе **readFlags** определяем, какие флаги содержатся в readFlags, и отмечаем это в логе.

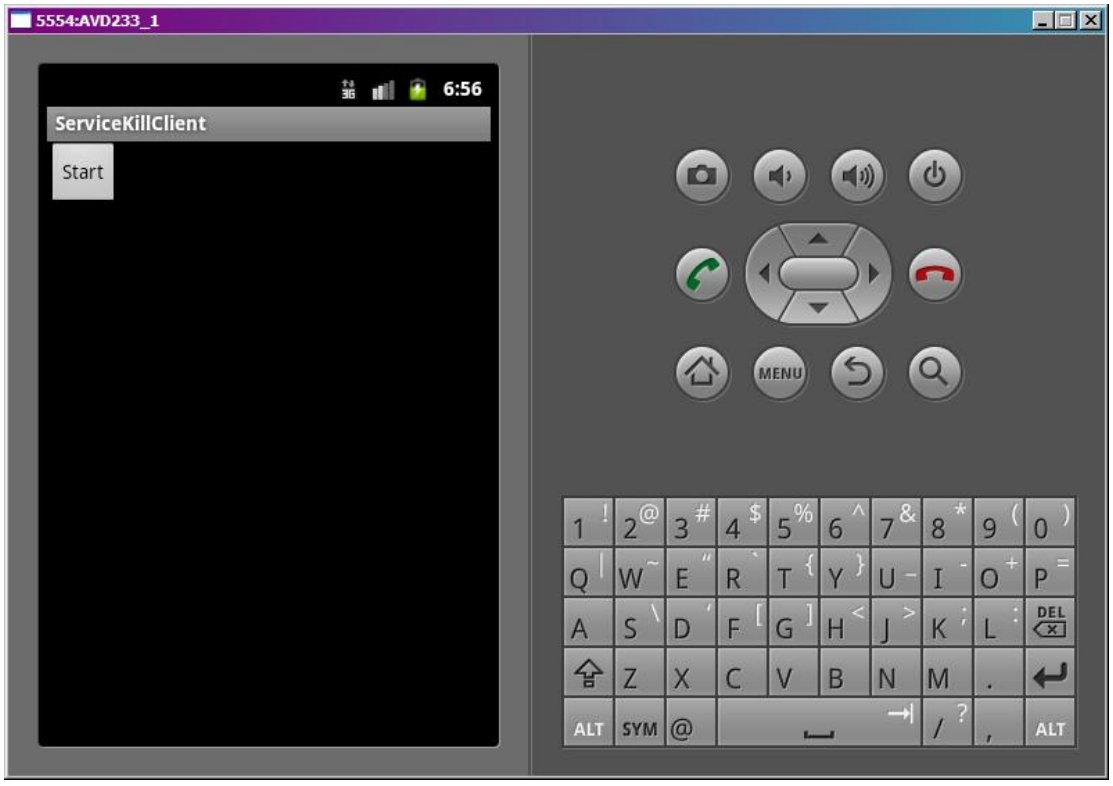
Прописываем сервис в манифесте и добавляем к нему IntentFilter с action = *ru.startandroid.develop.p0942servicekillserver.MyService*.

Все сохраняем. Для начала, запускаем второй проект (P0942_ServiceKillServer), он установится в эмулятор, но ничего не произойдет, т.к. Activity никаких не было, только сервис.

В консоли будут такие строки:

Installing P0942_ServiceKillServer.apk...
Success!
\P0942_ServiceKillServer\bin\P0942_ServiceKillServer.apk installed on device
Done!

Теперь запускаем первый проект. Приложение установится в эмулятор и запустится.

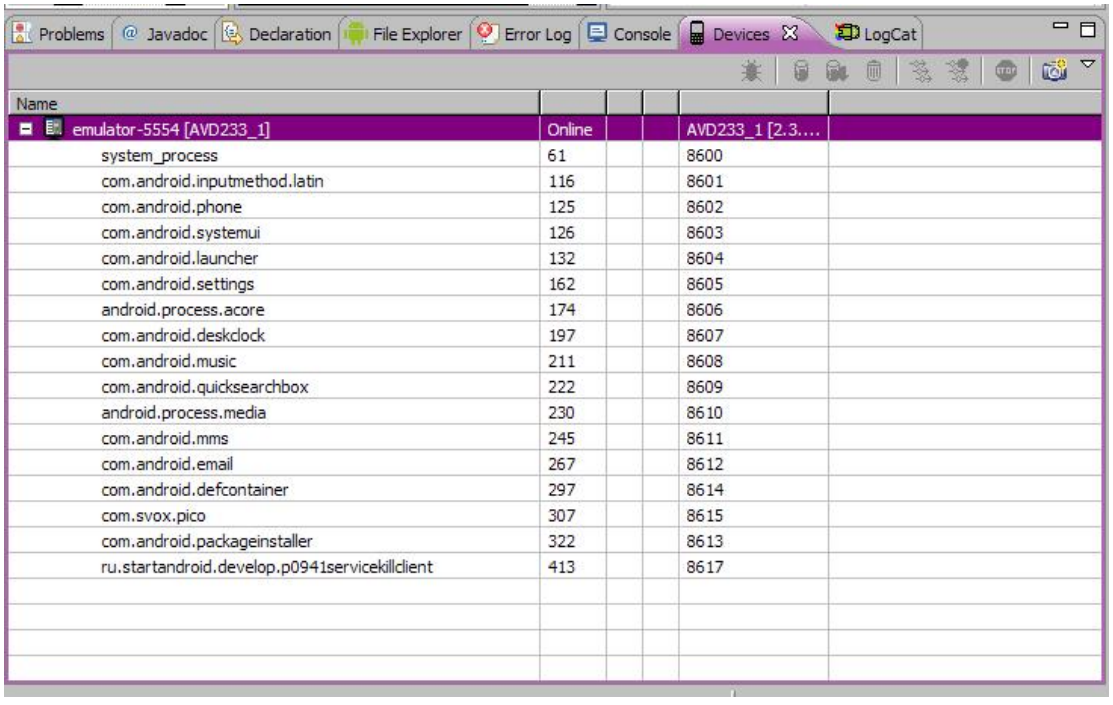


Нам надо получить доступ к процессам. Для этого можно открыть перспективу DDMS и там увидеть все процессы.

А можно и в нашу текущую перспективу добавить это окно.

В Eclipse меню Window > Show View > Other > Android > Devices.

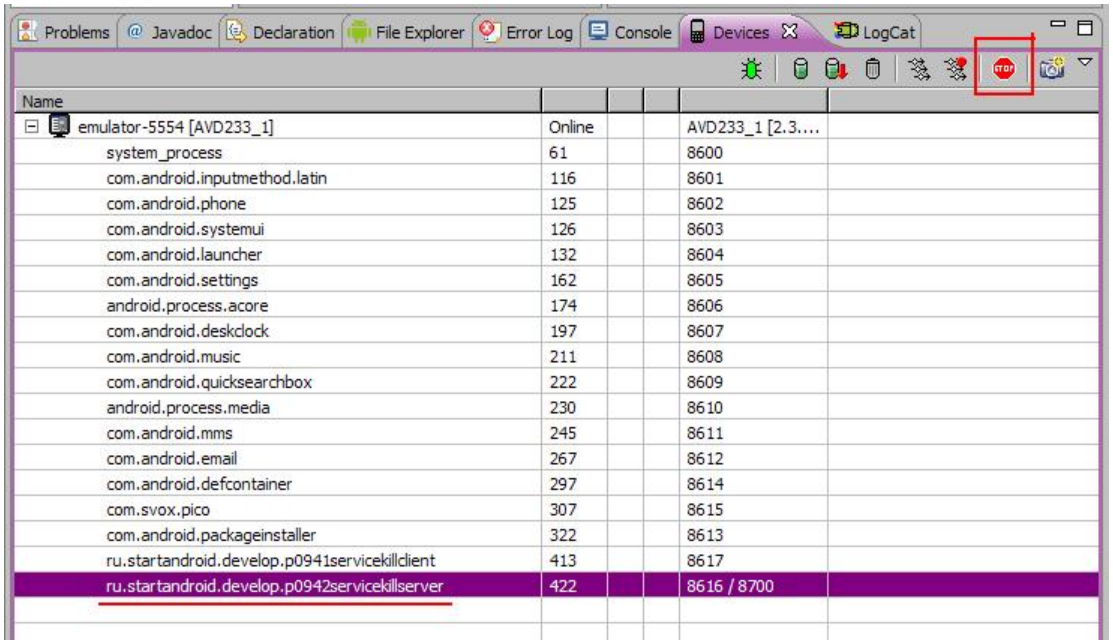
Снизу должна появиться вкладка **Devices**. В ней показан наш эмулятор и запущенные процессы.



Мы видим, что запущено наше приложение. А когда мы нажмем в приложении кнопку **Start**, запустится сервис и в списке процессов появится еще один процесс. Вот его нам и надо будет остановить.

У нас будет 15 секунд, чтобы удалить его. Потому что через 15 секунд он успешно вызовет **stopSelfResult** и сам закроется.

Итак, жмем **Start**. Появился процесс



Выделяем его и жмем кнопку **Stop Process**. Процесс исчез. Смотрим логи:

MyService onCreate
MyService onStartCommand, name = value
START_FLAG_RETRY
MyRun#1 create
MyRun#1 start

Сервис создался и начал выполнять вызов startService. При этом почему-то мы получили флаг START_FLAG_RETRY. Хотя это у нас первая попытка вызова сервиса. Это я, к сожалению, никак не могу прокомментировать.

Далее видим, как создался и начал работу MyRun#1. А потом мы убили процесс, и все закончилось на этом. Метод onStartCommand вернул START_NOT_STICKY и поэтому не был восстановлен после убийства.

Давайте проверим следующую константу. Перепишем **onStartCommand**:

```
public int onStartCommand(Intent intent, int flags, int startId) {
  Log.d(LOG_TAG, "MyService onStartCommand, name = " + intent.getStringExtra("name"));
  readFlags(flags);
  MyRun mr = new MyRun(startId);
  new Thread(mr).start();
  return START_STICKY;
}
```

Теперь будем возвращать START_STICKY. Это значит, что сервис будет восстановлен после убийства.

Все сохраняем, запускаем (CTRL+F11). Наш сервис инсталлится на эмулятор. А приложение так и работает до сих пор.

Жмем Start и сразу же в процессах убиваем появившийся процесс сервиса. Но! Фокус еще не закончен. Проходит несколько секунд (у меня около 5, у вас может быть по-другому), и процесс снова появляется в списке! Система восстановила убитый сервис и сколько вы его теперь не грохайте, он постоянно будет восстанавливаться. Это нам обеспечила константа START_STICKY, которую мы возвращаем в onStartCommand.

Смотрим логи:

MyService onCreate
MyService onStartCommand, name = value
START_FLAG_RETRY
MyRun#1 create
MyRun#1 start

Почти все так же, как и в прошлый раз. Сервис запустился, получил вызов startService, создал и запустил MyRun. Потом мы его убили - в логах это никак не отражено.

MyService onCreate

И далее сервис снова создан (MyService onCreate). Но вызов startService, который он получил в прошлый раз, потерян.

Для того, чтобы этот вызов не терялся, нужна константа START_REDELIVER_INTENT. Будем возвращать ее в методе onStartCommand:

```
public int onStartCommand(Intent intent, int flags, int startId) {
  Log.d(LOG_TAG, "MyService onStartCommand, name = " + intent.getStringExtra("name"));
  readFlags(flags);
  MyRun mr = new MyRun(startId);
  new Thread(mr).start();
  return START_REDELIVER_INTENT;
}
```

Сохраняем, инсталлим сервис. Жмем **Start** и убиваем процесс. Процесс снова появляется спустя какое-то время. Смотрим логи:

MyService onCreate
MyService onStartCommand, name = value
START_FLAG_RETRY
MyRun#1 create
MyRun#1 start

Сервис начал работу и был убит, как в прошлые разы.

MyService onCreate
MyService onStartCommand, name = value
START_FLAG_REDELIVERY
START_FLAG_RETRY
MyRun#1 create
MyRun#1 start
MyRun#1 end, stopSelfResult(1) = true
MyService onDestroy

А далее он был снова запущен, снова получил вызов startService и успешно его отработал вплоть до stopSelfResult. После чего сервис сам остановился. Обратите внимание, что Intent мы получили тот же, что и в первую попытку, с параметром name = value. Т.е. система сохранила все, а когда процесс был убит, восстановила его и вернула нам данные.

В этот раз кроме START_FLAG_RETRY мы получили еще флаг **START_FLAG_REDELIVERY**, который говорит нам о том, что мы уже получали такой вызов, но в прошлый раз он по каким-то причинам не завершился методом stopSelf или stopSelfResult.

Если во время убийства сервиса будет висеть несколько таких работающих START_FLAG_REDELIVERY вызовов, для которых еще не был выполнен stopSelf, то все они будут снова отправлены в сервис при его восстановлении.

Вот такие три замечательные константы, которые позволяют нам управлять поведением сервиса после его насильственной смерти.

Наверняка возникает вопрос типа «Как поведет себя сервис, если отправить ему несколько вызовов и в каждом onStartCommand возвращать разные константы». Я в одном уроке не смогу рассмотреть все эти комбинации и варианты. Предлагаю потестить вам это самостоятельно.

И еще имеет смысл сказать, что метод onStartCommand появился только начиная с Android 2.0. До этого использовался более простой onStart, без механизма восстановления.

На следующем уроке:

- получаем из сервиса результат с помощью PendingIntent

Урок 95. Service. Обратная связь с помощью PendingIntent

В этом уроке:

- получаем из сервиса результат с помощью PendingIntent

В прошлых уроках мы стартовали сервисы, передавали им данные, но ничего не получали обратно в вызывающее Activity. Но это возможно, и существует несколько способов. В этом уроке рассмотрим PendingIntent.

Я особо не буду вдаваться в объяснение, что такое PendingIntent и что он умеет - урок не о нем. Я покажу, как с его помощью можно в Activity получать результаты работы сервиса.

Схема такая:

- в Activity создаем PendingIntent с помощью метода createPendingResult

- кладем этот PendingIntent в обычный Intent, который используем для старта сервиса и вызываем startService

- в сервисе извлекаем PendingIntent из полученного в методе onStartCommand объекта Intent

- когда нам необходимо передать результаты работы из сервиса в Activity, вызываем метод send для объекта PendingIntent

- эти результаты из сервиса ловим в Activity в методе onActivityResult

Т.е. фишка PendingIntent здесь в том, что он содержит некую связь с Activity (в котором он был создан) и, когда вызывается метод send, он идет в это Activity и несет данные, если необходимо. В общем, этакий почтовый голубь, который точно знает, как ему вернуться домой.

Схема в целом несложная. Попробуем нарисовать пример по ней. У нас будет приложение, которое будет отправлять в сервис на выполнение три задачи. А сервис будет информировать, когда он начал каждую задачу выполнять, когда закончил и с каким результатом. Все это будем выводить на экран Activity.

Кстати, чтобы легче было воспринимать это все, замечу, что алгоритм очень похож на работу startActivityForResult. Только там мы взаимодействуем не с сервисом, а с Activity. Если подзабыли, посмотрите уроки 29 и 30. Воспринимать текущий материал будет гораздо легче.

Создадим проект:

Project name: P0951_ServiceBackPendingIntent
Build Target: Android 2.3.3
Application name: ServiceBackPendingIntent
Package name: ru.startandroid.develop.p0951servicebackpendingintent
Create Activity: MainActivity

Добавим в **strings.xml** строки:

```
    <string  name="start">Start</string>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
     <TextView
          android:id="@+id/tvTask1"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="">
     </TextView>
     <TextView
          android:id="@+id/tvTask2"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="">
     </TextView>
     <TextView
          android:id="@+id/tvTask3"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="">
     </TextView>
     <Button
          android:id="@+id/btnStart"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onClickStart"
          android:text="@string/start">
     </Button>
</LinearLayout>
```

Три TextView, в которые будем выводить инфу, поступающую из сервиса. И кнопка старта сервиса.

Создаем класс для сервиса - **MyService.java**. И пропишем его в манифесте. Пока в нем ничего не кодим.

MainActivity.java:

```
package ru.startandroid.develop.p0951servicebackpendingintent;

import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
```

```
public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final int TASK1_CODE = 1;
  final int TASK2_CODE = 2;
  final int TASK3_CODE = 3;

  public final static int STATUS_START = 100;
  public final static int STATUS_FINISH = 200;

  public final static String PARAM_TIME = "time";
  public final static String PARAM_PINTENT = "pendingIntent";
  public final static String PARAM_RESULT = "result";

  TextView tvTask1;
  TextView tvTask2;
  TextView tvTask3;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tvTask1 = (TextView) findViewById(R.id.tvTask1);
    tvTask1.setText("Task1");
    tvTask2 = (TextView) findViewById(R.id.tvTask2);
    tvTask2.setText("Task2");
    tvTask3 = (TextView) findViewById(R.id.tvTask3);
    tvTask3.setText("Task3");
  }

  public void onClickStart(View v) {
    PendingIntent pi;
    Intent intent;

    // Создаем PendingIntent для Task1
    pi = createPendingResult(TASK1_CODE, null, 0);
    // Создаем Intent для вызова сервиса, кладем туда параметр времени
    // и созданный PendingIntent
    intent = new Intent(this, MyService.class).putExtra(PARAM_TIME, 7)
        .putExtra(PARAM_PINTENT, pi);
    // стартуем сервис
    startService(intent);

    pi = createPendingResult(TASK2_CODE, null, 0);
    intent = new Intent(this, MyService.class).putExtra(PARAM_TIME, 4)
        .putExtra(PARAM_PINTENT, pi);
    startService(intent);

    pi = createPendingResult(TASK3_CODE, null, 0);
    intent = new Intent(this, MyService.class).putExtra(PARAM_TIME, 6)
        .putExtra(PARAM_PINTENT, pi);
    startService(intent);
  }

  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
```

```
    Log.d(LOG_TAG, "requestCode = " + requestCode + ", resultCode = "
        + resultCode);

    // Ловим сообщения о старте задач
    if (resultCode == STATUS_START) {
      switch (requestCode) {
      case TASK1_CODE:
        tvTask1.setText("Task1 start");
        break;
      case TASK2_CODE:
        tvTask2.setText("Task2 start");
        break;
      case TASK3_CODE:
        tvTask3.setText("Task3 start");
        break;
      }
    }

    // Ловим сообщения об окончании задач
    if (resultCode == STATUS_FINISH) {
      int result = data.getIntExtra(PARAM_RESULT, 0);
      switch (requestCode) {
      case TASK1_CODE:
        tvTask1.setText("Task1 finish, result = " + result);
        break;
      case TASK2_CODE:
        tvTask2.setText("Task2 finish, result = " + result);
        break;
      case TASK3_CODE:
        tvTask3.setText("Task3 finish, result = " + result);
        break;
      }
    }
  }
}
```

В **onCreate** мы находим TextView и присваиваем им первоначальный текст. Для каждой задачи свой TextView.

В **onClickStart** мы создаем PendingIntent методом createPendingResult. На вход методу передаем только код запроса – можно считать это идентификатором. По этому коду мы потом будем определять, какая именно задача вернула ответ из сервиса. Два остальных параметра – это Intent и флаги. Нам они сейчас не нужны, передаем соответственно null и 0.

Далее создаем Intent для вызова сервиса MyService, помещаем туда параметр времени (который будем использовать для паузы в сервисе) и PendingIntent. После чего, отправляем это все в сервис.

Аналогичные действия производим для Task2 и Task3.

В **onActivityResult** проверяем, какой тип сообщения пришел из сервиса.

Если о том, что задача начала работу (STATUS_START), то определяем, какая именно задача, и пишем об этом в соответствующий TextView.

А если о том, что задача завершена (STATUS_FINISH), то читаем из Intent-а результат выполнения, определяем, какая именно задача, и пишем инфу об этом в соответствующий TextView.

Всю инфу о поступающих сообщениях пишем в лог.

Как вы понимаете, коды STATUS_START и STATUS_FINISH, а также результат мы сейчас будем формировать в сервисе.

Кодим сервис **MyService.java**:

```
package ru.startandroid.develop.p0951servicebackpendingintent;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import android.app.PendingIntent;
import android.app.PendingIntent.CanceledException;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {

  final String LOG_TAG = "myLogs";
  ExecutorService es;

  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "MyService onCreate");
    es = Executors.newFixedThreadPool(2);
  }

  public void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "MyService onDestroy");
  }

  public int onStartCommand(Intent intent, int flags, int startId) {
    Log.d(LOG_TAG, "MyService onStartCommand");

    int time = intent.getIntExtra(MainActivity.PARAM_TIME, 1);
    PendingIntent pi = intent.getParcelableExtra(MainActivity.PARAM_PINTENT);

    MyRun mr = new MyRun(time, startId, pi);
    es.execute(mr);

    return super.onStartCommand(intent, flags, startId);
  }

  public IBinder onBind(Intent arg0) {
    return null;
  }

  class MyRun implements Runnable {

    int time;
    int startId;
    PendingIntent pi;

    public MyRun(int time, int startId, PendingIntent pi) {
      this.time = time;
      this.startId = startId;
      this.pi = pi;
```

```
      Log.d(LOG_TAG, "MyRun#" + startId + " create");
    }

    public void run() {
      Log.d(LOG_TAG, "MyRun#" + startId + " start, time = " + time);
      try {
        // сообщаем об старте задачи
        pi.send(MainActivity.STATUS_START);

        // начинаем выполнение задачи
        TimeUnit.SECONDS.sleep(time);

        // сообщаем об окончании задачи
        Intent intent = new Intent().putExtra(MainActivity.PARAM_RESULT, time * 100);
        pi.send(MyService.this, MainActivity.STATUS_FINISH, intent);

      } catch (InterruptedException e) {
        e.printStackTrace();
      } catch (CanceledException e) {
        e.printStackTrace();
      }
      stop();
    }

    void stop() {
      Log.d(LOG_TAG, "MyRun#" + startId + " end, stopSelfResult("
          + startId + ") = " + stopSelfResult(startId));
    }
  }
}
```

Мы снова используем знакомую по прошлым урокам схему экзекутора и Runnable.

В **onCreate** создаем экзекутор с двумя потоками. Т.е. когда сервис получит три задачи, он сразу начнет выполнять две из них, а третья будет ждать свободного потока.

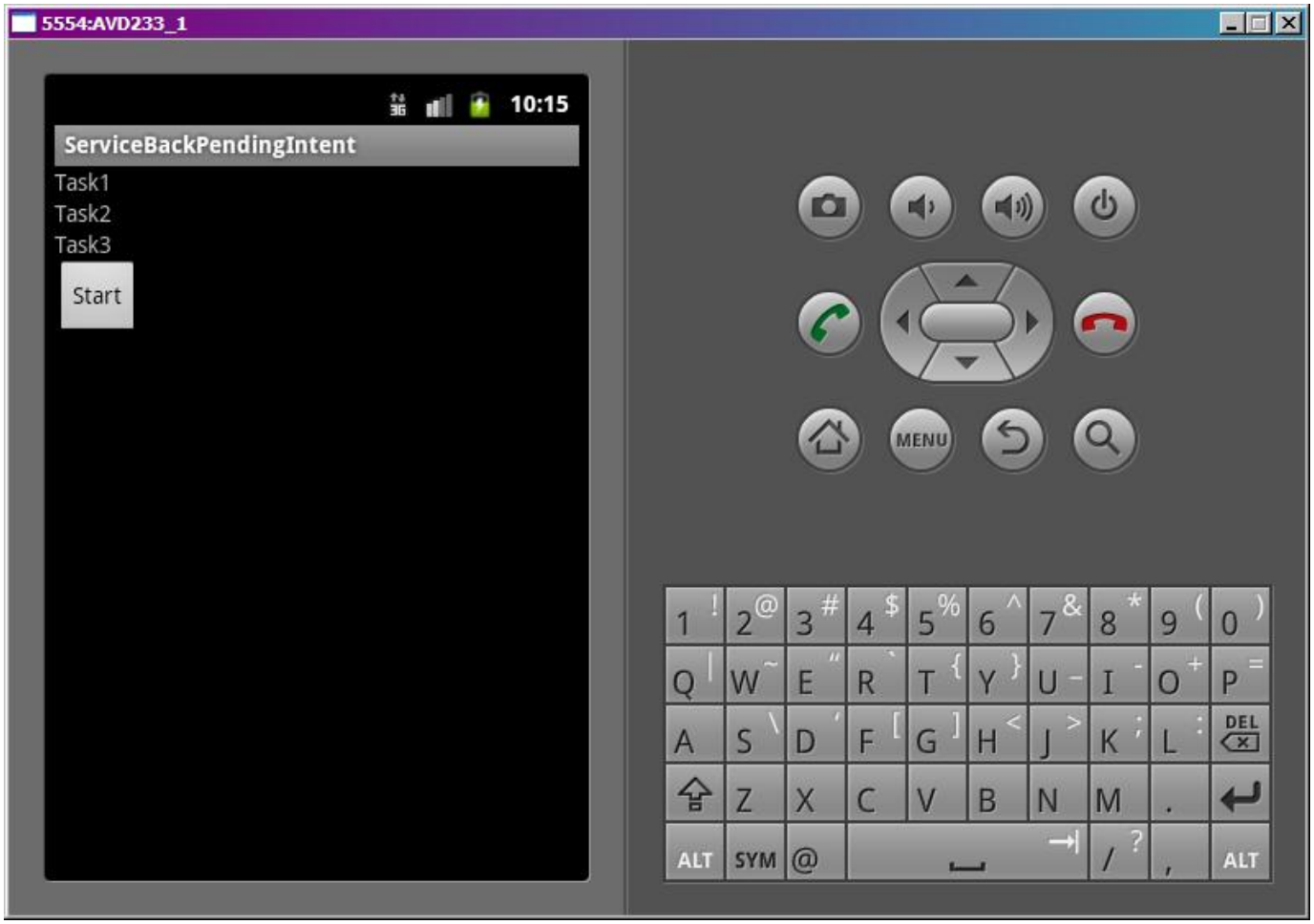
В **onStartCommand** вытаскиваем из Intent-а параметр времени для паузы и PendingIntent. Создаем MyRun и передаем ему эти данные. Передаем MyRun экзекутору на выполнение.

MyRun при запуске вызывает метод send для PendingIntent и передает туда тип сообщения STATUS_START. Это приходит в метод onActivityResult в Activity и на экране мы увидим, как в одном из TextView появится текст, что задача начала работать.

Далее мы эмулируем работу, как обычно, просто поставив паузу. А после этого создаем Intent с результатом работы (просто время * 100), и вызываем немного другую реализацию метода send. Кроме типа сообщения (STATUS_FINISH), мы передаем туда Intent с результатом и указываем контекст. Это идет в метод onActivityResult в Activity и на экране мы увидим, как в одном из TextView появится текст, что задача закончила работу с неким результатом.

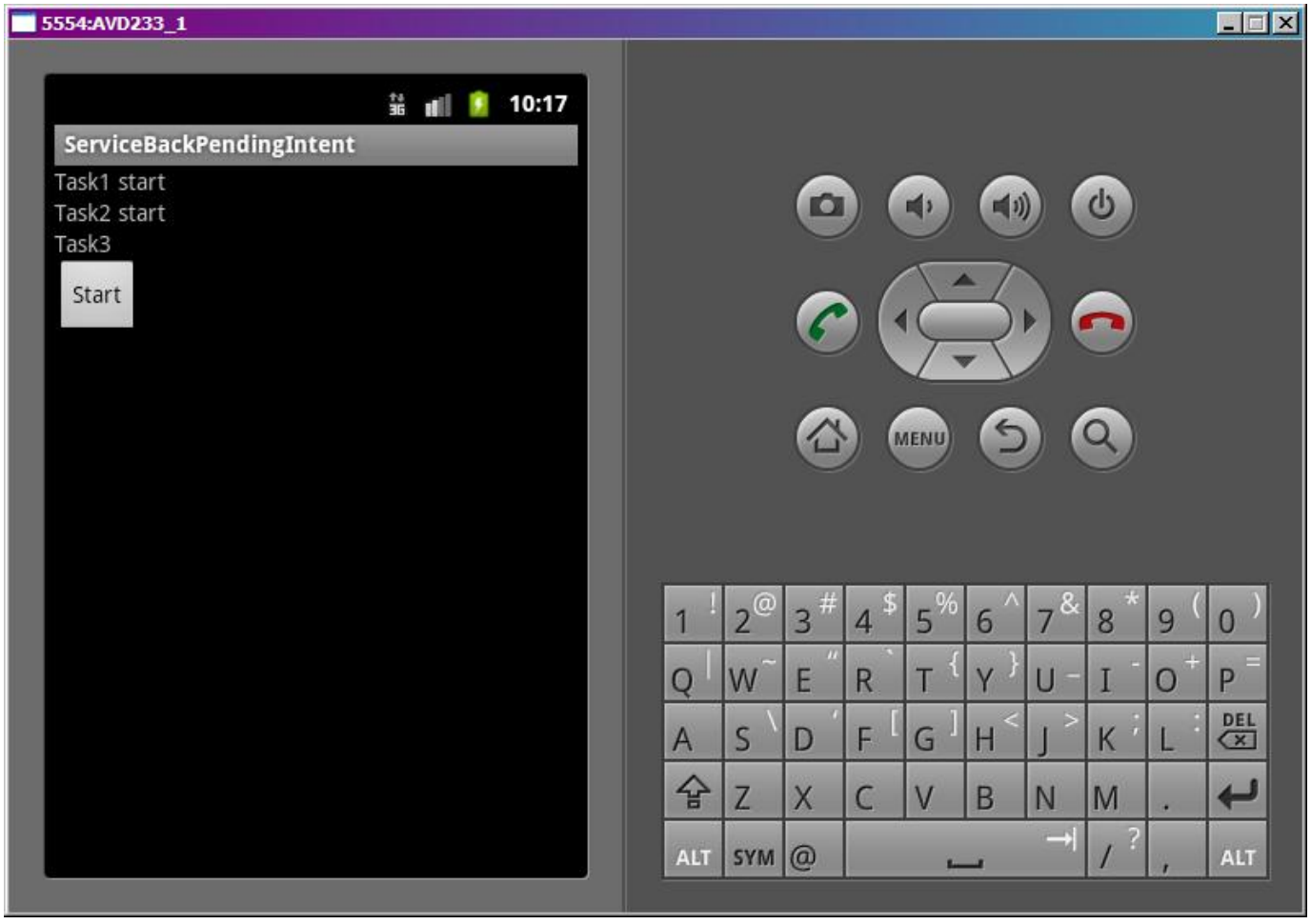
Далее вызываем метод stop, в котором вызывается метод stopSelfResult.

Все сохраняем и запускаем приложение.

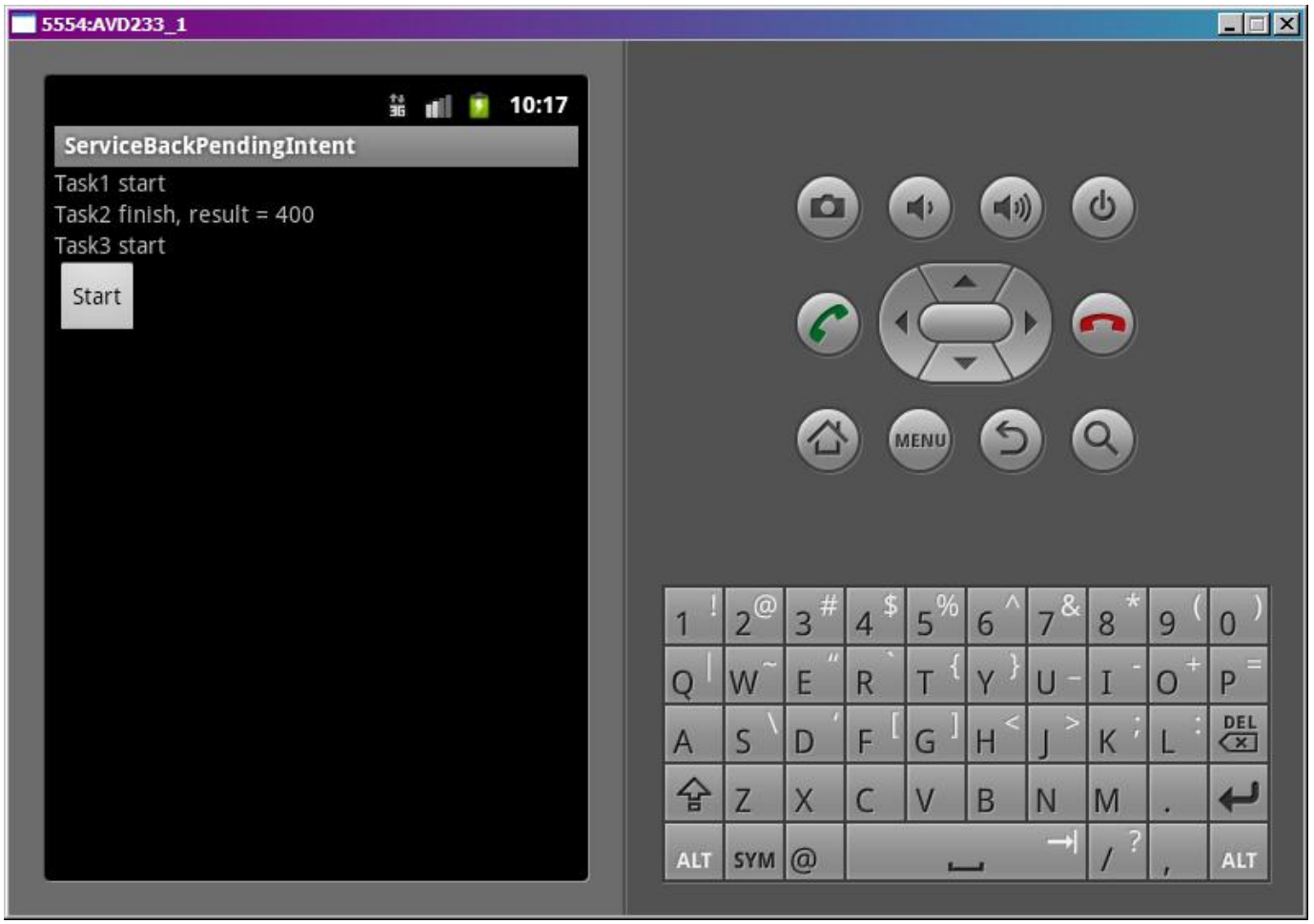


Жмем **Start**.

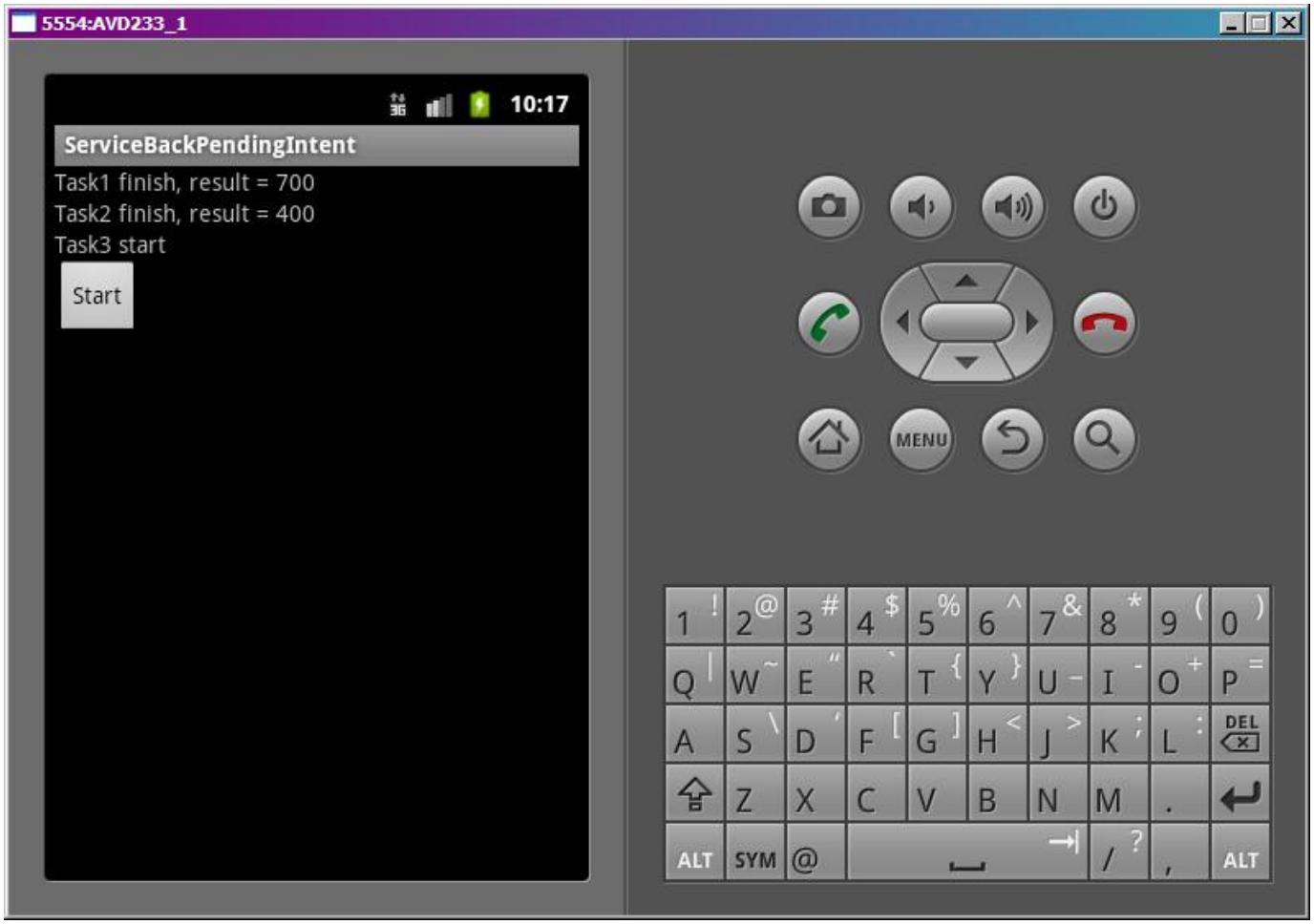
Видим, что две задачи начали работать, т.к. экзекутор настроен на два потока.



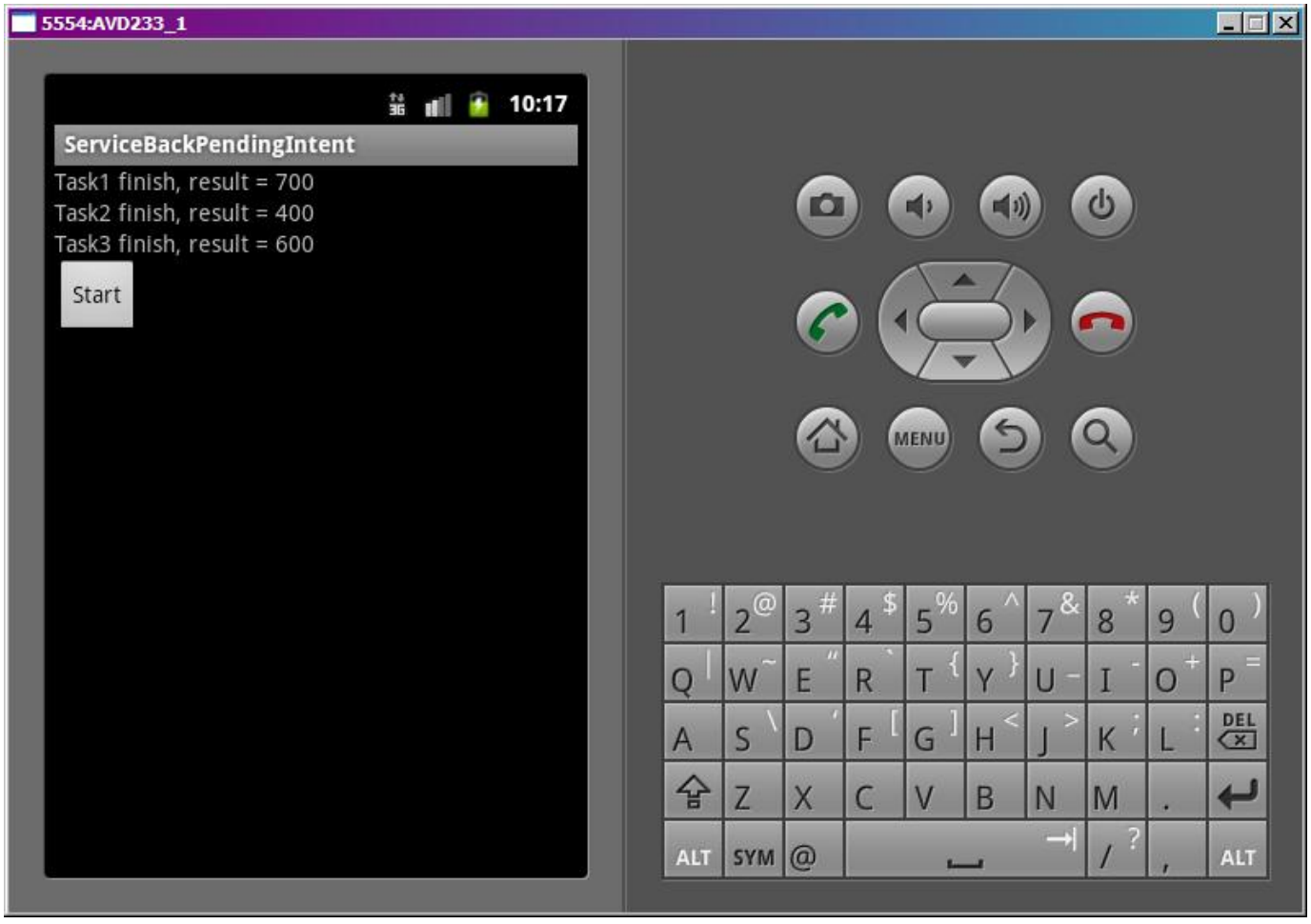
Одна задача завершилась и показала результат, поток освободился, стартует оставшаяся задача.



Еще одна задача завершилась.



Последняя заверш илась.



Смотрим логи (у вас может быть немного другая последовательность записей в логах):

MyService onCreate
MyService onStartCommand
MyRun#1 create
MyService onStartCommand
MyRun#2 create
MyRun#1 start, time = 7
MyService onStartCommand
MyRun#3 create

Сервис создался и получил все три вызова.

requestCode = 1, resultCode = 100

MyRun1 начал работать и отправил сообщение об этом: requestCode = TASK1_CODE, resultCode = STATUS_START.

MyRun#2 start, time = 4
requestCode = 2, resultCode = 100

MyRun2 начал работать и отправил сообщение об этом: requestCode = TASK2_CODE, resultCode = STATUS_START.

MyRun#2 end, stopSelfResult(2) = false
MyRun#3 start, time = 6
requestCode = 2, resultCode = 200
requestCode = 3, resultCode = 100

MyRun2 завершил работу и отправил соответствующее сообщение: requestCode = TASK2_CODE, resultCode = STATUS_FINISH. Поток в экзекуторе освободился и стартует MyRun3. MyRun3 начал работать и отправил сообщение об этом: requestCode = TASK3_CODE, resultCode = STATUS_START.

MyRun#1 end, stopSelfResult(1) = false
requestCode = 1, resultCode = 200

MyRun1 завершил работу и отправил соответствующее сообщение: requestCode = TASK1_CODE, resultCode = STATUS_FINISH.

requestCode = 3, resultCode = 200
MyRun#3 end, stopSelfResult(3) = true

MyRun3 завершил работу и отправил соответствующее сообщение: requestCode = TASK3_CODE, resultCode = STATUS_FINISH.

MyService onDestroy

Последний поступивший вызов выполнил метод stopSelfResult, сервис останавливается.

Изначально хотел пример попроще сделать, но чет увлекся, и получилось посложнее, но и поинтереснее.

Еще раз проговорю, что мы здесь поверхностно использовали PendingIntent и не стали копать его подробно, т.к. урок не о нем. Скоро мы еще раз встретимся с этим объектом, когда будем изучать уведомления (Notifications).

На следующем уроке:

- получаем из сервиса результат с помощью BroadcastReceiver

Урок 96. Service. Обратная связь с помощью BroadcastReceiver

В этом уроке:

- получаем из сервиса результат с помощью BroadcastReceiver

В прошлом уроке мы использовали PendingIntent для получения обратной связи от сервиса. В этом уроке для этой же цели используем BroadcastReceiver.

Схема такая:

- в Activity создаем BroadcastReceiver, а также создаем IntentFilter, настроенный на определенный Action, и регистрируем (включаем) эту пару. Теперь BroadcastReceiver будет получать Intent-ы подходящие под условия IntentFilter

- в сервисе, когда нам понадобится передать данные в Activity, мы создаем Intent (с Action из предыдущего пункта), кладем в него данные, которые хотим передать, и посылаем его на поиски BroadcastReceiver

- BroadcastReceiver в Activity ловит этот Intent и извлекает из него данные

Т.е. тут все аналогично вызовам Activity с использованием Action и IntentFilter. Если Action в Intent (отправленном из сервиса) и в IntentFilter (у BroadcastReceiver в Activity) совпадут, то BroadcastReceiver получит этот Intent и сможет извлечь данные для Activity.

Пример сделаем полностью аналогичный прошлому уроку. У нас будет приложение, которое будет отправлять в сервис на выполнение три задачи. А сервис будет информировать, когда он начал каждую задачу выполнять, когда закончил и с каким результатом. Все это будем выводить на экран Activity.

Создадим проект:

Project name: P0961_ServiceBackBroadcast
Build Target: Android 2.3.3
Application name: ServiceBackBroadcast
Package name: ru.startandroid.develop.p0951servicebackbroadcast
Create Activity: MainActivity

Добавим в **strings.xml** строки:

```
    <string  name="start">Start</string>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <TextView
         android:id="@+id/tvTask1"
```

```
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="">
        </TextView>
        <TextView
            android:id="@+id/tvTask2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="">
        </TextView>
        <TextView
            android:id="@+id/tvTask3"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="">
        </TextView>
        <Button
            android:id="@+id/btnStart"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClickStart"
            android:text="@string/start">
        </Button>
    </LinearLayout>
```

Три TextView, в которые будем выводить инфу, поступающую из сервиса. И кнопка старта сервиса.

Создаем класс для сервиса **MyService.java**. И пропишем его в манифесте. Пока в нем ничего не кодим.

MainActivity.java:

```
package ru.startandroid.develop.p0961servicebackbroadcast;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  final int TASK1_CODE = 1;
  final int TASK2_CODE = 2;
  final int TASK3_CODE = 3;

  public final static int STATUS_START = 100;
  public final static int STATUS_FINISH = 200;
```

```
public final static String PARAM_TIME = "time";
public final static String PARAM_TASK = "task";
public final static String PARAM_RESULT = "result";
public final static String PARAM_STATUS = "status";

public final static String BROADCAST_ACTION = "ru.startandroid.develop.p0961servicebackbroadcast";


TextView tvTask1;
TextView tvTask2;
TextView tvTask3;
BroadcastReceiver br;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  tvTask1 = (TextView) findViewById(R.id.tvTask1);
  tvTask1.setText("Task1");
  tvTask2 = (TextView) findViewById(R.id.tvTask2);
  tvTask2.setText("Task2");
  tvTask3 = (TextView) findViewById(R.id.tvTask3);
  tvTask3.setText("Task3");

  // создаем BroadcastReceiver
  br = new BroadcastReceiver() {
    // действия при получении сообщений
    public void onReceive(Context context, Intent intent) {
      int task = intent.getIntExtra(PARAM_TASK, 0);
      int status = intent.getIntExtra(PARAM_STATUS, 0);
      Log.d(LOG_TAG, "onReceive: task = " + task + ", status = " + status);

      // Ловим сообщения о старте задач
      if (status  == STATUS_START) {
        switch (task) {
        case TASK1_CODE:
          tvTask1.setText("Task1 start");
          break;
        case TASK2_CODE:
          tvTask2.setText("Task2 start");
          break;
        case TASK3_CODE:
          tvTask3.setText("Task3 start");
          break;
        }
      }

      // Ловим сообщения об окончании задач
      if (status == STATUS_FINISH) {
        int result = intent.getIntExtra(PARAM_RESULT, 0);
        switch (task) {
        case TASK1_CODE:
          tvTask1.setText("Task1 finish, result = " + result);
          break;
        case TASK2_CODE:
          tvTask2.setText("Task2 finish, result = " + result);
          break;
        case TASK3_CODE:
          tvTask3.setText("Task3 finish, result = " + result);
```

```
            break;
          }
        }
      }
    };
    // создаем фильтр для BroadcastReceiver
    IntentFilter intFilt = new IntentFilter(BROADCAST_ACTION);
    // регистрируем (включаем) BroadcastReceiver
    registerReceiver(br, intFilt);
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    // дерегистрируем (выключаем) BroadcastReceiver
    unregisterReceiver(br);
  }



  public void onClickStart(View v) {
    Intent intent;

    // Создаем Intent для вызова сервиса,
    // кладем туда параметр времени и код задачи
    intent = new Intent(this, MyService.class).putExtra(PARAM_TIME, 7)
        .putExtra(PARAM_TASK, TASK1_CODE);
    // стартуем сервис
    startService(intent);

    intent = new Intent(this, MyService.class).putExtra(PARAM_TIME, 4)
        .putExtra(PARAM_TASK, TASK2_CODE);
    startService(intent);

    intent = new Intent(this, MyService.class).putExtra(PARAM_TIME, 6)
        .putExtra(PARAM_TASK, TASK3_CODE);
    startService(intent);
  }

}
```

В **onCreate** находим TextView и присваиваем им начальные тексты. Далее создаем BroadcastReceiver и реализуем в нем метод onReceive. Все Intent-ы, которые получит BroadcastReceiver, будут переданы в этот метод нам на обработку. Мы извлекаем из Intent-а данные о задаче (код и статус) и меняем информацию о ней в соответствующем TextView. Если пришел статус STATUS_START – задача начала работу. Если STATUS_FINISH – закончила работу и Intent должен содержать результат (PARAM_RESULT).

Далее мы создаем IntentFilter и настраиваем его на Action = MainActivity.BROADCAST_ACTION. В сервисе мы будем создавать Intent с тем же Action и отправлять на поиски. В итоге они должны состыковаться.

Регистрируем BroadcastReceiver методом registerReceiver, передаем туда IntentFilter. Теперь BroadcastReceiver включен и ждет подходящих Intent.

В методе **onDestroy** мы дерегистрируем (выключаем) BroadcastReceiver методом unregisterReceiver.

В **onClickStart** мы создаем Intent-ы, помещаем в них данные о длительности паузы и код задачи и отправляем в сервис.

Теперь кодим сервис.

MyService.java:

```
package ru.startandroid.develop.p0961servicebackbroadcast;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {

  final String LOG_TAG = "myLogs";
  ExecutorService es;

  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "MyService onCreate");
    es = Executors.newFixedThreadPool(2);
  }

  public void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "MyService onDestroy");
  }

  public int onStartCommand(Intent intent, int flags, int startId) {
    Log.d(LOG_TAG, "MyService onStartCommand");

    int time = intent.getIntExtra(MainActivity.PARAM_TIME, 1);
    int task = intent.getIntExtra(MainActivity.PARAM_TASK, 0);

    MyRun mr = new MyRun(startId, time, task);
    es.execute(mr);

    return super.onStartCommand(intent, flags, startId);
  }

  public IBinder onBind(Intent arg0) {
    return null;
  }

  class MyRun implements Runnable {

    int time;
    int startId;
    int task;

    public MyRun(int startId, int time, int task) {
      this.time = time;
      this.startId = startId;
      this.task = task;
      Log.d(LOG_TAG, "MyRun#" + startId + " create");
    }
```

```
    public void run() {
      Intent intent = new Intent(MainActivity.BROADCAST_ACTION);
      Log.d(LOG_TAG, "MyRun#" + startId + " start, time = " + time);
      try {
        // сообщаем об старте задачи
        intent.putExtra(MainActivity.PARAM_TASK, task);
        intent.putExtra(MainActivity.PARAM_STATUS, MainActivity.STATUS_START);
        sendBroadcast(intent);

        // начинаем выполнение задачи
        TimeUnit.SECONDS.sleep(time);

        // сообщаем об окончании задачи
        intent.putExtra(MainActivity.PARAM_STATUS, MainActivity.STATUS_FINISH);
        intent.putExtra(MainActivity.PARAM_RESULT, time * 100);
        sendBroadcast(intent);

      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      stop();
    }

    void stop() {
      Log.d(LOG_TAG, "MyRun#" + startId + " end, stopSelfResult("
          + startId + ") = " + stopSelfResult(startId));
    }
  }
}
```

Как и в прошлом уроке, используем экзекутор (на два потока) для параллельного выполнения задач.

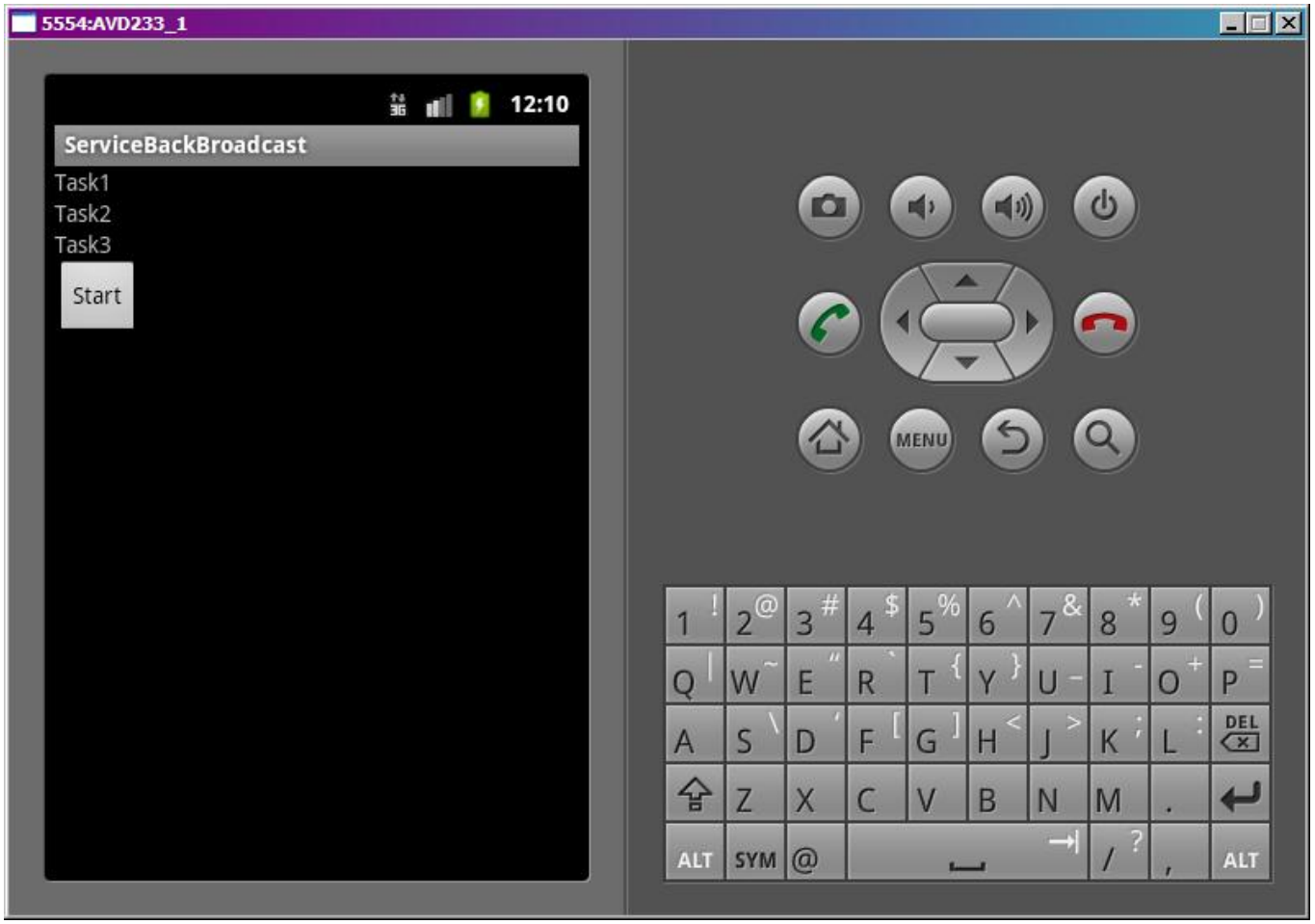
В методе run класса **MyRun** будем, как обычно, ставить паузу и сообщать в Activity о начале и завершении задачи.

Чтобы отправить данные в Activity, создаем Intent с Action = MainActivity.BROADCAST_ACTION и помещаем в него данные, которые хотим передать. Чтобы передать информацию о том, что задача начала работать, мы передаем код задачи (task) и статус начала (MainActivity.STATUS_START). И методом sendBroadcast отправляем Intent искать подходящий BroadcastReceiver. Он найдется в нашем Activity, обработает Intent и обновит инфу о задачах в TextView.

Чтобы передать информацию о том, что задача закончила работу, мы передаем статус завершения (MainActivity. STATUS_FINISH) и результат (время * 100). task в Intent не пишем, т.к. он ранее уже был записан (при первой отправке, в начале работы задачи). Методом sendBroadcast отправляем Intent искать подходящий BroadcastReceiver. Он найдется в нашем Activity, обработает Intent и обновит инфу о задачах в TextView.

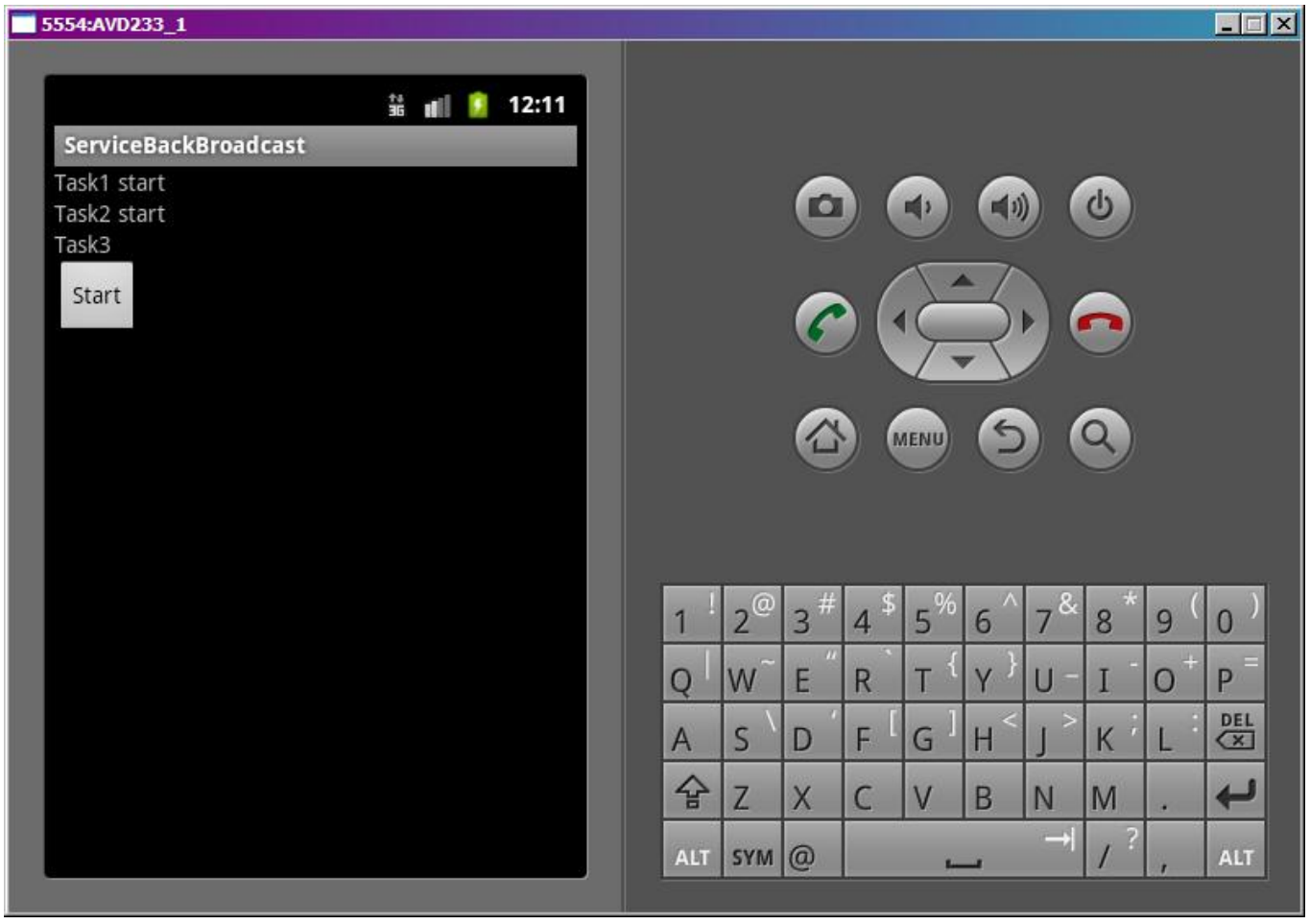
После всего этого вызываем stopSelfResult.

Все сохраняем и запускаем приложение.

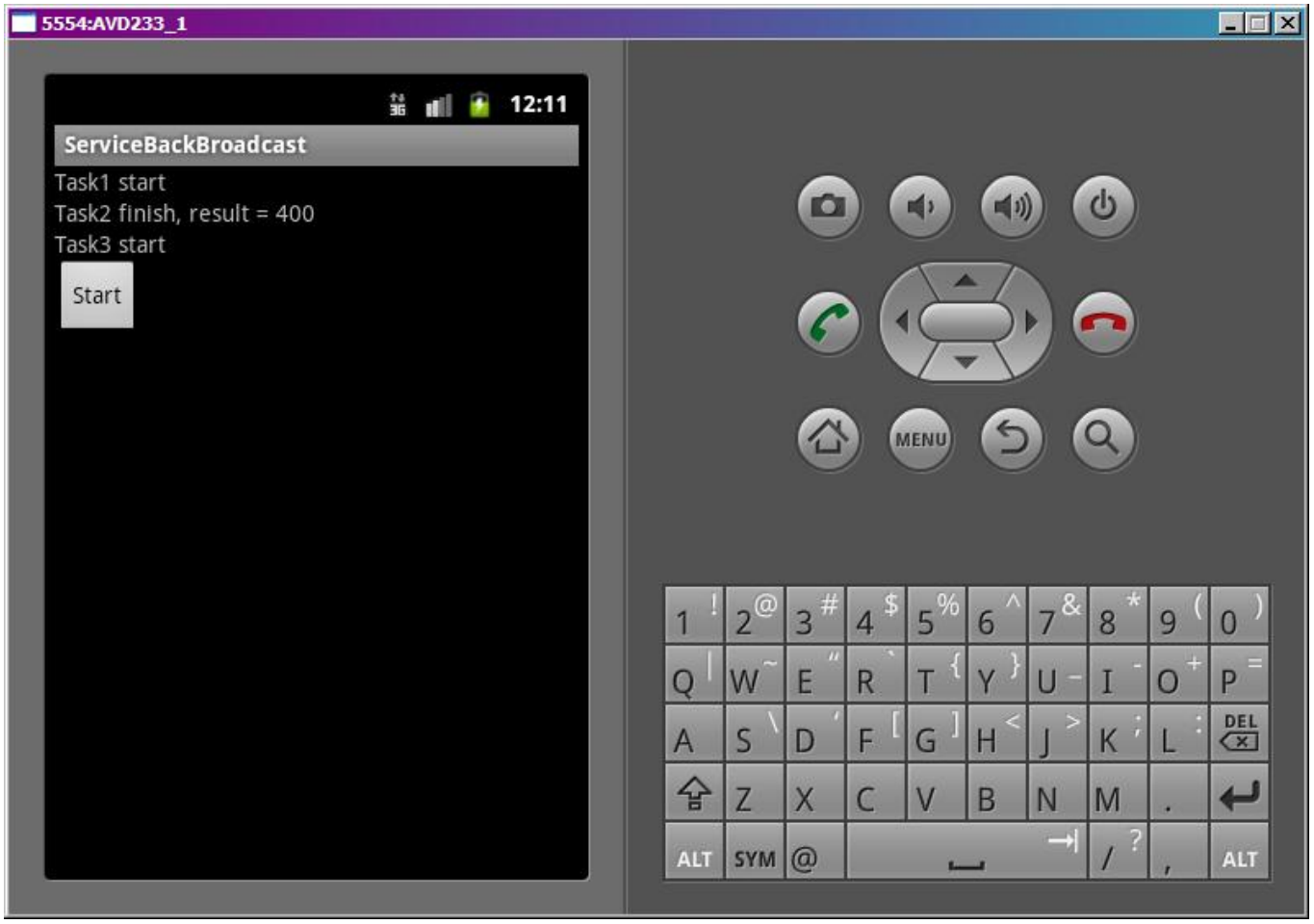


Жмем **Start**.

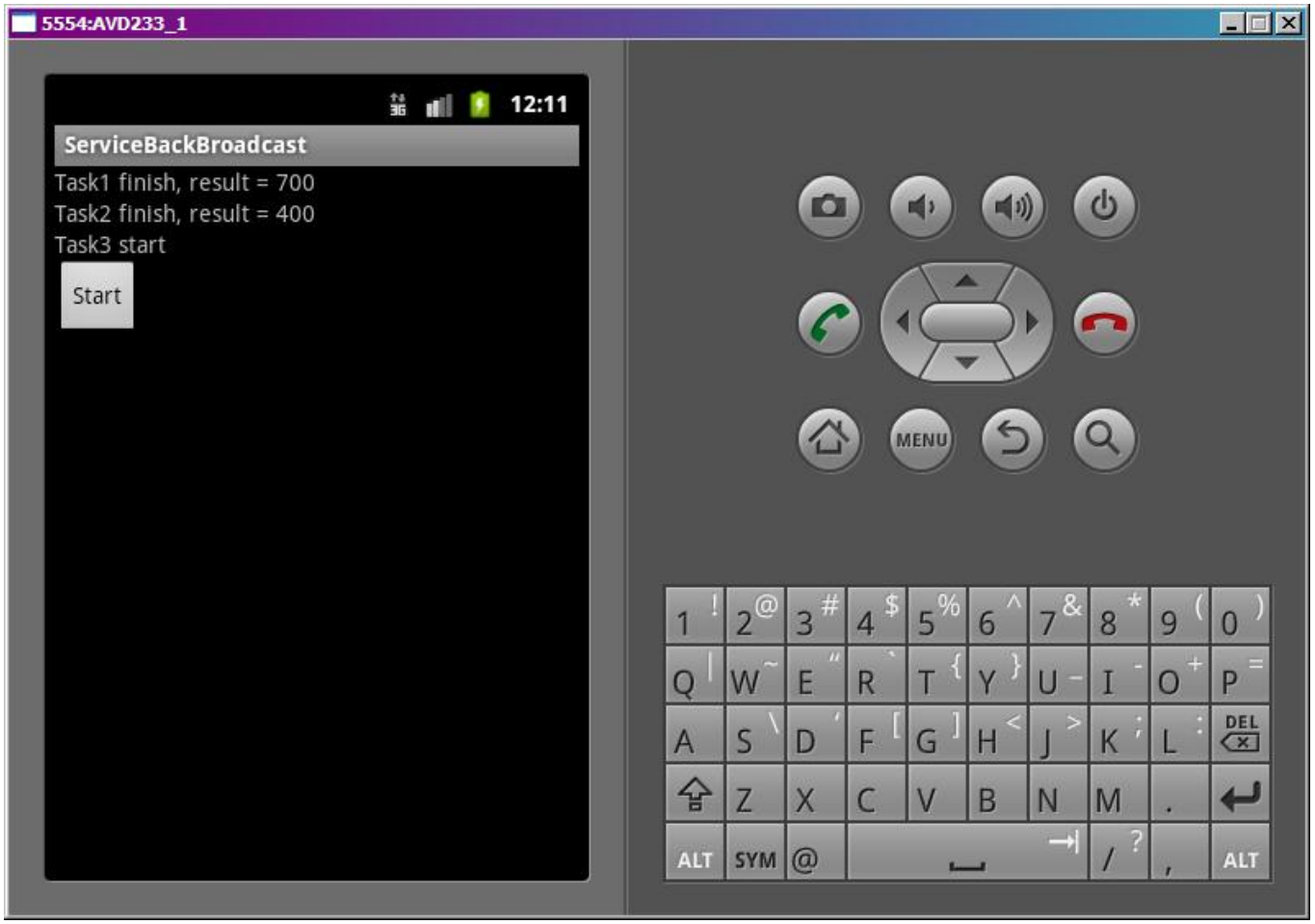
Видим, что две задачи начали работать, т.к. экзекутор настроен на два потока.



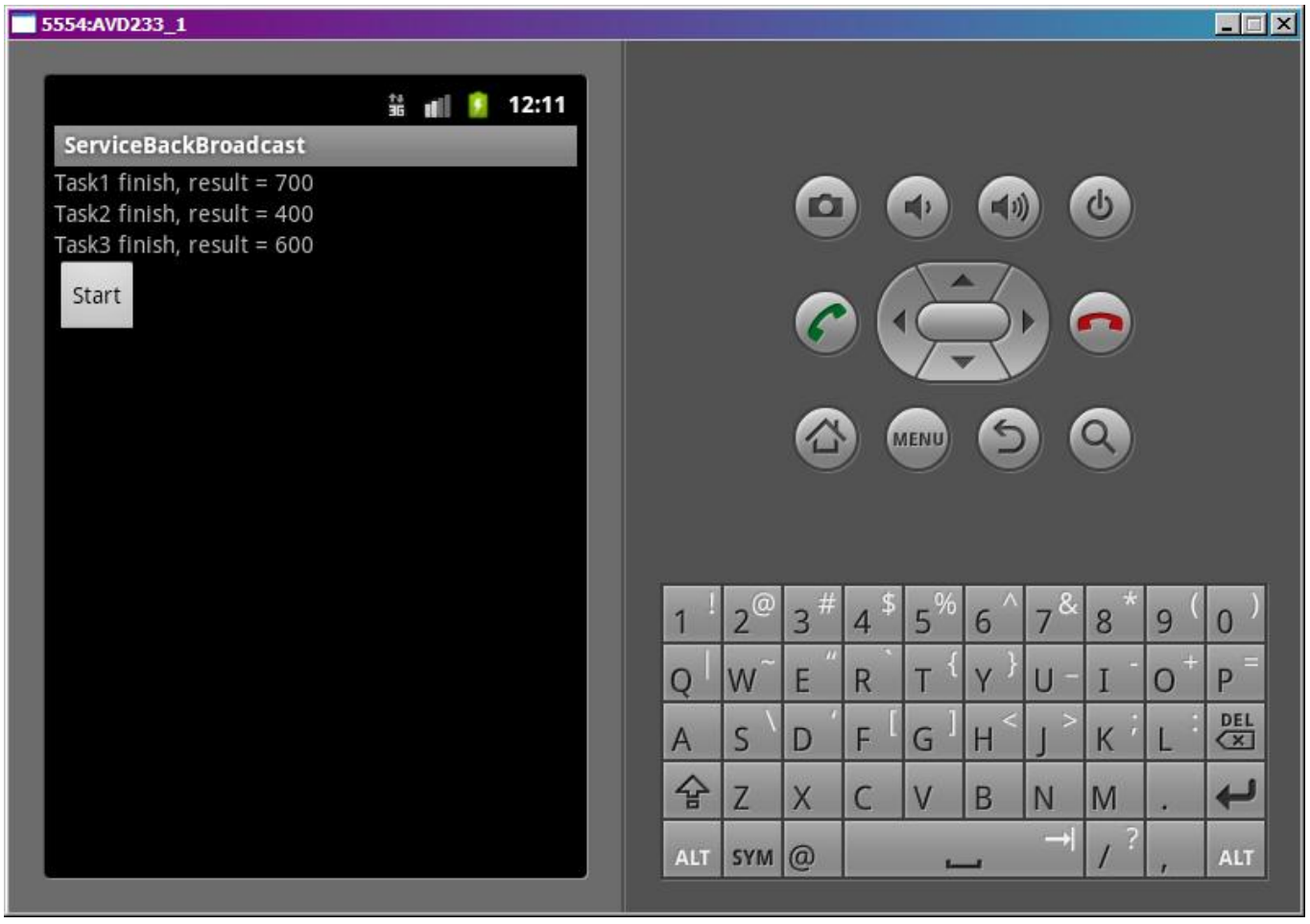
Одна задача завершилась и показала результат, поток освободился, стартует оставшаяся задача.



Еще одна задача завершилась.



Последняя завершилась.



Смотрим логи (т.к. используем потоки, у вас может быть немного другая последовательность записей в логах):

MyService onCreate
MyService onStartCommand
MyRun#1 create
MyService onStartCommand
MyRun#2 create
MyRun#1 start, time = 7
MyService onStartCommand
MyRun#3 create

Сервис создался и получил все три вызова.

onReceive: task = 1, status = 100
MyRun#2 start, time = 4
onReceive: task = 2, status = 100

В Activity получаем сообщение о том, что первая (task = 1) и вторая (task = 2) задачи начали работать (status = 100)

onReceive: task = 2, status = 200
MyRun#2 end, stopSelfResult(2) = false

MyRun#2 завершена и в Activity получаем сообщение о том, что вторая (task = 2) задача закончила работать (status = 200)

MyRun#3 start, time = 6
onReceive: task = 3, status = 100

MyRun#3 начала работать в освободившемся после MyRun#2 потоке. В Activity получаем сообщение о том, что третья (task = 3) задача начала работать (status = 100)

MyRun#1 end, stopSelfResult(1) = false
onReceive: task = 1, status = 200

MyRun#1 завершена и в Activity получаем сообщение о том, что первая (task = 1) задача закончила работать (status = 200)

onReceive: task = 3, status = 200
MyRun#3 end, stopSelfResult(3) = true

MyRun#3 завершена и в Activity получаем сообщение о том, что третья (task = 3) задача закончила работать (status = 200)

MyService onDestroy

Сервис закрылся.

Разумеется, моя схема нумерации задач и статусов взята из головы. Вы можете придумать и использовать свои какие угодно статусы. Я только показал еще один механизм, как можно получать и обрабатывать данные из сервиса.

Ну и регистрируете и дерегистрируете BroadcastReceiver необязательно в onCreate и onDestroy. Делаете там, где это необходимо.

На следующем уроке:

- используем биндинг для подключения к сервису

Урок 97. Service. Биндинг. ServiceConnection

В этом уроке:

- используем биндинг для подключения к сервису

В прошлых уроках мы общались с сервисом **асинхронно**. Т.е. мы отправляли запрос через startService, а ответ нам приходил когда-нибудь потом посредством PendingIntent или BroadcastReceiver.

Но есть и **синхронный** способ взаимодействия с сервисом. Он достигается с помощью биндинга (binding, я также буду использовать слово «подключение»). Мы подключаемся к сервису и можем взаимодействовать с ним путем обычного вызова методов с передачей данных и получением результатов. В этом уроке передавать данные не будем. Пока что разберемся, как подключаться и отключаться.

Как вы помните, для запуска и остановки сервиса мы использовали методы startService и stopService. Для биндинга используются методы bindService и unbindService.

Создадим два Application. В одном будет приложение, в другом сервис.

Создадим первый проект:

Project name: P0971_ServiceBindClient
Build Target: Android 2.3.3
Application name: ServiceBindClient
Package name: ru.startandroid.develop.p0971servicebindclient
Create Activity: MainActivity

Добавим в **strings.xml** строки:

```
    <string  name="start">Start</string>
    <string  name="stop">Stop</string>
    <string  name="bind">Bind</string>
    <string  name="unbind">Unbind</string>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <Button
         android:id="@+id/btnStart"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:onClick="onClickStart"
         android:text="@string/start">
    </Button>
```

```
    <Button
          android:id="@+id/btnStop"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onClickStop"
          android:text="@string/stop">
    </Button>
    <Button
          android:id="@+id/btnBind"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onClickBind"
          android:text="@string/bind">
    </Button>
    <Button
          android:id="@+id/btnUnBind"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onClickUnBind"
          android:text="@string/unbind">
    </Button>
</LinearLayout>
```

4 кнопки: для запуска, остановки и биндинга сервиса

MainActivity.java:

```
package ru.startandroid.develop.p0971servicebindclient;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;
import android.view.View;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  boolean bound = false;
  ServiceConnection sConn;
  Intent intent;

  /** Called when the activity is first created. */
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    intent = new Intent("ru.startandroid.develop.p0972servicebindserver.MyService");

    sConn = new ServiceConnection() {
      public void onServiceConnected(ComponentName name, IBinder binder) {
```

```
        Log.d(LOG_TAG, "MainActivity onServiceConnected");
        bound = true;
      }

      public void onServiceDisconnected(ComponentName name) {
        Log.d(LOG_TAG, "MainActivity onServiceDisconnected");
        bound = false;
      }
    };
  }

  public void onClickStart(View v) {
    startService(intent);
  }

  public void onClickStop(View v) {
    stopService(intent);
  }

  public void onClickBind(View v) {
    bindService(intent, sConn, BIND_AUTO_CREATE);
  }

  public void onClickUnBind(View v) {
    if (!bound) return;
    unbindService(sConn);
    bound = false;
  }

  protected void onDestroy() {
    super.onDestroy();
    onClickUnBind(null);
  }
}
```

В **onCreate** мы создаем Intent, который позволит нам добраться до сервиса.

Объект ServiceConnection позволит нам определить, когда мы подключились к сервису и когда связь с сервисом потеряна (если сервис был убит системой при нехватке памяти). При подключении к сервису сработает метод onServiceConnected. На вход он получает имя компонента-сервиса и объект Binder для взаимодействия с сервисом. В этом уроке мы этим Binder пока не пользуемся. При потере связи сработает метод onServiceDisconnected.

Переменную bound мы используем для того, чтобы знать – подключены мы в данный момент к сервису или нет. Соответственно при подключении мы переводим ее в true, а при потере связи в false.

Далее идут обработчики кнопок. В **onClickStart** мы стартуем сервис, в **onClickStop** – останавливаем.

В **onClickBind** – соединяемся с сервисом, используя метод bindService. На вход передаем Intent, ServiceConnection и флаг BIND_AUTO_CREATE, означающий, что, если сервис, к которому мы пытаемся подключиться, не работает, то он будет запущен.

В **onClickUnBind** с помощью bound проверяем, что соединение уже установлено. Далее отсоединяемся методом unbindService, на вход передавая ему Intent. И в bound пишем false, т.к. мы сами разорвали соединение. Метод onServiceDisconnected не сработает при явном отключении.

Создадим второй проект, без Activity:

Project name: P0972_ServiceBindServer
Build Target: Android 2.3.3
Application name: ServiceBindServer
Package name: ru.startandroid.develop.p0972servicebindserver

Создаем сервис **MyService.java**:

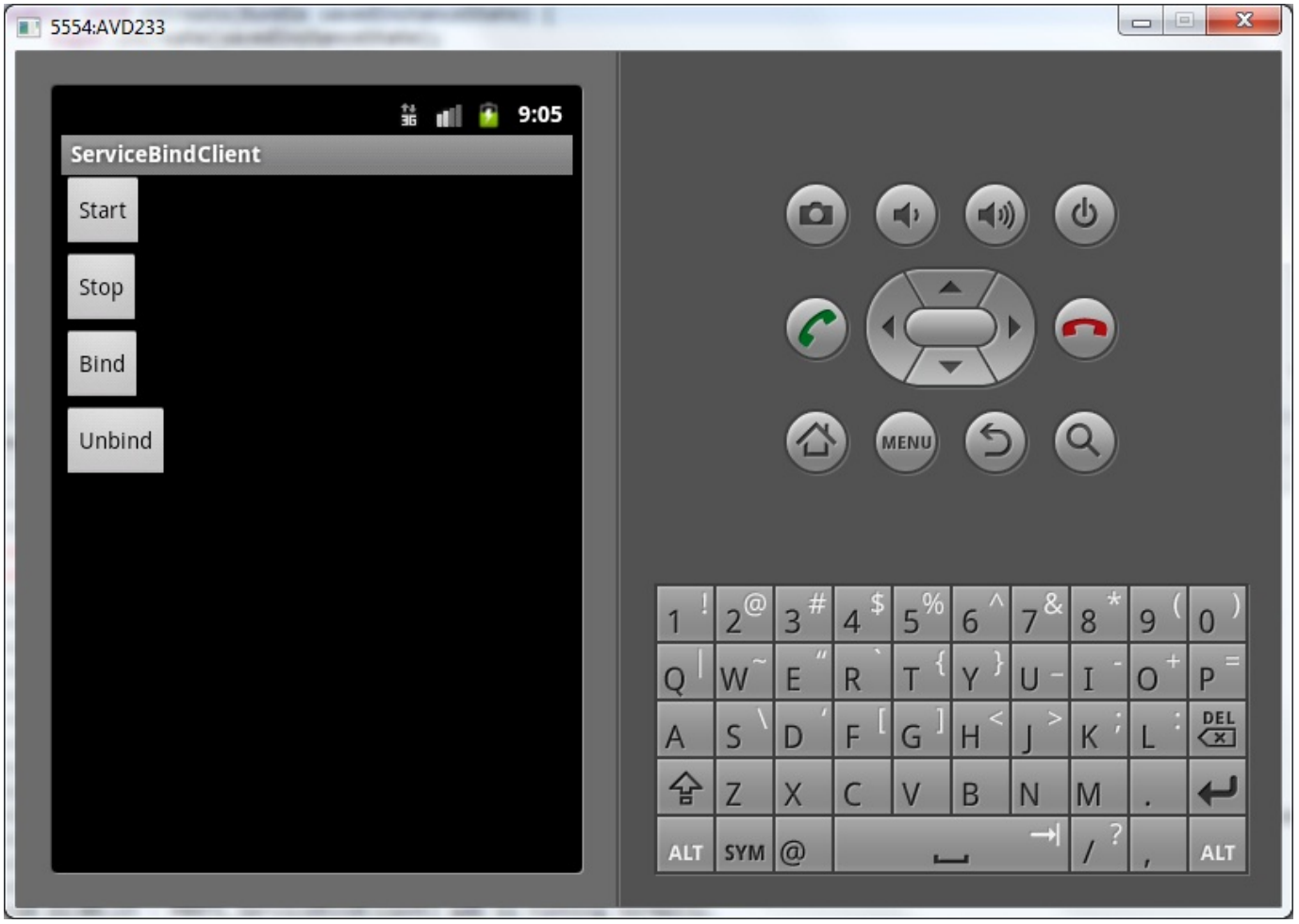
```
package ru.startandroid.develop.p0972servicebindserver;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {

  final String LOG_TAG = "myLogs";

  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "MyService onCreate");
  }

  public IBinder onBind(Intent intent) {
    Log.d(LOG_TAG, "MyService onBind");
    return new Binder();
  }

  public void onRebind(Intent intent) {
    super.onRebind(intent);
    Log.d(LOG_TAG, "MyService onRebind");
  }

  public boolean onUnbind(Intent intent) {
    Log.d(LOG_TAG, "MyService onUnbind");
    return super.onUnbind(intent);
  }

  public void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "MyService onDestroy");
  }
}
```

Методы **onCreate** и **onDestroy** нам знакомы – они вызываются при создании и уничтожении сервиса. А onBind, onRebind и onUnbind используются при биндинге и мы далее будем смотреть как именно. Метод onStartCommand не используем.

В методе **onBind** возвращаем пока объект-заглушку Binder. В этом уроке он не будет использован.

Прописываем сервис в манифесте и настраиваем для него IntentFilter с Action = *ru.startandroid.develop.p0972servicebindserver.MyService*.

Все сохраняем, инсталлим сервис и запускаем приложение.



На самом экране ничего происходить не будет. Все внимание на логи.

Попробуем подключиться к неработающему сервису. Жмем **Bind**. В логах:

MyService onCreate
MyService onBind
MainActivity onServiceConnected

Сервис создался и сработал его метод **onBind**. Также сработал метод **onServiceConnected** в ServiceConnection, т.е. Activity теперь знает, что подключение к сервису установлено.

Попробуем отключиться. Жмем **Undind**.

MyService onUnbind
MyService onDestroy

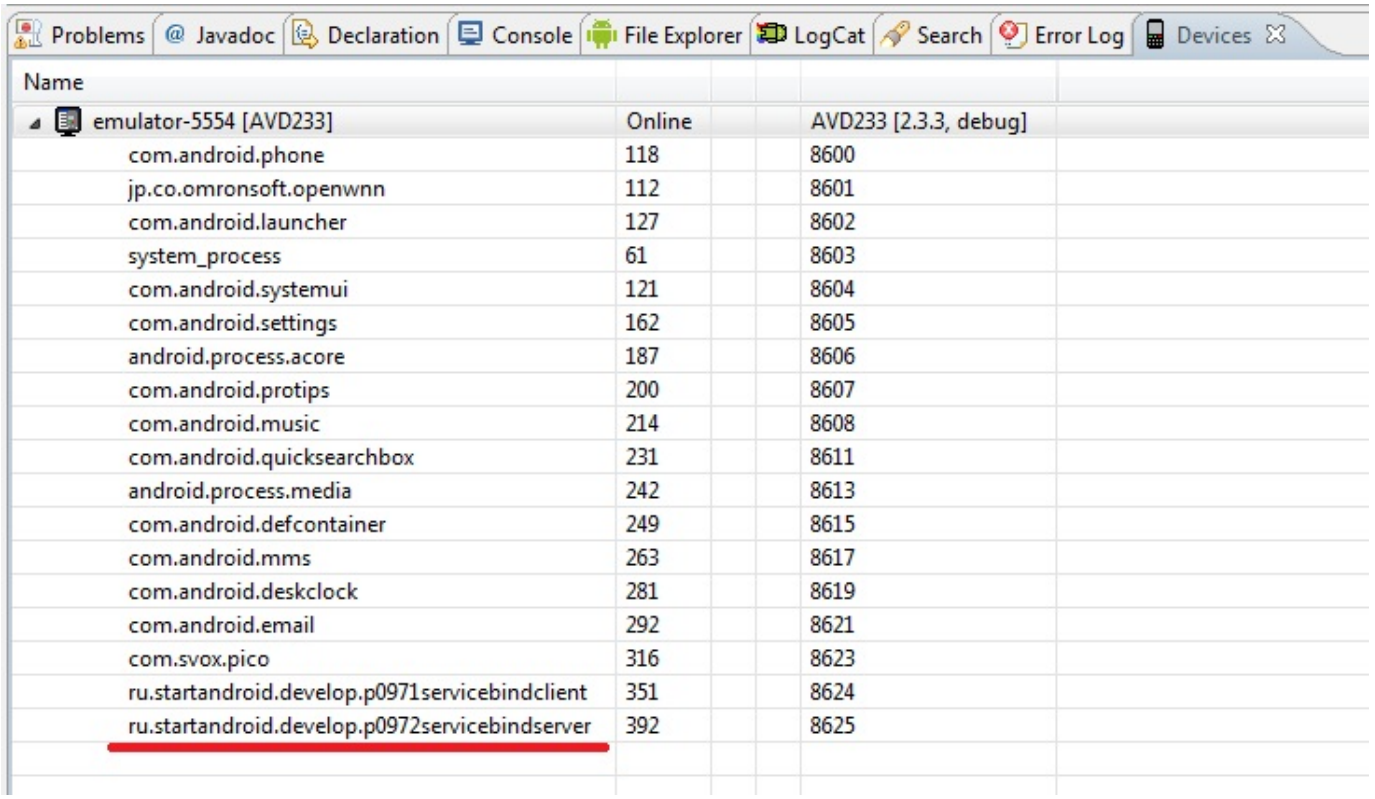
Сработал метод **Unbind** в сервисе и сервис закрылся. Т.е. если мы биндингом запустили сервис, он будет жить, пока живет соединение. Как только мы отключаемся, сервис останавливается. onServiceDisconnected не сработал, т.к. мы сами отключились.

Теперь попробуем соединиться с сервисом и убить его. Посмотрим, что станет с соединением. Жмем **Bind**.

MyService onCreate
MyService onBind
MainActivity onServiceConnected

Подключились.

Теперь в процессах убиваем процесс сервиса



Смотрим лог:

MainActivity onServiceDisconnected

Сработал метод **onServiceDisconnected** объекта ServiceConnection. Тем самым Activity уведомлено, что соединение разорвано.

Теперь немного подождем (у меня это заняло 5 секунд) и в логах появляются строки:

MyService onCreate
MyService onBind
MainActivity onServiceConnected

Сервис запустился и соединение восстановилось. Очень удобно.

Жмем **Unbind** и отключаемся.

Попробуем снова подключиться к незапущенному сервису, но уже без флага BIND_AUTO_CREATE. Перепишем **onClickBind** в **MainActivity.java**:

```
public void onClickBind(View v) {
  bindService(intent, sConn, 0);
}
```

Вместо флага BIND_AUTO_CREATE мы написали 0.

Сохраняем, запускаем. Жмем **Bind**. В логах ничего. Мы убрали флаг, сервис сам не создается при подключении.

Давайте запустим его методом startService. Жмем **Start**. В логах:

MyService onCreate
MyService onBind
MainActivity onServiceConnected

Сервис создался и приложение подключилось к нему. Т.е. попыткой биндинга мы оставили некую «заявку» на подключение, и когда сервис был запущен методом startService, он эту заявку увидел и принял подключение.

Жмем **UnBind**.

MyService onUnbind

Отключились от сервиса. Но сервис продолжает жить, потому что он был запущен не биндингом, а методом startService. А там уже свои правила закрытия сервиса. Это мы проходили в прошлых уроках.

Жмем **Stop**.

MyService onDestroy

Сервис завершил работу.

Попробуем запустить сервис методом **startService** и, пока он работает, несколько раз подключимся и отключимся. Жмем **Start**.

MyService onCreate

Сервис запущен.

Подключаемся и отключаемся, т.е. жмем **Bind**

MyService onBind
MainActivity onServiceConnected

а затем **Unbind**.

MyService onUnbind

Сработали методы **onBind** и **onUnbind** в сервисе, и **onServiceConnected** в ServiceConnection.

Еще раз подключаемся и отключаемся - жмем **Bind**, а затем **Unbind**

MainActivity onServiceConnected

При повторном подключении к сервису методы **onBind** и **onUnbind** не сработали. Только **onServiceConnected**.

И далее, сколько бы мы не подключались, так и будет.

Остановим сервис – нажмем **Stop**.

Это поведение можно скорректировать. Для этого необходимо возвращать **true** в методе **onUnbind**. Сейчас мы там вызываем метод супер-класса, а он возвращает false.

Перепишем метод **Unbind** в **MyService.java**:

```
public boolean onUnbind(Intent intent) {
  Log.d(LOG_TAG, "MyService onUnbind");
  return true;
}
```

Сохраним и инсталлим сервис. Жмем **Start**, а затем жмем поочередно **Bind** и **Unbind**, т.е. подключаемся и отключаемся. Смотрим логи:

MyService onCreate
MyService onBind
MainActivity onServiceConnected
MyService onUnbind

Сервис создан, подключились и отключились. Продолжаем подключаться и отключаться.

MyService onRebind
MainActivity onServiceConnected
MyService onUnbind
MyService onRebind
MainActivity onServiceConnected
MyService onUnbind

Последующие подключения и отключения сопровождаются вызовами методов **onRebind** и **onUnbind**. Таким образом, у нас есть возможность обработать в сервисе каждое повторное подключение/отключение.

Вот примерно такой **Lifecycle** имеет **биндинг** сервиса. Разумеется, я рассмотрел не все возможные комбинации запуска методов startService, stopService, bindService и unbindService. Оставляю это вам, если есть интерес. Я рассмотрел только типичные случаи.

Также я не рассматривал возможность одновременного подключения нескольких приложений или сервисов к одному сервису. А такая возможность существует. В этом случае, если сервис запущен биндингом, то он будет жить, пока не отключатся все подключившиеся.

Насколько я понял по хелпу, не рекомендуется оставлять незавершенные подключения к сервисам по окончании работы приложения. В хелповских примерах подключение к сервису производится в методе onStart, а отключение - в onStop. Но, разумеется, если вам надо, чтобы подключение оставалось при временном уходе Activity в background, то используйте onCreate и onDestroy. Также есть четкая рекомендация от гугла - не использовать для этих целей onResume и onPause.

На следующем уроке:

- обмен данными в биндинге

Урок 98. Service. Локальный биндинг

В этом уроке:

- обмен данными в биндинге

Подключаться к сервису мы теперь умеем. В этом уроке попробуем обменяться с ним данными.

В сервисе метод onBind возвращает объект, наследующий интерфейс IBinder. Проще всего использовать для этого объект Binder и расширить его необходимыми нам методами. Т.е. создаем свой класс MyBinder с предком Binder и рисуем в нем свои методы.

При биндинге, в методе onServiceConnected мы получаем объект Binder. Мы можем привести его к типу MyBinder (из сервиса) и вызывать его методы, а реализация будет срабатывать в сервисе, где мы описывали этот класс.

Как вы понимаете, это сработает только, если сервис и приложение выполняются в одном процессе. Поэтому такой биндинг называется локальным.

Нарисуем пример. У нас будет сервис, который с определенным интервалом будет что-то выводить в лог. Мы к нему подключимся, и будем повышать или понижать интервал и получать новое значение интервала обратно.

Создадим проект:

Project name: P0981_ServiceBindingLocal
Build Target: Android 2.3.3
Application name: ServiceBindingLocal
Package name: ru.startandroid.develop.p0981servicebindinglocal
Create Activity: MainActivity

Добавим в **strings.xml** строки:

```
    <string  name="start">Start</string>
    <string  name="up">Up</string>
    <string  name="down">Down</string>
```

Экран **main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical">
    <TextView
         android:id="@+id/tvInterval"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="">
```

```
    </TextView>
    <Button
          android:id="@+id/btnStart"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onClickStart"
          android:text="@string/start">
    </Button>
    <Button
          android:id="@+id/btnUp"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onClickUp"
          android:text="@string/up">
    </Button>
    <Button
          android:id="@+id/btnDown"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:onClick="onClickDown"
          android:text="@string/down">
    </Button>
</LinearLayout>
```

Кнопки для запуска сервиса, повышения интервала и понижения интервала.

Создаем сервис **MyService.java**:

```
package ru.startandroid.develop.p0981servicebindinglocal;

import java.util.Timer;
import java.util.TimerTask;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service {

  final String LOG_TAG = "myLogs";

  MyBinder binder = new MyBinder();

  Timer timer;
  TimerTask tTask;
  long interval = 1000;

  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "MyService onCreate");
    timer = new Timer();
    schedule();
```

```
  }

  void schedule() {
    if (tTask != null) tTask.cancel();
    if (interval > 0) {
      tTask = new TimerTask() {
        public void run() {
          Log.d(LOG_TAG, "run");
        }
      };
      timer.schedule(tTask, 1000, interval);
    }
  }

  long upInterval(long gap) {
    interval = interval + gap;
    schedule();
    return interval;
  }

  long downInterval(long gap) {
    interval = interval - gap;
    if (interval < 0) interval = 0;
    schedule();
    return interval;
  }

  public IBinder onBind(Intent arg0) {
    Log.d(LOG_TAG, "MyService onBind");
    return binder;
  }

  class MyBinder extends Binder {
    MyService getService() {
      return MyService.this;
    }
  }
}
```

Здесь мы используем таймер – Timer. Он позволяет повторять какое-либо действие через заданный промежуток времени. Кратко распишу принцип действия. TimerTask – это задача, которую Timer будет периодически выполнять. В методе run – кодим действия этой задачи. И далее для объекта Timer вызываем метод schedule, в который передаем задачу TimerTask, время через которое начнется выполнение, и период повтора. Чтобы отменить выполнение задачи, необходимо вызвать метод cancel для TimerTask. Отмененную задачу нельзя больше запланировать, и если снова надо ее включить – необходимо создать новый экземпляр TimerTask и скормить его таймеру.

Итак, в методе **onCreate** мы создаем таймер и выполняем метод schedule, в котором стартует задача.

Метод **schedule** проверяет, что задача уже создана и отменяет ее. Далее планирует новую, с отложенным на 1000 мс запуском и периодом = interval. Т.е. можно сказать, что этот метод перезапускает задачу с использованием текущего интервала повтора (interval), а если задача еще не создана, то создает ее. Сама задача просто выводит в лог текст run. Если interval = 0, то ничего не делаем.

Метод **upInterval** получает на вход значение, увеличивает interval на это значение и перезапускает задачу. Соответственно задача после этого будет повторяться реже.

Метод **downInterval** получает на вход значение, уменьшает interval на это значение (но так, чтоб не меньше 0) и перезапускает задачу. Соответственно задача после этого будет повторяться чаще.

onBind возвращает binder. Это объект класса MyBinder.

MyBinder расширяет стандартный Binder, мы добавляем в него один метод getService. Этот метод возвращает наш сервис MyService.

Т.е. в подключаемом Activity, в методе onServiceConnected мы получим объект, который идет на выход метода onBind. Далее преобразуем его к типу MyBinder, вызовем getService и вуаля - у нас в Activity будет ссылка на объект-сервис MyService.

Кодим **MainActivity.java**:

```
package ru.startandroid.develop.p0981servicebindinglocal;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

  final String LOG_TAG = "myLogs";

  boolean bound = false;
  ServiceConnection sConn;
  Intent intent;
  MyService myService;
  TextView tvInterval;
  long interval;


  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tvInterval = (TextView) findViewById(R.id.tvInterval);
    intent = new Intent(this, MyService.class);
    sConn = new ServiceConnection() {

      public void onServiceConnected(ComponentName name, IBinder binder) {
        Log.d(LOG_TAG, "MainActivity onServiceConnected");
        myService = ((MyService.MyBinder) binder).getService();
        bound = true;
      }

      public void onServiceDisconnected(ComponentName name) {
        Log.d(LOG_TAG, "MainActivity onServiceDisconnected");
        bound = false;
      }
    };
  }
```

```
  @Override
  protected void onStart() {
    super.onStart();
    bindService(intent, sConn, 0);
  }

  @Override
  protected void onStop() {
    super.onStop();
    if (!bound) return;
    unbindService(sConn);
    bound = false;
  }

  public void onClickStart(View v) {
    startService(intent);
  }

  public void onClickUp(View v) {
    if (!bound) return;
    interval = myService.upInterval(500);
    tvInterval.setText("interval = " + interval);
  }

  public void onClickDown(View v) {
    if (!bound) return;
    interval = myService.downInterval(500);
    tvInterval.setText("interval = " + interval);
  }
}
```

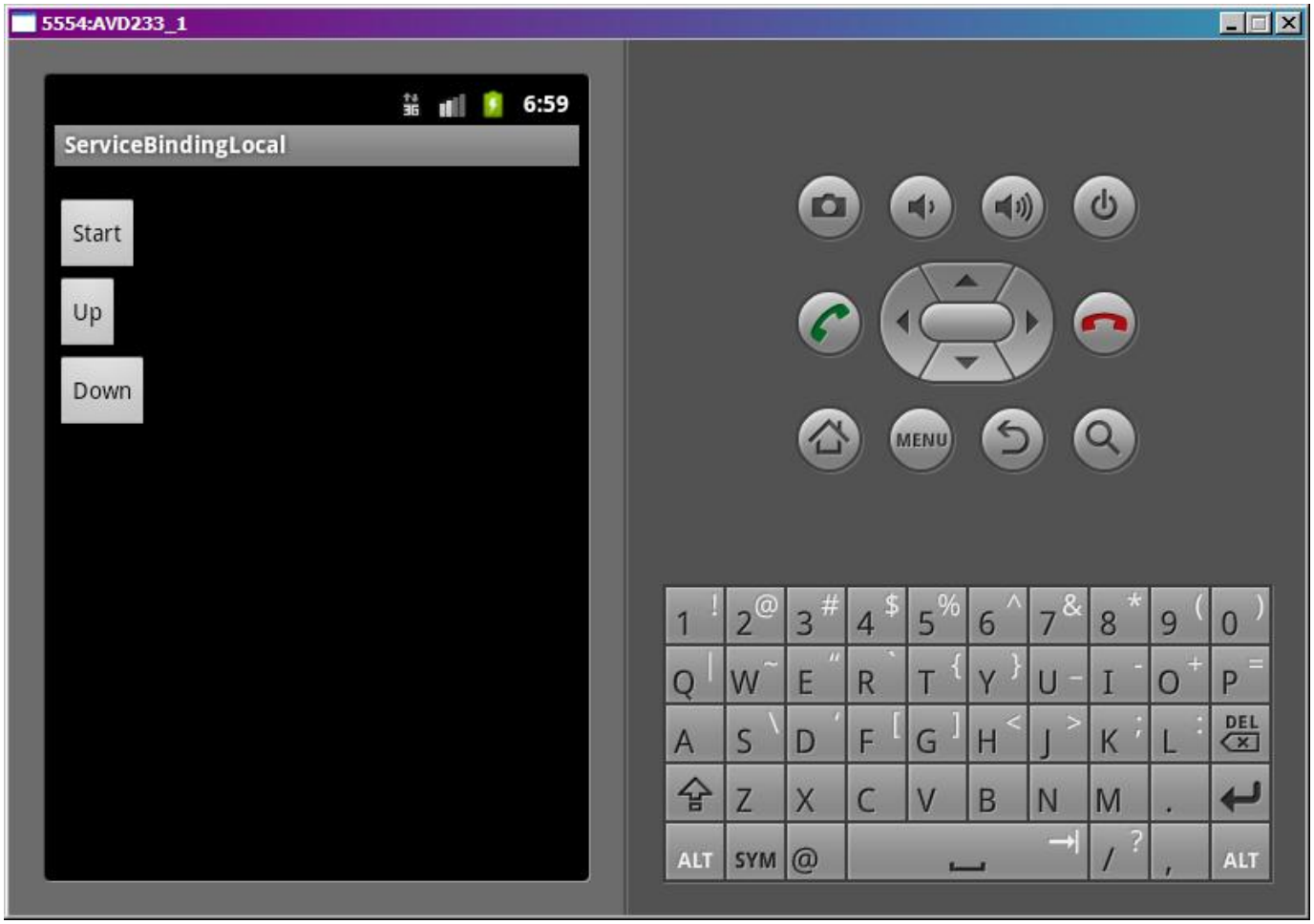
В **onCreate** создаем Intent для доступа к сервису и ServiceConnection. В методе onServiceConnected мы берем binder, преобразуем его к MyService.MyBinder, вызываем метод getService и получаем наш сервис MyService. Теперь мы можем выполнять методы сервиса. Нас интересуют методы увеличения и понижения интервала.

В методах **onStart** и **onStop** мы соответственно подключаемся к сервису и отключаемся.

В **onClickStart** запускаем сервис.

В **onClickUp** и **onClickDown** проверяем, что соединение с сервисом есть, и вызываем соответствующие методы сервиса для увеличения или понижения интервала. В качестве дельты изменения интервала передаем 500. В ответ эти методы передают нам новое значение интервала, и мы выводим его в TextView.

Все сохраним и запускаем.



Жмем **Start** – запускаем сервис. В логах:

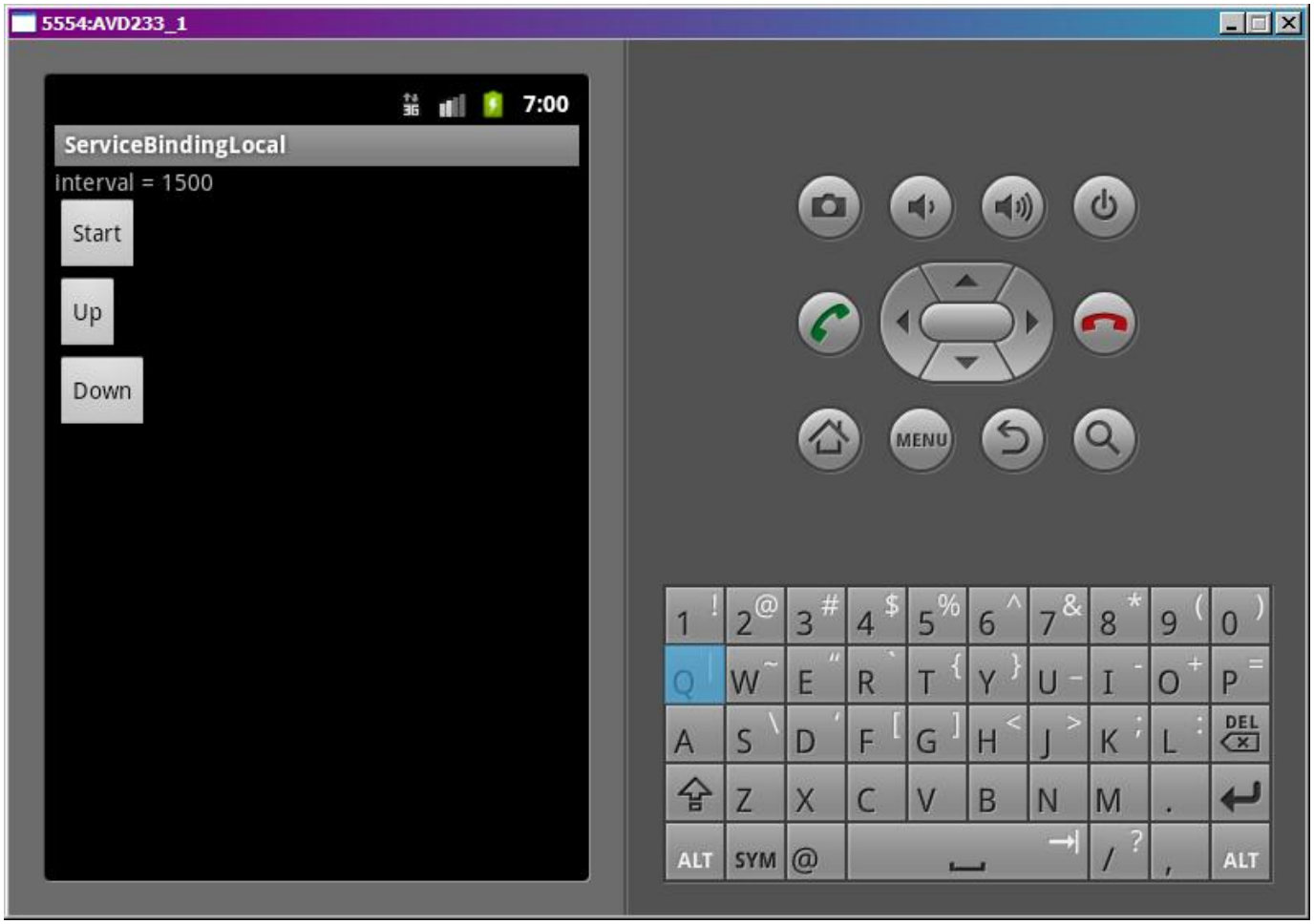
MyService onCreate
MyService onBind
MainActivity onServiceConnected

Создается сервис, и мы к нему подключаемся. Биндинг был ранее вызван в onStart и, когда сервис стартанул, мы автоматически подключились. В прошлом уроке мы разбирали такую ситуацию, когда биндинг идет к незапущенному пока сервису.

run
run
run
...
run

Далее с интервалом в одну секунду в лог падает текст run.

Попробуем увеличить интервал. Жмем **Up**.



На экране появилось текущее значение интервала. И по времени записей в логах видим, что именно с этим интервалом срабатывает задача (вывод текста run в лог) .

Далее можно нажимать **Up** и **Down** и наблюдать, как меняется интервал выполнения задачи. Таким образом, мы из Activity подключились к сервису и управляем его работой.

В начале урока я отметил, что все это будет работать, если приложение и сервис в одном процессе. Если же они в разных процессах, то используется немного более сложный вариант для обмена данными. Но это гораздо менее используемый случай, поэтому сейчас я про него не рассказываю. Но попозже об этом обязательно будет статья.

На следующем уроке:

- шлем уведомление из сервиса

Урок 99. Service. Уведомления - notifications

В этом уроке:

- шлем уведомление из сервиса

В принципе, **уведомления** – отдельная от **сервисов** тема. Но чаще всего уведомления используются именно в сервисах, поэтому я решил дать эту тему сейчас.

В андроид (версии < 3) есть строка вверху экрана. Называется она статус-бар. Туда обычно в виде иконок сваливаются различные уведомления для пользователя (новые письма, смс и прочие). Пользователь открывает статус бар – видит там чуть более подробную инфу о событии. Дальше он может либо стереть это уведомление, либо нажать на него и перейти непосредственно к событию.

В этом уроке научимся все это проделывать. Для начала разберем уведомление на логические составляющие, чтобы проще было понять как его создавать и отправлять.

Первая часть – то, что видно в статус-баре, когда уведомление только приходит – иконка и текст. Текст потом исчезает и остается только иконка.

Вторая часть – то, что мы видим, когда открываем статус бар (тянем вниз). Там уже полноценный View с иконкой и двумя текстами, т.е. более подробная информация о событии.

Третья часть – то, что произойдет, если мы нажмем на View из второй части. Тут обычно идет вызов Activity, где мы можем просмотреть полную информацию и обработать событие.

Кроме этого есть еще несколько возможностей, по которым совсем кратко пробежимся в конце урока.

Создадим приложение и сервис. Сервис, как будто загружает файл и посылает уведомление, по нажатию на которое будет открываться приложение и отображать имя файла.

Создадим проект:

Project name: P0991_ServiceNotification
Build Target: Android 2.3.3
Application name: ServiceNotification
Package name: ru.startandroid.develop.p0991servicenotification
Create Activity: MainActivity

Добавим в **strings.xml** строки:

```
    <string  name="start">Start</string>
    <string  name="stop">Stop</string>
```

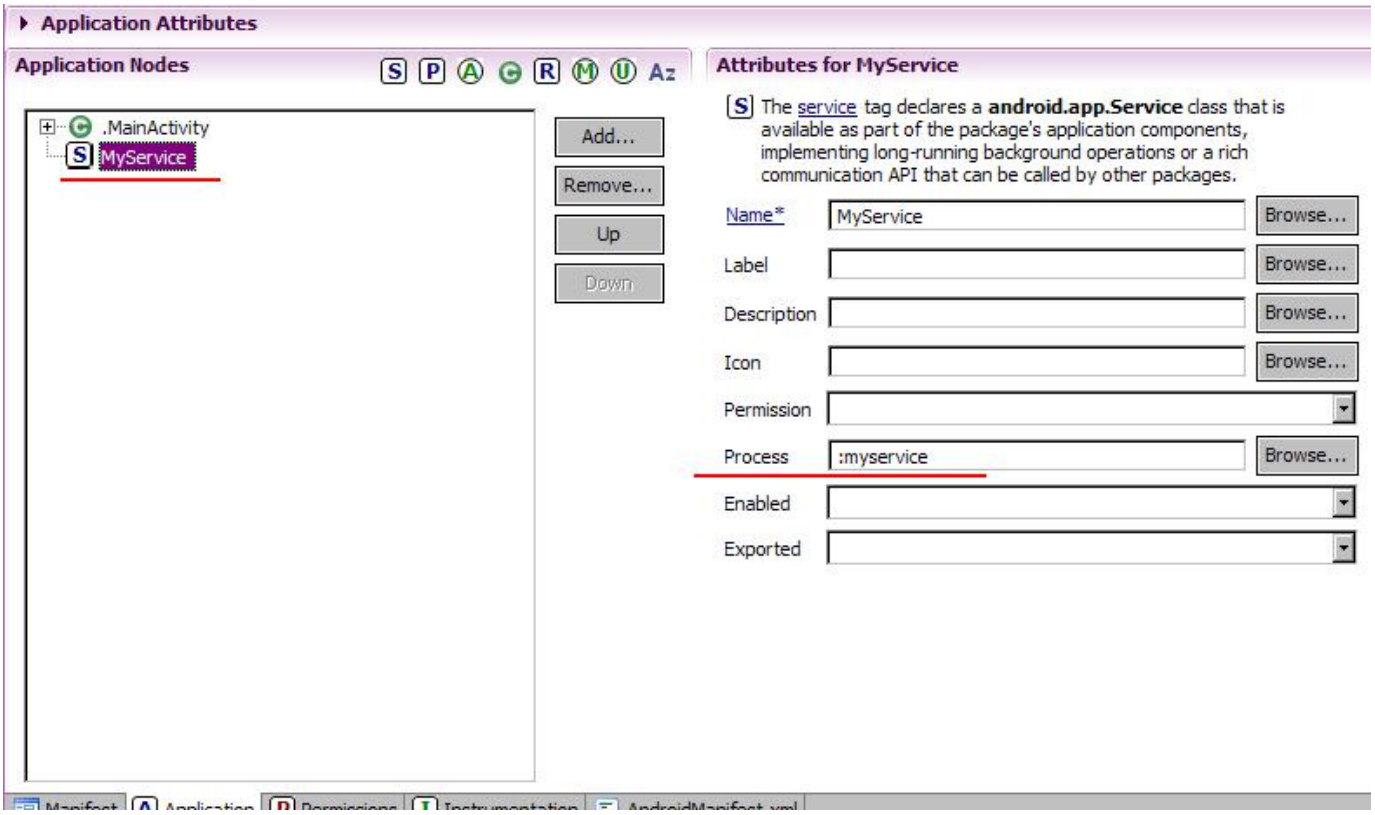
main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:orientation="vertical">
  <Button
        android:id="@+id/btnStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClickStart"
        android:text="@string/start">
  </Button>
  <Button
        android:id="@+id/btnStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClickStop"
        android:text="@string/stop">
  </Button>
  <TextView
        android:id="@+id/tv"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="">
  </TextView>
</LinearLayout>
```

Кнопки для старт/стопа сервиса и TextView для отображения результата

Создаем сервис **MyService**.java и прописываем его в манифесте. В манифесте же настроим сервис так, чтобы он работал в отдельном процессе. Для этого надо в его атрибуте process написать двоеточие и какое-нить слово.



Система эту строку добавит к package сервиса и, тем самым, получит название нового процесса, в котором и запустит сервис

com.android.systemui	120			8613
com.android.protips	210			8615
com.android.music	219			8616
com.android.quicksearchbox	227			8618
android.process.media	235			8620
com.android.mms	251			8623
com.android.email	270			8624
com.android.defcontainer	298			8628
com.svox.pico	308			8630
ru.startandroid.develop.p0991servicenotification	329			8626
ru.startandroid.develop.p0991servicenotification:myservice	340			8632

MainActivity.java:

```
package ru.startandroid.develop.p0991servicenotification;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

  public final static String FILE_NAME = "filename";

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    TextView tv = (TextView) findViewById(R.id.tv);

    Intent intent = getIntent();

    String fileName = intent.getStringExtra(FILE_NAME);
    if (!TextUtils.isEmpty(fileName))
      tv.setText(fileName);
  }

  public void onClickStart(View v) {
    startService(new Intent(this, MyService.class));
  }


  public void onClickStop(View v) {
    stopService(new Intent(this, MyService.class));
  }
}
```

В **onCreate** мы вытаскиваем из intent и кладем в TextView текст. Этот текст мы будем отправлять из сервиса через уведомление.

onClickStart и **onClickStop** – это обработчики кнопок. Стартуют и останавливают сервис.

MyService.java:

```
package ru.startandroid.develop.p0991servicenotification;

import java.util.concurrent.TimeUnit;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MyService extends Service {
  NotificationManager nm;

  @Override
  public void onCreate() {
    super.onCreate();
    nm = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
  }

  public int onStartCommand(Intent intent, int flags, int startId) {
    try {
      TimeUnit.SECONDS.sleep(5);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    sendNotif();
    return super.onStartCommand(intent, flags, startId);
  }

  void sendNotif() {
    // 1-я часть
    Notification notif = new Notification(R.drawable.ic_launcher, "Text in status bar",
      System.currentTimeMillis());

    // 3-я часть
    Intent intent = new Intent(this, MainActivity.class);
    intent.putExtra(MainActivity.FILE_NAME, "somefile");
    PendingIntent pIntent = PendingIntent.getActivity(this, 0, intent, 0);

    // 2-я часть
    notif.setLatestEventInfo(this, "Notification's title", "Notification's text", pIntent);

    // ставим флаг, чтобы уведомление пропало после нажатия
    notif.flags |= Notification.FLAG_AUTO_CANCEL;

    // отправляем
    nm.notify(1, notif);
  }

  public IBinder onBind(Intent arg0) {
```

```
        return null;
    }
}
```

В **onCreate** получаем менеджер уведомлений – NotificationManager. Он нам понадобится, чтобы отправить уведомление.

В **onStartCommand** запускаем паузу на 5 секунд (эмулируем закачку файла) и после этого отправляем уведомление. Именно из-за этой паузы мы и используем другой процесс, чтобы не тормозило основное приложение.

В **sendNotif** мы создаем и отправляем уведомление. Правда, немного в иной последовательности, что я описывал выше. Сначала первая часть, потом третья, потом вторая.

Первая часть – создаем Notification. В конструкторе указываем иконку и текст, которые будут видны в статус-баре. Также мы здесь указываем время. Обычно это текущее время. Но можно указать и прошлое и будущее. По этому времени уведомления будут отсортированы в статус-баре и в его раскрытой части.

Третья часть – создаем Intent, который мы бы использовали для вызова нашего Activity. Туда помещаем имя загруженного файла. Activity его достанет и поместит в TextView. Далее мы оборачиваем этот Intent в PendingIntent, с помощью метода getActivity. На вход ему передаем контекст и Intent. Второй параметр не используется (так написано в хелпе). А четвертый – это флаги, влияющие на поведение PendingIntent. Они не относятся к теме урока, мы их не используем.

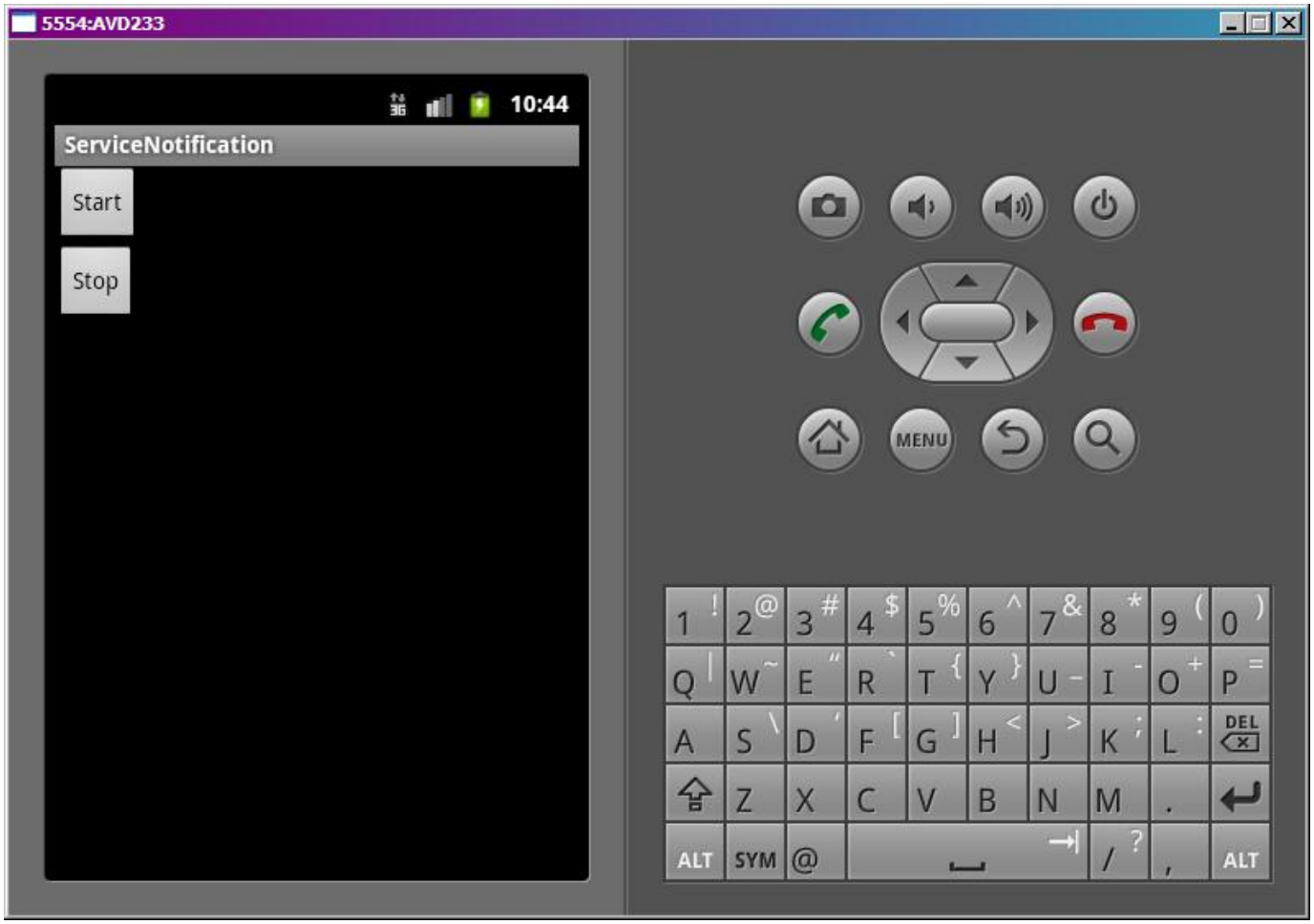
Теперь этот созданный PendingIntent содержит информацию о том, что надо вызывать Activity, а также объект Intent, который для этой цели надо использовать. Это будет использовано при нажатии на уведомлении.

Вторая часть – вызываем метод setLatestEventInfo. Передаем на вход контекст, текст-заголовок, подробный текст и PendingIntent. Теперь, когда мы откроем статус-бар, мы увидим два этих текста (заголовок и подробный). А, когда нажмем на уведомление, система использует PendingIntent для запуска Activity.

Далее мы для созданного уведомления ставим флаг FLAG_AUTO_CANCEL, чтобы оно исчезло из статус-бара после нажатия. По умолчанию оно не исчезает и продолжает висеть.

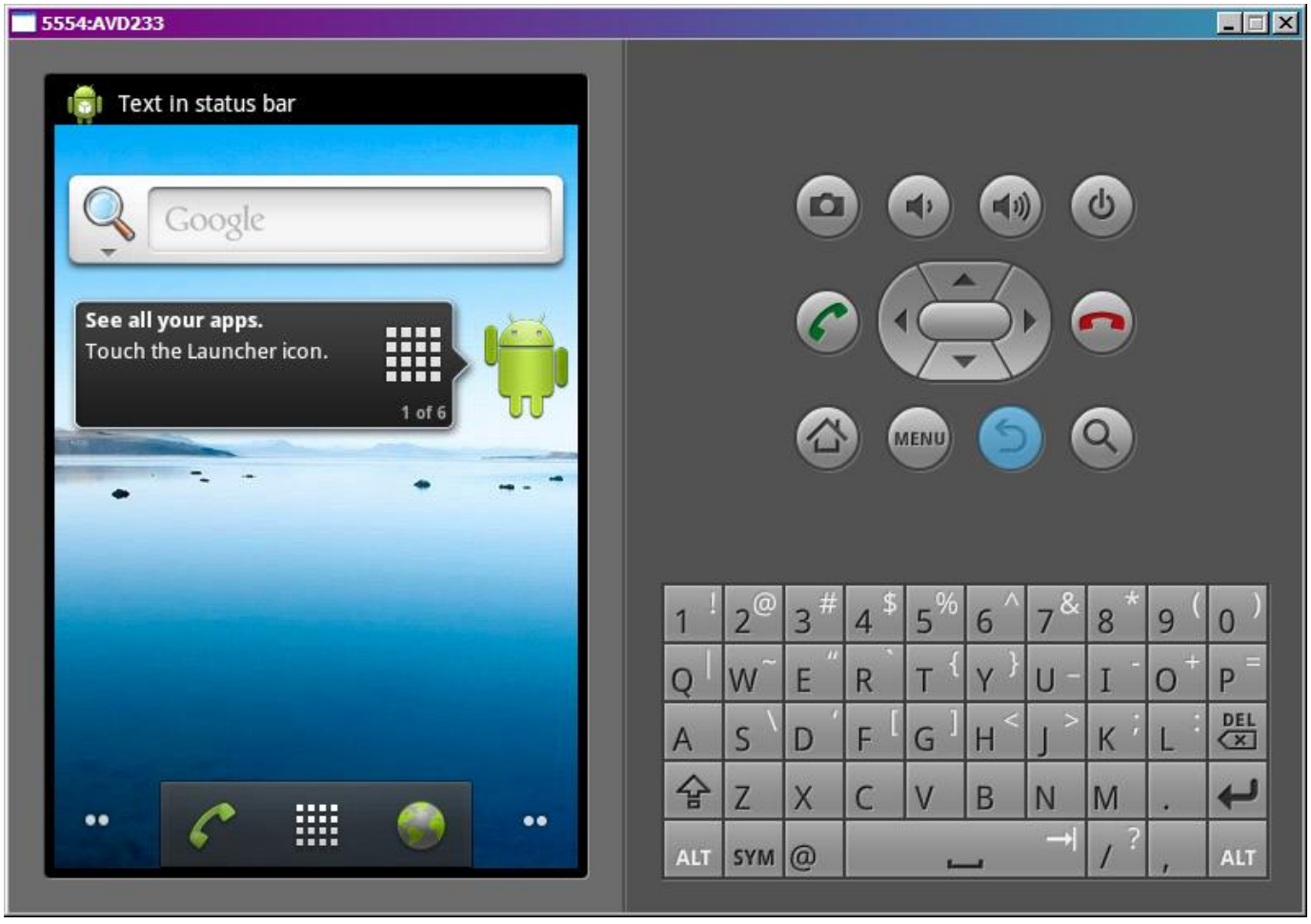
Далее вызываем метод notify для менеджера уведомлений и передаем туда ID и созданное уведомление. ID используется, если мы хотим изменить или удалить уведомление.

Все сохраним, запустим.

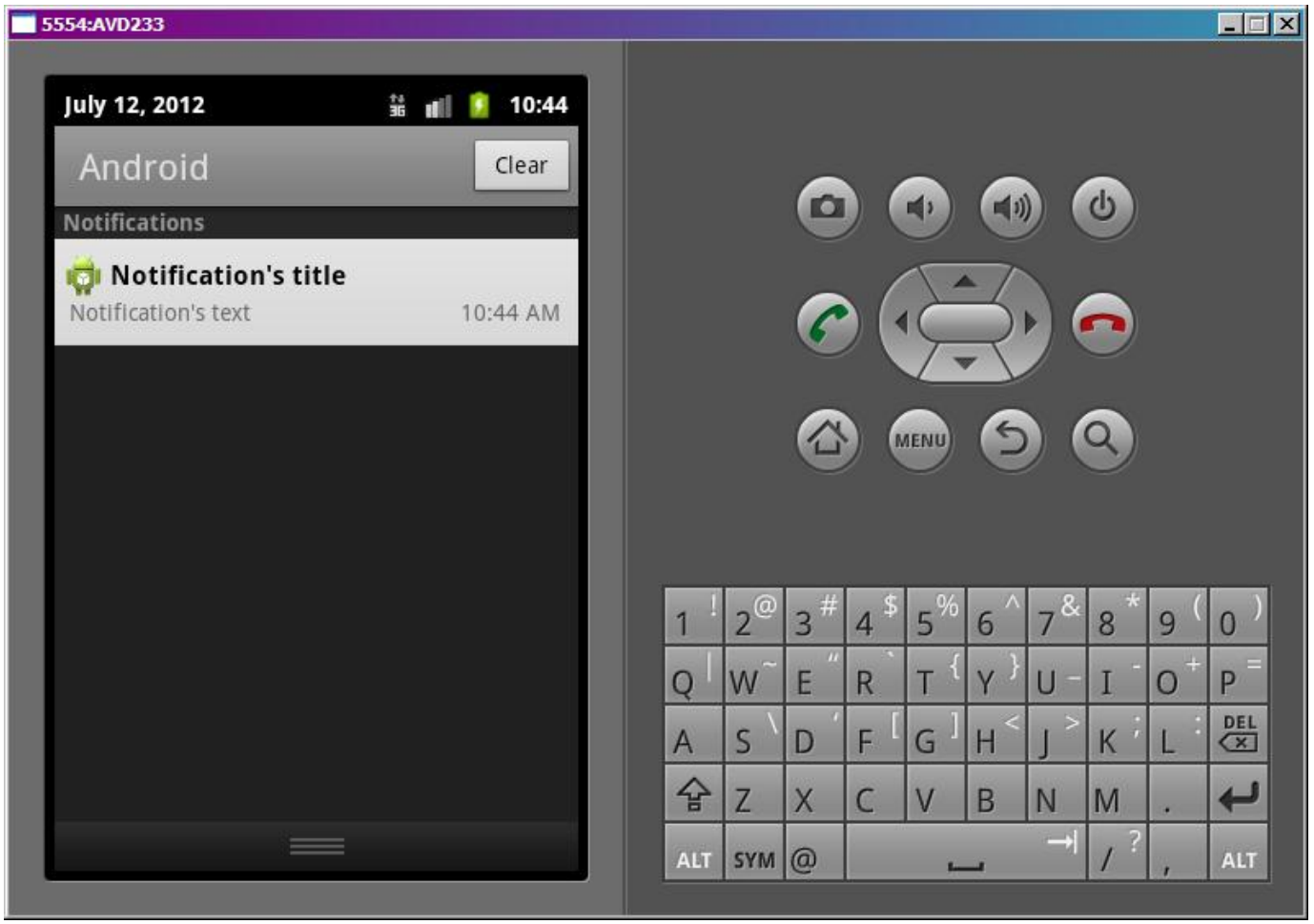


Жмем **Start** и сразу закрываем приложение кнопкой **Назад**.

Проходит 5 сек и появляется уведомление (первая часть)

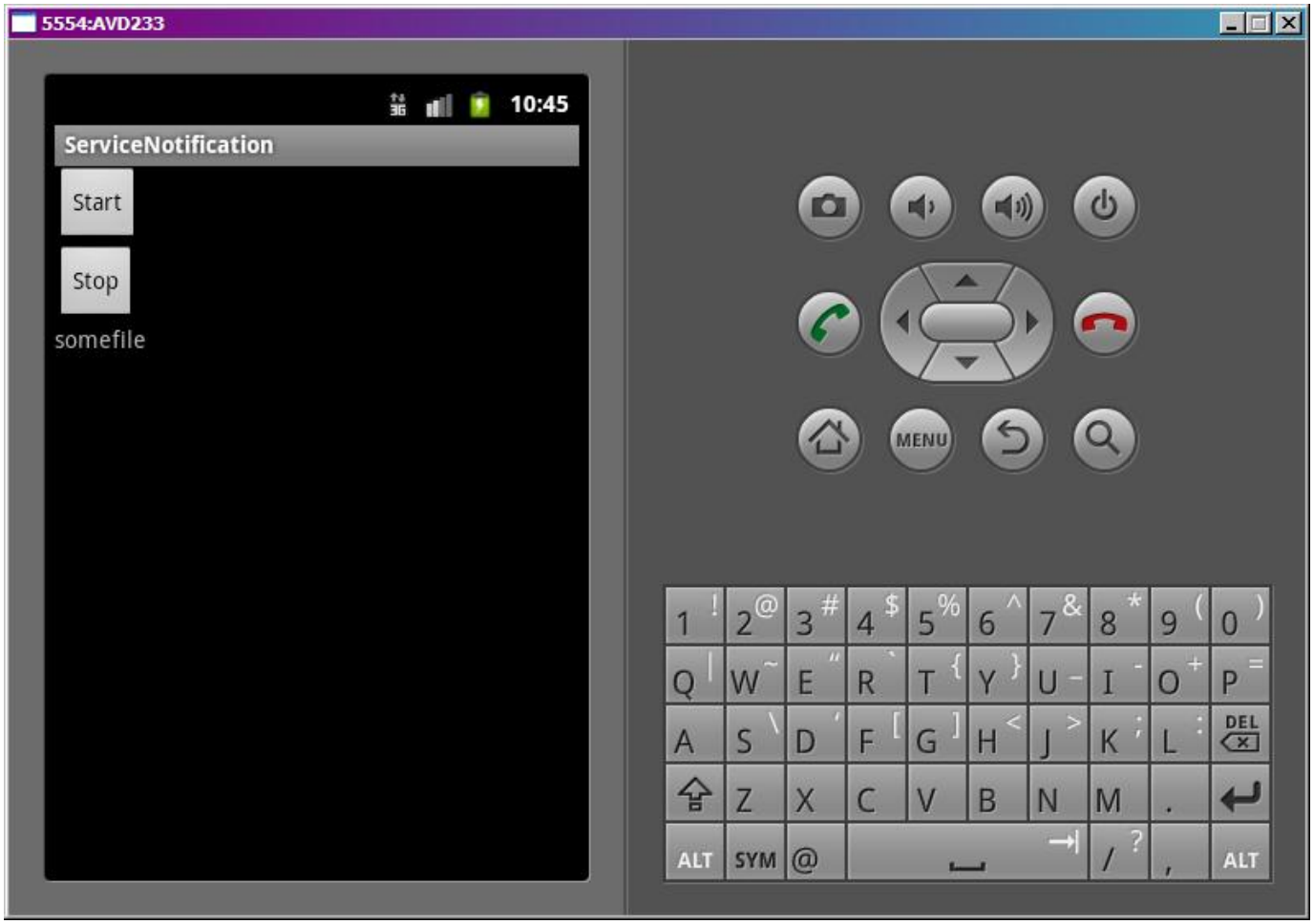


Открываем статус-бар и видим более подробную инфу (вторая часть)



Жмем на уведомление.

Открывается наше приложение (третья часть) и в TextView появляется текст, переданный из сервиса.



Теперь вкратце пробежимся по остальным интересным возможностям уведомлений.

Обновление старого или новое уведомление

Если вы создадите новое уведомление и отправите его (notify) с тем же ID, что и у уже существующего уведомления, то новое заменит старое. Таким образом, вы можете уведомления обновлять.

Если же надо показать новое уведомление, то используйте другой ID.

Удаление

Чтобы убрать уведомление из статус-бара, используется метод cancel у менеджера уведомлений. На вход подается ID. Либо используйте метод cancelAll, чтобы удалить все уведомления.

Звук

Если хотите, чтобы уведомление появилось со стандартным звуком, добавьте флаг Notification.DEFAULT_SOUND в поле уведомления defaults.

А для использования своих звуков используется поле sound.

Чтобы проиграть файл с **SD**:

notif.sound = Uri.parse("file:///sdcard/notification/ringer.mp3");

Чтобы использовать какую-либо из стандартных мелодий, используем **Content Provider**:

notif.sound = Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6");

Подробнее здесь.

Вибра

Если хотите, чтобы уведомление появилось со стандартной виброй, добавьте флаг Notification.DEFAULT_VIBRATE в поле уведомления defaults.

А для использования своей комбинации вибры используется поле vibrate. В это поле помещается массив long-чисел. Первое – длительность паузы (в миллисекундах) перед началом вибрирования, второе – длительность вибрирования, третье – длительность паузы, четвертое – длительность вибрирования … и т.д. Т.е. создаете свою комбинацию пауз и вибрирования. И мобила при получении уведомления вам ее провибрирует.

Подробнее здесь.

Для работы вибры необходимо прописать права VIBRATE в манифесте.

Индикатор

Если хотите, чтобы уведомление появилось с миганием индикатора, добавьте флаг Notification.DEFAULT_LIGHTS в поле уведомления defaults.

А для использования своей комбинации мигания индикатора используются поля

ledARGB – здесь задается цвет

ledOnMS – время «горения»

ledOffMS – время «не горения»

И в поле flags надо добавить флаг Notification.FLAG_SHOW_LIGHTS.

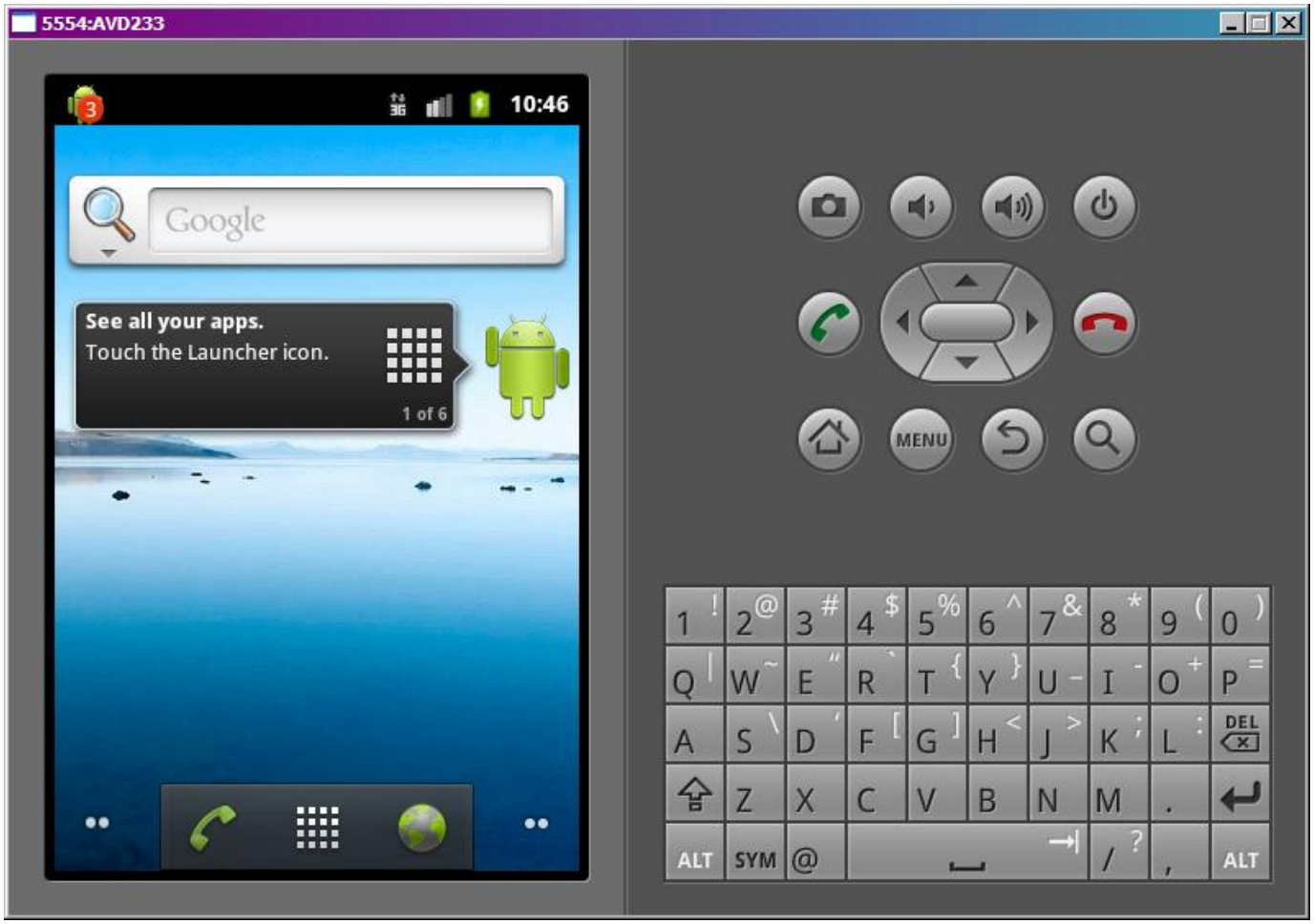
В итоге индикатор будет мигать с заданными значениями и с заданным цветом. В хелпе написано, что не все девайсы поддерживают разные цвета. Поэтому выбранный вами цвет не гарантируется.

Подробнее здесь.

Число

У Notification есть поле number. Вы можете поместить туда число больше нуля и оно отобразится на уведомлении.

Например, при notif.number = 3 уведомление будет выглядеть так:

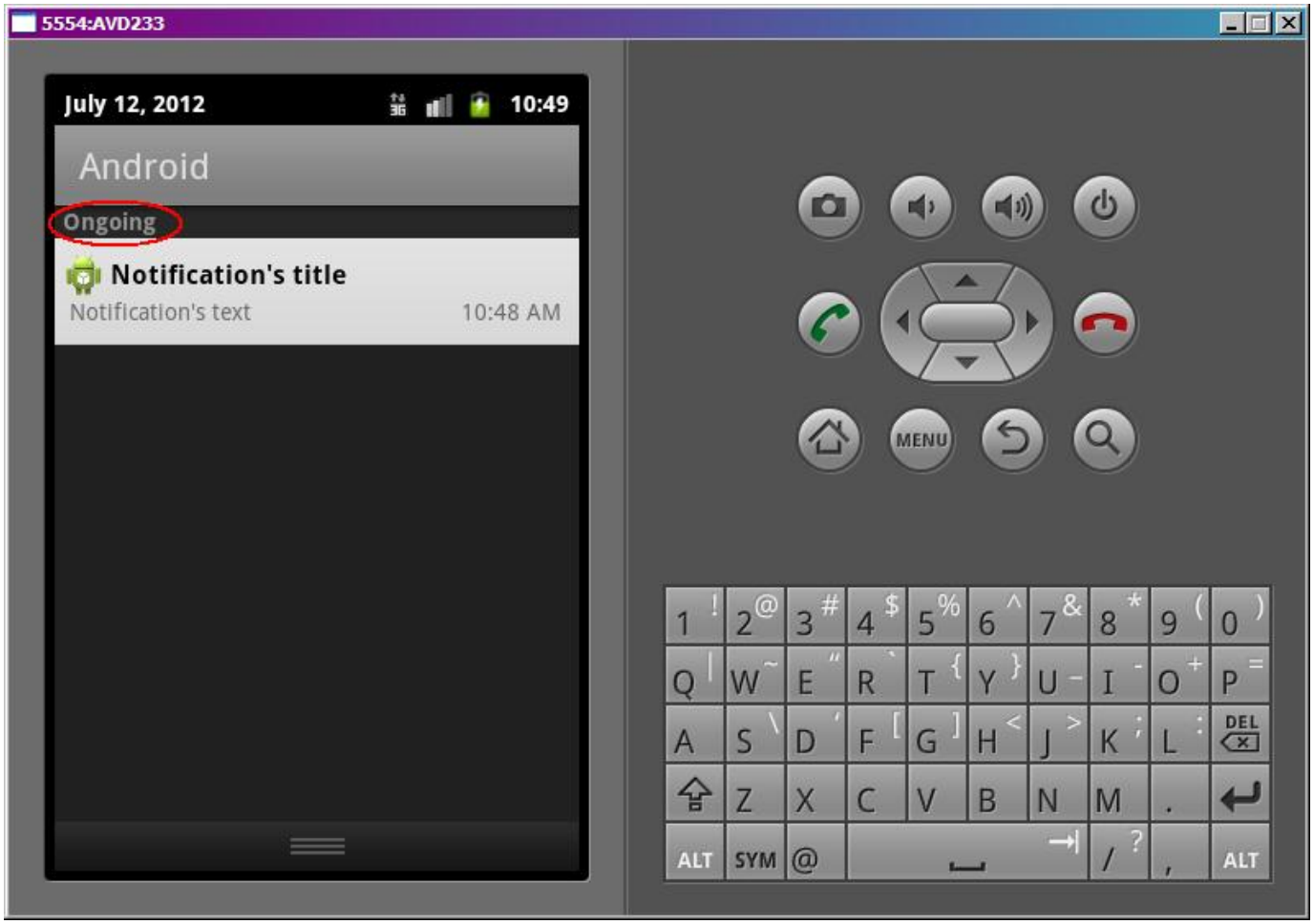


Флаги

Добавляются в поле flags

FLAG_INSISTENT – звук уведомления будет повторяться, пока не откроют статус-бар

FLAG_ONGOING_EVENT – уведомление появляется не в обычной секции, а в ongoing (постоянные). Уведомления из этой секции не удаляются при нажатии кнопки очистки уведомлений.



FLAG_NO_CLEAR – уведомление не удалится при очистке всех уведомлений

Не очень понимаю, в чем разница между ongoing и тем, что уведомление не удалится после нажатия на кнопку очистки всех уведомлений. Но флаги такие есть, и я о них упомянул.

На следующем уроке:

- изучаем IntentService
- включаем режим Foreground для сервиса
- помещаем сервис в автозагрузку

Урок 100. Service. IntentService. Foreground. Автозагрузка сервиса

В этом уроке:

- изучаем IntentService
- включаем режим Foreground для сервиса
- помещаем сервис в автозагрузку

Строили мы, строили, и, наконец, построили. Урок номер 100, с чем всех нас и поздравляю)

В этом уроке рассмотрим еще несколько полезных вещей про сервисы. Выносить каждую из них в отдельный урок я не стал, вполне можно в одном все рассмотреть. Проекты здесь тоже создавать не будем, чтобы урок не получился слишком громоздким. Я просто приведу некоторые куски кода и скрины для наглядности своих рассуждений. А если у вас будет желание, вы по этим наработкам сами можете создать проекты-примеры.

IntentService

Это подкласс обычного Service. Он используется, если вам в сервисе надо выполнять какие-то тяжелые задачи, и вы не хотите сами возиться с асинхронностью. Принцип работы этого вида сервиса прост. Он создает новый поток для своей работы. Затем берет все Intent пришедшие ему в onStartCommand и отправляет их на обработку в этот поток. Как именно обрабатываются Intent – зависит от нас, т.к. мы сами кодим это в методе onHandleIntent.

Т.е. приложение сыпет в сервис вызовами startService, в которых передает Intent-ы. IntentService принимает эти вызовы в onStartCommand, берет Intent-ы и отправляет их в очередь на обработку. И далее они поочередно обрабатываются в отдельном процессе методом onHandleIntent. Когда последний Intent из очереди обработан, процесс сам завершает свою работу.

Пример

В приложении делаем три вызова:

```
startService(intent.putExtra("time", 3).putExtra("label", "Call 1") );
startService(intent.putExtra("time", 1).putExtra("label", "Call 2") );
startService(intent.putExtra("time", 4).putExtra("label", "Call 3") );
```

Где **time** – это время паузы, которую будем делать в сервисе, а **label** – просто метка, чтобы отличать вызовы.

Код сервиса:

```
public class MyService extends IntentService {

  final String LOG_TAG = "myLogs";

  public MyService() {
    super("myname");
  }

  public void onCreate() {
    super.onCreate();
    Log.d(LOG_TAG, "onCreate");
  }
```

```
  @Override
  protected void onHandleIntent(Intent intent) {
    int tm = intent.getIntExtra("time", 0);
    String label = intent.getStringExtra("label");
    Log.d(LOG_TAG, "onHandleIntent start " + label);
    try {
      TimeUnit.SECONDS.sleep(tm);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    Log.d(LOG_TAG, "onHandleIntent end " + label);
  }

  public void onDestroy() {
    super.onDestroy();
    Log.d(LOG_TAG, "onDestroy");
  }
}
```

Здесь необходим конструктор, в котором вызываем конструктор супер-класса и указываем какое-нить имя. Оно будет использовано для наименования потока.

В методе onHandleIntent кодим обработку Intent-ов. Достаем из них time и label, запускаем паузу на time секунд и выводим в лог label в начале и в конце.

В итоге, при запуске в логах видим:

11:07:37.880: D/myLogs(4137): onCreate
11:07:37.880: D/myLogs(4137): onHandleIntent start Call 1
11:07:40.880: D/myLogs(4137): onHandleIntent end Call 1
11:07:40.880: D/myLogs(4137): onHandleIntent start Call 2
11:07:41.880: D/myLogs(4137): onHandleIntent end Call 2
11:07:41.880: D/myLogs(4137): onHandleIntent start Call 3
11:07:45.890: D/myLogs(4137): onHandleIntent end Call 3
11:07:45.890: D/myLogs(4137): onDestroy

Сервис создался, вызовы выполнились по очереди и сервис завершил работу. От нас понадобилось только накодить обработку.

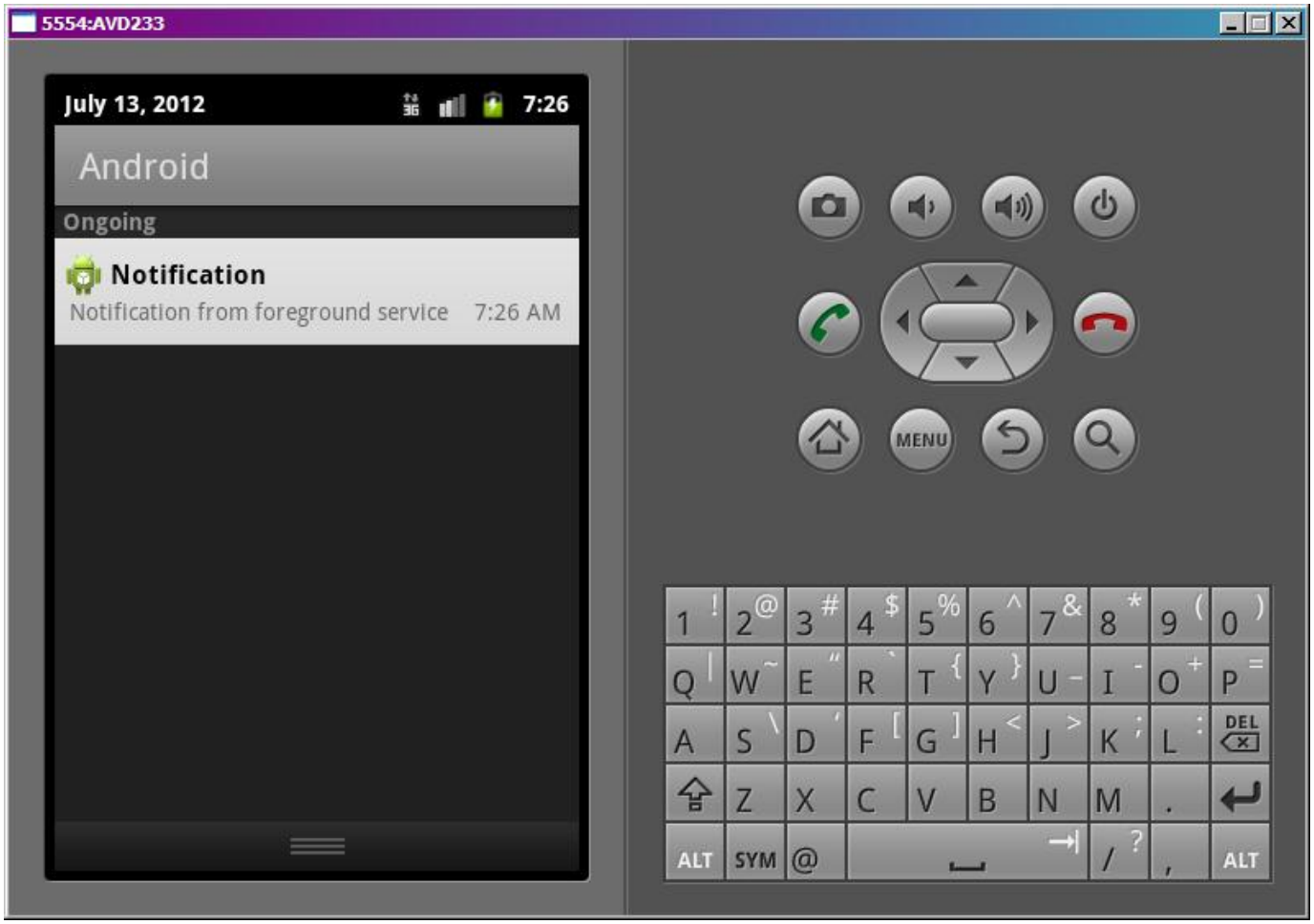
Foreground

Вы можете сказать системе, что ваш сервис очень важен для пользователя и его нельзя грохать при нехватке памяти. Это актуально, например, для музыкального плеера. В статус-бар при этом будет помещено уведомление.

Делается это методом startForeground (int id, Notification notification).

На вход он принимает те же параметры, что и NotificationManager.notify – ID и Notification.

Т.е. вы создаете уведомление, назначаете ему ID и передаете это в startForeground. Сервис переходит в режим IDDQD :), а в статус-баре появилось уведомление.



Оно появилось в разделе для постоянных уведомлений (Ongoing).

Метод stopForeground (boolean removeNotification) - возвращает сервису способность быть убитым системой в случае острой нехватки памяти. А на вход он принимает boolean-значение – удалять уведомление из статус-бара или нет.

Уведомление также пропадет, когда сервис будет остановлен.

Эти методы работают, начиная с Android 2.0. Пример реализации для более ранних версий есть в хелпе.

Напомню, что уведомления мы научились создавать на прошлом уроке.

Автозагрузка

Сервисы для получения погоды или почты имеет смысл помещать в автозагрузку. Для этого нам надо создать BroadcastReceiver, настроить его IntentFilter на Action = **android.intent.action.BOOT_COMPLETED**, и добавить права **android.permission.RECEIVE_BOOT_COMPLETED**. Этот BroadcastReceiver будет вызван системой при старте системы и в нем мы кодим запуск сервиса.

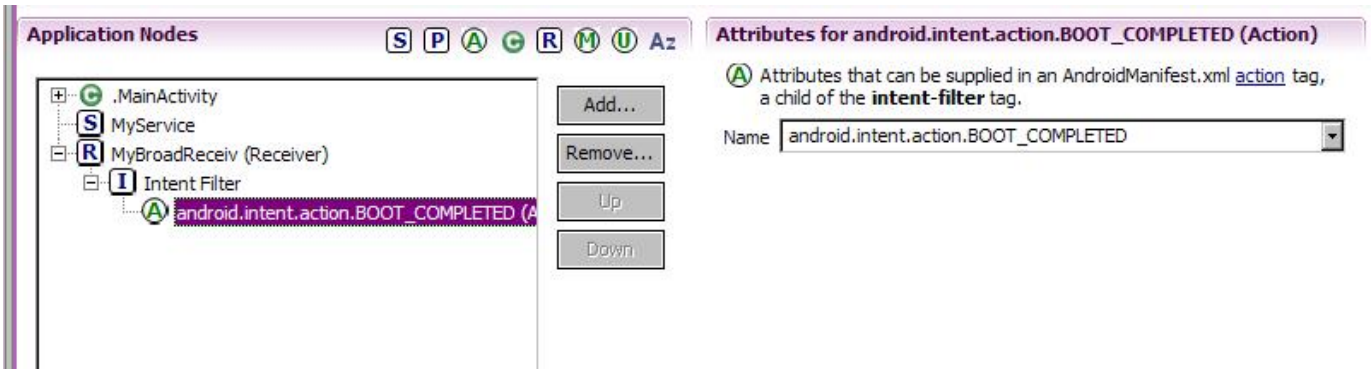
Допустим, есть проект с сервисом MyService.

Создаем в проекте класс **BootBroadReceiv**

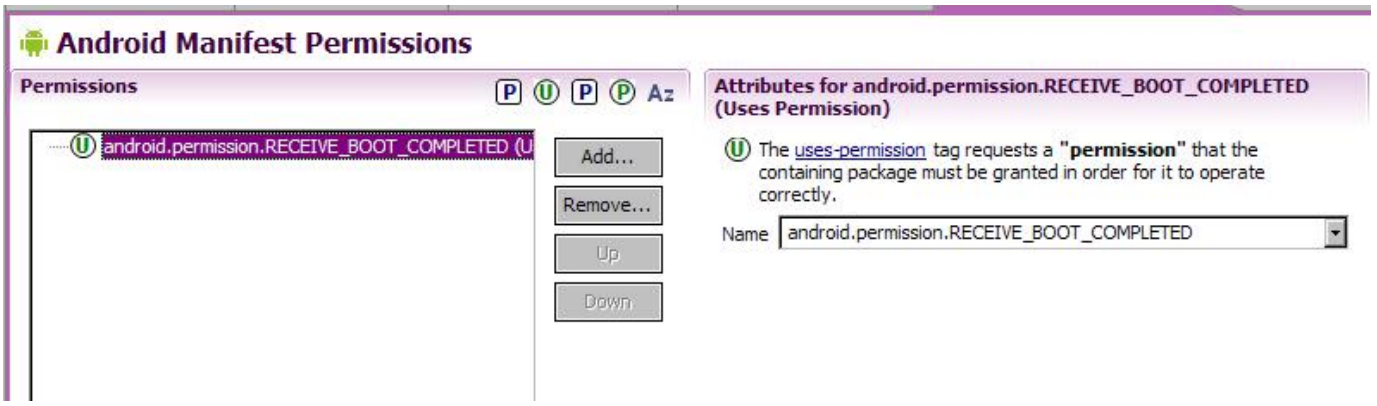
```
public class MyBroadReceiv extends BroadcastReceiver {

  final String LOG_TAG = "myLogs";
```

```
    public void onReceive(Context context, Intent intent) {
      Log.d(LOG_TAG, "onReceive " + intent.getAction());
      context.startService(new Intent(context, MyService.class));
    }
}
```

В манифесте добавляем его как **Receiver** и настраиваем фильтр



Добавляем права на получение сообщения о загрузке



Инсталлим проект на AVD. Закрываем AVD. Запускаем через меню в Eclipse: Window > AVD Manager. Находим там наш эмулятор и запускаем вручную.

Когда он запустился, смотрим логи

onReceive android.intent.action.BOOT_COMPLETED
MyService onCreate
MyService onStartCommand

Сработал BroadcastReceiver и запустил сервис.

Если после запуска AVD логи не отображаются, то откройте DDMS и во вкладке Devices явно выберите ваш AVD.

P.S. Я уже писал об этом, но напишу еще раз. Последующие уроки будут выходить по более свободному графику. Следите за обновлениями.

На следующем уроке:

- создаем свой ContentProvider