CS 4985 – Fall 2013
Lab 2 – Part 2, 10 pts.
Due: Tuesday, Sept. 24th at 9a

Objectives

- Be able to create a context menu.
- Be able to create a submenu.

Specifications

This lab is the second part of Lab 2. You will need to complete Lab 2 – Part 1 before starting on the second part of the lab exercise.

A. Getting Started

1. Open your completed Lab2 – Part 1 exercise, the name of this project should be *FirstnameLastname*Lab2, e.g. MarySmithLab3.

B. Add a context menu for when an individual student is invoked with a long-press

1. If an activity uses a ListView and you want all list items to provide a context menu, register all items for a context menu by passing the ListView to registerForContextMenu(). Do this by adding the following code to the StudentListActivity::onCreate method:

```
ListView lv = getListView();
registerForContextMenu(lv);
```

2. The above code will allow the user to invoke an action, typically a context menu, on each individual element. For now we will just display a toast. In this instance, the toast will display the name of the item that was invoked with a long press. To do this add the following code to: StudentListActivity::onCreate method.

```
lv.setOnItemClickListener(new OnItemClickListener() {
   @Override
   public void onItemClick(AdapterView<?> parent, View view, int position,
   long id)
   {
         Toast.makeText(getApplicationContext(),
                ((TextView)view).getText(), Toast.LENGTH_SHORT).show();

   }
 });
```

The AdapterView.onItemClickListener allows a callback when an item in the AdapterView has been clicked.

3. Extract all the above code in to a private helper method.

4. Build and run the program and verify a toast is display when upon context menu activation of an individual list item.

    a. To verify it works, click on any name in the list and that name should be displayed as a toast.

C. Add a context menu

In Step B. above, when a context menu activation occurs for a ListView item, we just responded to the onItemClick by posting a Toast. In this section, functionality to display an actual context menu will be implemented.

1. Comment out all the lv.setOnItemClickListener code in the private helper method you created. This is done as we will be adding functionality to display an actual context menu.

2. Add a new menu resource that will have three options: 1) Edit 2) Delete 3) Send email.

    a. The name of the menu resource should be: studentscontext.xml.

    b. Add the following elements to the menu element:

```
<item android:id="@+id/editStudent"
       android:title="@string/edit_student" />

<item android:id="@+id/deleteStudent"
       android:title="@string/delete_student" />

<item android:id="@+id/emailStudent"
       android:title="@string/email_student" />
```

        i. Create the following string resources for edit_student, delete_student, and email_student: “Edit”, “Delete”, and “Send email” respectively.

3. Note: we have already registered all the items in the ListView to be context menu enabled by adding the following two lines of code.

```
ListView lv = getListView();
registerForContextMenu(lv);
```

4. Create and display a context menu by overriding the onCreateContextMenu method as follows:

    a. Add the following method stub: (Eclipse can add this stub for you by using Source→Override/Implement methods.)

    ```
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
                ContextMenuInfo menuInfo) {

          super.onCreateContextMenu(menu, v, menuInfo);

    }
    ```

    b. Add code within this method to determine the list item that is being invoked. Once we determine the element we can set the context menu title with the setHeaderTitle method. Add the following code to the onCreateContextMenu method:

    ```
    AdapterView.AdapterContextMenuInfo info =
          (AdapterView.AdapterContextMenuInfo)menuInfo;
    menu.setHeaderTitle(students[info.position]);
    ```

    You will need to replace the students array with the name of your array.

5. Add code to the onCreateContextMenu to inflate the studentcontext menu. This will be done like using a MenuInflater like was done for the options menu.

6. Run the code and verify the context menu with appropriate header title is displayed.

7. Add the event handler code for when a context menu item is selected. This will be done by overriding the onContextItemSelected method as follows:

    a. Add the following line of code to the onContextItemSelected method, which will get the extra menu information provided to the onCreateContextMenu callback when a context menu is brought up for this AdapterView.

    ```
    AdapterContextMenuInfo info =
          (AdapterContextMenuInfo)item.getMenuInfo();
    ```

    b. Determine which context menu item was selected by adding a switch statement that gets the MenuItem's id. Once the menu item's id was selected we will just do a simple toast to show that we have responded to the correct item. Of course, normally, we would not display a toast, but would add the necessary event handling code for each item. Add the event handling code by adding the following switch statement under the line of code from step b.

    ```
    switch (item.getItemId()) {
    case R.id.editStudent:
          Toast.makeText(getApplicationContext(), "Edit: " +
    ```

```
                students[info.position], Toast.LENGTH_SHORT).show();
                return true;
            case R.id.deleteStudent:
                Toast.makeText(getApplicationContext(), "Delete: " +
                students[info.position], Toast.LENGTH_SHORT).show();
                return true;
            case R.id.sendStudent:
                Toast.makeText(getApplicationContext(), "Send: " +
                students[info.position], Toast.LENGTH_SHORT).show();
                return true;
            default:
                return super.onContextItemSelected(item);
            }
```

8. Test your code and verify the context menu that you created is displayed and the appropriate toast is displayed for each menu item invocation.

D. Add a submenu

A *submenu* is a good way for the user to quickly specify some setting option, e.g., a difficulty level for a game. In this step, we will add a submenu that will be displayed off an options menu item invocation. Note this can be done **programmatically**, by building the entire options menu in code as follows instead of declaratively as follows in the onCreateOptionsMenu:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

  super.onCreateOptionsMenu(menu);

  // Set the intent for the Add student options menu item
  menu.add(R.string.addstudent).setIntent(new Intent(this,
        GetStudentNameActivity.class));

  // Create a submenu, the addSubMenu will be the name that is displayed
  // both in the options menu and as the header in the submenu
  // The submenu will be displayed as a group of checkable boxes
  SubMenu nameDisplay = menu.addSubMenu(R.string.namedisplay);
  nameDisplay.add(NAME_DISPLAY, FIRSTONLY, 1,
        R.string.firstonly).setChecked(this.firstOnly);
  nameDisplay.add(NAME_DISPLAY, FIRSTANDLAST, 2,
        R.string.firstandlast).setChecked(!this.firstOnly);
  nameDisplay.setGroupCheckable(NAME_DISPLAY, true, true);

  return true;
}
```

For this code `NAME_DISPLAY, FIRSTONLY,` and `FIRSTANDLAST` are user-defined constant IDs and `this.firstOnly` is a boolean flag to hold this particular status.

The code to act upon the invocation of these submenu items would be added to the onOptionsItemSelected as follows:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    switch (item.getItemId()) {
    case FIRSTONLY:
        if (!this.firstOnly) {
              this.firstOnly = true;
        }
        Toast.makeText(getApplicationContext(), "First",
              Toast.LENGTH_SHORT).show();
        return true;
    case FIRSTANDLAST:
        if (this.firstOnly) {
              this.firstOnly = false;
        }
        Toast.makeText(getApplicationContext(), "First and Last",
              Toast.LENGTH_SHORT).show();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

The above approach is all done programmatically. However, it is best to try to do all UI creation **declaratively**, if at all possibly. This lab exercise will create and display a submenu declaratively. This will be done as follows:

1. Currently, the options menu has the following items in it: Add and Delete All. Add a third option to this menu in the studentsoption.xml file by adding the following element to the menu:

```
<item android:id="@+id/nameDisplay"
      android:title="@string/name_display" />
```

   a. You will need to add a string resource name_display that has the following text: "Name display"

2. The Name display options menu item can be turned into a submenu by adding a menu as a child element of the Name Display item element . Do this by modifying the Name Display item as follows:

```
<item android:id="@+id/nameDisplay"
      android:title="@string/name_display">
      <menu>
          <item
              android:id="@+id/firstOnly"
              android:title="@string/first_only"></item>
          <item
              android:id="@+id/firstAndLast"
              android:title="@string/first_and_last"></item>
      </menu>
</item>
```

3. Add event handler code to the onOptionsItemSelected method to respond to the selection of the First only and First and Last menu items. To verify the event handler has been invoked just display a toast message for each.

4. Run the code and verify the appropriate toast message is displayed for each menu item invocation.

E. Turn the submenu into a checkable group

At the start of *Section D*, example code was given that created a submenu programmatically as a checkable group. However, when we implemented the submenu declaratively, the menu items were not treated as a checkable group. We can implement the checkable group behavior for the submenu declaratively as follows:

1. In the studentoptions.xml file modify the submenu element to be contained within a checkable group and then set the First only item checked state to true. You can do this by modifying the code as follows: (The checked attribute will set the First only item to be checked initially.)

```
<menu>
    <group android:checkableBehavior="single">
          <item
              android:id="@+id/firstOnly"
              android:title="@string/first_only">
              android:checked="true"></item>
          <item
              android:id="@+id/firstAndLast"
              android:title="@string/first_and_last">
              android:title="First and last"></item>
   </group>
</menu>
```

2. In the onOptionsItemSelected method modify the event handler code for the First only and First and Last item selections by adding a call to:

   ```
   item.setChecked(true);
   ```

   before the toast message is displayed. This will ensure that the appropriate item is checked on each invocation of the submenu.

3. Modify the code so that the first only selection is checked by default on the first invocation of the app.

4. Run the code and verify the appropriate option is checked on each invocation of the submenu.

Submission

Name your zip file *FirstnameLastname*Lab2.zip, e.g, JohnDoeLab2.zip and submit the file in Moodle by the due date.

Grading

- 10 pts. – Application works correctly and the filename for submission is correct.
- 7 pts. – Application works nearly correctly, but some of the functionality is missing or was implemented incorrectly.
- 4 pts. – Application is missing most of the major functionality or crashes when invoking some of it.
- 0 pts. – Application crashes when started, the lab was not submitted, nearly all the functionality was not implemented.