# Modulo 4

L'ambiente di sviluppo

# 4.1

# Introduzione a Java e XML

Lo sviluppo di applicazioni Java, di solito, è *molto* diverso...

# 4.1.1 Cos'è Java

Linguaggio orientato agli oggetti, general purpose.

*"Write once, run everywhere."*

1995 → 2011 (Java SE 7)

Implementazione
di riferimento Sun.

10 milioni di utenti.

# 4.1.1 Cos'è Java

Linguaggio orientato agli oggetti, general purpose.

*"Write once, run everywhere."*

Menzogna

1995 → 2011 (Java SE 7)

Implementazione
di riferimento Sun.

10 milioni di utenti.

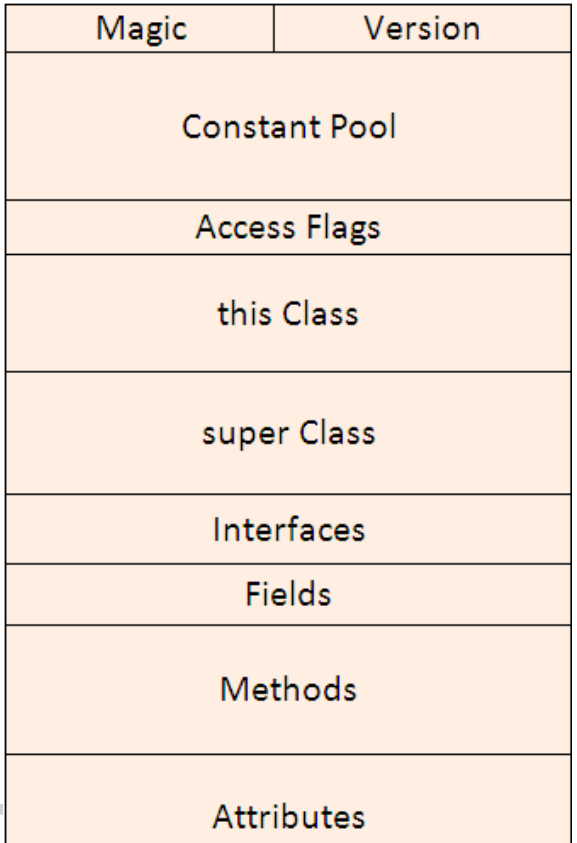
# 4.1.1 Cos'è Java (2)

Basato su **Java bytecode**:

Java source (`.java`) → Java class (`.class`)

**Componenti di un file class:**

- Costanti
- Classe e gerarchia di derivazione
- Interfacce
- Campi
- Metodi (bytecode)
- Attributi



# 4.1.1 Cos'è Java (3)

Basato su **Java bytecode**:

Java source (`.java`) → Java class (`.class`)

Per provare:

```
javac test.java
```

produce un file `.class` funzionante ed eseguibile.

```
java -cp . TestClass
```

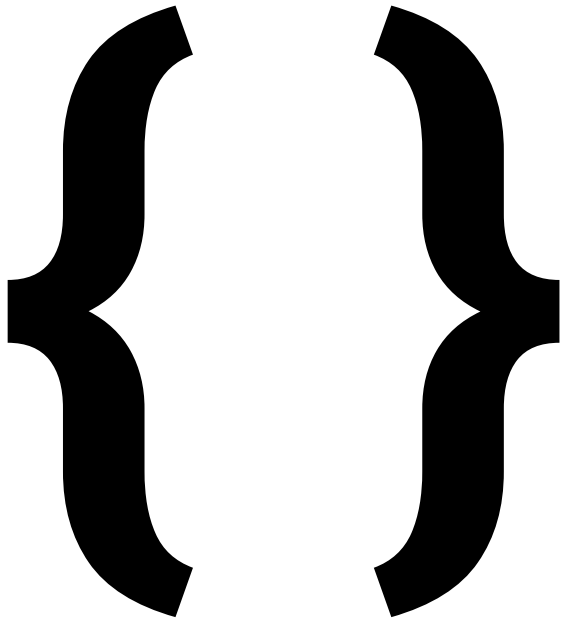
# 4.1.1 Cos'è Java (4)

Il solito codice d'esempio:

```
public class TestClass {
   public static void main(
      String[] args){

      System.out.println("Hello!");
   }
}
```

# 4.1.2 Sintassi Java

Sintassi ispirata al C/C++.



*Strong typing*, classi

Allocazione memoria

Modificatori d'accesso

# 4.1.2 Sintassi Java

Poche primitive per la gestione della memoria.

`new`

~~`delete`~~

`sun.misc.Unsafe` (per i temerari)

# 4.1.3 Classi e namespace

```
public Gatto extends Quadrupede {
   public Gatto(String colore){
      this.colore = colore;
   }
   private String colore;
   public String getColore(){
      return this.colore;
   }
}
```

# 4.1.3 Classi e namespace

**Package**: gruppo di classi Java, definite nello stesso namespace.

```
package it.neunet.csa;
```

Quando si utilizzano:

```
import it.neunet.csa.ClasseTest;
import it.neunet.csa.*;
```

# 4.1.4 Type system

## Static.

Almeno fino a Java 8.

## Strong.

Liskov: "un tipo passato come parametro deve essere compatibile".

## Safe.

Parzialmente. Non *safe* su array, generics ed altro.

# 4.1.4 Type system (Tipi primitivi)

Tipi primitivi:

- int
- float
- double
- byte
- char
- boolean
- ~~String~~...?

Rappresentazione efficiente in memoria.

Copia *by value*.

# 4.1.4 Type system (Oggetti)

Tutte le classi sono oggetti.
`system.lang.Object`

Copia *by reference*.

**Attenzione**: Object è il tipo radice per tutte le classi, ma **non tutto** è una classe in Java.

`int` ≠ `Integer`

# 4.1.4 Type system (Oggetti)

```
public class Object {
    public int hashCode();
    public boolean equals(Object obj);
    public String toString();
}
```

Classe radice.

Attenzione al metodo `equals`.

# 4.1.4 Type system (Boxing)

Convertire un oggetto primitivo in un oggetto.

**Non** è automatico:

```
Integer boxInt = new Integer(42);
```

Unboxing è il procedimento inverso:

```
int unboxInt = boxInt.intValue();
```

# 4.1.4 Type system (Boxing)

Convertire un oggetto primitivo in un oggetto.

**Non** è automatico:

```
Integer boxInt = new Integer(42);
```

U

```
in
```

Molte classi Java richiedono istanze di oggetti:

```
HashMap<Integer, String> mappaAssociativa;
```

# 4.1.4 Attenzione al type system

```
int a = 10, int b = 10;
a == b; //true

Integer a = 1000; Integer b = 1000;
a == b; //false

Integer a = 10; Integer b = 10;
a == b;
```

# 4.1.4 Attenzione al type system

```
int a = 10, int b = 10;
a == b; //true

Integer a = 1000; Integer b = 1000;
a == b; //false

Integer a = 10; Integer b = 10;
a == b; //true!
```

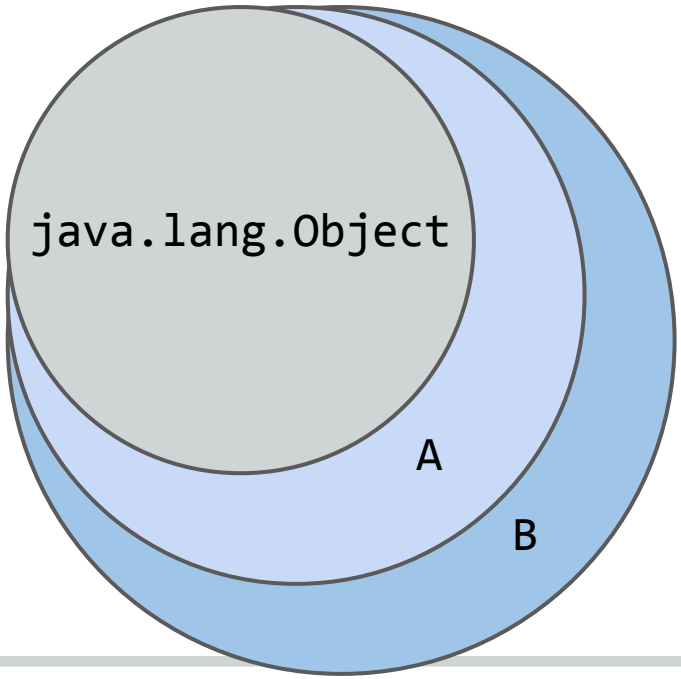
# 4.1.4 Attenzione al type system

```
int a = 10, int b = 10;
a == b; //true

Integer a = 1000; Integer b = 1000;
a == b; //false

Integer a = 10; Integer b = 10;
a == b; //true!
a.equals(b); //true
```

# 4.1.5 Derivazione

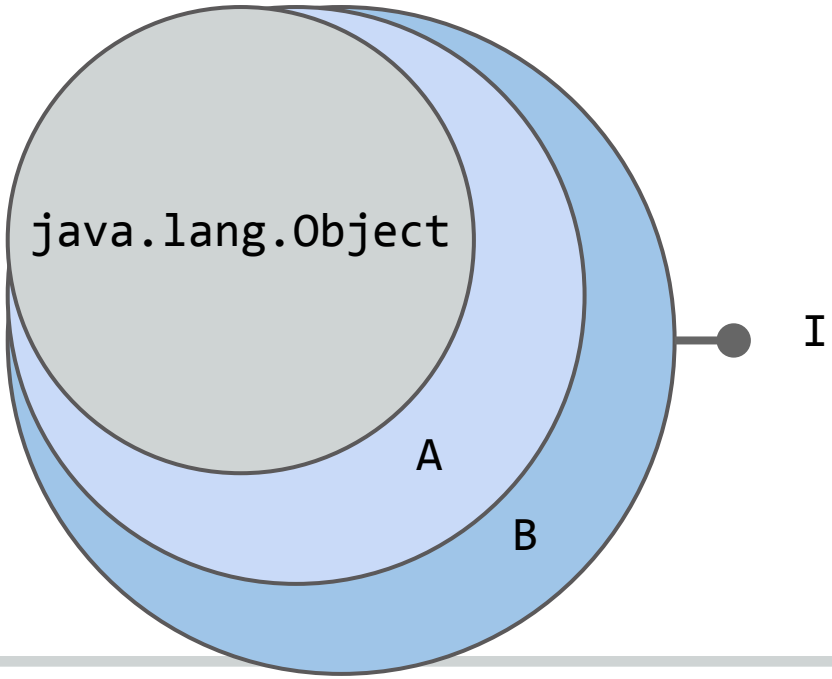
```
public B extends A { }
```

Java ha un type system a derivazione singola.



# 4.1.5 Interfacce

```
public B extends A
         implements I { }
```



# 4.1.5 Interfacce

```
public interface I {
   public void Metodo();
}
```

Non contengono codice.

Un insieme di metodi pubblici che vengono implementati dalla classe (`implements`).

# 4.1.5 Implementazione

Una classe può:

- **Derivare** da **1** classe.
- **Implementare** da un numero **qualsiasi** di interfacce.

**Tutti** i metodi delle interfacce ed i *metodi astratti* vanno implementati.

(Altrimenti si ottiene una *classe astratta*.)

I metodi della classe base **possono** essere sovrascritti (*overriding*).

# 4.1.5 Implementazione

```
public class TestClass
   implements Comparable<TestClass> {

   @Override
   public String toString() { }

   public int compareTo(TestClass o) { }

   public abstract int myAbstract();
```

# 4.1.6 Generics

Controlli a tempo di compilazione: un tipo che fa riferimento esplicito ad un altro.

```
ArrayList<Block> arr = new ArrayList<>();
arr.add(new Block());
Block b = arr[0];
```

Value type inference.

Runtime ignora i generics: *type erasure*.

# 4.1.6 Generics

Controlli a tempo di compilazione: un tipo che fa riferimento esplicito ad un altro.

```
var arr = new ArrayList<Block>();
arr.add(new Block());
Block b = arr[0];
```

Expression type inference.

Value type inference.

Runtime ignora i generics: *type erasure*.

# 4.1.7 Classi locali / anonime

Classi senza nome, generate dal compilatore, che implementano o estendono un tipo.

```
Runnable r = new Runnable() {
   public void run() {
      // ...
   }
}
```

Sostituiscono *closure* funzionali (Java 8).

# 4.1.7 Classi locali / anonime

Classi senza nome, generate dal compilatore, che implementano o estendono un tipo.

```
Runnable r = new Runnable() {
   public void run() {
      //
```

```
package java.lang;
public interface Runnable {
  public void run();
}
```

ava 8).

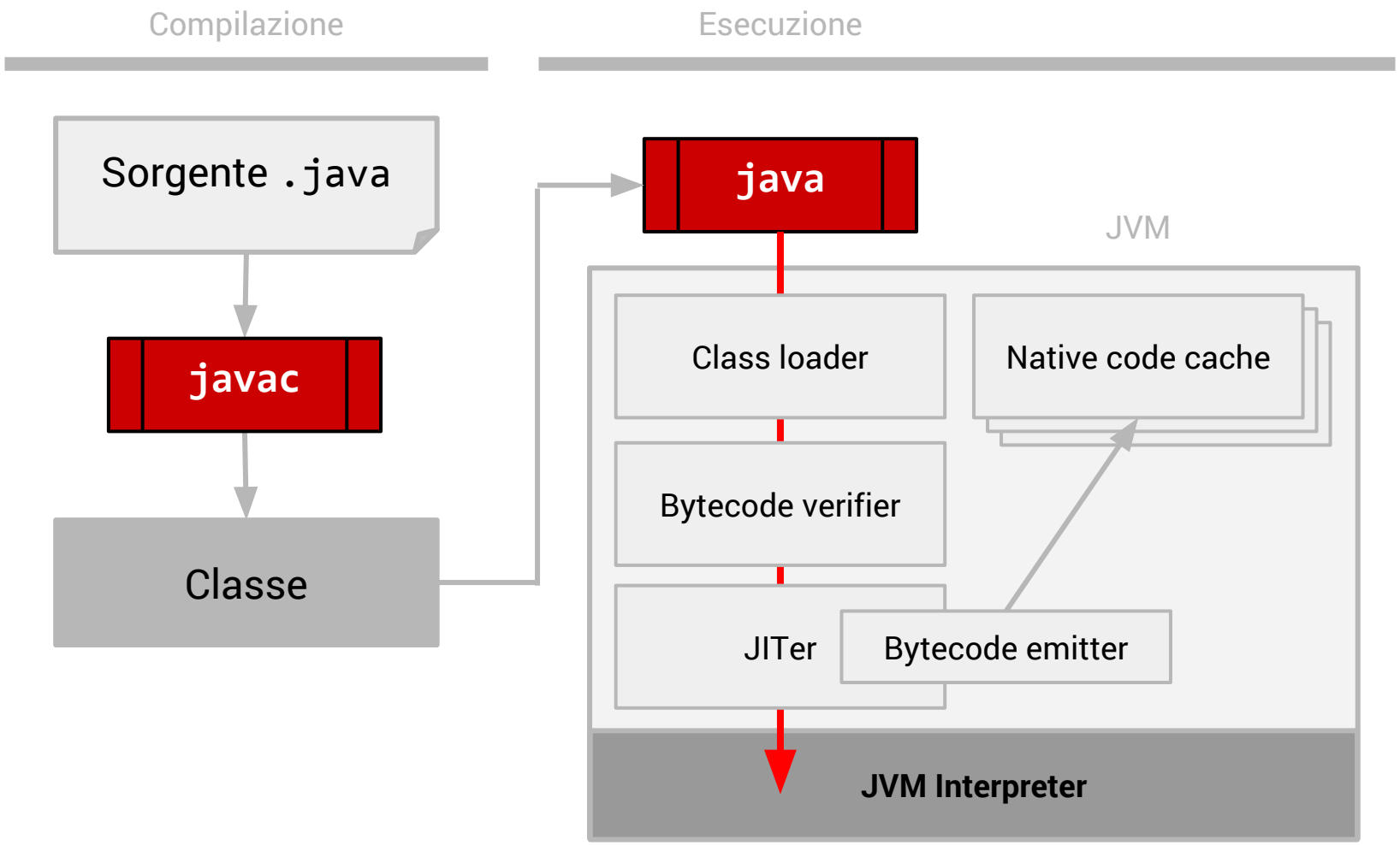
# 4.1.8 Java Virtual Machine

Macchina virtuale che esegue Java bytecode.

Implementata su HW reale o su sistema operativo ospite: stesse funzionalità ovunque.

- Java runtime environment (JRE)
  - Java application launcher
  - Java bytecode verifier
  - Managed heap + garbage collector
  - Automated exception handling
- Java class library

# 4.1.8 Java Virtual Machine



# 4.1.8 Modello di memoria

JVM con unico *"managed heap"*:

- Unico spazio di allocazione visibile al programmatore.
- Indipendente da spazio utilizzato da JVM per classi, JIT code, metadati, etc.
- Allocazione esplicita. Deallocazione gestita.
- Paradigma di "***memoria infinita***" parziale: grandezza massima fissa per ragioni storiche.

Stack ed altro sono dettagli implementativi.

# 4.1.8 Modello di memoria

JVM con unico *"managed heap"*:

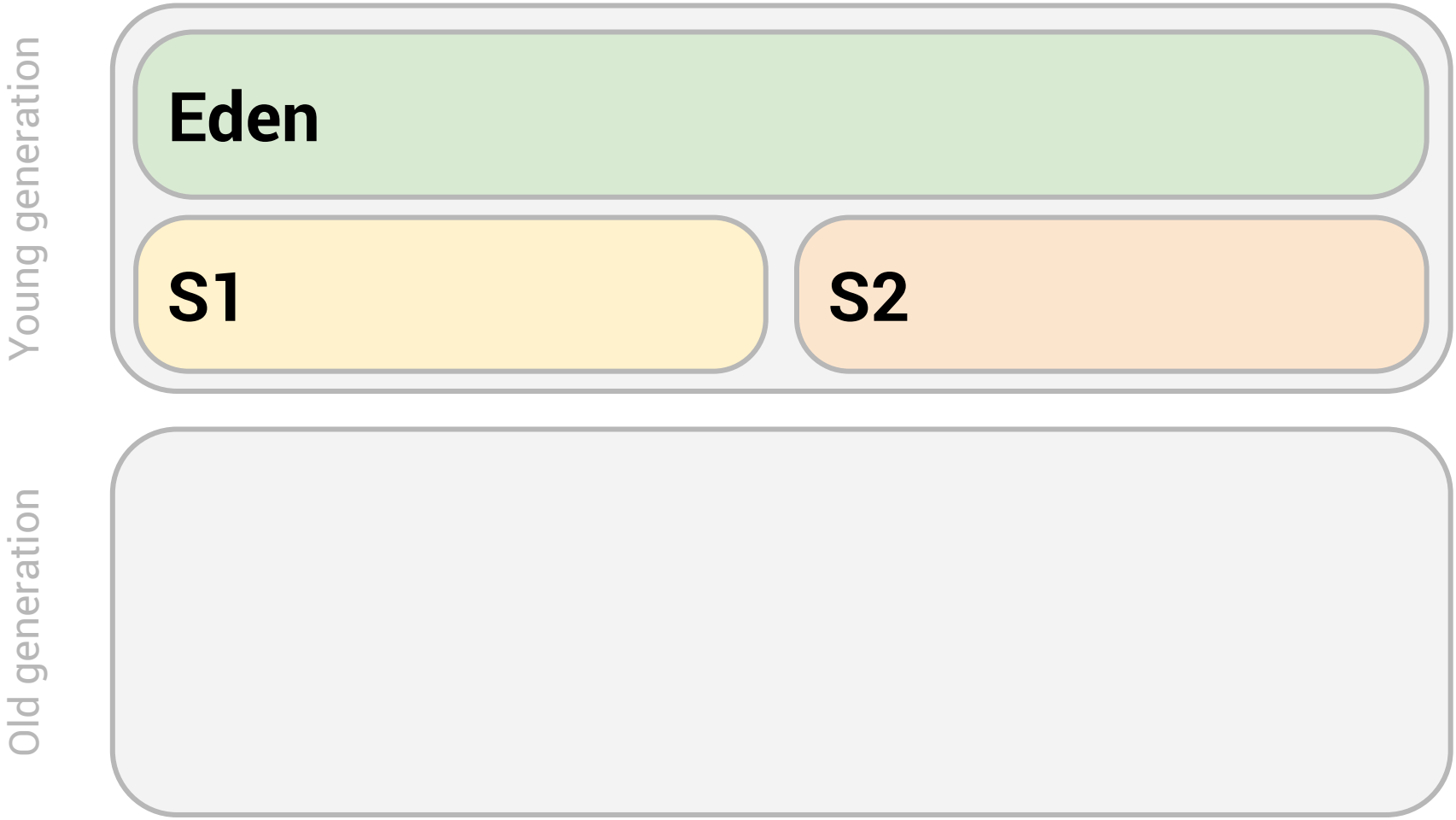
- Un
- Ind
  co
- All
- Par
  ma

Stack e [illegible] sono dettagli implementativi.

Infatti **Dalvik** interpreta codice scritto in Java e *non* prevede un'architettura a stack, bensì a registri.

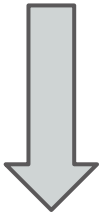
Non interessa il programmatore.

# 4.1.8 Garbage collection



# 4.1.8 Garbage collection (2)

- *Minor* collection su generazioni giovani.
- *Major* collection su tutto *heap*.

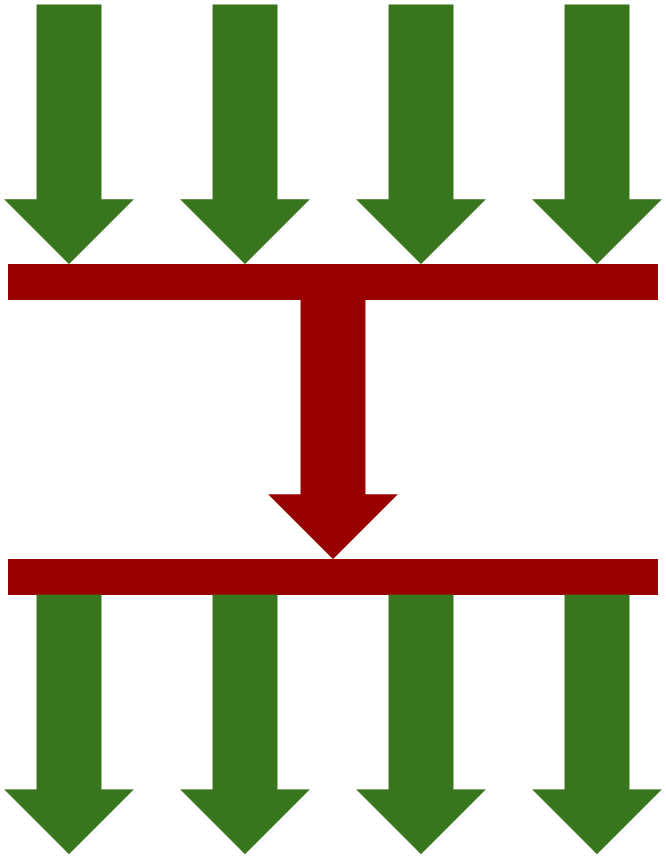


Garbage collection è *bloccante*.

Se *heap* affollato, può svolgere compattamento della memoria.

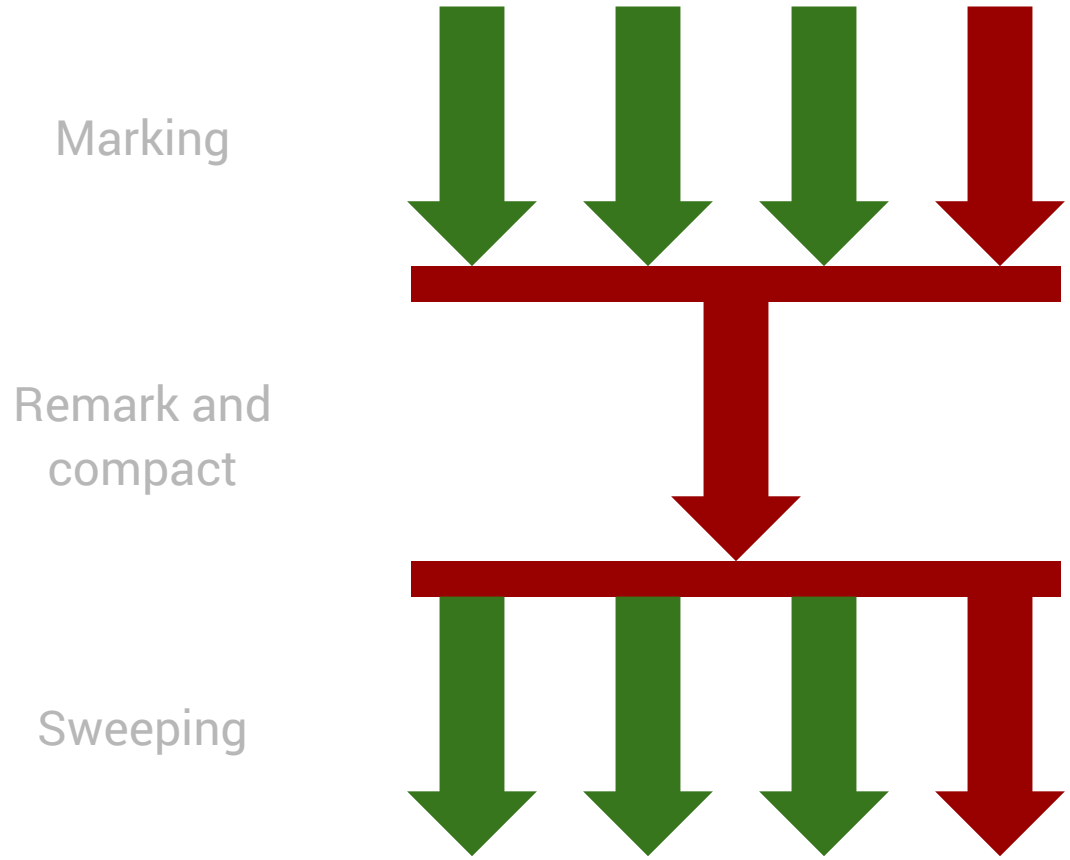
# 4.1.8 Garbage collection (3)

Garbage collector thread



Serial/Parallel Sweep & Compact

# 4.1.8 Garbage collection (4)




Concurrent Mark & Sweep

# 4.1.8 Garbage collection su Android

Garbage collector *"tracing"* ("reachable" bit):

- Senza generazioni.
- Mark & Sweep.
- Bloccante.
- Multithreaded.

# 4.1.9 XML: eXtended Markup Language

Formato per la strutturazione dei dati.

*Estensibile*: ossia, può essere adattato a svariati usi.

- HTML, per gli ipertesti.
- RSS.
- Android manifest.

# 4.1.9 XML: eXtended Markup Language

Rappresenta una **gerarchia** di elementi,
ognuno dei quali può contenere altri elementi.

```
<world>
  <europe>
    <italy code="it" />
    <germany code="de" />
    ...
```

Ogni elemento può avere degli attributi.

# 4.1.9 XML: eXtended Markup Language

La struttura e gli elementi sono definiti da namespace e DTD.

Ci interessa per **manifest** e **layout** Android:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="string"
          android:sharedUserId="string"
          android:sharedUserLabel="string resource"
          android:versionCode="integer"
          android:versionName="string"
          android:installLocation=["auto" | "internalOnly" | "preferExternal"] >
    . . .
</manifest>
```

# 4.2

Panoramica SDK

# 4.2 Requisiti per lo sviluppo

- PC/Mac
- ADT
- Smartphone/Tablet
- JDK 1.6+

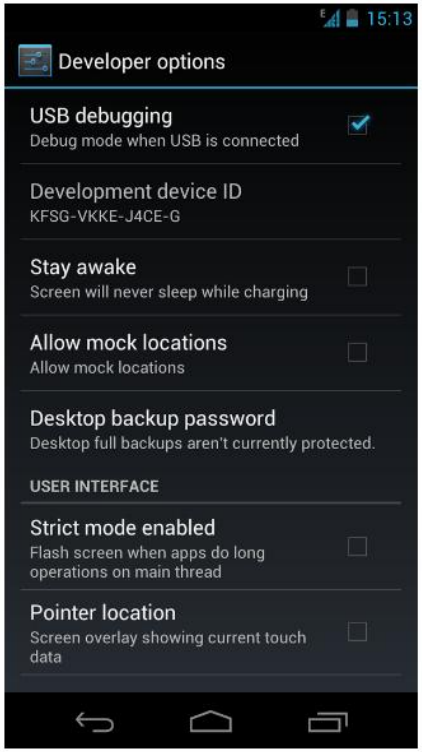
- account Google Play per la pubblicazione

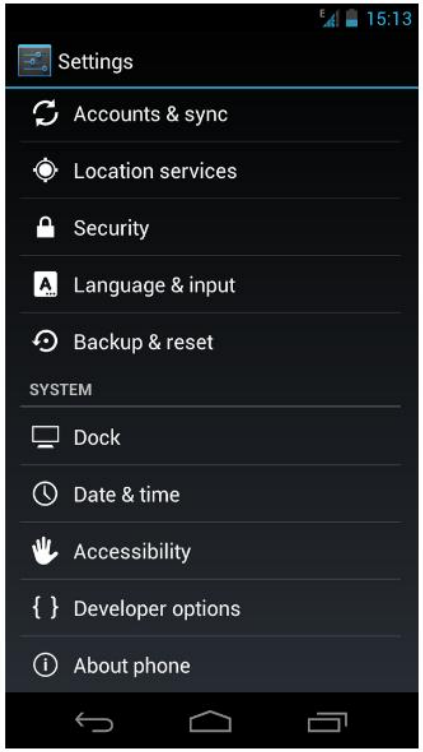
Google play

# 4.2 Requisiti per lo sviluppo

- Device debug mode

- *trick per Android 4.2+*
  - Impostazioni
  - Info sul telefono
  - 7 tap su "Numero Build"



# 4.3 Installazione

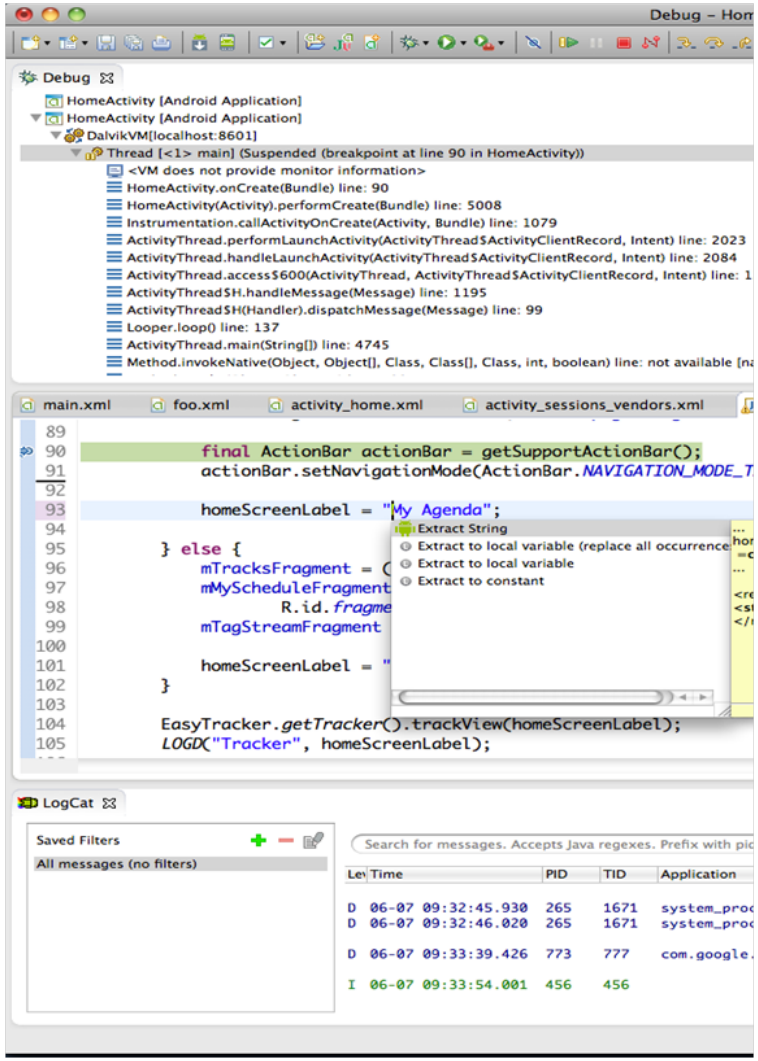
- Installazione JDK specifica per ogni S.O
- ADT bundle: zip omnicomprensivo
  - http://developer.android.com/sdk/index.html
- Alternativamente
  - Eclipse
  - Android SDK tools
  - Plugin ADT per Eclipse
- Istruzioni dettagliate nel Google Group

# 4.4 Strumenti

- Eclipse
- Android Developer Tools
- Dalvik Debug Monitor Server
- Emulatore

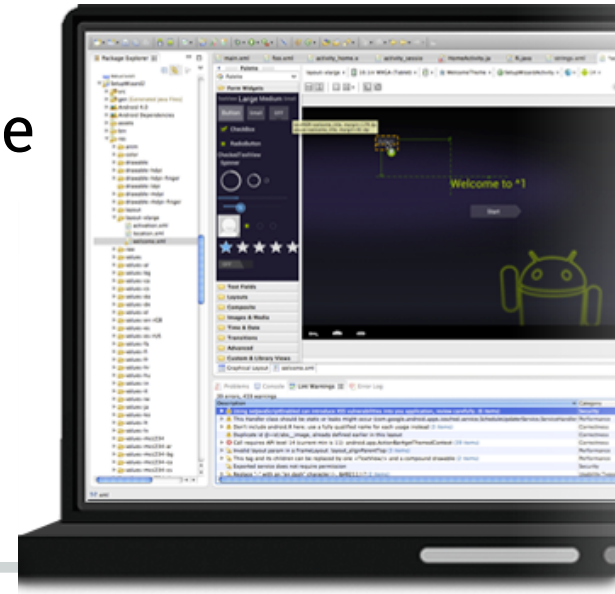
# 4.4.1 Eclipse

- Ambiente di sviluppo open-source, multi-piattaforma e multi-linguaggio.
- È la piattaforma di riferimento per Android.
- Alternative:
  - ANT, NetBeans…



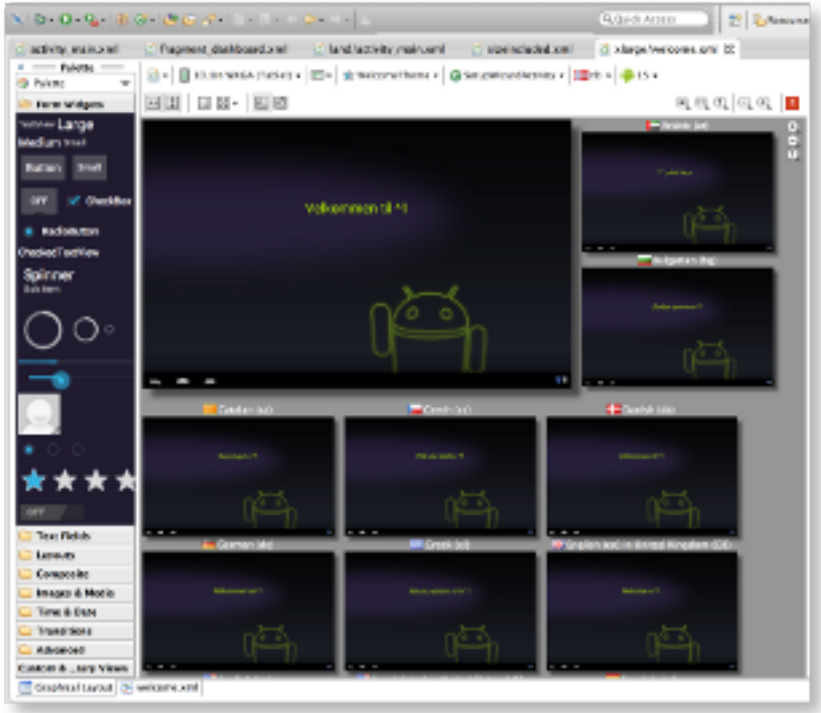
# 4.4.2 Android Developer Tools

- Plugin per l'IDE Eclipse che fornisce strumenti avanzati per semplificare lo sviluppo su Android:
  - Creazione, compilazione, distribuzione e debugging
  - Integrazione con i tool di sviluppo Android (DDMS, logcat, lint, etc.)
  - Editor grafico ed XML per interfacce grafiche e risorse
  - Documentazione integrata in linea



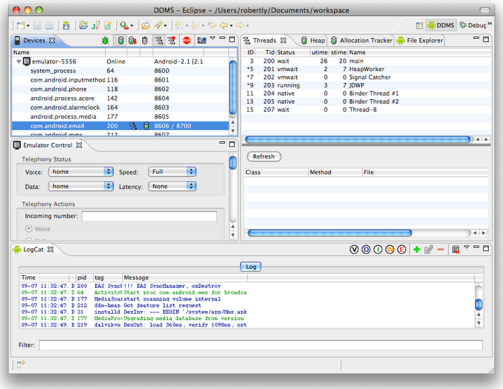
# 4.4.2 Android Developer Tools

- Dalla versione 21 l'editor grafico permette
  - il preview contemporaneo su target personalizzati
    - oneroso
  - il refactoring delle risorse nell'XML



# 4.4.3 Dalvik Debug Monitor Server

- Strumento adatto al debug di un dispositivo Android che consente di catturare screenshot, raccogliere informazioni relative a processi, thread ed heap, interfacciarsi a logcat ed inviare informazioni al dispositivo per l'emulazione di eventi.

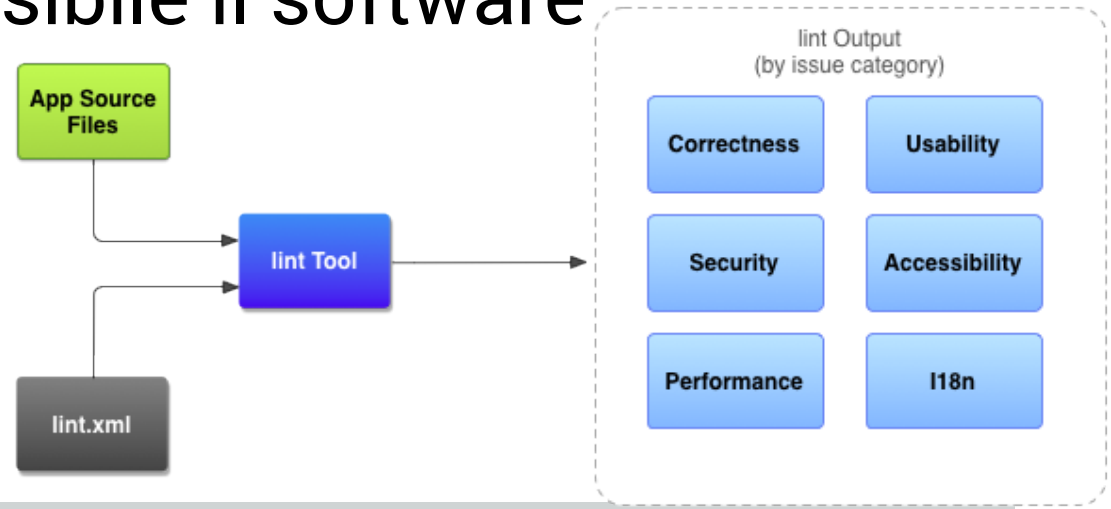


# 4.4.4 Logcat

- Software adatto al debug che consente di raccogliere i log prodotti da un dispositivo Android.
- Consente di filtrare i messaggi in base a TAG specifici, applicazioni, livelli di importanza, etc.

# 4.4.5 Lint

- Software adatto all'ottimizzazione che effettua un'analisi statica del codice prodotto alla ricerca di errori comuni che potrebbero rallentare, rendere meno sicuro, usabile ed accessibile il software sviluppato.



# 4.4.6 Proguard

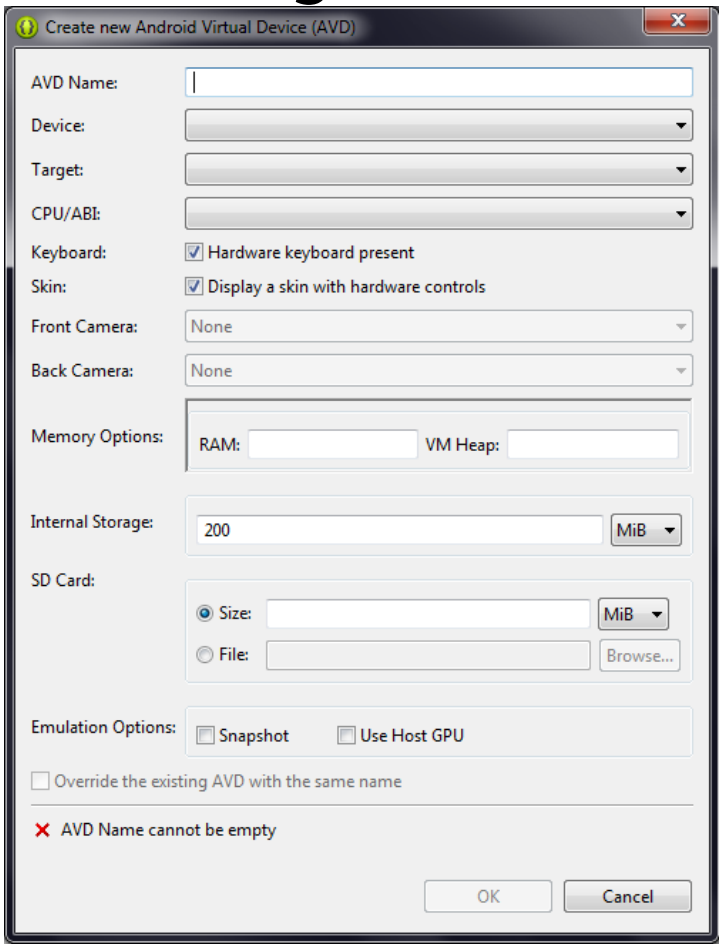
- Strumento che riduce la dimensione del codice prodotto, lo ottimizza e lo offusca per renderlo più robusto nei confronti del reverse engineering e renderne l'esecuzione più rapida.

# 4.4.7 Emulatore

- Software che emula il funzionamento e l'aspetto di una configurazione hardware virtuale (AVD: Android Virtual Device).

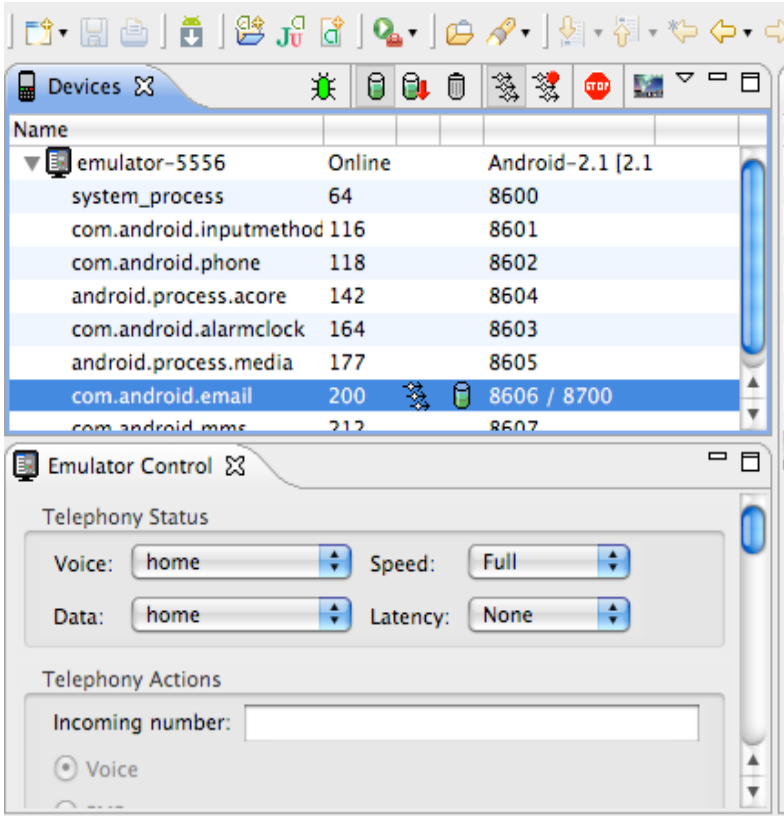
# 4.4.7 Emulatore

- Interfaccia *user-friendly* per configurare i propri AVD
  - Target API
  - CPU/ABI
  - RAM / Heap
  - Storage
  - Fotocamere
  - Skin



# 4.4.7 Emulatore

- Possibilità di pilotare
  - Stato di rete per voce e dati
  - Velocità e latenza della rete
  - Arrivo di chiamate e/o SMS
  - Posizione geografica (manuale, GPX, KML)



# 4.4.8 Google APIs

- Insieme di API fornite da Google fondamentali per alcuni utilizzi specifici
  - Mappe (Google Maps)
  - Acquisti In-App (Google Play In-App billing)
  - Google+
  - Statistiche (Google Analytics)
  - Notifiche push (Google Cloud messaging)
  - Pubblicità (ADmob ads)

# 4.4.9 Support library

- Libreria statica che fa il back-porting di molte API fornite nelle nuove versioni di Android ed in particolare Fragment, Loader e notifiche
  - progetti indipendenti intorno a questa libreria di supporto per il porting quasi totale delle API (**ActionBarSherlock**)

# 4.4.10 Demo

Logcat

# 4.4.10 Demo

Toast

# 4.4.10 Demo

TextView

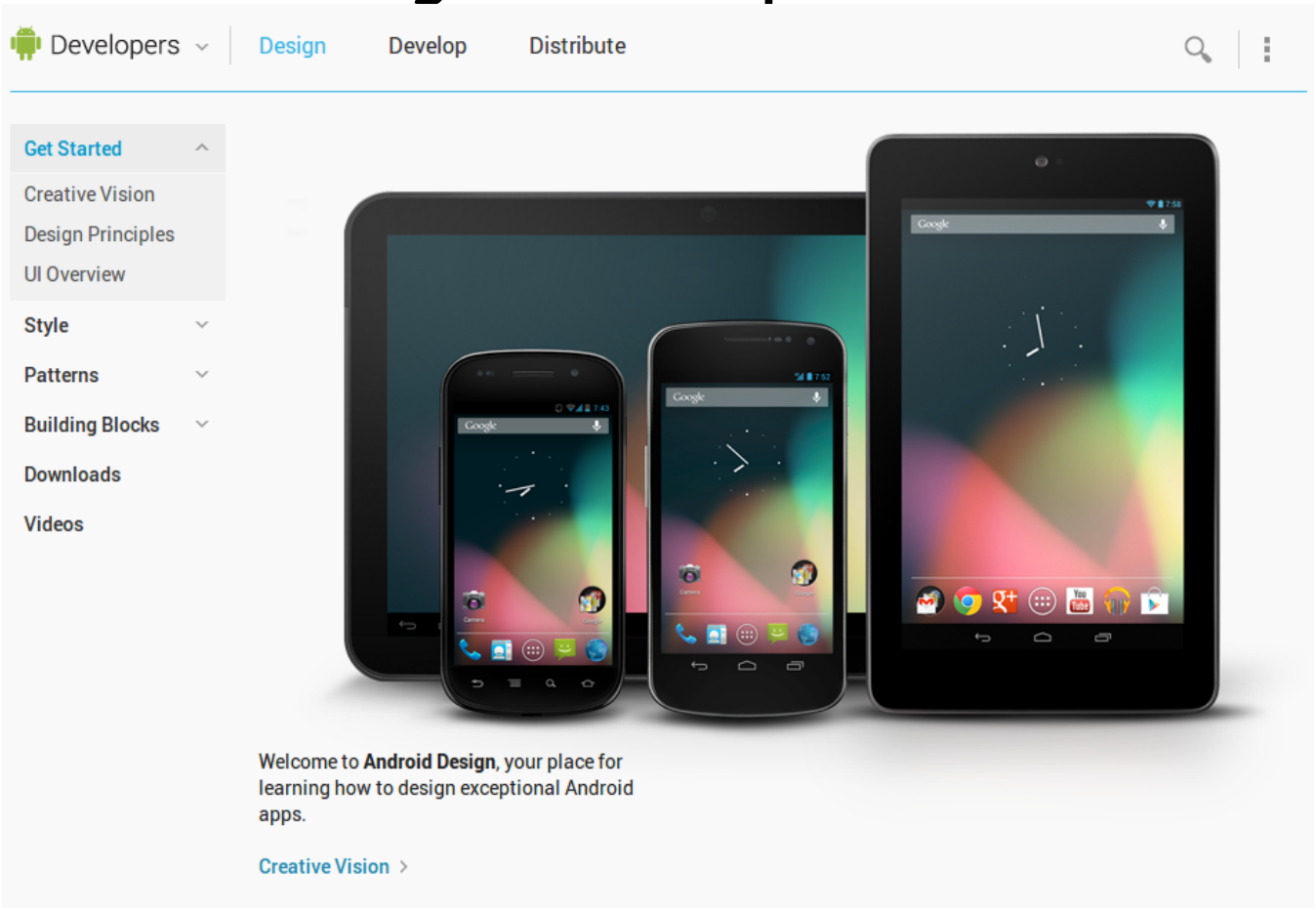
# 4.4.11 Documentazione

- Risorse online ed all'interno dell'IDE

- Per esempio per una TextView
  - on-line http://developer.android.com/reference/android/widget/TextView.html
  - e la documentazione all'interno di Eclipse

# 4.4.11 Documentazione

- Struttura design, develop, distribute

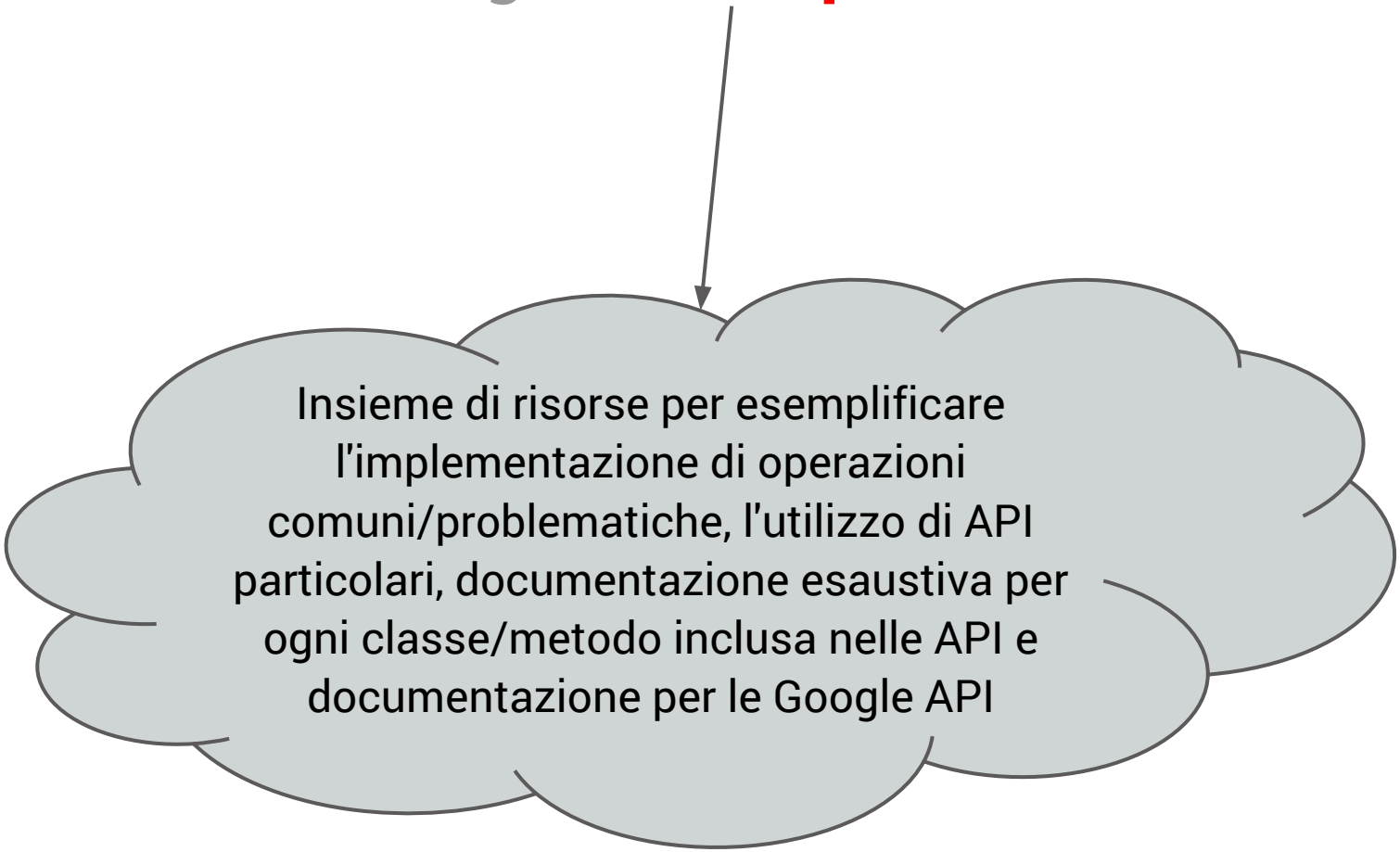


# 4.4.11 Documentazione

- Struttura **design**, develop, distribute

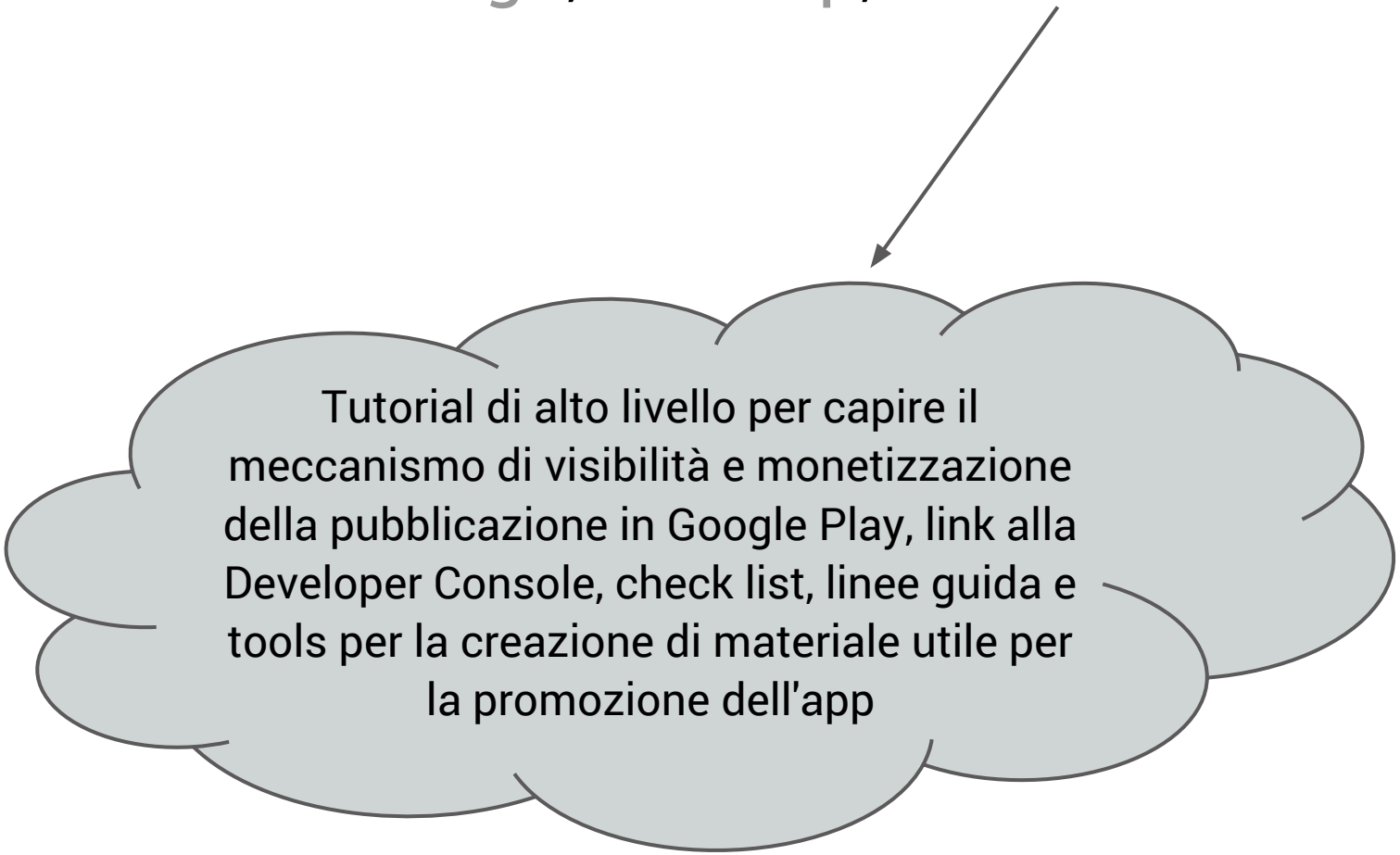
# 4.4.11 Documentazione

- Struttura design, **develop**, distribute



# 4.4.11 Documentazione

- Struttura design, develop, **distribute**



# 4.5

# Struttura base di un progetto

# 4.5 Struttura base di un progetto

File e directory.

Gestione delle risorse.

Manifest dell'applicazione.

# 4.5.1 Struttura base di un progetto

File e directory.

Gestione delle risorse.

Manifest dell'applicazione.

# 4.5.1 Directories

**/src**: sorgenti Java (oppure .aidl o altri)

**/bin**: compilati (.apk, ecc…)

**/libs**: librerie esterne

**/gen**: sorgenti generati (R.java)

**/assets**: risorse unmanaged (nel .apk)

**/res**: risorse managed (processati)

**/AndroidManifest.xml**: metadati

**.properties, .cfg, .xml**: altri files

# 4.5.2 Gestione delle risorse

Demo

# 4.5.2 Struttura base di un progetto

File e directory.

Gestione delle risorse.

Manifest dell'applicazione.

# 4.5.2 Gestione delle risorse

File, stringhe, dimensioni, colori, drawable, ecc.

Ad ogni risorsa all'interno della directory ***/res*** viene assegnato un identificativo.

R.string.hello

Ogni *identificativo* corrisponde ad entry in una mappa <identificativo, ID intero>.

# 4.5.2 Gestione delle risorse

In pratica...

Viene generata una classe R.java con:

Una classe `public static final` per ogni tipo di risorsa.

Un campo intero con l'identificativo della risorsa.

La classe viene aggiunta nel namespace base.

# 4.5.2 Gestione delle risorse

Demo

# 4.5.2.1 Dichiarazione delle risorse

Dichiarazione tramite XML:

```
<string name="hello_world"> ciao mondo </string>
<Button android:id="@+id/nice_button" />
```

I file vengono creati/importati nelle specifiche sotto-directory di **/res**.

# 4.5.2.2 Recupero delle risorse

Tramite XML:

```
@[<package_name>:]<resource_type>/<resource_name>
<Button android:text="@string/hello_world" />
```

Tramite codice:

```
[<package_name>.]R.<resource_type>.<resource_name>
ImageView imgView = (ImageView)findViewById(R.id.myimageview);
imgView.setImageResource(R.drawable.myimage);
```

# 4.5.2.3 Risorse e configurazione

Due (o più) risorse possono condividere lo stesso *identificativo*.

Come mai?

Due risorse con lo stesso identificativo costituiscono due *versioni* della stessa risorsa (alternative tra loro).

# 4.5.2.3 Risorse e configurazione

Le versioni delle risorse sono caratterizzate dal nome della directory in cui sono locate.

- *layout-long*
- *anim-v14*
- *drawable-en-rUS-land-night-notouch*

Lingua in inglese, in America, orientato orizzontalmente, è notte e non si dispone di interfaccia touch.

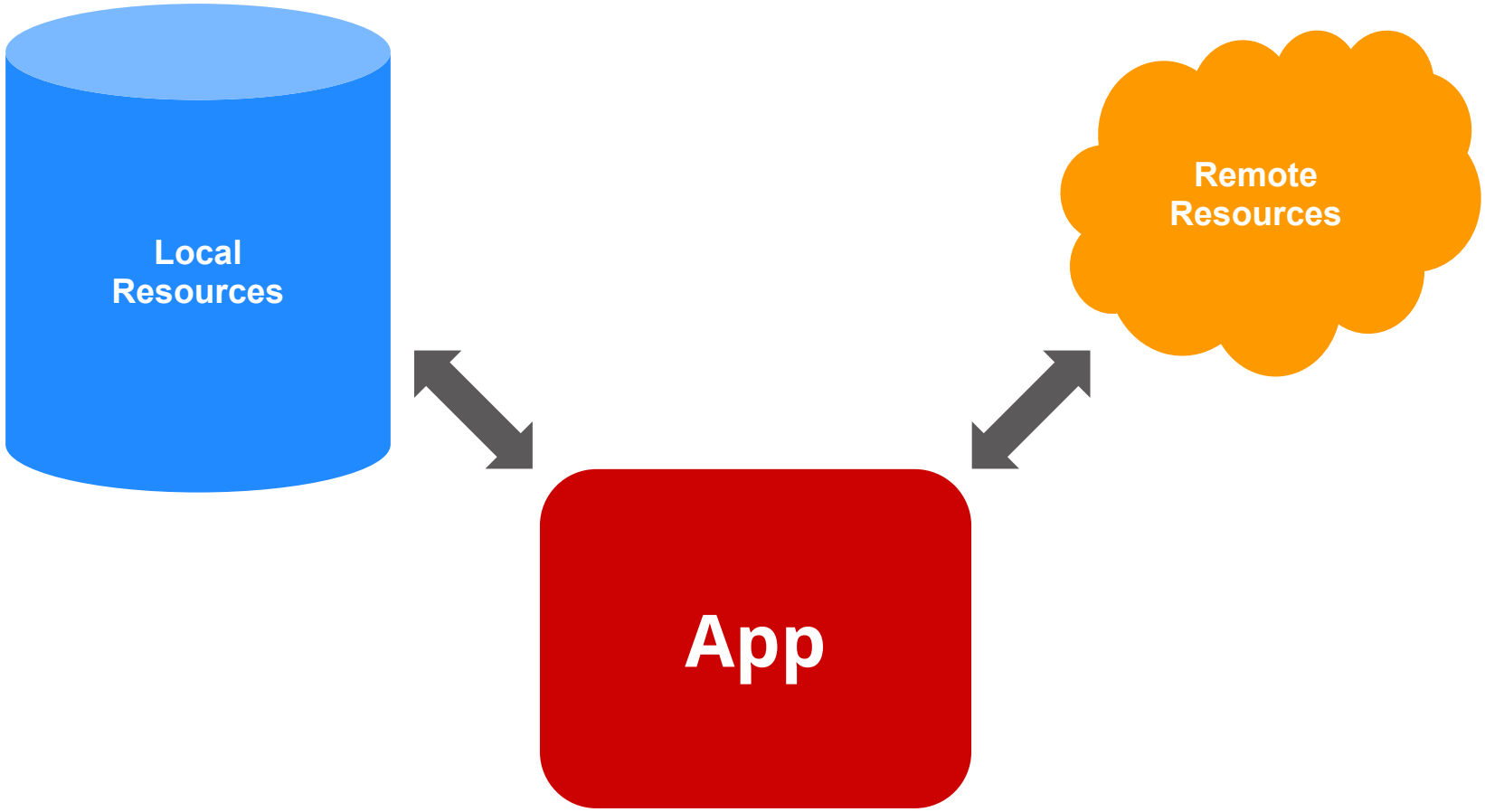
- Risorsa scelta a runtime dal framework.

# 4.5.2.3 Risorse e configurazione

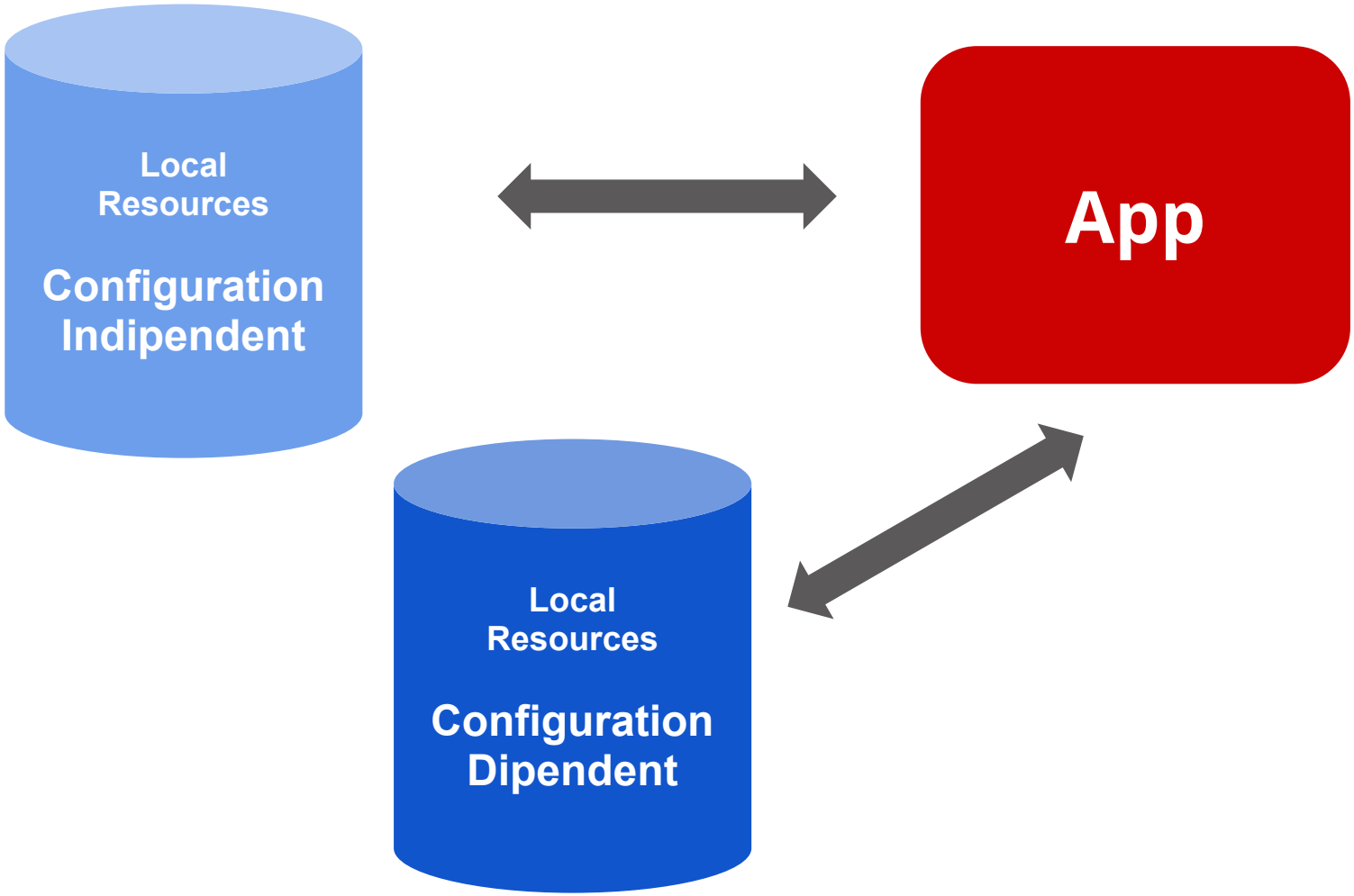
Demo

# 4.5.2.3 Risorse e configurazione

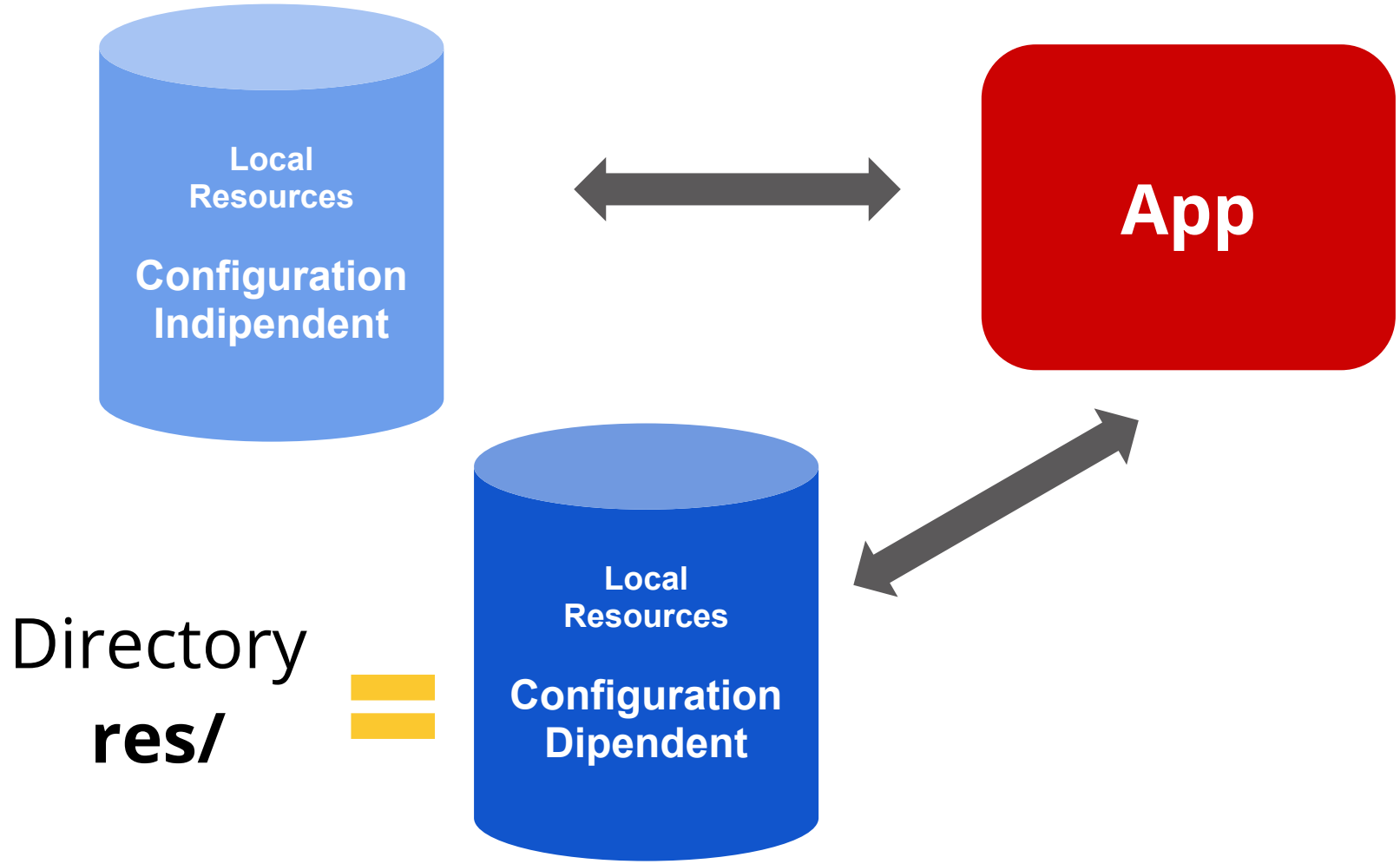
# 4.5.2.3 Risorse e configurazione



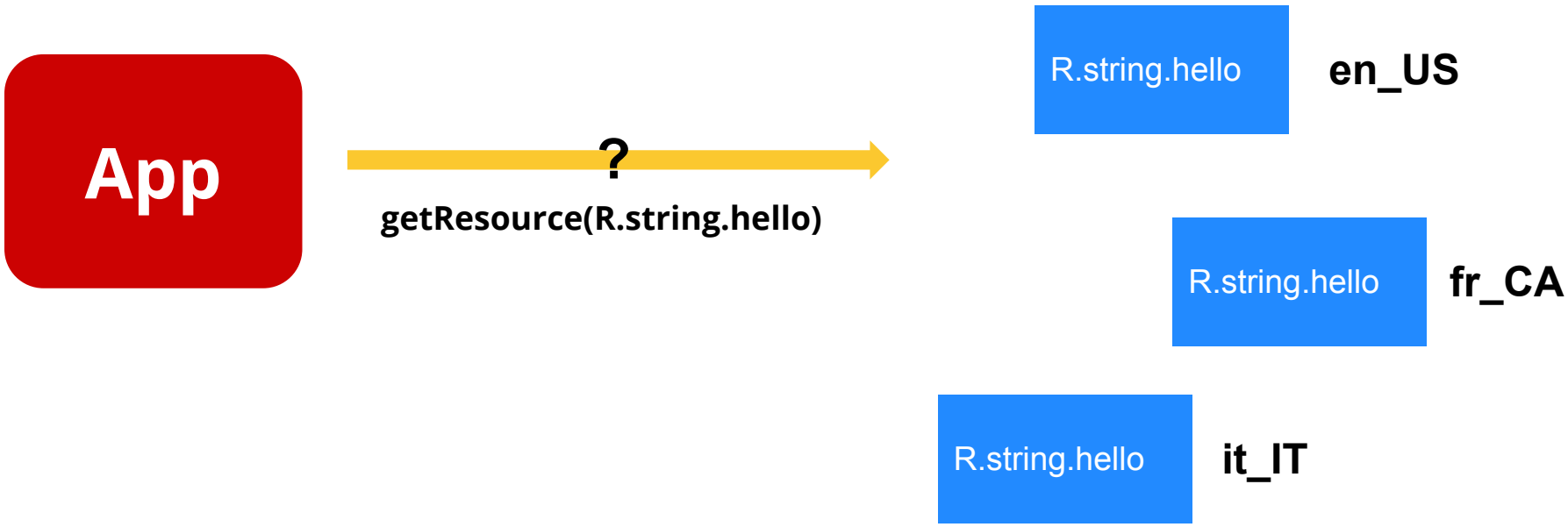
# 4.5.2.3 Risorse e configurazione



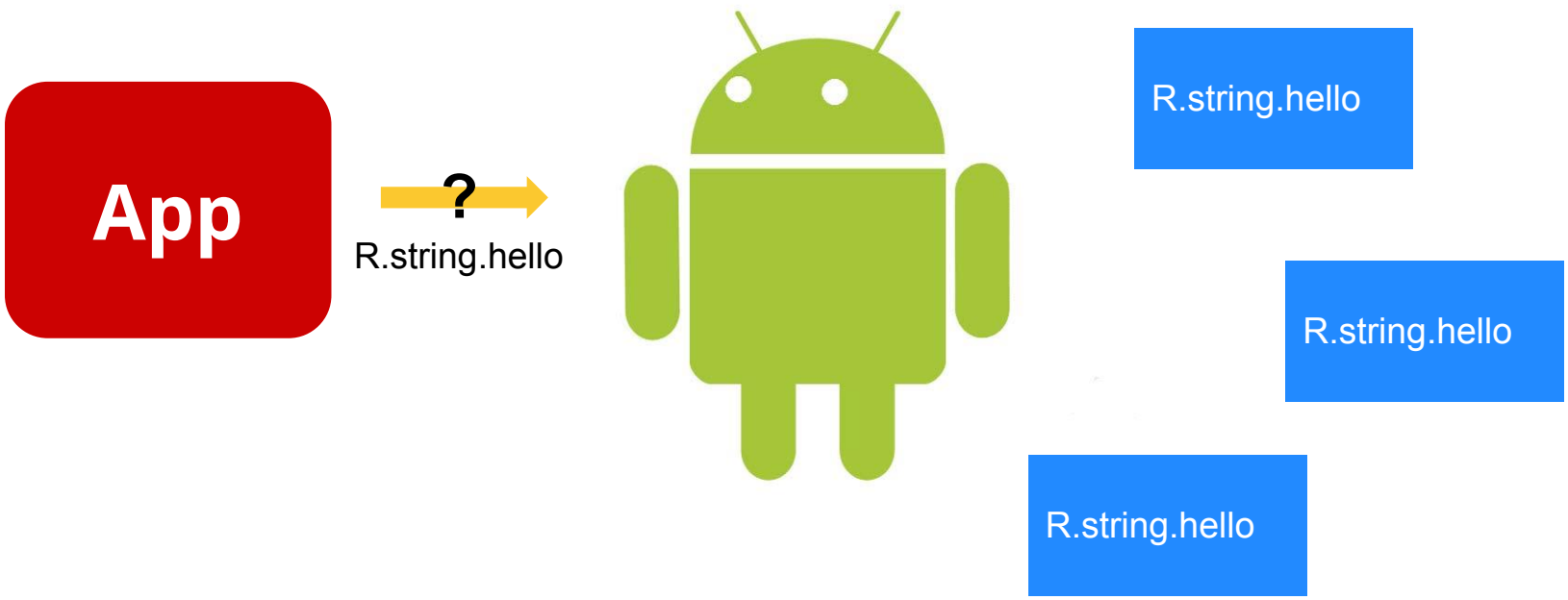
# 4.5.2.3 Risorse e configurazione



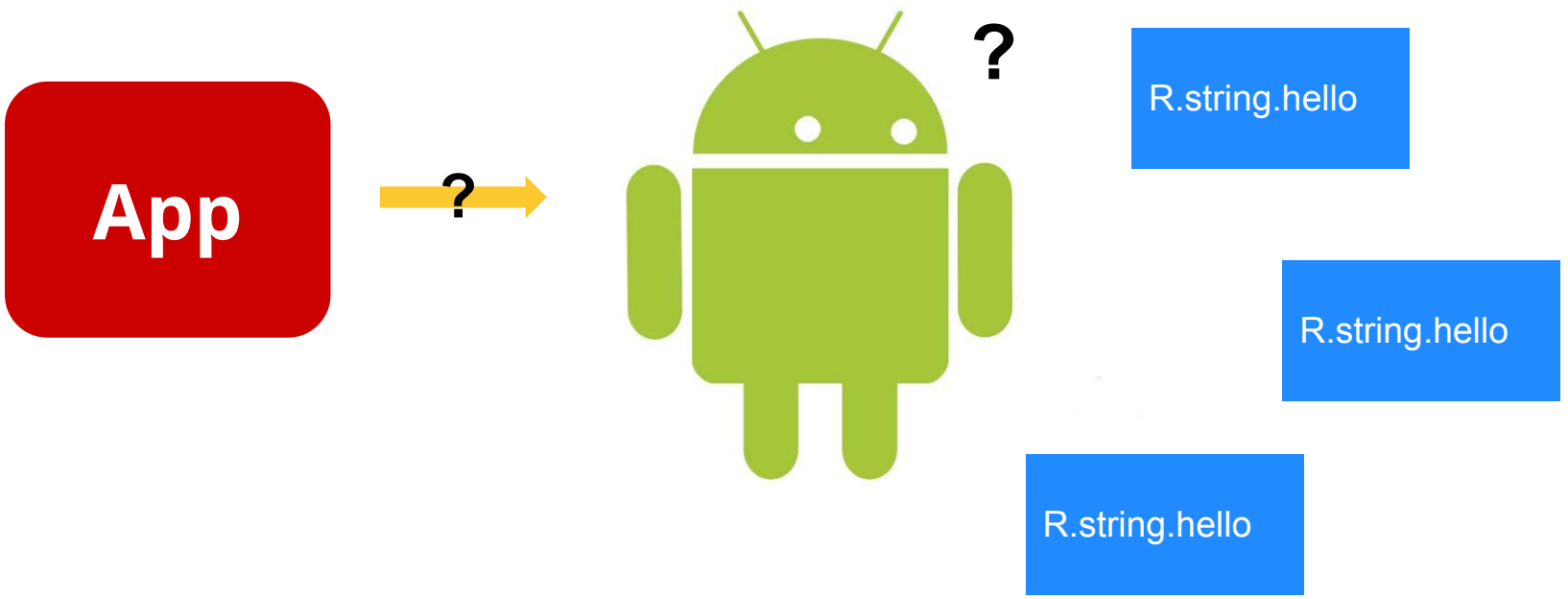
# 4.5.2.3 Risorse e configurazione



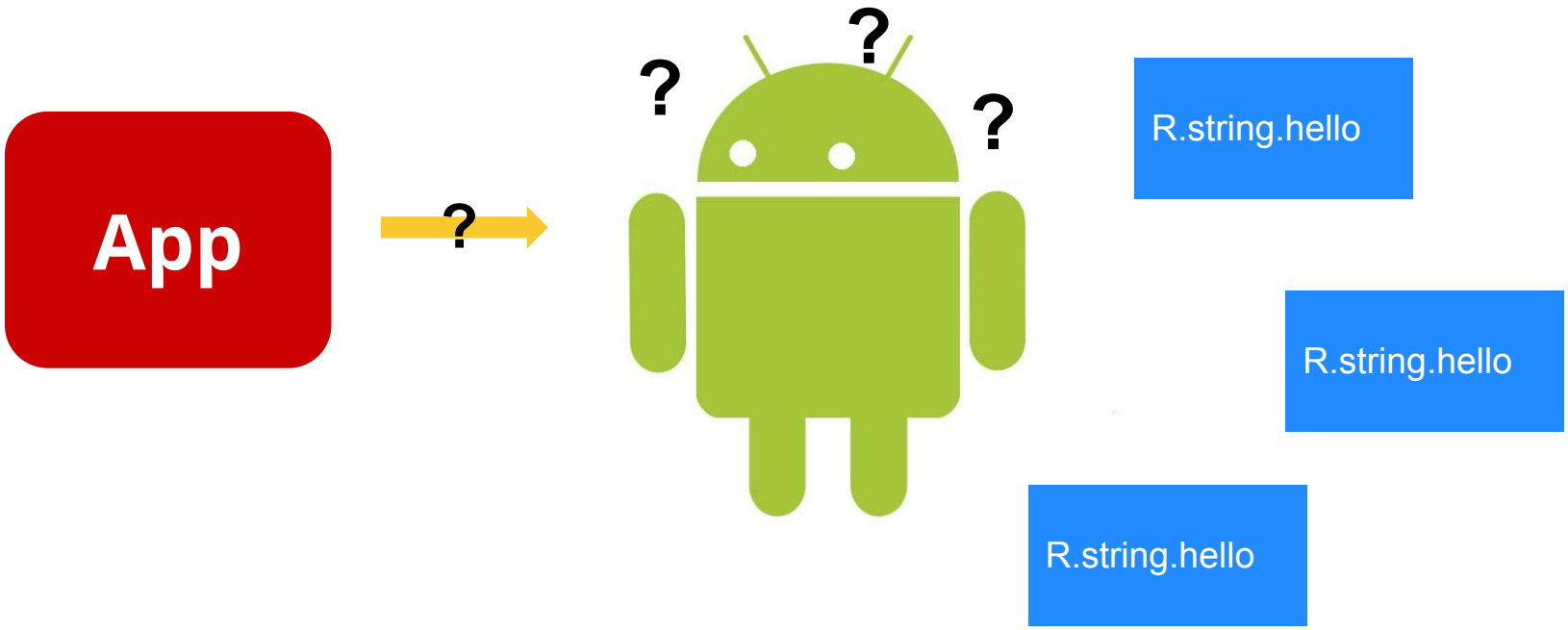
# 4.5.2.3 Risorse e configurazione



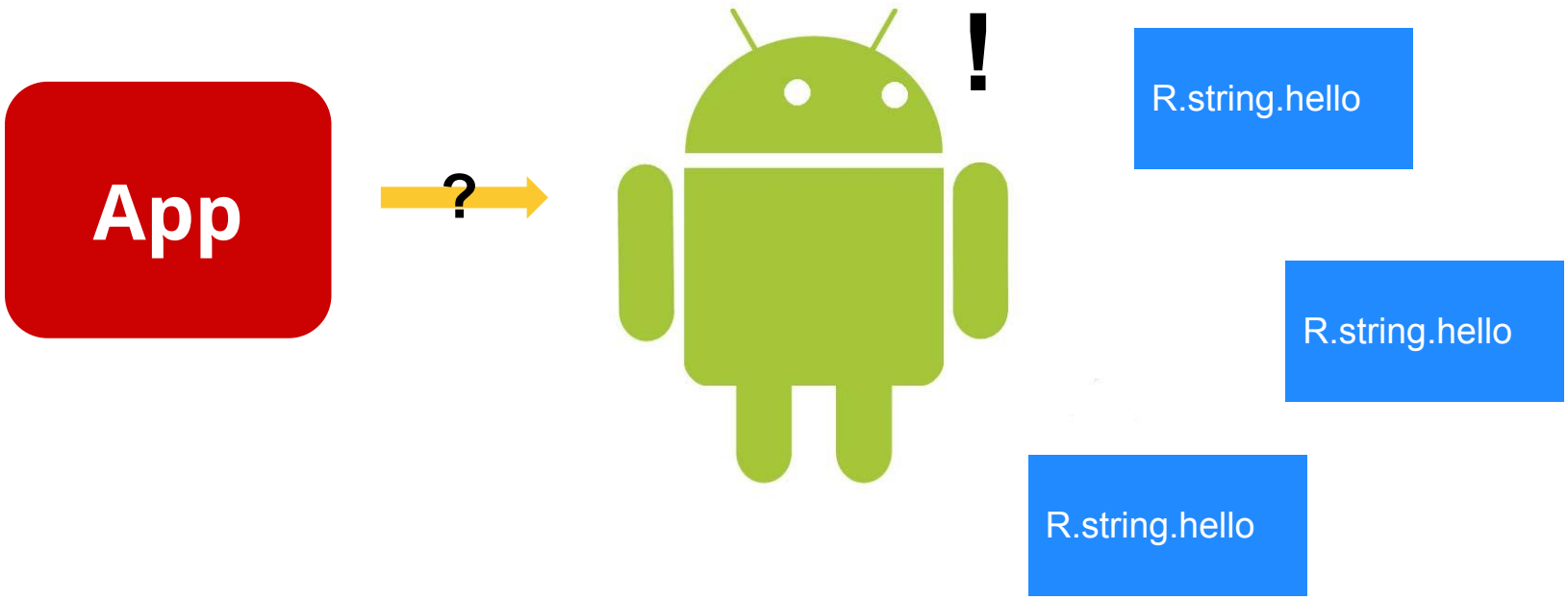
# 4.5.2.3 Risorse e configurazione



# 4.5.2.3 Risorse e configurazione



# 4.5.2.3 Risorse e configurazione



# 4.5.2.3 Risorse e configurazione

App

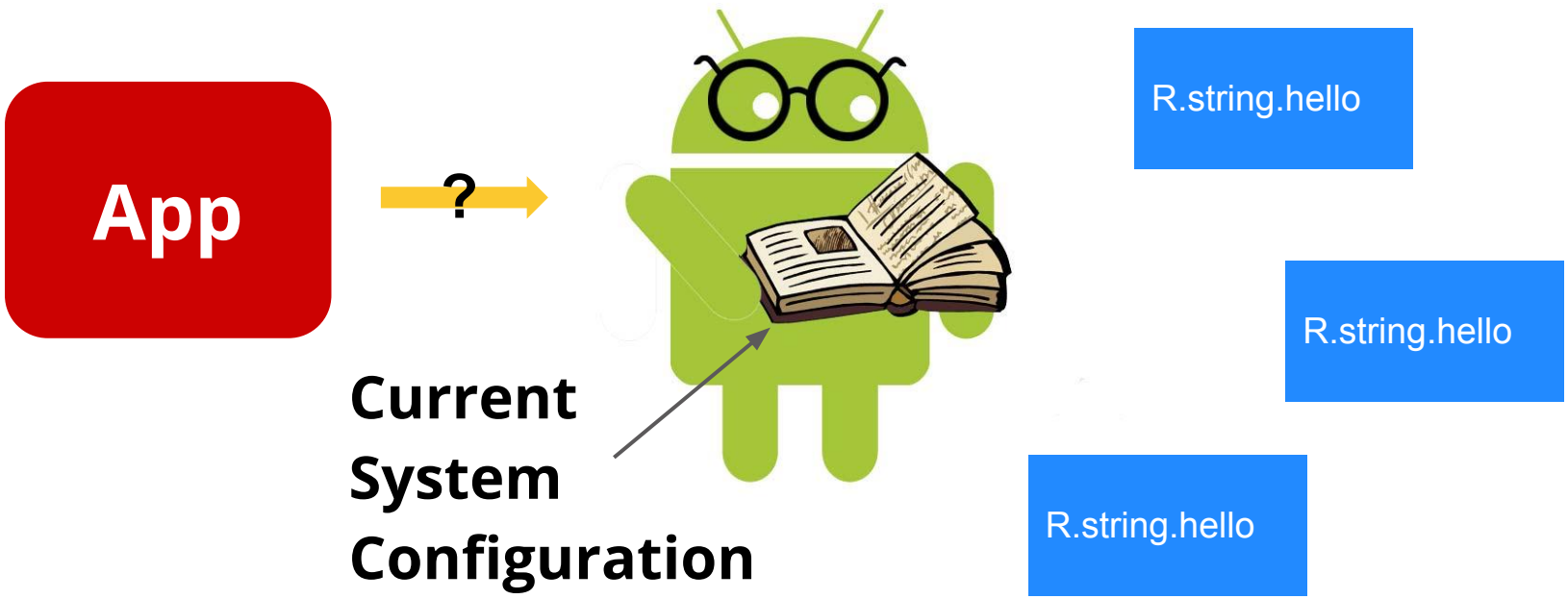
?

R.string.hello

R.string.hello

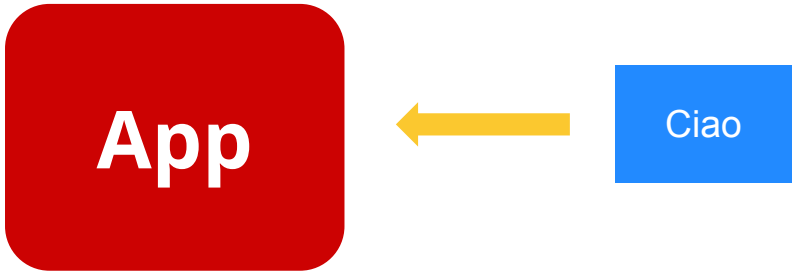
R.string.hello

# 4.5.2.3 Risorse e configurazione



# 4.5.2.3 Risorse e configurazione

# 4.5.2.3 Risorse e configurazione

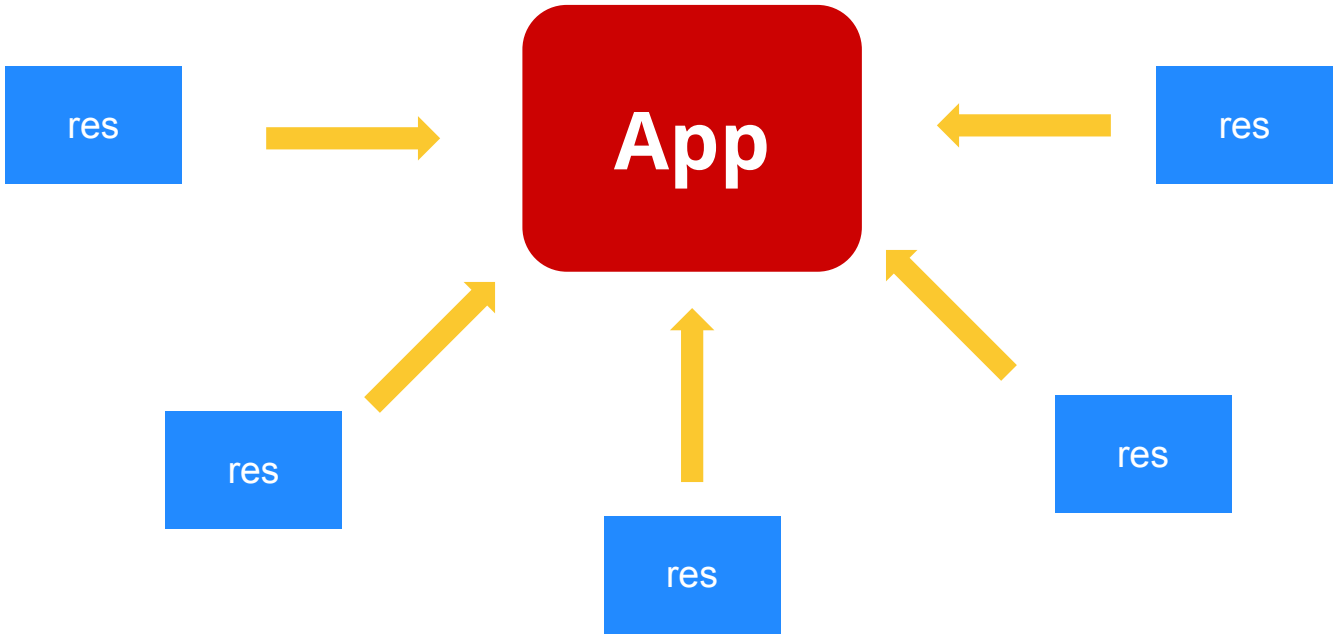


# 4.5.2.3 Risorse e configurazione

Demo

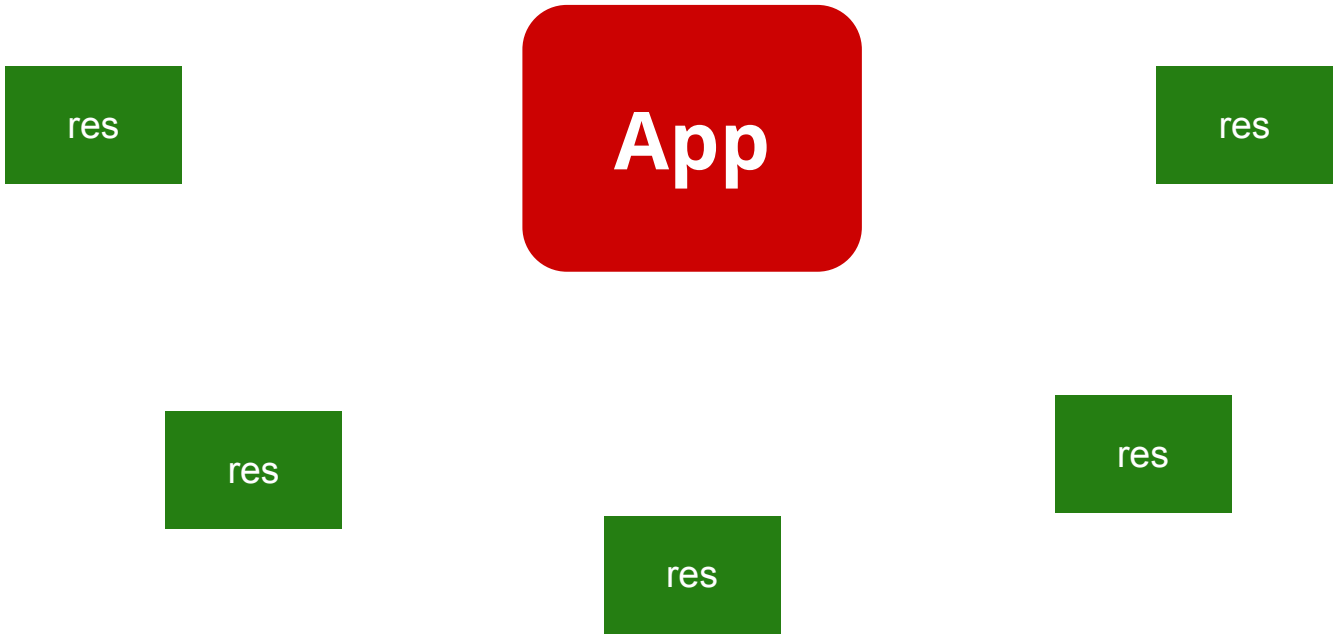
# 4.5.2.4 Cambio di configurazione

Cosa avviene se la configurazione cambia?



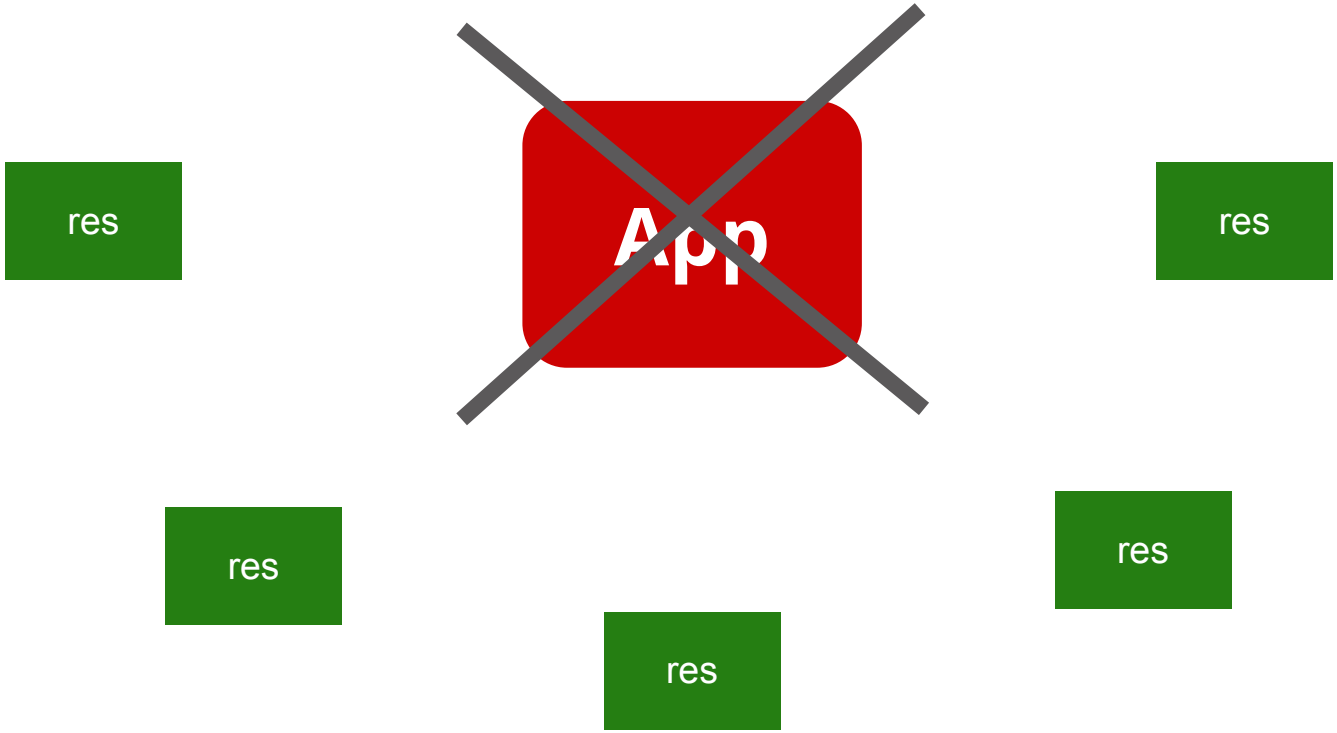
# 4.5.2.4 Cambio di configurazione

Cosa avviene se la configurazione cambia?



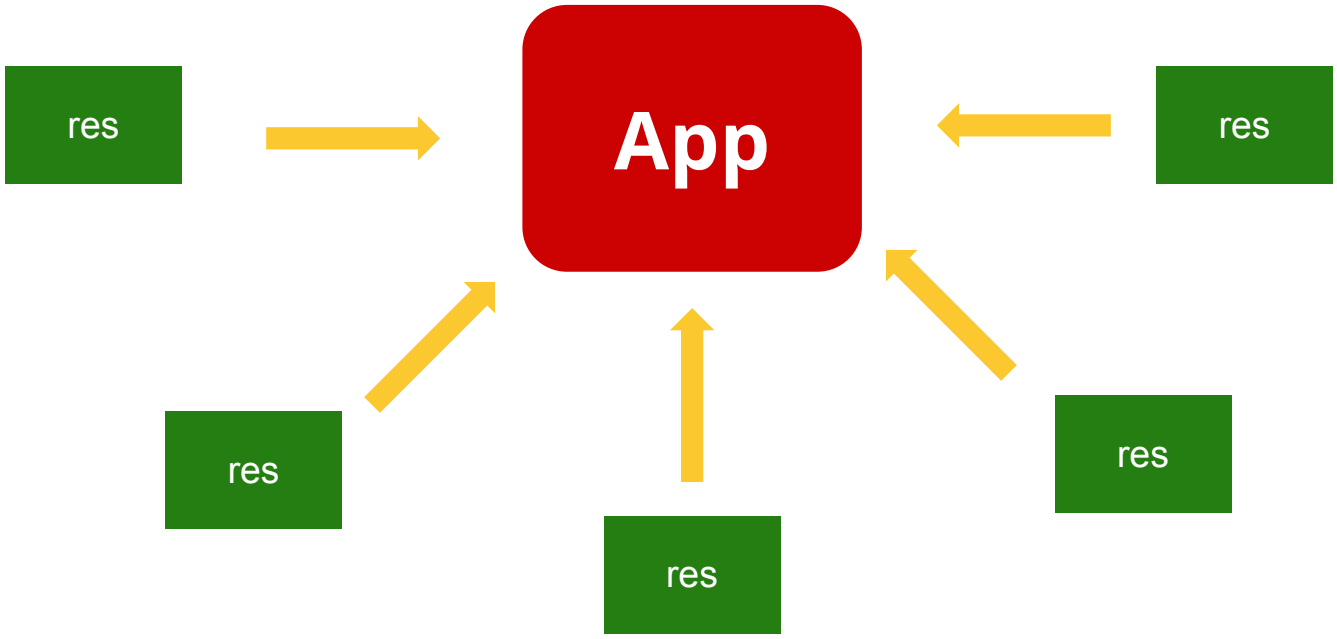
# 4.5.2.4 Cambio di configurazione

Il componente che usa le risorse viene ricreato



# 4.5.2.4 Cambio di configurazione

...con le risorse corrette

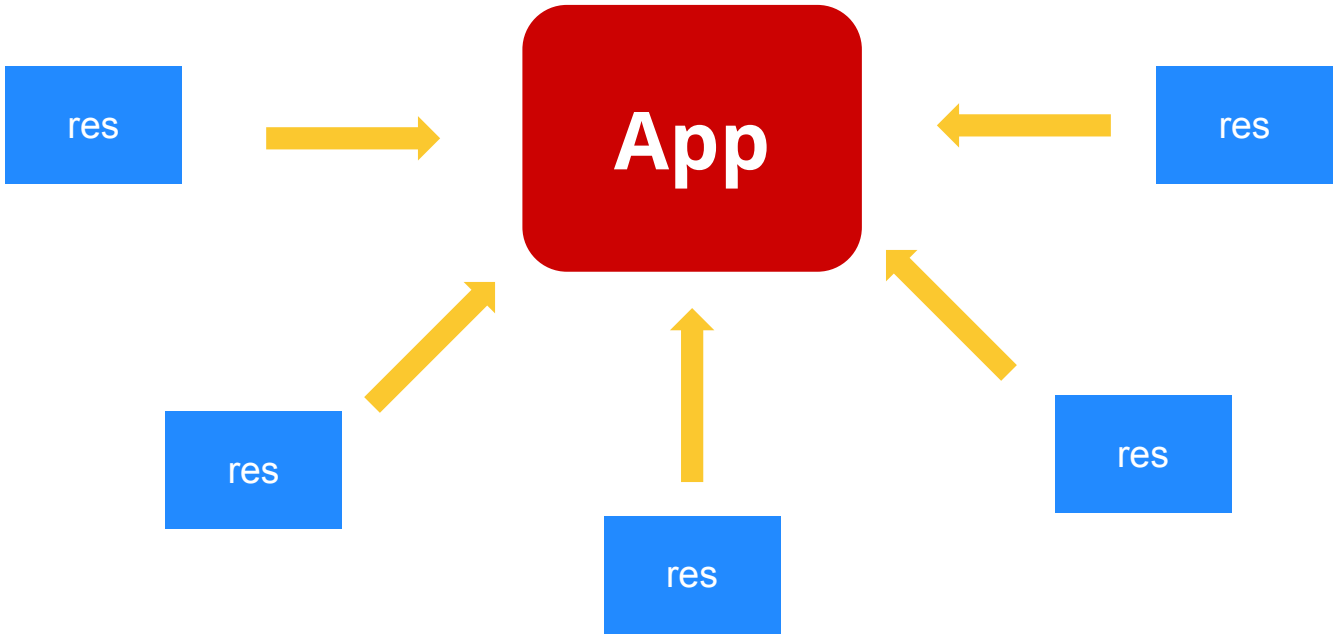


# 4.5.2.4 Cambio di configurazione

Demo

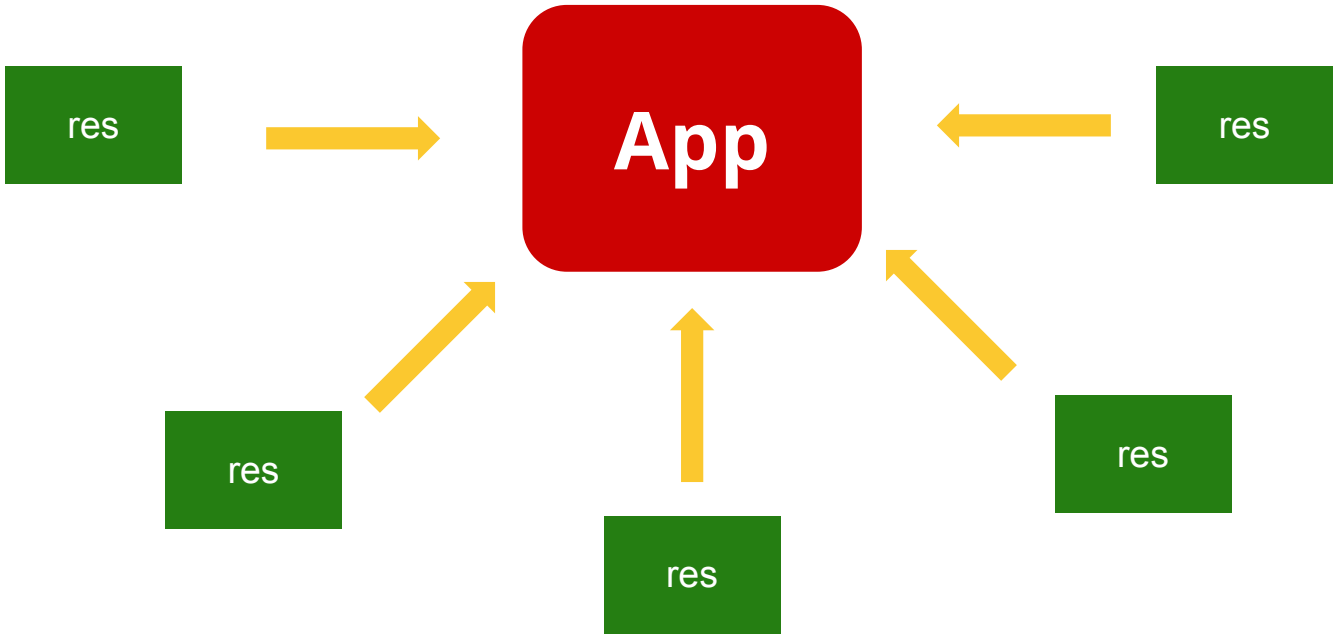
# 4.5.2.4 Cambio di configurazione

Se il comportamento del componente non deve varia allora...



# 4.5.2.4 Cambio di configurazione

il cambio di configurazione può essere gestito dal programmatore (`onConfigurationChange`).



# 4.5.2.4 Cambio di configurazione

Demo

# 4.5.3 Struttura base di un progetto

File e directory.

Gestione delle risorse.

Manifest dell'applicazione.

# 4.5.3 Manifest

File `AndroidManifest.xml` nella root.

Descrive l'applicazione, il nome del package, il namespace, i permessi.

Dichiara che livello di API è richiesto.

Include la specifica di ogni componente: Activity, Service, Receiver, etc.

# 4.5.3 Manifest

Demo