“Pensando Android”

Introduzione alla programmazione mobile con Android

Di che cosa parleremo...

- Che cos'è Android?
- Diffusione di Android
- Linguaggio di programmazione Java su Android
- Prime applicazioni su Android
- Esempi Pratici

Di che cosa non parleremo...

- Html 5 con Android (esempi Phonegap e Titanium studio)
- Sono applicazioni che facilitano molto la programmazione ma non garantiscono la creazione di progetti complessi
- La **piattaforma nativa** di programmazione ottimizzazione e controllo del codice, debug, facilità nel reperire documentazione, informazioni e componenti di terze parti.
- E' necessario capire anche il suo **linguaggio di programmazione.**

Android

- Android è un sistema operativo per dispositivi mobili costituito da uno stack software che include un sistema operativo di base, i middleware per le comunicazioni e le applicazioni di base.Caratteristiche principali di Android sono la struttura **open source** e il suo basarsi su kernel **Linux**.
- La piattaforma usa il database *SQLite*, la libreria dedicata *SG*L per la grafica bidimensionale (invece del classico server X delle altre distribuzioni linux) e supporta lo standard OpenGL ES 2.0 per la grafica tridimensionale.
- Le applicazioni vengono eseguite tramite la Dalvik virtual machine, una macchina virtuale adattata per l'uso su dispositivi mobili. Android è fornito di una serie di applicazioni preinstallate: un browser, basato su WebKit, una rubrica e un calendario.
- Dopo una serie di progetti avviati nel 2005 da Google , la prima versione risale al 12 Novembre 2007 dove erano inclusi sdk, librerie del dispositivo e documentazione in Inglese

Tappe crescita di Android

- 2005: Google acquista Android inc.
- 05/11/2007: nasce la Open Handset Alliance
- 12/11/2007: viene rilasciato l'SDK Android
- 23/10/2008: nasce il primo telefono Android – T-Mobile G1

Versioni Android Esistenti

- Android 1.0
- Android 1.5: Cupcake
- Android 1.6: Donut
- Android 2.0: Eclair
- Android 2.2: Froyo
- Android 2.3: Gingerbread
- Android 3.0: Honeycomb (tablet)
- Android 4.0: Ice Cream Sandwich
- Android 4.1: Jelly Bean

Miglioramenti nelle versioni Android

- Android 1.5: Widgets , Tastiera a Schermo, Copia e Incolla, Video
- Android 1.6: Market
- Android 2.0: Multitouch, HTML 5 Browser, Gmaps GPS
- Android 2.1: Effetti 3D
- Android 2.2: JIT Compiler, Tethering, Applicazioni su SD
- Android 2.3: Near Field Communication (tecnologia wireless con connessione a corto raggio fino a 10 cm)
- Android 3.0: Introduzione dei Fragment, CPU Multicore e Interfaccia per i tablet
- Android 4.0: Multitasking migliorato, nuovo tema e riconoscimento facciale

Alcune aziende che utilizzano Android

Sony Ericsson

PHILIPS

ARCHOS

MOTOROLA

acer

ASUS

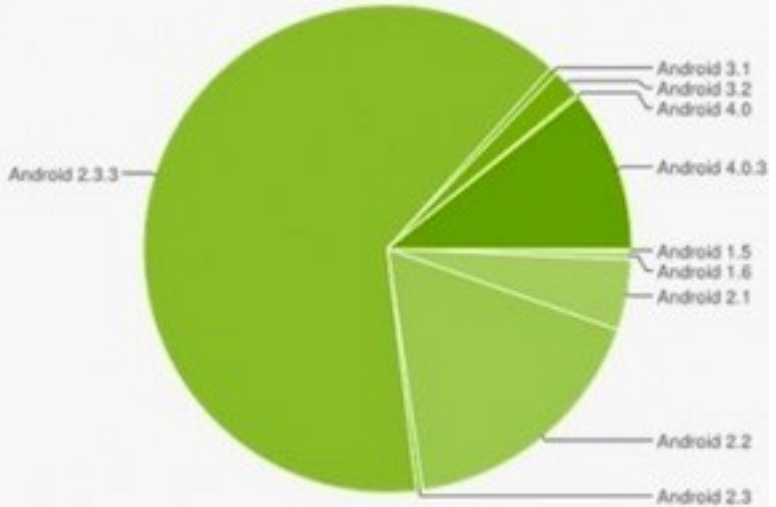
TOSHIBA

DELL

Diffusione di Android

- Grafico del 2 Luglio 2012

Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.7%
2.2	Froyo	8	17.3%
2.3 - 2.3.2	Gingerbread	9	0.4%
2.3.3 - 2.3.7		10	63.6%
3.1	Honeycomb	12	0.5%
3.2		13	1.9%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.2%
4.0.3 - 4.0.4		15	10.7%



Data collected during a 14-day period ending on July 2, 2012

Caratteristiche

- Open source
- Linux
- Framework Applicativo
- Dalvik Virtual Machine (nasconde al Sistema Operativo la gestione della memoria e dei thread)
- Browser integrato basato su WebKit
- SQLite, OpenGL, Xml
- Rich development environment
- Supporto multimediale
- Bluetooth, EDGE, 3G, WiFi, NFC
- Fotocamera, GPS, bussola, accelerometro,giroscopio, sensore di prossimità, sensore di luminosità, barometro, ...

Java

- Il linguaggio principale con cui vengono generati i programmi Android è il linguaggio Java.
- Il Java è un linguaggio che supporta la programmazione orientata agli oggetti.
- Nasce nel 1992 per opera di James Gosling e degli ingegneri della Sun Microsystems, ora registrato dalla Oracle
- Attualmente è alla versione 7 (2011)

Caratteristiche del linguaggio Java

- essere orientato agli oggetti;
- essere indipendente dalla piattaforma;
- contenere strumenti e librerie per il networking;
- essere progettato per eseguire codice da sorgenti remote in modo sicuro.

Programmazione Object Oriented

- **Classe** : modello astratto che crea oggetti software

  Al suo interno troviamo:

- **Attributi** : caratteristiche di una classe
- **Metodi:** comportamenti della classe , le sue procedure

Classe (1/4)

- Una classe definisce un **tipo**

  Esiste un **costruttore**

  Metodo speciale per allocare gli oggetti

  Non restituisce nessun tipo di ritorno (neanche void)
- Esiste un metodo **toString()** - serve a rappresentare l'oggetto in formatestuale
- E' possibile usare l'**overloading** (sovraccarico) del costruttore

  Esistono altri metodi importanti

  equals()

  clone()

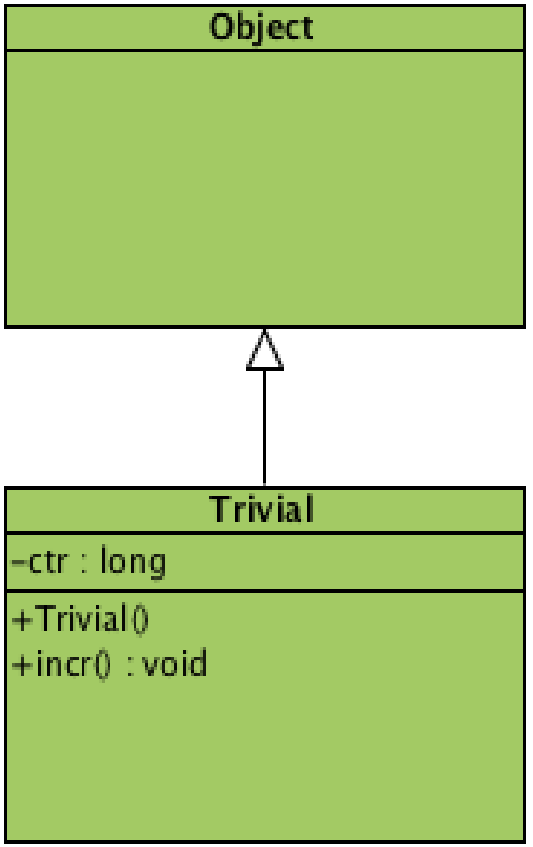
  finalize()

Classe (2/4)

```
public class Contatore {

        private int conta;

        public void incr() {
            conta++;
        }

}
```

Classe (3/4)

```
public class Contatore extends Object{

        private int conta;

        public Contatore(){
            super();
        }

        public void incr() {
             conta++;
        }

}
```



Classe(4/4)

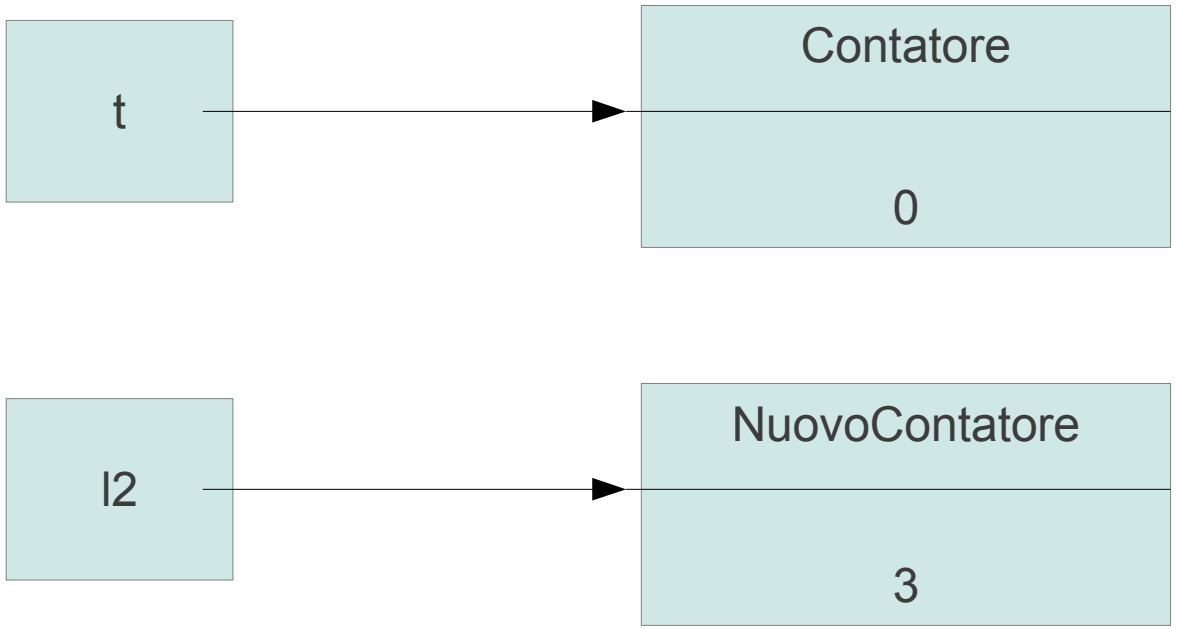
```
public class Contatore extends Object {
        private long conta;

        public LessTrivial(long conta){
            this.conta = conta;
        }

        public void incr() {
            conta++;
        }
}
```

Oggetti

- Un oggetto è un rappresentante della classe
- La creazione avviene mediante il richiamo di un **costruttore**
- I valori degli attributi sono tutti inizializzati a 0 se interi, a false se booleani ecc
- I valori degli oggetti attributo della classe sono inizializzati a **null**

Oggetti

- Contatore t = new Contatore(); // costruttore implicito
- NuovoContatore l = new NuovoContatore(); // errore!
- NuovoContatore l2 = new NuovoContatore(3); // giusto!



Riferimenti e Puntatori

- Nel linguaggio Java non esistono i puntatori, ma i **riferimenti** ovvero una locazione di memoria automatizzata

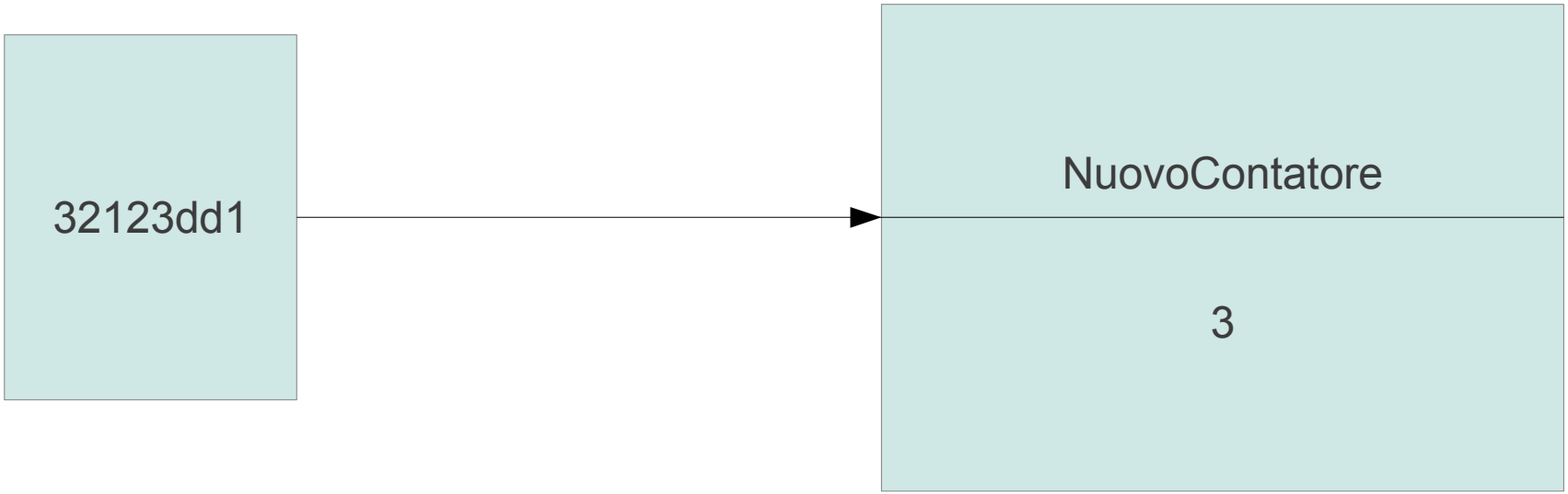
  Es: it.pinkopallino.Test@32123dd1

  it.pinkopallino.test (**package**)

  Test (**classe**)

  32123dd1 (**hashing dell'indirizzo di memoria**)

Riferimenti e Puntatori

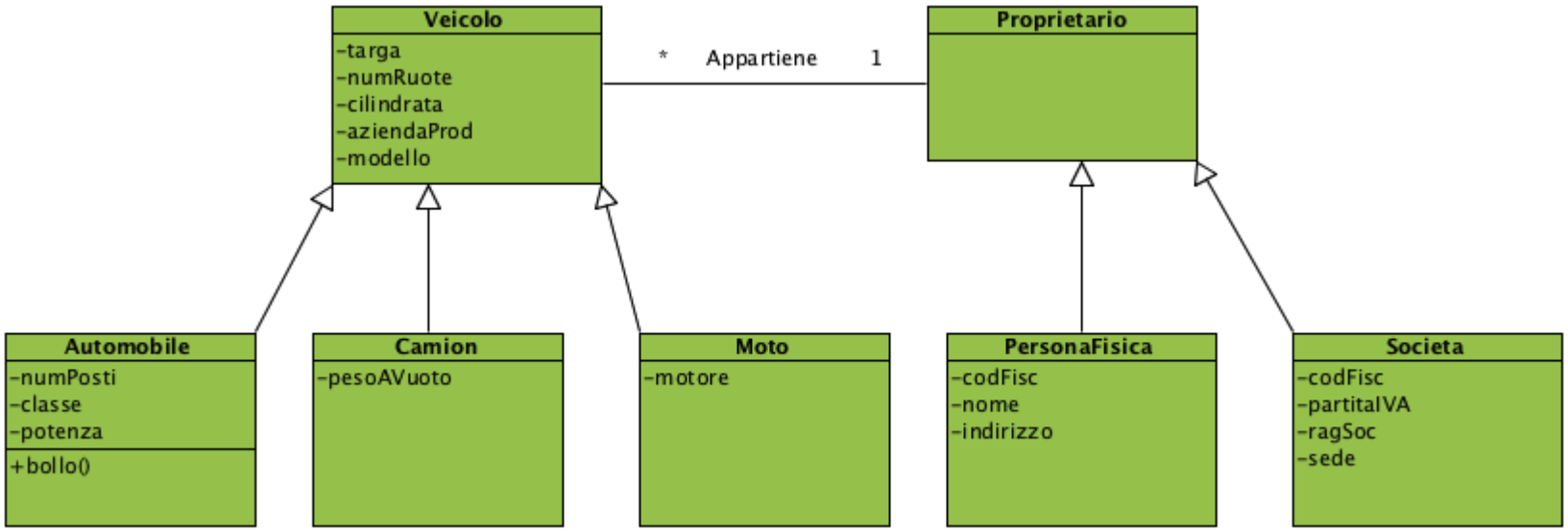


Ereditarietà (1/3)

- Concetto di astrazione che permette

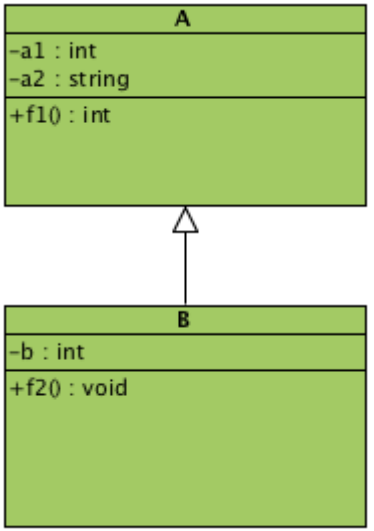
  -) riutilizzo del codice

  -) creazione delle gerarchie di classi



Ereditarietà(2/3)

- A è detta **superclasse** e B **classe derivata**
- B eredita gli attributi e i metodi della classe padre (solo attributi e metodi pubblici e protected)
- La relazione tra A e B è detta relazione **is-a**



Ereditarietà(3/3)

- public class B **extends** A {

  private int b; // oppure **protected**

  public void f2()

  {

  ...

  }

}

Override

- Ridefinisce una classe derivata di metodi già definiti nella **superclasse**
- public class B extends A{

```
    @Override
    public void f1() {
    // ridefinisco le funzionalità di f1
    }
}
```

Super

- Fa riferimento ai metodi e agli attributi definiti nella **superclasse**

  nel codice dei metodi di una classe derivata

- public class B extends A{

  @Override

  public void f1() {

  - **super**.f1();

  }

}

Polimorfismo

- **Un oggetto assume forme differenti**
- Viene adottato il **“binding dinamico**”

  A a = new A();

  // b è un anche un oggetto di tipo A

  A b = new B();

  // richiamo di un metodo definito nella classe padre A a “runtime”

  b.f1();

-

Final e Static

- **final :** qualcosa che rimane **immodificabile** in riferimento alle classi
- **static :** “qualcosa” che è invocabile indipendentemente dall'esistenza di oggetti istanza della classe

  es. Math.sqrt(n) , main

Interfacce(1/2)

- Definisce un tipo senza definire un'implementazione, usando costanti e metodi astratti
- Implementa l'eredità multipla

```
public interface Growable {
    void grow(Fertilizer food, Water water);
}
public interface Eatable {
    void munch();
}
```

Interfacce(2/2)

- public class Beans **implements** Growable, Eatable {

```
    @Override
    public void grow(Fertilizer food, Water water) {
    // ...
    }
    @Override
    public void munch() {
    // ...
    }
}
```

Generics (1/2)

- Costruisce una classe **Generica** partendo da una o più classi specificate evitando problemi di conversione **cast**

  Le classi Vector e ArrayList sono classi generiche,dove specifichiamo il tipo degli elementi

  - Vector<E>

  - ArrayList<E>

  dove E è il tipo parametrico.
- **Esempio:**
- **List<Studente> studenti = new ArrayList<Studente>();**
- **studenti.add(new Studente(“Rossi”,”Mario”,”5n”));**
-

Generics (2/2)

- I tipi primitivi non vengono utilizzati come parametri, al loro posto si devono usare leclassi involucro (wrapper).

- E' noto anche come auto-boxing/unboxing.

- Vector<Integer> v = new Vector<Integer>();

Programmare in Java con Android

- Ambiente di sviluppo Eclipse
- Sdk di Android (virtualizza il nostro sistema operativo)
- Java Virtual Machine
- Cellulare con Android (se vogliamo velocizzare)
- E' funzionante su tutti i sistemi operativi

Dalvik Virtual Machine

- Sebbene la programmazione sul sistema Android avvenga in Java, non viene utilizzata la J2ME per ragioni economiche e di performance. Al posto della V.M. per Java viene usata la Dalvik V.M.

  Tale **virtual machine** ha le seguenti caratteristiche:

  1) non accetta file .class ma un unico pacchetto .apk con un .dex che contiene tutte le classi. In questo modo di ottimizzano le dimensioni del file (no ripetizioni costanti, no ripetizione header etc.). Non è di contro possibile avere il caricamento dinamico delle classi, le classi caricabili sono solo quelle presenti nel .dex.

  2) il **bytecode** è orientato ai registri (e non allo stack come la Java V.M.) per una maggiore velocità di esecuzione e compattezza del codice (a scapito di una maggiore difficoltà di compilazione e più basso riuso);

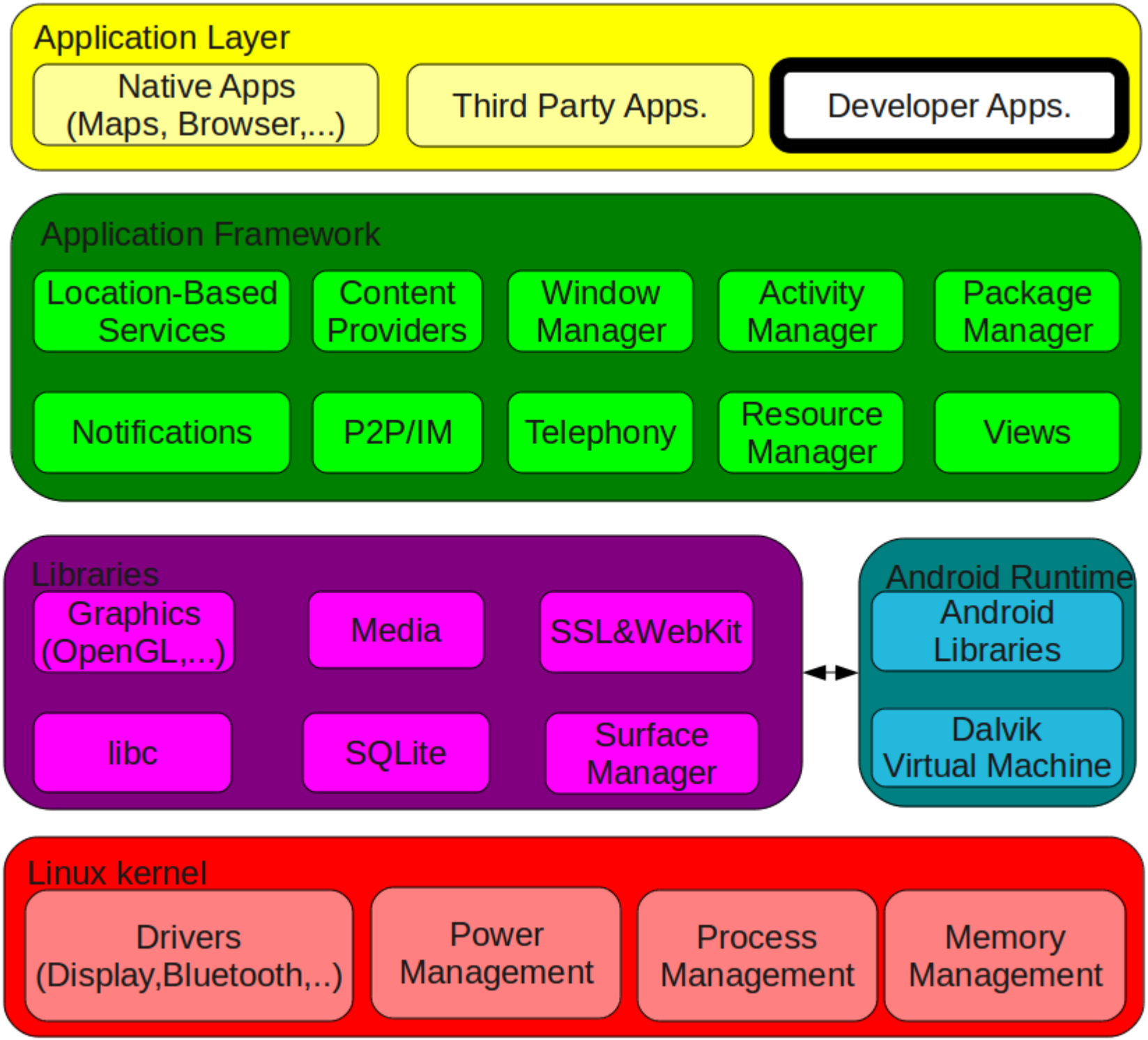
  Si ottiene così una riduzione del 50% della dimensione del bytecode prodotto e del 30% del numero di istruzioni da eseguire.

  Per aumentare le performance dalla versione 2.2 di Android è previsto anche la compilazione Just In Time a granularità di trace che ha queste caratteristiche rispetto a un JIT a granularità di metodo:

  1) utilizzo minimo di memoria aggiuntiva

  2) tempo veloce di avvio a scapito di una minore ottimizzazione (e possibilità di condivisione tra processi).

  Per aumentare la velocità di startup delle DVM su cui girano i vari processi, il caricamento della DVM stessa deve essere molto veloce. Per questo si è usato una tecnica detta **Zygote** che permette la condivisione e il precaricamento di tutte le librerie core,

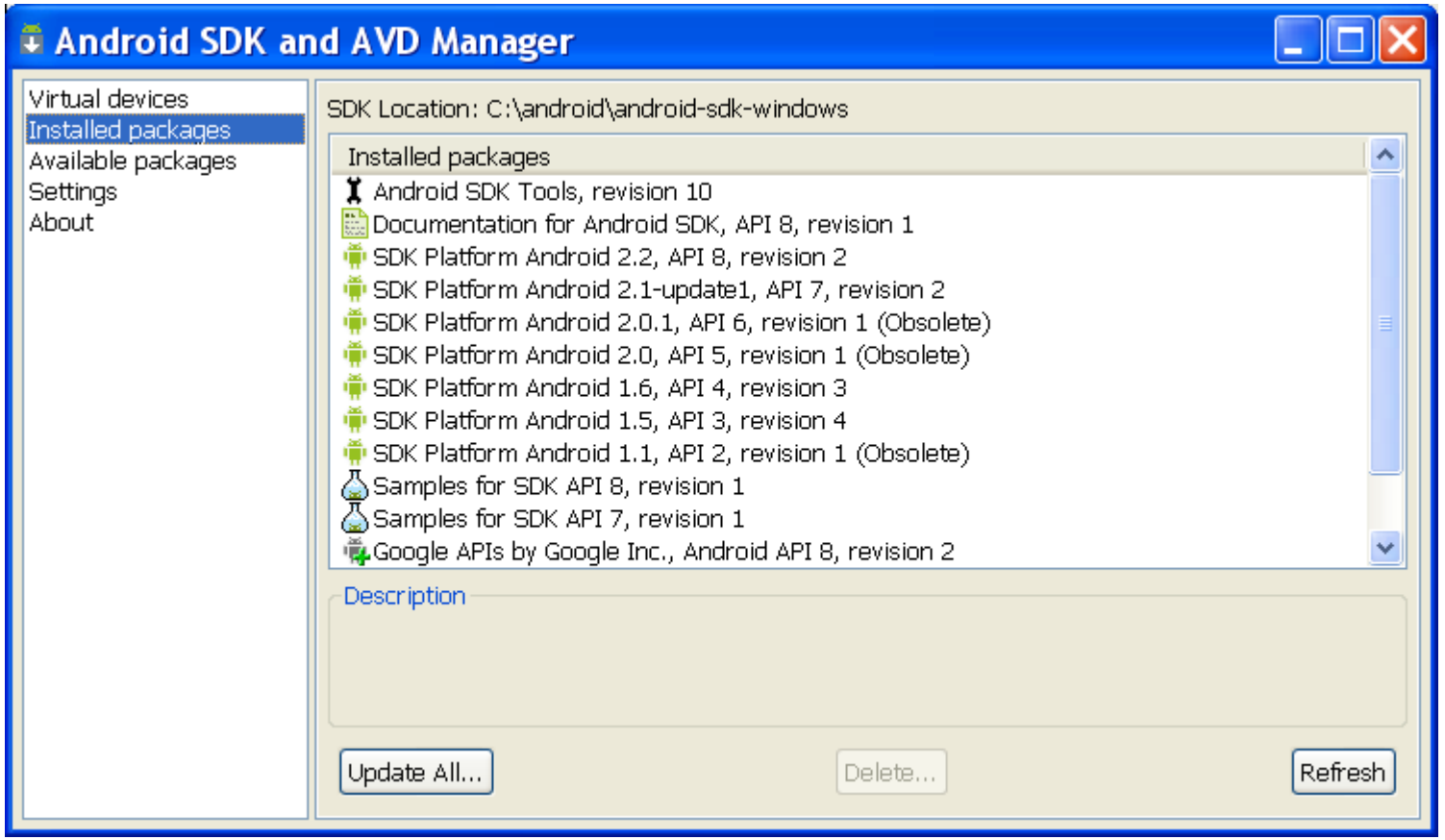

Application Layer
Native Apps (Maps, Browser,...)
Third Party Apps.
Developer Apps.
Application Framework
Location-Based Services
Content Providers
Window Manager
Activity Manager
Package Manager
Notifications
P2P/IM
Telephony
Resource Manager
Views
Libraries
Graphics (OpenGL,...)
Media
SSL&WebKit
libc
SQLite
Surface Manager
Android Runtime
Android Libraries
Dalvik Virtual Machine
Linux kernel
Drivers (Display,Bluetooth,..)
Power Management
Process Management
Memory Management

SDK

- Si procede innanzitutto all'installazione della SDK:
- La SDK per Android richiede l'installazione della SDK di Java. Dopo aver installato Java, procediamo con l'installazione per Android Opzionalmente ma vivamente consigliato e assunto nei seguito, si può procedere all'installazione del plugin ADT per Eclipse
- Successivamente si possono aggiungere altre piattaforme e componenti alla SDK di Android tramite l'Android SDK and AVD Manager. A esempio, possono essere installate varie versioni della SDK, pacchetti di documentazione o varie librerie.
- Maggiori dettagli sono disponibili a link: http://developer.android.com/sdk/installing.html

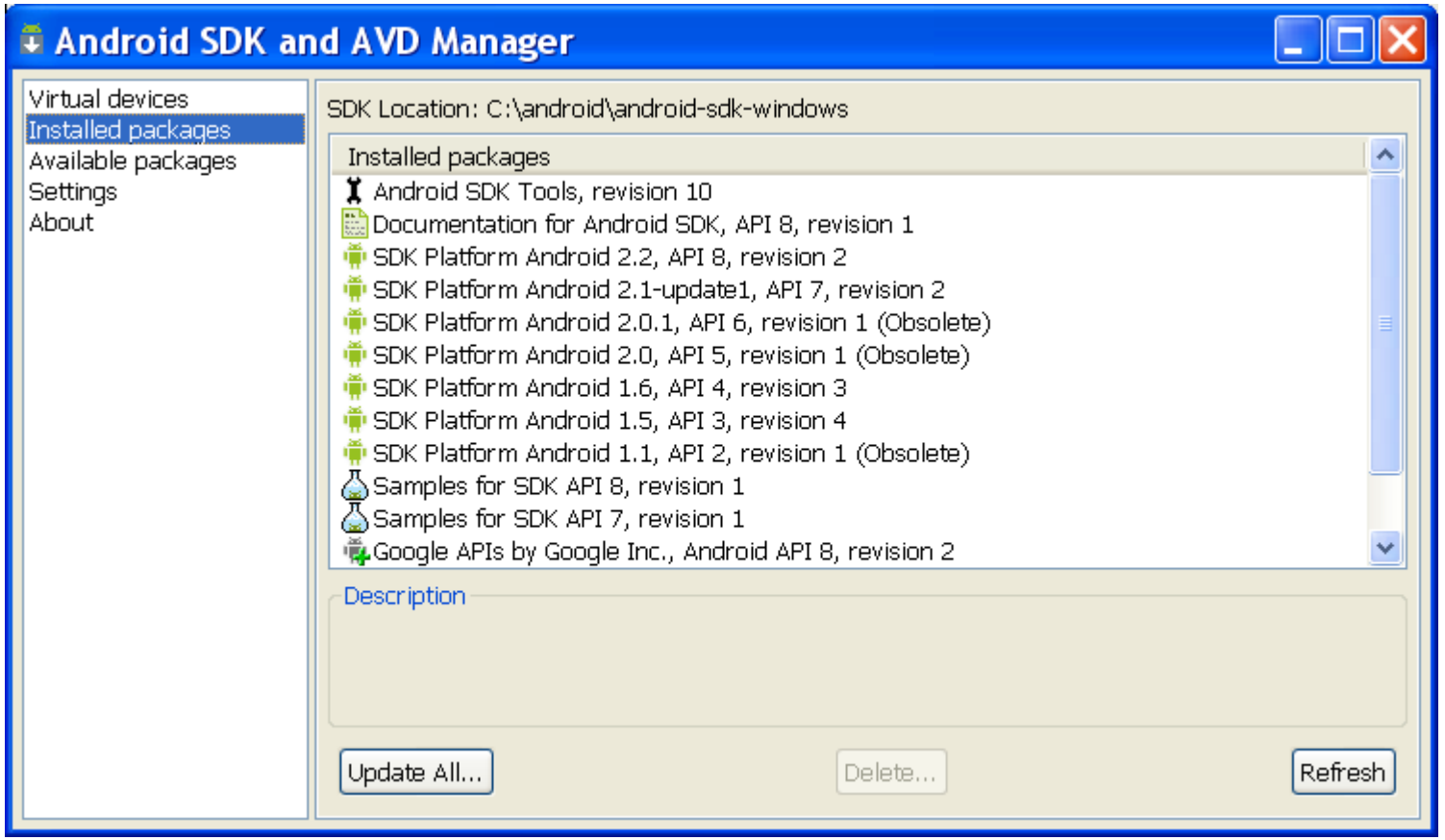
SDK

- Aspetto SDK Manager dopo l'aggiunta delle piattaforme (Window -> Android SDK and AVD Manager):



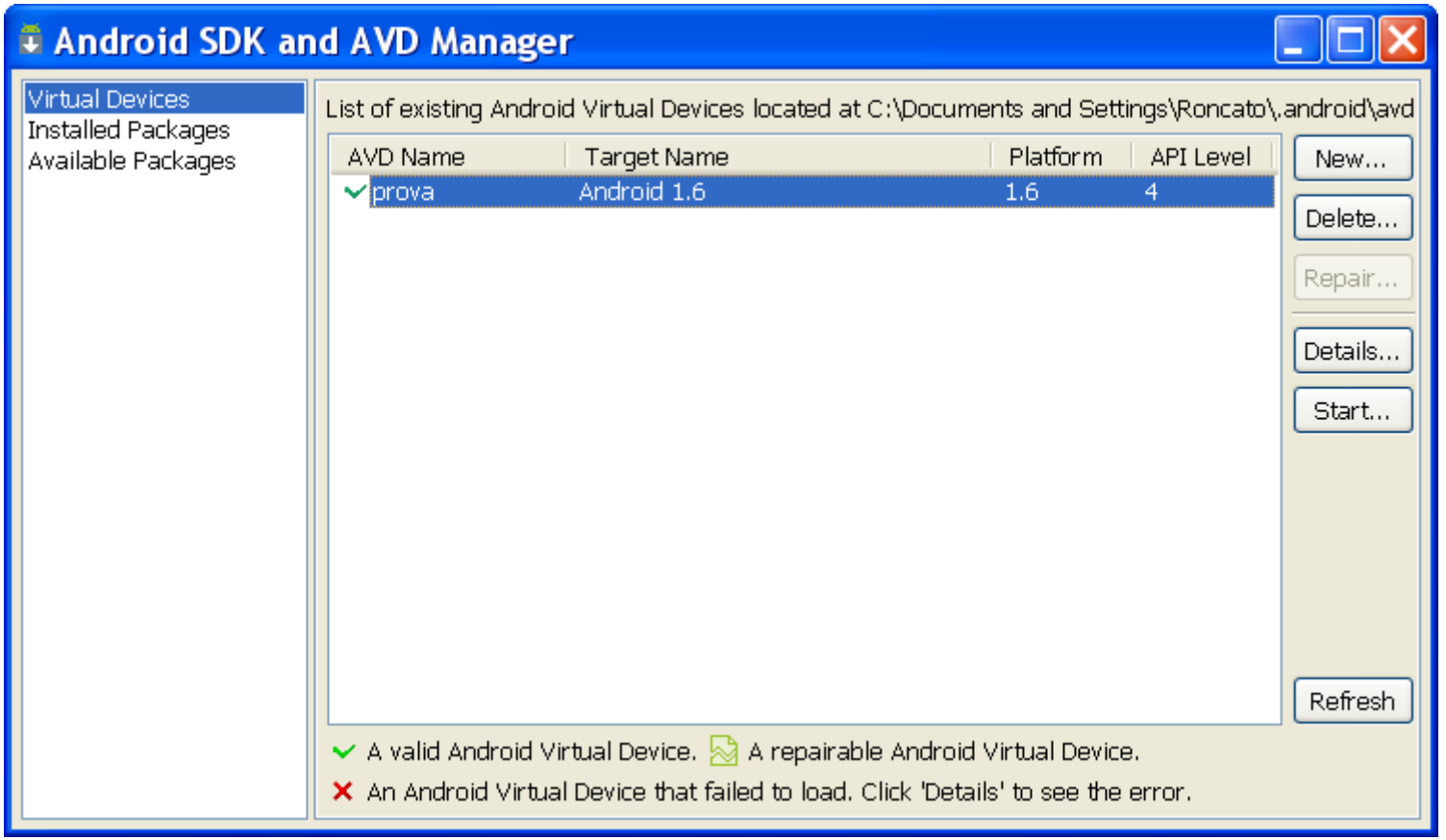
SDK

- Aspetto SDK Manager dopo l'aggiunta delle piattaforme (Window -> Android SDK and AVD Manager):



SDK

- La SDK contiene un emulatore mobile con il quale possiamo selezionare il tipo di versione dell'API supportata da esso più la presenza o meno dei dispositivi hardware (es GPS, accelerometri, quantità di memoria etc.)



SDK

- Simulatore in partenza

SDK

- Simulatore in esecuzione

Architettura

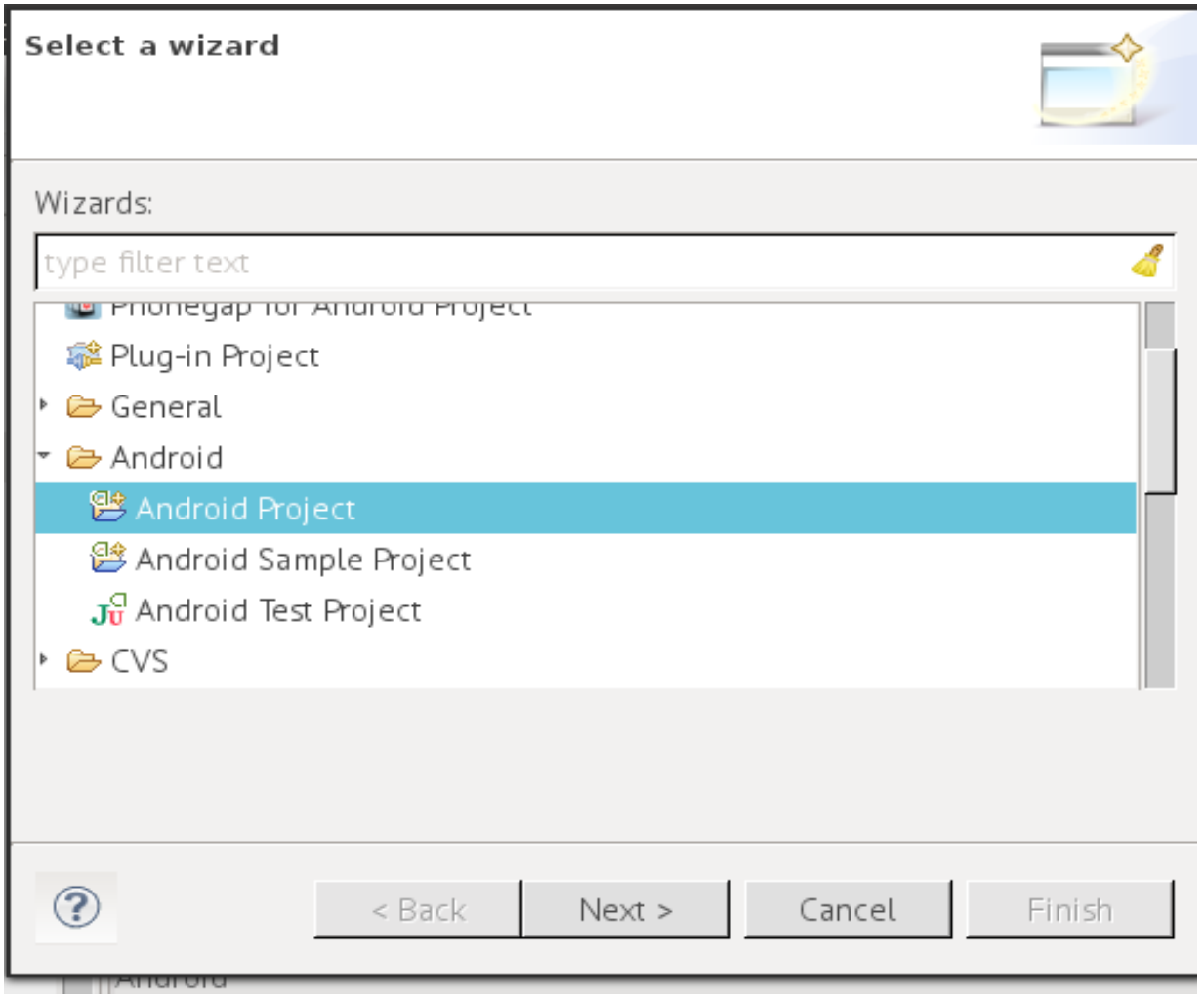
- L'architettura è orientata al riuso dei componenti ottenuto tramite file xml con la descrizione dell'aspetto dell'applicazione, file di risorse esterni al codice Java.

  I componenti di un'applicazione possono essere:

- **Activity**: schermate visibili , la singola cosa che l'utente puo fare
- **Services**: attività in background
- **Content Provider**: fornitori di contenuti condivisi
- **Broadcast Receiver**: ricevono e reagiscono a eventi in broadcast, sono consumatori di intent
- **Intent**: componenti che attivano altri componenti, utilizzano il meccanismo del passaggi dei messaggi
- **Notifications:** notificazioni agli utenti

Creazione di un progetto con Android

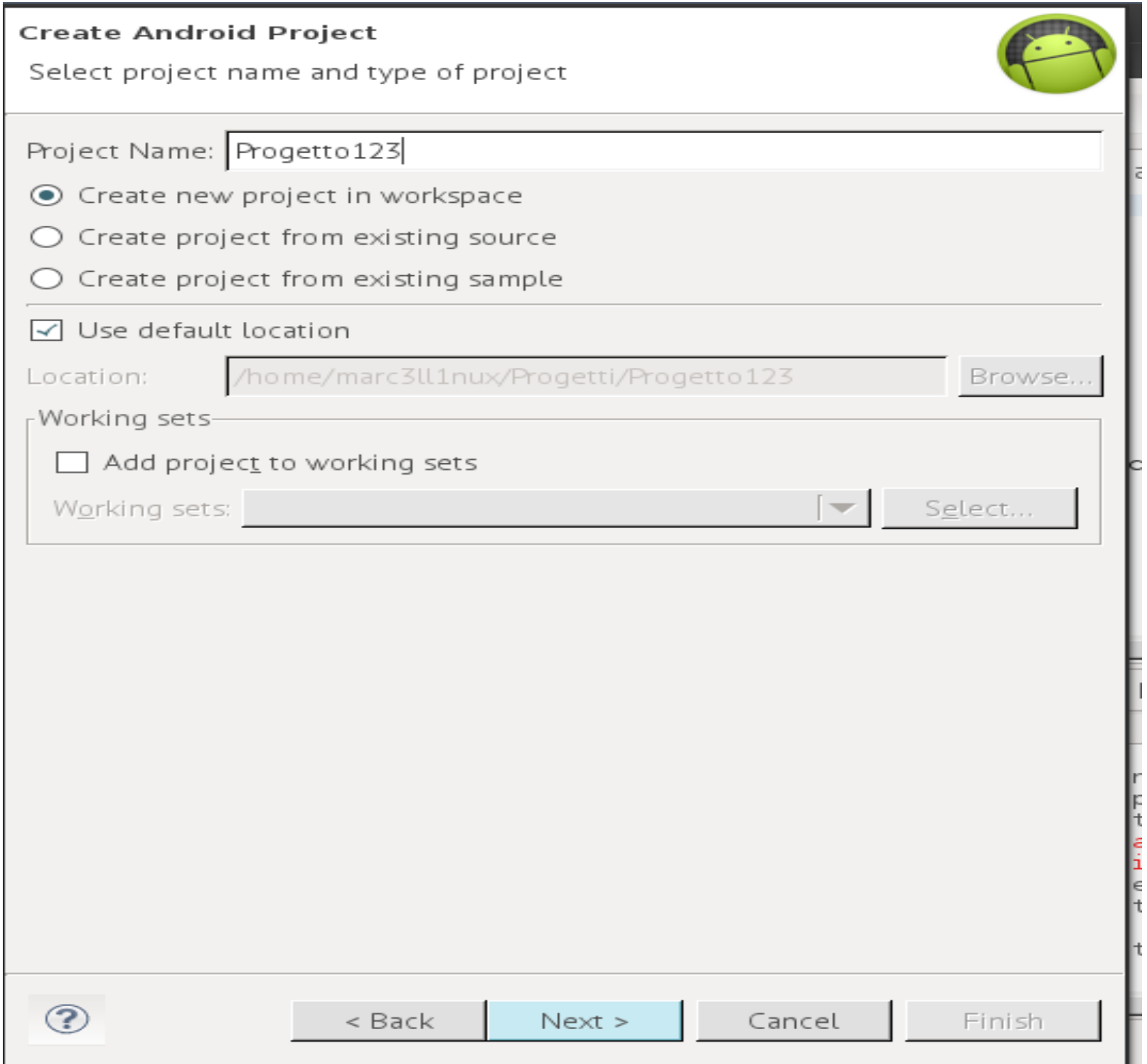
- **File -> New -> Project**

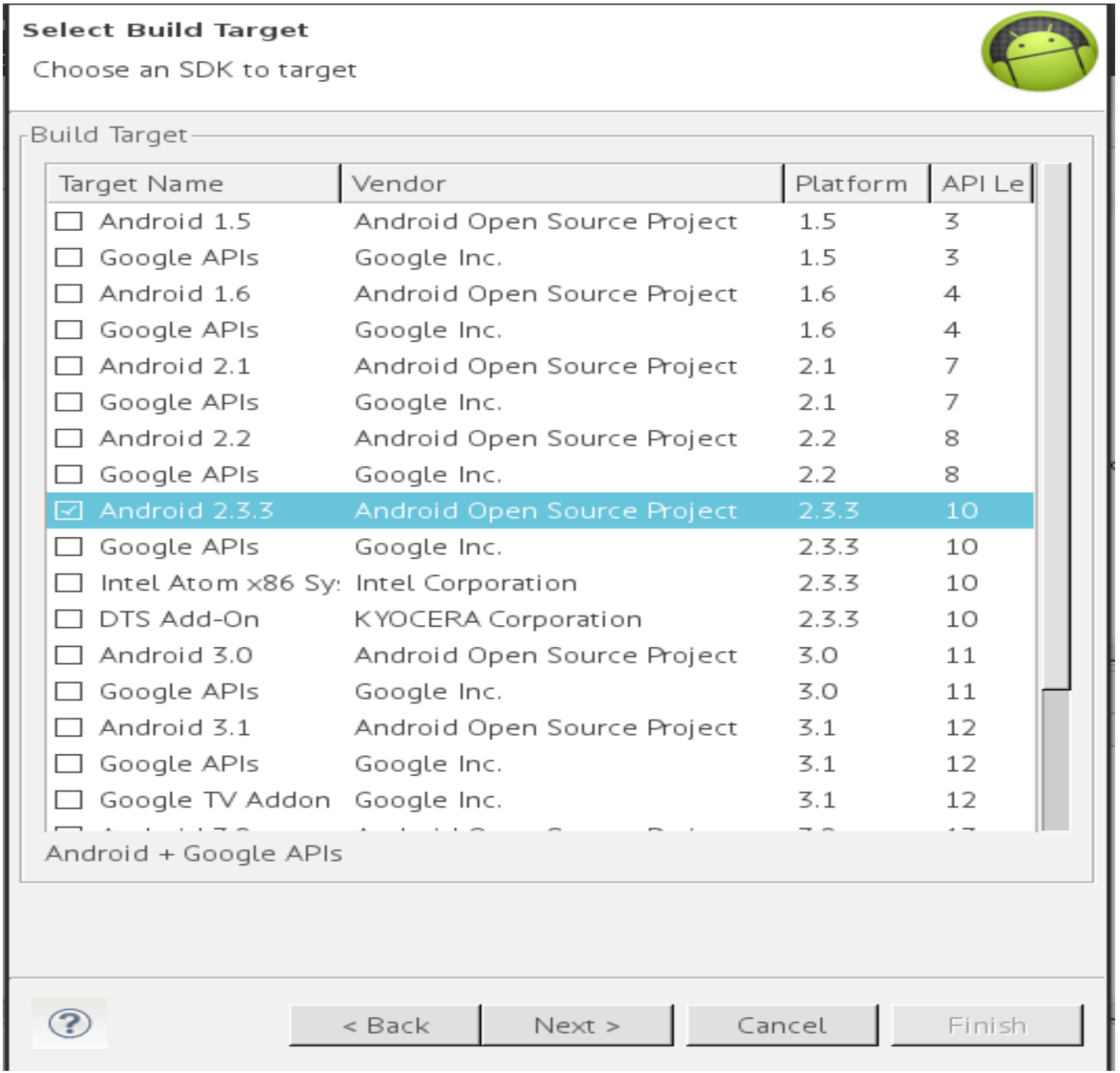


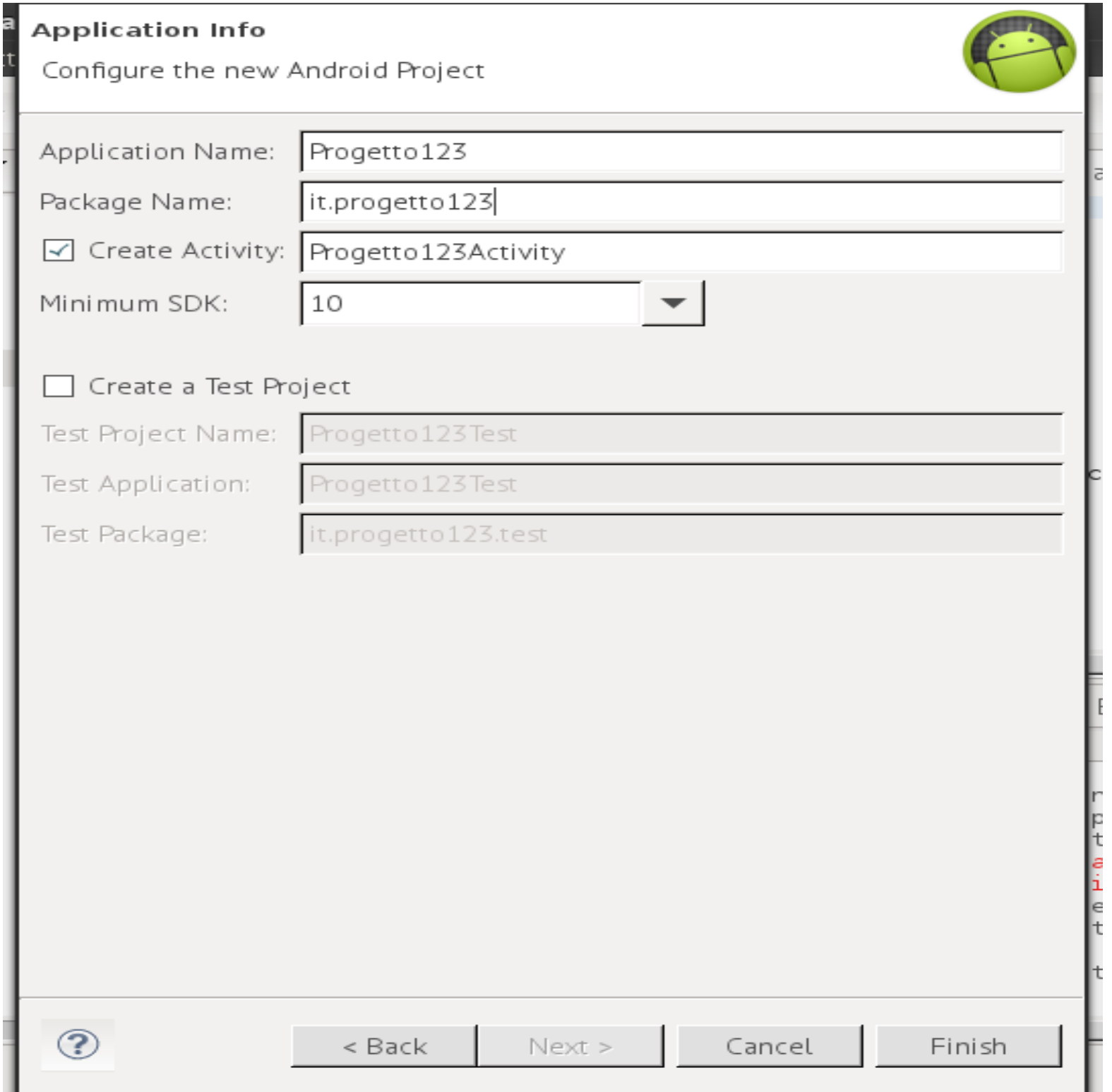
Creazione di un progetto con Android

Successivamente scrivere:

- 1) nome progetto
- 2) piattaforma su cui far girare l'applicazione
- 3) il nome dell'applicazione
- 4) il package
- 5) il nome dell'attività
- 6) la minima piattaforma su cui deve girare l'applicazione.


Create Android Project
Select project name and type of project
Project Name: Progetto123
Create new project in workspace
Create project from existing source
Create project from existing sample
Use default location
Location: /home/marc3ll1nux/Progetti/Progetto123
Browse...
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Cancel
Finish

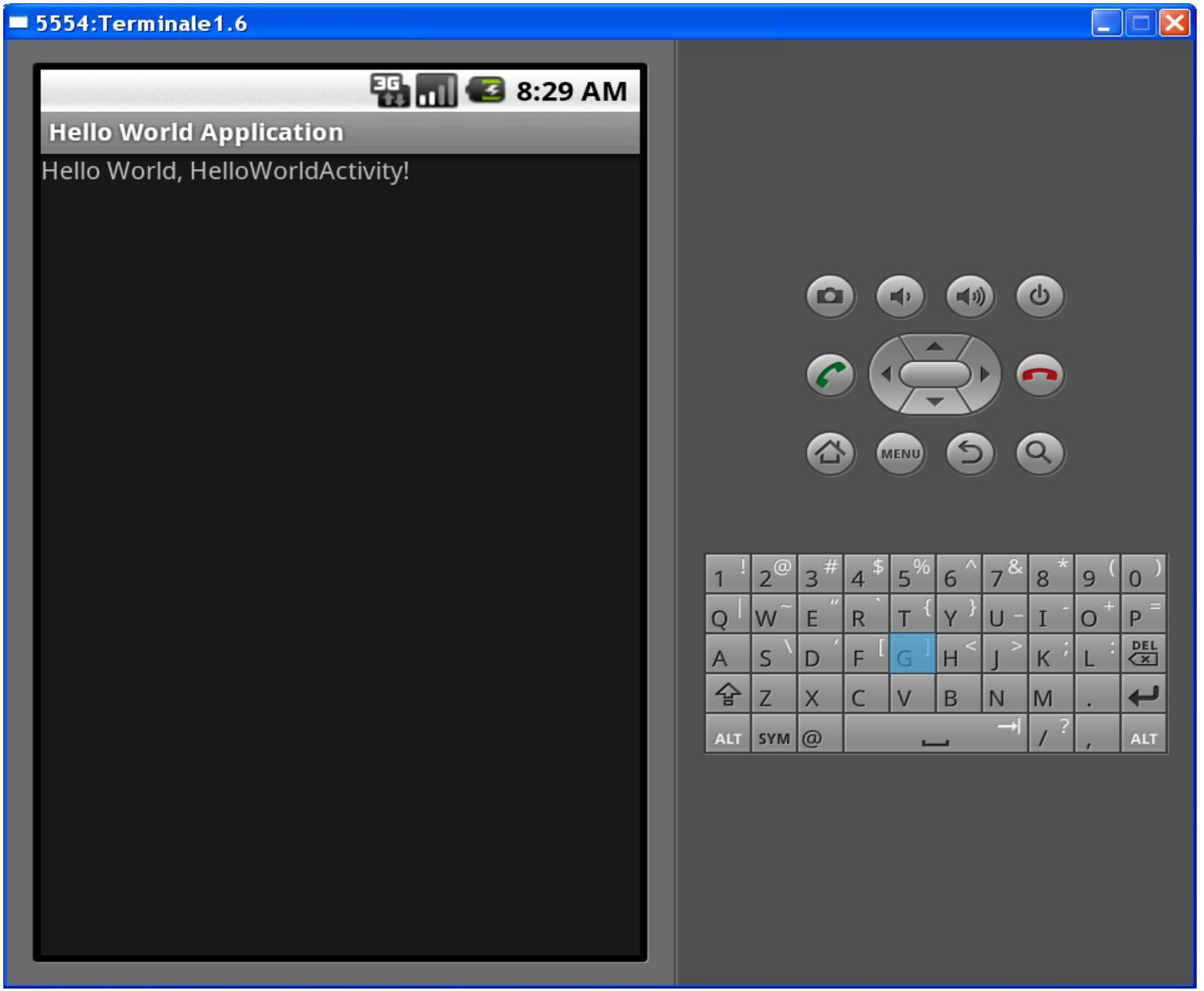

Select Build Target
Choose an SDK to target
Build Target
Target Name
Vendor
Platform
API Le
Android 1.5
Android Open Source Project
1.5
3
Google APIs
Google Inc.
1.5
3
Android 1.6
Android Open Source Project
1.6
4
Google APIs
Google Inc.
1.6
4
Android 2.1
Android Open Source Project
2.1
7
Google APIs
Google Inc.
2.1
7
Android 2.2
Android Open Source Project
2.2
8
Google APIs
Google Inc.
2.2
8
Android 2.3.3
Android Open Source Project
2.3.3
10
Google APIs
Google Inc.
2.3.3
10
Intel Atom x86 Sy:
Intel Corporation
2.3.3
10
DTS Add-On
KYOCERA Corporation
2.3.3
10
Android 3.0
Android Open Source Project
3.0
11
Google APIs
Google Inc.
3.0
11
Android 3.1
Android Open Source Project
3.1
12
Google APIs
Google Inc.
3.1
12
Google TV Addon
Google Inc.
3.1
12
Android + Google APIs
< Back
Next >
Cancel
Finish


Application Info
Configure the new Android Project
Application Name:
Progetto123
Package Name:
it.progetto123
Create Activity:
Progetto123Activity
Minimum SDK:
10
Create a Test Project
Test Project Name:
Progetto123Test
Test Application:
Progetto123Test
Test Package:
it.progetto123.test
< Back
Next >
Cancel
Finish

Creazione di un progetto con Android

- Dopo aver generato in automatico il progetto verrà fuori questo che segue

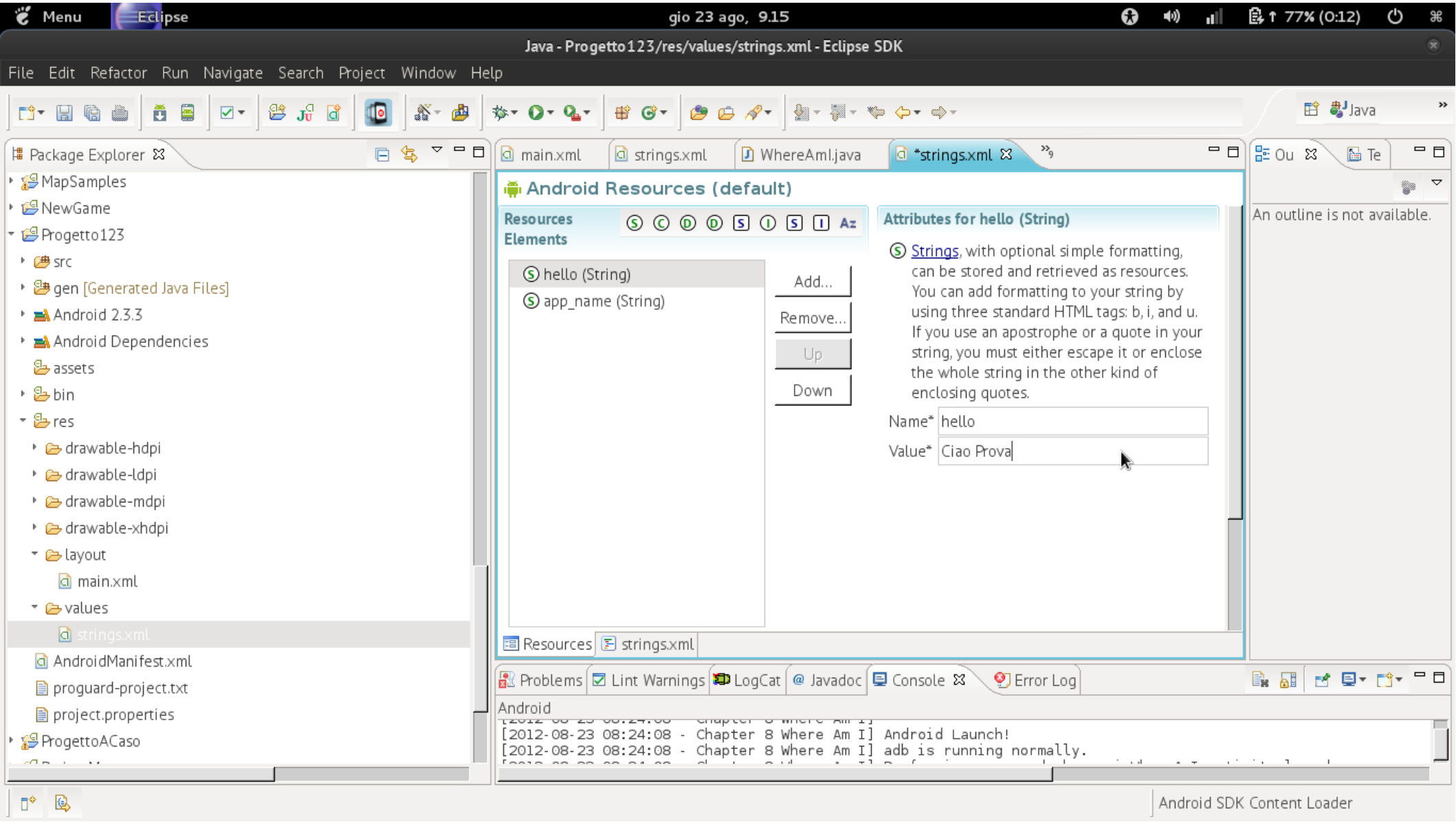
Creazione di un progetto con Android



Creazione di un progetto con Android

- Senza modificare il codice Java ma solo le risorse del progetto che contengono le stringhe possiamo modificare il progetto:

Creazione di un progetto con Android



Creazione di un progetto con Android

- Che corrisponde a strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello">Hello World, HelloWorldActivity!</string>
<string name="app_name">Hello World Application</string>
</resources>
```

Creazione di un progetto con Android

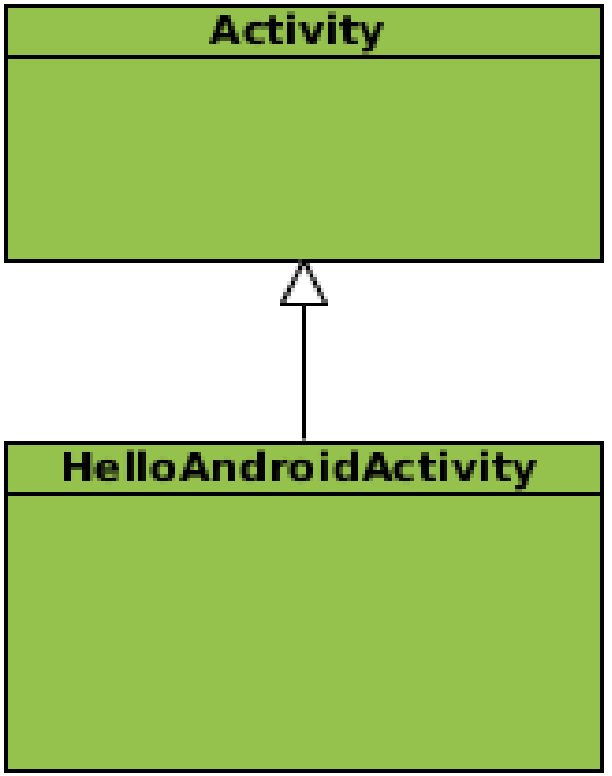
- Risultato

Creazione di un progetto con Android

```
package it.progetto123;
import android.app.Activity;
import android.os.Bundle;
//Bundle: classe che mappa i valori tramite utilizzo di chiavi univoche
public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //permette la visualizzazione di un layout xml all'interno del nostro programma sulla cartella res/layout
    }
}
```

Creazione di un progetto con Android

- Funzionamento di un'Activity



Elementi di un applicazione Android

- Classe **HelloWorldActivity**.java
- File gestione stringhe **strings.xml**
- File gestione layout **main.xml**
- File **AndroidManifest.xml**
- Classe **R.java** generata automaticamente

Elementi di un applicazione Android

- File gestione layout **main.xml**

  <?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"

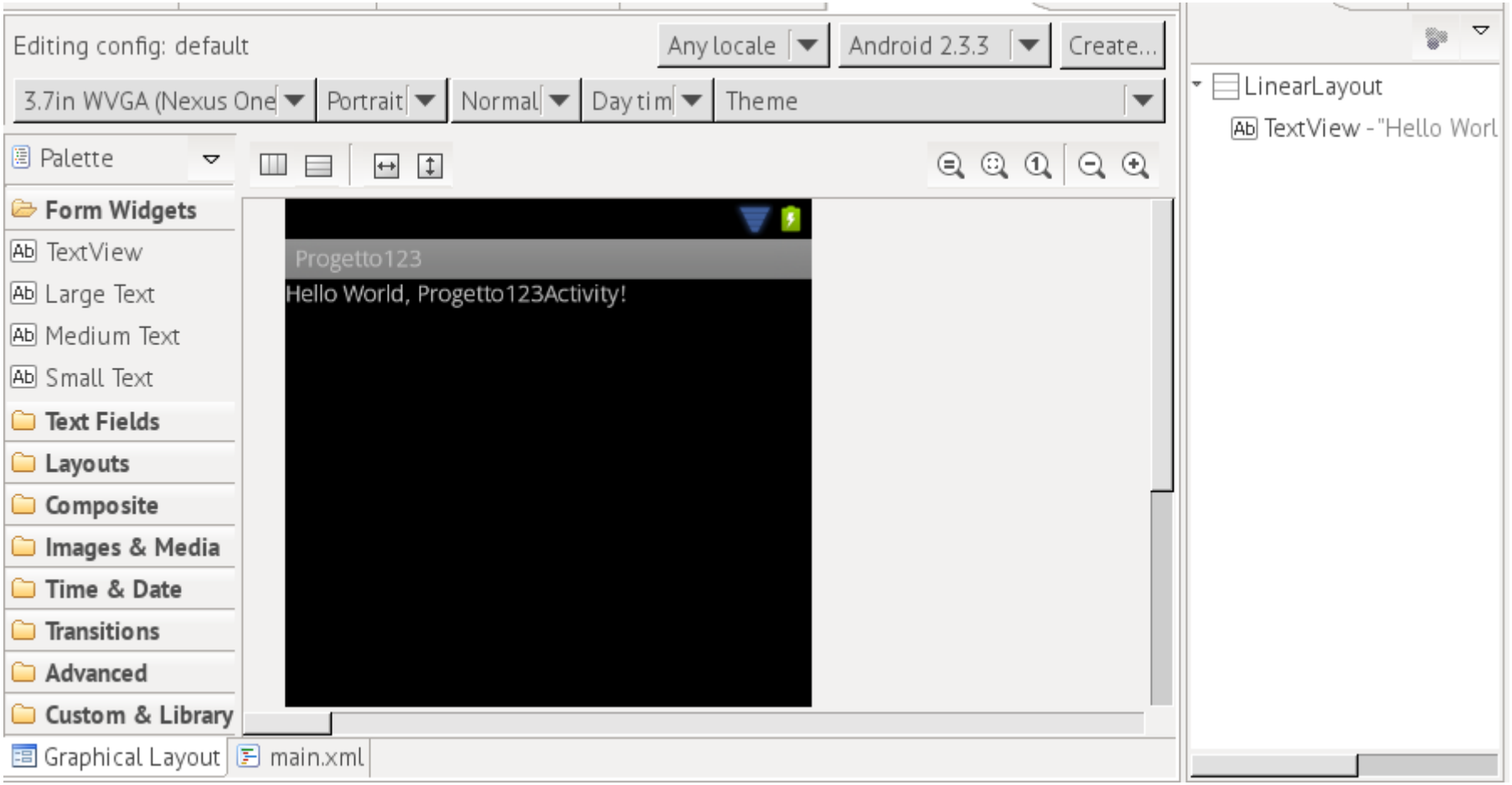
  android:text="@string/**hello**"/>

  </LinearLayout>

-

Elementi di un applicazione Android

- Visualizzazione grafica di un layout xml



Elementi di un applicazione Android

- **AndroidManifest.xml** -> definisce la struttura e i metadati dell'applicazione.
- Include un nodo per ogni componente (Attività, Servizi, Content Providers e Broadcast Receiver) e attraverso gli Intent Filter e i Permission determina come ogni componente interagisce con gli altri e le altre applicazioni.
- La root del file xml è il tag “manifest”:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="it.progetto123"
android:versionCode="1"
android:versionName="1.0">
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
//permessi di accesso a servizi
<uses-permission
android:name="android.permission.SIGNAL_PERSISTENT_PROCESSES"></uses-permission>
<application android:icon="@drawable/icon" android:label="@string/app_name"> // specifica i
metadati dell'applicazione
<activity android:name=".Prova" android:label="@string/app_name">
<intent-filter> // componenti che gestiscono i vari intent
<action android:name="android.intent.action.MAIN" /> //applicazione di partenza
<category android:name="android.intent.category.LAUNCHER" /> //applicazione di partenza
</intent-filter>
</activity>
</application>
</manifest>
```

R.java

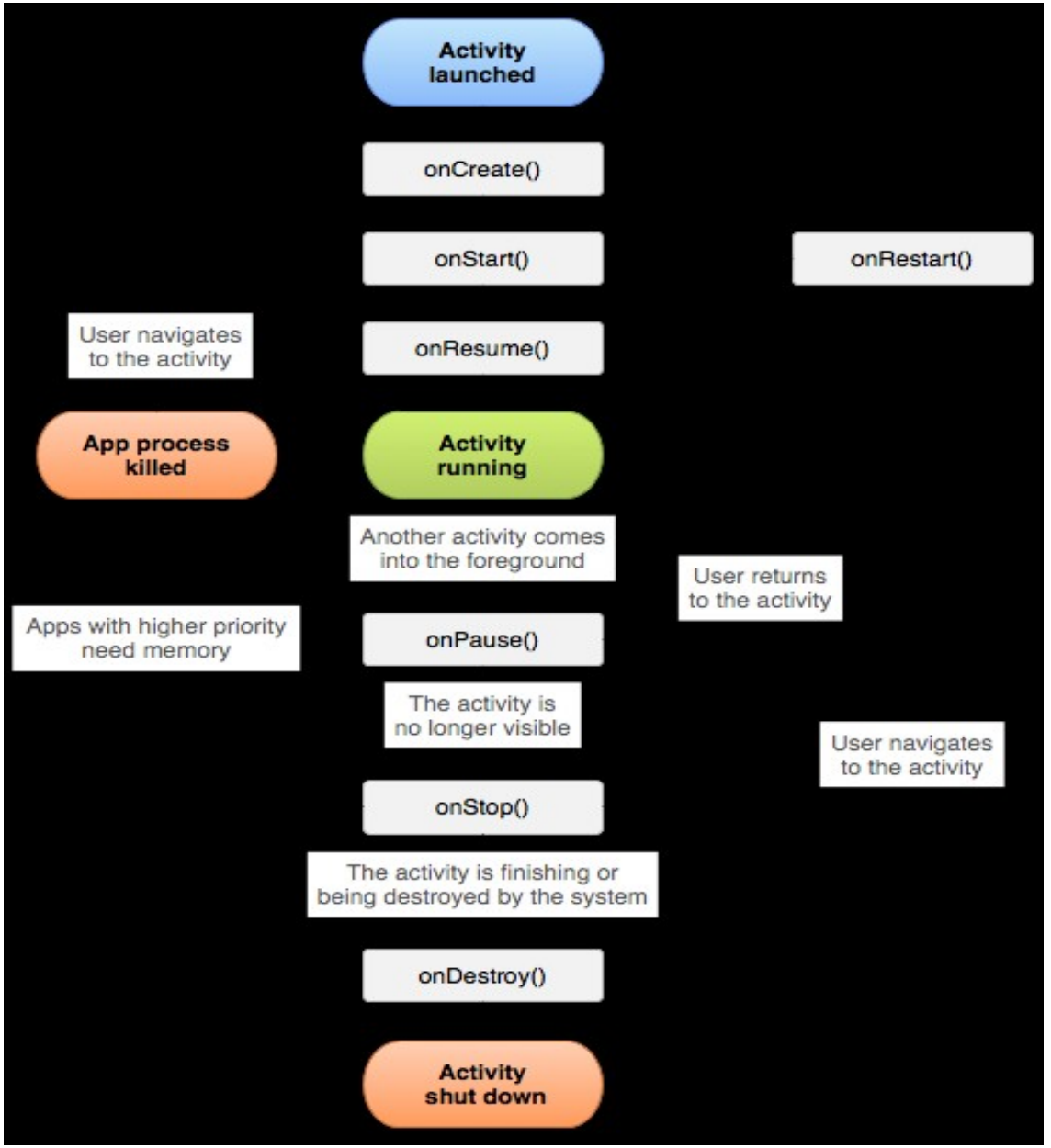
Genera gli oggetti grafici su Android - Da non modificare

```
package it.progetto.123;
public final class R {
public static final class attr {
}
public static final class drawable {
public static final int icon=0x7f020000;
}
public static final class layout {
public static final int main=0x7f030000;
}
public static final class string {
public static final int app_name=0x7f040001;
public static final int hello=0x7f040000;
}
}
```

Ciclo di vita software Android

- La superclasse **Activity** definisce eventi che governano il ciclo di vita di un'activity. Spetta poi alle classi che estendono da Activity implementarli. Questi eventi sono:
- **onCreate()** — chiamato quando l'activity viene lanciata per la prima volta
- **onStart()** — chiamato quando l'activity diventa visibile all'utente
- **onResume()** — chiamato quando l'activity inizia ad interagire con l'utente
- **onPause()** — chiamato quando l'attuale activity viene messa in pausa e un'activity precedente viene ripristinata
- **onStop()** — chiamato quando l'activity non è più visibile all'utente
- **onDestroy()** — chiamato prima che l'activity venga distrutta (manualmente o dal sistema operativo per liberare memoria)
- **onRestart()** — chiamato dopo che l'activity era stata stoppata e quando è pronta ad essere ripristinata

Ciclo di vita software Android



Intent

- L'**intent** è una descrizione astratta di un'operazione da eseguire.
- Il metodo **startActivity()** è utilizzato per far partire una nuova attività, che verranno messi a nella stack superiore delle attività.

Intent

```
public class MainActivity extends Activity {
...
// da qualche parte nell'attività
Intent intent1 = new Intent(this, SubActivity1.class);
startActivity(intent1);
Intent intent2 = new Intent(this, SubActivity2.class);
startActivity(intent2);
...
}
```

Intent

```
public class SubActivity1 extends Activity {
// fai qualcosa con SubActivity1
}
public class SubActivity2 extends Activity {
// da qualche parte nella SubActivity2
Intent intent3 = new Intent(this,SubActivity3.class);
startActivity(intent3);
}
public class SubActivity3 extends Activity {
// fai qualcosa con SubActivity3
}
```

Intent

```
<manifest ... >
<application ... >
<activity android:name=".MainActivity"
android:icon="@drawable/icon.png"
android:label="@string/app_name"
<intent-filter>
<action android:name="...MAIN" />
<category android:name="...LAUNCHER" />
</intent-filter>
</activity>
<activity android:name=".SubActivity1".../>
<activity android:name=".SubActivity2".../>
<activity android:name=".SubActivity3".../>
</application>
```

Toast

Un toast è una messaggio a pop up che compare sulla superficie della finestra.

Toast

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Toast

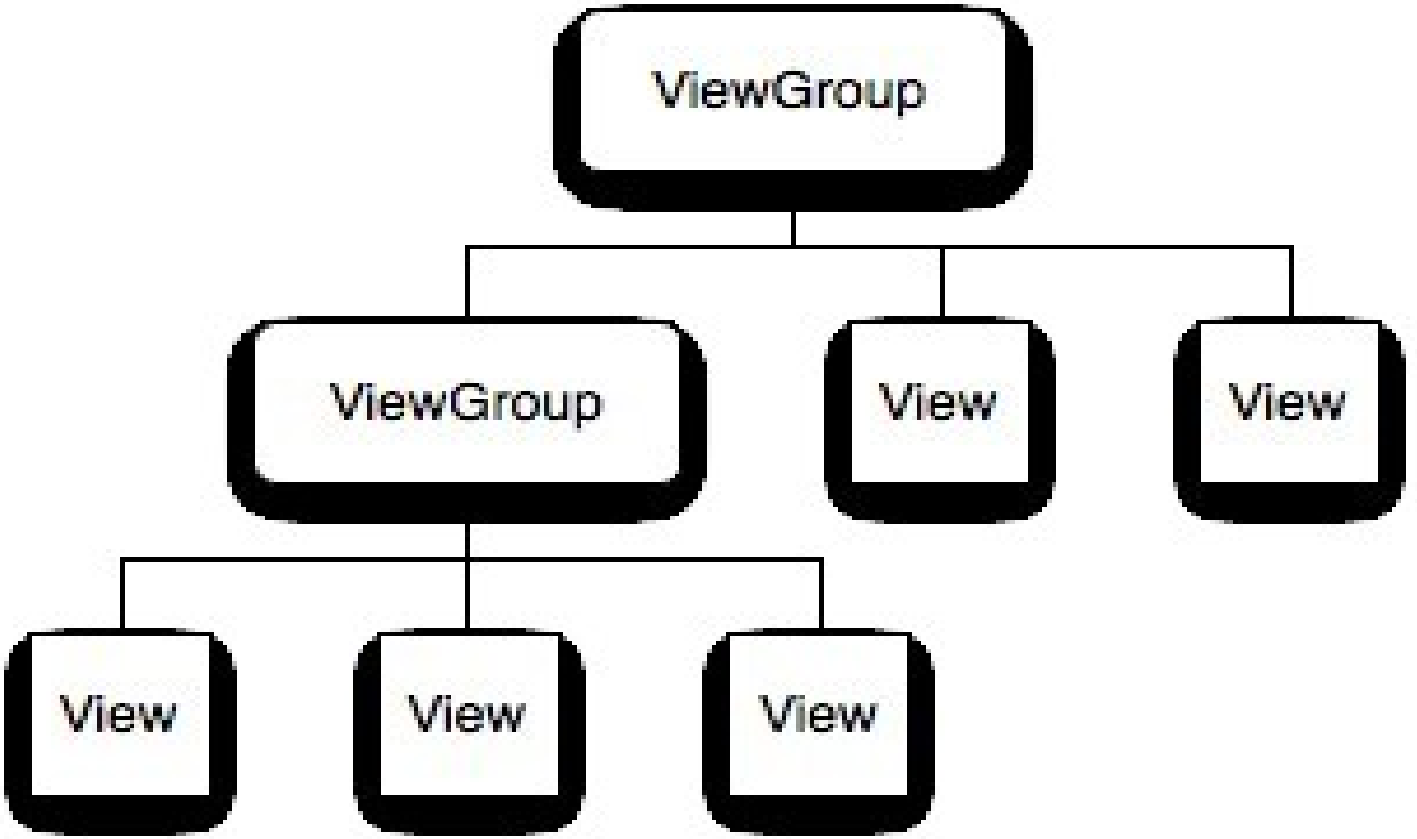
```
public class MiaActivity extends Activity {
...
private void toast(String text) {
    Context context = getApplicationContext();
    int duration = Toast.LENGTH_SHORT;
    Toast toast = Toast.makeText(context, text,
    duration);
    toast.show();
}

}
```

•

View vs View Group

- La classe **View** è la base per le sottoclassi chiamati "widgets" che offrono completamente widget e oggetti dell'interfaccia utente implementati, come campi di testo e pulsanti.
- La classe **ViewGroup** serve come base per le sottoclassi chiamate "layout" che offrono tipi diversi layout di architettura, come il layout lineare, tabellare e relativo.

View vs View Group



Events

- Quando consideriamo gli eventi all'interno del vostra UI, l'approccio è quello di catturare gli eventi dall'oggetto View specifico che l'utente interagisce con lui.

Events

- Classi interne.
- Classi anonime.
- Origine e ascoltatore coincidenti.

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    <TextView ... />
    ...
    <Button android:text="@string/mioTesto"
    android:id="@+id/mioBottone">
    </Button>
    ...
</LinearLayout>
```

Classi interne

```
protected void onCreate(Bundle savedValues) {
...
Button mioBottone = (Button)findViewById(R.id.mioBottone);
MioBottoneListener mioBottoneListener = new MioBottoneListener();
mioBottone.setOnClickListener(mioBottoneListener);
} // fine metodo onCreate
// continua...
```

Classi interne

```
private class MioBottoneListener implements
android.view.View.OnClickListener {
        @Override
    public void onClick(View v) {
    // fa qualcosa
    }
    Creazione di una classe interna
} // fine classe interna
```

Classi anonime

```
protected void onCreate(Bundle savedValues) {
...
Button mioBottone = (Button)findViewById(R.id.mioBottone);
mioBottone.setOnClickListener(
new OnClickListener() {
@Override
public void onClick(View v) {
// fai qualcosa
}
});
Creazione di un ascoltatore
...
} // fine metodo onCreate
```

-

Origine e ascoltatore coincidenti

```
public class MiaActivity extends Activity implements
OnClickListener {
    protected void onCreate(Bundle savedValues) {
    ...
    Button mioBottone =(Button)findViewById(R.id.mioBottone);
        mioBottone.setOnClickListener(this);
    }
    }
    // Implementa OnClickListener callback
    public void onClick(View v) {
    // fai qualcosa
}
...
```

-

Pubblicare su Play Store

1) Esportare l'applicazione

2) Registrarsi su Android Market/Google Play (costo 25 $)

Bibliografia

1) http://developers.android.com

2) http://www.thenewboston.com

3) http://www.mauriziocozzetto.it -> Android (Cozzetto,Sarasini)

4) http://it.wikipedia.org

5) http://en.wikipedia.org

6) http://www.dsi.unive.it/~taw -> Tecnologie e Applicazioni per il Web (Roncato)

7) http://www.androidgeek.it

8) http://www.coreservlets.com/android-tutorial/

Info

Marcello Cannarsa
Dott. In Informatica
Indirizzo: Via Padova 38 , 86039 Termoli(CB) – Molise – Italia
Sito web: http://www.marcellinux.it
Facebook & Twitter : marc3ll1nux

E-mail : info@marcellinux.it - marcellinux86@gmail.com