

ANDROID DEVELOPMENT

Interesting Web Sites

<http://developer.android.com>

<http://source.android.com>

<http://www.frandroid.com>

www.openhandsetalliance.com

Preparation

Discover tools

What will you use?



Discover tools

Preview

- Android SDK
- Eclipse
- ADT Plug-in for Eclipse





Discover tools

Android SDK

- Android Software Development Kit
- Available on Windows, Mac OS X and Linux
- Used in order to develop Android Application
- Contains the emulator, APIs, tools and more
- Provided by Google





Discover tools

Eclipse

- Multi-language software development environment
 - With an extensible plug-in system
- Written in Java
- Free and Open Source
- Platform independent
- Supported by the Eclipse Foundation

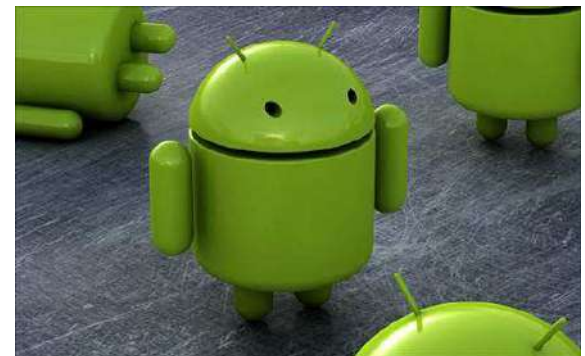




Discover tools

ADT plug-in for Eclipse

- Android Development Tools
- Plug-in for Eclipse IDE
- Give a powerful, integrated environment to build Android applications
- Extends the capabilities of Eclipse
 - Android Project support
 - Debug yours applications
 - Export them as APKs



Preparation

Install tools

Download and install



Install tools

Preview

- Android SDK installation
- Eclipse installation
- ADT plug-in installation
- Link ADT plug-in with Android SDK
- SDK Platform Components installation





Install tools

Android SDK installation

- <http://developer.android.com/sdk/index.html>
-
-
-
-



Install tools

Eclipse installation

■ Download Eclipse <http://www.eclipse.org/downloads/>



Eclipse Downloads

Packages Projects

Compare Packages

Older Versions

Eclipse Hellos (3.6.1) Packages



Eclipse IDE for Java Developers, 98 MB

Downloaded 701,573 Times

[Details](#)



Eclipse Classic 3.6.1, 169 MB

Downloaded 520,504 Times

[Details](#)

[Other Downloads](#)



Eclipse IDE for Java EE Developers, 204 MB

Downloaded 449,253 Times

[Details](#)



Eclipse IDE for C/C++ Developers, 87 MB

Downloaded 300,157 Times

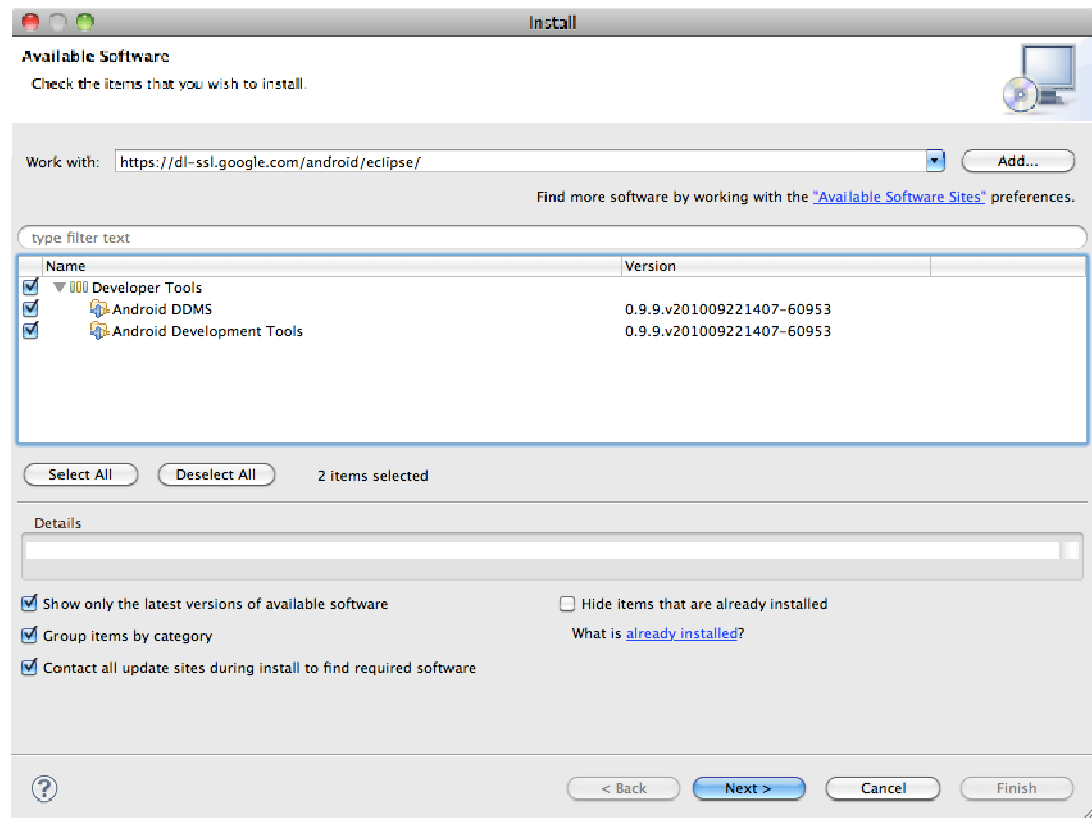
[Details](#)



Install tools

ADT plug-in installation

- Launch Eclipse
- Go to Help -> Install New Software...
- Enter <https://dl-ssl.google.com/android/eclipse/> in the “Work with” field





Install tools

ADT plug-in installation

- Click Next until you can click Finish
- Eclipse's going to download and install the plug-in
- When finished, restart Eclipse
- Now, you can see some new Buttons in your toolbar

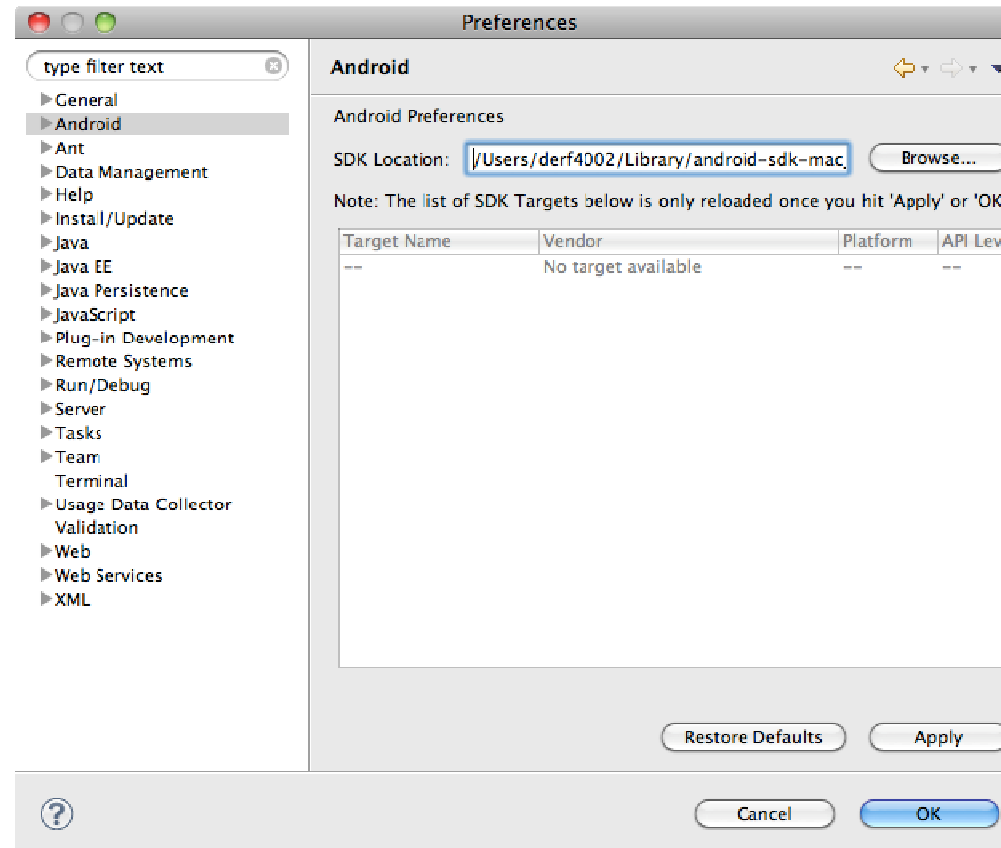




Install tools

Link ADT plug-in with Android SDK

- Go to Eclipse Preferences
- Click in the left panel on Android
- Complete the field “**SDK Location**”





Install tools

SDK Platform Components installation

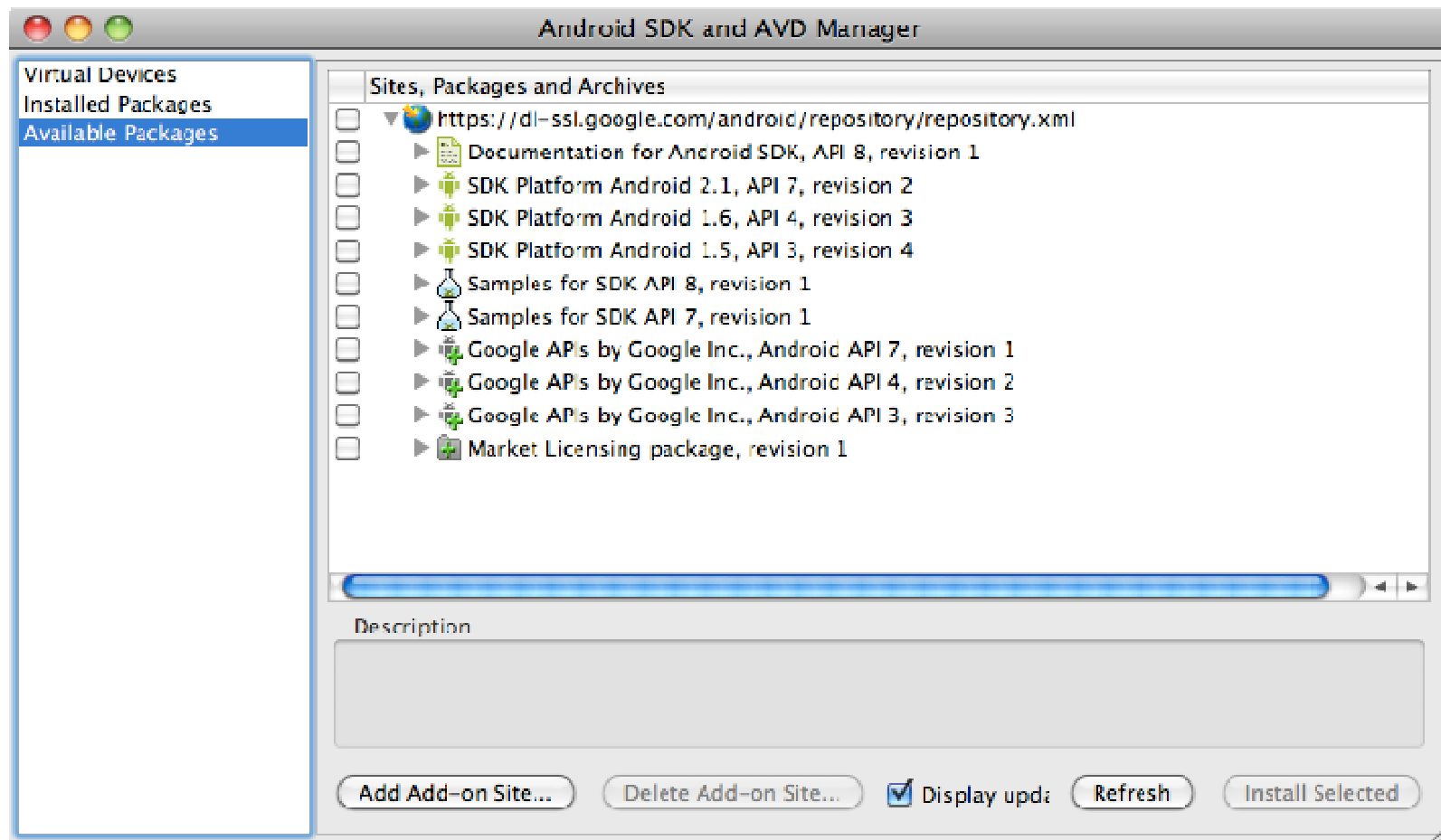
- Go to Window -> Android SDK and AVD Manager
- Select Available Packages on the left hand side
- Expand the Google Android site in the "Site, Packages, and Archives" Tree





Install tools

SDK Platform Components installation





Install tools

SDK Platform Components installation

- Check the following items:
 - SDK Platform Android 2.2
 - Google APIs by Google Inc.
- Click Install Selected
- We're now ready to develop Android applications !



Preparation



Test your installations



Test

Create a Project

- Create a new Android Project named HelloDroid with :
 - Android 2.2 selected in Build Target list
 - Application name : “**Hello Droid !**”
 - Package name : “**com.efrei.hellodroid**”
 - Activity name : “**Main**”
- Look at the project contents





Test

Execute it in an Emulator

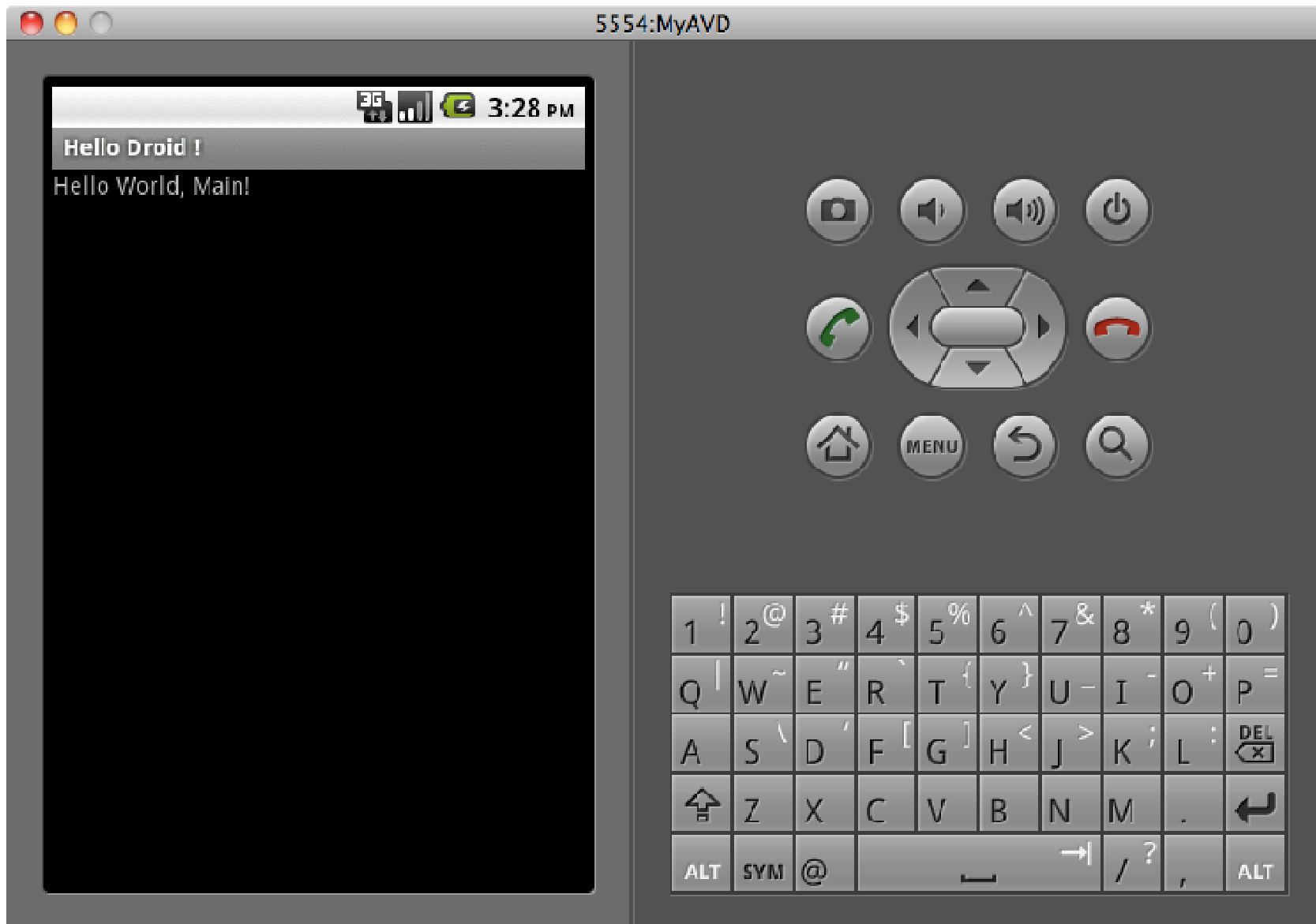
- Setup an Android Virtual Device
 - Return to **Android SDK and AVD Manager**
 - Select **Virtual Devices**
 - Click **New...** and fill the form to create the new AVD
- Execute your application in your emulator
 - Right click on your project
 - Run As...
 - Android Application





Test

Execute it in an Emulator



Next topics covered

- The Android Platform
- Android Project Tree
- Activities
- User Interface
- Intent
- Persistence
- REST Web Services



The Android Platform

Discover what is really Android.



The Android Platform

Introduction

- Apple transformed the telephone market with the iPhone :
 - New Functionalities
 - New Uses
 - New Opportunities (Application market)
- Difficulties for competitors...

- End of 2007 : **Open Handset Alliance** creation
 - Coalition to promote a new OS : **Android**
 - 34 IT companies at the creation
 - 79 today



The Android Platform

Open Handset Alliance

ASUS®



Sprint®



htc
quietly brilliant



MOTOROLA



vodafone™

Google™

ARM™

• T-Mobile •

ebay



NVIDIA.



The Android Platform

Presentation

- Android Platform Key points :
 - Innovative : Integrates all the latest phone technology
 - Touchscreen, GPS, Camera, Accelerometer...
 - Accessible :
 - No need to buy specific Software or Hardware to develop
 - Java is the most widely used language, no need to learn another little used language
 - Open :
 - Open Source license : everyone can see the sources and working on it



The Android Platform

Presentation

The definition of open according to Android chief
Andy Rubin :

```
the definition of open: "mkdir android ;  
cd android ; repo init -u  
git://android.git.kernel.org/platform/m  
anifest.git ; repo sync ; make"
```

about 2 hours ago via web

Retweeted by 100+ people

Reply Retweet



Arubin

Andy Rubin



The Android Platform Presentation

- Android is designed for mobile devices in a broad sense :
 - Phones
 - Tablets
 - Televisions
 - Headphones
 - Microwaves
 - ...





The Android Platform

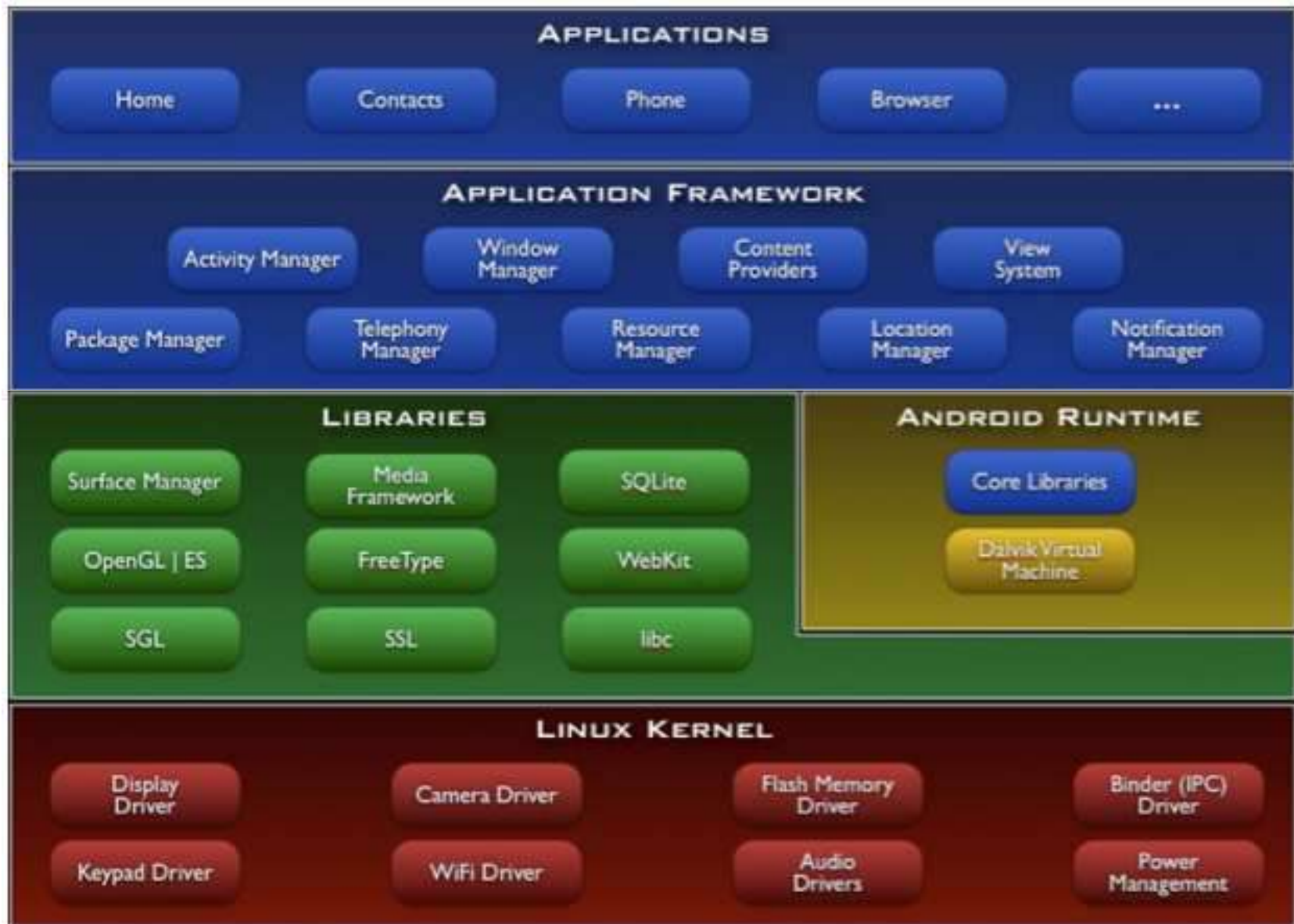
Platform Components

- Android is composed of different layers :
 - A Linux Kernel
 - Libraries for UI, Multimedia, Persistence, etc...
 - A specific Java Virtual Machine called **Dalvik**
 - An Application Framework with many features
 - Applications



The Android Platform

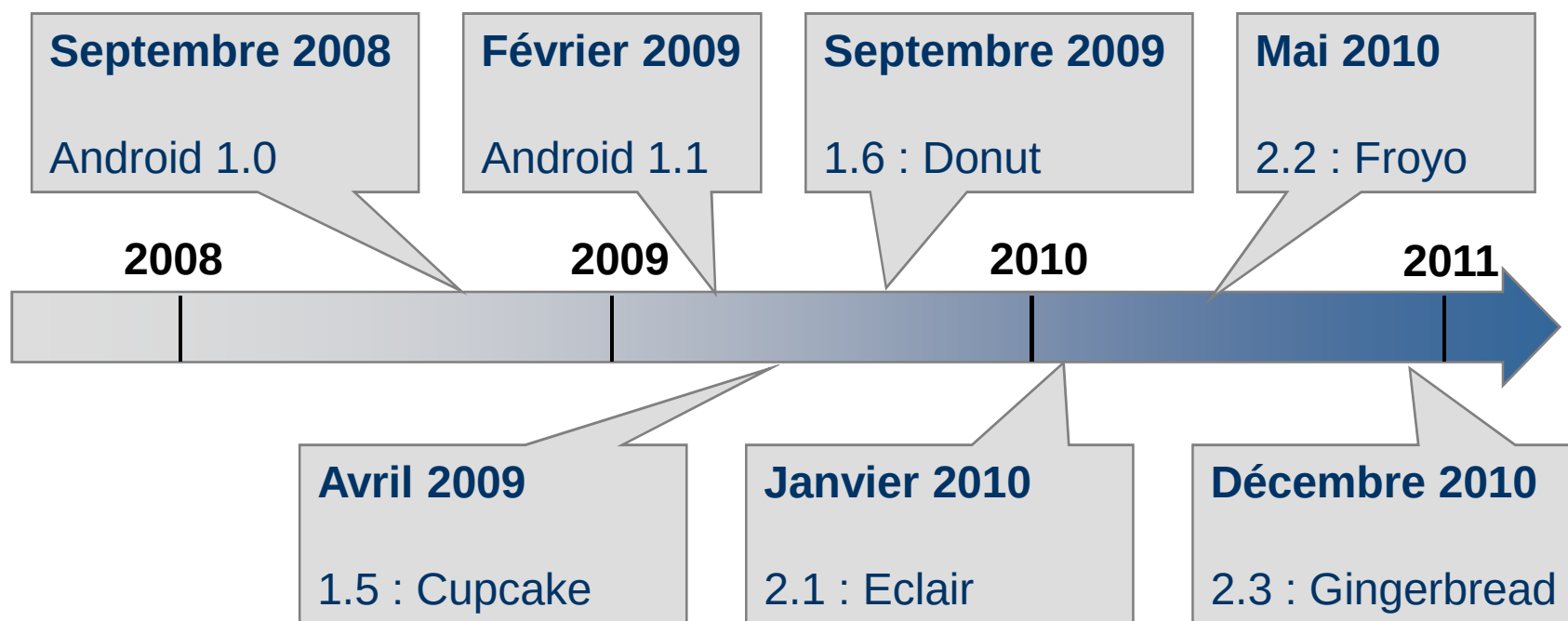
Platform Components





The Android Platform

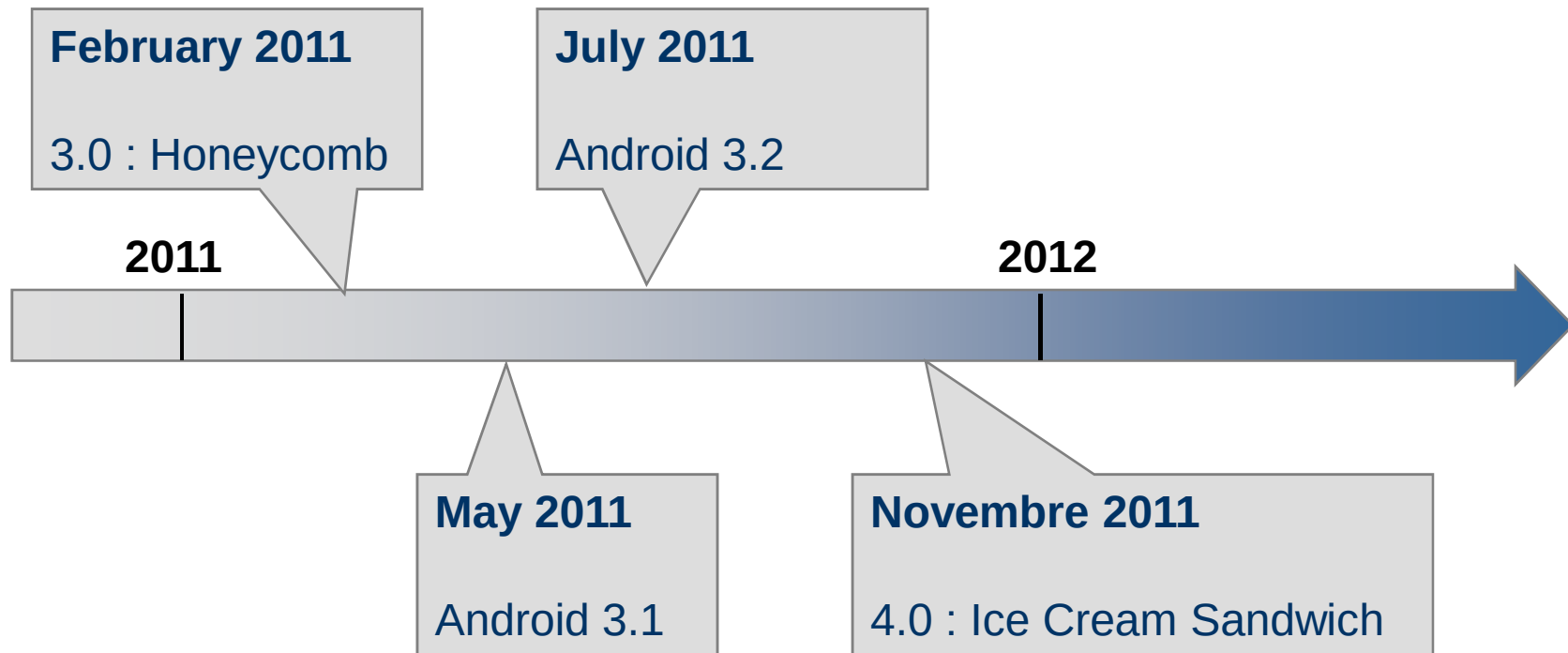
Versions





The Android Platform

Versions



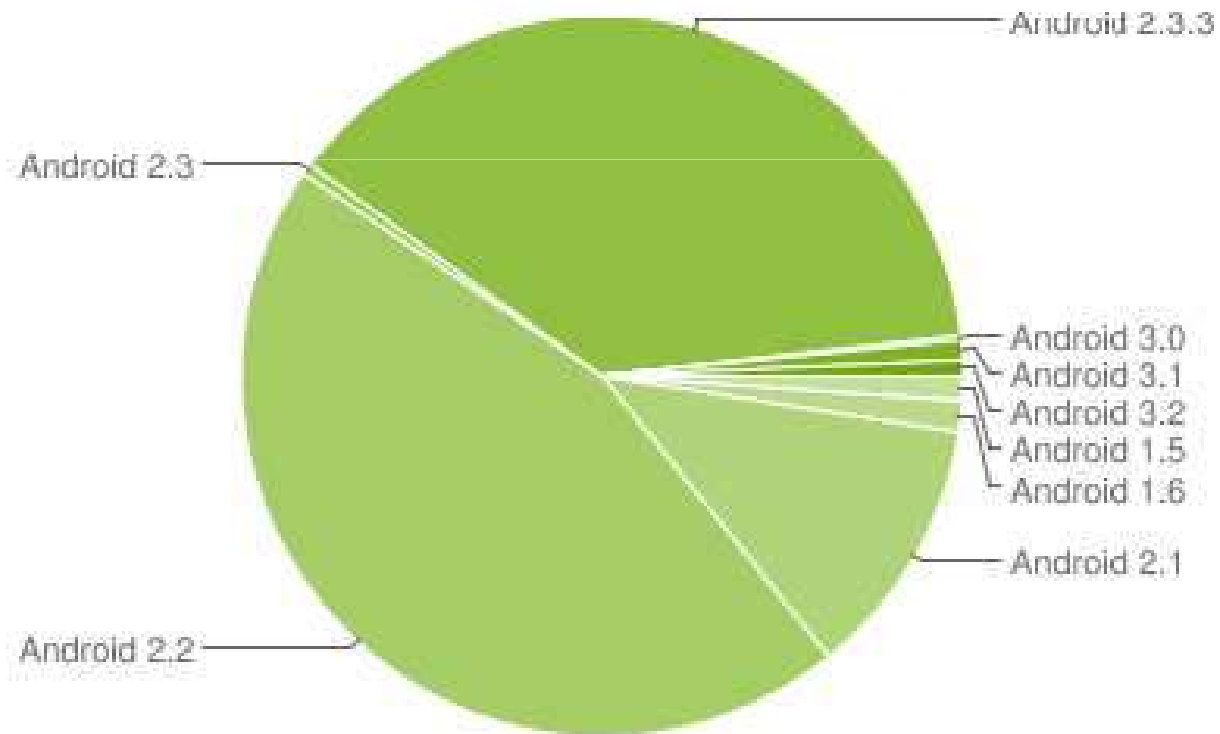


The Android Platform

Android Fragmentations

■ Version $\geq$ Android 2.0 $\rightarrow$ 97.5 %

■ Version $\geq$ Android 2.2 $\rightarrow$ 85.8 %





The Android Platform

Android and Competitors

- Today, Android is the No. 1 Worldwide Mobile Operating System !

**Worldwide Smartphone Sales to End Users by Operating System in 2Q11
(Thousands of Units)**

Operating System	2Q11 Units	2Q11 Market Share (%)	2Q10 Units	2Q10 Market Share (%)
Android	46,775.9	43.4	10,652.7	17.2
Symbian	23,853.2	22.1	25,386.8	40.9
iOS	19,628.8	18.2	8,743.0	14.1
Research In Motion	12,652.3	11.7	11,628.8	18.7
Bada	2,055.8	1.9	577.0	0.9
Microsoft	1,723.8	1.6	3,058.8	4.9
Others	1,050.6	1.0	2,010.9	3.2
Total	107,740.4	100.0	62,058.1	100.0

Source: Gartner (August 2011)

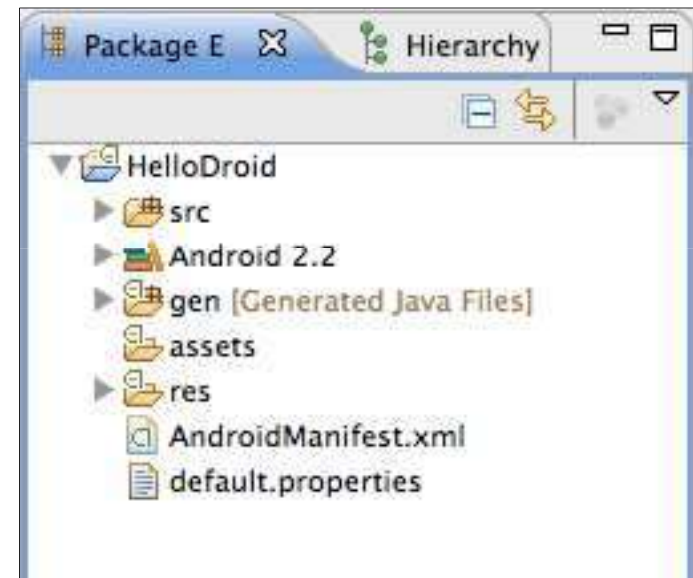
Android Project Tree



Android Project Tree

Overview

- When ADT create a project, it generate numbers of folders and files :
 - **src** folder
 - **res** folder
 - **gen** folder
 - **assets** folder
 - **AndroidManifest.xml** file
 - **default.properties**





Android Project Tree

Folder : src

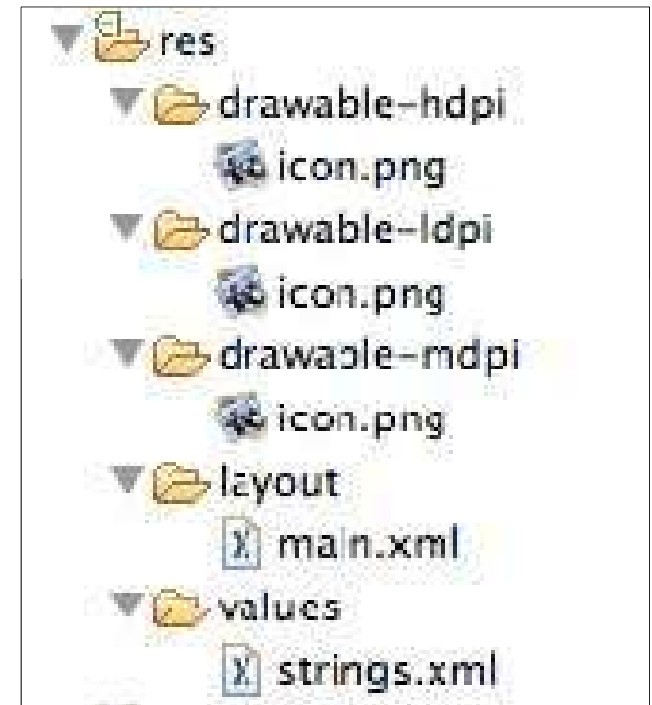
- The same folder as other Java Project
- Contain packages and Java code files



Android Project Tree

Folder : res

- Contain all the resources needed by the application
- Three main types :
 - **drawable** : images and animations
 - **layout** : XML layout files used to construct user interfaces
 - **values** : value type resources such as string and integer constants





Android Project Tree

Folder : gen

- Contain Java files auto-generated by ADT
- Contain the class **R** :
 - Special static class
 - Reference the data contained in resource files
 - Contain one static inner class by resource type

```
public final class R{  
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
    ... //code omitted  
}
```



Android Project Tree

Folder : assets

- Contain asset files
 - Quite similar to resources
 - Accessed in a classic file manipulation style
 - With stream of bytes manipulation
 - Need to use `AssertManager` class to open them
 - Not for an usage as extensive as resources





Android Project Tree

File: AndroidManifest.xml



- Mandatory file in every Android projects
- Contain information needed by Android to run the application
 - Package name of the application
 - List of Activities, Services, Broadcast Receivers, ...
 - Permissions needed by the application
 - etc...



File: AndroidManifest.xml

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.efrei.hellodroid"
  android:versionCode="1"
  android:versionName="1.0">

  <application android:icon="@drawable/icon"
    android:label="@string/app_name">
    <activity android:name=".HelloDroid"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category
          android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    </application>

</manifest>
```



Android Project Tree

File: default.properties



- Contain all the project settings :
 - The build target
 - If the project is a library or not
 - Library references
 - etc...

- Never edit this file manually :
 - Edit this properties by the Properties Project window of Eclipse

Activities

Presentation layer



Activities

Preview

This is the contain we'll see :

- Presentation
- Example
- Life Cycle

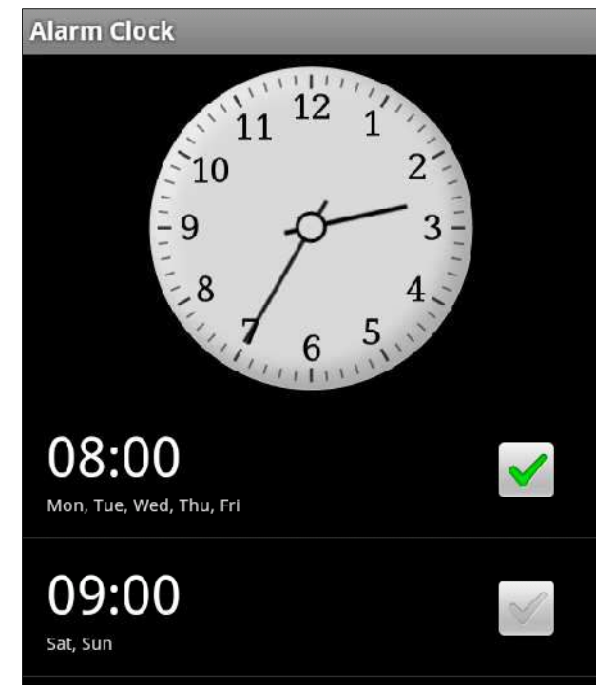




Activities

Presentation

- An activity is a sort of screen composed of several views and controls :
 - For instance an add contact form or a personalized Google Map
- As many activities as application screens
- Presentation layer of an application





Activities

Presentation

- Composed of two parts :
 - **The Activity Logic :**
 - Define in Java inside a class extending **`android.app.Activity`**
 - **The User Interface :**
 - Define either in Java inside the Activity class or inside a XML file (in the folder **`/res/layout/`**)



Activities

Example

- Activity class simple example :

```
package com.efrei.hellodroid;

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

public class Main extends Activity {

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

}
```



Activities

Example

- Layout file simple example :

```
<?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>
```



Activities

Life Cycle

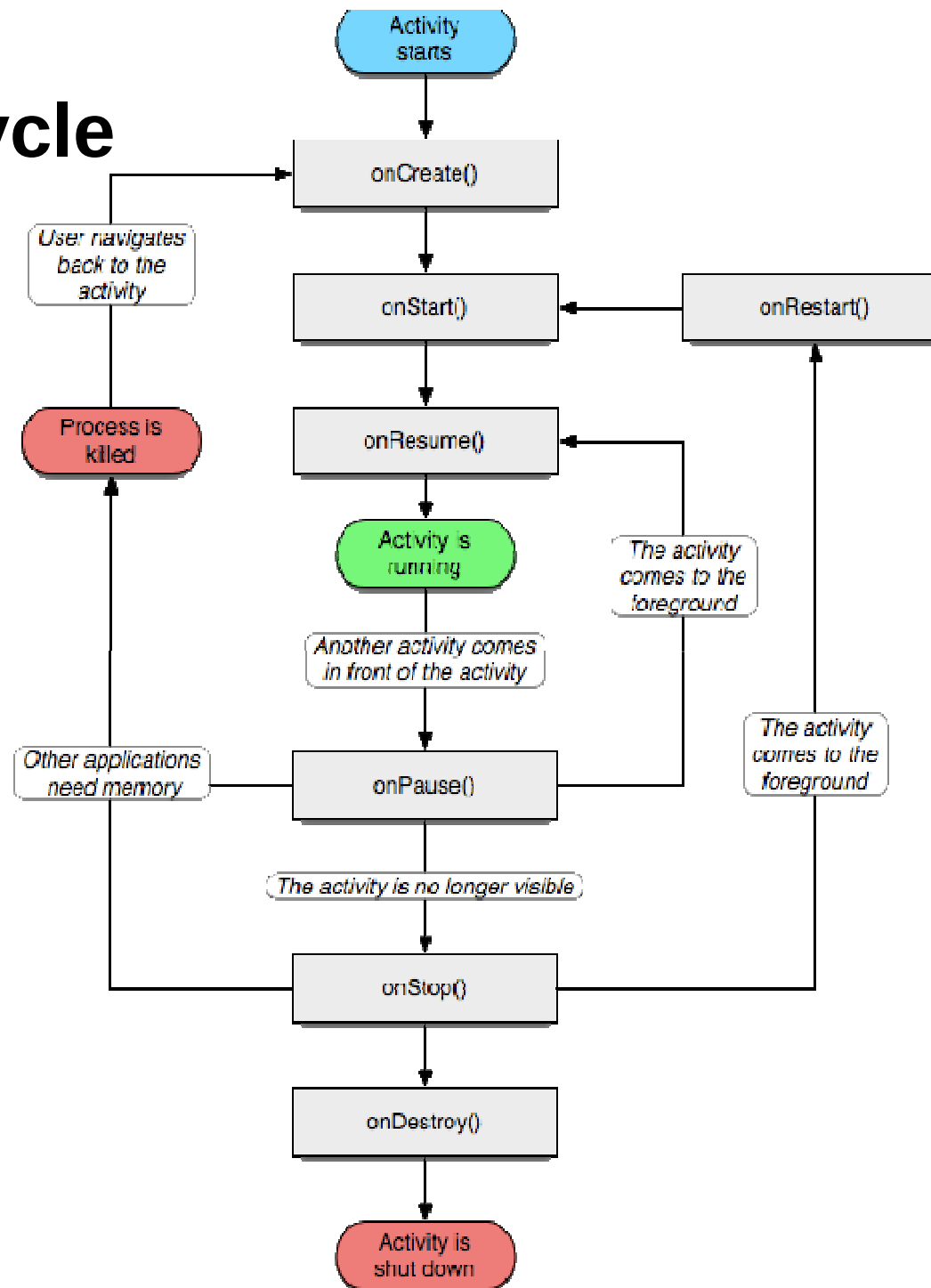
- An activity can have three states :
 - **Active**
 - The activity is visible and has the user focus
 - **Paused**
 - The activity is at least partly visible but doesn't have the focus
 - **Stopped**
 - The activity is not visible
- Activity class defines methods to manage life cycle

Activity	
m onCreate(Bundle)	void
m onStart()	void
m onResume()	void
m onPause()	void
m onStop()	void
m onDestroy()	void
m onRestart()	void
m onSaveInstanceState(Bundle)	void
m onRestoreInstanceState(Bundle)	void



Activities

Life Cycle





Activities

Exercise (1/2)

- Create a new Android Project named ActivityLifeCycle with :
 - Application name : “**Activity Life Cycle**”
 - Package name : “**com.efrei.lifecycle**”
 - Activity name : “**MyActivity**”

- Override the methods below :

■ onCreate(Bundle)	■ onPause()
■ onStart()	■ onStop()
■ onResume()	■ onDestroy()



Activities

Exercise (2/2)

- In each method, add a log statement like this :

```
Log.d("LifeCycle", <method name>);
```

Where <method name> is the name of the method in which you are.

- Deploy and run your application
- Look at the Android Logcat
- Play with emulator to see how the activity Life Cycle work.
 - Display the DDMS perspective inside Eclipse
 - Try to simulate an incoming call or SMS and look how your activity responds

Resources

How to use them.



Resources

Preview

This is the contain we'll see :

- Presentation
- Use Resources
- System Resources
- Simples Values
- Images





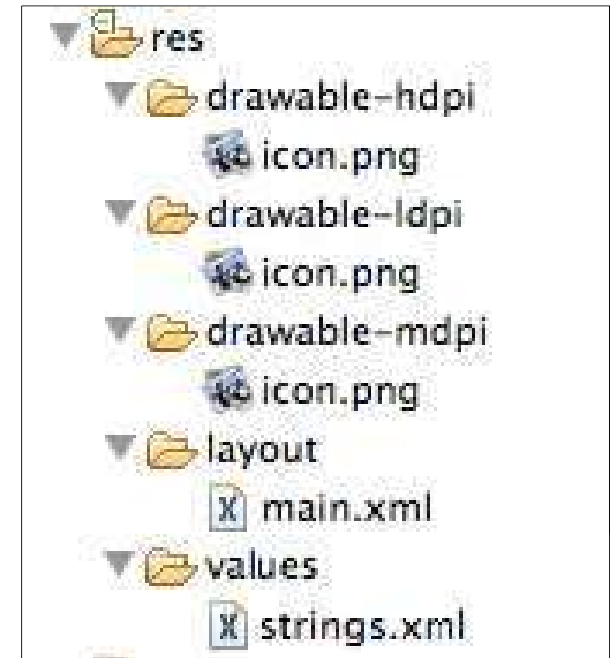
Resources

Presentation

- Android externalize resources like :

- Images
- Strings
- User Interface description
- ...

- Easier to manage and maintain them
- Contained inside the **res** folder





Resources

Use the resources

- Resources are accessible inside the code thanks to the static class : **R**
 - This class is automatically generated by ADT
 - When you add a resource inside the **res** folder, ADT add a reference to it inside the R class
- The syntax to retrieve a resource reference is :

R.resource_type.resource_name



Resources

Example

```
public final class R{  
    public static final class string {  
        public static final int app_name=0x7f020000;  
    }  
    public static final class layout {  
        public static final int my_screen=0x7f030000;  
    }  
    ... //code omitted  
}
```

```
// Define the layout of an activity  
setContentView(R.layout.my_screen);
```

```
// Retrieve the application name  
Resources resources = getResources();  
String appName = resources.getString(R.string.app_name);
```



Resources

Use the resources

- You can also use resources inside XML resources
- Very much used in layouts
- Syntax :

```
"@[package_name:]resource_type/resource_identifier"
```

- Example :

```
...  
    <TextView  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:text="@string/hello"  
    />  
...
```



Resources

System Resources

- Android already includes a number of resources
 - Predefined Colors
 - Predefined Strings
 - Predefined Images
- Examples :

```
String cancel =  
    resources.getString(android.R.string.cancel);
```

```
...  
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:textColor="@android:color/darker_gray"  
    android:text="@string/hello"  
/>  
...
```



Resources

Simple values

- Simple values are stored in XML files inside **/res/values** folder
- You can declare
 - **Strings**
 - You can use the HTML tags ****, **<i>** and **<u>**
 - **Colors**
 - Accept #RGB, #ARGB, #RRGGBB and #AARRGGBB format
 - **Dimensions**
 - In pixels (px), inches (in), millimeters (mm), points (pt), density-independent pixel (dp) or scale-independent pixel (sp)
 - **Arrays**
 - Of Integers or Strings.



Resources

Simple values

■ Example :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="Cyan">#00FFFF</color>
  <string name="first_name">Brice</string>

  <string-array name="my_array">
    <item>A string</item>
    <item>Another String</item>
  </string-array>

  <integer-array name="my_other_array">
    <item>123</item>
    <item>456</item>
  </integer-array>

  <dimen name="my_dimension">4dp</dimen>
  <dimen name="text_size">4px</dimen>
</resources>
```



Resources

Images

- Android accept different bitmap format for resources :
 - **PNG** (advised by the documentation)
 - **JPEG**
 - **GIF** (deprecated)
- From Android 1.6, three folders :
 - **drawable-hdpi** : resource for high-resolution screens
 - **drawable-mdpi** : resources for medium-resolution screens
 - **drawable-ldpi** : resources for low-resolution screens



Resources

Exercise (1/3)

- Create a new Android Project named **GeekQuote** with :
 - Application name : “**Geek Quote**”
 - Package name : “**com.efrei.geekquote**”
 - Activity name : “**QuoteListActivity**”

- Create a package “**com.efrei.geekquote.model**”

- Create a JavaBean class named **Quote** inside it with :
 - A string field named **strQuote**
 - An int field named **rating**
 - A date field named **creationDate**



Resources

Exercise (2/3)

- Define an ArrayList field in your activity :
 - It will contain all our quote objects

- Add a String Array inside your **strings.xml** resource file
 - Set a few quotes that we'll use to initialize our list

- Create a method in your activity class :
 - **void addQuote(String strQuote)**
 - It must convert the string received in parameter to Quote object and add it to your ArrayList



Resources

Exercise (3/3)

- In **onCreate(...)** method
 - Retrieve your string array resource
 - Convert its content to Quote object and add them to your list
 - Use the **addQuote(...)** method to do that

- To see if it works you can use :
 - The Eclipse Debugger to explore your field content
 - Loggers to display it in the LogCat
 - Toast class to display it to the screen as a notification popup



User Interfaces

Views, Controls, and Layouts.



User Interface

Preview

This is the content we'll see

- Presentation
- XML Definition VS Java Definition
- ID Attributes
- Layouts
- Views
- List Adapters
- Events

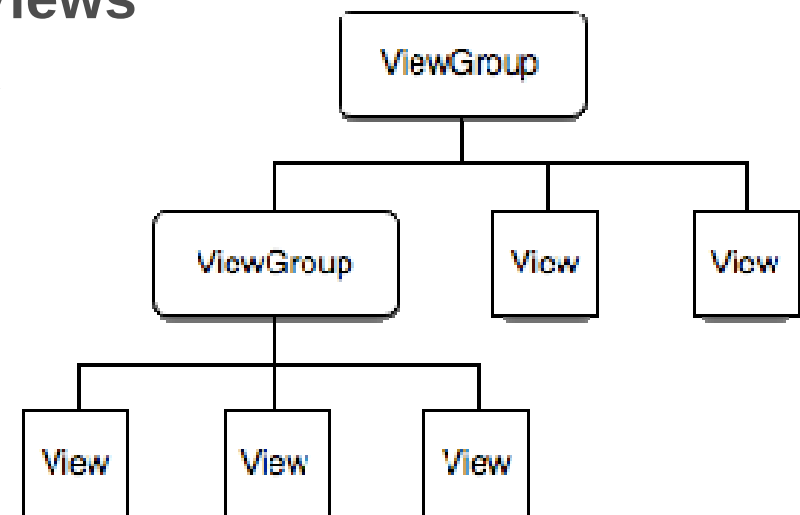




User Interface

Presentation

- A user interface is a set of graphical components like :
 - Button
 - Text
 - Form field
 - Component composed of other components...
- These components are called **Views**
- The last one is a special view called **ViewGroup**





User Interface

Presentation

The image shows a screenshot of an Android application titled "Add a Contact". The interface includes a status bar at the top with "3G", signal strength, battery, and the time "11:36 AM". The form contains two text input fields labeled "First Name" and "Last Name", and two buttons labeled "Add Contact" and "Cancel". Annotations are used to identify UI components: a blue border outlines the main form area, representing a ViewGroup, and green borders outline individual input fields and buttons, representing Views.

— Views

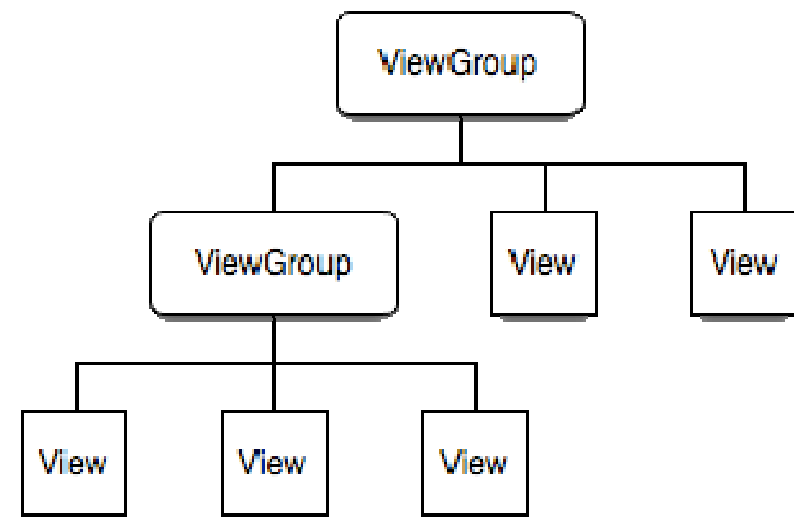
— ViewGroups



User Interface

Presentation

- User interfaces can be defined :
 - In XML, inside a layout resource file
 - Directly in the Activity code
- We're going to see both





User Interface

XML Definition VS Java Definition

- Use XML layout to define user interfaces is advised :
 - Separate interface structure and interface logic
 - Easier to maintain
- But java definition can also be useful :
 - Adding components dynamically





XML Definition : Example

```
<?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="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/first_name"
    />
```

```
<EditText
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/first_name"
    />
```

```
</LinearLayout>
```

Views

GroupView



Java Definition : Example

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
  
    LinearLayout layout = new LinearLayout(this);  
    layout.setOrientation(LinearLayout.VERTICAL);  
    layout.setLayoutParams(  
        new LayoutParams(  
            LayoutParams.FILL_PARENT,  
            LayoutParams.FILL_PARENT));  
  
    TextView textView = new TextView(this);  
    textView.setText(R.string.first_name);  
  
    EditText editText = new EditText(this);  
  
    layout.addView(textView);  
    layout.addView(editText);  
  
    setContentView(layout);  
}
```



ID Attribute

- Ids are typically assigned in the layout XML files, and are used to retrieve specific views inside the Activity code.
- You can ask to ADT to generate one with the special syntax :

```
"@+id/resource_identifier"
```

Instead of :

```
"@id/resource_identifier"
```



User Interface

ID Attribute

■ Example :

```
<EditText  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/first_name"  
/>
```

```
EditText txtFirstName =  
    (EditText) findViewById(R.id.first_name);
```



User Interface

Layouts

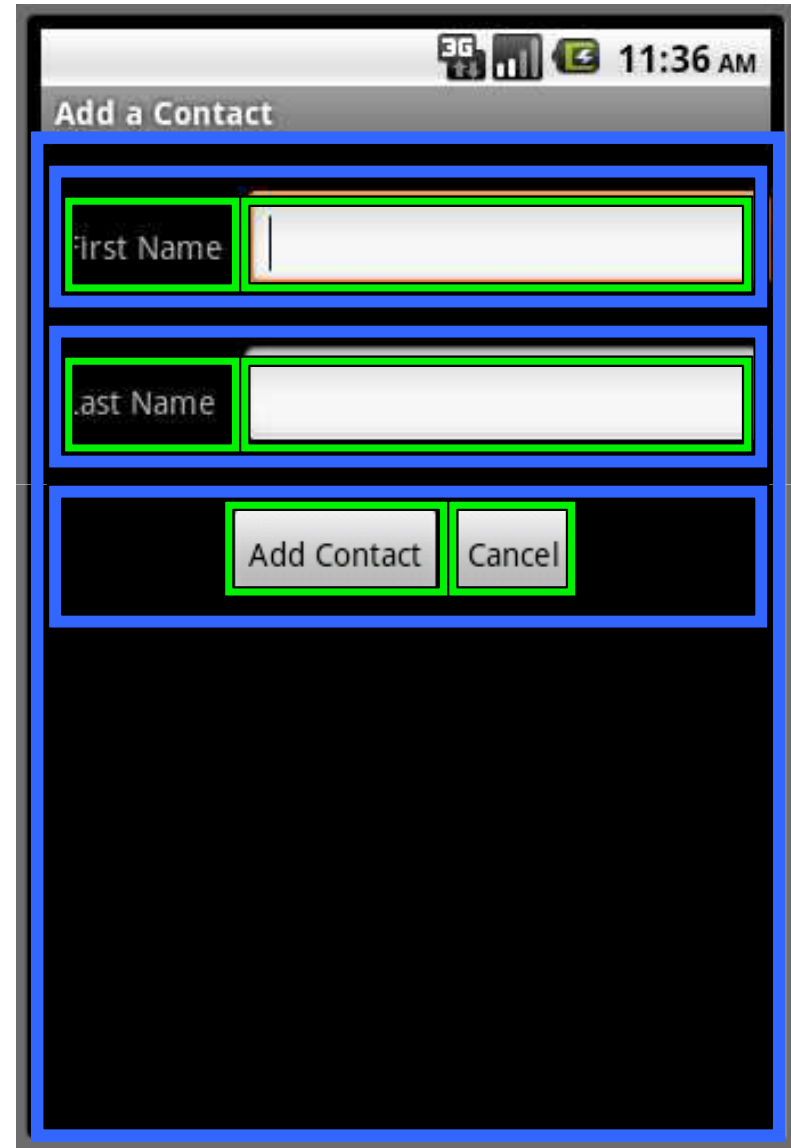
- A layout is a ViewGroup which help us to position our views
- A layout is also a view
- A layout can contain other layouts
- Common layouts provide by the SDK are :
 - **LinearLayout**
 - **RelativeLayout**
 - **FrameLayout**
 - **TableLayout**
- We're going to see only the first one.



User Interface

LinearLayout

- A Layout that arranges its children in a single column or a single row.
- This layout is the more use in Android development
 - It can almost do everything others can do
 - With nesting layout !





LinearLayout : Component Size

- The size of its components can be defined with :
 - In XML with **layout_width** and **layout_height** attributes

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="10px"  
>
```

- In Java with a **LayoutParams** object
- Their values may be a dimension or one of the special constants :

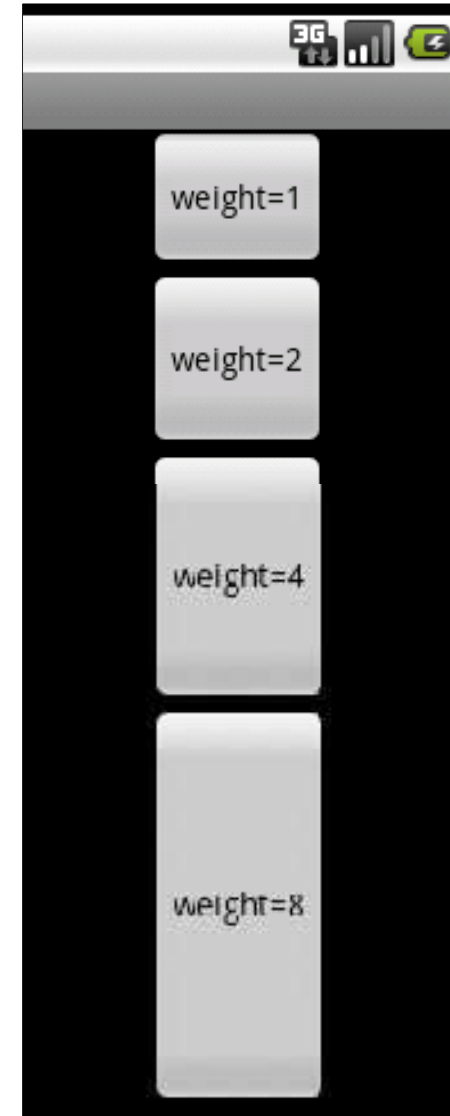
FILL_PARENT, MATCH_PARENT, WRAP_CONTENT



LinearLayout : Weight

- Defined how views on the same row share the layout size
- Useful when you want several views share all the screen
 - Example :

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_weight="2"  
    android:layout_text="weight=2"  
>
```





LinearLayout : Gravity

- Specify how to align the text by the view's x- and/or y-axis when the content is smaller than the view
- Must be one or more (separated by '|') of the **Gravity** class constant values :
 - LEFT / RIGHT
 - TOP / BOTTOM
 - CENTER
 - ...

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:gravity="top|right"  
>
```



LinearLayout : Padding

- By default, components are tightened each other
- You can define space between them thanks to padding !
- Padding is defined as space between the edges of the view and the view's content
- Value in pixels
- Five padding attributes exist :
 - **padding**
 - **paddingLeft**
 - **paddingRight**
 - **paddingTop**
 - **paddingBottom**





User Interface

LinearLayout : Padding

■ XML example :

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:padding="20px"  
>
```

■ Java example :

```
EditText txtFirstName = ... ;  
// left, top, right, bottom  
txtFirstName.setPadding(20, 30, 10, 20);
```



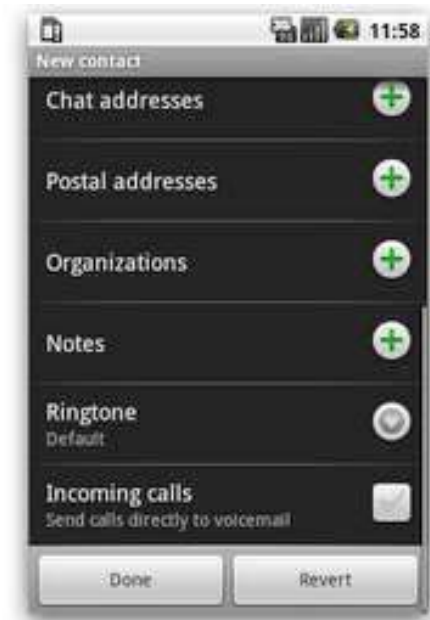
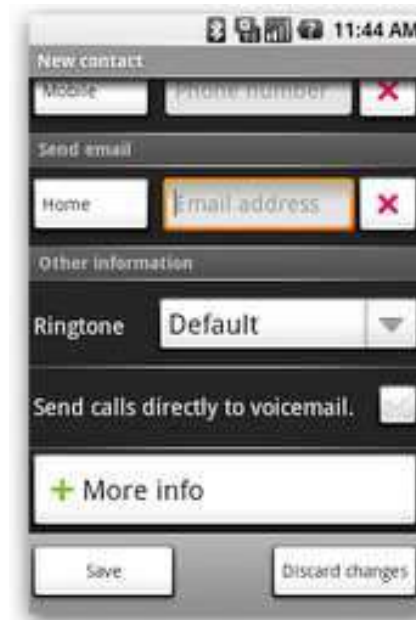
User Interface

Views

- Android SDK propose many common components :

- TextView
- EditText
- AutoCompleteTextView
- RadioButton
- CheckBox
- Spinner
- RatingBar
- Button
- ...

- We're going to see them.





User Interface

TextView

- Displays text to the user
- Can be editable
 - But disable by default

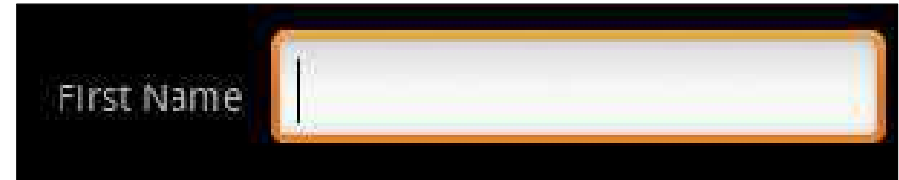


```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:gravity="top|right"  
>
```



User Interface

EditText



- EditText is a subclass of TextView
 - Editable by default !

```
<EditText  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:id="@+id/first_name"  
>
```



User Interface

CheckBox

- A check box is a two-states button that can be either checked or unchecked

```
<CheckBox  
    android:id="@+id/checkbox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="check it out"  
>
```





User Interface

RadioButton

- A radio button is a two-states button that can be either checked or unchecked
- Contrary to checkbox, only one button by radio group can be checked

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/radio_group"
>

<RadioButton
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Easy"

/>

...

</RadioGroup>
```



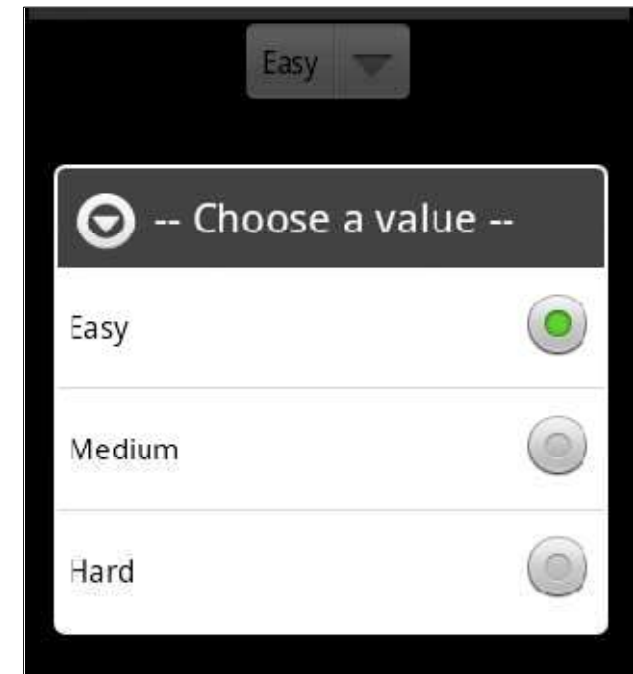
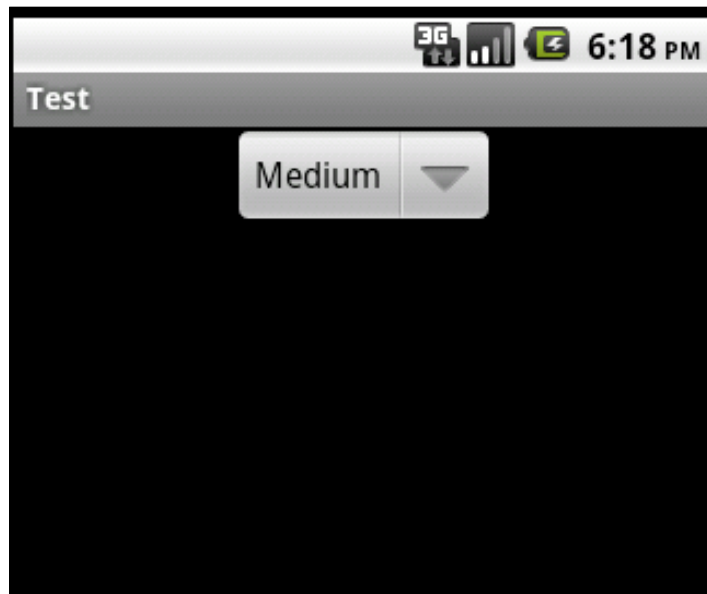


User Interface

Spinner

- A spinner is the Android version of the combo box

```
<Spinner  
    android:id="@+id/spinner"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:prompt="@string/planet_prompt"  
>
```





Spinner : Adapter

- To set spinner options, you need to use a **ListAdapter** object

```
String[] values = { "Easy", "Medium", "Hard" };

ListAdapter adapter =
    new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item,
        values);

adapter.setDropDownViewResource
    (android.R.layout.simple_spinner_dropdown_item);

Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setAdapter(adapter);
```

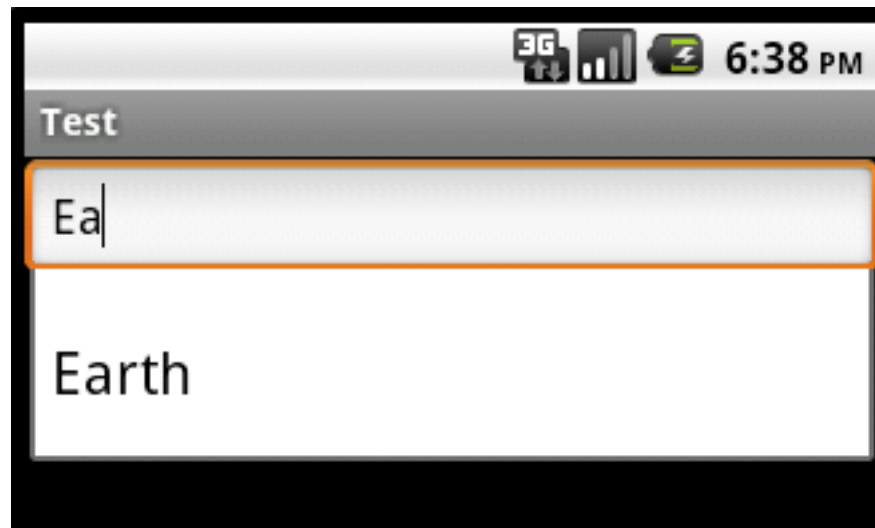


User Interface

AutoCompleteTextView

- An editable text view that shows completion suggestions automatically while the user is typing

```
<AutoCompleteTextView  
    android:id="@+id/autocomplete_planet"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
/>
```





User Interface

AutoCompleteTextView : Adapter

- To set AutoCompleteTextView options, you need to use a **ListAdapter** object again

```
String[] values = { "Mercury", "Venus", "Earth", "Mars" };

ListAdapter adapter =
    new ArrayAdapter<String>(this,
        android.R.layout.simple_dropdown_item_1line,
        values);

AutoCompleteTextView textView = (AutoCompleteTextView)
    findViewById(R.id.autocomplete_planet);
textView.setAdapter(adapter);
```



User Interface

RatingBar

- A RatingBar is a component which represent a rating in stars



- Two special attributes :
 - **numStars** : the number of stars to display
 - **stepSize** : the number equivalent to one star

```
<RatingBar
    android:id="@+id/rating_bar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:stepSize="1.0"
/>
```

In that example, the maximum rate is 5 (numStars * stepSize)



User Interface

Button

- Represents a push-button widget
- Push-buttons can be pressed, or clicked, by the user to perform an action

```
<Button  
    android:id="@+id/my_button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_text"  
>
```





User Interface

ImageButton

- Represents a push-button widget but with an image instead of text inside

```
<ImageButton  
    android:id="@+id/my_button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/logo_google"  
>
```



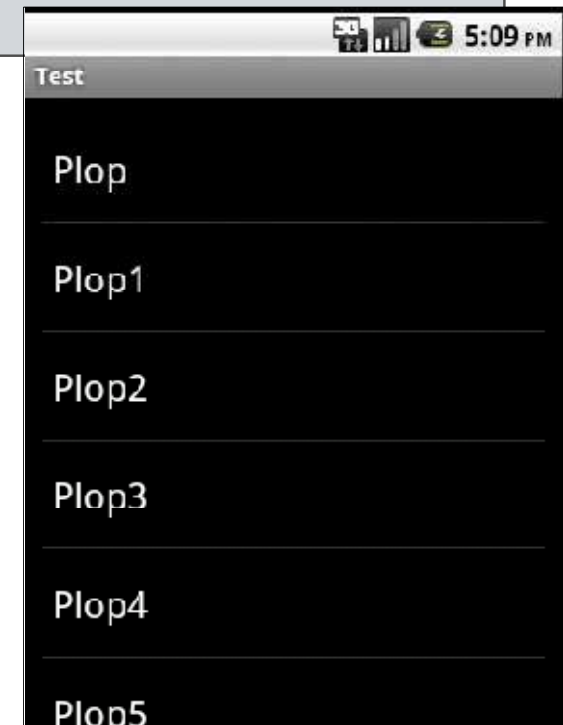


User Interface

ListView

- A view that shows items in a vertically scrolling list

```
<ListView  
    android:id="@+id/my_list_view"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
/>
```





User Interface

ListView : Adapter

- To populate the list, you need to use an **ListAdapter** object again

```
ListView listView =  
    (ListView) findViewById(R.id.my_list_view);  
  
Cursor cursor = new PersonDao(this).getAllPersons();  
  
ListAdapter adapter =  
    new SimpleCursorAdapter(this,  
        android.R.layout.simple_list_item_1,  
        cursor, new String[] { "name" },  
        new int[] { android.R.id.text1 });  
  
listView.setAdapter(adapter);
```



User Interface

More about ListAdapter...

- The bridge between a component and the data that backs the list
- The most used concrete subclasses are :
 - **ArrayAdapter**
 - Adapter to map object arrays or object lists to a view
 - **SimpleCursorAdapter**
 - Adapter to map columns of a cursor to a view
 - We'll see more about cursor later...
- Constructors of these classes take a resource id :
 - The layout to apply to the item of the view
 - You can use one of proposed by the SDK
 - You can define your own layout
- Remember : **android.R ≠ R**

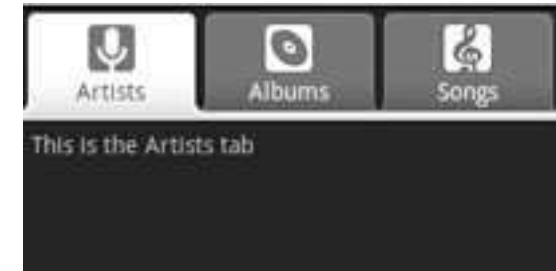


User Interface

Other views

- Now you understand the principle
- Go to see the [Android Documentation](#) for more information
- You will see many more views :

- ImageView
- WebView
- GridView
- DatePicker
- DigitalClock
- ProgressBar
- ToggleButton
- VideoView
- ...





User Interface

Exercise (1/3)

- We're going to design the user interface of our application !
- Your activity layout file have to define :
 - A vertical LinearLayout containing :
 - A horizontal LinearLayout containing :
 - A TextField to enter new quotes
 - A Button to create new quotes



Exercise (2/3)

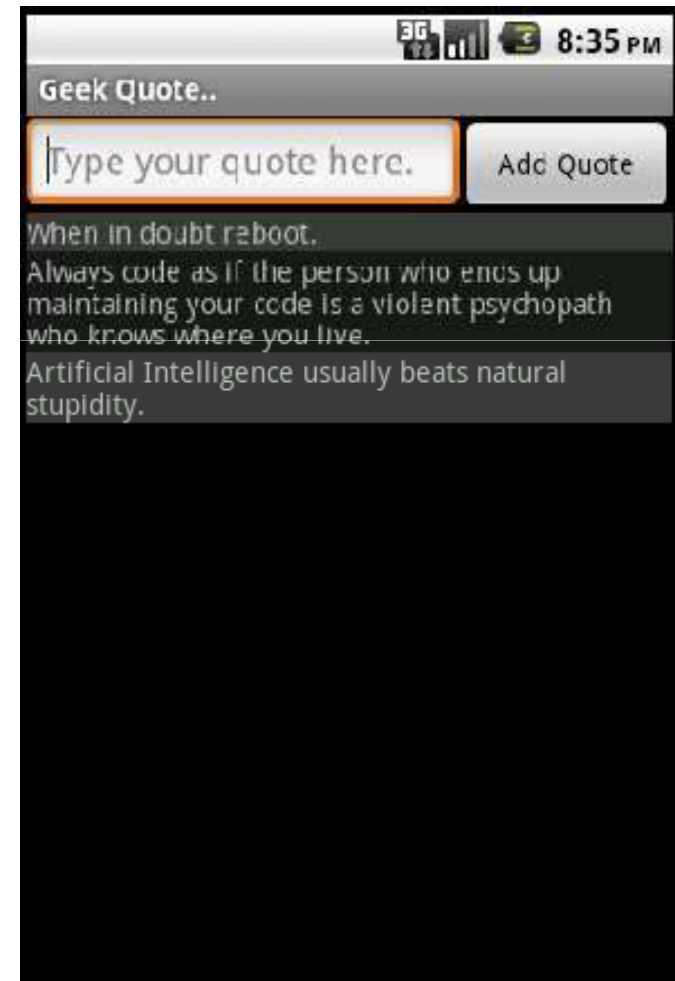
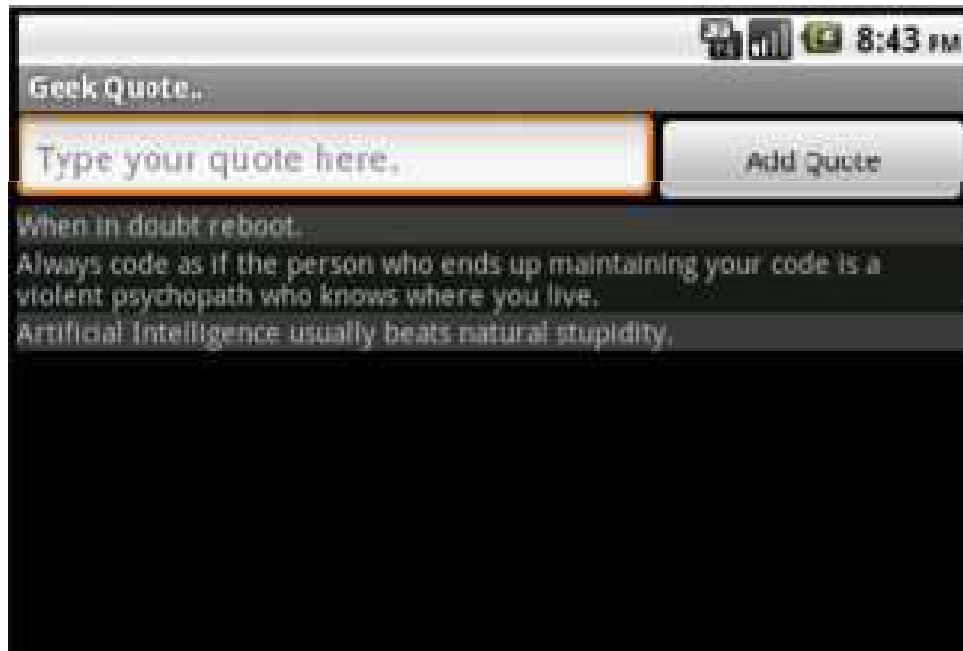
- Modify the **addQuote(...)** method :
 - After add your quote in the ArrayList :
 - Create a **TextView** object
 - Set the quote string as text of it
 - Define a different background-color if the ArrayList size is odd or even
 - Add the TextView to the vertical LinearLayout



User Interface

Exercise (3/3)

- You should get something like that :





User Interface

Events

- With Android, all user actions are events
 - Click
 - Long click
 - Key pressed
 - Item selected
 - ...
- You can link behaviors to this events
- The interception mechanism based on the Listener notion
 - As with Swing !



User Interface

Click Event

- To add a listener to a click event on a view :
 - **`setOnClickListener(View.OnClickListener)`**

- OnClickListener is an inner interface of the View class

- You have three possibilities :
 - Make your activity implements it
 - Create a new class implementing it
 - Create an anonymous class



User Interface

Click Event

■ First solution :

```
public class MyActivity extends Activity
    implements View.OnClickListener {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

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

    public void onClick(View view) {
        // Display a notification popup during 1 second.
        Toast.makeText(this, "Button clicked !", 1000).show();
    }

}
```



User Interface

Click Event

■ Second solution :

```
public class MyActivity extends Activity {  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        Button button = (Button) findViewById(R.id.my_button);  
        button.setOnClickListener(new ButtonClickListener());  
    }  
}
```

```
public class ButtonClickListener  
        implements View.OnClickListener {  
  
    public void onClick(View view) {  
        // Display a notification popup during 1 second.  
        Toast.makeText(this, "Button clicked !", 1000).show();  
    }  
}
```



User Interface

Click Event

■ Third solution :

```
public class MyActivity extends Activity {  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        Button button = (Button) findViewById(R.id.my_button);  
  
        button.setOnClickListener(new View.OnClickListener() {  
  
            public void onClick(View view) {  
                // Display a notification popup during 1 second.  
                Toast.makeText(MyActivity.this, "Clicked!", 1000)  
                    .show();  
            }  
  
        });  
    }  
}
```



Other Events

- All events are based on the same principle
- Some have to return if the event has been consumed
 - If true, the event does not fire other listeners

```
EditText editText = (EditText) findViewById(R.id.my_text);  
editText.setOnTouchListener(new View.OnTouchListener() {  
    public boolean onTouch(View view, MotionEvent e) {  
        Toast.makeText(MyActivity.this, "Touch!", 1000)  
            .show();  
        return true;  
        // True means the listener has consumed the event.  
    }  
});
```



User Interface

Exercise (1/4)

- Now you know how to use listeners :
 - When the user click on the add quote button :
 - Create a new Quote object and add it to the ArrayList and to the screen
 - Empty the text field



User Interface

Exercise (2/4)

- We're going to use a ListView instead of create TextViews in our code
- Modify the layout and add a ListView component
- Create an ArrayAdapter object to link our ArrayList and the ListView
 - Don't forget to override the **toString()** method of **Quote** class
 - ArrayAdapter call this method to display elements in the ListView



User Interface

Exercise (3/4)

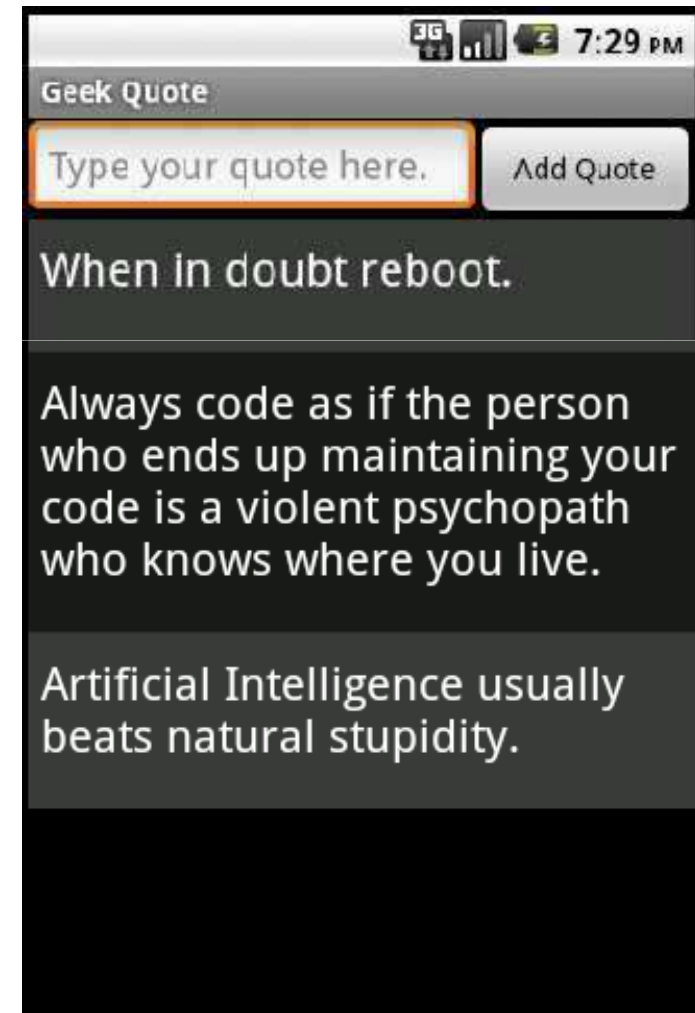
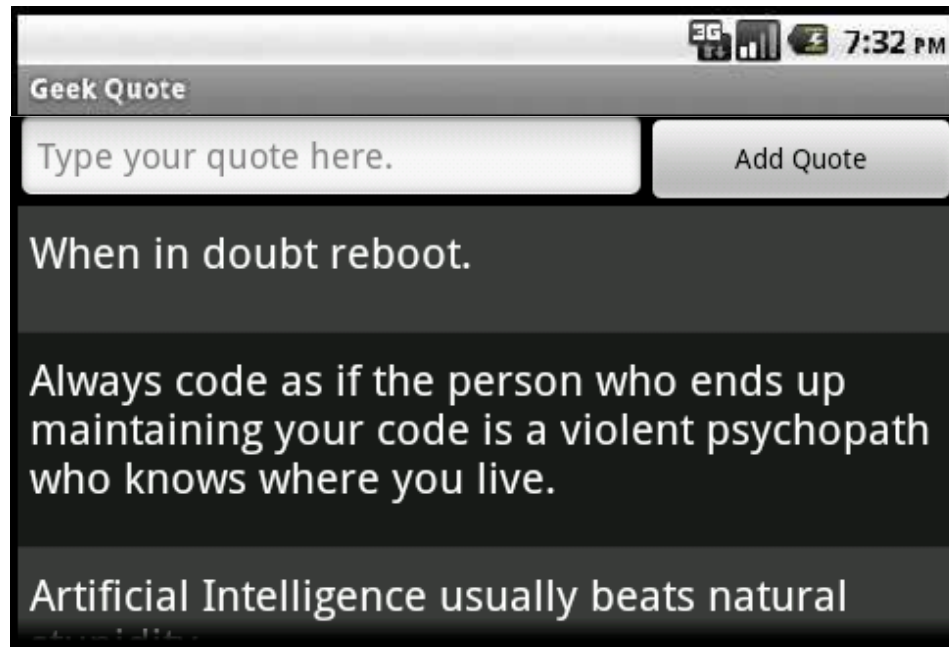
- Use **toString()** method to display something to the user is not a good practice...
 - Create your own **ListAdapter** is better !
- Create a class named **QuoteListAdapter** which :
 - Extends **BaseAdapter** class
 - Override the **getView(...)** method :
 - Look at the ArrayAdapter code of that method :
http://greppcode.com/snapshot/repository.greppcode.com/java/extension/com.google.android/android/2.2_r1.1/
 - Make your own to :
 - Avoid to use **toString()** method
 - Change background color according to parity



User Interface

Exercise (4/4)

- You should get something like that :





Intent

Communication between components.



Intent

Preview

This is the content we'll see

- Presentation
- Launch an Activity
- Include Extra Data
- Implicit Intents
- Native Actions





Intent

Presentation

- An intent is an abstract description of an operation to be performed
- We can use it to :
 - Launch an Activity
 - Communicate with components like :
 - Background Services
 - Broadcast Receivers
- The first one is the most common usage
 - We'll only see it





Intent

Launch an Activity

- To simply launch an activity :

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

- One of Instant constructors take only this two parameters :
 - The context of the intent, here the activity instance creating it
 - The component class used for the intent
- **startActivity(Intent) :**
 - An instance method of Activity class to start a new activity with an intent



Intent

Note

Remember :

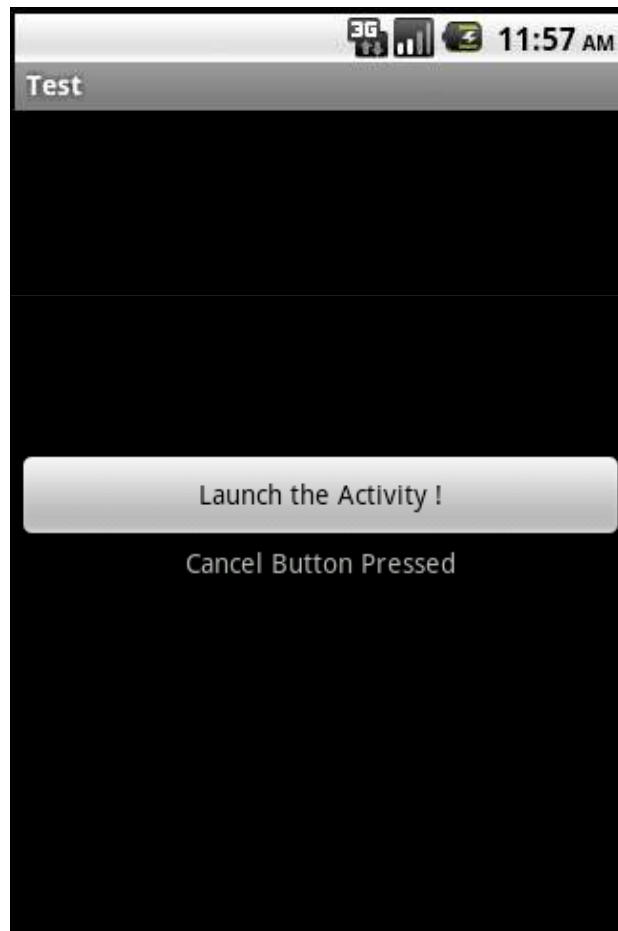
An Activity has to be declared inside Android Manifest file to be launched.



Intent

Launch an Activity

- You can also start an Activity and wait a result code from it :





Intent

Launch an Activity

- To do that, just use the method **startActivityForResult(...)** instead of **startActivity(...)** :

```
...
```

```
private static final int MY_ACTIVITY_CODE = 1;
```

```
...
```

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

```
// MY_ACTIVITY_CODE Constant represent an id for the  
// request that will be used later identify the activity  
// returning the result.
```

```
startActivityForResult(intent, MY_ACTIVITY_CODE);
```



Intent

Launch an Activity

- In the launched activity, use the **setResult(...)** method to return a result code to the launching activity :

```
Button submitButton = (Button) findViewById(R.id.submit);
submitButton.setOnClickListener(new View.OnClickListener() {

    public void onClick(View view) {
        setResult(RESULT_OK);
        finish();
    }

});
```



Intent

Launch an Activity

- In the launching activity, override the **onActivityResult(...)** method :

```
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
    switch (requestCode) {
    case MY_ACTIVITY_CODE:
        TextView textView = ...

        switch (resultCode) {
        case RESULT_CANCELED :
            textView.setText("Cancel Button Pressed");
            break;
        case RESULT_OK :
            textView.setText("Submit Button Pressed");
            break;
        }
        ...
    }
}
```



Intent

Include Extra Data

- When you launch another activity, you often need to communicate some information
- You can use the intent methods below :
 - **void putExtra(...)**
 - **Bundle getExtras(...)**
- Supported types are :
 - Primitives : byte, short, int, long, float, double, ...
 - Primitive Arrays : int[], long[], ...
 - Strings
 - Serializable objects



Intent

Include Extra Data

- To put an extra data :

```
Intent intent = new Intent(this, MyActivity.class);  
intent.putExtra("smthg", "Hi Activity.");  
startActivity(intent, MY_ACTIVITY_CODE);
```

- To retrieve it in the launched Activity :

```
Bundle extras = getIntent().getExtras();  
  
if(extras != null) {  
    String message = extras.getString("smthg");  
}
```

- Intent getIntent() :

- Return the intent that started this activity



Intent

Include Extra Data

- If an Activity has been launched by **startActivityForResult(...)** method :
 - It can send information to the launching Activity
 - In sending an intent in addition to result code
- You can retrieve it in the launching Activity in the **onActivityResult(...)** method





Intent

Include Extra Data

■ Launched Activity :

```
...
setResult(RESULT_OK);
getIntent()
    .putExtra("message", "Thank you for calling me");
finish();
...
```

■ Launching Activity :

```
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data)
{
    ...

    Bundle extras = data.getExtras();
    if(extras != null) {
        String message = extras.getString("message");
    }
}
```



Intent

Implicit Intents

- Two primary forms of intents :
 - **Explicit Intents** :
 - Provide the exact class to be run
 - **Implicit Intents** :
 - Component to run determined by the system
- We just saw the first one
- We're going to see the second one



Intent

Implicit Intents

- Implicit Intents are based on Actions
- Android provide many native Actions
 - But you can create your own.
- You have mainly two constructors to create an implicit Intent :
 - **Intent (String action)**
 - **Intent (String action, Uri uri)**



Intent

Native Actions

Action	Definition
ACTION_ANSWER	Handle an incoming phone call.
ACTION_CALL	Perform a call to someone specified by the data.
ACTION_DELETE	Start an Activity to delete the given data from its container.
ACTION_DIAL	Shows an UI with the number being dialed, allowing the user to explicitly initiate the call.
ACTION_EDIT	Provide explicit editable access to the given data.
ACTION_SEARCH	Perform a search.
ACTION_SEND	Deliver some data to someone else by SMS or e-mail.
ACTION_SENDTO	Send a message to someone specified by the data.
ACTION_VIEW	Starting the default activity associated with the data to view it.
ACTION_WEB_SEARCH	Perform a web search.



Intent

Native Actions

■ Example :

■ Launch the Android Market :



```
Uri uri = Uri.parse("market://search?"  
                    + "q=pname:com.google.android.stardroid");  
Intent intent = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(intent);
```

■ Launch a Web browser :

```
Uri uri = Uri.parse("http://www.efrei.com");  
Intent intent = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(intent);
```



Intent

Native Actions

- Example :

- Call a number :

```
Uri uri = Uri.parse("tel:0607080910");  
Intent intent = new Intent(Intent.ACTION_CALL, uri);  
startActivity(intent);
```

- To make this piece of code work

- You have to specify that the application have the permission to call.
 - Just add a **<use-permission>** element in your Android Manifest.

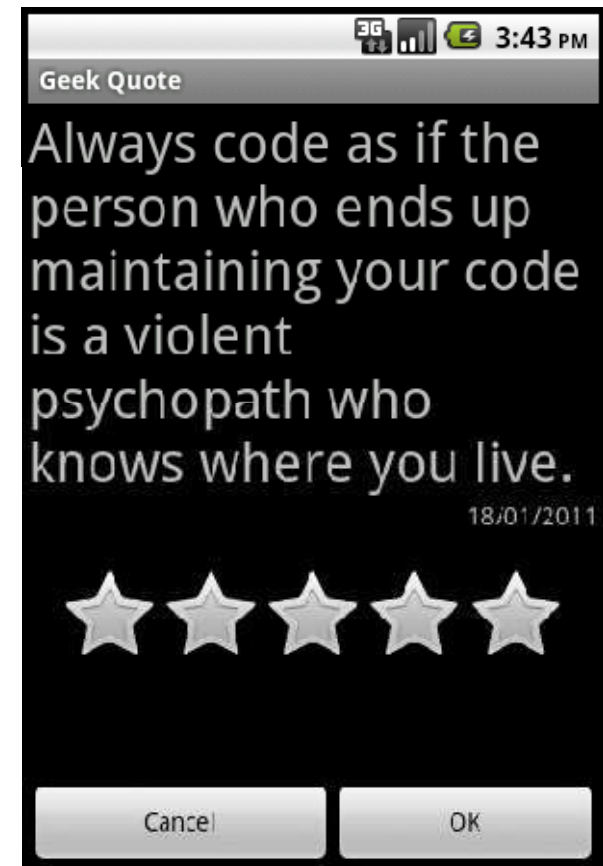
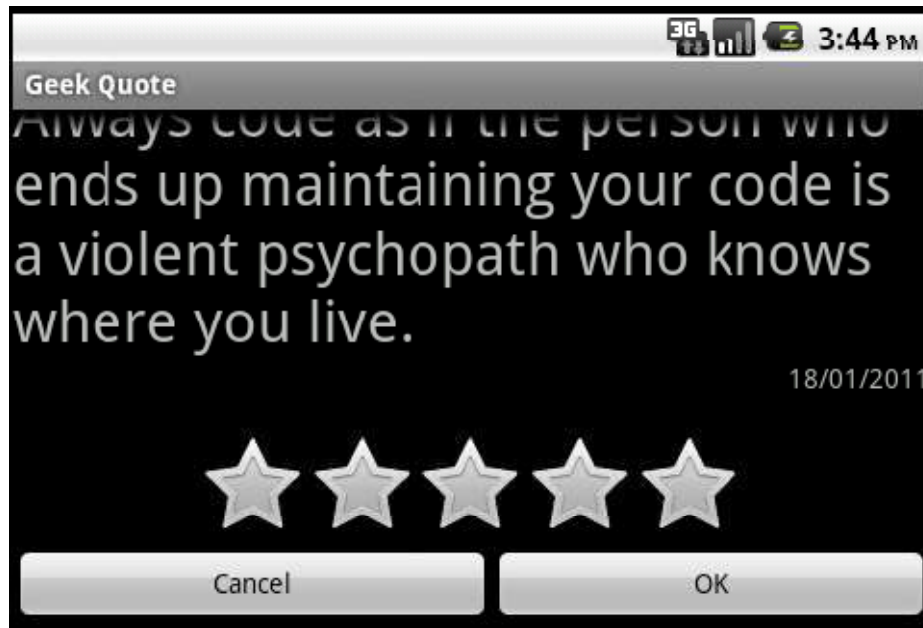
```
<uses-permission  
    android:name="android.permission.CALL_PHONE" />
```



Intent

Exercise (1/3)

- Now, you know how to use Intents :
 - Create a new Activity named **QuoteActivity** :
 - User Interface should be something like that :





Intent

Exercise (2/3)

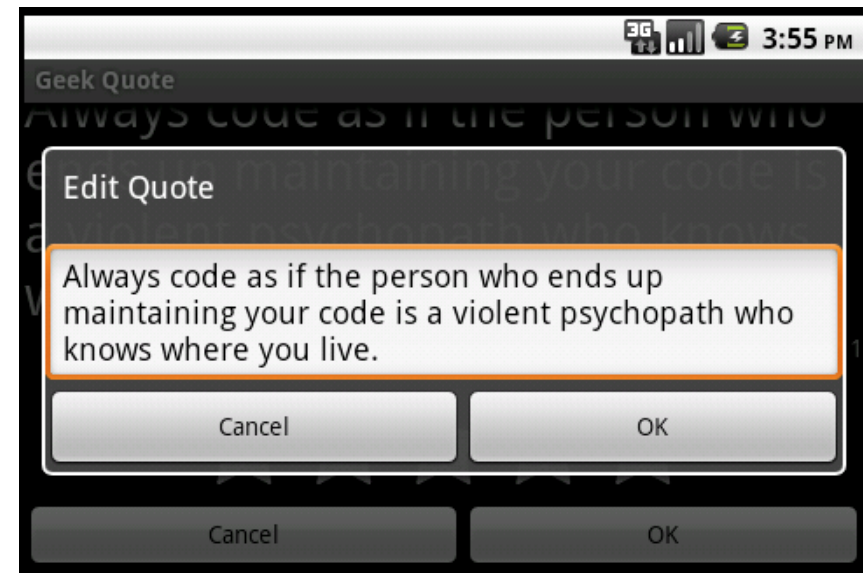
- This activity have to be launch when an user click on a element of the ListView
- It have to display :
 - A TextView with the text of the selected quote
 - Another TextView with the date of the selected quote
 - A RatingBar with the rating of the selected quote
 - This element is editable to allow user to change the rating
 - A Cancel button which return to the previous Activity without save the new ratting
 - A Ok button which return to the previous Activity with the new ratting value



Intent

Exercise (3/3)

- Add a long click event to the TextView containing quote text
- Display a Dialog box allowing user to edit it
- Use Android documentation about Dialog class if you need :
 - <http://developer.android.com/guide/topics/ui/dialogs.html#AlertDialog>



Persistence

How to store data...



Persistence

Preview

This is the contain we'll see :

- Presentation
- Instance State
- Shared Preferences
- SQLite





Persistence

Presentation

- Android provide four ways to store data :
 - Instance State
 - Shared Preferences
 - SQLite databases
 - Files
- We're going to see the first three.



Persistence

Instance State

- You have seen earlier activities life cycle
- A background activity can be unloaded if another one need memory
- How to save activity state to allow user to retrieve his activity as before ?
 - Thanks to Instance State !
- We're going to see the two activity methods to manage instance state :

onSaveInstanceState(...)
onRestoreInstanceState(...)

Activity	
onCreate(Bundle)	void
onStart()	void
onResume()	void
onPause()	void
onStop()	void
onDestroy()	void
onRestart()	void
onSaveInstanceState(Bundle)	void
onRestoreInstanceState(Bundle)	void



Persistence

Instance State

■ onSaveInstanceState(Bundle)

- Called to retrieve per-instance state from an activity before being killed so that the state can be restored in onCreate(Bundle) or onRestoreInstanceState(Bundle) (the Bundle populated by this method will be passed to both).

■ onRestoreInstanceState(Bundle)

- This method is called after onStart() when the activity is being re-initialized from a previously saved state, given here in Bundle type parameter.



Persistence

Instance State

- By default, Instance State save the values of all views with **id** attribute
- If you want to save more information, just override the two methods we have just seen

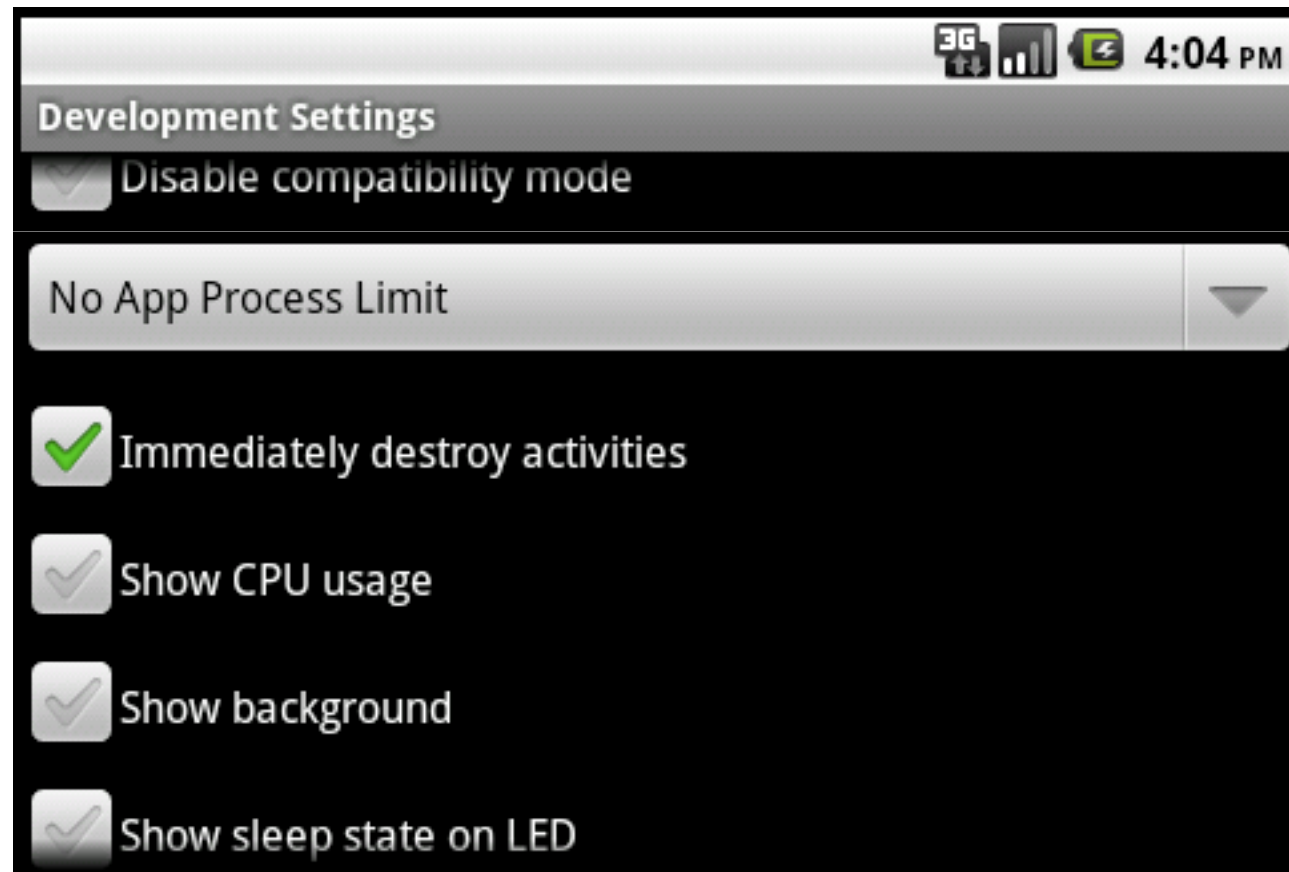
```
private String myInformation;  
...  
  
protected void onSaveInstanceState(Bundle outState) {  
    outState.putString("anotherInformation", myInformation);  
    super.onSaveInstanceState(outState);  
}  
  
protected void onRestoreInstanceState(Bundle savedInstanceState){  
    super.onRestoreInstanceState(savedInstanceState);  
    myInformation =  
        savedInstanceState.getString("anotherInformation");  
}
```



Persistence

Exercise (1/2)

- Open the app launcher
 - Go to **Dev Tools > Development Settings**.





Persistence

Exercise (2/2)

- Re-launch your Geek Quote application
 - Create a new quote
 - Simulate an incoming call
 - Look at your list content...
-
- Use Instance States to correct that !



Persistence

Shared Preferences

- Share across all components in an application
- Set of key/value pair
- Can only store **boolean**, **int**, **long**, **float** and **String** values
- Permission can be given :
 - **MODE_PRIVATE**
 - Default value, the created file is only accessible by the application that created it.
 - **MODE_WORLD_READABLE**
 - Other applications can read the file but not modify it.
 - **MODE_WORLD_WRITEABLE**
 - Other applications can modify the file.



Persistence

Shared Preferences

- Examples :

- Retrieve shared preferences :

```
SharedPreferences prefs =  
    getPreferences(Context.MODE_PRIVATE);  
  
// If there is no value for "username", return null  
String username = prefs.getString("username", null);  
  
// If there is no value for "isAdmin", return false  
boolean admin = prefs.getBoolean("isAdmin", false);  
  
// If there is no value for "id", return zero  
long id = prefs.getLong("id", 0L);
```



Persistence

Shared Preferences

- Examples :
 - Save shared preferences :

```
SharedPreferences prefs =  
    getPreferences(Context.MODE_PRIVATE);  
  
SharedPreferences.Editor editor = prefs.edit();  
editor.putString("username", "Droid");  
editor.putBoolean("isAdmin", true);  
editor.commit();
```



Persistence

SQLite Databases

- Relational Database Management System
- Useful to stock complex data
- Each database is dedicated to only one application
- An application can have several databases
- To share data with another application, you can use a Content Provider (out of the course's scope)

SQLite



Persistence

SQLite Databases

- Don't design your SQLite database as a MySQL or PostgreSQL ones
- Mobile devices are not dedicated database server
 - Little storage space
 - Little memory
- Store only what you need
- Avoid frequent requests
- Design SQLite databases with :
 - A simple structure
 - Data easily identifiable
- Don't store binary data !



Persistence

SQLiteOpenHelper

- To simplify your code to create or update a Database schema, the SDK propose you a Helper class named : **SQLiteOpenHelper**.
- To use it, create your proper class and extend it.

Abstract
Methods

SQLiteOpenHelper		
m	SQLiteOpenHelper(Context, String, CursorFactory, int)	
m	getWritableDatabase()	SQLiteDatabase
m	getReadableDatabase()	SQLiteDatabase
m	close()	void
m	onCreate(SQLiteDatabase)	void
m	onUpgrade(SQLiteDatabase, int, int)	void
m	onOpen(SQLiteDatabase)	void



Persistence

SQLiteOpenHelper

■ Example :

```
public class MyOpenHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "my.db";
    private static final int DATABASE_VERSION = 2;
    private static final String TABLE_NAME = "persons";
    private static final String TABLE_CREATE =
        "CREATE TABLE " + TABLE_NAME + " (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "name TEXT NOT NULL);";

    public MyOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    ...
}
```



Persistence

SQLiteOpenHelper

■ Example :

```
...
```

```
public void onCreate(SQLiteDatabase db) {  
    db.execSQL(TABLE_CREATE);  
}
```

```
public void onUpgrade(SQLiteDatabase db,  
                      int oldVersion, int newVersion) {
```

```
    Log.w("Example", "Upgrading database, this will drop"  
           + "tables and recreate.");
```

```
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);  
    onCreate(db);
```

```
}
```

```
}
```



Persistence

SQLiteOpenHelper

- This class provides two other methods very useful :
 - **SQLiteDatabase getWritableDatabase()**
 - Return a SQLiteDatabase instance to read or write in the Database. Throw an exception if the database cannot be opened for writing (bad permission or full disk).
 - **SQLiteDatabase getReadableDatabase()**
 - Return a SQLiteDatabase instance with read-only access to the database.
- Both will create the database if it doesn't exist.



Persistence

SQLiteDatabase

- Exposes methods to manage a SQLite database
- Has methods to create, delete, execute SQL commands, and perform other common database management tasks
- We're going to see some useful methods :
 - **void execSQL(...)**
 - **long insert(...)**
 - **int update(...)**
 - **int delete(...)**
 - **Cursor query(...)**



Persistence

SQLiteDatabase

- `void execSQL(String sql) :`
 - Execute a single SQL statement that is not a query
 - For example, CREATE TABLE, DELETE, INSERT, etc.
 - Example :

```
SQLiteDatabase db = ...  
  
db.execSQL("DROP TABLE IF EXISTS my_table");
```



Persistence

SQLiteDatabase

- **long insert (String table, String nullColumnHack, ContentValues values) :**
 - Convenience method for inserting a row into the database
 - Three parameters :
 - **table** : The table to insert the row into
 - **nullColumnHack** :
 - SQL doesn't allow inserting a completely empty row
 - If initialValues is empty this column will explicitly be assigned a NULL value
 - **values** :
 - Map containing the column values for the row
 - The keys should be the column names
 - The values the column values



Persistence

SQLiteDatabase

- `long insert (String table, String nullColumnHack, ContentValues values) :`

- Return the row ID of the inserted row

- Example :

```
SQLiteDatabase db = ...

ContentValues values = new ContentValues();
values.put("name", "Cartman");

db.insert("persons", null, values);
```



Persistence

SQLiteDatabase

- `int update (String table, ContentValues values, String whereClause, String[] whereArgs) :`
 - Convenience method for updating rows in the database
 - Four parameters :
 - **table** : the table to update in
 - **values** : a map from column names to new column values
 - **whereClause** : the optional WHERE clause to apply when updating
 - **whereArgs** : an array of the value to apply to the WHERE clause
- Return the number of rows affected



Persistence

SQLiteDatabase

■ `int update (String table, ContentValues values,
String whereClause, String[] whereArgs) :`

■ Example :

```
SQLiteDatabase db = ...  
  
ContentValues values = new ContentValues();  
values.put("name", "John");  
String[] whereArgs = { "1" };  
  
db.update("persons", values, "id=?", whereArgs);
```



Persistence

SQLiteDatabase

- `int delete (String table, String whereClause, String[] whereArgs) :`
 - Convenience method for deleting rows in the Database
 - Three parameters :
 - **table** : the table to delete from
 - **whereClause** : the optional WHERE clause to apply when deleting
 - **whereArgs** : an array of the value to apply to the WHERE clause
- Return the number of rows affected



Persistence

SQLiteDatabase

■ `int delete (String table, String whereClause, String[] whereArgs) :`

■ Example :

```
SQLiteDatabase db = ...  
  
String[] whereArgs = { "1" };  
db.delete("persons", "id=?", whereArgs);
```



Persistence

SQLiteDatabase

- `Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy) :`
- Query the given table, returning a Cursor over the result set
- Seven parameters :
 - **table** : The table name to compile the query
 - **columns** : A list of which columns to return
 - **selection** : A filter declaring which rows to return, formatted as an SQL WHERE clause



Persistence

SQLiteDatabase

- **Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy) :**
 - Seven parameters :
 - **selectionArgs** : You may include ?s in selection, which will be replaced by the values from selectionArgs
 - **groupBy** : A filter declaring how to group rows, formatted as an SQL GROUP BY clause
 - **having** : A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause
 - **orderBy** : How to order the rows, formatted as an SQL ORDER BY clause



Persistence

Cursor

- Provide access to the result set returned by a database query
- Methods commonly used are :
 - **getCount()** : returns the number of rows
 - **moveToFirst()** : moves the cursor to the first row
 - **moveToNext()** : moves the cursor to the next line
 - **isAfterLast()** : returns true if the cursor position is after the last row
 - **getColumnNames()** : returns a string array holding the names of all of the columns in the result set
 - **getColumnIndex(String name)** : return the index of the corresponding column name



Persistence

Cursor

■ Example of use :

```
String[] columns = { "id", "name"};

Cursor result = db.query("persons", columns, null,
                        null, null, null, null);
List<Person> persons = new ArrayList<Person>();

result.moveToFirst();
while(!result.isAfterLast()) {
    Person person = new Person();
    person.setId(result.getLong(0));
    person.setName(result.getString(1));
    persons.add(person);

    result.moveToNext();
}
result.close();

return persons;
```



Persistence

Exercise

- Instance States are cool but it is not the better way to our list of Geek Quotes
- Refactor your application to persist your quotes inside a SQLite Database !
- **Tip** : If you want to explore your database, all SQLite Databases are by default stocked inside **/data/data** folder
 - Use eclipse to import file to your computer and use a SQLite browser to see what are inside



REST Web Services

How to consume them ?



REST Web Services

Preview

This is the contain we'll see :

- Presentation
- Instance State
- Shared Preferences
- SQLite

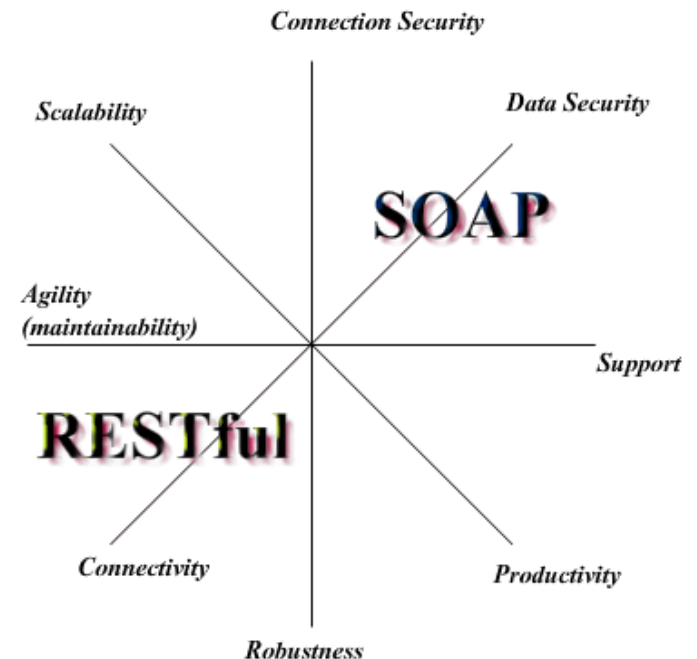




REST Web Services

Presentation

- Android apps can consume SOAP and REST Web Services
- The libraries needed for REST Web Services are directly available in the SDK
- The libraries for SOAP are not





REST Web Services

HttpClient

- Android SDK provides the **HttpClient** Library from the Apache Foundation
- Provide an efficient, up-to-date, and feature-rich package implementing the client side of the HTTP standards and recommendations





REST Web Services

HttpClient

■ Each request with HttpClient follows the steps above :

1. Create an instance of *HttpClient* class
2. Create an instance of *HttpRequest*
3. Configure properties of the request
4. Execute the request with the *HttpClient* instance
5. Analysis and processing of the response



Get Request

```
public String sendGetRequest(String address) {  
    String result = null;  
  
    try {  
        HttpClient httpClient = new DefaultHttpClient();  
        HttpGet httpGet = new HttpGet();  
  
        URI uri = new URI(address);  
        httpGet.setURI(uri);  
  
        HttpResponse response = httpClient.execute(httpGet);  
  
        result = EntityUtils.toString(response.getEntity());  
    } catch (Exception e) {  
        Log.e(LOG_TAG, e.getMessage(), e);  
    }  
  
    return result;  
}
```



Post Request

```
public void sendPostRequest(String address,
                             String entity) {
    try {
        HttpClient httpClient = new DefaultHttpClient();
        HttpGet post = new HttpPost();

        URI uri = new URI(address);
        post.setURI(uri);
        post.setEntity(new StringEntity(entity));

        httpClient.execute(post);
    } catch (Exception e) {
        Log.e(LOG_TAG, e.getMessage(), e);
    }
}
```



Entity

- HTTP Request and HTTP Response can contain data (entity) in different format :
 - JSON
 - XML
 - Simple Text
 - HTML
 - ...
- You have to generate and/or parse them !



REST Web Services

Entity

- To parse XML, Android SDK provides SAX library :
 - <http://www.saxproject.org/quickstart.html>
- To parse JSON, Android SDK provides JSON library :
 - <http://json.org/java/>





JSON Example

```
public Student convertToStudent(JSONObject object)
    throws JSONException {
    Student student = new Student();

    student.setIdBooster(object.getLong(ID_BOOSTER_FIELD));
    student.setFirstName(object.getString(FIRST_NAME_FIELD));
    student.setLastName(object.getString(LAST_NAME_FIELD));
    String formattedDate = object.getString(BIRTH_DATE_FIELD);

    try {
        student.setBirthDate(dateFormat.parse(formattedDate));
    } catch (ParseException e) {
        Log.e(LOG_TAG, "Parse Exception!", e);
    }

    return student;
}
```



JSON Example

```
public JSONObject convertToJson(Student student)
                                throws JSONException {

    JSONObject jsonStudent = new JSONObject();

    jsonStudent.put(ID_BOOSTER_FIELD,
student.getIdBooster());
    jsonStudent.put(FIRST_NAME_FIELD,
student.getFirstName());
    jsonStudent.put(LAST_NAME_FIELD, student.getLastName());
    jsonStudent.put(BIRTH_DATE_FIELD,
                    dateFormat.format(student.getBirthDate()));

    return jsonStudent;
}
```



Exercise

- Create a new Java Web Application :
 - Develop a persistence layer with JPA to store our Geek Quotes
 - Develop a REST Web Service layer with JAX-RS to manipulate our Geek Quotes
 - Refactor your Android Application to use your new Web Services

