



# Getting started: Hello Android

Marco Ronchetti  
Università degli Studi di Trento

---

# HelloAndroid

```
package com.example.helloandroid;

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

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

2



# HelloAndroid

```
package com.example.helloandroid;
```

```
import android.app.Activity;
```

```
import android.os.Bundle;
```

```
import android.widget.TextView;
```

```
public class HelloAndroid extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        TextView tv = new TextView(this);  
        tv.setText("Hello, Android");  
        setContentView(tv);  
    }  
}
```

3

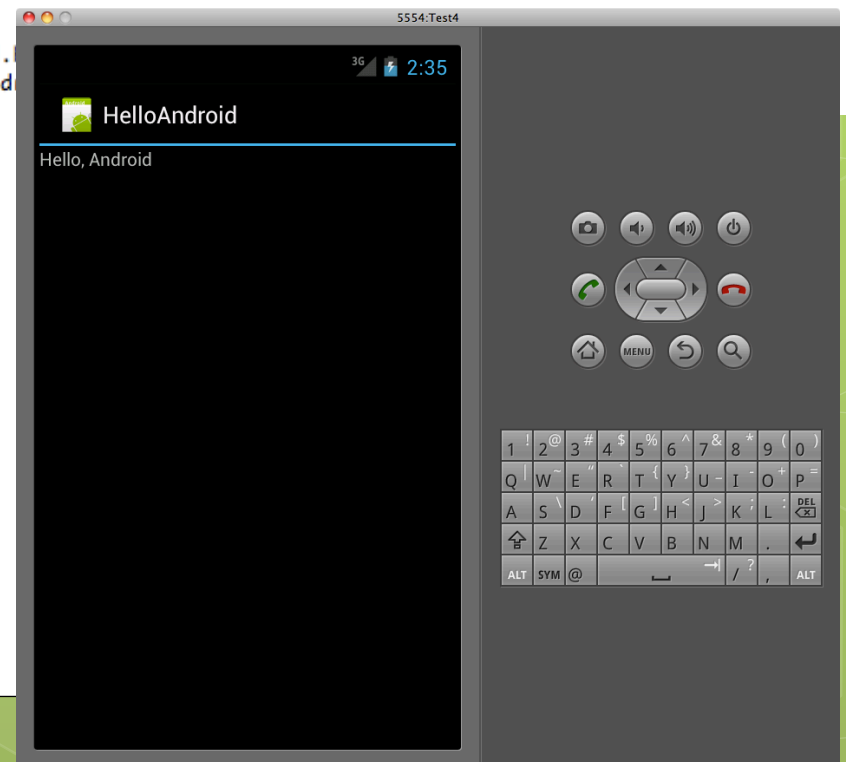


# Launching the emulator...

Problems Javadoc Declaration Console

Android

```
[2012-02-28 14:33:05 - HelloAndroid] -----
[2012-02-28 14:33:05 - HelloAndroid] Android Launch!
[2012-02-28 14:33:05 - HelloAndroid] adb is running normally.
[2012-02-28 14:33:05 - HelloAndroid] Performing com.example.helloandroid.HelloAndroidActivity activity launch
[2012-02-28 14:33:05 - HelloAndroid] Automatic Target Mode: launching new emulator with compatible AVD 'Test4'
[2012-02-28 14:33:05 - HelloAndroid] Launching a new emulator with Virtual Device 'Test4'
[2012-02-28 14:33:07 - Emulator] 2012-02-28 14:33:07.475 emulator-arm[3911:80b] Warning once: This application, or a library it uses, is using
[2012-02-28 14:33:07 - Emulator] emulator: WARNING: Unable to create sensors port: Connection refused
[2012-02-28 14:33:07 - HelloAndroid] New emulator found: emulator-5554
[2012-02-28 14:33:07 - HelloAndroid] Waiting for HOME ('android.process.acore') to be launched...
[2012-02-28 14:33:39 - HelloAndroid] HOME is up on device 'emulator-5554'
[2012-02-28 14:33:39 - HelloAndroid] Uploading HelloAndroid.apk onto device 'emulator-5554'
[2012-02-28 14:33:39 - HelloAndroid] Installing HelloAndroid.apk...
[2012-02-28 14:34:04 - HelloAndroid] Success!
[2012-02-28 14:34:04 - HelloAndroid] Starting activity com.example.helloandroid.
[2012-02-28 14:34:05 - HelloAndroid] ActivityManager: Starting: Intent { act=and
```





# HelloAndroid: questions.

```
package com.example.helloandroid;
```

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

- What is an Activity?
- What is onCreate?
- What is a Bundle?
- What is R?

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

5



```
        TextView tv = new TextView(this);  
        tv.setText("Hello Android");
```

- What is a TextView??



# Dissecting the HelloWorld

Marco Ronchetti  
Università degli Studi di Trento

---

android.app

## Class Activity

# Class Activity

[java.lang.Object](#)

└ [android.content.Context](#)

└ [android.content.ContextWrapper](#)

└ [android.view.ContextThemeWrapper](#)

└ **android.app.Activity**

### All Implemented Interfaces:

[ComponentCallbacks](#), [KeyEvent.Callback](#), [LayoutInflater.Factory](#), [View.OnCreateContextMenuListener](#), [Window.Callback](#)

### Direct Known Subclasses:

[ActivityGroup](#), [AliasActivity](#), [ExpandableListActivity](#), [ListActivity](#)

An activity is a single, focused thing that the user can do.

Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with `setContentView(int)`.

Doesn't it reminds you of "JFrame" and "setContentPane()?"



## android.app Class Activity

# Class Activity

```
java.lang.Object
├── android.content.Context
│   ├── android.content.ContextWrapper
│   │   └── android.view.ContextThemeWrapper
│   └── android.app.Activity
```

Interface to global information about an application environment.

### All Implemented Interfaces:

[ComponentCallbacks](#), [KeyEvent.Callback](#), [LayoutInflater.Factory](#), [View.OnCreateContextMenuListener](#), [Window.Callback](#)

### Direct Known Subclasses:

[ActivityGroup](#), [AliasActivity](#), [ExpandableListActivity](#), [ListActivity](#)

An activity is a single, focused thing that the user can do.

Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with `setContentView(int)`.

Doesn't it reminds you of "JFrame" and "setContentPane()?"



## Class Activity

While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with `R.attr.windowIsFloating` set) or embedded inside of another activity (using `ActivityGroup`).



# Resources

You should always **externalize resources** (e.g. images and strings) from your application code, so that you can:

- **maintain them independently.**
- **provide alternative resources, e.g.:**
  - different languages
  - different screen sizes

Resources must be organized in your project's **res/** directory, with various sub-directories that group resources by type and configuration.



# The R class

When your application is compiled, aapt generates the **R class**, which contains resource IDs for all the resources in your `res/` directory.

For each type of resource, there is an R subclass (for example, **R.layout** for all layout resources) and for each resource of that type, there is a static integer (for example, **R.layout.main**). This integer is the **resource ID** that you can use to retrieve your resource.

11

More about resources in future lectures.



# R.Java in gen/

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.  
 *  
 * This class was automatically generated by the  
 * aapt tool from the resource data it found. It  
 * should not be modified by hand.  
 */
```

```
package com.example.helloandroid;  
public final class R {  
    public static final class attr {  
    }  
    public static final class drawable {  
        public static final int ic_launcher=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;  
    }  
}
```

12





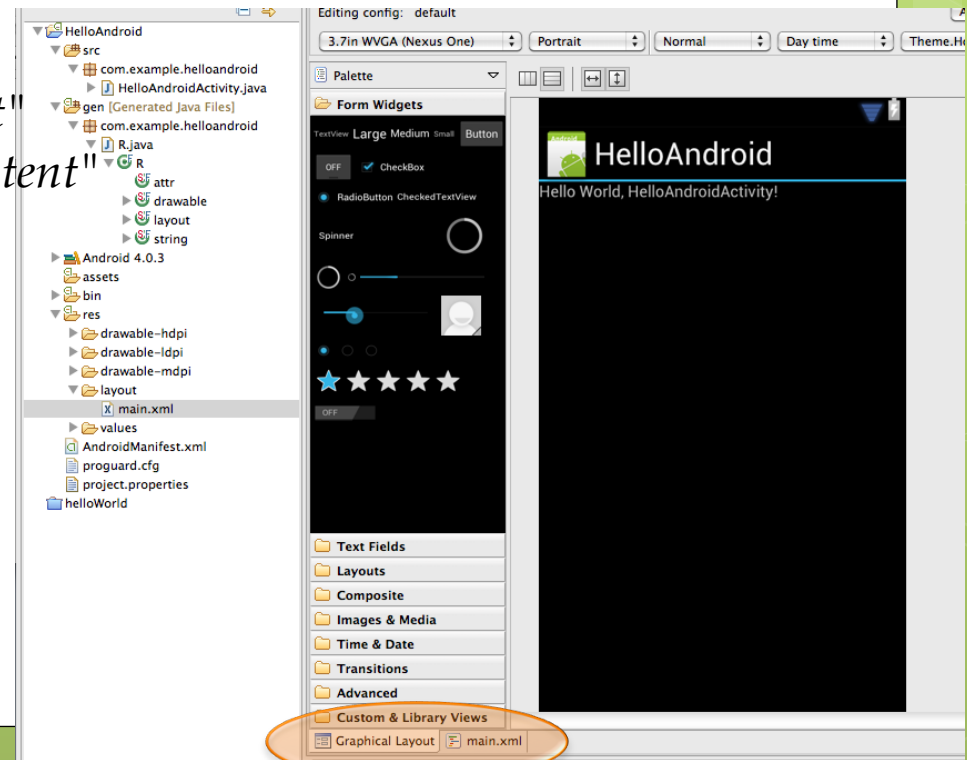
# Res/layout/main.xml

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

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

```
</LinearLayout>
```

13



## onCreate(Bundle b)

Callback invoked when the activity is starting.

This is where most initialization should go.

If the activity is being re-initialized after previously being shut down then this **Bundle** contains **the data it most recently supplied** in onSaveInstanceState (Bundle), otherwise it is null.

Note: a Bundle is a sort of container for serialized data.



# TextView

Displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing; see EditText for a subclass that configures the text view for editing.

android.widget

## Class TextView

java.lang.Object

└ [android.view.View](#)

└ android.widget.TextView

This class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components (buttons, text fields, etc.).

Doesn't it remind you the java.awt.Component?

### All Implemented Interfaces:

[Drawable.Callback](#), [AccessibilityEventSource](#), [KeyEvent.Callback](#), [ViewTreeObserver.OnPreDrawListener](#)

### Direct Known Subclasses:

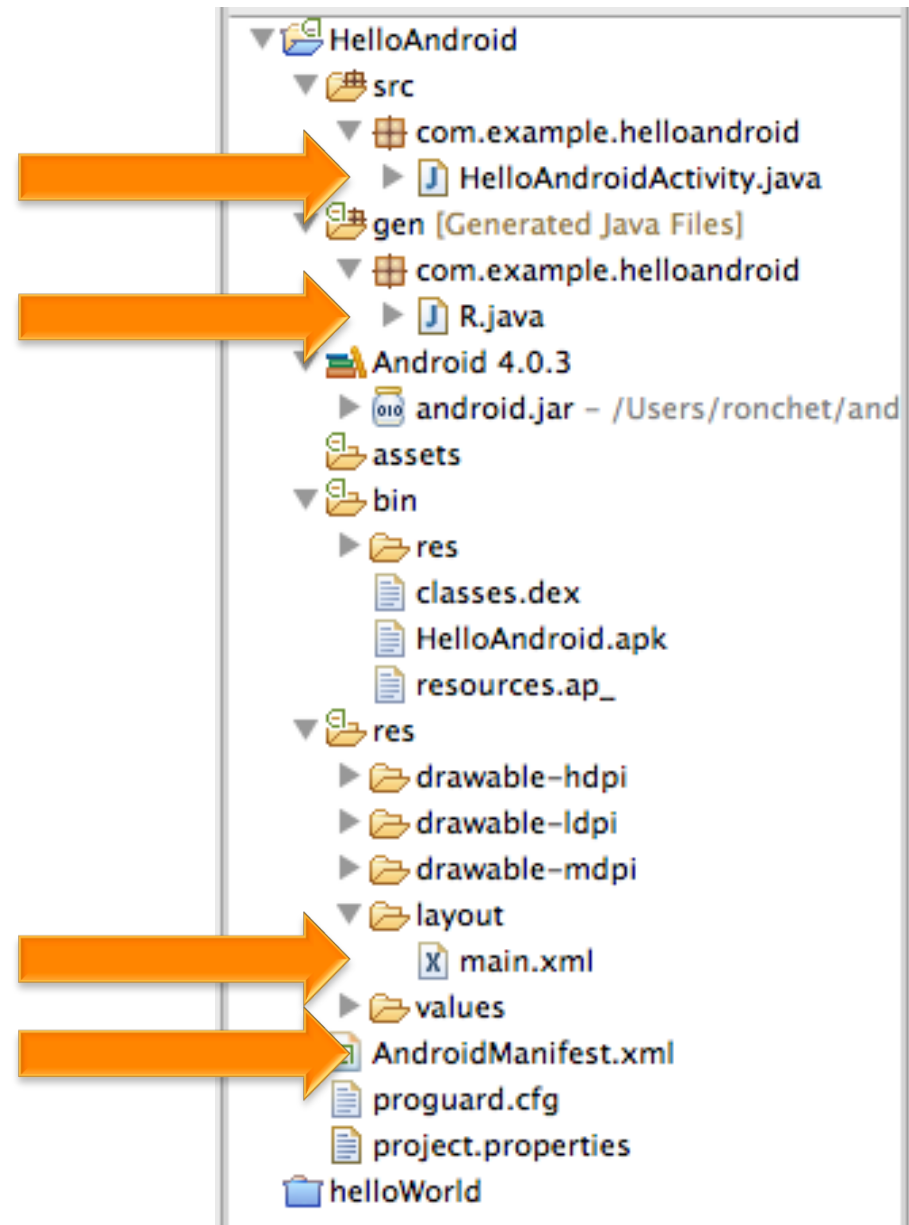
[Button](#), [CheckedTextView](#), [Chronometer](#), [DigitalClock](#), [EditText](#)

15

```
public class TextView
extends View
implements ViewTreeObserver.OnPreDrawListener
```



# The project



# AndroidManifest.xml

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

```
<manifest xmlns:android="http://schemas.android.com/ank/res/android"
```

```
  package="com.example.helloandroid"
```

```
  android:versionCode="1"
```

```
  android:versionName="1.0" >
```

```
  <uses-sdk android:minSdkVersion="15" />
```

```
  <application
```

```
    android:icon="@drawable/ic_launcher"
```

```
    android:label="@string/app_name" >
```

```
      <activity
```

```
        android:name=".HelloAndroidActivity"
```

```
        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>
```

## Platform versions

Platform Version	API Level	VERSION_CODE
<a href="#">Android 4.0.3</a>	15	ICE CREAM SANDWI
<a href="#">Android 4.0, 4.0.1, 4.0.2</a>	14	ICE CREAM SANDWI
<a href="#">Android 3.2</a>	13	HONEYCOMB MR2
<a href="#">Android 3.1.x</a>	12	HONEYCOMB MR1
<a href="#">Android 3.0.x</a>	11	HONEYCOMB
<a href="#">Android 2.3.4</a> <a href="#">Android 2.3.3</a>	10	GINGERBREAD MR1
<a href="#">Android 2.3.2</a> <a href="#">Android 2.3.1</a> <a href="#">Android 2.3</a>	9	GINGERBREAD
<a href="#">Android 2.2.x</a>	8	FROYO
<a href="#">Android 2.1.x</a>	7	ECLAIR MR1

Nov. 2011

Feb 2011

Dic 2010

Mag 2010

17



# project.properties

```
# This file is automatically generated by Android Tools.  
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!  
#  
# This file must be checked in Version Control Systems.  
#  
# To customize properties used by the Ant build system use,  
# "ant.properties", and override values to adapt the script to your  
# project structure.  
  
# Project target.  
target=android-15
```





# Basic tips: having troubles...

Marco Ronchetti  
Università degli Studi di Trento

---

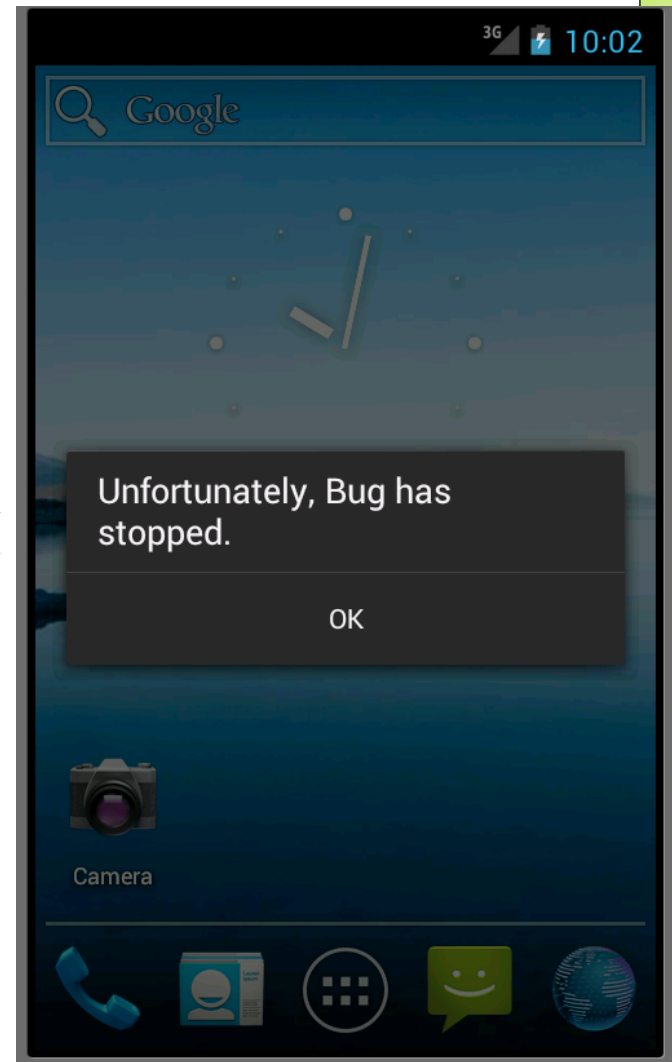
# A bugged program

```
package com.example.helloandroid;

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

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

20







# Basic tips: printing on the console

Marco Ronchetti  
Università degli Studi di Trento

---

# Printing in Eclipse

The screenshot shows the Eclipse IDE with the following components:

- Navigator:** Shows the project structure for 'ButtonActivatedActions', including 'src', 'gen', 'assets', 'bin', 'res', 'layout', 'values', 'AndroidManifest.xml', 'proguard.cfg', and 'project.properties'.
- Editor:** Displays the code for 'Action1.java'. The code includes imports for 'android.app.Activity', 'android.os.Bundle', 'android.view.View', and 'android.widget.Button'. The 'onCreate' method calls 'super.onCreate( savedInstanceState )'. The 'onClickListener' is set on a button, and the 'onClick' method is defined as follows:

```
public void onClick(View v) {  
    System.out.println("CLICK 1 !");  
}
```

A red arrow points from this code block to a callout box:

```
button.setOnClickListener(new View.OnClickListener()  
{  
    public void onClick(View v) {  
        System.out.println("CLICK 1 !");  
    }  
});
```

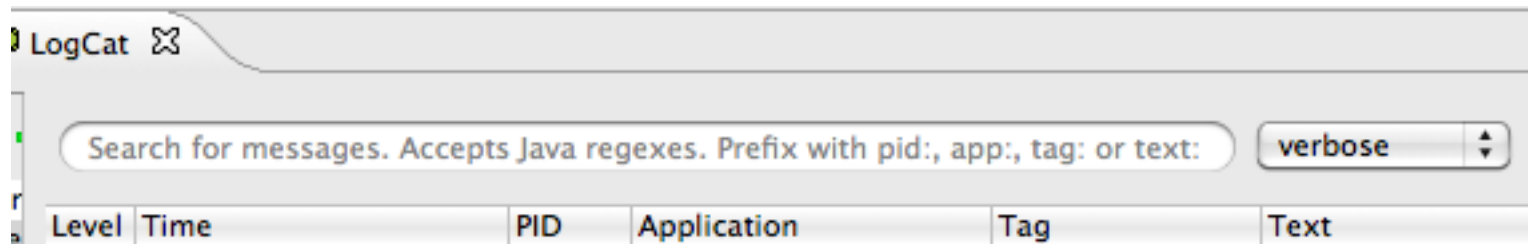
The **Console** window at the bottom right shows the output of the application:

```
60 it.unitt.science... gralloc_gold... Emulator witho  
60 it.unitt.science... System.err CLICK 1 !  
60 it.unitt.science... System.err CLICK 1 !  
006 it.unitt.science... gralloc_gold... Emulator witho  
006 it.unitt.science... System.out CLICK 1 !
```

Red circles highlight the 'CLICK 1 !' messages in the console. A red arrow points from the 'onClick' method in the code to the console output.



# The Logger console



*Log.d("CalledActivity", "OnCreate ");*





# Testing and deploying on your device

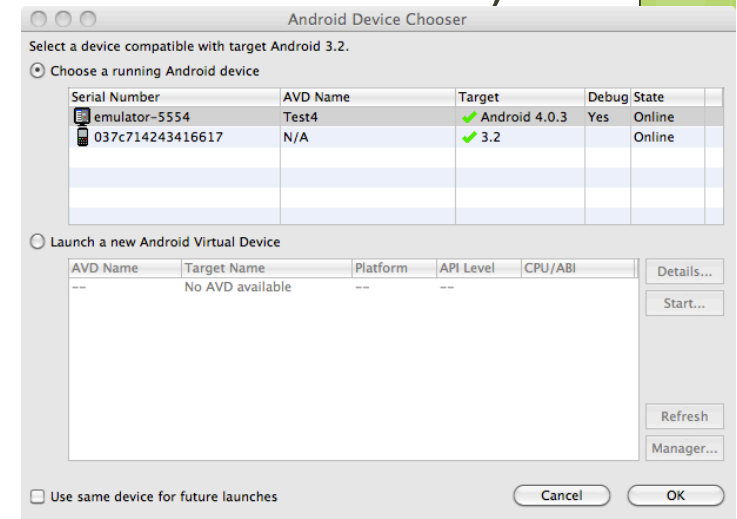
Marco Ronchetti  
Università degli Studi di Trento

---

# Configure device

- 1) **Turn on "USB Debugging"** on your device.  
On the device, go in
  - Android <4: **Settings > Applications > Development**
  - Android >=4: **Settings > Developer options**and enable **USB debugging**
- 2) **Load driver on PC** (win-linux, on Mac not needed)
- 3) Check in shell: **adb devices**
- 4) In Eclipse, you'll have the choice

Make sure the version of OS is correct both in project properties  
And in manifest!



See <http://developer.android.com/guide/developing/device.html>



# Alternative, simple way to deploy

e.g. to give your app to your friends

Get Dropbox both on PC and Android device

Copy your apk from bin/res into dropbox (on PC)

Open dropbox on Android device, and open your apk

By sharing your dropbox with others you can easily pass your app.





# The fundamental components

Marco Ronchetti  
Università degli Studi di Trento

---

# The fundamental components

- **Activity**
  - an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.
- **Fragment** (since 3.0)
  - a behavior or a portion of user interface in an Activity
- **View**
  - equivalent to Swing Component
- **Service**
  - an application component that can perform long-running operations in the background and does not provide a user interface
- **Intent**
  - a passive data structure holding an abstract description of an operation to be performed. It activates an activity or a service. It can also be (as often in the case of broadcasts) a description of something that has happened and is being announced.
- **Broadcast receiver**
  - component that enables an application to receive intents that are broadcast by the system or by other applications.
- **Content Provider**
  - component that manages access to a structured set of data.



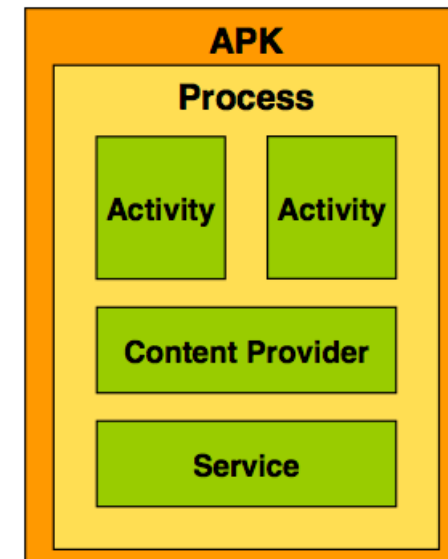


# Peeking into an application

## Packaging: APK File (Android Package)

Collection of components

- Components share a set of resources
  - Preferences, Database, File space
- Components share a Linux process
  - By default, one process per APK
- APKs are isolated
  - Communication via Intents or AIDL (Android Interface Definition Language)
- Every component has a managed lifecycle



29

**ONE APPLICATION, ONE PROCESS, MANY ACTIVITIES**

Slide borrowed from Dominik Gruntz (and modified)

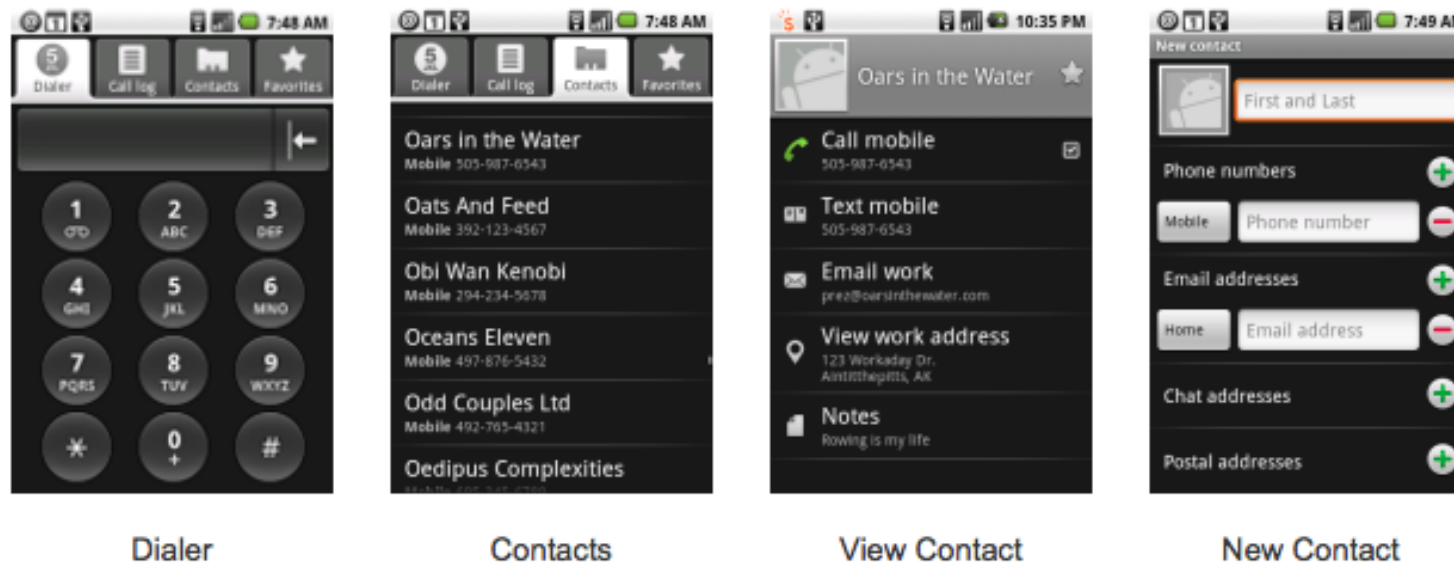


# Activity

An **application component** that provides **a screen with which users can interact in order to do something**, such as dial the phone, take a photo, send an email, or view a map.

Each activity is given a window in which to draw its user interface. The window **typically fills the screen**, but **may be smaller** than the screen and float on top of other windows, or be embedded in another activity (**activityGroup**).

## Activities of the dialer application

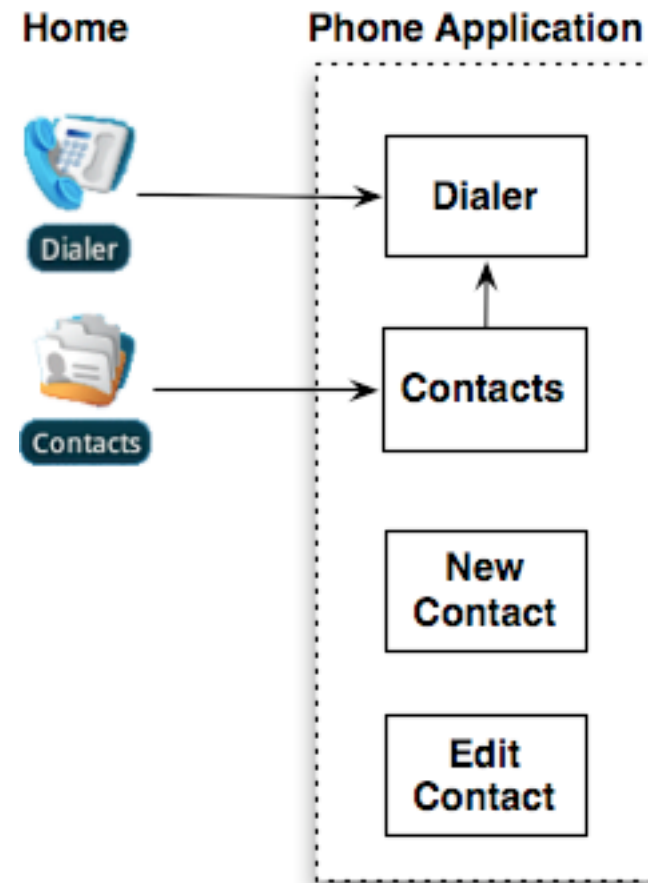


# Multiple entry-point for an app

Typically, **one activity in an application is specified as the "main" activity**, which is presented to the user when launching the application for the first time.

**BUT**

An application can have **multiple entry points**



# Activity

Each activity can **start another activity** in order to perform different actions.

Each time a new activity starts, the previous activity is **stopped**.

The system preserves the activity in a LIFO stack (the **"activity stack"** or **"back stack"**).

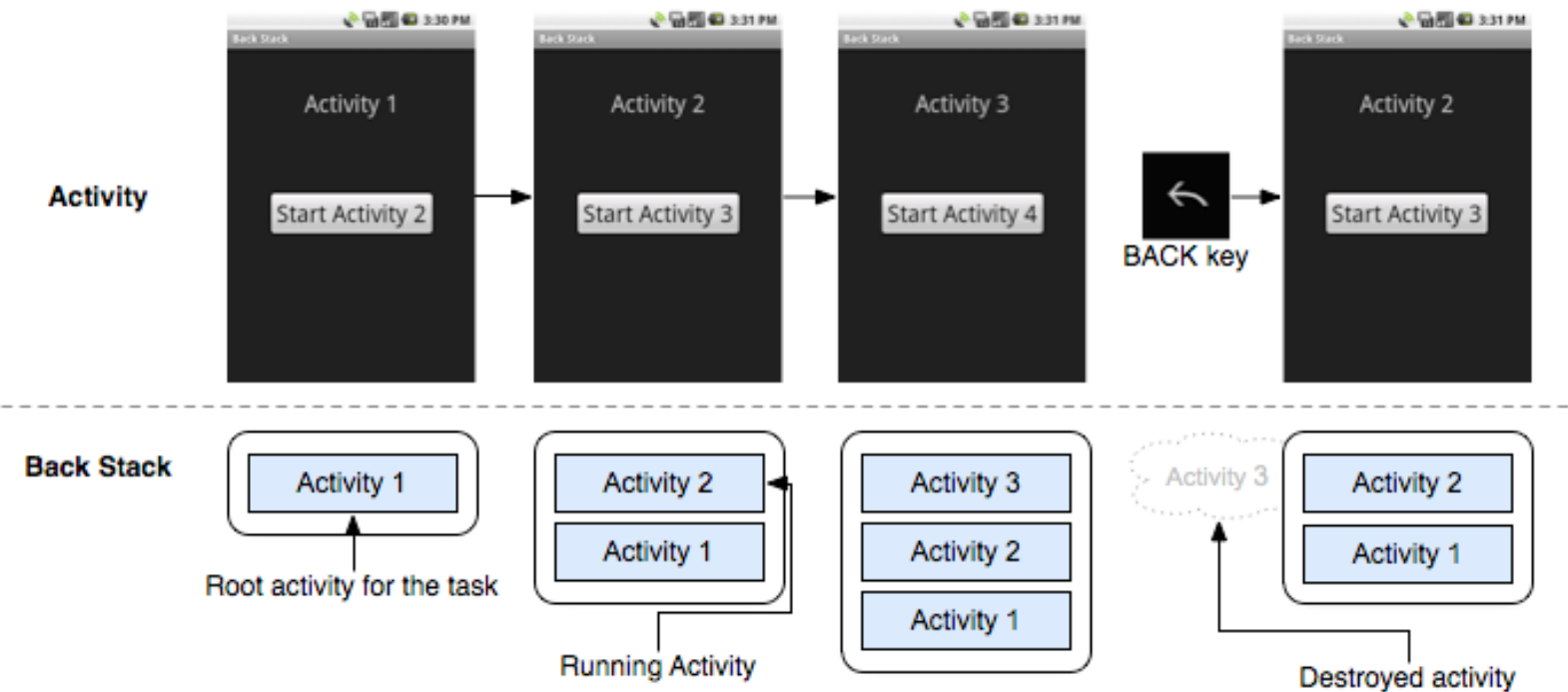
The new activity it is pushed on **top of the back stack** and takes **user focus**.

When the user is done with the current activity and presses the **BACK** button, the current activity is popped from the stack (and **destroyed**) and the **previous activity resumes**.



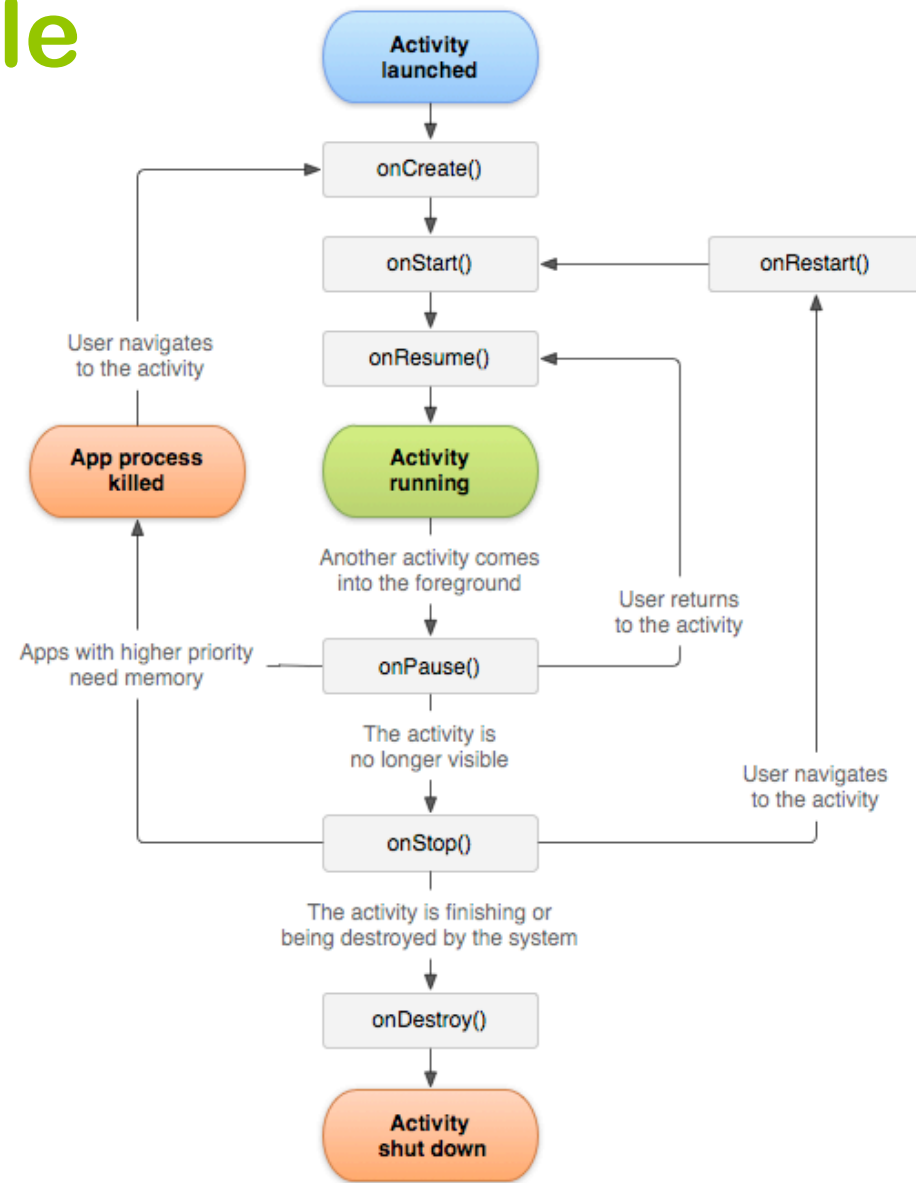
# The activity stack

It's similar to the function stack in ordinary programming, with some difference



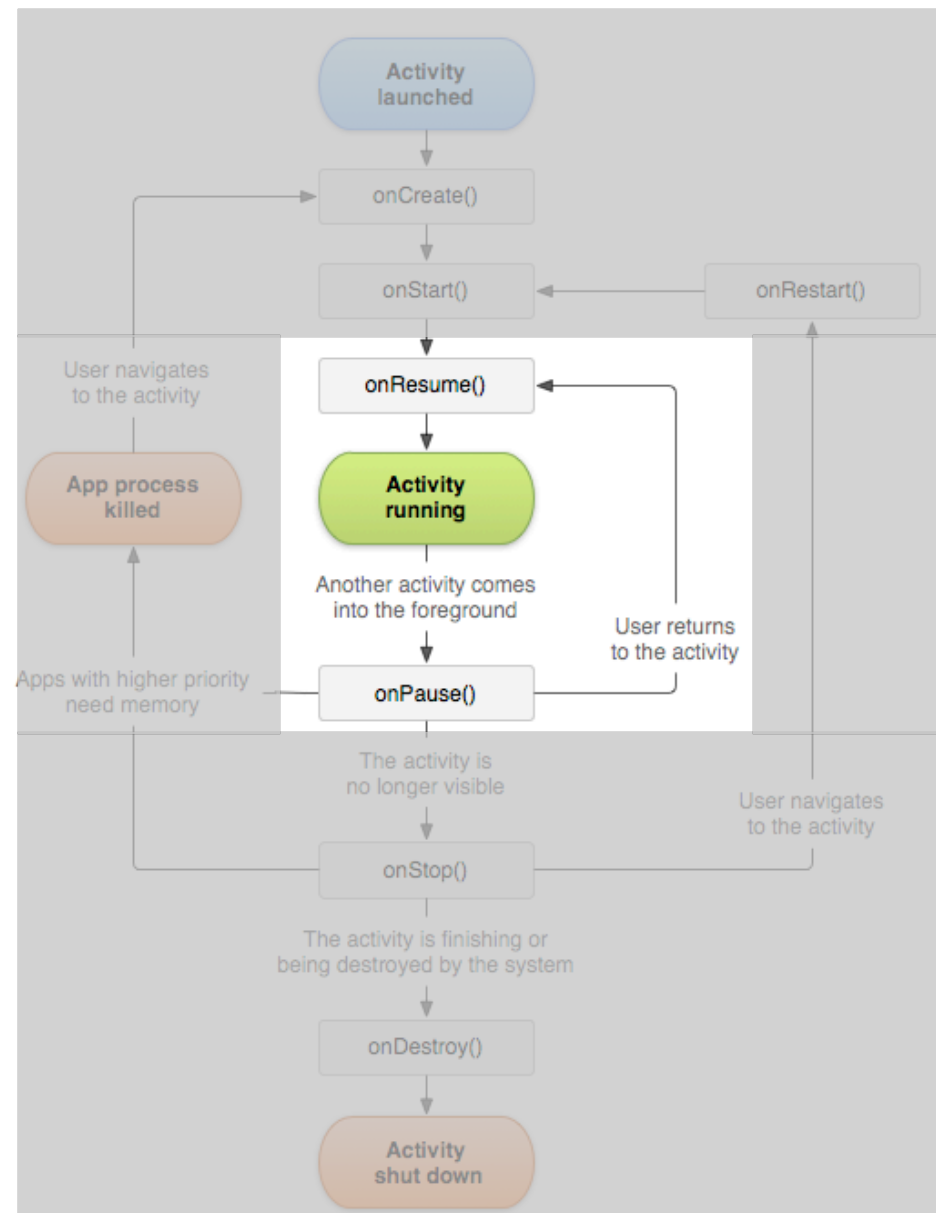
# Activity lifecycle

States (colored),  
and  
Callbacks (gray)



# Activity lifecycle

The FOREGROUND lifetime

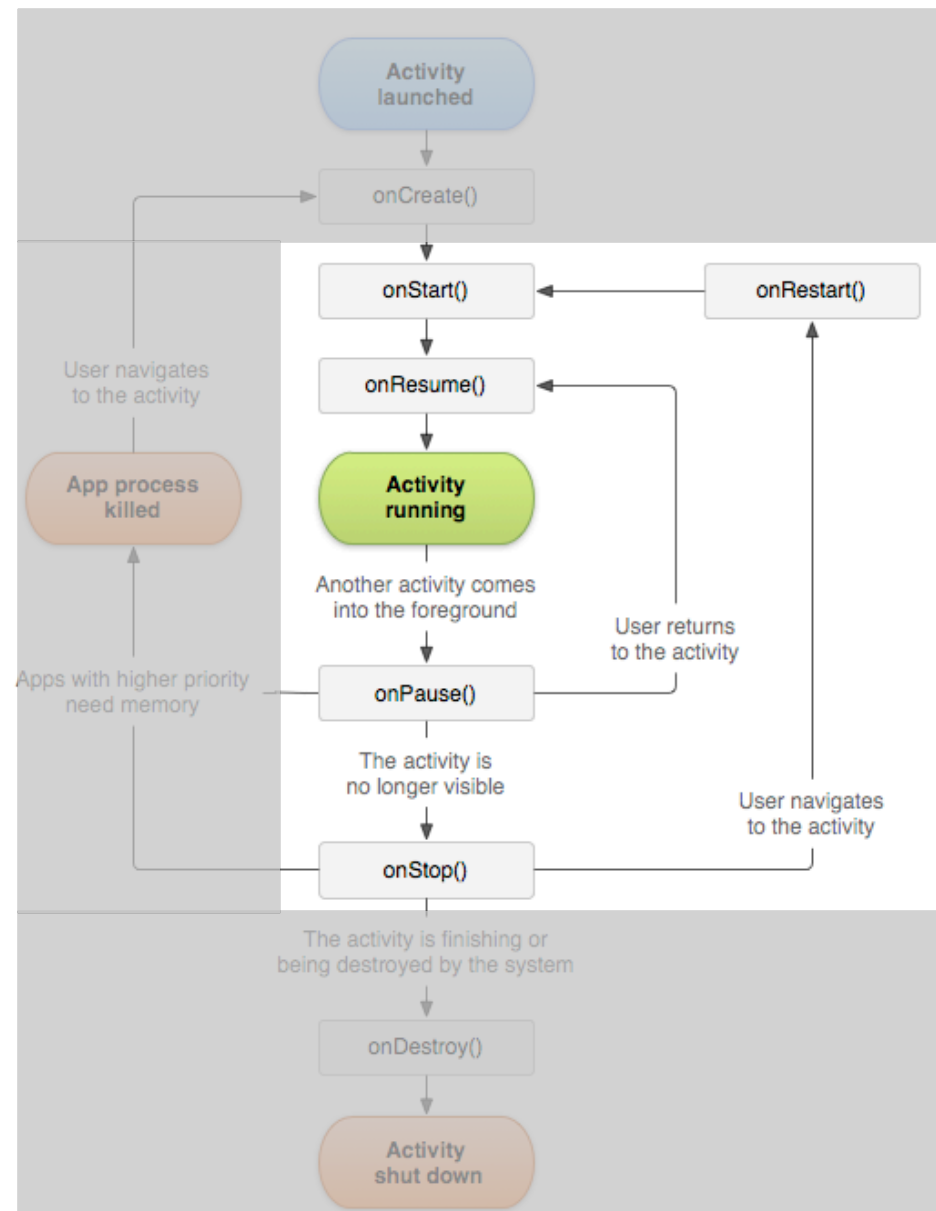


# Activity lifecycle

## The VISIBLE lifetime

When stopped, your activity should release costly resources, such as network or database connections.

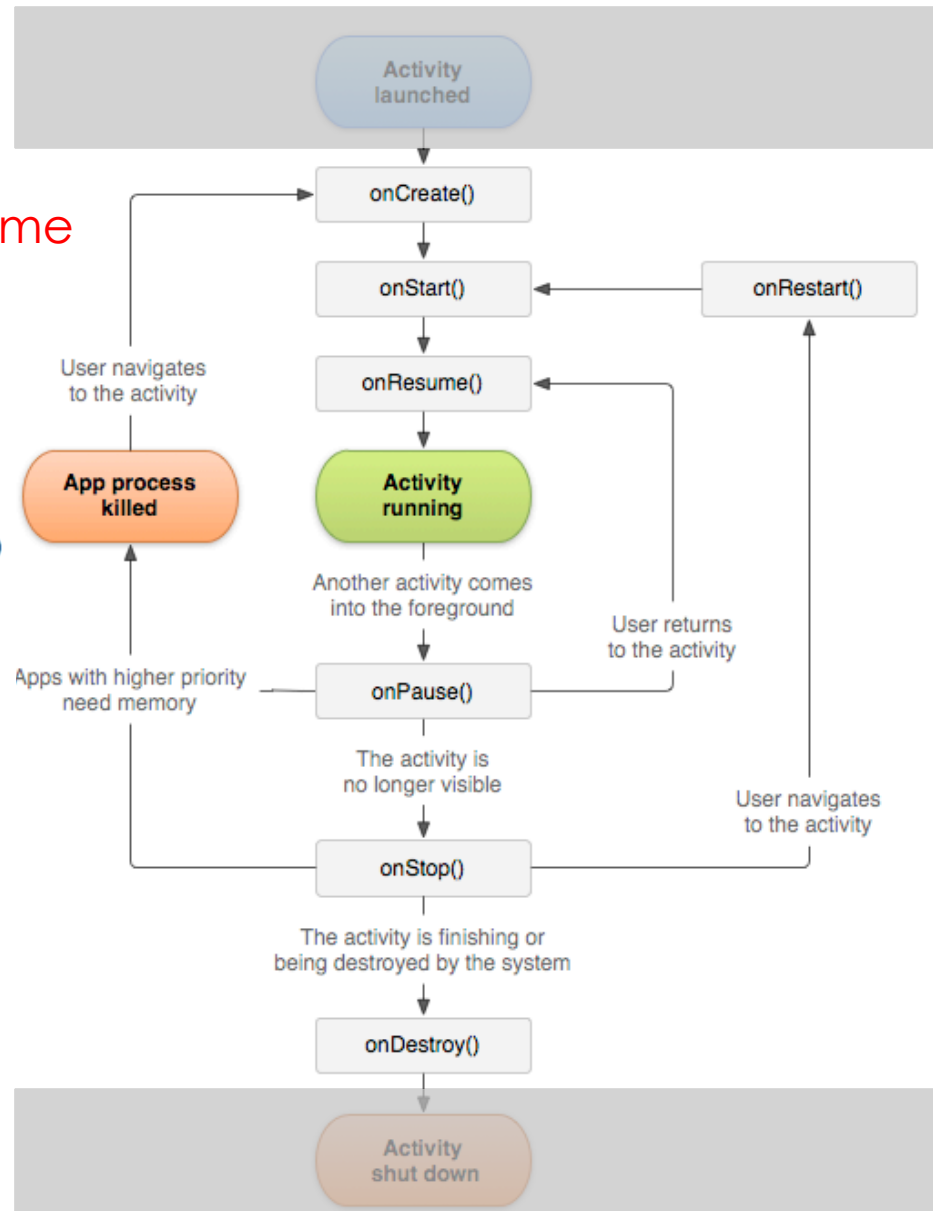
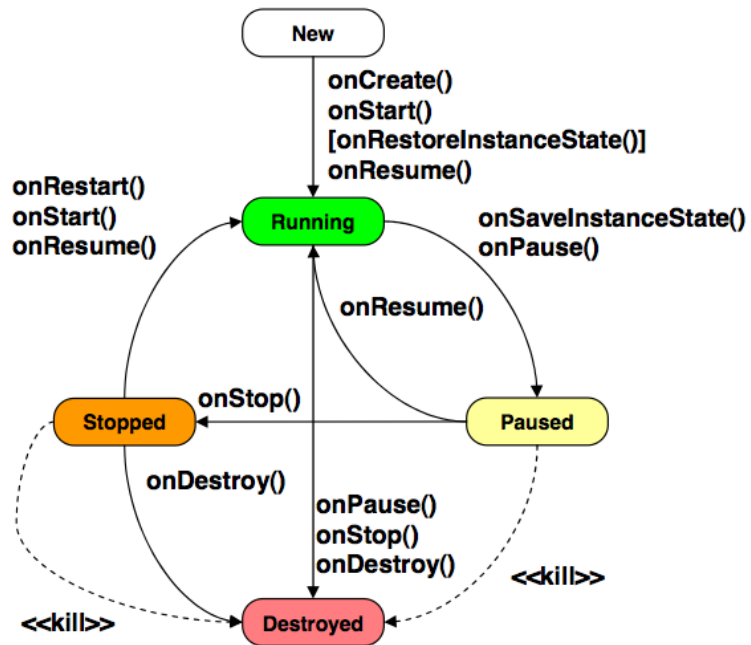
When the activity resumes, you can reacquire the necessary resources and resume actions that were interrupted.





# Activity lifecycle

## The ENTIRE lifetime



37

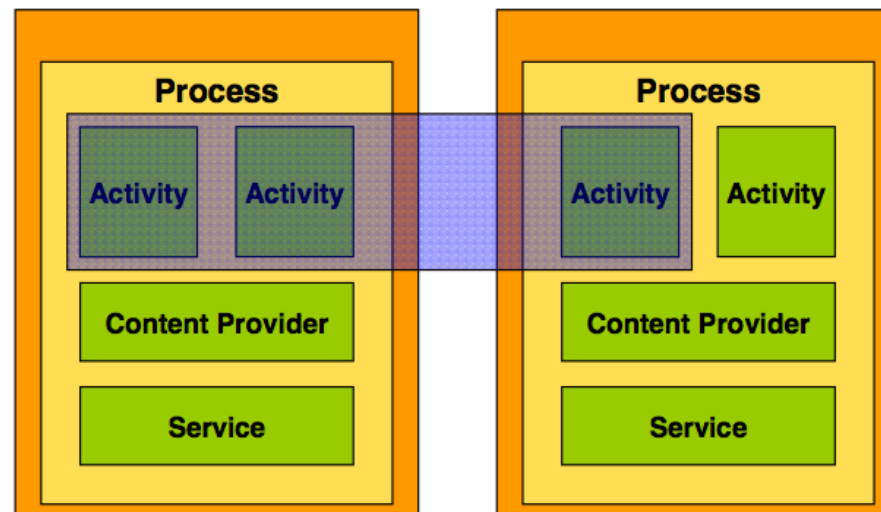


# The shocking news...

An activity can start  
a second activity in  
**a DIFFERENT application!**  
(and hence in a different process...)

We need a name  
for this “thing”:

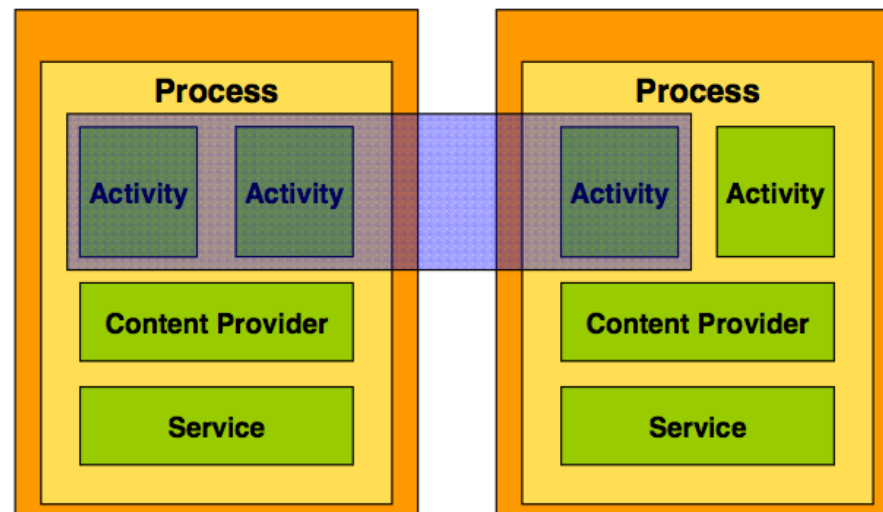
We’ll call it  
“a task”



# Tasks

## Task (**what users view as application**)

- Collection of related activities
- Capable of spanning multiple processes
- Associated with its own UI history stack



# Tasks

An App defines **at least one task**, may define more.

Activities may come from different applications (favoring reuse).

Android maintains a seamless user experience by keeping the activities in the same task.

Tasks may be moved in the **background**.



# Tasks

The **Home screen** is the starting place for most tasks.

When the user touches an icon in the application launcher (or a shortcut on the Home screen), that application's task **comes to the foreground**.

If no task exists for the application (the application has not been used recently), then **a new task is created** and the "main" activity for that application opens as the root activity in the stack.

If the application has been used recently, **its task is resumed** (in general with its state preserved: more on this in the next lecture).



# Switching among apps

To switching among apps:

**long press the home button** and you'll see a window of the 6 most recently used apps.

Tap the app you want to switch to.





# Basic UI elements: Android Buttons (Basics)

Marco Ronchetti  
Università degli Studi di Trento

---

# Let's work with the listener

```
Button button = ...;  
    button.setOnClickListener(new View.OnClickListener() {  
        public void onClick(View v) {  
            Log.d("TRACE", "Button has been clicked ");  
        }  
    });
```

Anonymous  
Inner Class

In Swing it was

JButton button=...

```
button.addActionListener (new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        ...;  
    }  
});
```

44



public class

**Button**

extends [TextView](#)

[java.lang.Object](#)

↳ [android.view.View](#)

↳ [android.widget.TextView](#)

↳ [android.widget.Button](#)

► Known Direct Subclasses

[CompoundButton](#)

► Known Indirect Subclasses

[CheckBox](#), [RadioButton](#), [Switch](#), [ToggleButton](#)



# Let's work with the listener

```
Button button = ...;  
button.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        Log.d("TRACE", "Button has been clicked ");  
    }  
});
```

The event target  
is passed

MAIN DIFFERENCE

In Swing it was

```
JButton button=...
```

```
button.addActionListener (new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        ...;  
    }  
});
```

The event  
is passed



## An alternative

The various View classes expose **several public callback methods** that are useful for UI events.

These methods are called by the Android framework when the respective action occurs on that object. For instance, when a View (such as a Button) is touched, the **onTouchEvent()** method is called on that object.

However, in order to intercept this, **you must extend the class and override the method.**



# Extending Button to deal with events

Class MyButton extends Button {

public boolean onTouchEvent(MotionEvent event) {

int eventaction = event.getAction();

switch (eventaction) {

case MotionEvent.ACTION\_DOWN: // finger touches the screen

...;

break;

case MotionEvent.ACTION\_MOVE: // finger moves on the screen

...;

break;

case MotionEvent.ACTION\_UP: // finger leaves the screen

...;

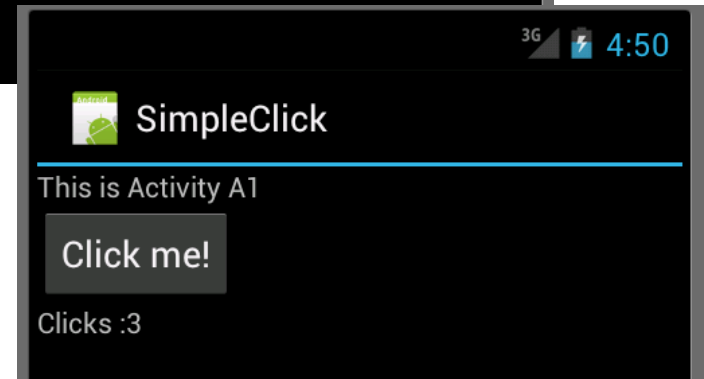
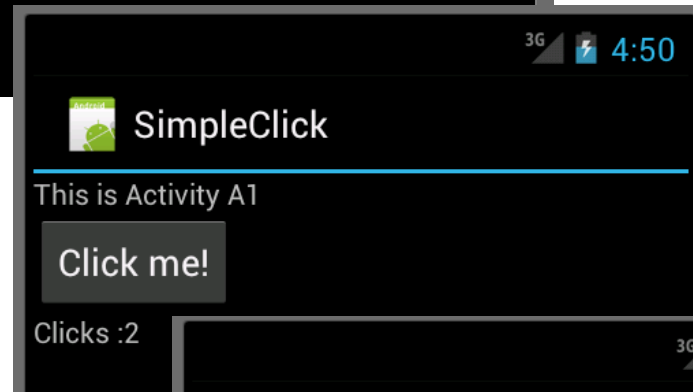
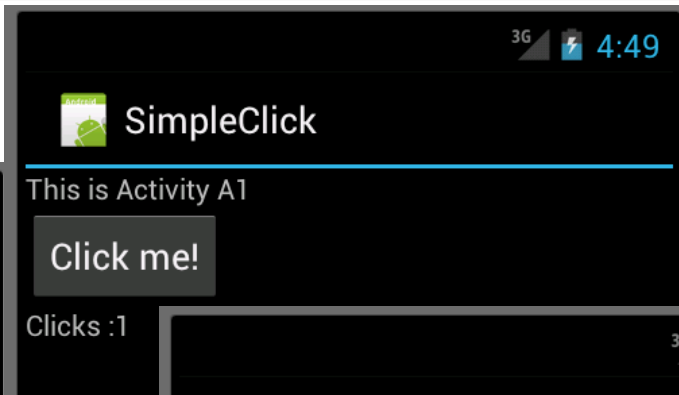
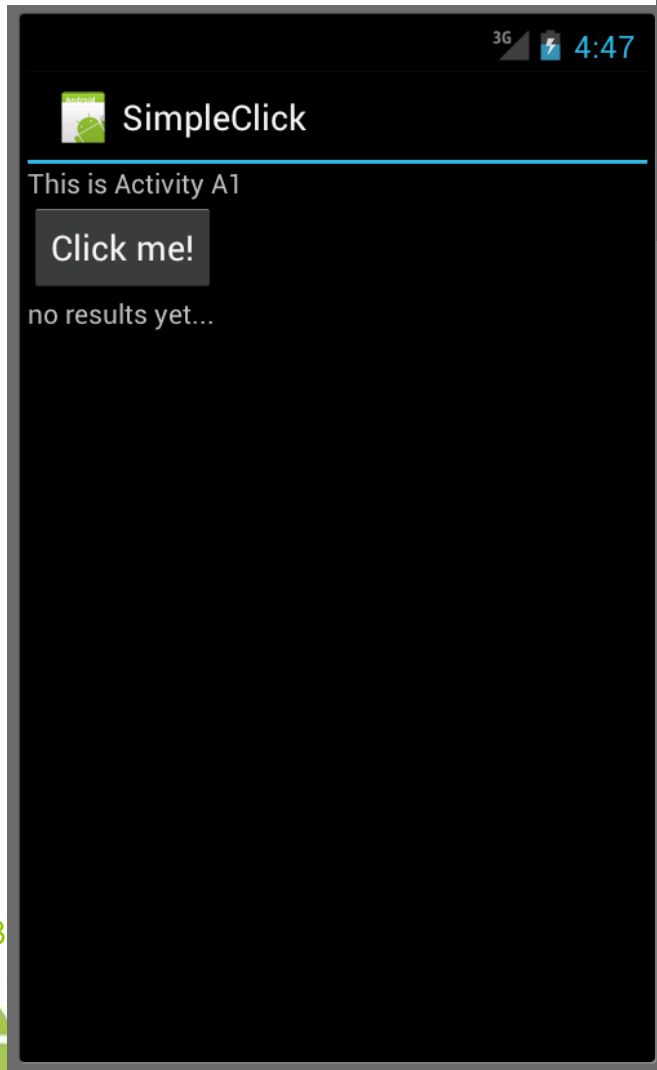
break;

}

47 // tell the system that we handled the event and no further processing is needed  
return true;



# SimpleClick

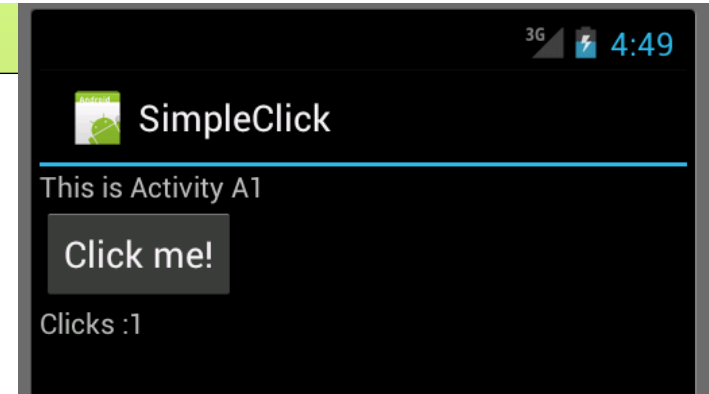
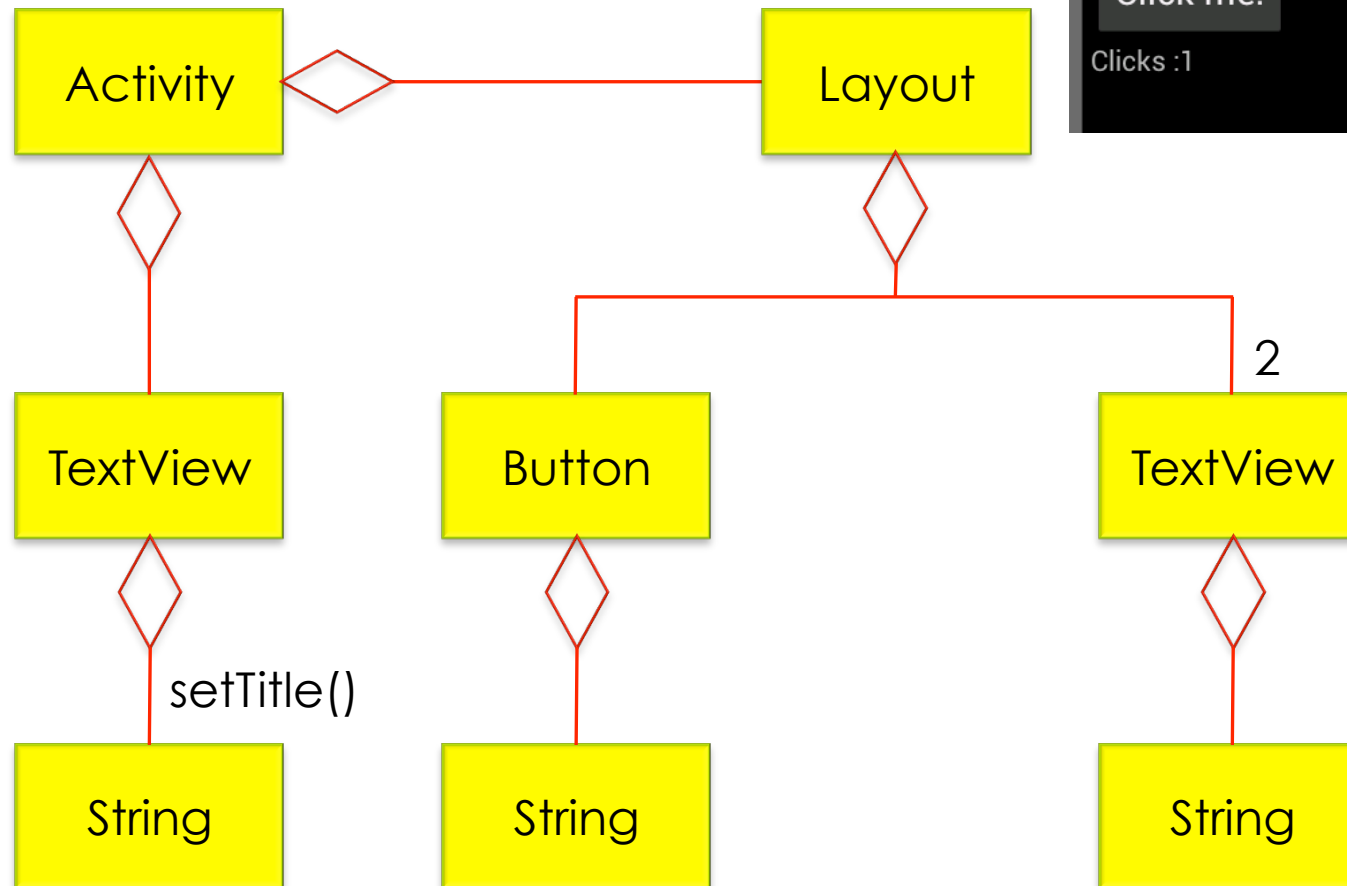


# Let's recap how to build an app

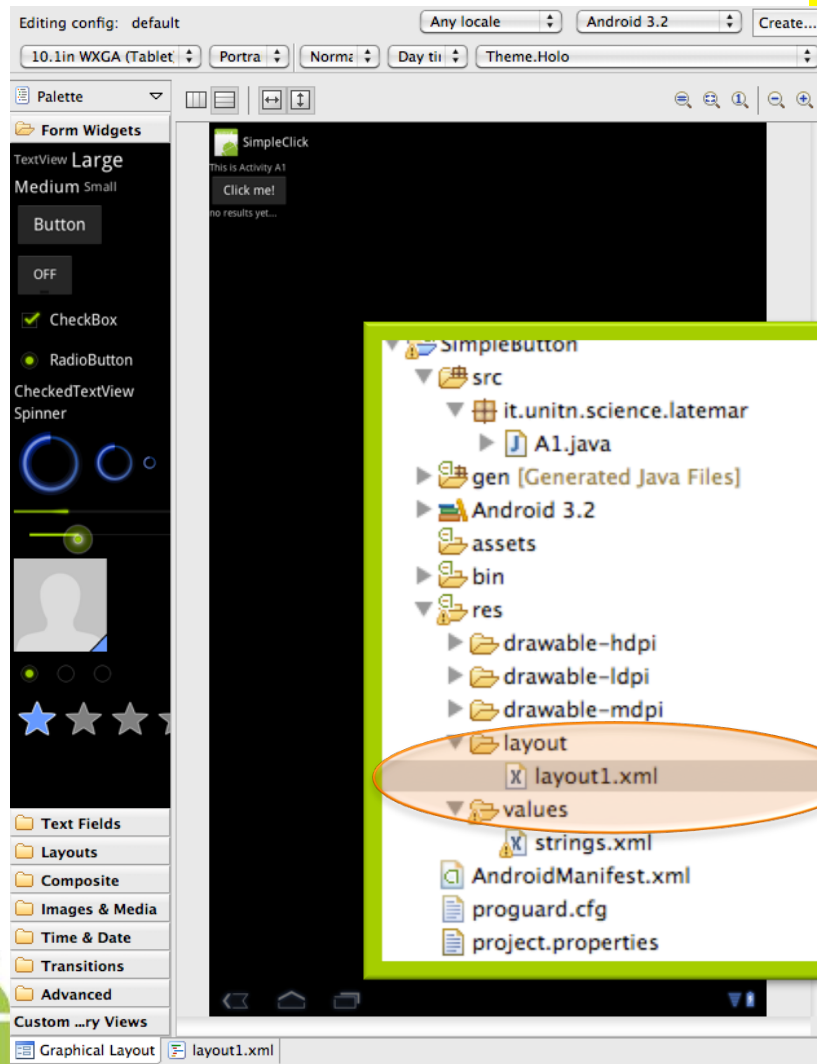
- 1) Define the Activity Resources
  - 1) Choose a Layout
  - 2) Add the components via XML
  - 3) Define the strings
- 2) Code the activity
- 3) Add info to the Manifest (if needed)



# UML Diagram



# Let's define the aspect of layout1



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

```
<LinearLayout xmlns:android=
```

```
    "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```

```
<TextView
```

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

```
<Button
```

```
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button1_label" />
```

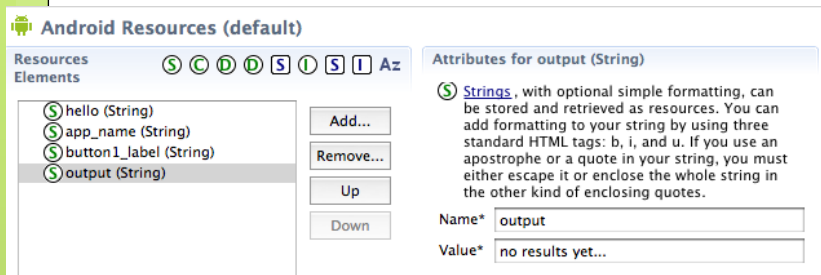
```
<TextView
```

```
    android:id="@+id/tf1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/output" />
```

```
</LinearLayout>
```



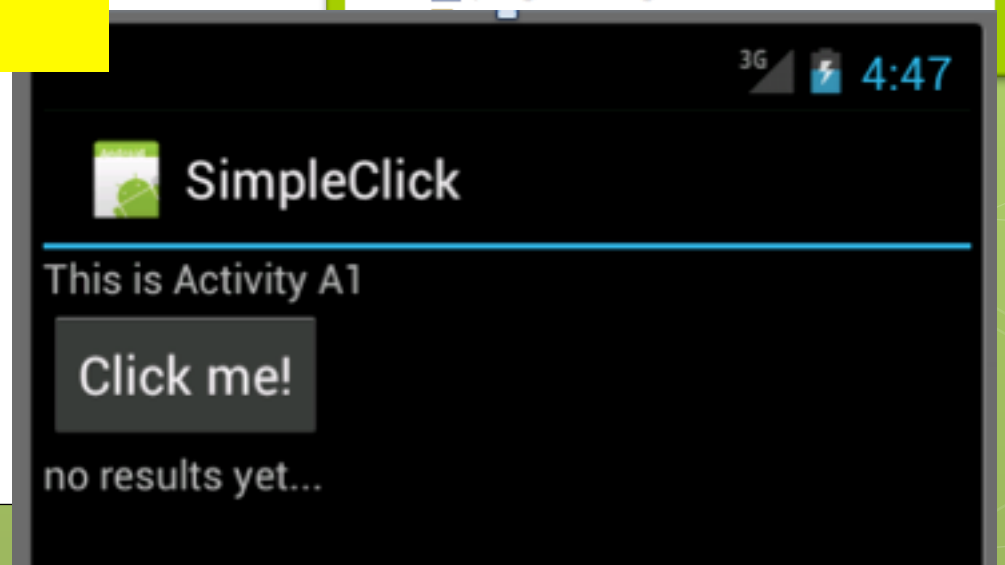
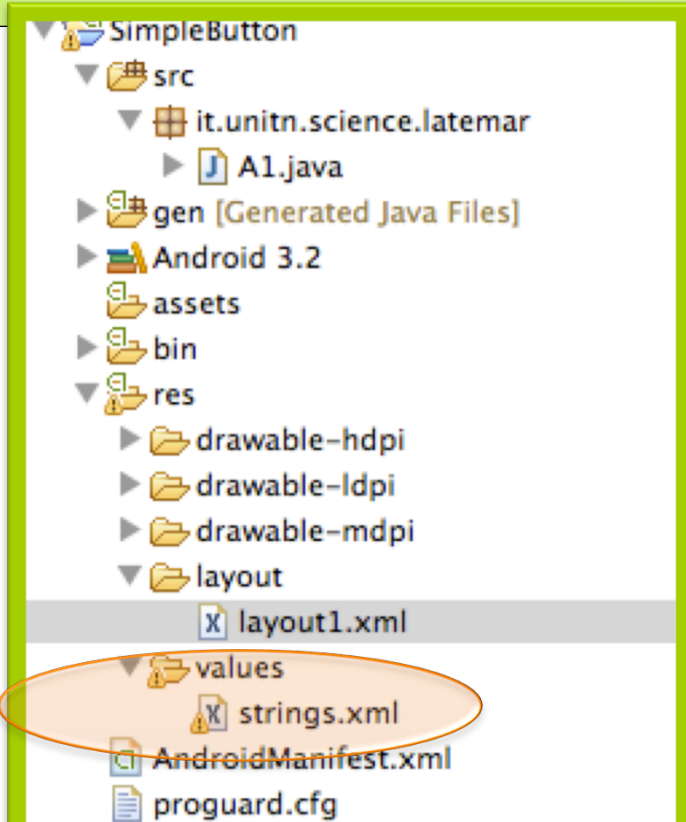
# Let's define the strings



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

    <string name="hello">This is Activity A1</string>
    <string name="app_name">SimpleClick</string>
    <string name="button1_label">Click me!</string>
    <string name="output">no results yet...</string>

</resources>
```





# SimpleClick – A1

```
package it.unitn.science.latemar;

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

public class A1 extends Activity {
    int nClicks=0;
    protected void onCreate(Bundle b) {
        super.onCreate(b);

        setContentView(R.layout.layout1);

        final Button button = (Button) findViewById(R.id.button1);

        final TextView tf = (TextView) findViewById(R.id.tf1);

        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                tf.setText("Clicks :"+(++nClicks));
            }
        });
    }
}
```





# Calling Activities: Explicit Intents

Marco Ronchetti  
Università degli Studi di Trento

---

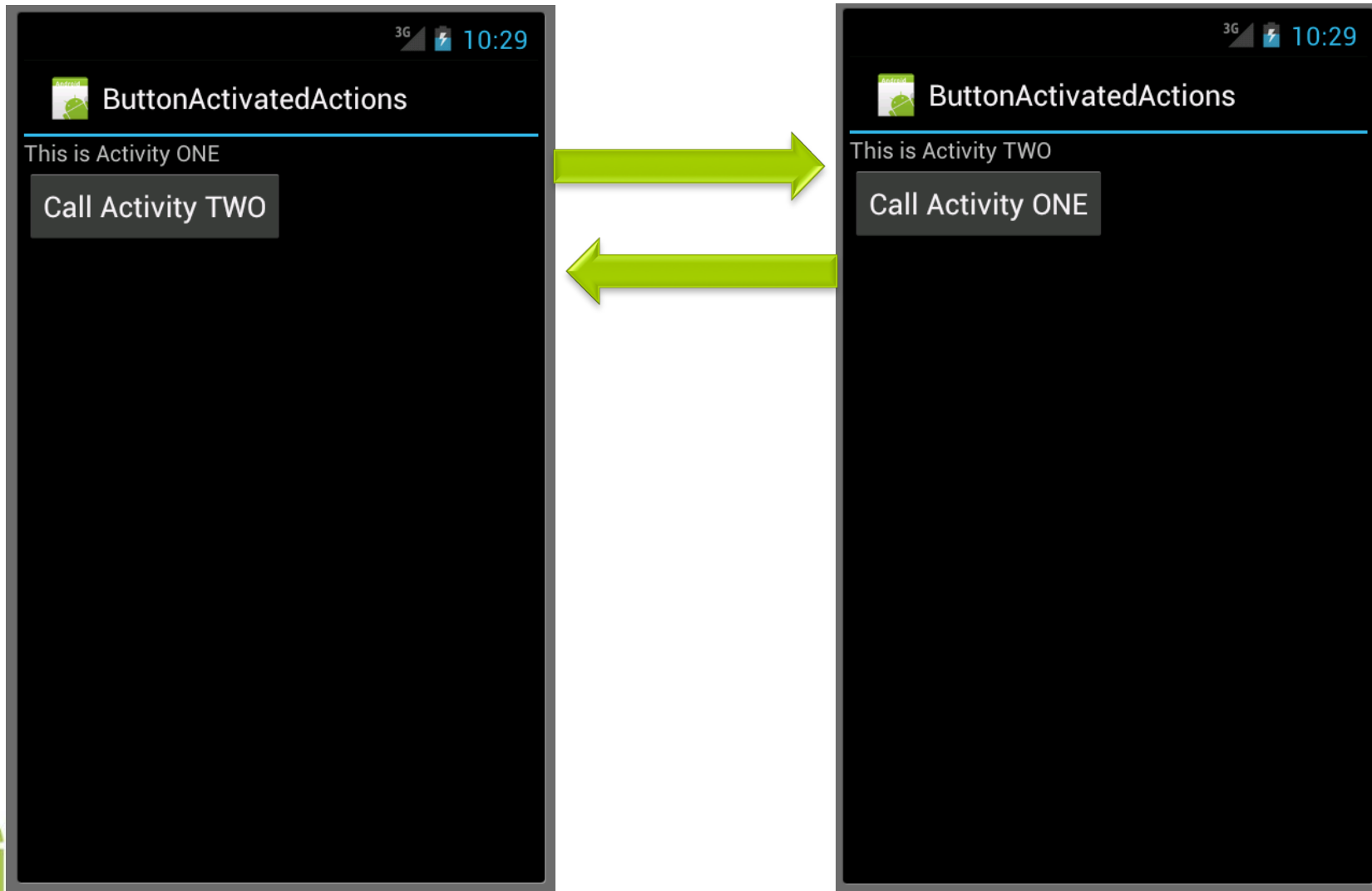
# startActivity(Intent x)

startActivity(Intent x)

- starts a **new activity**, which will be placed at the **top of the activity stack**.
- takes a single argument which describes the activity to be executed.
- An **intent** is an **abstract description of an operation to be performed**.



# A simple example: A1 calls A2



# Explicit Intents

We will use the basic mode:

“Explicit starting an activity”

**Explicit Intents** specify the exact class to be run.

Often these will not include any other information, simply being a way for an application to launch various internal activities it has as the user interacts with the application.



# Intent

The context of the sender



The class to be activated



```
new Intent(Context c, Class c);
```

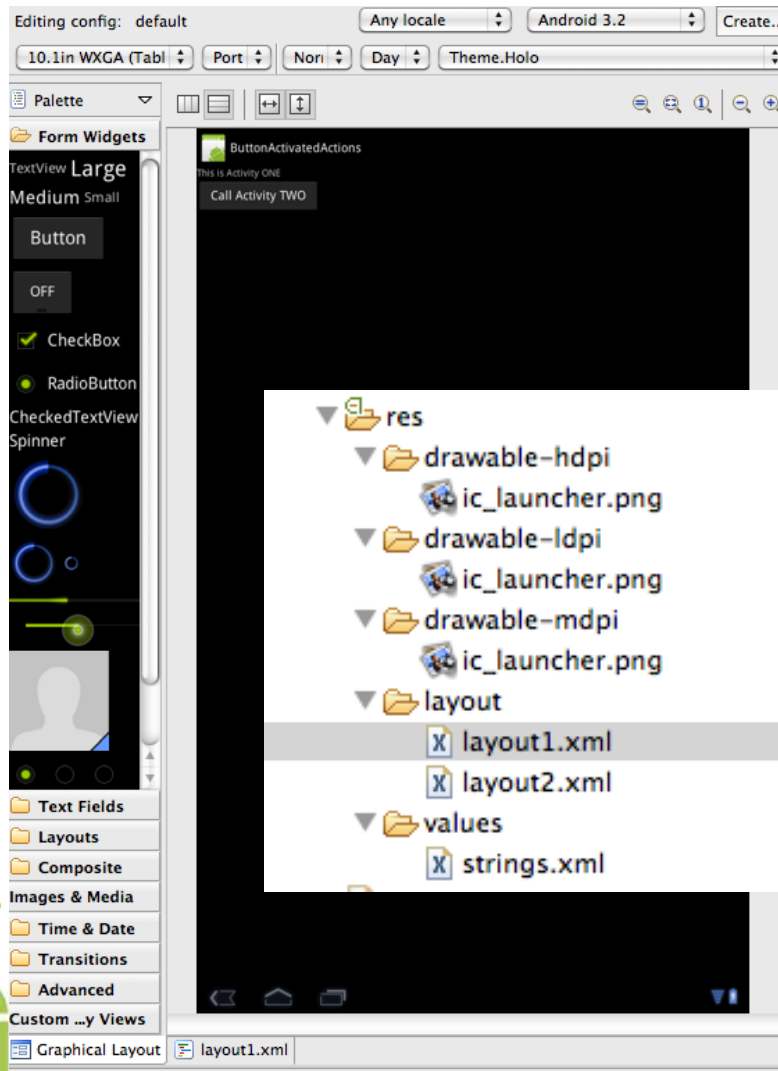
Remember that a Context is a wrapper for global information about an application environment, and that Activity subclasses Context

**Equivalent form:**

```
Intent i=new Intent();  
i.setClass(Context c1, Class c2);
```



# Let's define the aspect of layout1

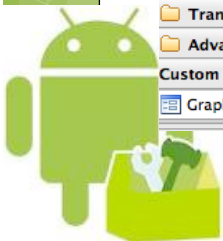


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```

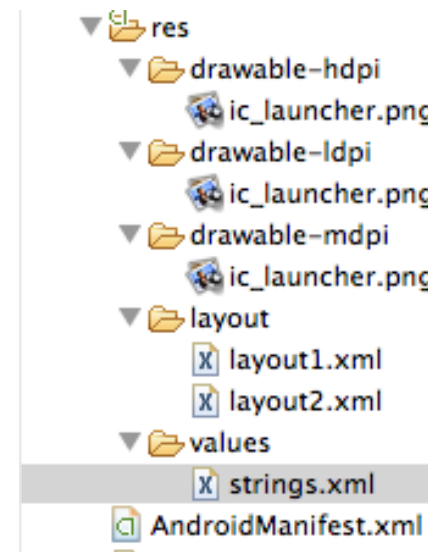
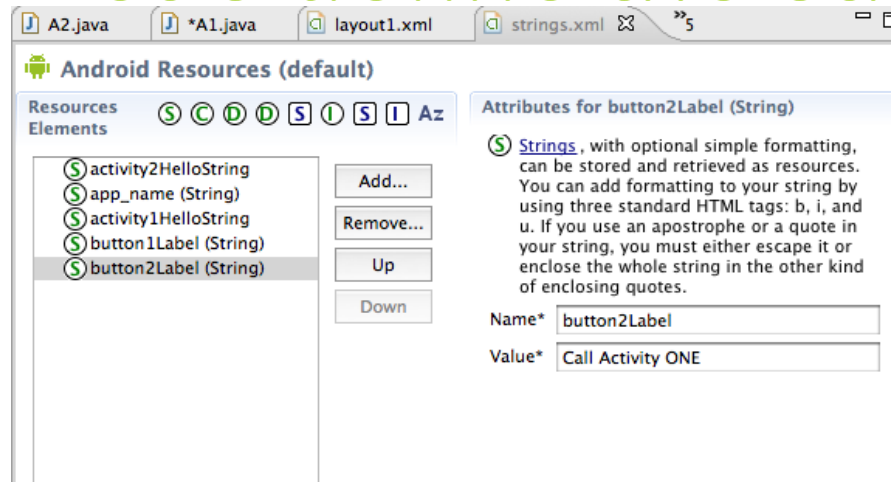
```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/activity1HelloString" />
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button1Label" />
```

```
</LinearLayout>
```



# Let's define the strings



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

```
    <string name="activity2HelloString">This is Activity TWO</string>
    <string name="app_name">ButtonActivatedActions</string>
    <string name="activity1HelloString">This is Activity ONE</string>
    <string name="button1Label">Call Activity TWO</string>
    <string name="button2Label">Call Activity ONE</string>
```

```
</resources>
```





# A1 and A2

```
package it.unitn.science.latemar;

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

public class A1 extends Activity {
    protected void onCreate(Bundle icle) {
        super.onCreate(icle);

        setContentView(R.layout.layout1);
        final Button button = (Button) findViewById(
            R.id.button1);
        button.setOnClickListener(
            new View.OnClickListener() {
                public void onClick(View v) {
                    Intent intent = new Intent(A1.this, A2.class);
                    startActivity(intent);
                }
            });
    }
}
```

Anonymous  
Inner Class

```
package it.unitn.science.latemar;

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

public class A2 extends Activity {
    protected void onCreate(Bundle icle) {
        super.onCreate(icle);

        setContentView(R.layout.layout2);
        final Button button = (Button) findViewById(
            R.id.button2);
        button.setOnClickListener(
            new View.OnClickListener() {
                public void onClick(View v) {
                    Intent intent = new Intent(A1.this, A2.class);
                    startActivity(intent);
                }
            });
    }
}
```



# A1.this ? What's that?

```
button.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        Intent intent = new Intent(A1.this, A2.class);  
        startActivity(intent);  
    }  
});
```

**final Activity me=this;**

```
button.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        Intent intent = new Intent(me, A2.class);  
        startActivity(intent);  
    }  
});
```

```
final Intent intent = new Intent(this, A2.class);  
button.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        startActivity(intent);  
    }  
});
```



# Let's declare A2 in the manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.unitn.science.latemar"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="13" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name="A1"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="A2"></activity>
    </application>

</manifest>
```

63



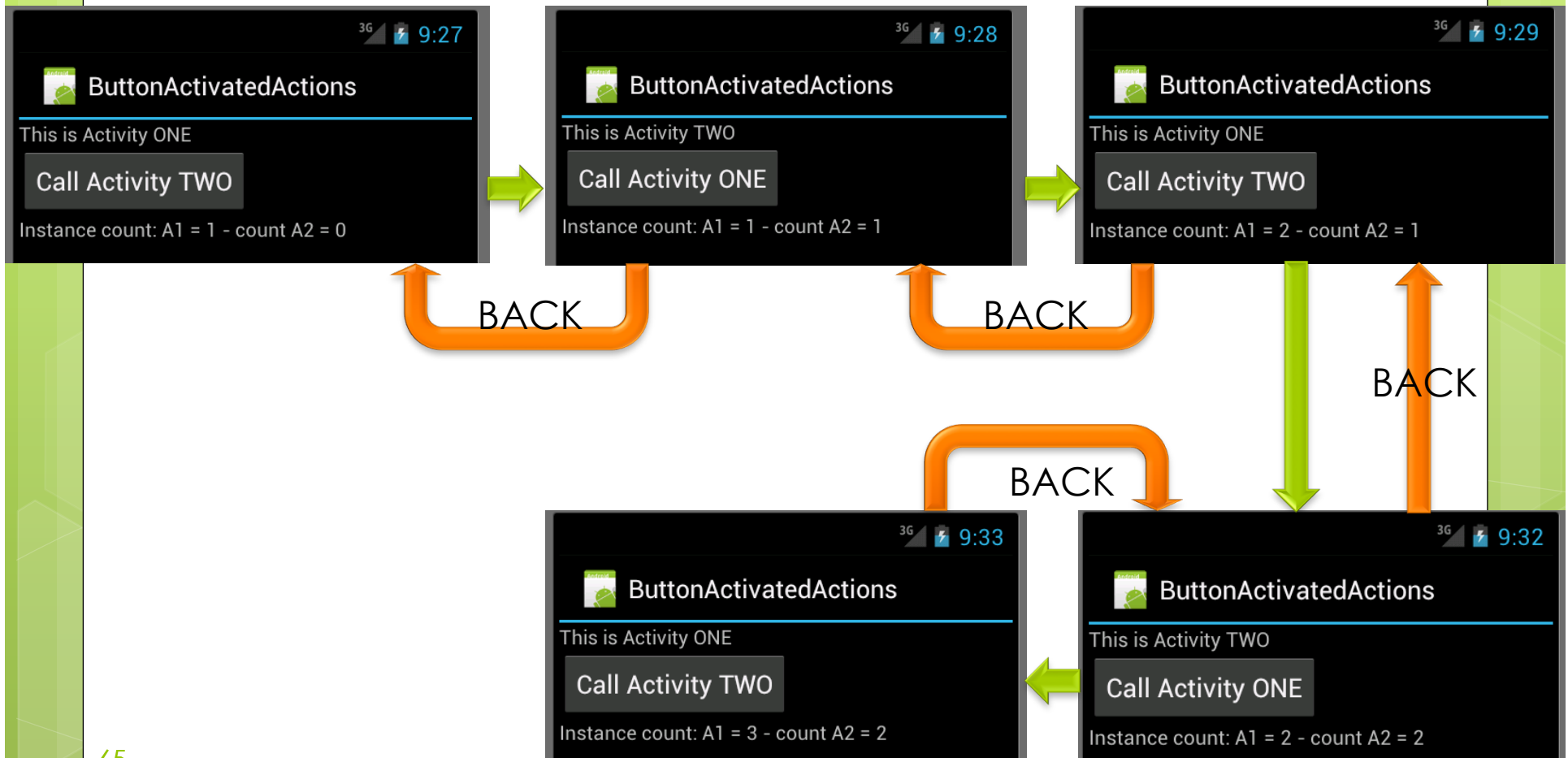


# How many instances?

Marco Ronchetti  
Università degli Studi di Trento

---

# How many instances?



65



## The code

```
public class A1 extends Activity {  
    static int instances = 0;  
    TextView tf = null;  
    protected void onCreate(Bundle icle) {  
        super.onCreate(icle);  
        instances++;  
        setContentView(R.layout.layout1);  
        tf = (TextView) findViewById(R.id.instanceCount);  
        final Button button = (Button) findViewById(R.id.button1);  
        final Intent intent = new Intent(this, A2.class);  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                startActivity(intent);  
            }  
        });  
    }  
}
```

```
    protected void onDestroy() {  
        super.onDestroy();  
        instances--;  
    }  
    protected void onResume() {  
        super.onResume();  
        if (tf != null)  
            tf.setText("Instance count: A1 = " +  
                A1.instances+" - count A2 = " +  
                A2.instances);  
    }  
}
```



# The xml

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/activity1HelloString" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button1Label" />

    <TextView
        android:id="@+id/instanceCount"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/instanceCount" />

</LinearLayout>
```

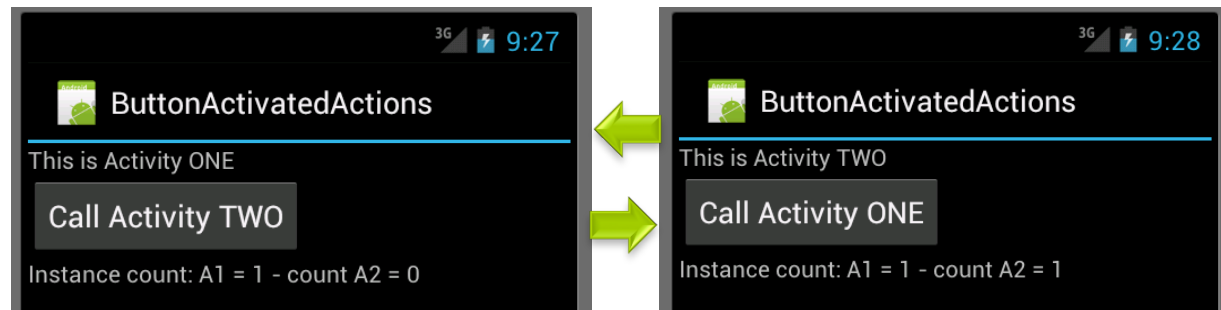
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="activity2HelloString">
        This is Activity TWO</string>
    <string name="app_name">
        ButtonActivatedActions</string>
    <string name="activity1HelloString">
        This is Activity ONE</string>
    <string name="button1Label">
        Call Activity TWO</string>
    <string name="button2Label">
        Call Activity ONE</string>
    <string name="instanceCount">
        Instance count: field not initialized</string>
    <string name="instanceCount2">
        Instance count: field not initialized</string>
</resources>
```



# Minimizing instances

```
protected void onCreate(Bundle icle) {  
    super.onCreate(icle);  
    instances++;  
    setContentView(R.layout.layout2);  
    tf2 = (TextView) findViewById(R.id.instanceCount2);  
    final Button button = (Button) findViewById(R.id.button2);  
    final Intent intent = new Intent(this, A1.class);  
    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);  
    button.setOnClickListener(new View.OnClickListener() {  
        public void onClick(View v) {  
            startActivity(intent);  
        }  
    });  
}
```

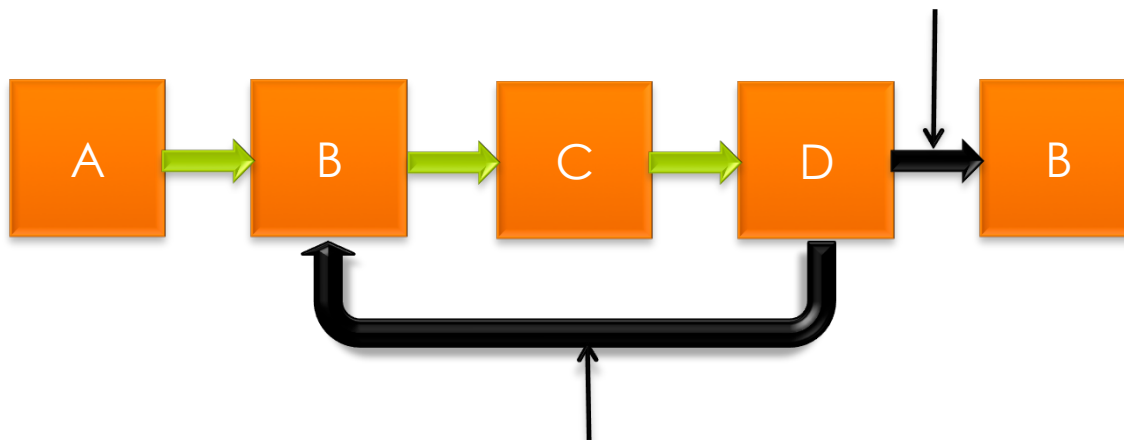
68





# FLAG\_ACTIVITY\_CLEAR\_TOP

senza **FLAG\_ACTIVITY\_CLEAR\_TOP**



con **FLAG\_ACTIVITY\_CLEAR\_TOP**  
(C e D vengono distrutte)



For details see [http://developer.android.com/reference/android/content/Intent.html#FLAG\\_ACTIVITY\\_BROUGHT\\_TO\\_FRONT](http://developer.android.com/reference/android/content/Intent.html#FLAG_ACTIVITY_BROUGHT_TO_FRONT)

# FLAGS

int	FLAG_ACTIVITY_BROUGHT_TO_FRONT	This flag is not normally set by application code, but set for you by the system as described in the <a href="#">launch</a>
int	FLAG_ACTIVITY_CLEAR_TASK	If set in an Intent passed to <code>Context.startActivity()</code> , this flag will cause any existing task that wo
int	FLAG_ACTIVITY_CLEAR_TOP	If set, and the activity being launched is already running in the current task, then instead of launching a r delivered to the (now on top) old activity as a new Intent.
int	FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET	If set, this marks a point in the task's activity stack that should be cleared when the task is reset.
int	FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS	If set, the new activity is not kept in the list of recently launched activities.
int	FLAG_ACTIVITY_FORWARD_RESULT	If set and this intent is being used to launch a new activity from an existing one, then the reply target of t
int	FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY	This flag is not normally set by application code, but set for you by the system if this activity is being laun
int	FLAG_ACTIVITY_MULTIPLE_TASK	<b>Do not use this flag unless you are implementing your own top-level application launcher.</b>
int	FLAG_ACTIVITY_NEW_TASK	If set, this activity will become the start of a new task on this history stack.
int	FLAG_ACTIVITY_NO_ANIMATION	If set in an Intent passed to <code>Context.startActivity()</code> , this flag will prevent the system from applyir
int	FLAG_ACTIVITY_NO_HISTORY	If set, the new activity is not kept in the history stack.
int	FLAG_ACTIVITY_NO_USER_ACTION	If set, this flag will prevent the normal <code>onUserLeaveHint()</code> callback from occurring on the current fron
int	FLAG_ACTIVITY_PREVIOUS_IS_TOP	If set and this intent is being used to launch a new activity from an existing one, the current activity will n of starting a new one.
int	FLAG_ACTIVITY_REORDER_TO_FRONT	If set in an Intent passed to <code>Context.startActivity()</code> , this flag will cause the launched activity to b
int	FLAG_ACTIVITY_RESET_TASK_IF_NEEDED	If set, and this activity is either being started in a new task or bringing to the top an existing task, then it v
int	FLAG_ACTIVITY_SINGLE_TOP	If set, the activity will not be launched if it is already running at the top of the history stack.
int	FLAG_ACTIVITY_TASK_ON_HOME	If set in an Intent passed to <code>Context.startActivity()</code> , this flag will cause a newly launching task to
int	FLAG_DEBUG_LOG_RESOLUTION	A flag you can enable for debugging: when set, log messages will be printed during the resolution of this
int	FLAG_EXCLUDE_STOPPED_PACKAGES	If set, this intent will not match any components in packages that are currently stopped.
int	FLAG_FROM_BACKGROUND	Can be set by the caller to indicate that this Intent is coming from a background operation, not from direc
int	FLAG_GRANT_READ_URI_PERMISSION	If set, the recipient of this Intent will be granted permission to perform read operations on the Uri in the I
int	FLAG_GRANT_WRITE_URI_PERMISSION	If set, the recipient of this Intent will be granted permission to perform write operations on the Uri in the I
int	FLAG_INCLUDE_STOPPED_PACKAGES	If set, this intent will always match any components in packages that are currently stopped.
int	FLAG_RECEIVER_REGISTERED_ONLY	If set, when sending a broadcast only registered receivers will be called -- no BroadcastReceiver compo
int	FLAG_RECEIVER_REPLACE_PENDING	If set, when sending a broadcast the new broadcast will replace any existing pending broadcast that mat



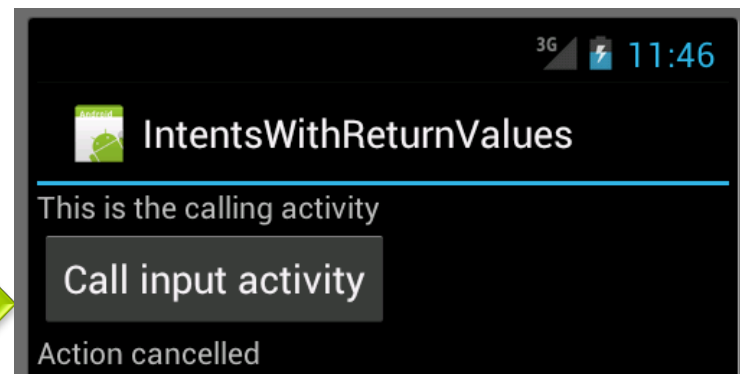
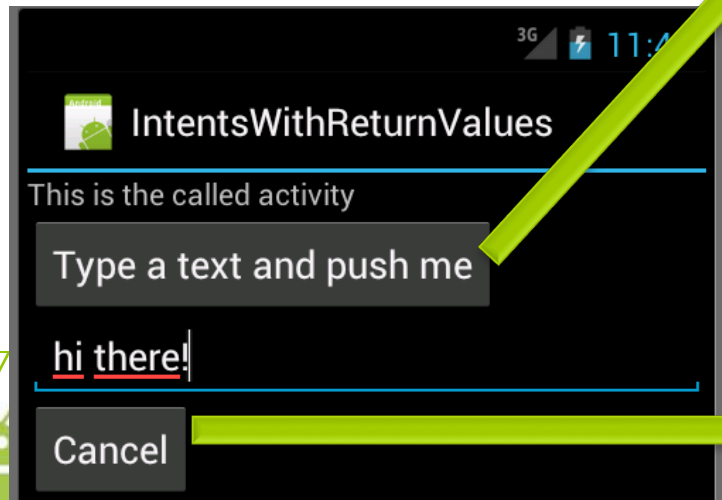
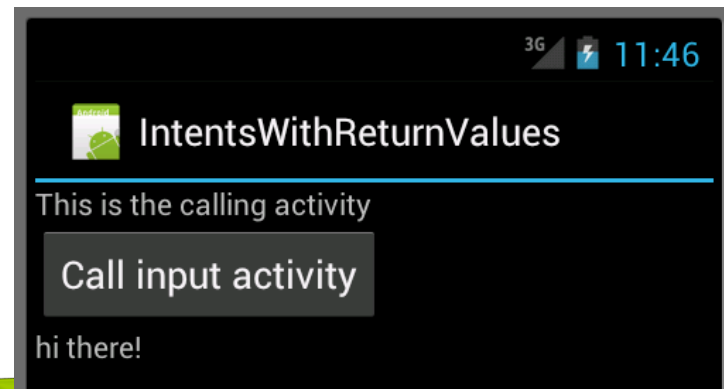
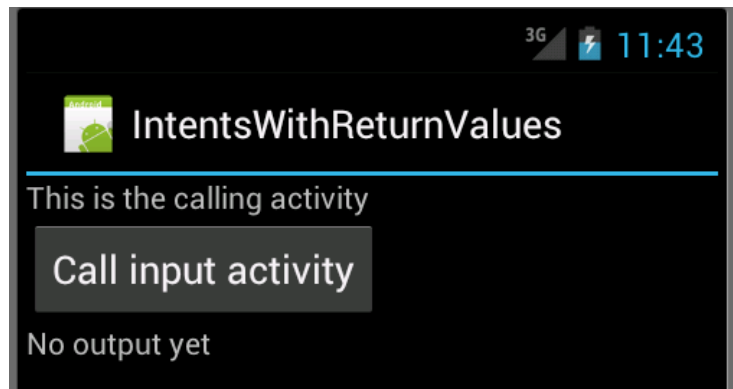


# Calling Activities with return values: `startActivityForResult`

Marco Ronchetti  
Università degli Studi di Trento

---

# Returning data

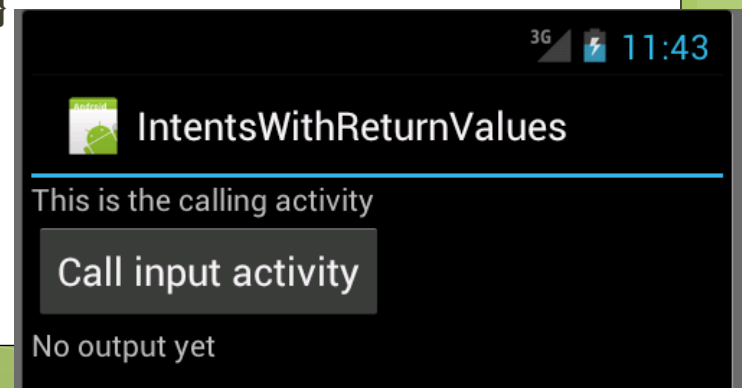


# Calling activity

```
public class CallingActivity extends Activity {  
    int requestCode=100;  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.layout1);  
        final Intent i = new Intent(this,CalledActivity.class);  
        final Button button = (Button) findViewById(R.id.invokebutton);  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) { startActivityForResult(i, requestCode); }  
        });  
    }  
  
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
        super.onActivityResult(requestCode, resultCode, data);  
        final TextView tf = (TextView) findViewById(R.id.output);  
        if (resultCode==1){  
            tf.setText(data.getStringExtra("payload")); }  
        else{  
            tf.setText("Action cancelled");  
        }  
    }  
}
```

```
package it.unitn.science.latemar;  
  
import ...
```

The name should be  
qualified with the package



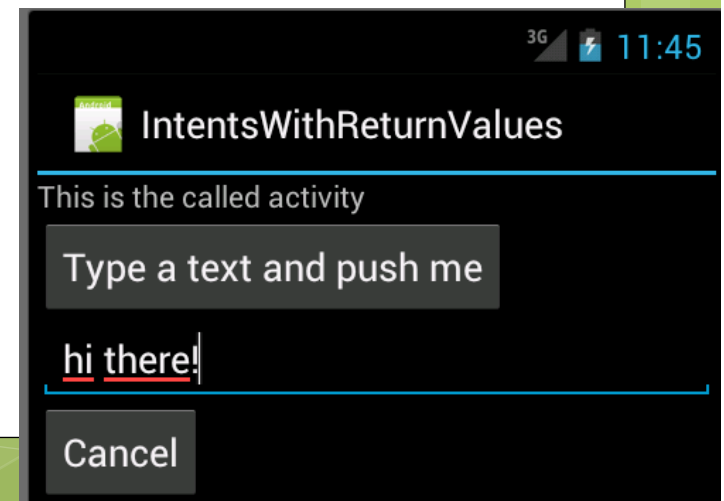
# Called activity

```
package it.unitn.science.latemar;  
  
import ...
```

```
public class CalledActivity extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.layout2);  
        final Intent in = new Intent();  
        final Button button = (Button) findViewById(R.id.OKbutton);  
        final EditText tf = (EditText) findViewById(R.id.editText1);  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                setResult(1,in);  
                in.putExtra("payload", tf.getText().toString());  
                finish();  
            }  
        });  
        final Button button_c = (Button) findViewById(R.id.cancelButton);  
        button_c.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                setResult(0,in);  
                finish();  
            }  
        });  
    }  
}
```

The name should be qualified with the package

74



# Remember to register the Activity!

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.unitn.science.latemar"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="13" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".CallingActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="CalledActivity"></activity>
    </application>
</manifest>
```

75



# Extras

`setExtra(String name, #TYPE# payload)`

where #TYPE# = one of

- Primitive data types
- String or various types of serializable objects

`get...Extra(String name)` where ... is the name of the data types available in getters

`removeExtra(String name)`

`replaceExtra` completely replace the Extras







# Basic UI elements: Views and Layouts, a deeper insight

Marco Ronchetti  
Università degli Studi di Trento

---

# Widgets examples

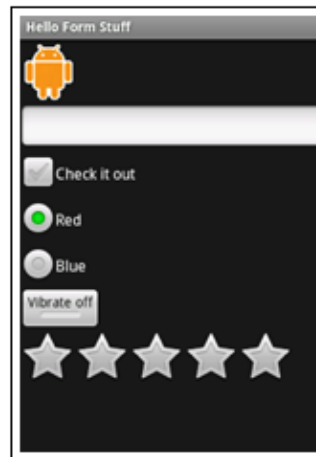
Date Picker



Time Picker



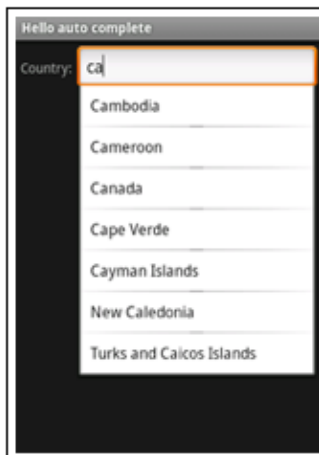
Form Stuff



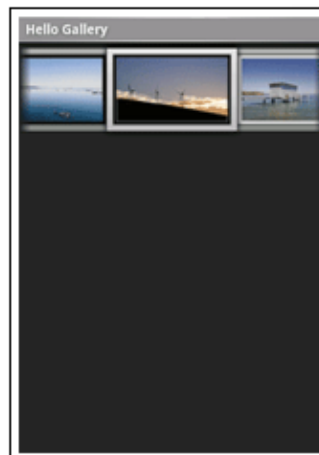
Spinner



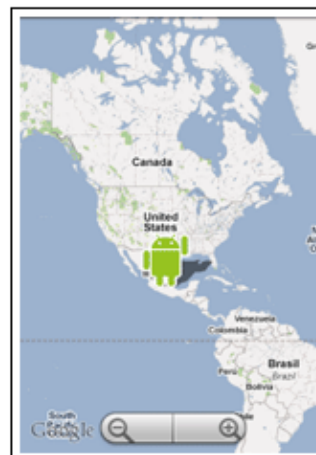
Auto Complete



Gallery



Google Map View



Web View



# Layout examples

Linear Layout



Relative Layout

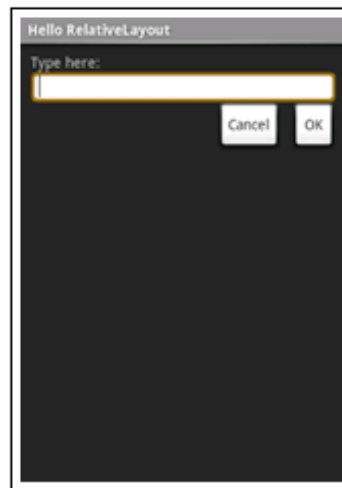
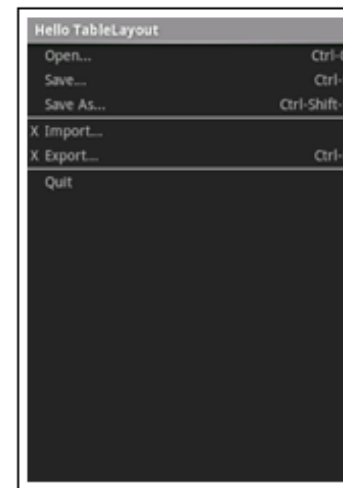
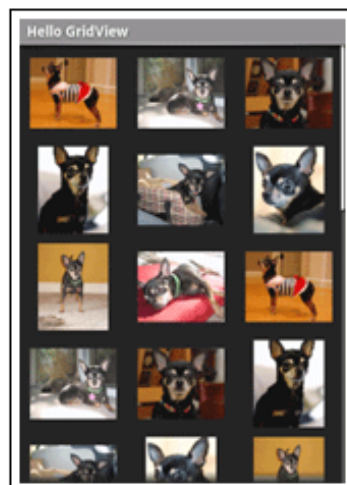


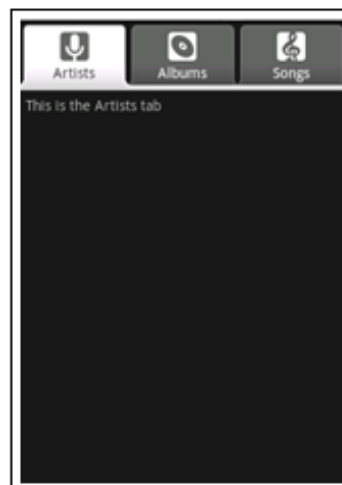
Table Layout



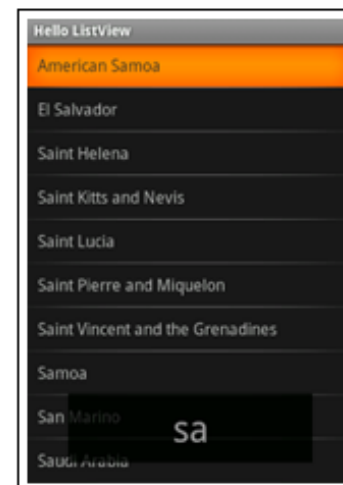
Grid View



Tab Layout



List View



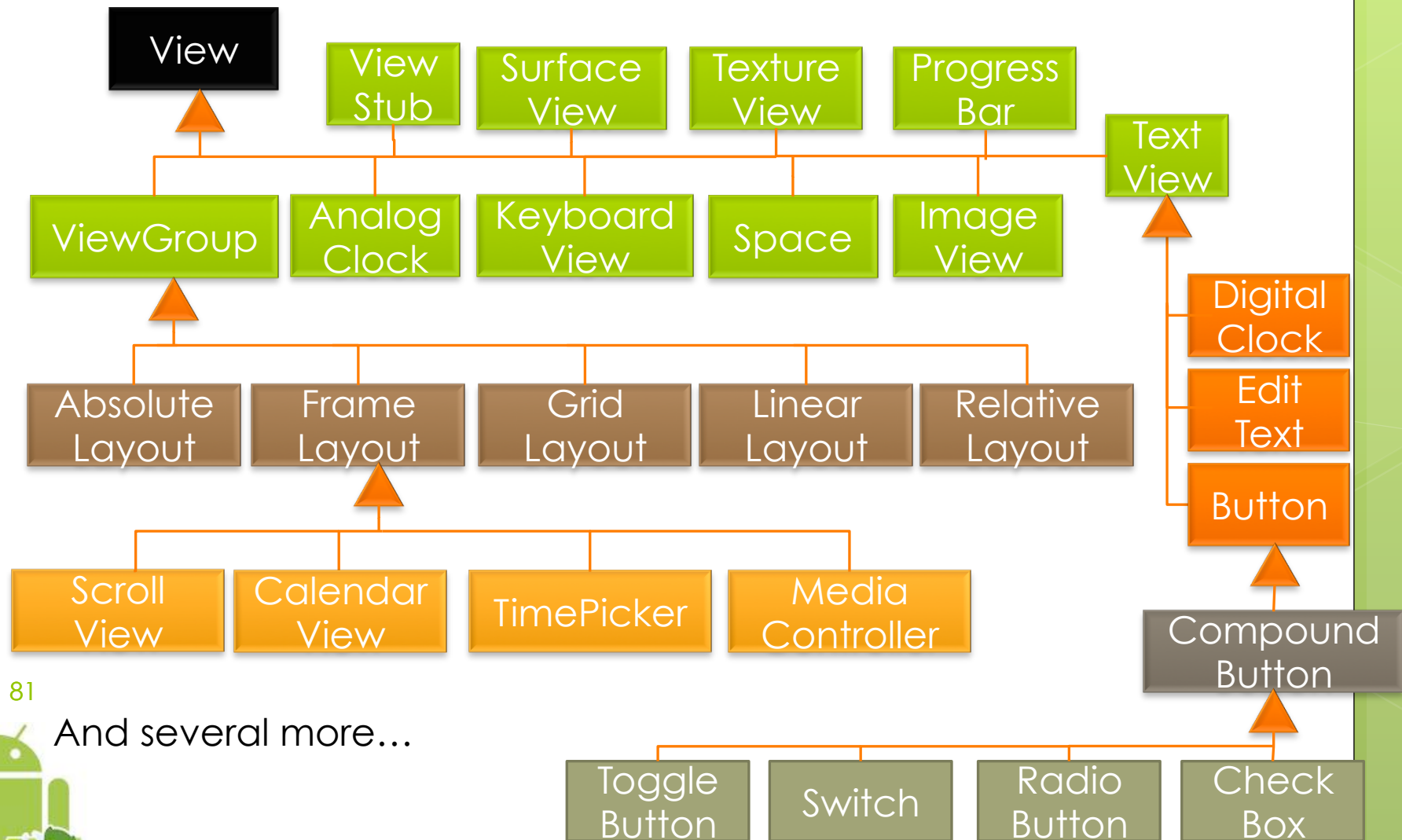
# Defining layouts in XML

Each layout file must contain **exactly one root element**, which must be a **View or ViewGroup** object.

Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.



# A view of the Views



81

And several more...



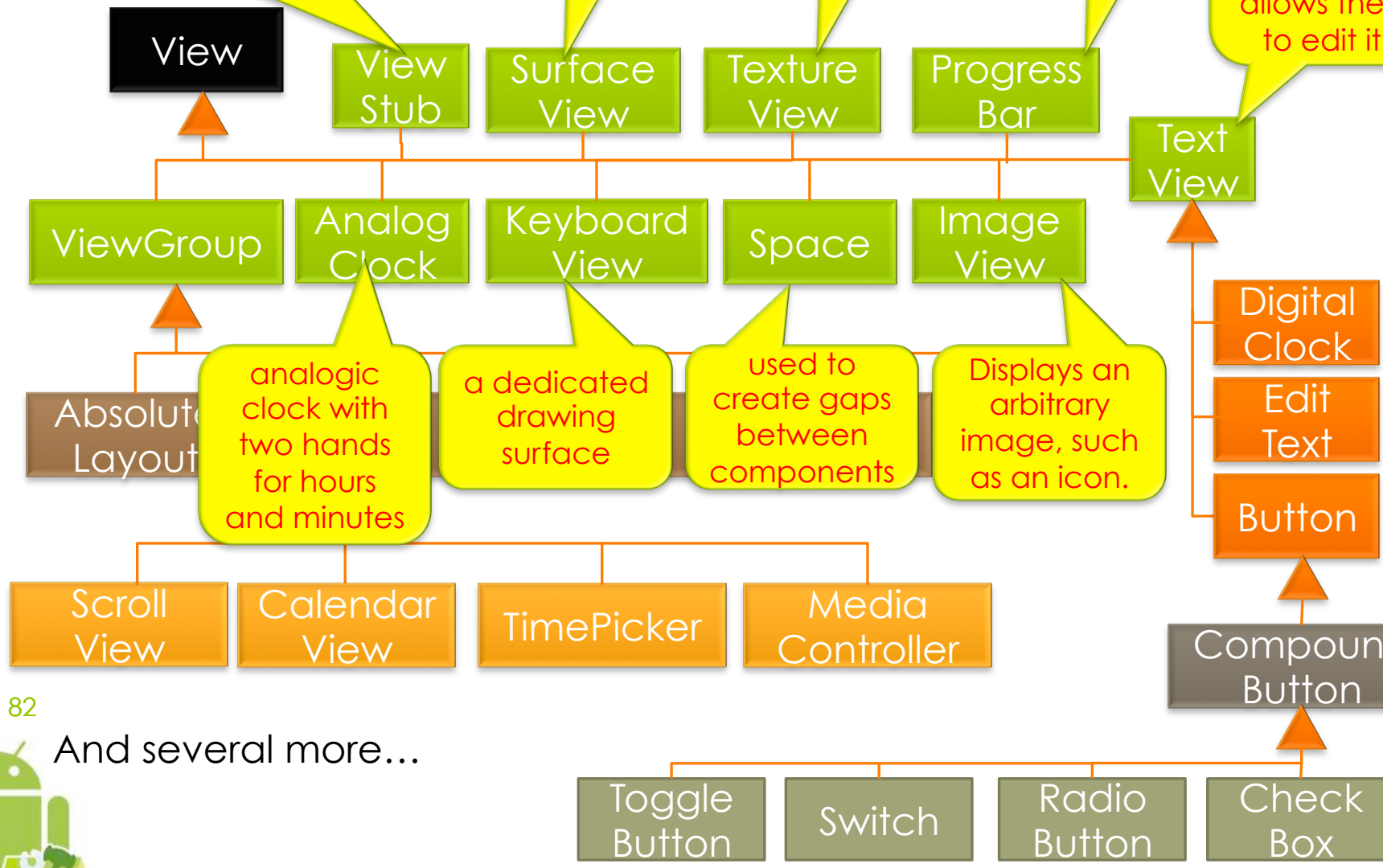
invisible, zero-sized View that can be used to lazily inflate layout resources at runtime.

a dedicated drawing surface

used to display a content stream

a dedicated drawing surface

Displays text to the user and optionally allows them to edit it.



82

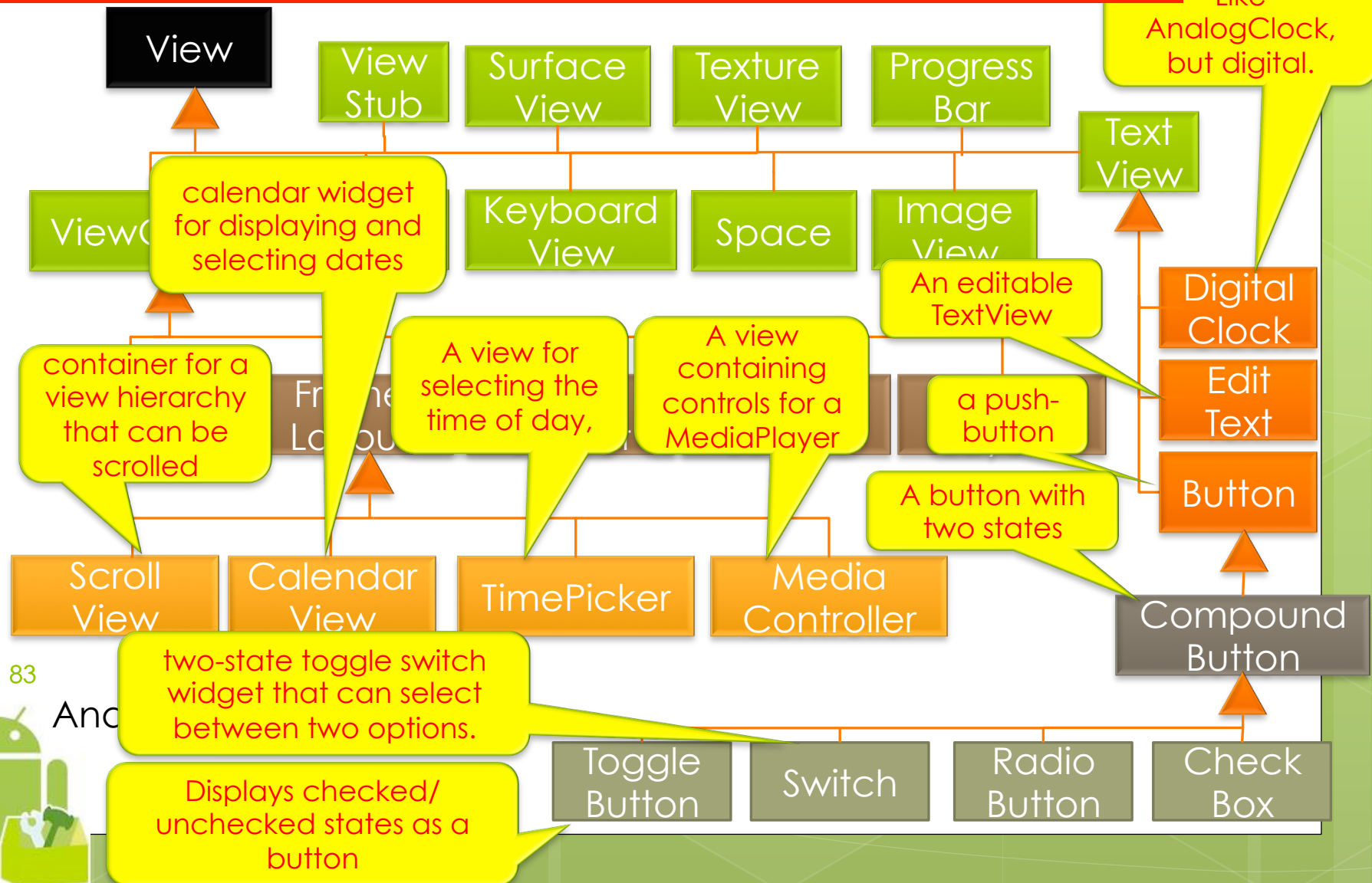
And several more...



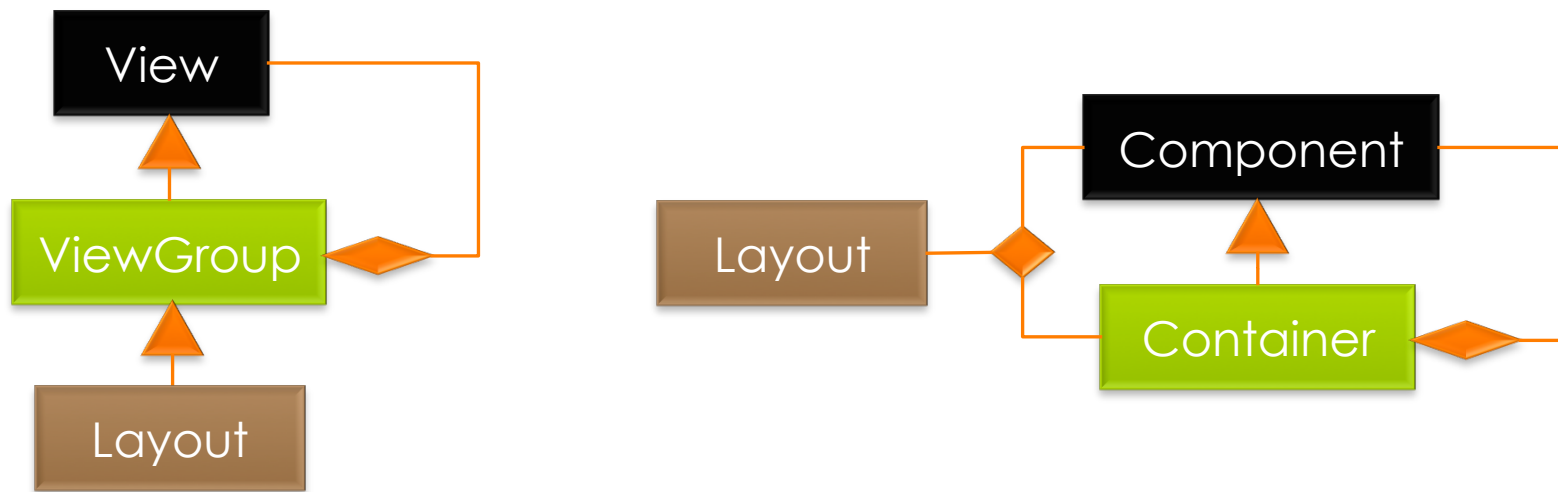
# A view of the Views

PLEASE READ THIS:

<http://developer.android.com/reference/android/widget/package-summary.html>

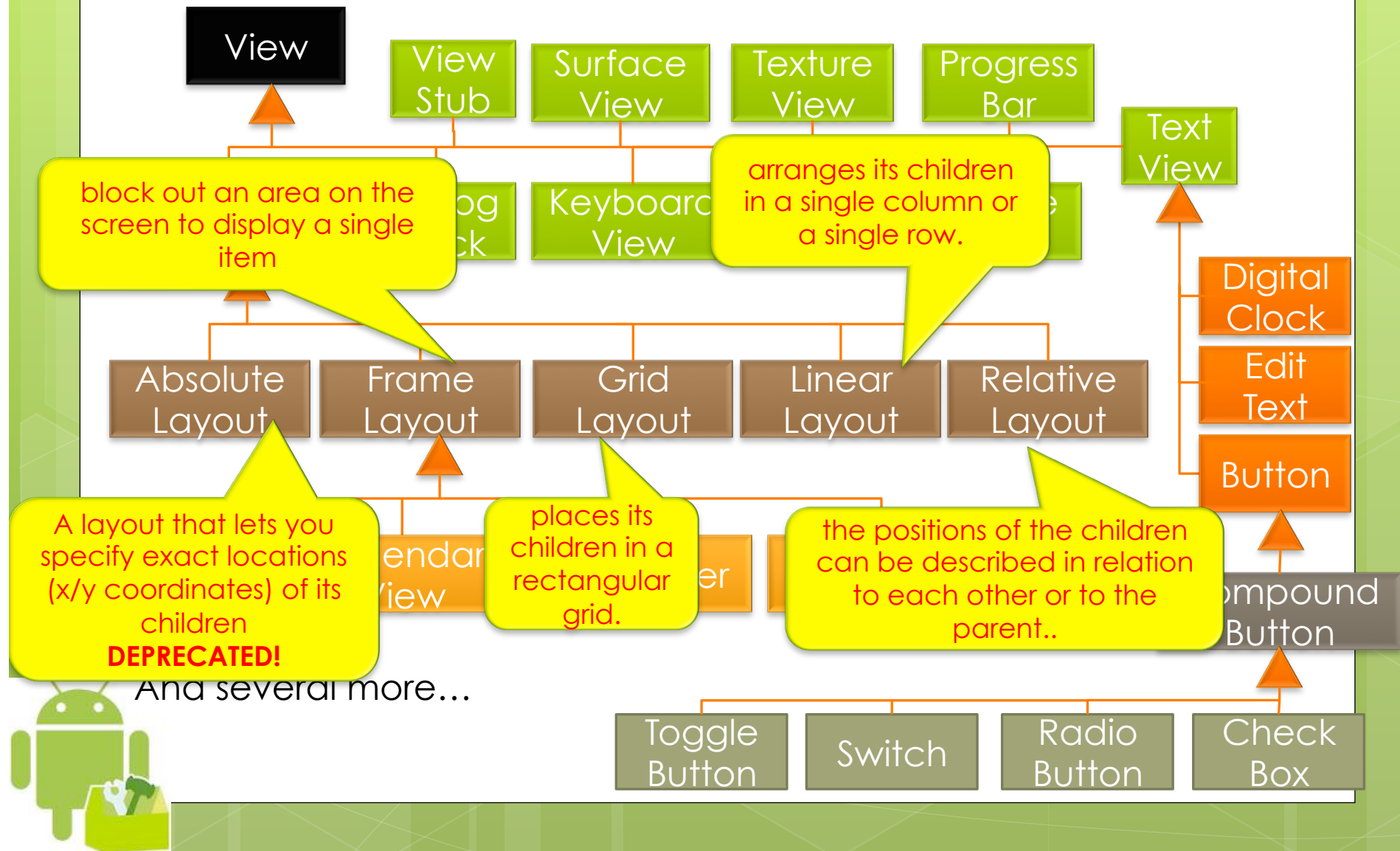


# Android vs. Swing architectures





# A view of the Views



# Layout examples

See several examples in

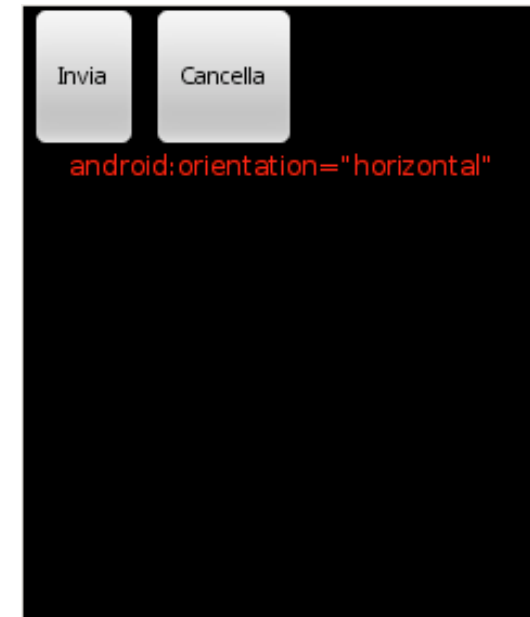
<http://luca-petrosino.blogspot.com/2011/03/android-view-e-layout.html>

We quickly discuss some of them here.



# Layout properties - horizontal

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:text="Invia"
        android:id="@+id/Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button
        android:text="Cancella"
        android:id="@+id/Button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```



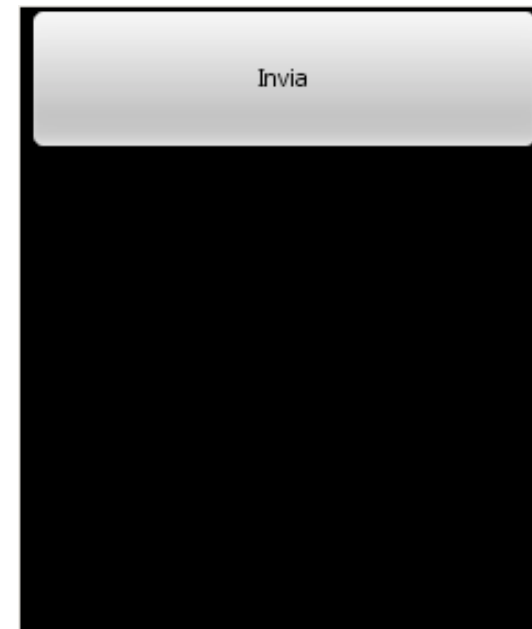
# Layout properties - margin

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:text="Invia"
        android:id="@+id/Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="25px"
        android:layout_marginRight="25px"/>
    <Button
        android:text="Cancella"
        android:id="@+id/Button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```



# Layout properties – fill\_parent

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:text="Invia"
        android:id="@+id/Button1"
        android:layout_width="fill_parent" ←
        android:layout_height="wrap_content"/>
    <Button
        android:text="Cancella"
        android:id="@+id/Button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```



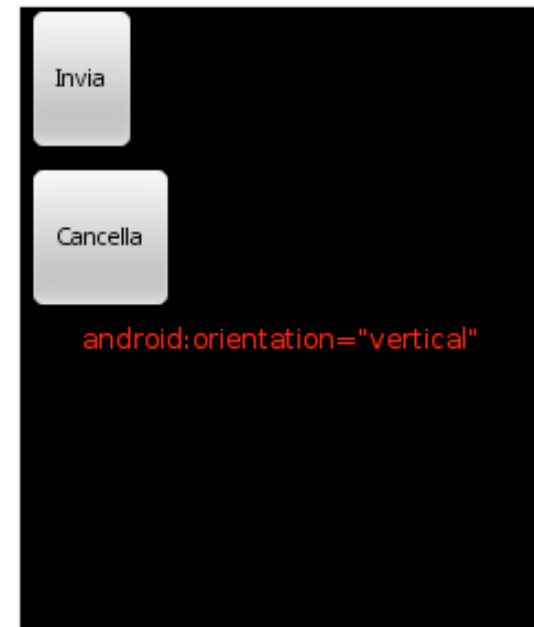
# Layout properties – weight

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:text="Invia"
        android:id="@+id/Button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <Button
        android:text="Cancella"
        android:id="@+id/Button2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"/>
</LinearLayout>
```



# Layout properties - vertical

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" ←
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:text="Invia"
        android:id="@+id/Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button
        android:text="Cancella"
        android:id="@+id/Button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```



# Basic operations on View

- **Set properties:** e.g. setting the text of a TextView (**setText()**). Available properties, getters and setters vary among the different subclasses of views. Properties can also be set in the XML layout files.
- **Set focus:** Focus is moved in response to user input. To force focus to a specific view, call **requestFocus()**.
- **Set up listeners:** e.g. **setOnFocusChangeListener** (**android.view.View.OnFocusChangeListener**). View subclasses offer specialized listeners.
- **Set visibility:** You can hide or show views using **setVisibility(int)**. (One of **VISIBLE**, **INVISIBLE**, or **GONE**.)

92



Invisible, but it still takes  
up space for layout  
purposes

Invisible, and it takes no  
space for layout  
purposes



# Size and location of a View

Although there are setters for position and size, their values are usually controlled (and overwritten) by the Layout.

- **getLeft(), getTop()** : get the coordinates of the upper left vertex.
- **getMeasuredHeight()** e **getMeasuredWidth()** return the preferred dimensions
- **getWidth()** e **getHeight()** return the actual dimensions.



# Custom views

To implement a custom view, you will usually begin overriding for some of the standard methods that the framework calls on all views (at least `onDraw()`).

For more details, see

<http://developer.android.com/reference/android/view/View.html>



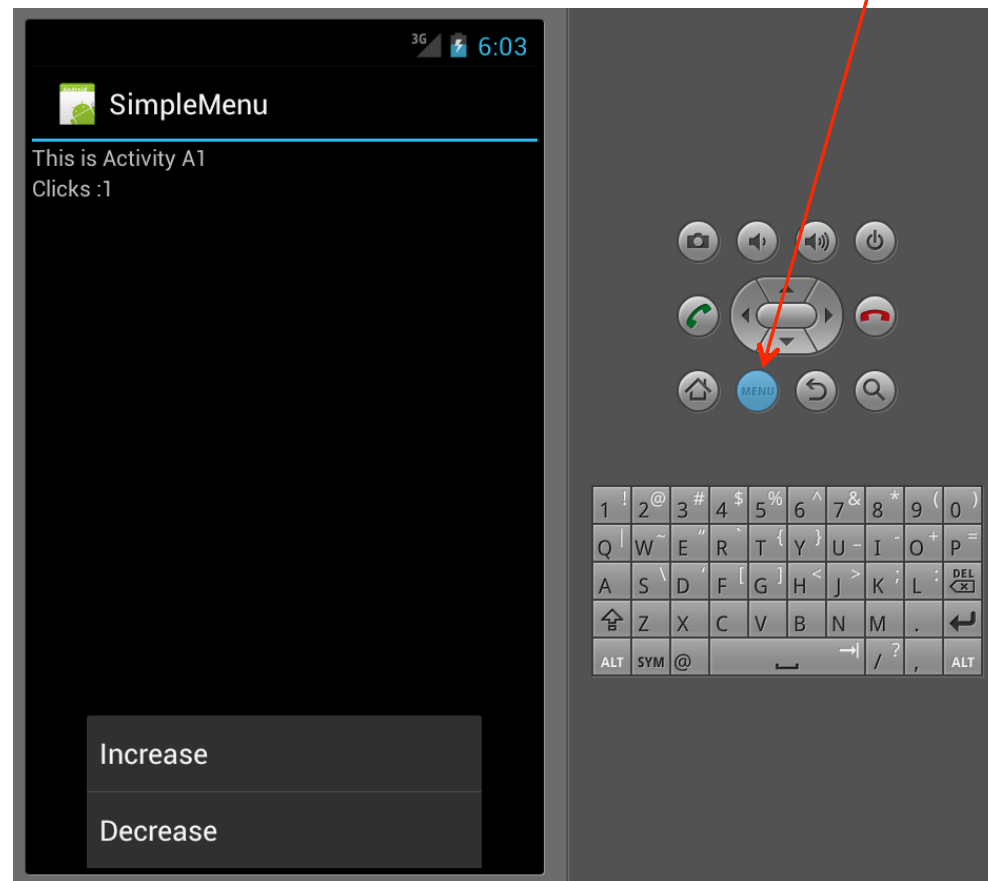
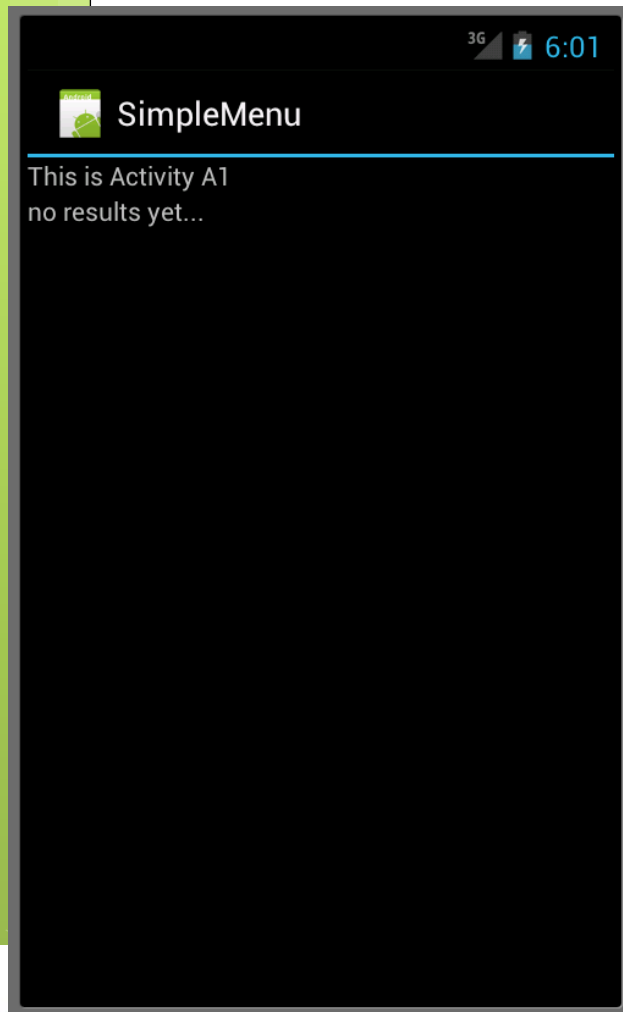


# Basic UI elements: Android Menus (basics)

Marco Ronchetti  
Università degli Studi di Trento

---

# SimpleMenu



# Layout & Strings

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
    <TextView
        android:id="@+id/tf1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/output" />
</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">This is Activity A1</string>
    <string name="app_name">SimpleMenu</string>
    <string name="output">no results yet...</string>
</resources>
```



# SimpleMenu – A1

```
public class A1 extends Activity {
    int nClicks=0;
    protected void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.layout1);
    }
    public boolean onCreateOptionsMenu(Menu menu){
        super.onCreateOptionsMenu(menu);
        int base=Menu.FIRST;
        MenuItem item1=menu.add(base,1,1,"Increase");
        MenuItem item2=menu.add(base,2,2,"Decrease");
        return true;
    }
    public boolean onOptionsItemSelected(MenuItem item) {
        TextView tf = (TextView) findViewById(R.id.tf1);
        if (item.getItemId()==1) increase();
        else if (item.getItemId()==2) decrease();
        else return super.onOptionsItemSelected(item);
        tf.setText("Clicks :"+nClicks);
        return true;
    }
}
```

```
package it.unitn.science.latemar;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

← Menu is created

Respond to a  
Menu event

```
private void increase() {
    nClicks++;
}
private void decrease() {
    nClicks--;
}
}
```





# Calling Activities in other apps: Android Intents

Marco Ronchetti  
Università degli Studi di Trento

---

# Re-using activities

When you create an application, you can assemble it from

- activities that you create
- activities you re-use from other applications.

An app can incorporate activities from other apps.

Yes, but how? By means of **Intents**

These activities are **bound at runtime**: newly installed applications can take advantage of already installed activities

100





# Our code – IntentUtils - 1

```
package it.unitn.science.latemar;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;

public class IntentUtils {

    public static void invokeWebBrowser(Activity activity) {
        Intent intent=new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("http://latemar.science.unitn.it"));
        activity.startActivity(intent);
    }

    public static void invokeWebSearch(Activity activity) {
        Intent intent=new Intent(Intent.ACTION_WEB_SEARCH,
            Uri.parse("http://www.google.com"));
        activity.startActivity(intent);
    }
}
```

101



## Our code – IntentUtils - 2

```
public static void dial(Activity activity) {  
    Intent intent=new Intent(Intent.ACTION_DIAL);  
    activity.startActivity(intent);  
}  
  
public static void showDirections(Activity activity){  
    Intent intent = new Intent(android.content.Intent.ACTION_VIEW,  
        Uri.parse("http://maps.google.com/maps? saddr=Bolzano&daddr=Trento"))  
    activity.startActivity(intent);  
}  
}
```



# Our Code: IntentsActivity -1

```
package it.unitn.science.latemar;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class IntentsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv=new TextView(this);
        tv.setText("Push the menu button!");
        setContentView(tv);
    }

    public boolean onCreateOptionsMenu(Menu menu){
        super.onCreateOptionsMenu(menu);
        int base=Menu.FIRST;
        MenuItem item1=menu.add(base,1,1,"invokeWebBrowser-VIEW");
        MenuItem item2=menu.add(base,2,2,"invokeWebBrowser-SEARCH");
        MenuItem item3=menu.add(base,3,3,"showDirections");
        MenuItem item5=menu.add(base,4,4,"dial");
        return true;
    }
}
```

103



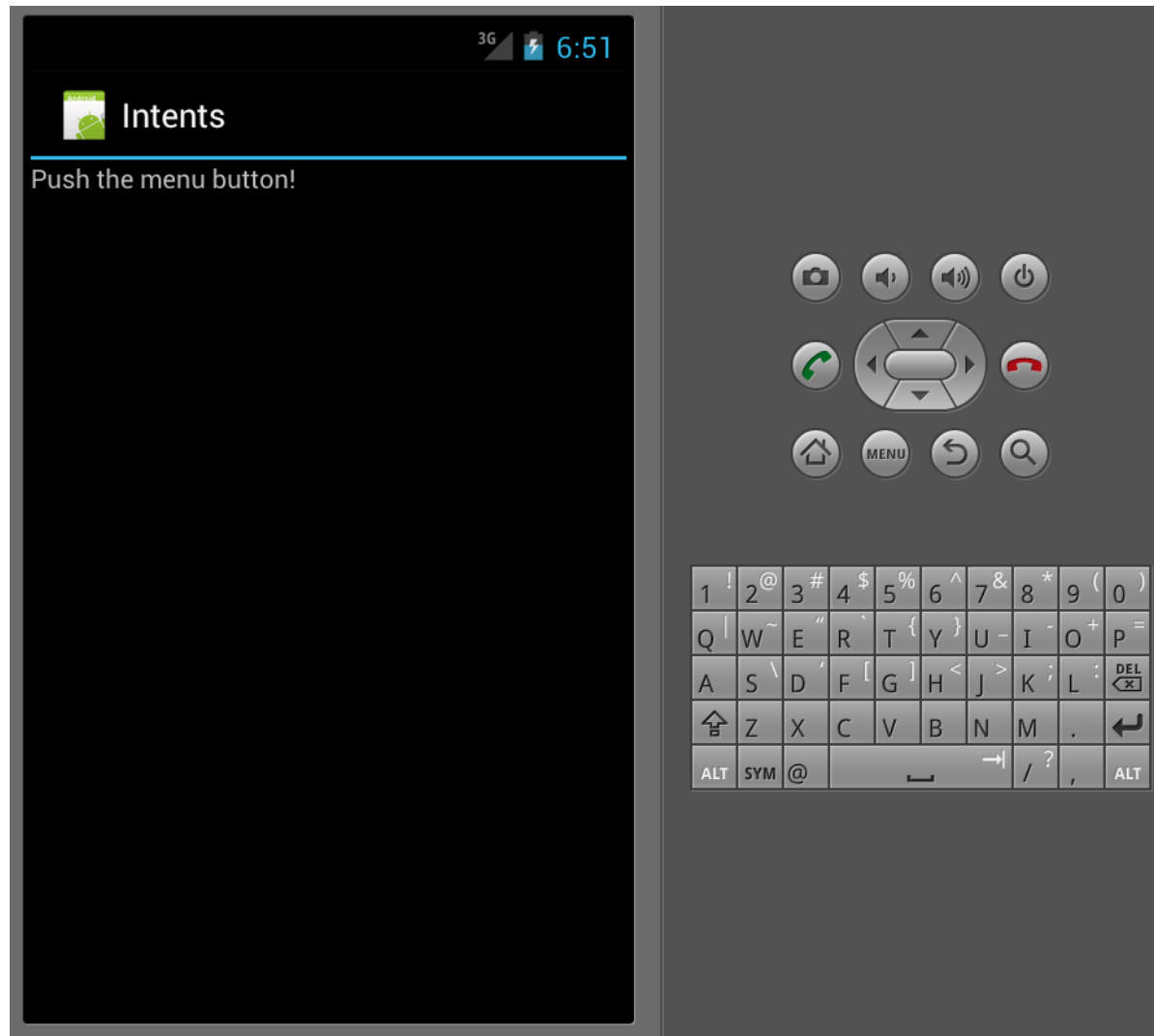
# Our Code: IntentsActivity -2

```
public boolean onOptionsItemSelected(MenuItem item) {  
    System.err.println("item="+item.getItemId());  
    if (item.getItemId()==1)  
        IntentUtils.invokeWebBrowser(this);  
    else if (item.getItemId()==2)  
        IntentUtils.invokeWebSearch(this);  
    else if (item.getItemId()==3)  
        IntentUtils.showDirections(this);  
    else if (item.getItemId()==4)  
        IntentUtils.dial(this);  
    else  
        return super.onOptionsItemSelected(item);  
    return true;  
}  
}
```

104



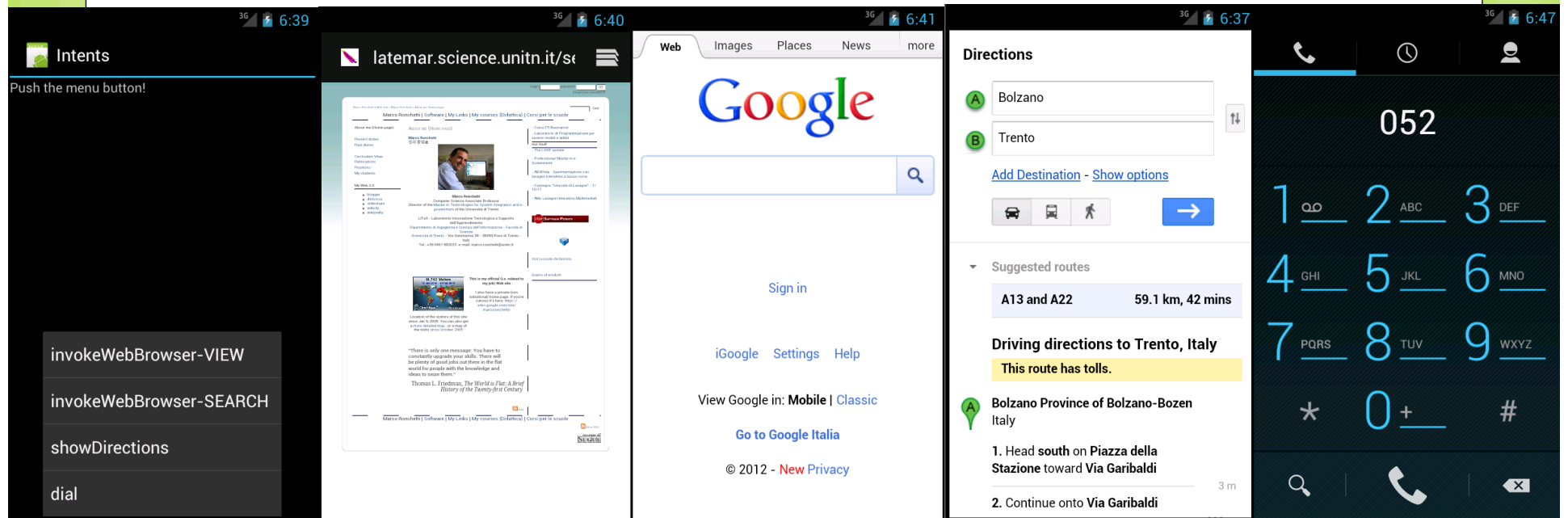
# Our App



105



# Our activities



106





# Intent structure and resolution

Marco Ronchetti  
Università degli Studi di Trento

---

# Intent structure

Who will perform the action?

- **Component name** (can be unnamed)

Which action should be performed?

- **Action identifier (a string)**

Which data should the action act on ?

- **Data** The URI of the data to be acted on

How we classify the action to be performed?

- **Category** A (usually codified) string.

How do we directly pass data?

- **Extras** Key-value pairs for additional information that should be delivered to the component handling the intent

How do we specify behavior modification?

- **Flags** Flags of various sorts.





# Examples of action/data pairs

**ACTION\_VIEW** content://contacts/people/1

- Display information about the person whose identifier is "1".

**ACTION\_DIAL** content://contacts/people/1

- Display the phone dialer with the person filled in.

**ACTION\_VIEW** tel:123

- Display the phone dialer with the given number filled in. Note how the VIEW action does what is considered the most reasonable thing for a particular URI.

**ACTION\_DIAL** tel:123

- Display the phone dialer with the given number filled in.

**ACTION\_EDIT** content://contacts/people/1

- Edit information about the person whose identifier is "1".

**ACTION\_VIEW** content://contacts/people/

- Display a list of people, which the user can browse through.



# Standard Actions Identifiers

ACTION\_MAIN  
ACTION\_VIEW  
ACTION\_ATTACH\_DATA  
ACTION\_EDIT  
ACTION\_PICK  
ACTION\_CHOOSER  
ACTION\_GET\_CONTENT  
ACTION\_DIAL  
ACTION\_CALL  
ACTION\_SEND  
ACTION\_SENDTO  
ACTION\_ANSWER  
ACTION\_INSERT  
ACTION\_DELETE  
ACTION\_RUN  
ACTION\_SYNC  
ACTION\_PICK\_ACTIVITY  
ACTION\_SEARCH  
ACTION\_WEB\_SEARCH  
ACTION\_FACTORY\_TEST



For details see [http://developer.android.com/reference/android/content/Intent.html#CATEGORY\\_ALTERNATIVE](http://developer.android.com/reference/android/content/Intent.html#CATEGORY_ALTERNATIVE)

# Standard Categories

String	CATEGORY_ALTERNATIVE	Set if the activity should be considered as an alternative action to the data the user is currently viewing.
String	CATEGORY_APP_BROWSER	Used with ACTION_MAIN to launch the browser application.
String	CATEGORY_APP_CALCULATOR	Used with ACTION_MAIN to launch the calculator application.
String	CATEGORY_APP_CALENDAR	Used with ACTION_MAIN to launch the calendar application.
String	CATEGORY_APP_CONTACTS	Used with ACTION_MAIN to launch the contacts application.
String	CATEGORY_APP_EMAIL	Used with ACTION_MAIN to launch the email application.
String	CATEGORY_APP_GALLERY	Used with ACTION_MAIN to launch the gallery application.
String	CATEGORY_APP_MAPS	Used with ACTION_MAIN to launch the maps application.
String	CATEGORY_APP_MARKET	This activity allows the user to browse and download new applications.
String	CATEGORY_APP_MESSAGING	Used with ACTION_MAIN to launch the messaging application.
String	CATEGORY_APP_MUSIC	Used with ACTION_MAIN to launch the music application.
String	CATEGORY_BROWSABLE	Activities that can be safely invoked from a browser must support this category.
String	CATEGORY_CAR_DOCK	An activity to run when device is inserted into a car dock.
String	CATEGORY_CAR_MODE	Used to indicate that the activity can be used in a car environment.
String	CATEGORY_DEFAULT	Set if the activity should be an option for the default action (center press) to perform on a piece of data.
String	CATEGORY_DESK_DOCK	An activity to run when device is inserted into a car dock.
String	CATEGORY_DEVELOPMENT_PREFERENCE	This activity is a development preference panel.
String	CATEGORY_EMBED	Capable of running inside a parent activity container.
String	CATEGORY_FRAMEWORK_INSTRUMENTATION_TEST	To be used as code under test for framework instrumentation tests.
String	CATEGORY_HE_DESK_DOCK	An activity to run when device is inserted into a digital (high end) dock.
String	CATEGORY_HOME	This is the home activity, that is the first activity that is displayed when the device boots.
String	CATEGORY_INFO	Provides information about the package it is in; typically used if a package does not contain a CATEGORY_LAUNCHER to provide
String	CATEGORY_LAUNCHER	Should be displayed in the top-level launcher.
String	CATEGORY_LE_DESK_DOCK	An activity to run when device is inserted into an analog (low end) dock.
String	CATEGORY_MONKEY	This activity may be exercised by the monkey or other automated test tools.
String	CATEGORY_OPENABLE	Used to indicate that a GET_CONTENT intent only wants URIs that can be opened with ContentResolver.openInputStream.
String	CATEGORY_PREFERENCE	This activity is a preference panel.
String	CATEGORY_SAMPLE_CODE	To be used as a sample code example (not part of the normal user experience).
String	CATEGORY_SELECTED_ALTERNATIVE	Set if the activity should be considered as an alternative selection action to the data the user has currently selected.
String	CATEGORY_TAB	Intended to be used as a tab inside of an containing TabActivity.
String	CATEGORY_TEST	To be used as a test (not part of the normal user experience).
String	CATEGORY_UNIT_TEST	To be used as a unit test (run through the Test Harness).



# Implicit intents and intent resolution

*Implicit intents* do not name a target (the field for the component name is blank).

In the absence of a designated target, the Android system must find the best component (or components) to handle the intent.

It does so by comparing the contents of the Intent object to *intent filters*, structures associated with components that can potentially receive intents.

Filters advertise the capabilities of a component and delimit the intents it can handle. They open the component to the possibility of receiving implicit intents of the advertised type. If a component does not have any intent filters, it can receive only explicit intents. A component with filters can receive both explicit and implicit intents.



# Intent Filters

Only three aspects of an Intent object are consulted when the object is tested against an intent filter:

- action

- data (both URI and data type)

- category

The extras and flags play no part in resolving which component receives an intent.



# Intents

Intent messaging is a facility for late run-time binding between components in the same or different applications.

The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed — or, often in the case of broadcasts, a description of something that has happened and is being announced.

There are separate mechanisms for delivering intents to each type of component.



# Using intents with activities

An Intent object is passed to `Context.startActivity()` or `Activity.startActivityForResult()` to launch an activity or get an existing activity to do something new. (It can also be passed to `Activity.setResult()` to return information to the activity that called `startActivityForResult()`.)



# Other uses of Intents

An Intent object is passed to `Context.startService()` to initiate a service or deliver new instructions to an ongoing service. Similarly, an intent can be passed to `Context.bindService()` to establish a connection between the calling component and a target service. It can optionally initiate the service if it's not already running.

Intent objects passed to any of the broadcast methods (such as `Context.sendBroadcast()`, `Context.sendOrderedBroadcast()`, or `Context.sendStickyBroadcast()`) are delivered to all interested broadcast receivers. Many kinds of broadcasts originate in system code.

In each case, the Android system finds the appropriate activity, service, or set of broadcast receivers to respond to the intent, instantiating them if necessary. There is no overlap within these messaging systems: Broadcast intents are delivered only to broadcast receivers, never to activities or services. An intent passed to `startActivity()` is delivered only to an activity, never to a service or broadcast receiver, and so on.

