Recap

Marco Ronchetti
Università degli Studi di Trento

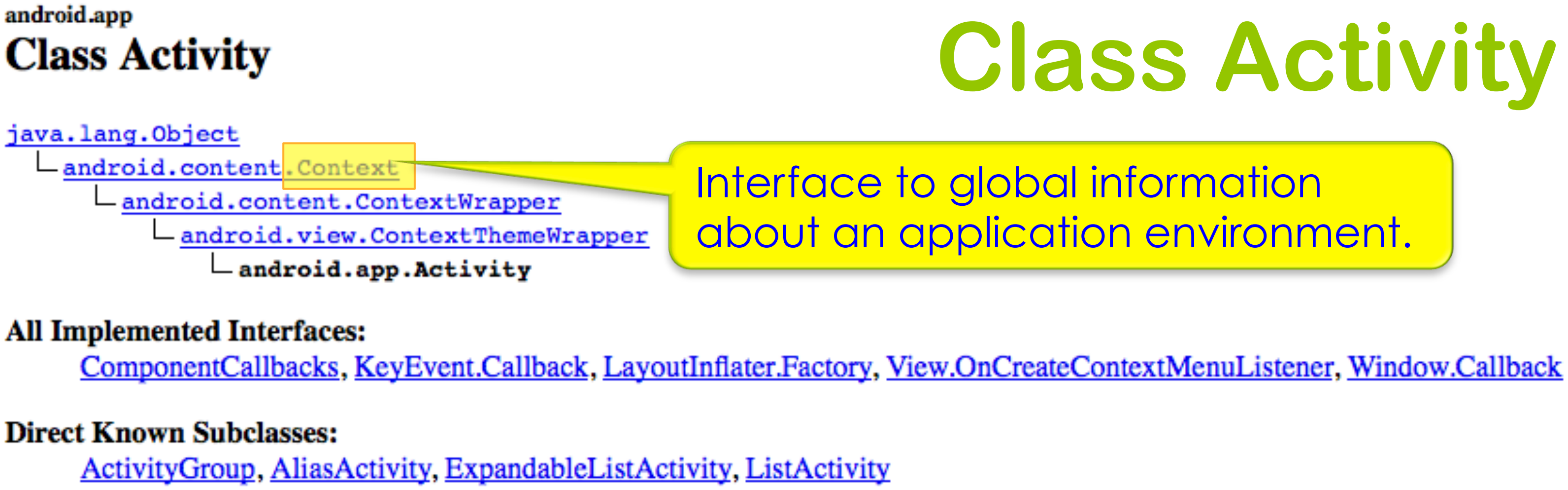
HelloAndroid

```
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);
   }
}
```

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

- What is a TextView??

Class Activity



android.app

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()?

Resources

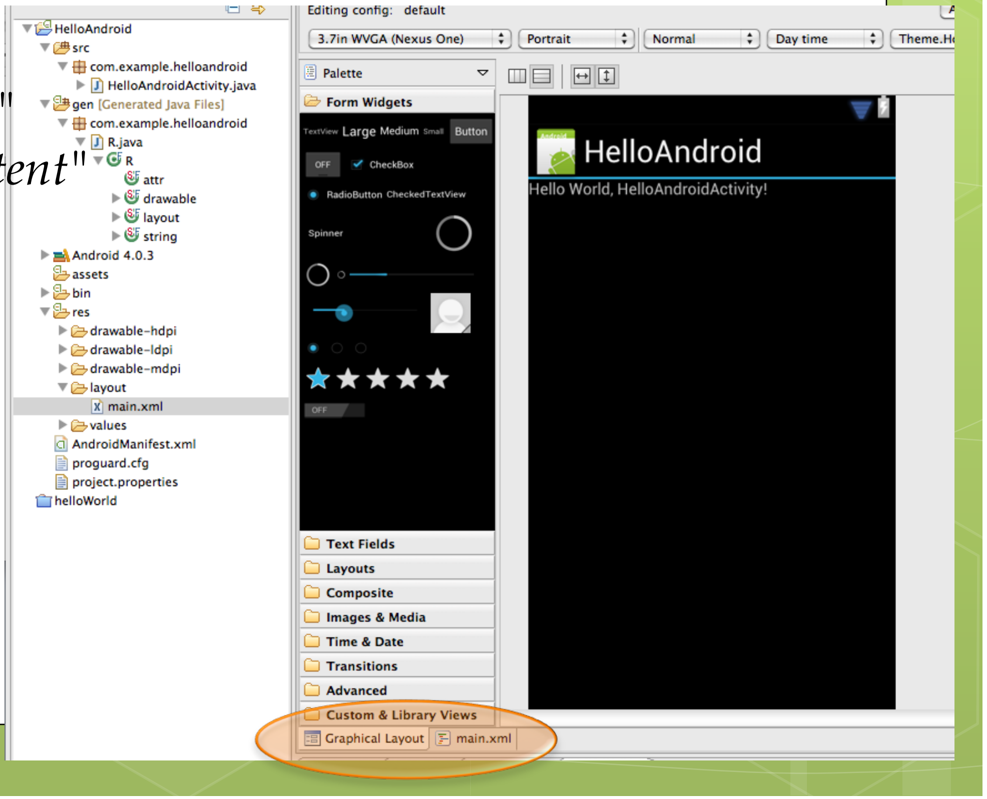
You should always externalize resources (e.g. images and strings) from your application code, so that you can:

- 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.

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>
```


Editing config: default

3.7in WVGA (Nexus One) | Portrait | Normal | Day time | Theme.H

HelloAndroid

Hello World, HelloAndroidActivity!

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.

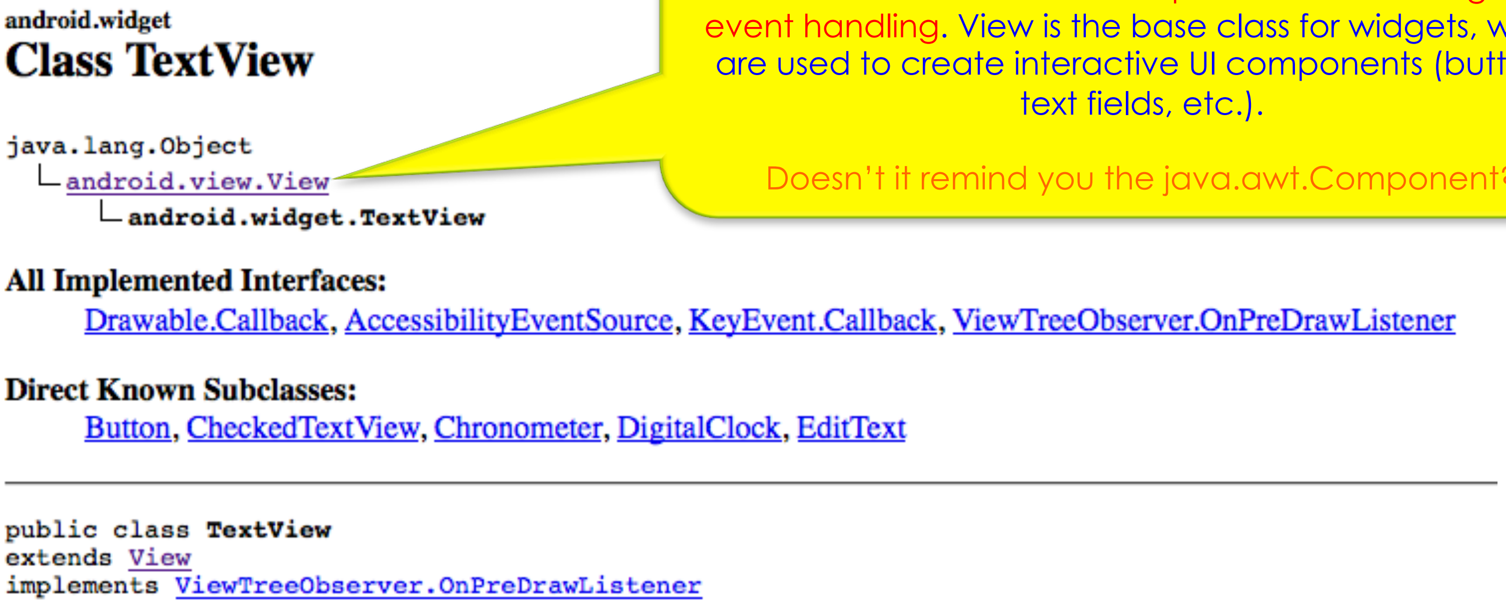
Note: a Bundle is a sort of container for serialized data.

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.

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?



android.widget

Class TextView

```
java.lang.Object
  └android.view.View
      └android.widget.TextView
```

All Implemented Interfaces:

Drawable.Callback, AccessibilityEventSource, KeyEvent.Callback, ViewTreeObserver.OnPreDrawListener

Direct Known Subclasses:

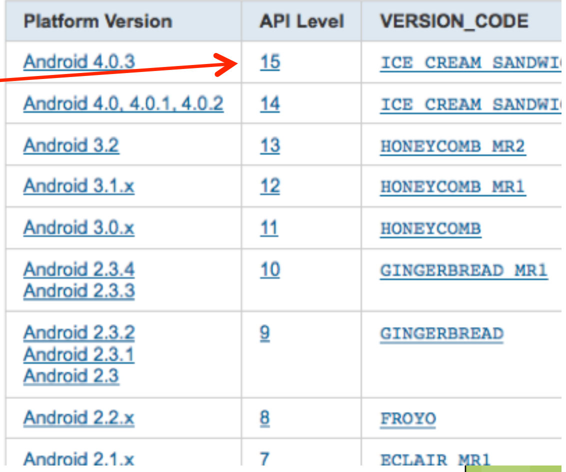
Button, CheckedTextView, Chronometer, DigitalClock, EditText

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

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/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
Android 4.0.3	15	ICE_CREAM_SANDWI
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWI
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1

Nov.2011

Feb 2011

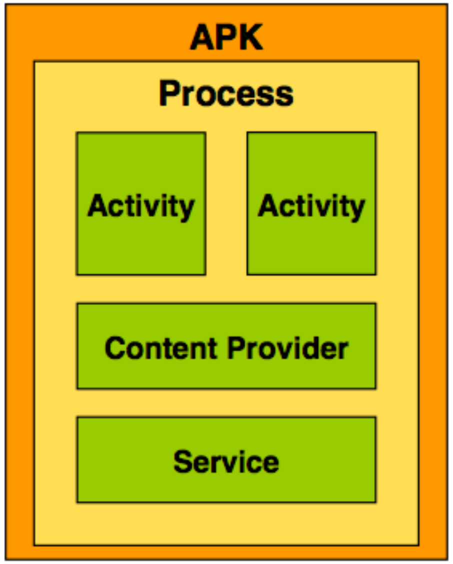
Dic 2010

Mag 2010

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

ONE APPLICATION, ONE PROCESS, MANY ACTIVITIES

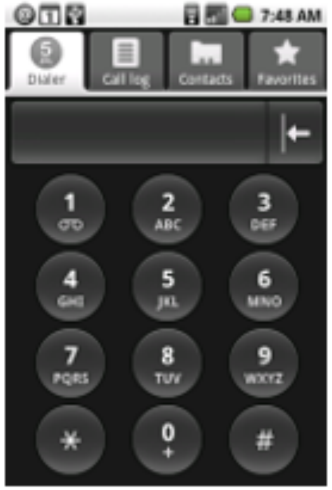
Slide borrowed from Dominik Gruntz (and modified)

Activity

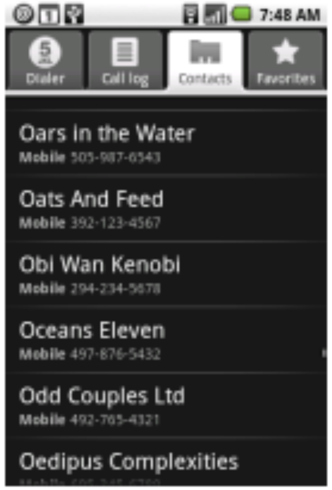
An application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.

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



Dialer



Oars in the Water
Mobile 505-987-6543

Oats And Feed
Mobile 392-123-4567

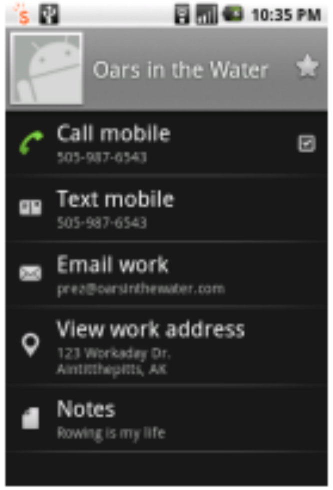
Obi Wan Kenobi
Mobile 294-234-5678

Oceans Eleven
Mobile 497-876-5432

Odd Couples Ltd
Mobile 492-765-4321

Oedipus Complexities

Contacts



Oars in the Water

Call mobile
505-987-6543

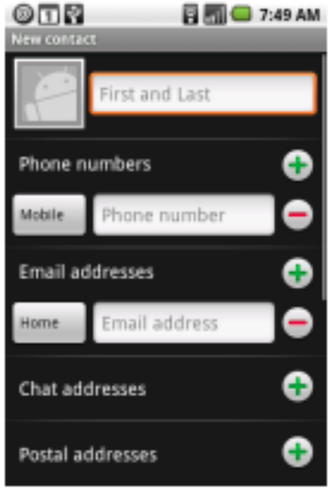
Text mobile
505-987-6543

Email work
prez@oarsinthewater.com

View work address
123 Workaday Dr.
Aintitthepitts, AK

Notes
Rowing is my life

View Contact



New contact

First and Last

Phone numbers
Mobile | Phone number

Email addresses
Home | Email address

Chat addresses

Postal addresses

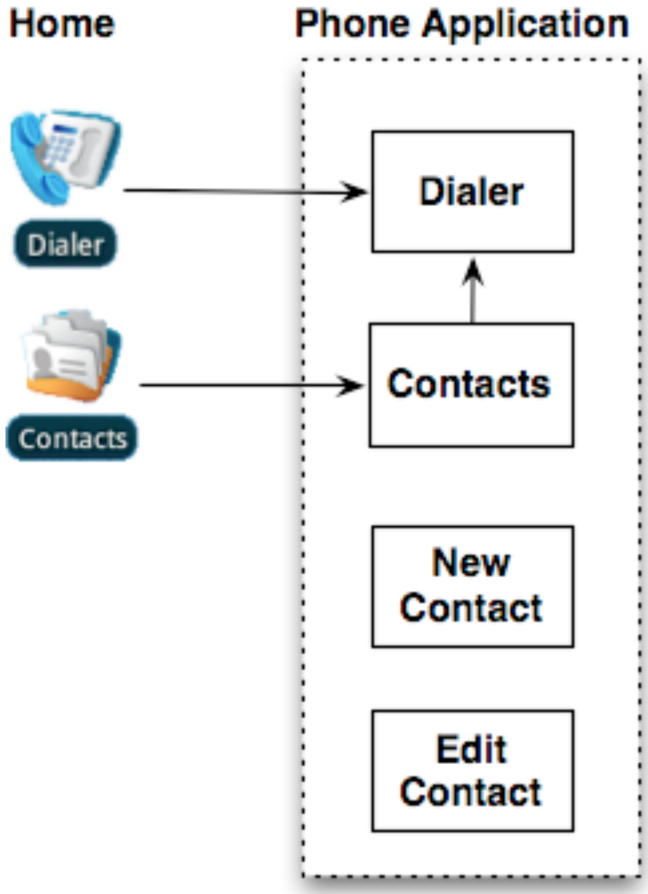
New Contact

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.

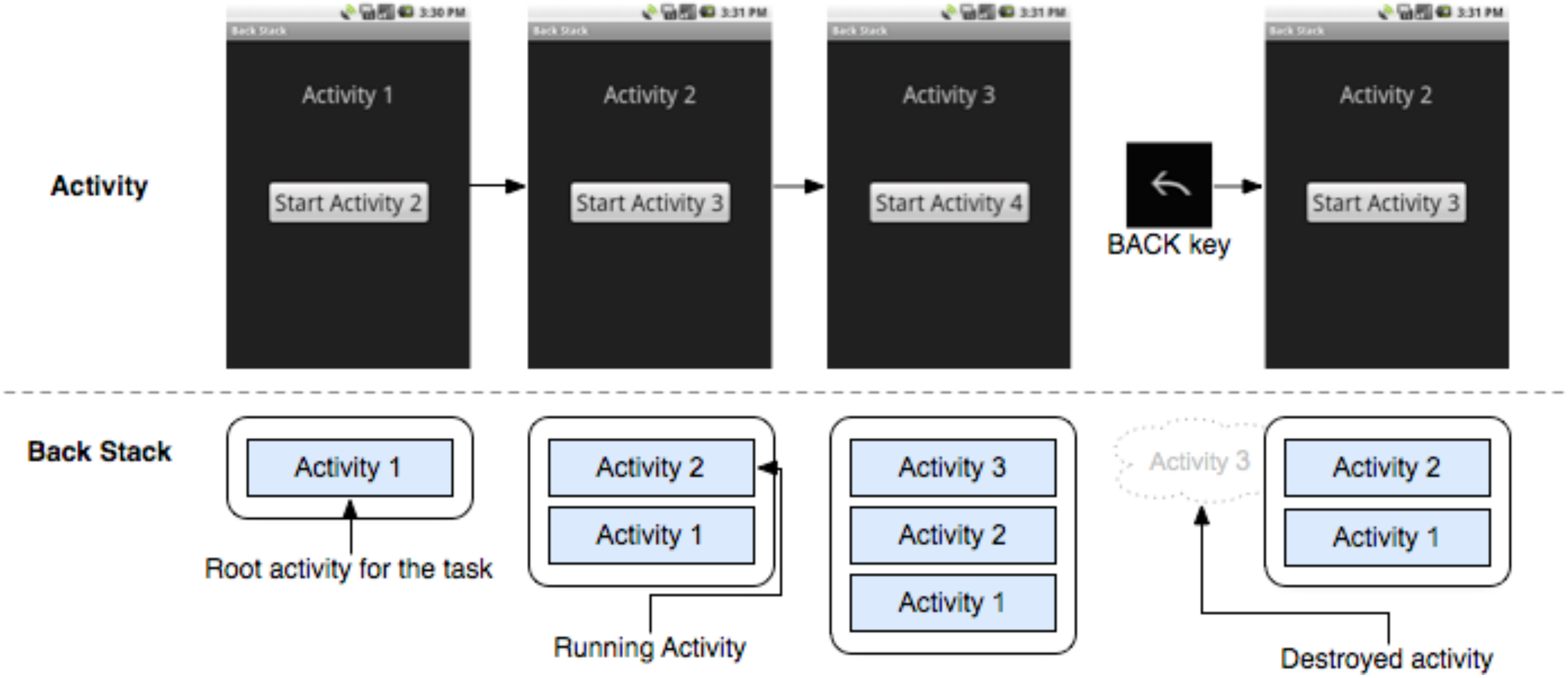
The system preserves the activity in a LIFO stack (the "activity stack" or "back stack").

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.

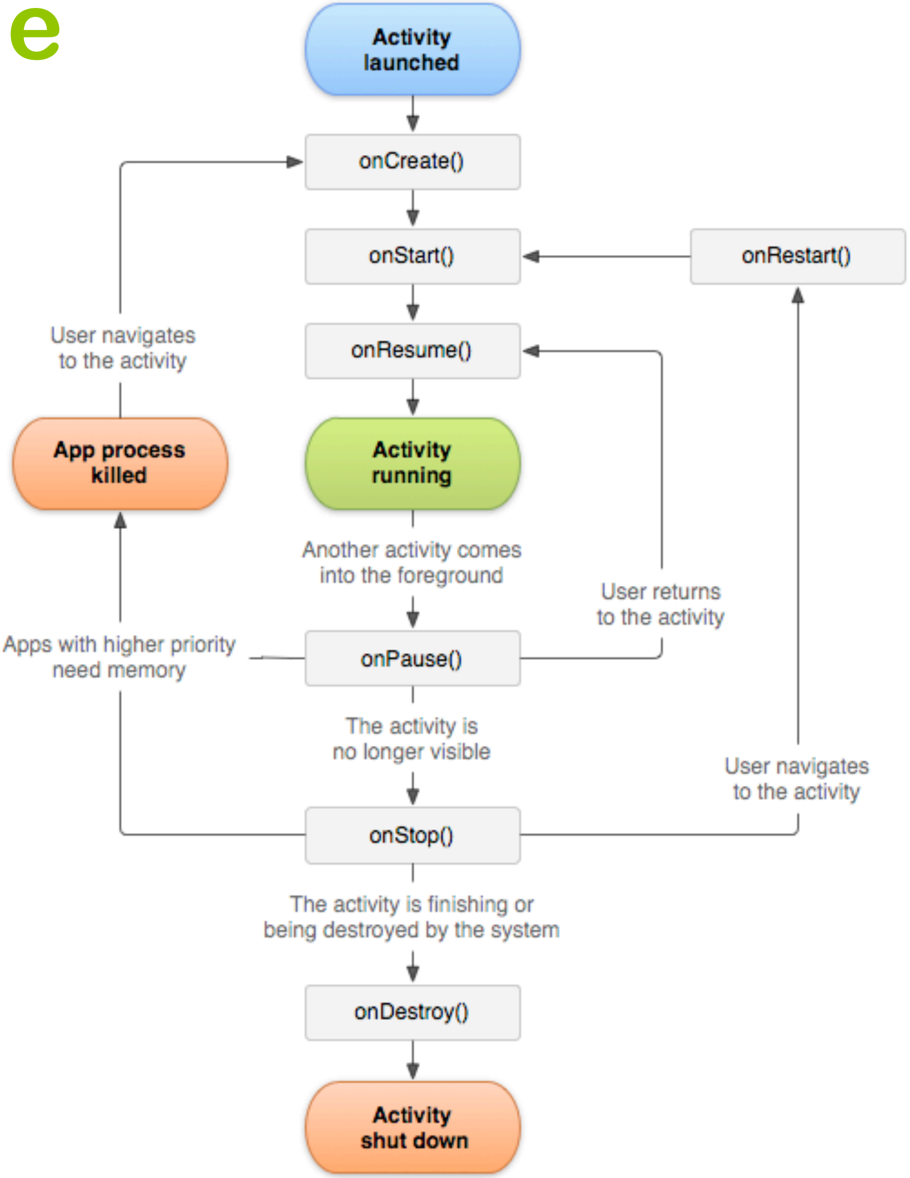
The activity stack

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



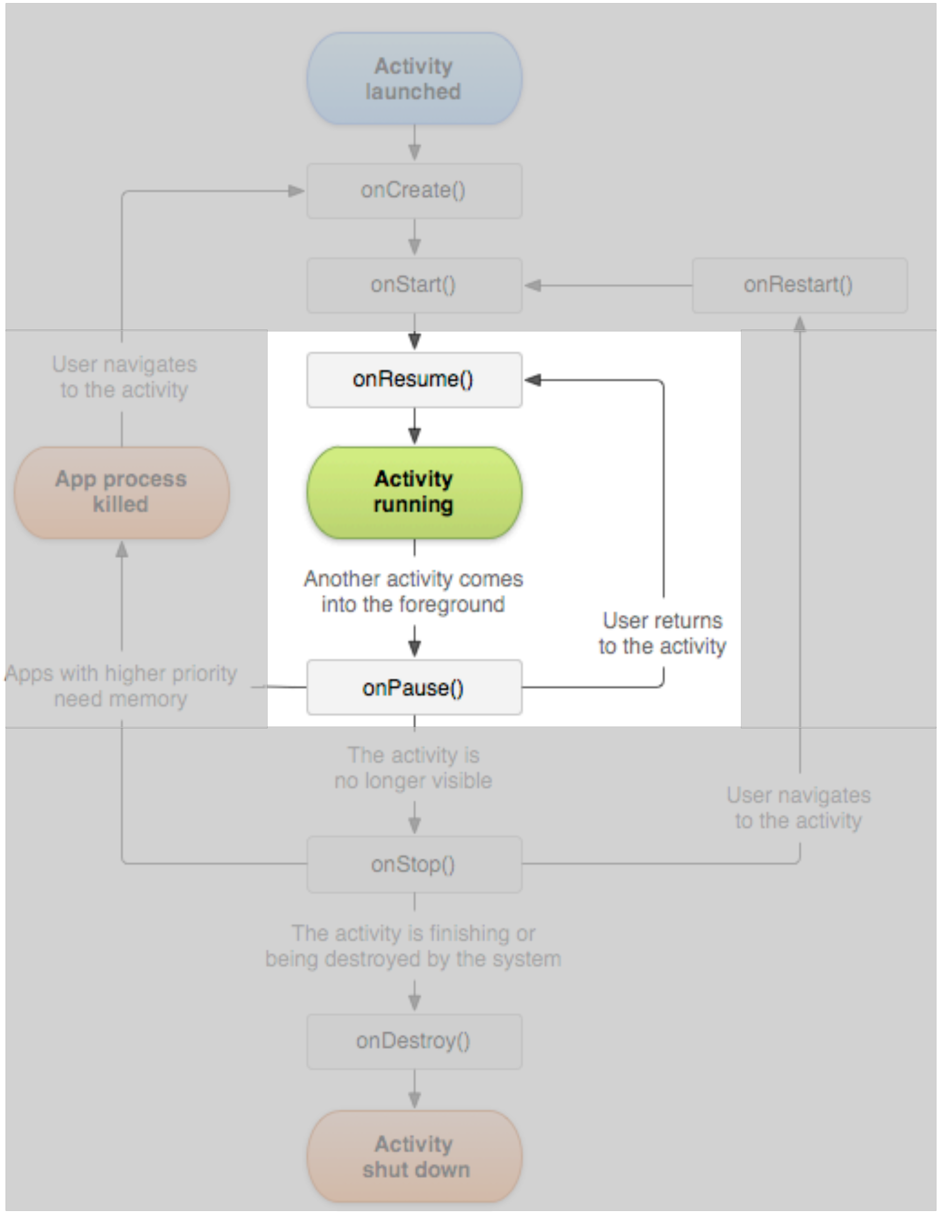
Activity lifecycle

States (colored),
and
Callbacks (gray)


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity lifecycle

The FOREGROUND lifetime



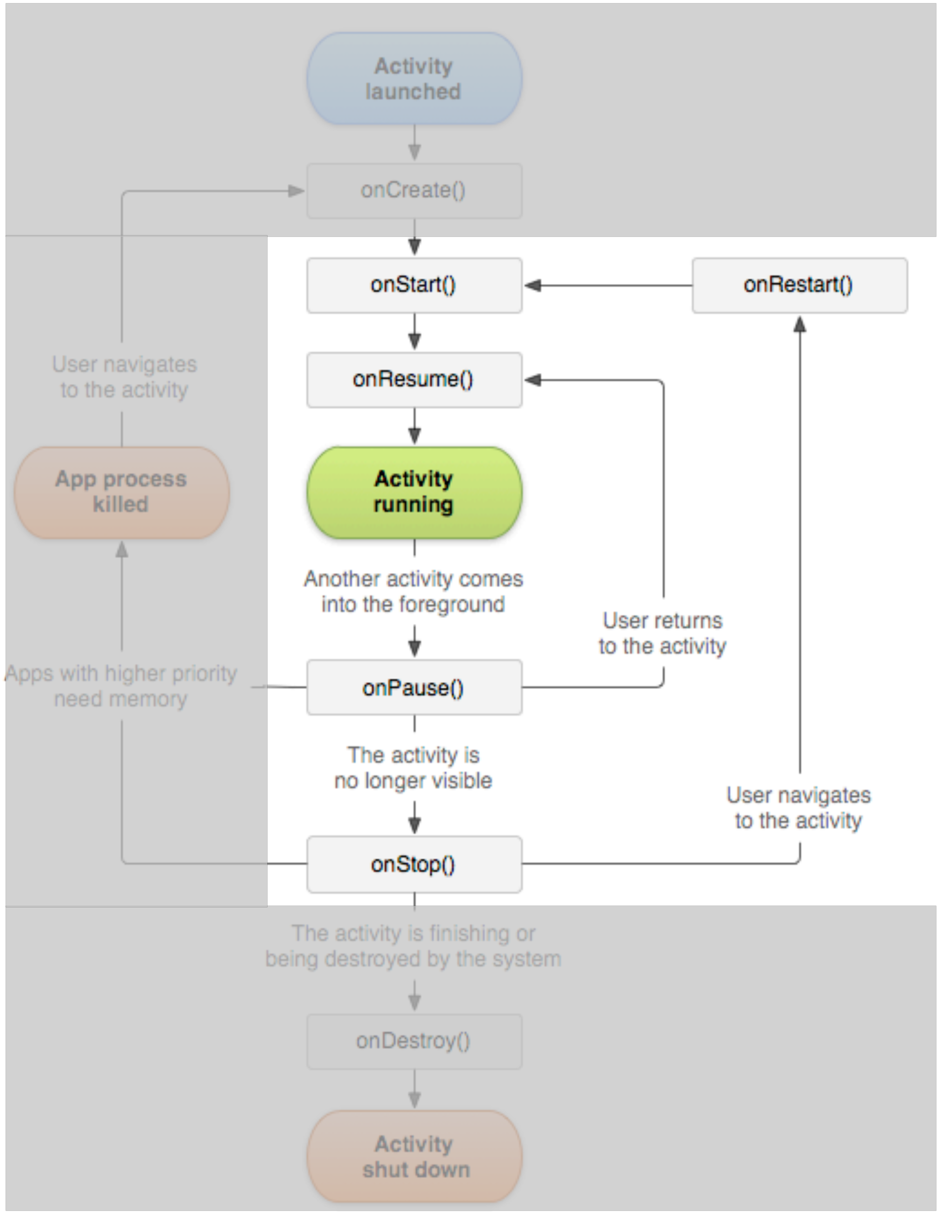
Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity lifecycle

The VISIBLE lifetime

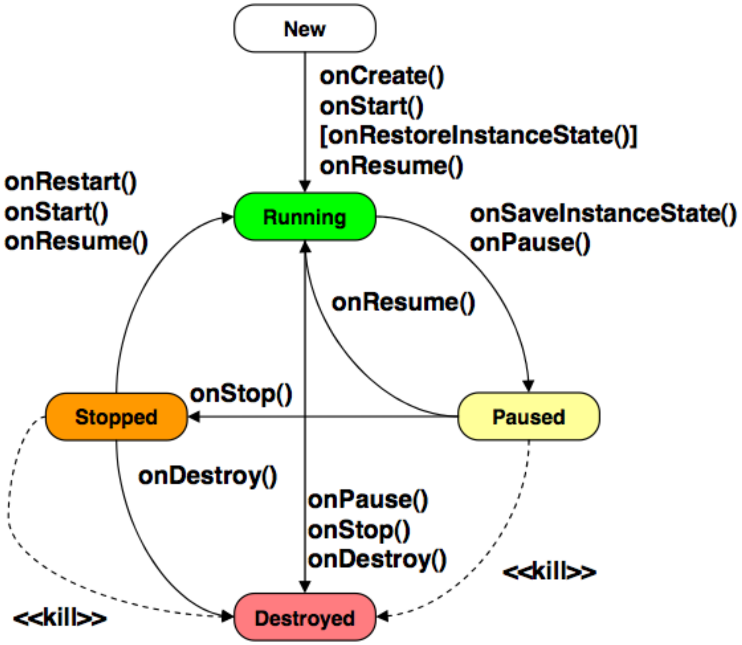
When stopped, your activity should release costly resources, such as network or database connections.

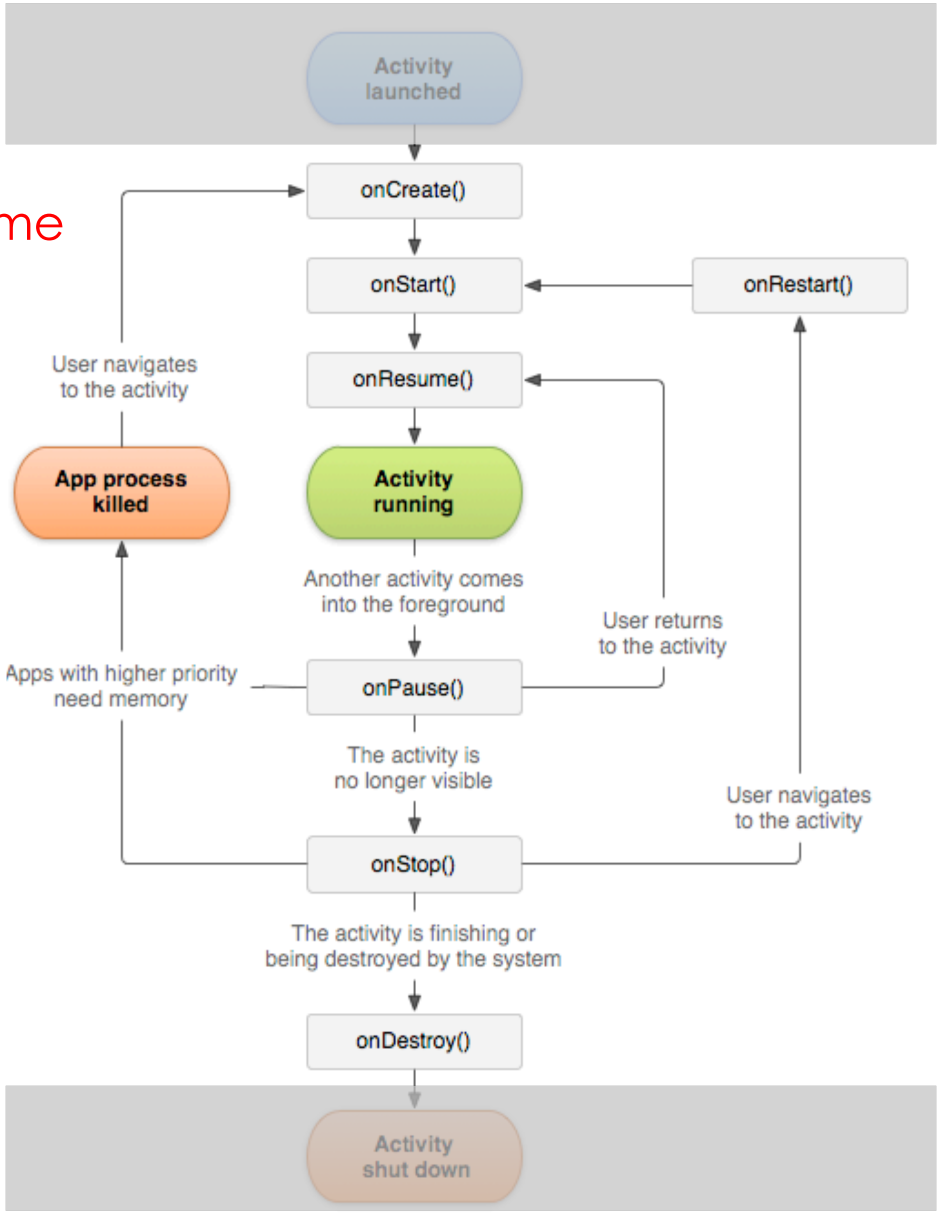
When the activity resumes, you can reacquire the necessary resources and resume actions that were interrupted.


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity lifecycle

The ENTIRE lifetime



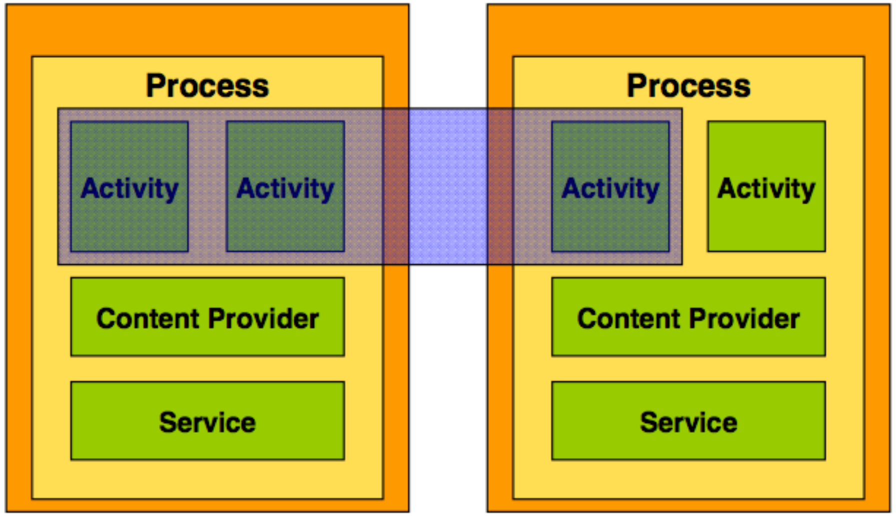

Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

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"

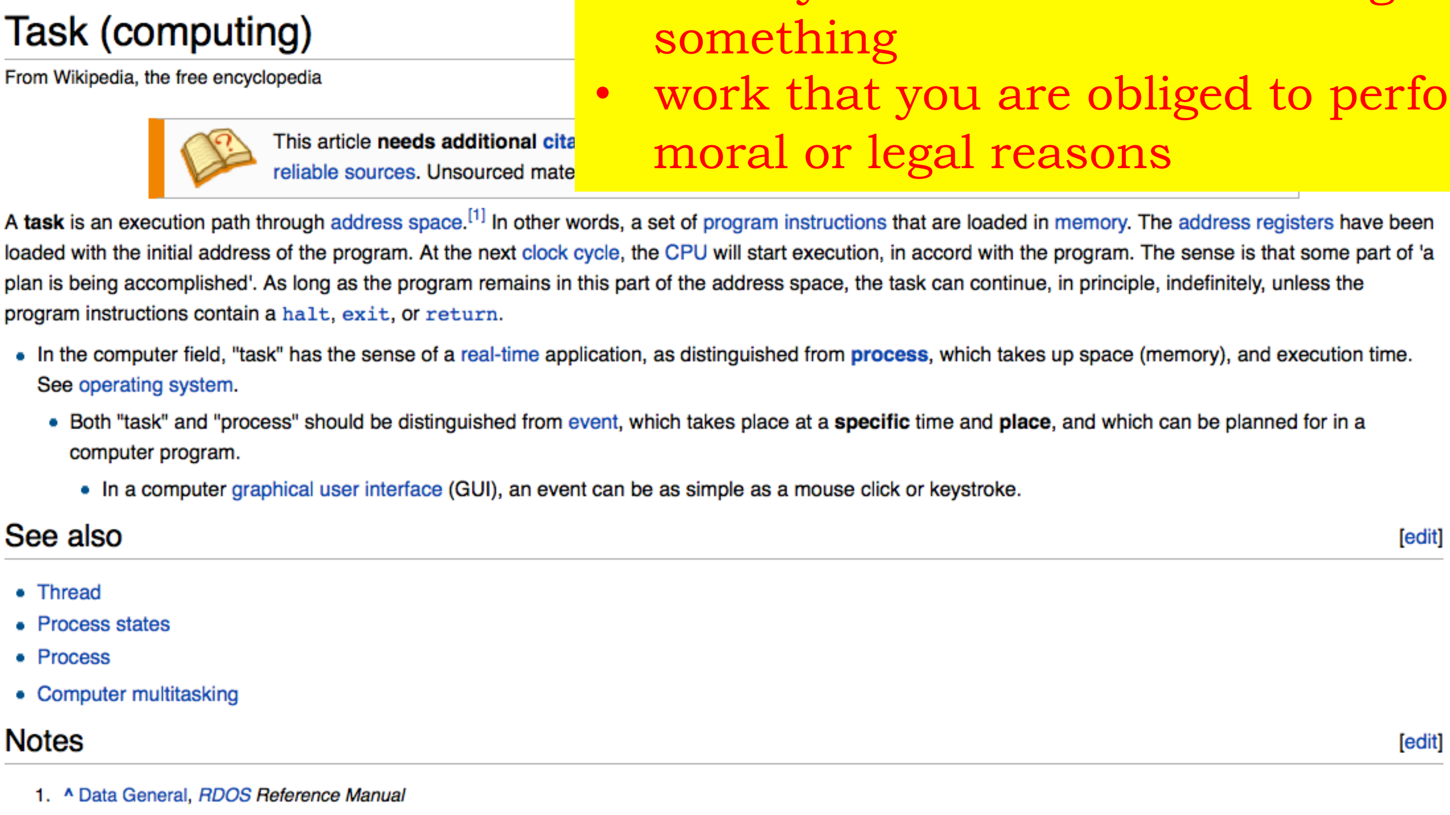


Task

Not exactly what you might imagine…

Wordnet definitions:

- activity directed toward making or doing something
- work that you are obliged to perform for moral or legal reasons



Task (computing)

From Wikipedia, the free encyclopedia

This article **needs additional cita**
reliable sources. Unsourced mate

A **task** is an execution path through address space.[1] In other words, a set of program instructions that are loaded in memory. The address registers have been loaded with the initial address of the program. At the next clock cycle, the CPU will start execution, in accord with the program. The sense is that some part of 'a plan is being accomplished'. As long as the program remains in this part of the address space, the task can continue, in principle, indefinitely, unless the program instructions contain a `halt`, `exit`, or `return`.

- In the computer field, "task" has the sense of a real-time application, as distinguished from **process**, which takes up space (memory), and execution time. See operating system.
  - Both "task" and "process" should be distinguished from event, which takes place at a **specific** time and **place**, and which can be planned for in a computer program.
    - In a computer graphical user interface (GUI), an event can be as simple as a mouse click or keystroke.

See also [edit]

- Thread
- Process states
- Process
- Computer multitasking

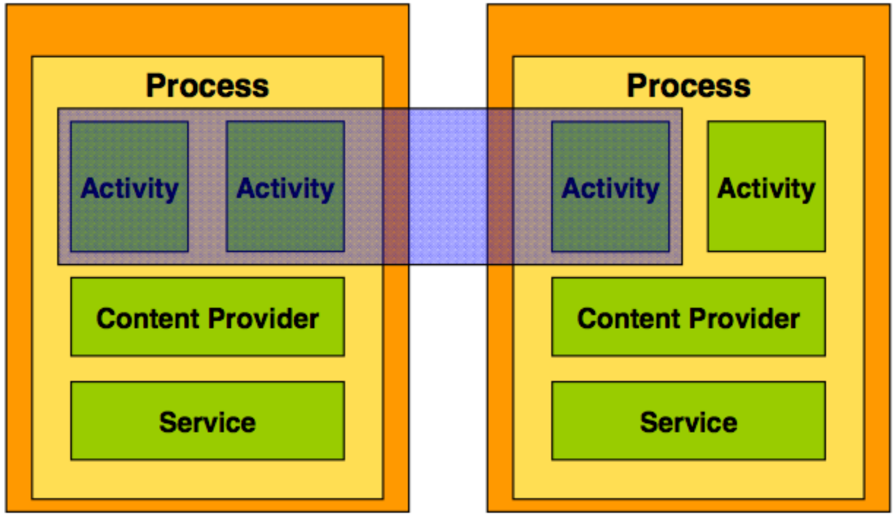
Notes [edit]

1. ^ Data General, *RDOS Reference Manual*

Tasks

Task (what users view as application)

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



Slide borrowed from Dominik Gruntz

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.

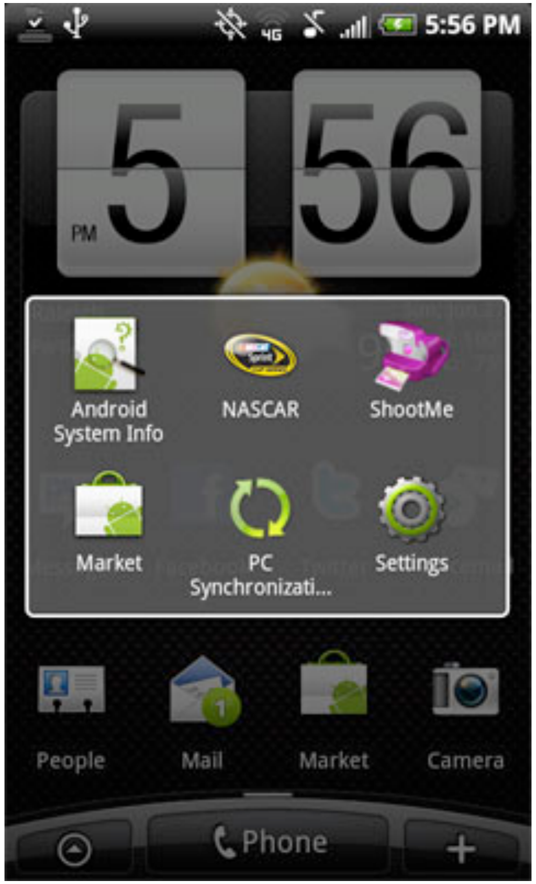
If no task exists for the application (the application has not been used recently), then a new task is created and the "main" activity for that application opens as the root activity in the stack.

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 scrollable set of open apps.



Tap the app you want to switch to.

Task Management

Default behavior:

New activity is added to the same task stack.

NOTE: Activity can have multiple instances, in different tasks or in the same task!

Google recommends:

“Let Android manage it for you. You do not need to bother with multitasking management!”

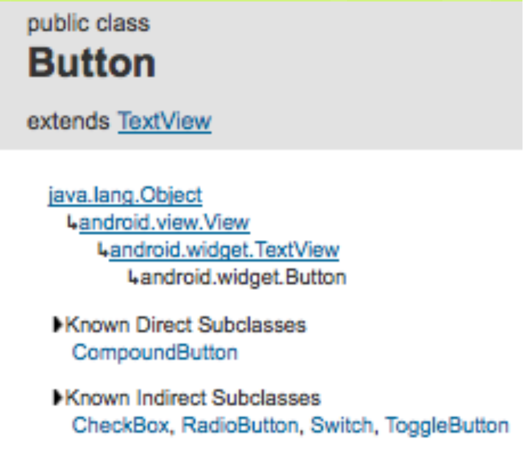
Process priorities

Active process	Critical priority
Visible process	High Priority
Started service process	
Background process	Low Priority
Empty process	

Basic UI elements: Android Buttons (Basics)

Marco Ronchetti
Università degli Studi di Trento

Let’s work with the listener



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

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

Anonymous Inner Class

In JavaFX it was

```
Button btn=…
btn.addEventHandler(new EventHandler() {
               public void handle(Event t) {
                         …;
               }
});
```

In Swing it was

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

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

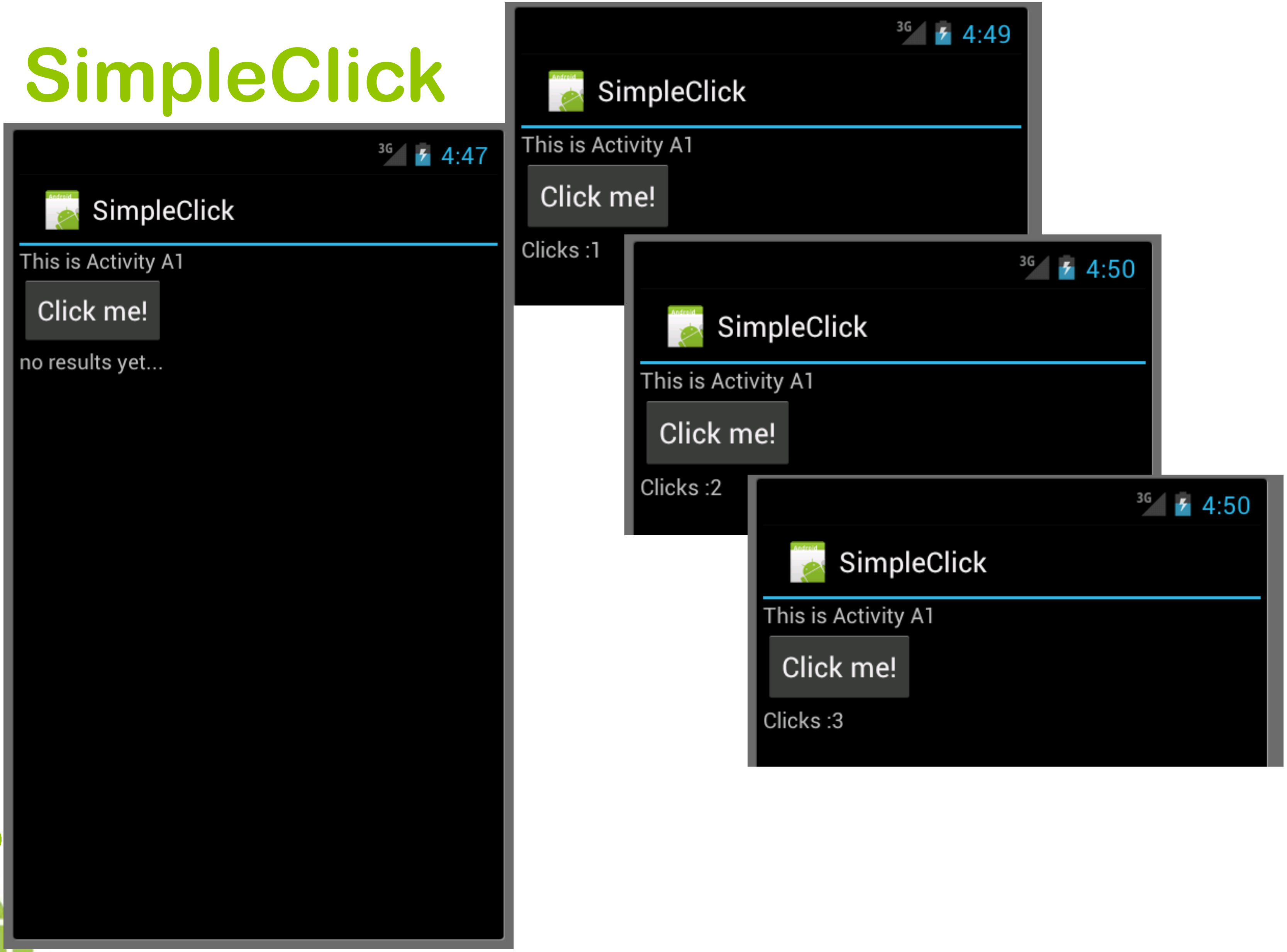
MAIN DIFFERENCE

In Swing (and in JavaFX):

The event
is passed

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

SimpleClick



4:47

SimpleClick

This is Activity A1

Click me!

no results yet...

4:49

SimpleClick

This is Activity A1

Click me!

Clicks :1

4:50

SimpleClick

This is Activity A1

Click me!

Clicks :2

4:50

SimpleClick

This is Activity A1

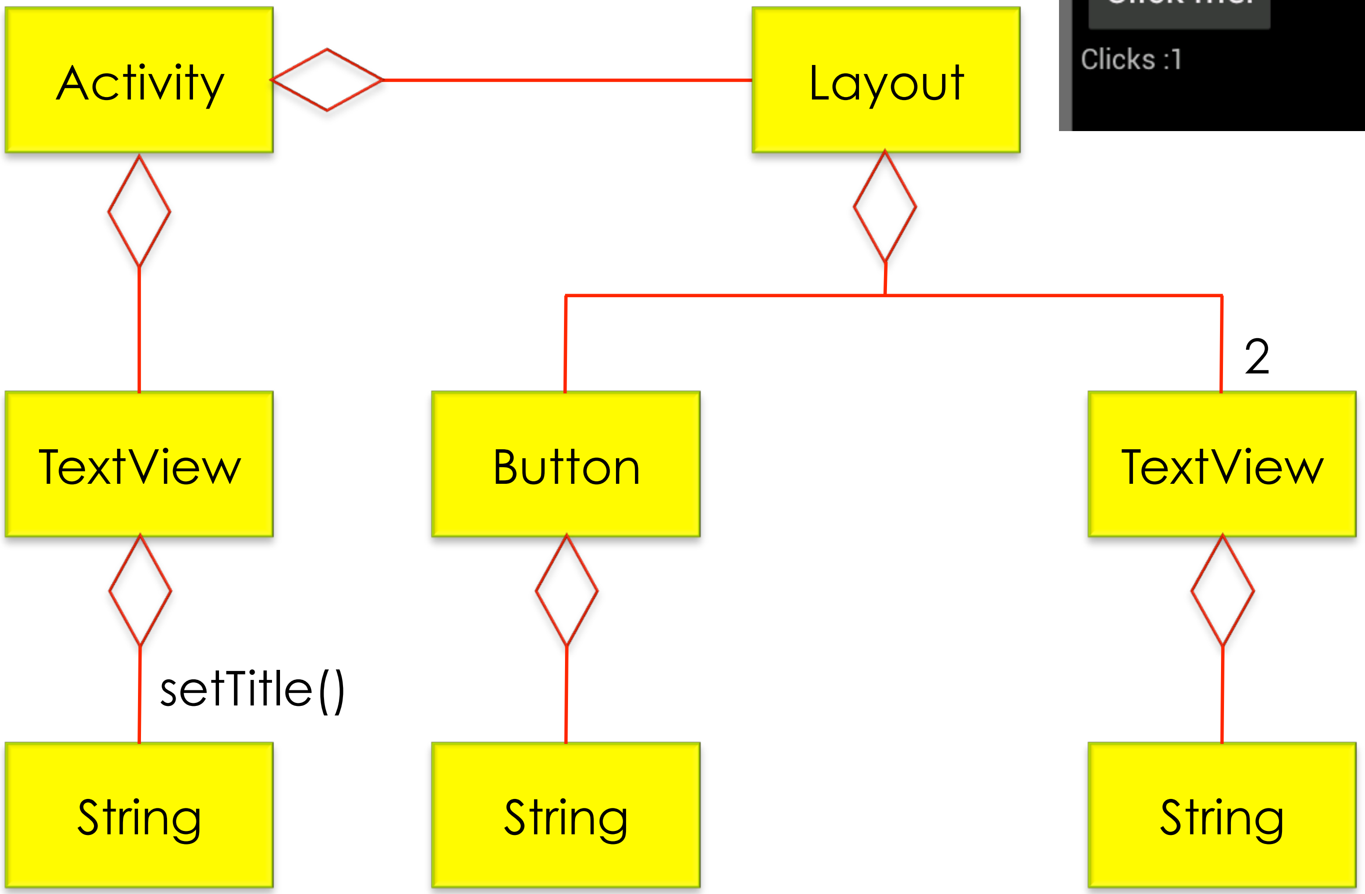
Click me!

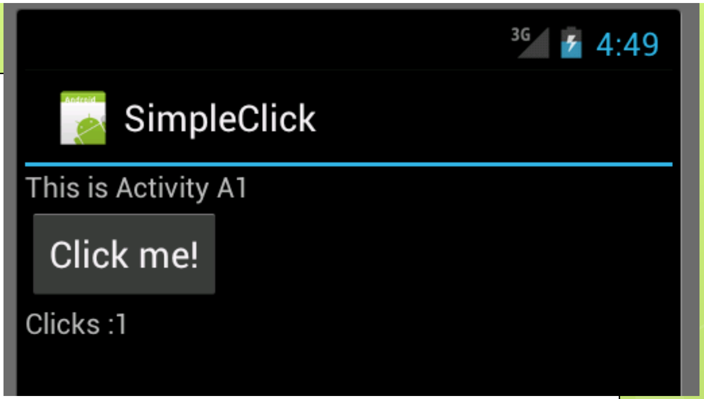
Clicks :3

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



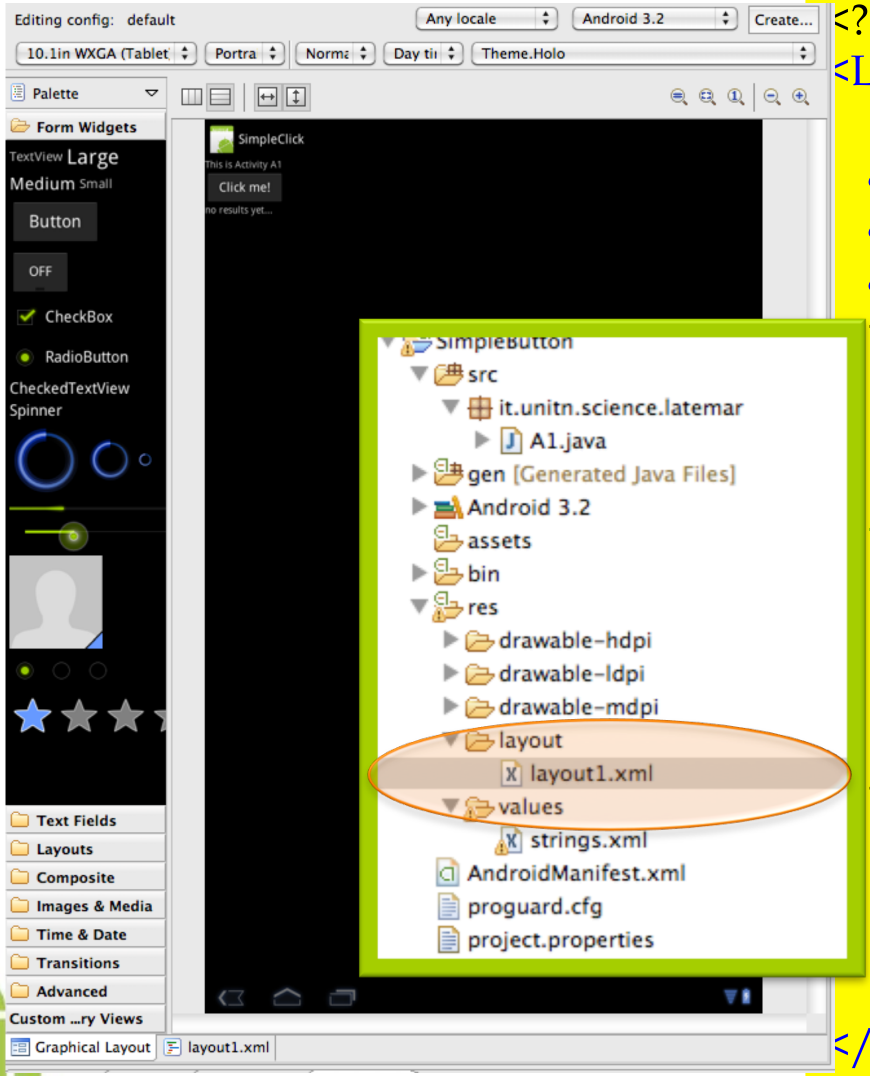


This is Activity A1

Click me!

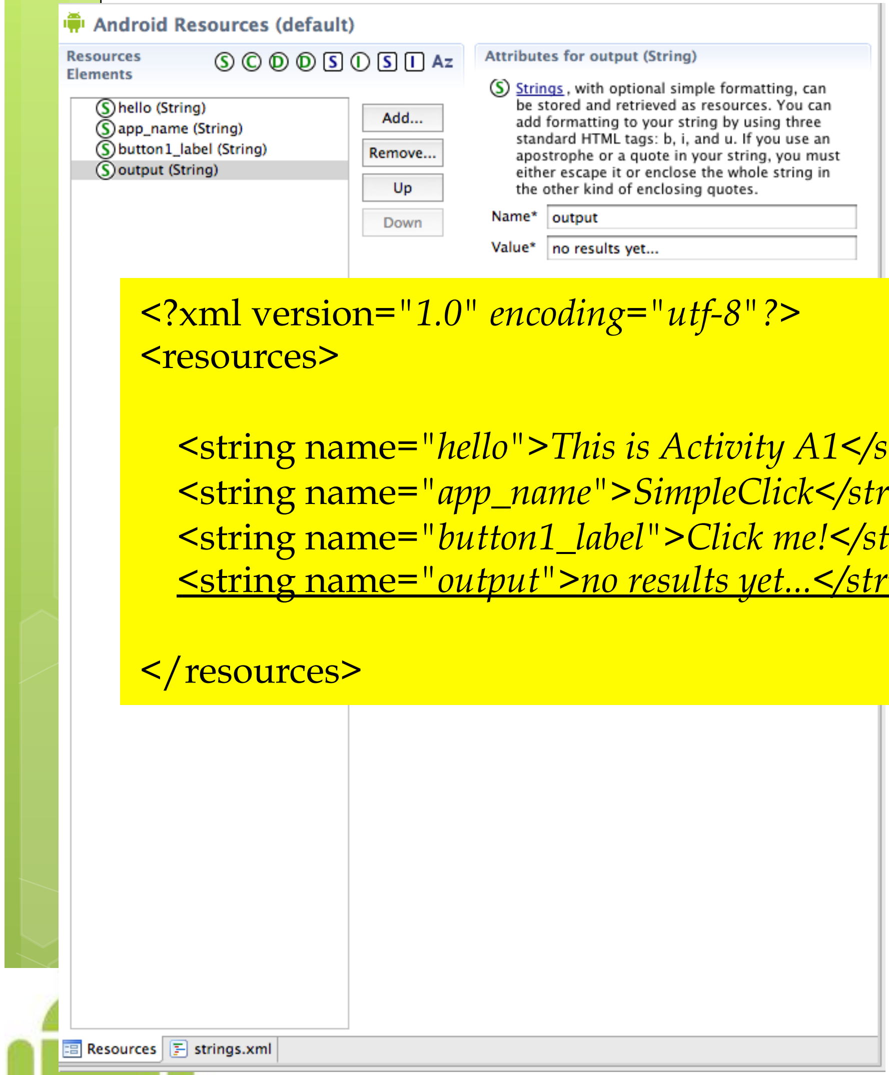
Clicks :1

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



Android Resources (default)

Resources Elements

- hello (String)
- app_name (String)
- button1_label (String)
- output (String)

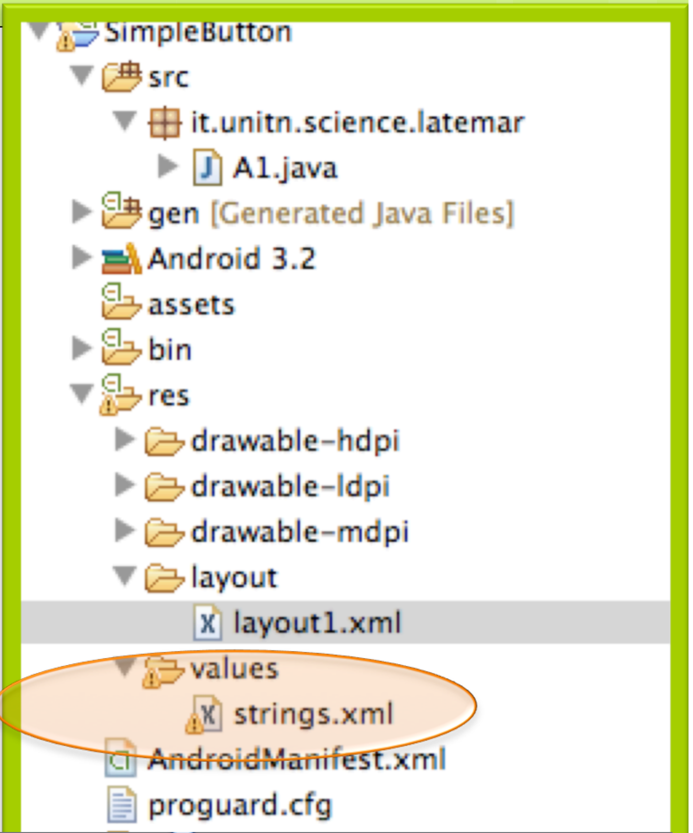
Attributes for output (String)

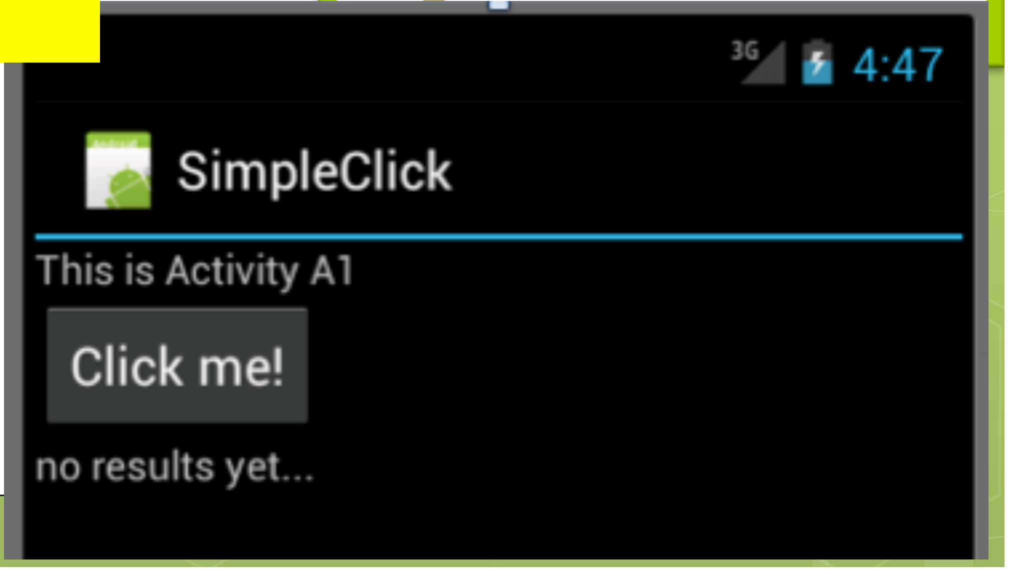
Strings, with optional simple formatting, can be stored and retrieved as resources. You can add formatting to your string by using three standard HTML tags: b, i, and u. If you use an apostrophe or a quote in your string, you must either escape it or enclose the whole string in the other kind of enclosing quotes.

Name* output

Value* no results yet...

```
<?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));
                     }
      });
   }
}
```

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

Calling Activities: Explicit Intents

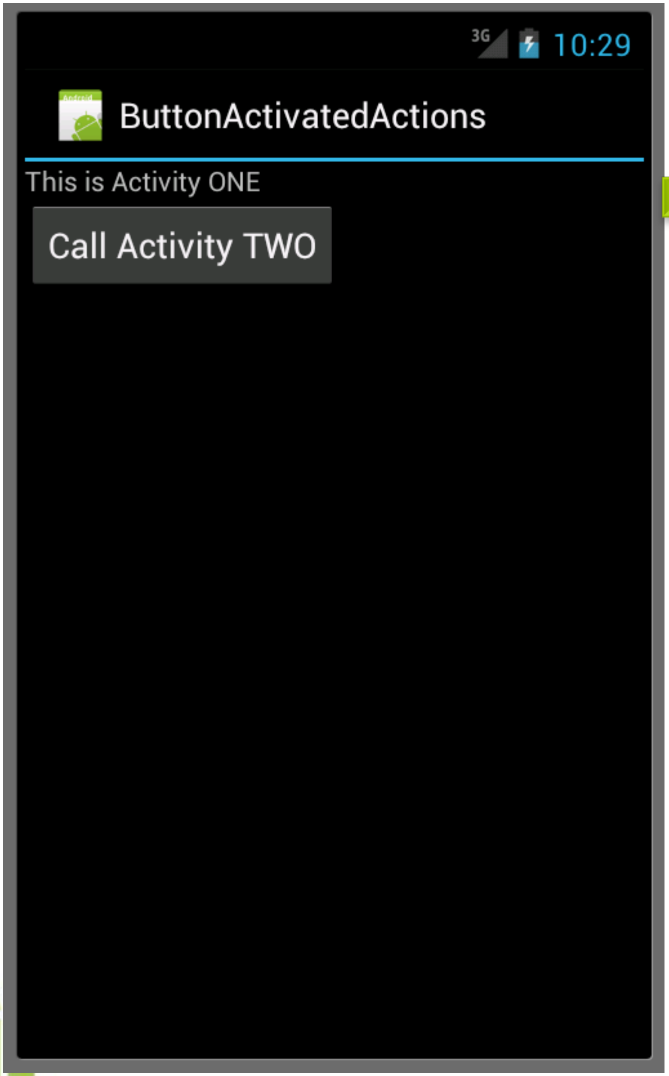
Marco Ronchetti
Università degli Studi di Trento

startActivity(Intent x)

startActivity(Intent x) (method of class Activity)

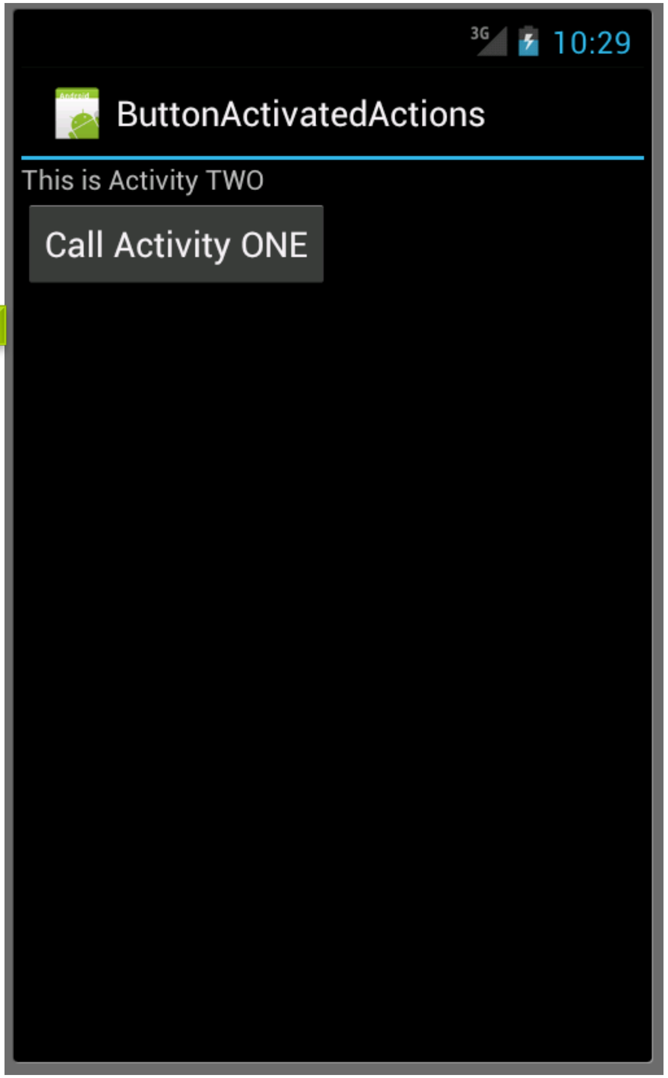
- starts a new activity, which will be placed at the top of the activity stack.
- takes a single argument which describes the activity to be executed.
- An intent is an abstract description of an operation to be performed.

A simple example: A1 calls A2



ButtonActivatedActions

This is Activity ONE



ButtonActivatedActions

This is Activity TWO

Explicit Intents

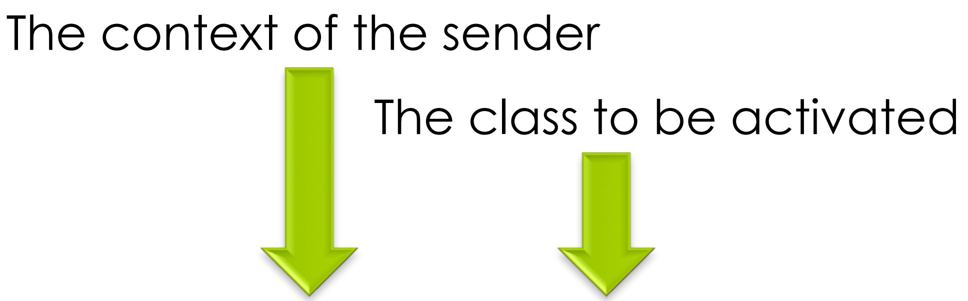
We will use the basic mode:

"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

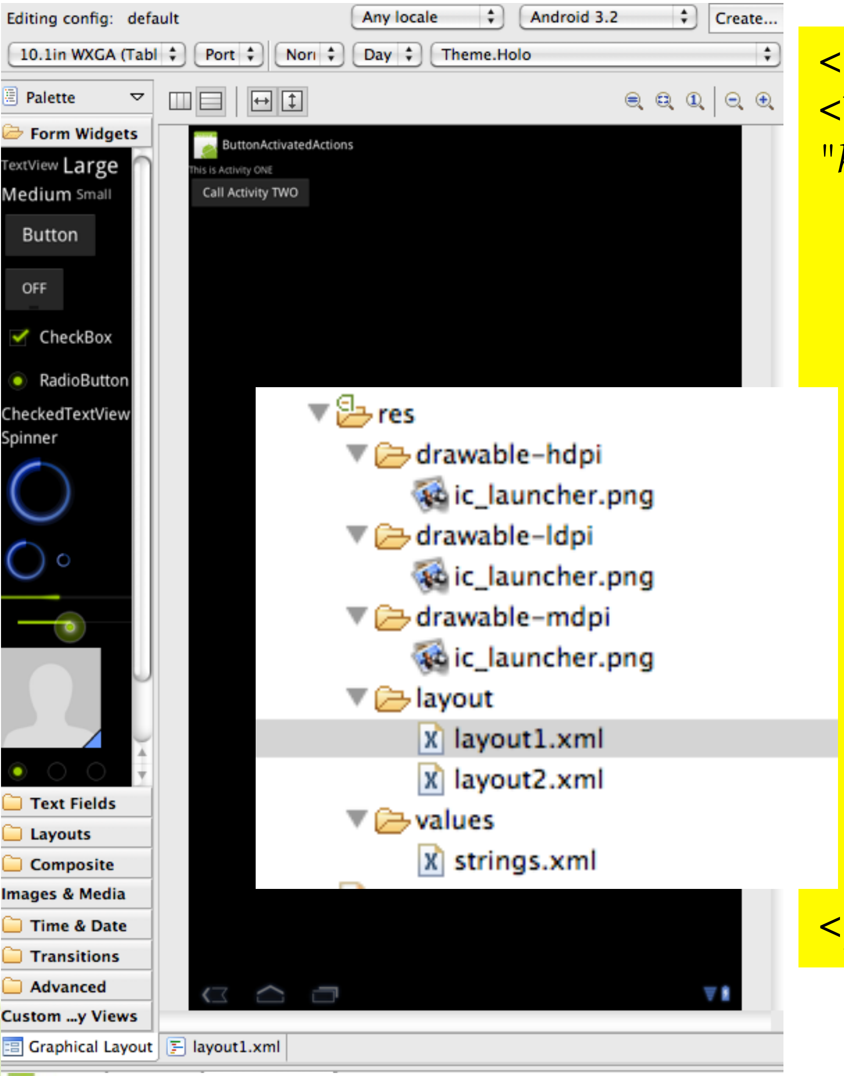
new Intent(Context c, Class c);

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

Equivalent form:

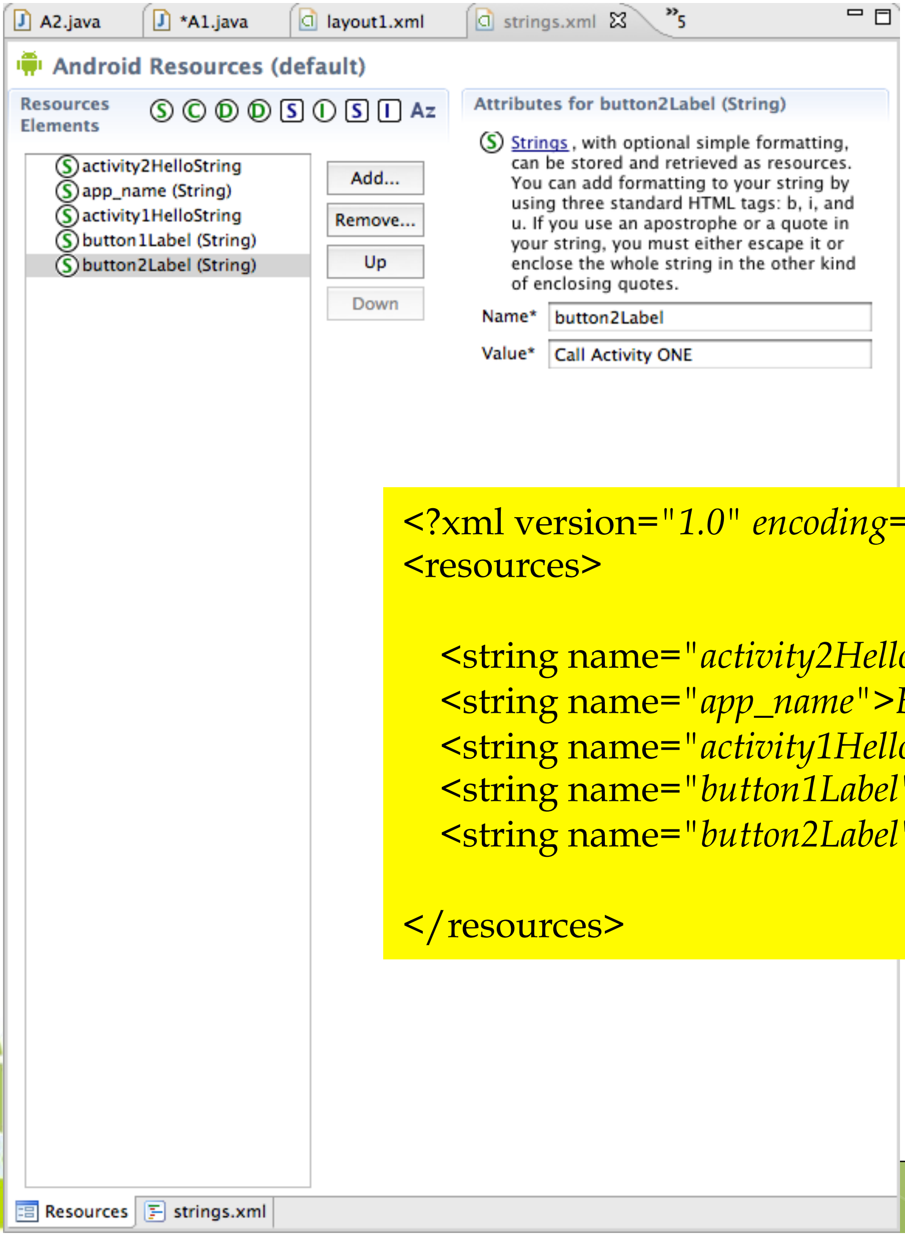
```
Intent i=new Intent();
i.setClass(Context c1, Class c2);
```

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



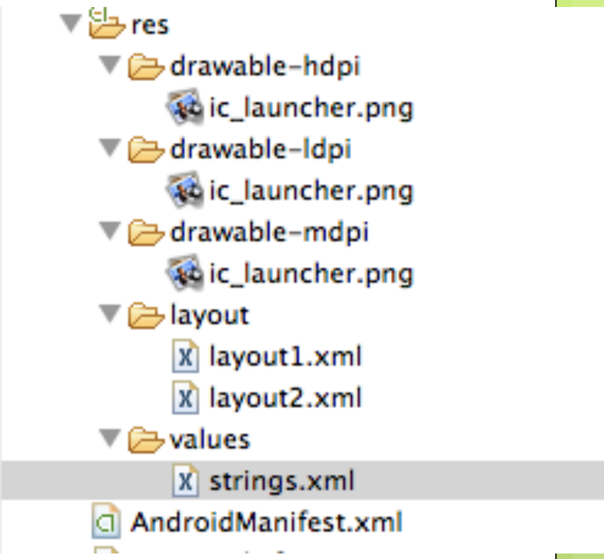
activity2HelloString
app_name (String)
activity1HelloString
button1Label (String)
button2Label (String)

Attributes for button2Label (String)

Strings, with optional simple formatting, can be stored and retrieved as resources. You can add formatting to your string by using three standard HTML tags: b, i, and u. If you use an apostrophe or a quote in your string, you must either escape it or enclose the whole string in the other kind of enclosing quotes.

Name* button2Label

Value* Call Activity ONE



```xml
<?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 icicle) {
     super.onCreate(icicle);

     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 icicle) {
     super.onCreate(icicle);

     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(A2.this, A1.class);
                     startActivity(intent);
                }
          });
   }
}
```

A1.this ? What's that?

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

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

3 different ways
to do the same thing

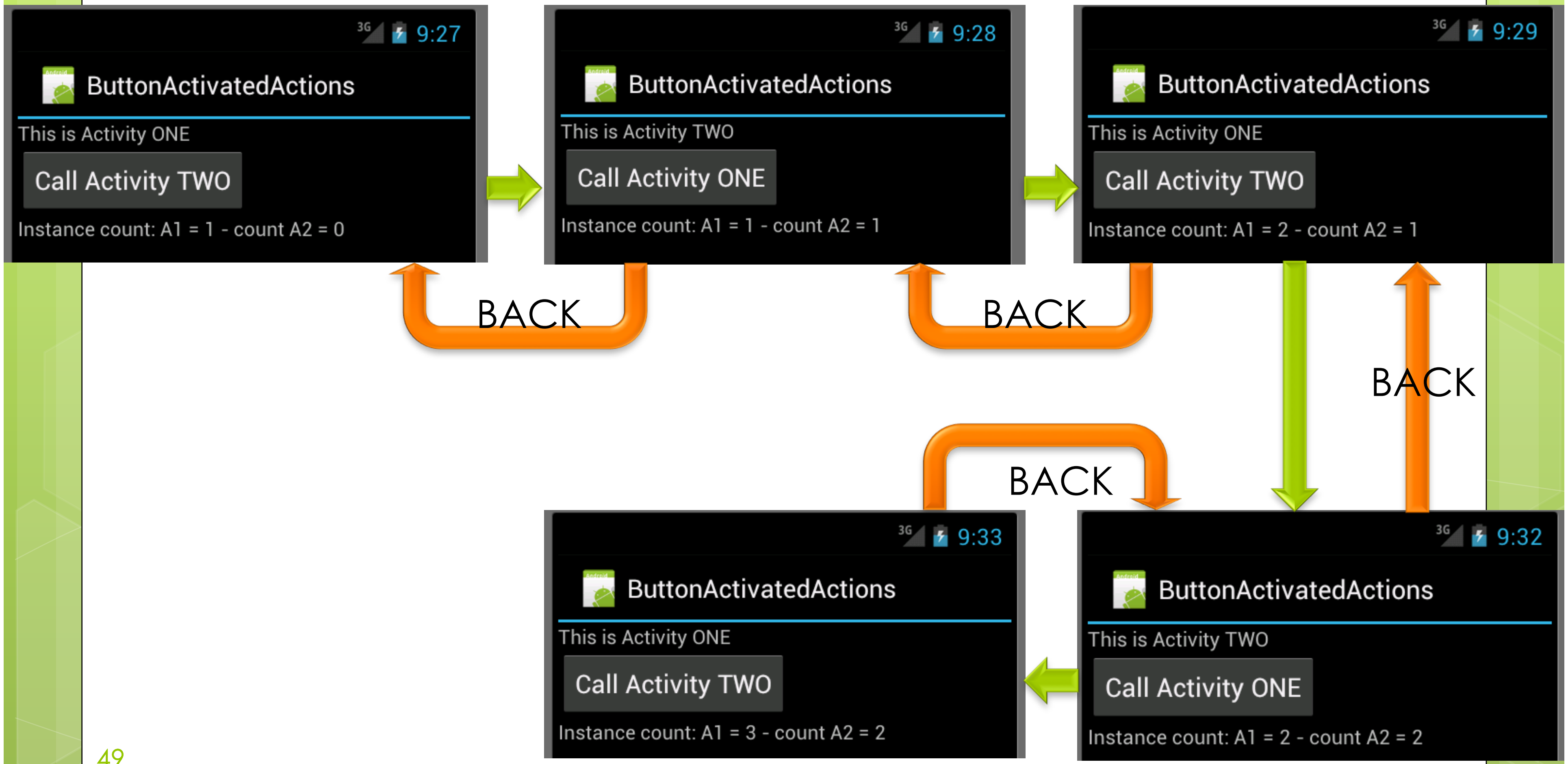
Let’s declare A2 in the manifest

```
<?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>
```

How many instances?

Marco Ronchetti
Università degli Studi di Trento

How many instances?



3G 9:27

ButtonActivatedActions

This is Activity ONE

Call Activity TWO

Instance count: A1 = 1 - count A2 = 0

3G 9:28

ButtonActivatedActions

This is Activity TWO

Call Activity ONE

Instance count: A1 = 1 - count A2 = 1

3G 9:29

ButtonActivatedActions

This is Activity ONE

Call Activity TWO

Instance count: A1 = 2 - count A2 = 1

3G 9:32

ButtonActivatedActions

This is Activity TWO

Call Activity ONE

Instance count: A1 = 2 - count A2 = 2

3G 9:33

ButtonActivatedActions

This is Activity ONE

Call Activity TWO

Instance count: A1 = 3 - count A2 = 2

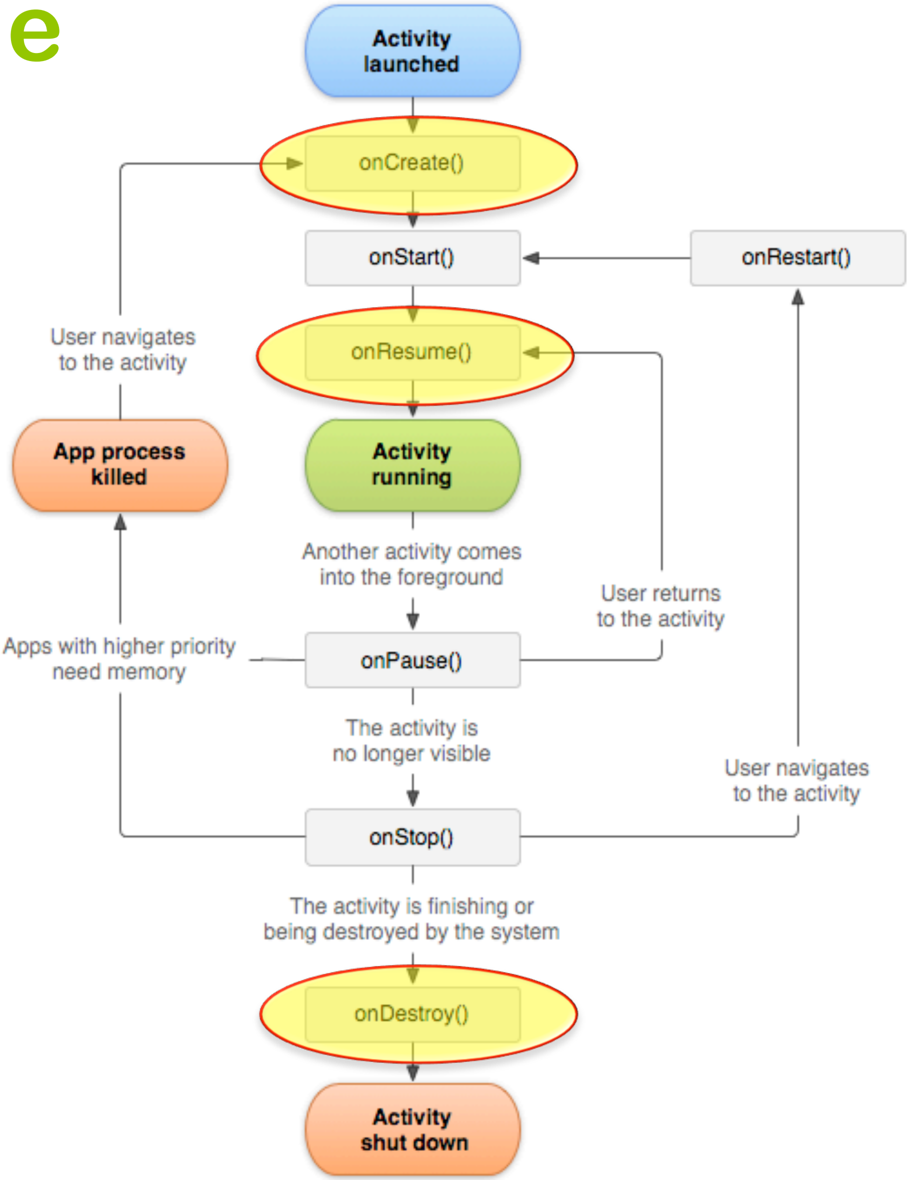
The code

```
public class A1 extends Activity {
  static int instances = 0;
  TextView tf = null;
  protected void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      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);
   }
}
```

Activity lifecycle

States (colored),
and
Callbacks (gray)


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

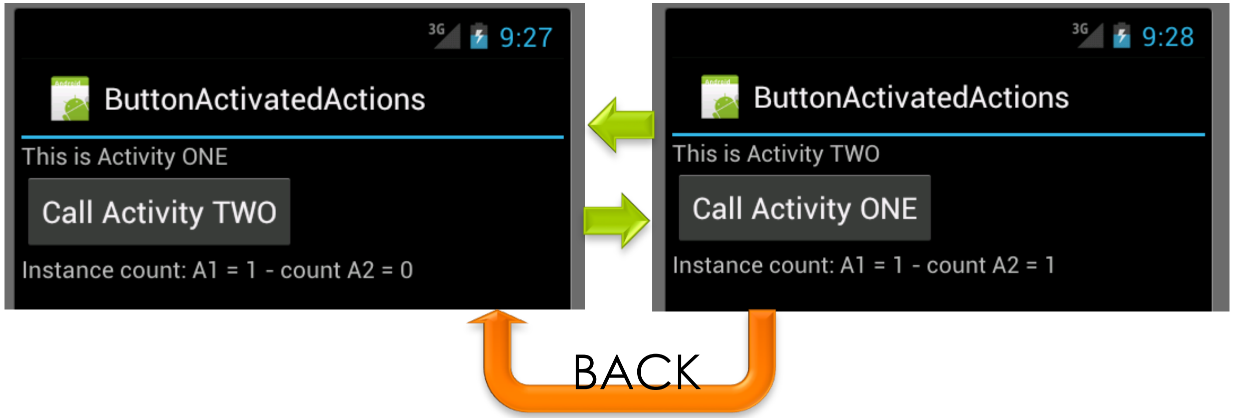
The xml

```
<?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 icicle) {
        super.onCreate(icicle);
        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);
           }
        });
}
```



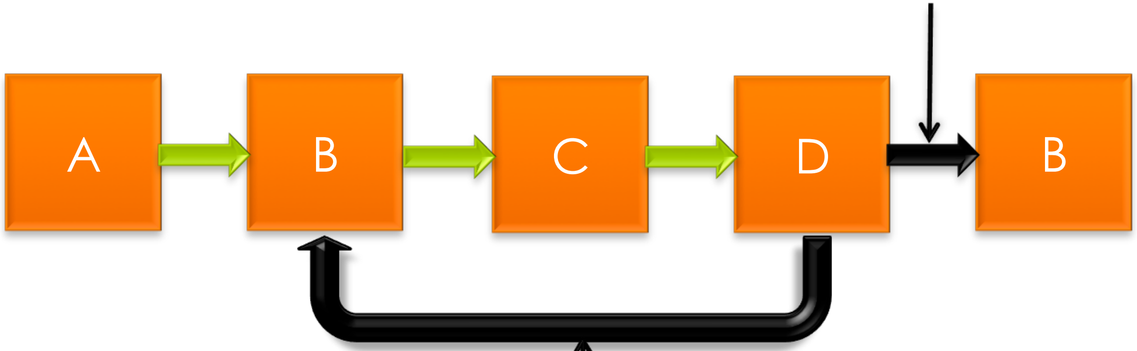
This is Activity ONE

Instance count: A1 = 1 - count A2 = 0

This is Activity TWO

Instance count: A1 = 1 - count A2 = 1

FLAG_ACTIVITY_CLEAR_TOP

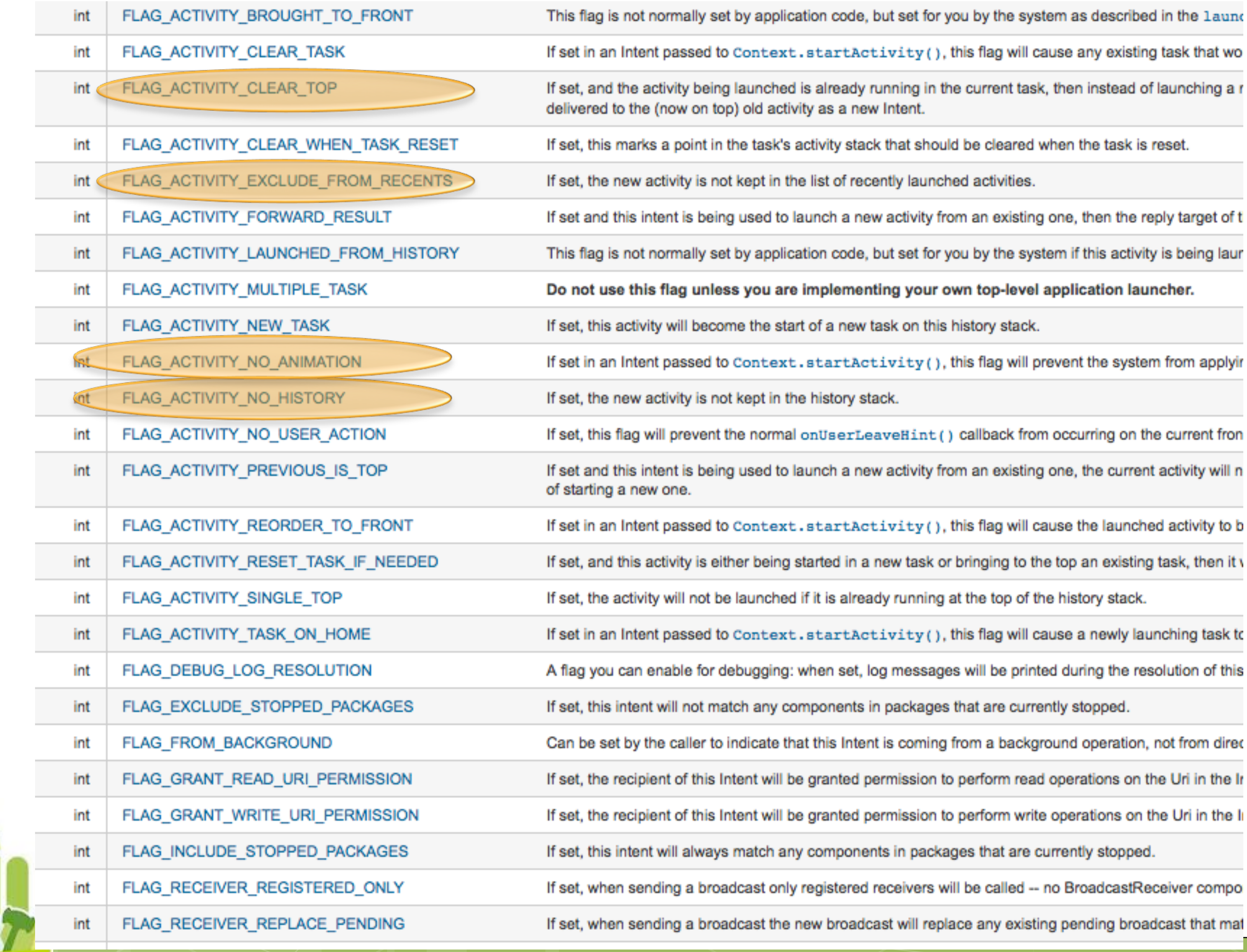


senza *FLAG_ACTIVITY_CLEAR_TOP*

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

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 `launc`
int	FLAG_ACTIVITY_CLEAR_TASK	If set in an Intent passed to `Context.startActivity()`, 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 laur
int	FLAG_ACTIVITY_MULTIPLE_TASK	**Do not use this flag unless you are implementing your own top-level application launcher.**
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 `Context.startActivity()`, 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 `onUserLeaveHint()` 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 `Context.startActivity()`, 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 `Context.startActivity()`, this flag will cause a newly launching task tc
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 Ir
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