



# Getting started: Installing IDE and SDK

Marco Ronchetti  
Università degli Studi di Trento

---

# Android Studio

<https://developer.android.com/studio>



2



# Tools behind the scenes

**dx**

- allows to convert Java .class files into .dex (Dalvik Executable) files.

**aapt** (Android Asset Packaging Tool)

- packs Android applications into an .apk (Android Package) file.

**adb** (Android debug bridge)

**ADT** (Android Development Tools for Eclipse)

- A development tool provided by Google to perform automatic conversion from .class to .dex files and to create the apk during deployment. It also provides debugging tools, and an Android device emulator.



# ADV - Android Virtual Device

An **emulator configuration** that lets you model an actual device by defining hardware and software options

An AVD consists of:

- **A hardware profile**
  - Defines the hardware features of the virtual device (whether it has a camera, a physical QWERTY keyboard or a dialing pad, how much memory it has etc.
- **A mapping to a system image:**
  - You can define what version of the Android platform will run on the virtual device
- Other options: the **emulator skin** (screen dimensions, appearance, etc.), **emulated SD card**
- A **dedicated storage area** on your development machine:
  - the device's user data (installed applications, settings, and so on) and emulated SD card are stored in this area.

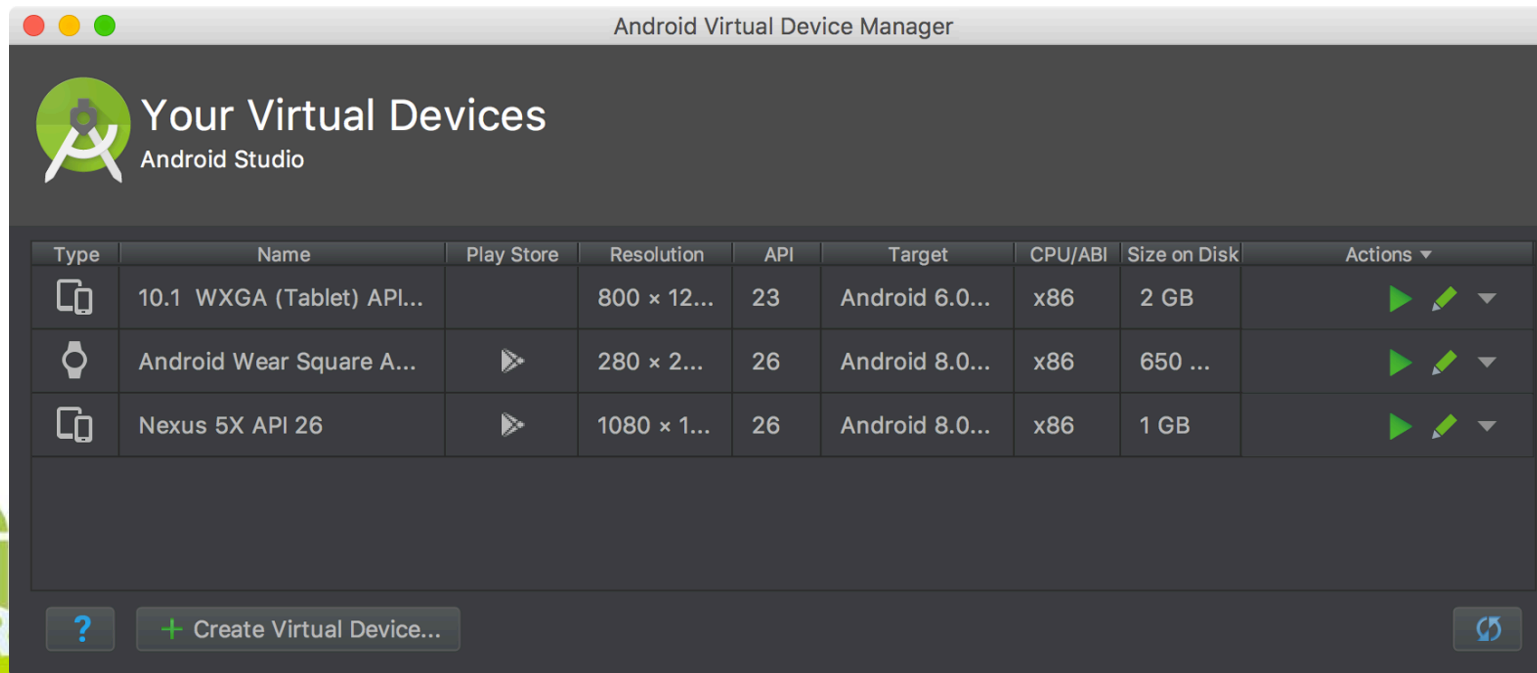


# ADV - Android Virtual Device

You create an AVD:

- from the command line (`$ android create avd`),
- Using ADV Tool on Android studio

<http://developer.android.com/guide/developing/devices/managing-avds-cmdline.html>





# Getting started: Hello Android

Marco Ronchetti  
Università degli Studi di Trento

---

# android.app.application

How shall we start?

First of all: there is no main...

But there is an "application" class in the API.  
(actually, `android.app.application`)

Probably we should `subclass` that, like we do with  
`javafx.application.Application` or with  
`javax.servlet.http.HttpServlet`?

7



# NO!

Application is a base class ONLY for keeping a global application state.

We need to subclass another thing: **Activity**





# HelloAndroid

```
package com.example.helloandroid;

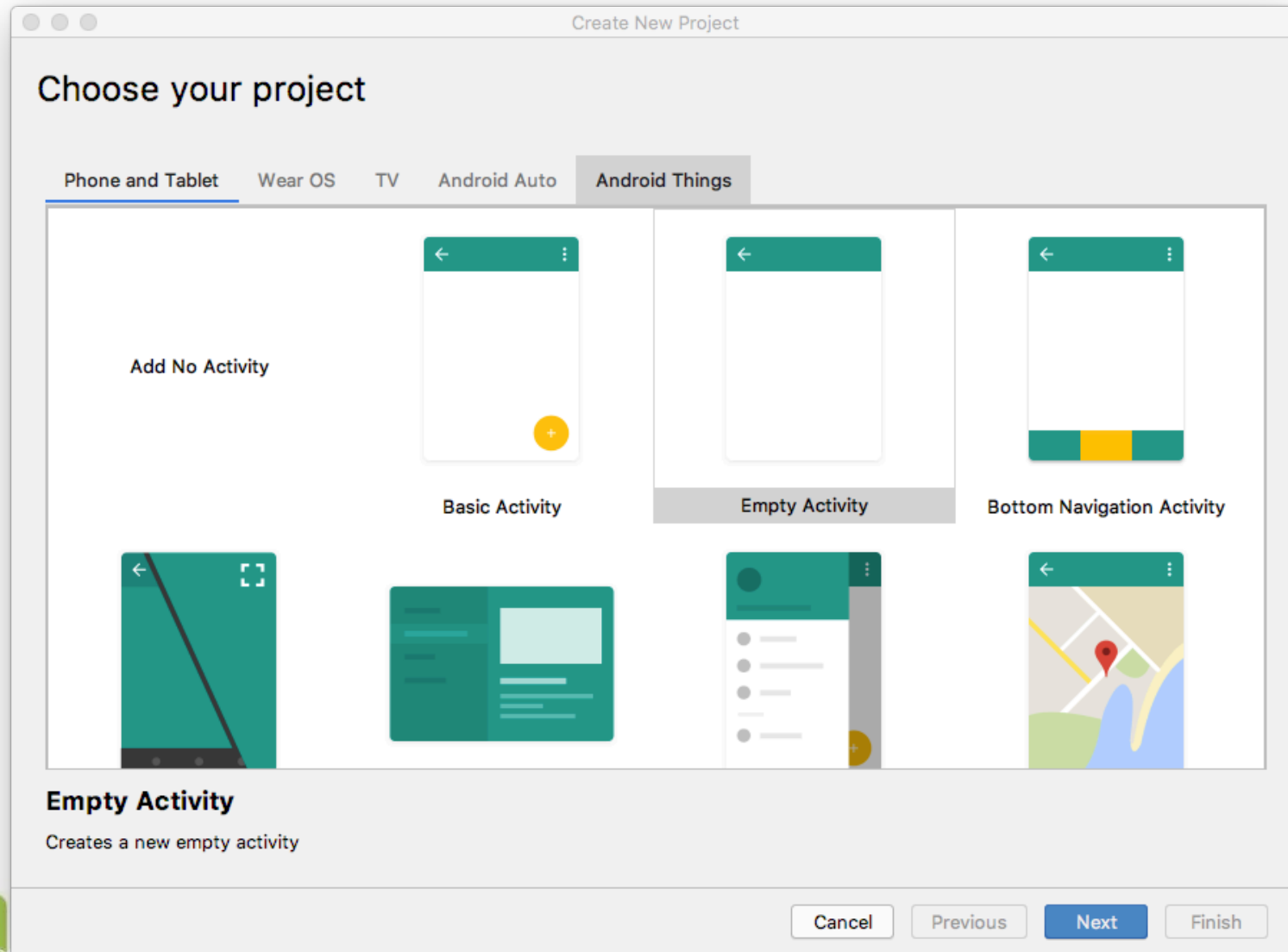
import android.app.Activity;
import android.os.Bundle;

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

7



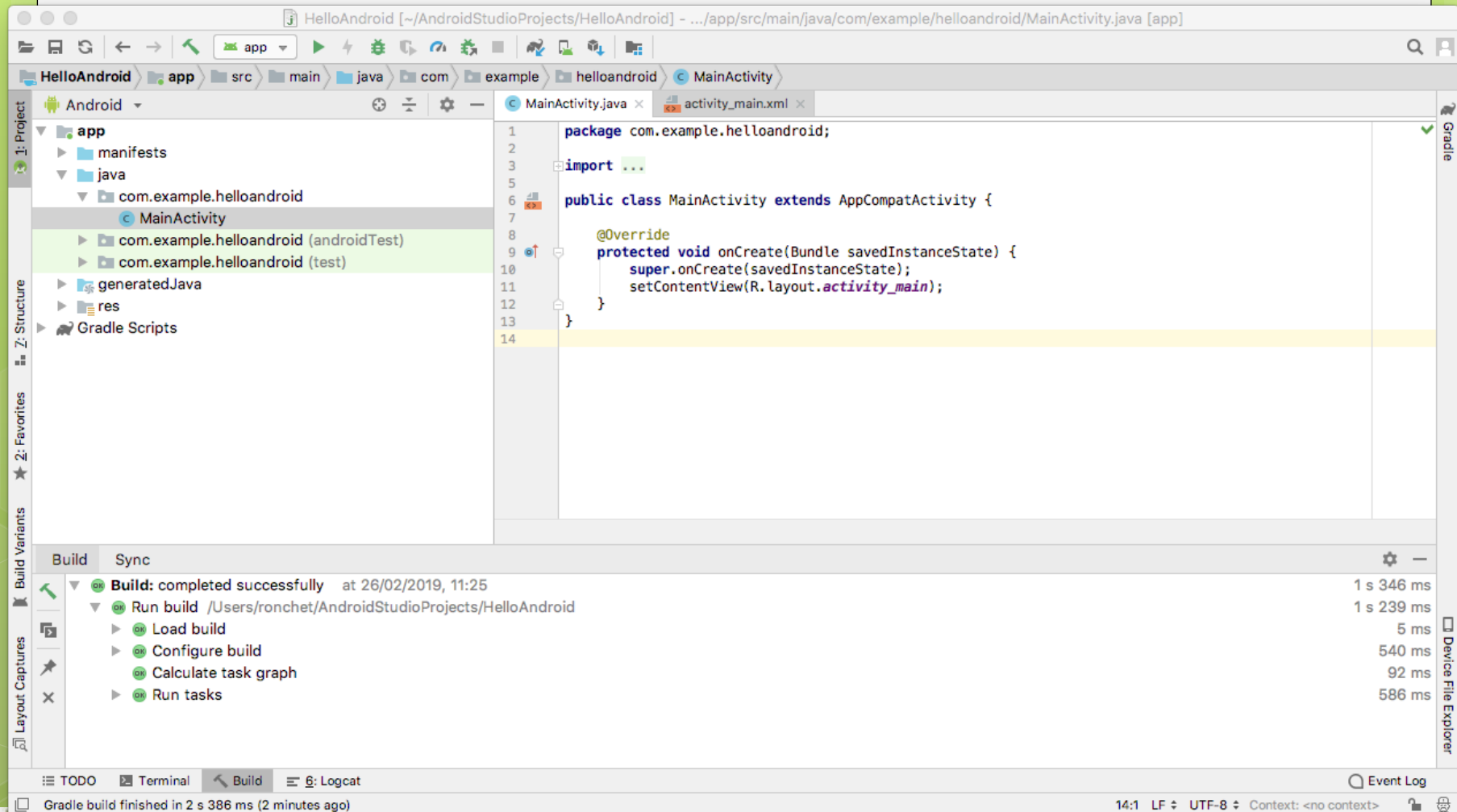
# Step 1a: creating project



10



# Step 2: editing the activity



The screenshot displays the Android Studio IDE with the following components:

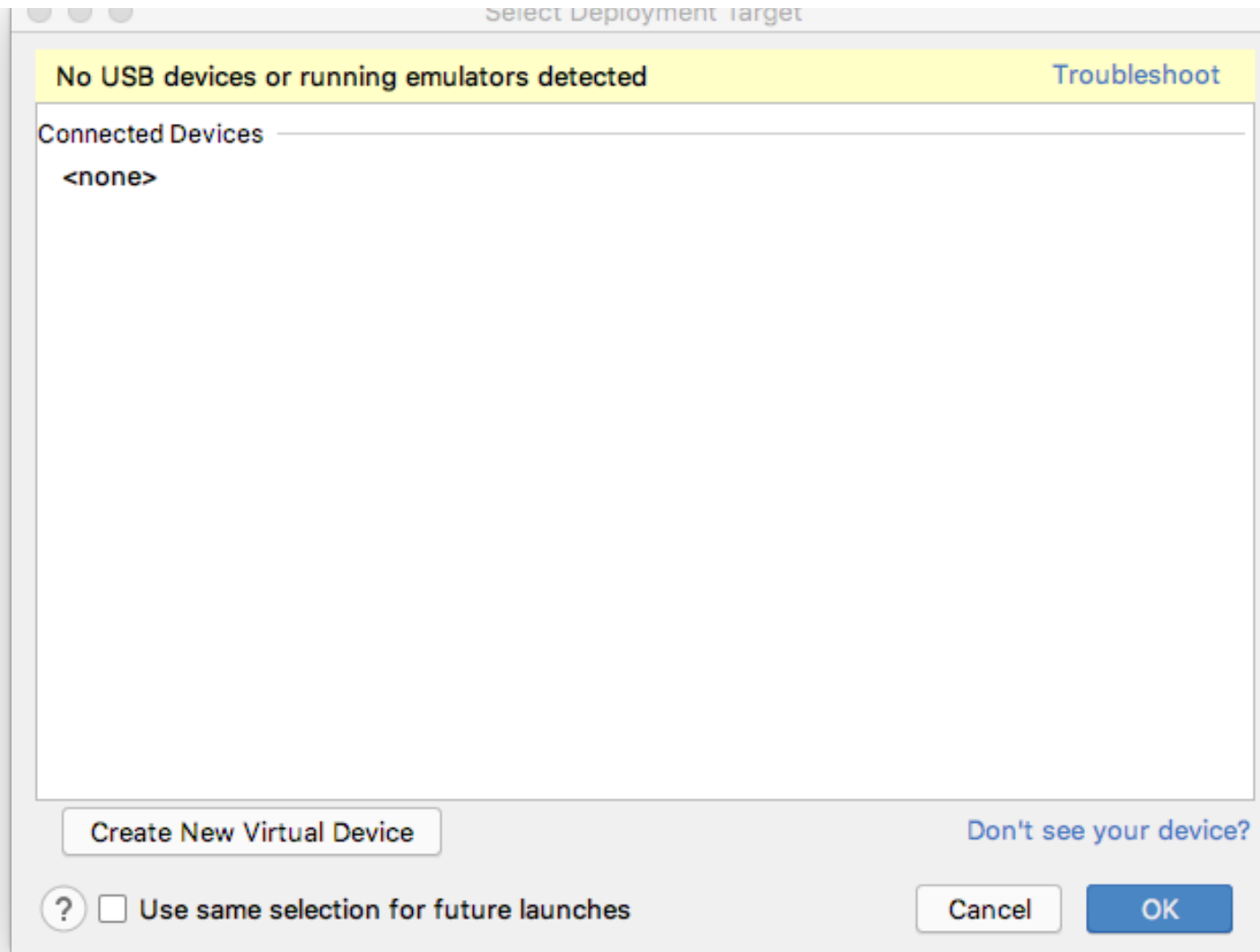
- Project View (Left):** Shows the project structure for 'HelloAndroid'. The 'app' module is expanded, showing 'src/main/java/com/example/helloandroid/MainActivity.java'.
- Code Editor (Center):** Displays the content of MainActivity.java:

```
1 package com.example.helloandroid;
2
3 import ...
4
5 public class MainActivity extends AppCompatActivity {
6
7     @Override
8     protected void onCreate(Bundle savedInstanceState) {
9         super.onCreate(savedInstanceState);
10        setContentView(R.layout.activity_main);
11    }
12 }
13
14
```
- Build Output Window (Bottom):** Shows the build process results:

```
Build: completed successfully at 26/02/2019, 11:25
Run build /Users/ronchet/AndroidStudioProjects/HelloAndroid
  Load build
  Configure build
  Calculate task graph
  Run tasks
```
- Toolbar (Top):** Contains standard IDE icons for file operations, running, and debugging.
- Status Bar (Bottom):** Indicates the current file is MainActivity.java, the encoding is UTF-8, and the context is <no context>.



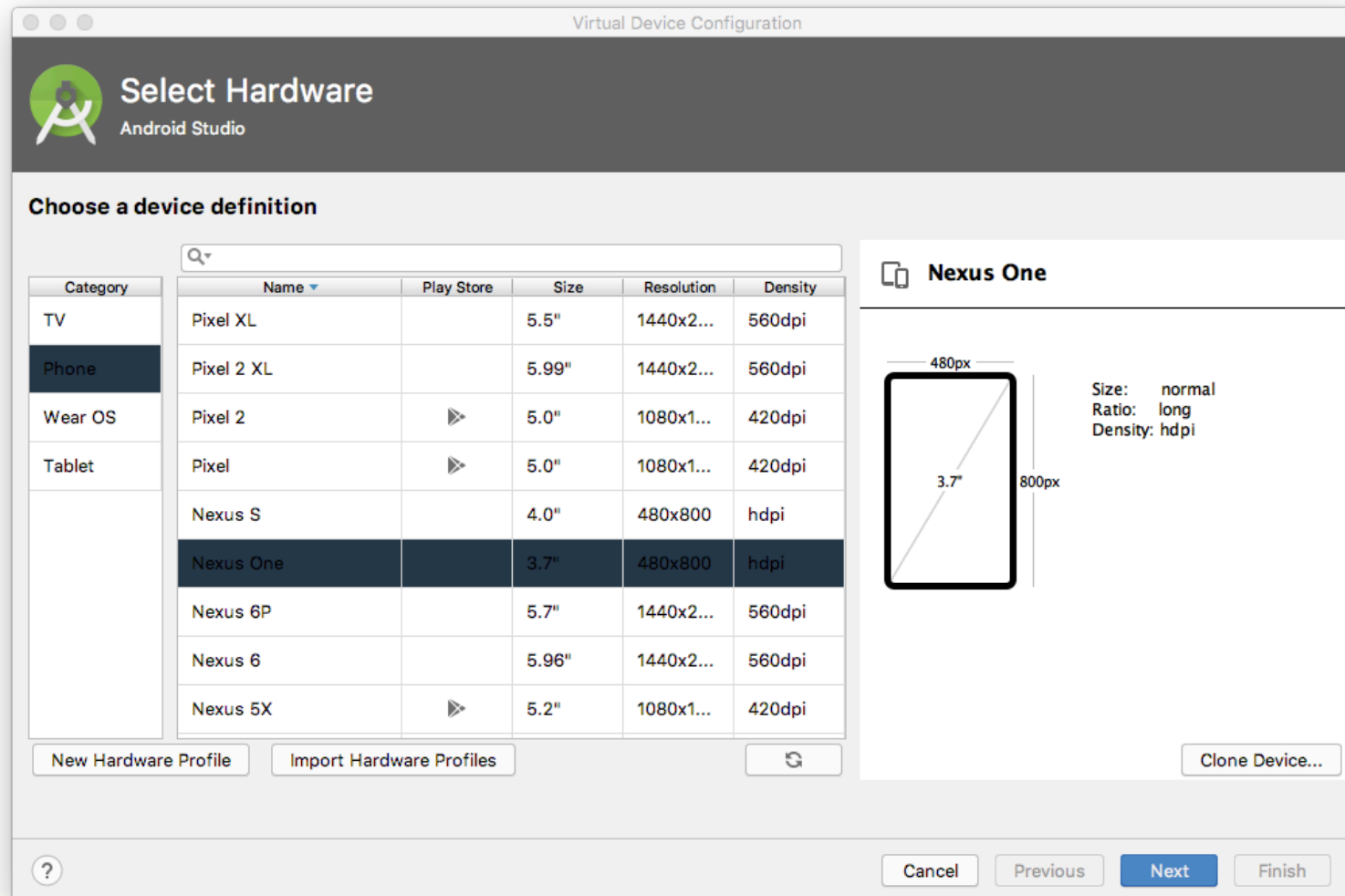
## Step 2a: installing an emulator



12



# Step 3b:




13



# Step 3c:

Virtual Device Configuration




System Image  
Android Studio

Select a system image

Recommendedx86 ImagesOther Images

Release Name	API Level	ABI	Target
<a href="#">Pie Download</a>	28	x86	Android 9.0 (Google APIs)
<a href="#">Oreo Download</a>	27	x86	Android 8.1 (Google APIs)
<a href="#">Oreo Download</a>	26	x86	Android 8.0 (Google APIs)
<a href="#">Nougat Download</a>	25	x86	Android 7.1.1 (Google APIs)
<a href="#">Nougat Download</a>	24	x86	Android 7.0 (Google APIs)
<b>Marshmallow</b>	<b>23</b>	<b>x86</b>	<b>Android 6.0 (Google APIs)</b>
<a href="#">Lollipop Download</a>	22	x86	Android 5.1 (Google APIs)

Marshmallow



API Level

**23**

Android

**6.0**

Google Inc.

System Image

**x86**

We recommend these images because they run the fastest and support Google APIs.

Questions on API level?  
[See the API level distribution chart](#)

?


CancelPreviousNextFinish

14




# Step 3c:


Virtual Device Configuration

 **Android Virtual Device (AVD)**  
Android Studio



**Verify Configuration**

AVD Name

 Nexus One 3.7 480x800 hdpi

 Marshmallow Android 6.0 x86

Startup orientation

 Portrait  Landscape

Emulated Performance Graphics:

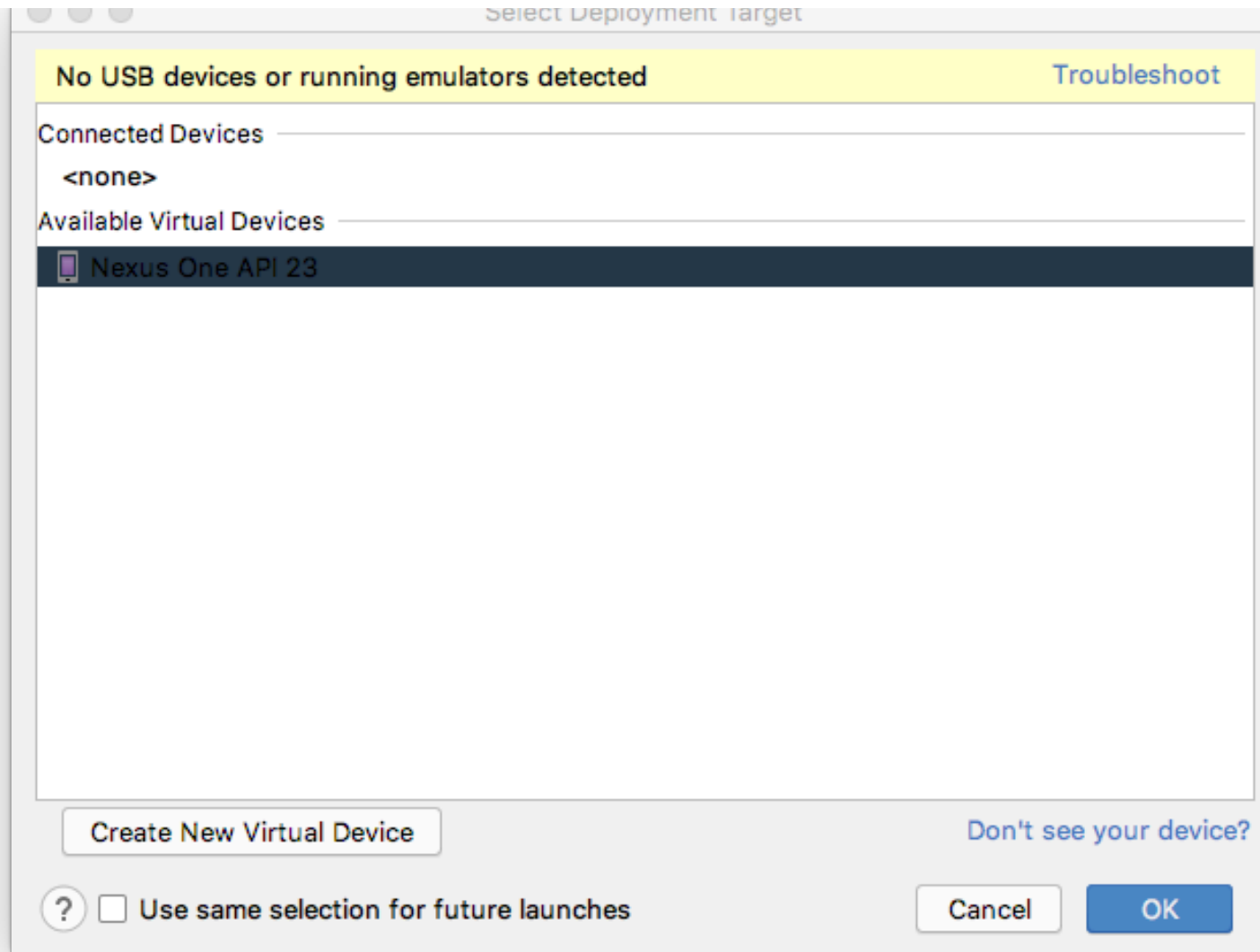
**AVD Name**

The name of this AVD.

15



## Step 3c:



16





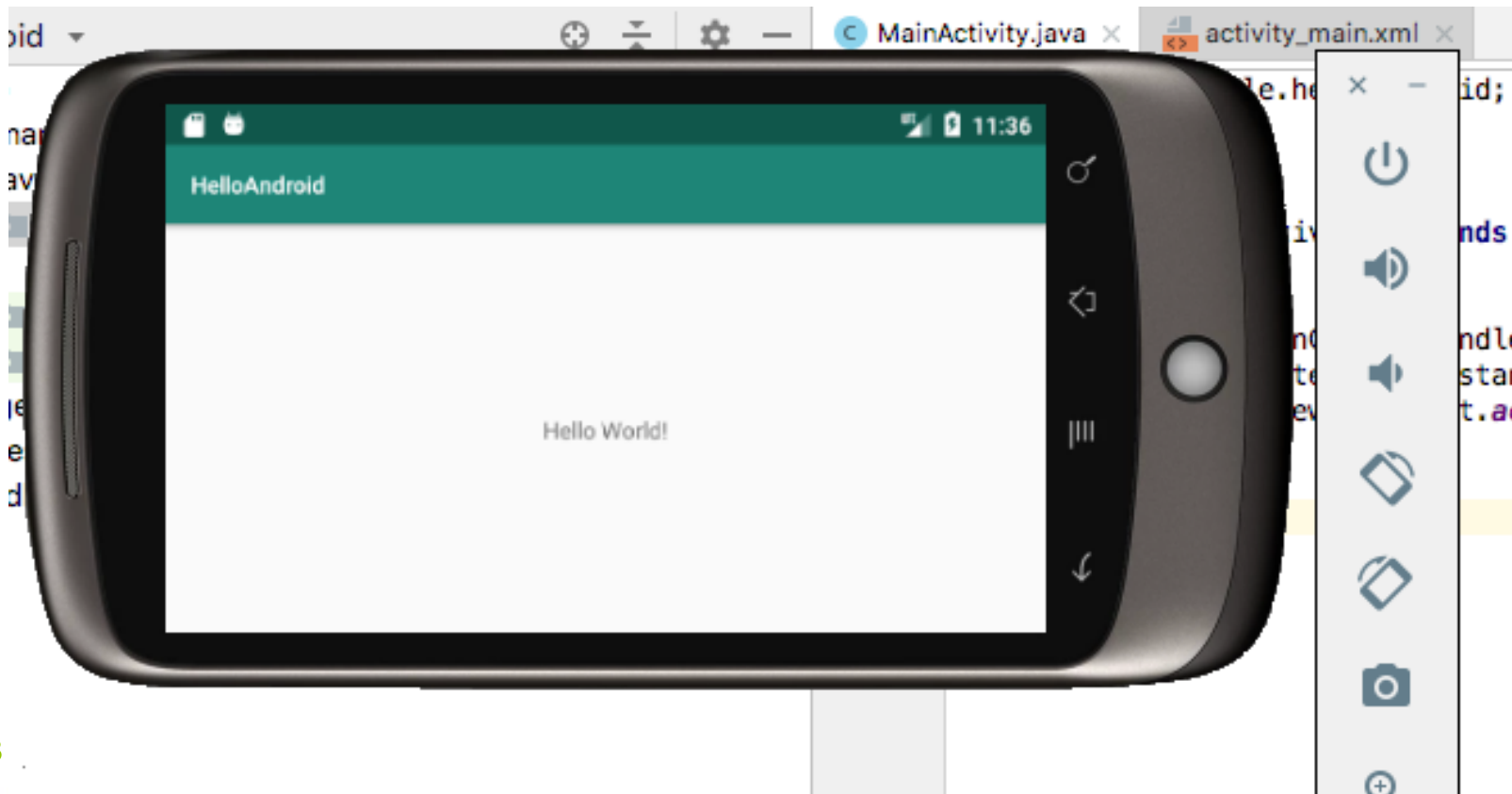
## Step 3c:



17



# And finally we run the project!



18



# Gradle? Kotlin?

**Gradle** is an open-source build automation system that builds upon the concepts of **Apache Ant** and **Apache Maven** and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven for declaring the project configuration.

**Groovy** is a Java-syntax-compatible object-oriented programming language for the Java platform.

**Kotlin** is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin is designed to interoperate fully with Java, and the JVM version of its standard library depends on the Java Class Library, but type inference allows its syntax to be more concise.



# 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);
    }
}
```

20



# 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);  
    }  
}
```

21



```
        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
```

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()?"  
Somehow similar to "Stage" and "Scene" in JavaFX.**



## 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.



# 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;  
    }  
}
```

27



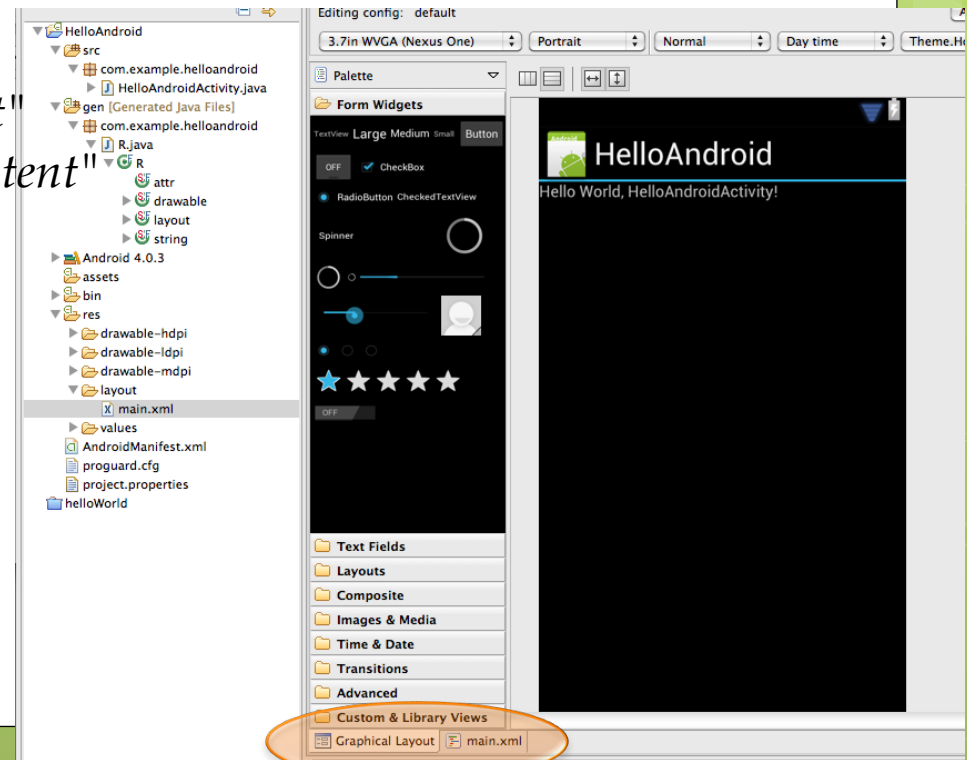
# Res/layout/main.xml

```
<?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>
```

28



## 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:

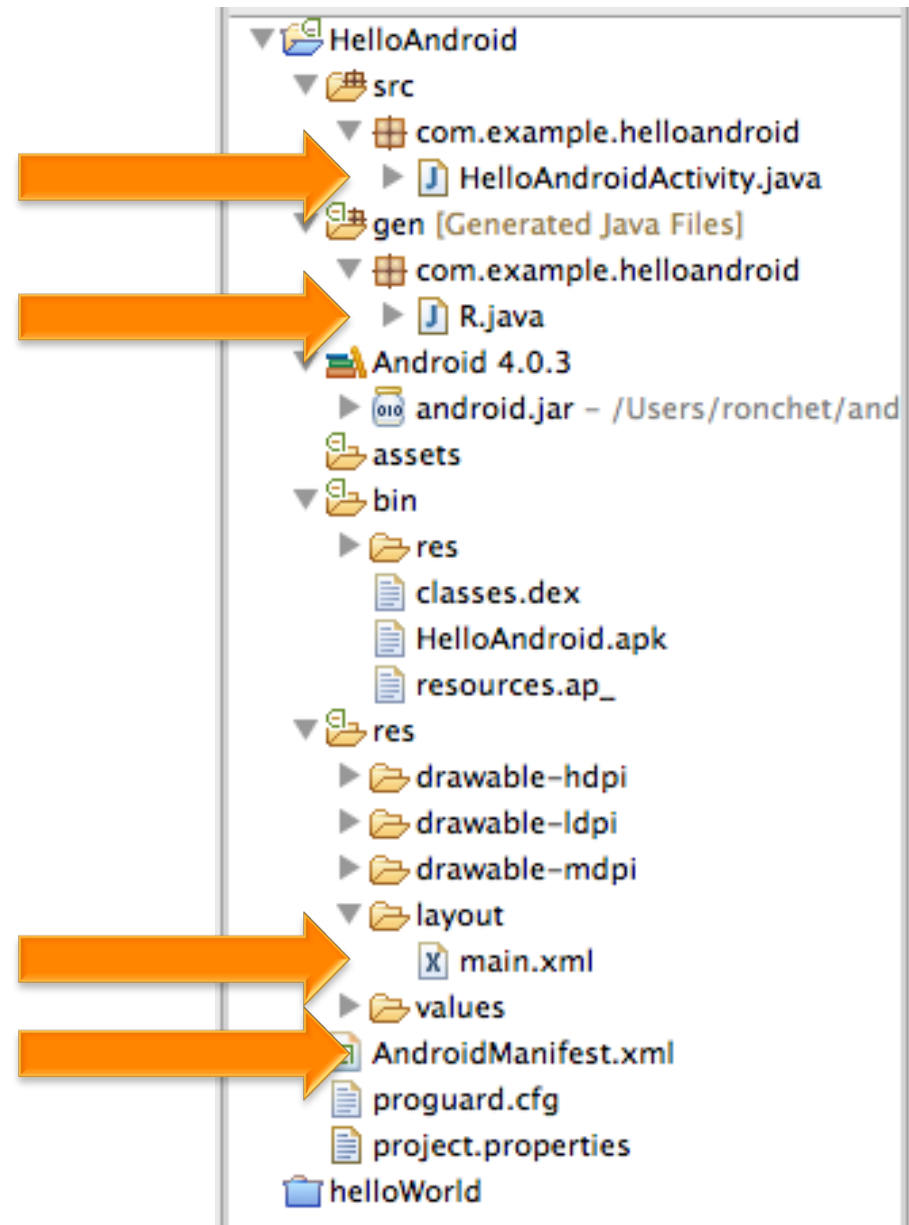
[Button](#), [CheckedTextView](#), [Chronometer](#), [DigitalClock](#), [EditText](#)

30

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

Nov. 2011

Feb 2011

Dic 2010

Mag 2010

32





# 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
```





# 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.

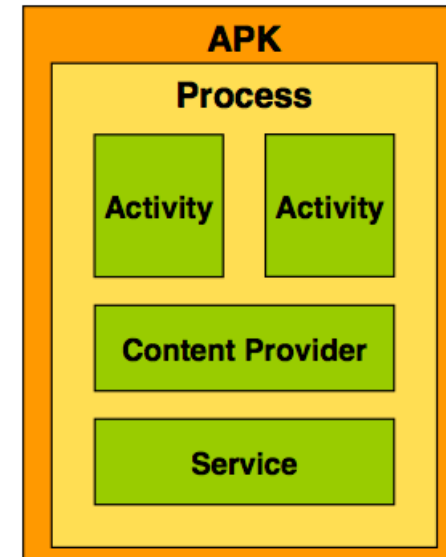


# Peeking into an application

## 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



36

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

Slide borrowed from Dominik Gruntz (and modified)



# Activity

Not exactly what you might imagine...

## Activity

From Wikipedia, the free encyclopedia

**Activity** may mean:

- [Action \(philosophy\)](#), in general
- The Aristotelian concept of [energeia](#), Latinized as *actus*
- [Physical exercise](#)
- [Activity \(UML\)](#), a major task in Unified Modeling Language
- [Activity diagram](#), a diagram representing activities in Unified Modeling Language
- *Activity*, an alternative name for the game [charades](#)
- *Activity*, the rate of catalytic activity, such as enzyme activity ([enzyme assay](#)), in physical chemistry and enzymology
- [Activity \(chemistry\)](#), the effective concentration of a solute for the purposes of mass action
- [Activity \(project management\)](#)
- [Activity \(radioactivity\)](#), [radioactive decay](#)/[Radioactive decay rates](#), the number of radioactive decays per second
- [Activity \(software engineering\)](#)
- [Activity \(soil mechanics\)](#)
- *HMS Activity (D94)*, an aircraft carrier of the Royal Navy
- in military parlance, a military agency or unit (e.g. [Intelligence Support Activity](#))
- [Activity Theory](#) , social constructivism (learning theory), Education

Wordnet definitions:

- something that people do or cause to happen
- a process occurring in living organisms
- a process existing in or produced by nature (rather than by the intent of human beings)



# Activities

A rather misleading term... it's not a “computer activity”, like a process.  
It's rather an environment where a “user activity” is performed

- “single” UI screens
- One visible at the time (Well. Almost...)
- One active at the time
- Stacked like a deck of cards

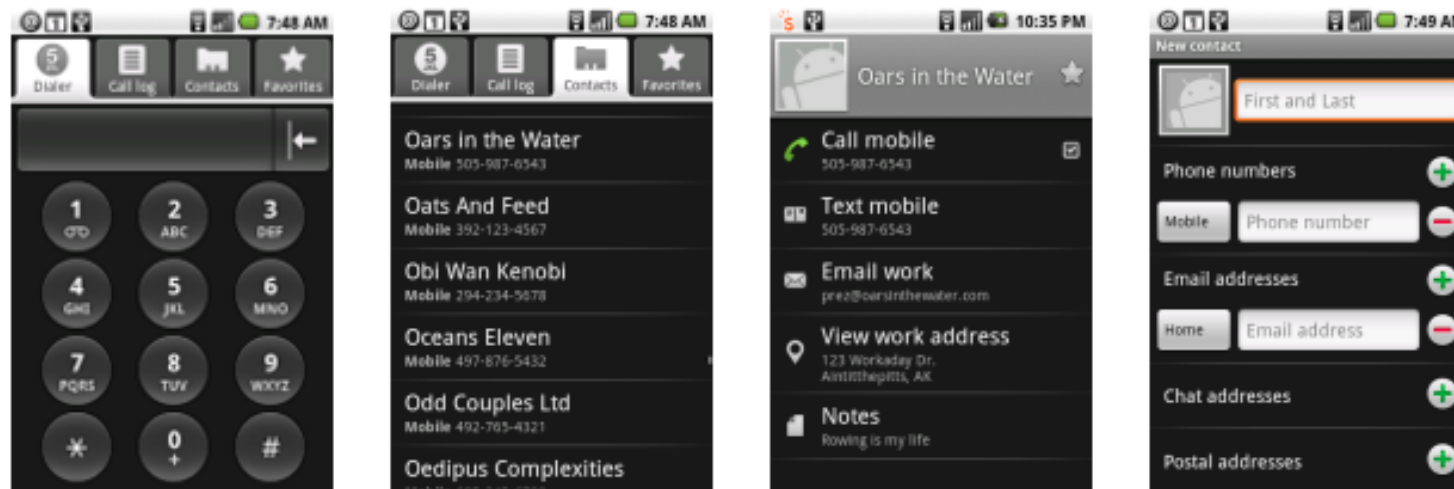


# Activity

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

Contacts

View Contact

New Contact

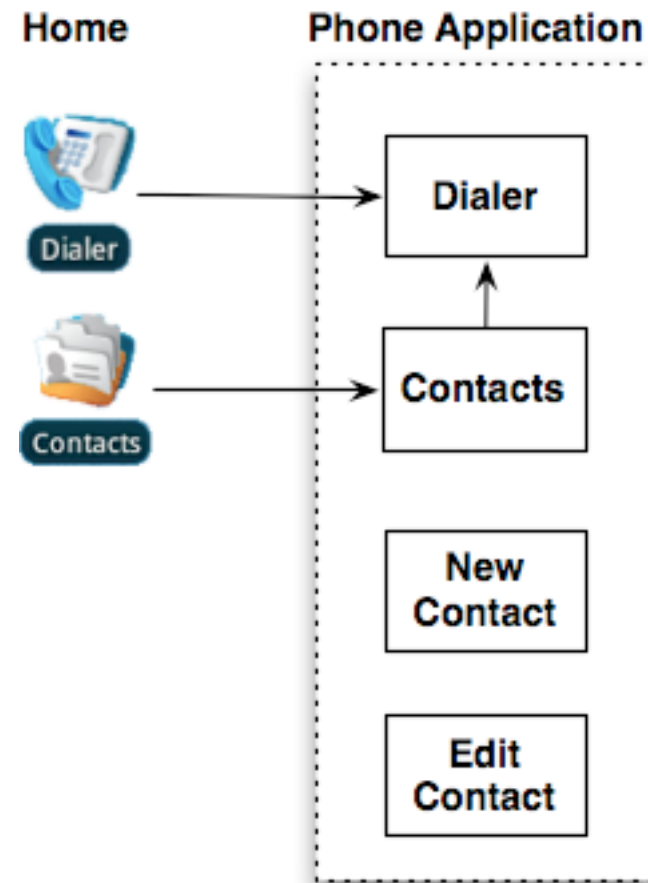


# Multiple entry-point for an app

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"**).

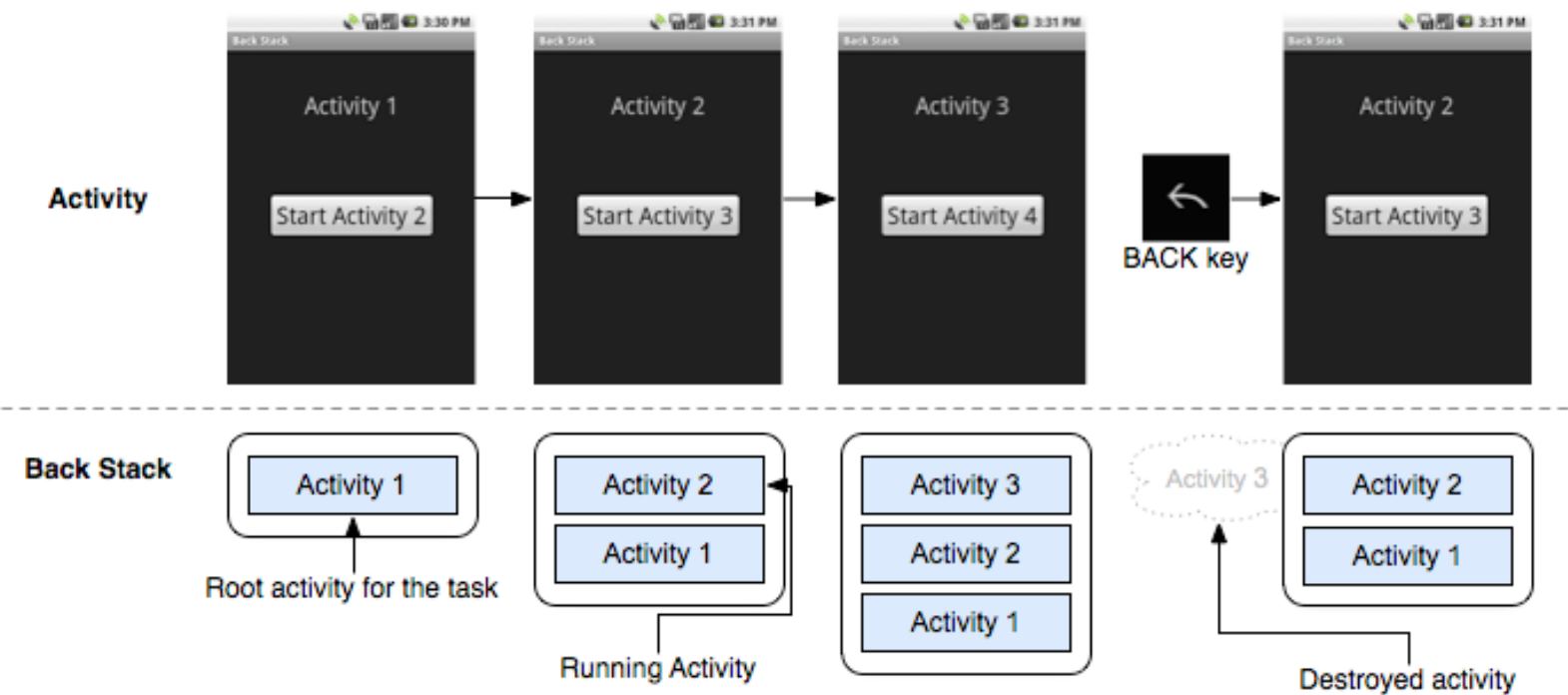
The new activity it is pushed on **top of the back stack** and takes **user focus**.

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**.



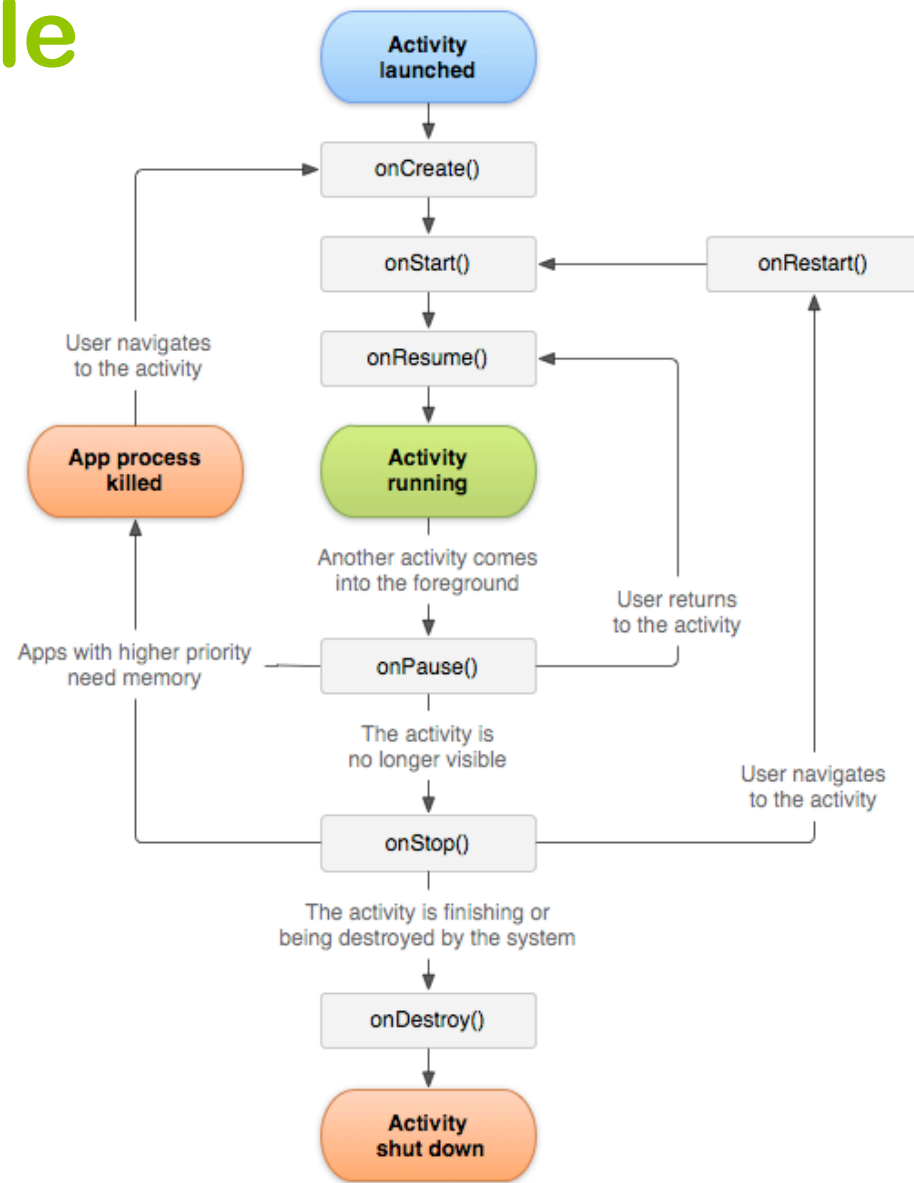
# The activity stack

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



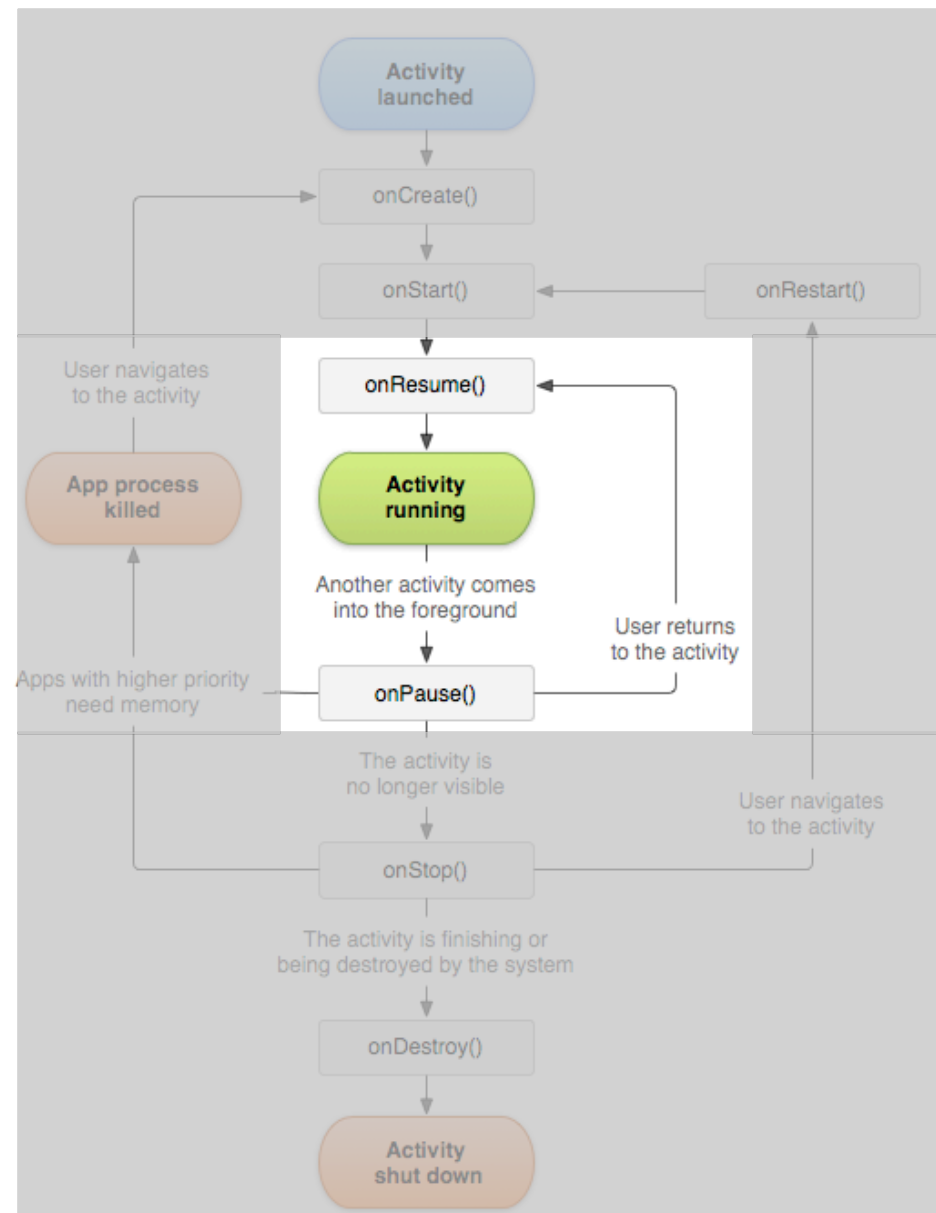
# Activity lifecycle

States (colored),  
and  
Callbacks (gray)



# Activity lifecycle

The FOREGROUND lifetime

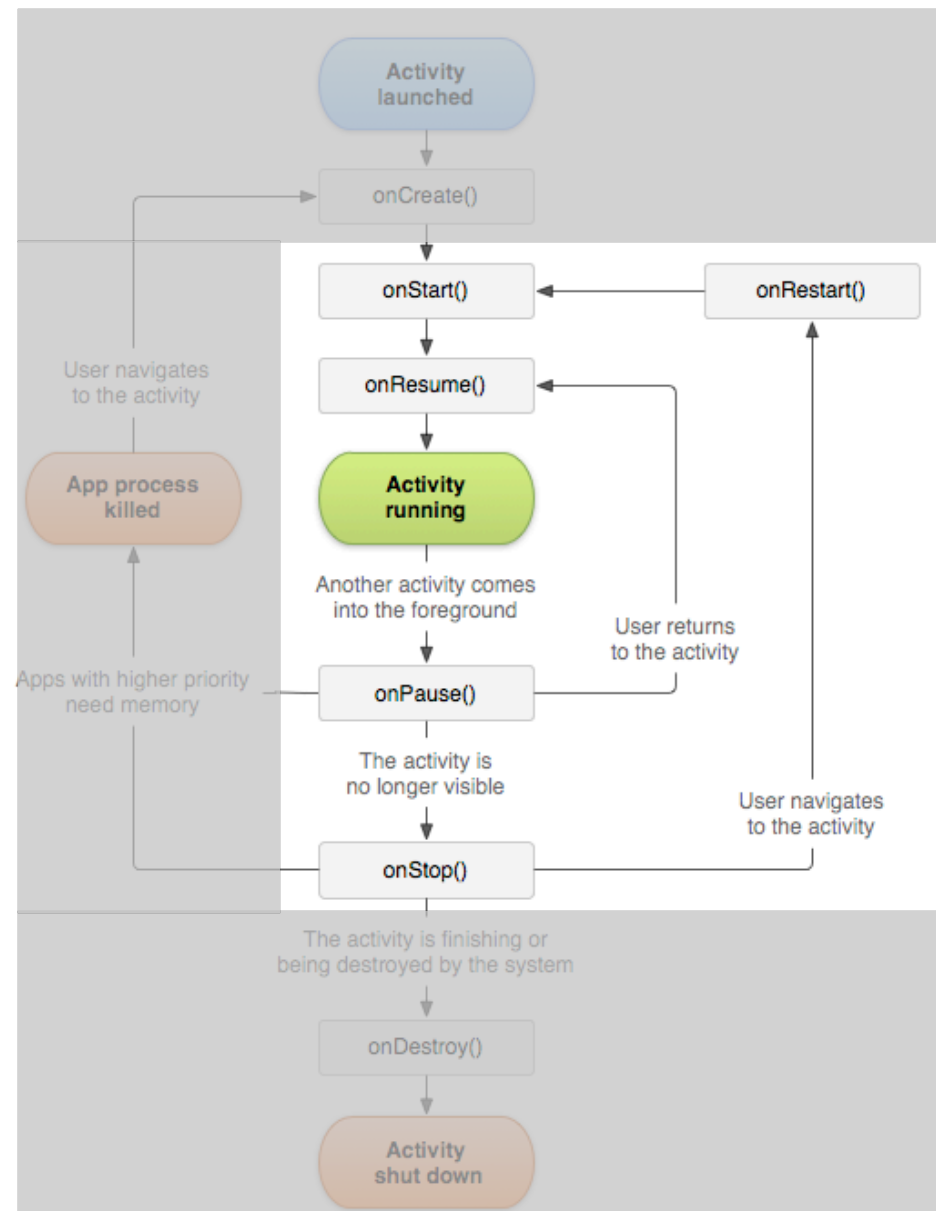


# Activity lifecycle

## The VISIBLE lifetime

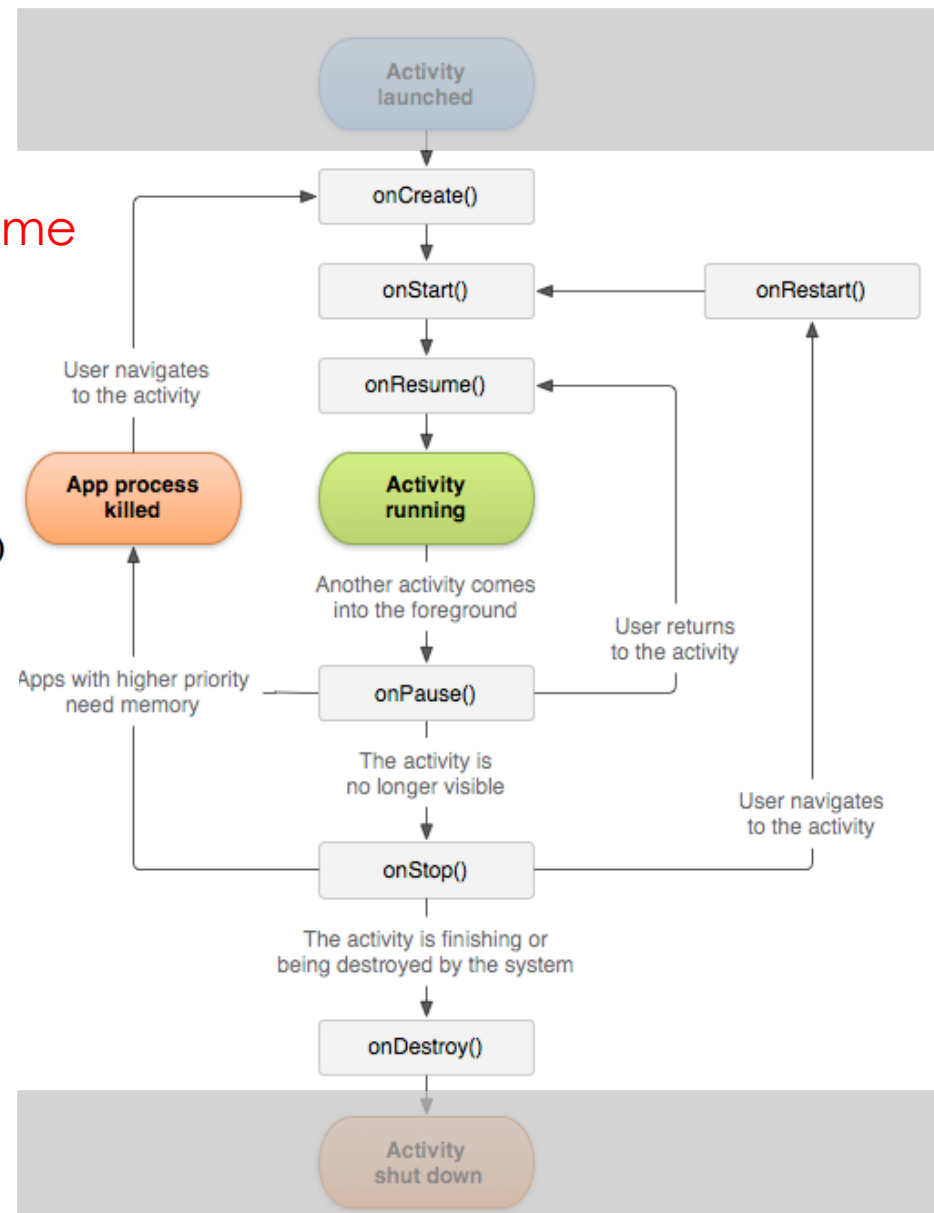
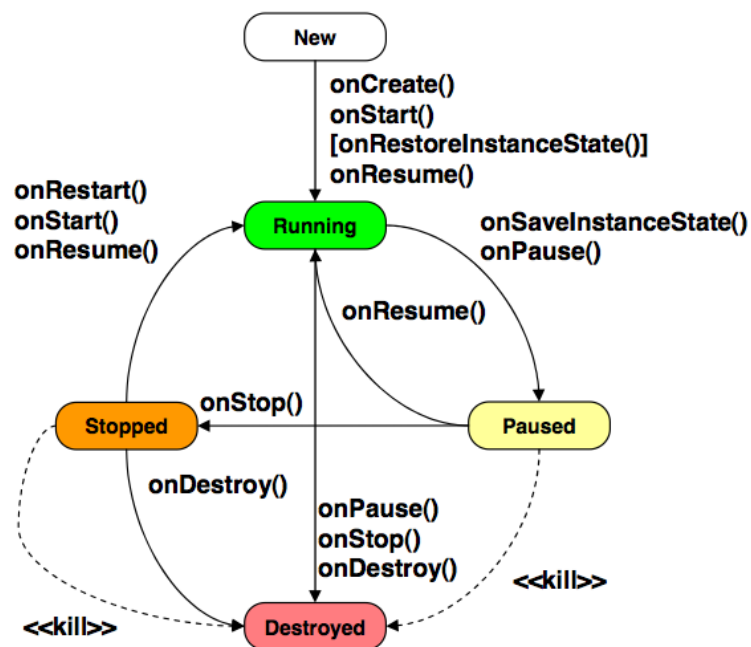
When stopped, your activity should release costly resources, such as network or database connections.

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



# Activity lifecycle

The ENTIRE lifetime

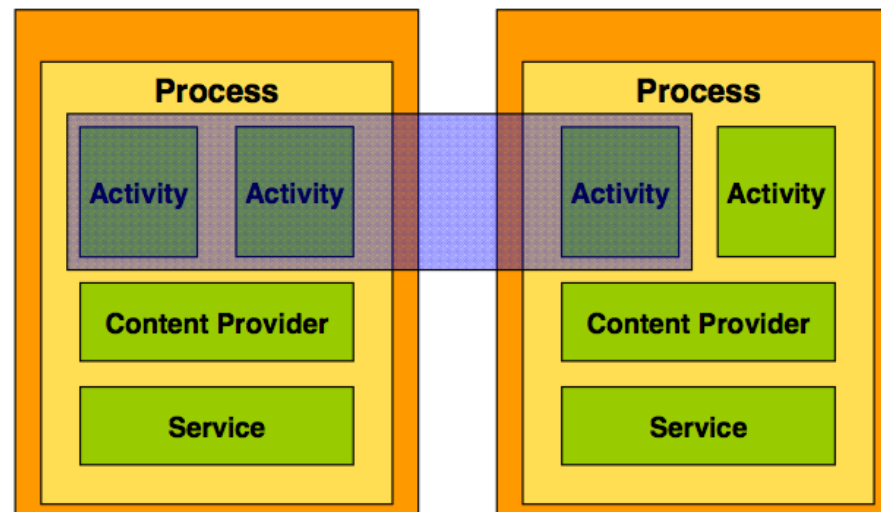


# The shocking news...

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...

## Task (computing)

From Wikipedia, the free encyclopedia



This article **needs additional citations** from **reliable sources**. Unsourced material may be challenged and removed.

### Wordnet definitions:

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

A **task** is an execution path through [address space](#).<sup>[1]</sup> 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](#)

### 48 Notes

[\[edit\]](#)

- ↑ [Data General](#), *RDOS Reference Manual*

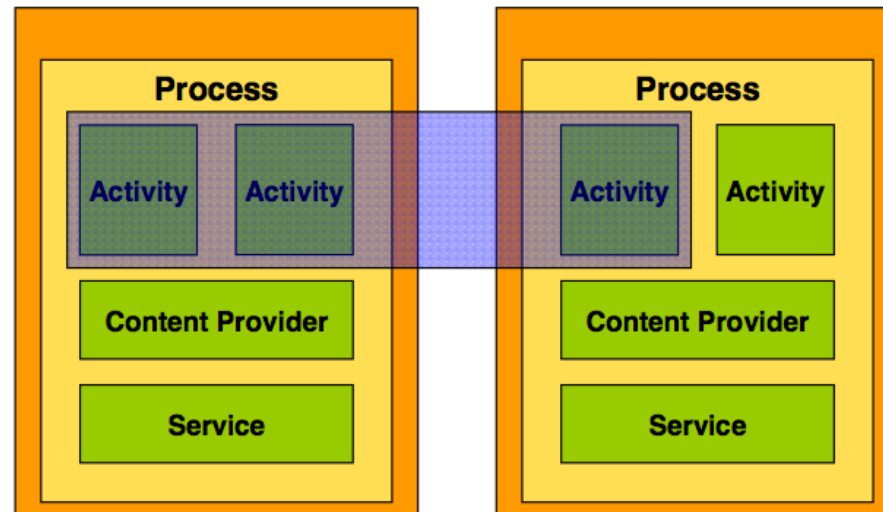




# Tasks

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

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



# 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).

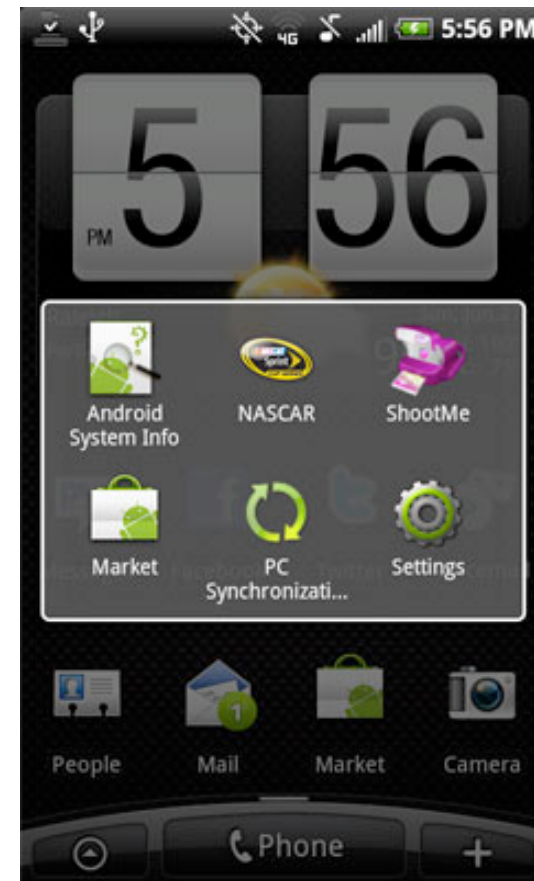


# Switching among apps

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.



# 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



# Task Managers ?

Several apps on the store offered a **task manager** functionality (to kill inactive apps). Are they needed?

*Lots of services and applications constantly run in the background just like they do on Windows. However, and this is important, they do not have to use up a ton of resources. A service or app can be loaded, yet use almost no additional memory, and **0% CPU** until it actually has to do something.*

*In general, killing off stuff is a waste of time. Android automatically asks apps to close when it needs more memory. Killing off processes also means it'll slow your phone down, as when you do need them again the system will need to reload them.*



# Fragment

A Fragment represents a behavior or **a portion of user interface** in an Activity.

You can **combine multiple fragments** in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.

You can think of a fragment as **a modular section of an activity**, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).





# View

the basic building block for user interface components, similar to the Java AWT Component or to the JavaFX Node.

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.)



# Service

A Service is an application component that can perform **long-running operations in the background and does not provide a user interface.**

Another application component can start a service and it will continue to run in the background even if the user switches to another application.

Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.



# Service

You can declare the service as **private**, in the manifest file, and block access from other applications.

Caution: **A service runs in the main thread of its hosting process** — the service does not create its own thread and does not run in a separate process (unless you specify otherwise). This means that, if your service is going to do any CPU intensive work or blocking operations (such as MP3 playback or networking), you should **create a new thread** within the service to do that work.

By using a separate thread, you will reduce the risk of Application Not Responding (ANR) errors and the application's main thread can remain dedicated to user interaction with your activities.



# Intents

Three of the core components of an application — activities, services, and broadcast receivers — are activated through messages, called 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.

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.



# Broadcast receiver

A broadcast receiver is a component that **responds to system-wide broadcast announcements**.

Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use.

Although **broadcast receivers don't display a user interface**, they may create a status bar notification to alert the user when a broadcast event occurs.



# Content Provider

Content **providers manage access to a structured set of data**. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process.

Android itself includes content providers that manage data such as audio, video, images, and personal contact information.

62

You can see some of them listed in the reference documentation for the `android.provider` package.





# Screen properties

Marco Ronchetti  
Università degli Studi di Trento

---



# Screen properties

Marco Ronchetti  
Università degli Studi di Trento

---

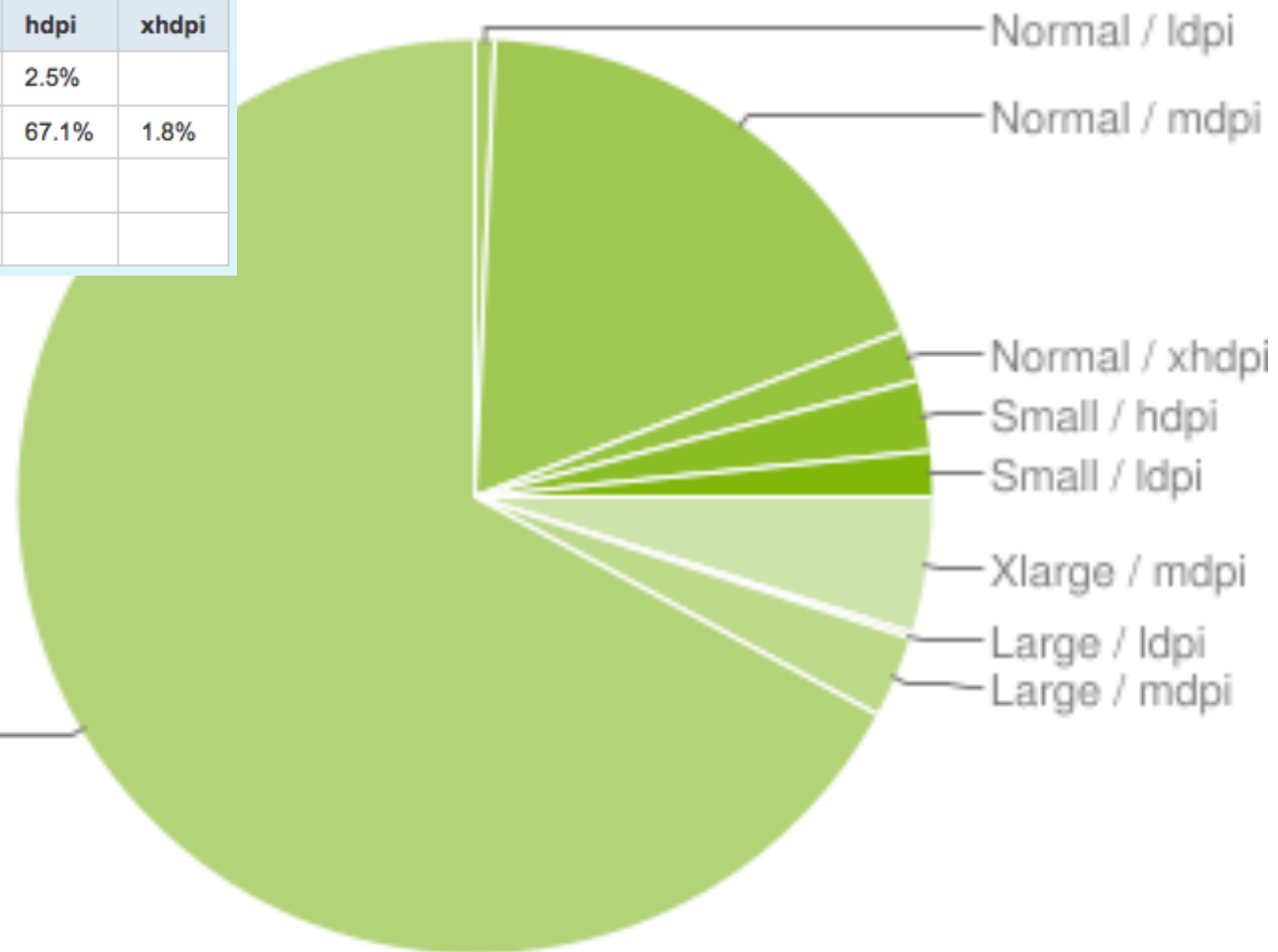


# Screen Sizes and Densities

	ldpi	mdpi	hdpi	xhdpi
small	1.6%		2.5%	
normal	0.7%	18.4%	67.1%	1.8%
large	0.2%	2.9%		
xlarge		4.8%		

Data of  
February 1<sup>st</sup>  
2012

Normal / hdpi



65

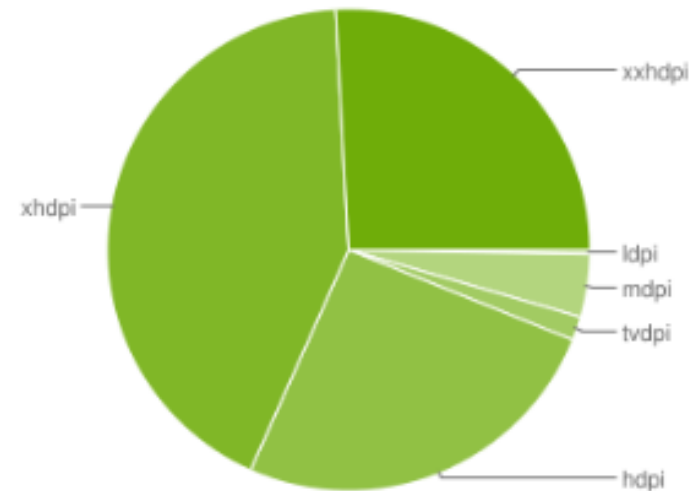
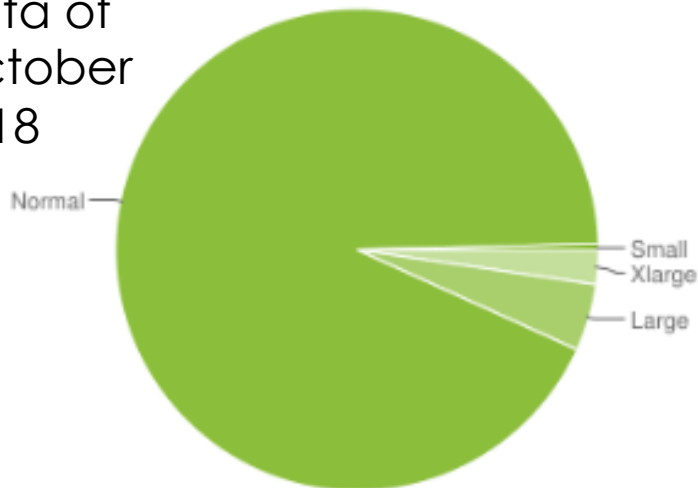


<http://developer.android.com/resources/dashboard/screens.html>

# Screen Sizes and Densities

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	0.3%					0.1%	0.4%
Normal		0.7%	0.3%	24.7%	41.9%	25.2%	92.8%
Large		2.0%	1.3%	0.4%	0.3%	0.5%	4.5%
Xlarge		1.5%		0.5%	0.3%		2.3%
Total	0.3%	4.2%	1.6%	25.6%	42.5%	25.8%	

Data of  
October  
2018



## Screen related terms and concepts

**Resolution** The total number of physical pixels on a screen. When adding support for multiple screens, applications do not work directly with resolution; applications should be concerned only with screen size and density, as specified by the generalized size and density groups.

**Screen size** Actual physical size, measured as the screen's diagonal.

**Screen density** The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch).

**Orientation** The orientation of the screen from the user's point of view. This is either landscape or portrait, meaning that the screen's aspect ratio is either wide or tall, respectively. Not only do different devices operate in different orientations by default, but the orientation can change at runtime when the user rotates the device.



# Density-independent pixel

*Density-independent pixel (dp)* A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way. The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, which is the baseline density assumed by the system for a "medium" density screen. At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use. The conversion of dp units to screen pixels is simple:  $px = dp * (dpi / 160)$ . For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels. You should always use dp units when defining your application's UI, to ensure proper display of your UI on screens with different densities.



*xlarge* screens are at least 960dp x 720dp  
*large* screens are at least 640dp x 480dp  
*normal* screens are at least 470dp x 320dp  
*small* screens are at least 426dp x 320dp



# Screen Sizes and Densities

Android divides the range of actual screen sizes and densities into:

A set of four generalized **sizes**:

*small, normal, large, and xlarge*

A set of four generalized **densities**:

*ldpi* (low), *mdpi* (medium), *hdpi* (high), and *xhdpi* (extra high)





# Una lettura consigliata...

Marco Ronchetti  
Università degli Studi di Trento

---

# Android design

<http://developer.android.com/design/index.html>

