Lab on apps development for tablets, smartphones and smartwatches

Week 2: Android basics

Dr. Marina Zapater, Prof. David Atienza

Mr. Grégoire Surrel, Ms. Elisabetta de Giovanni, Mr. Dionisijie Sopic, Ms. Halima Najibi

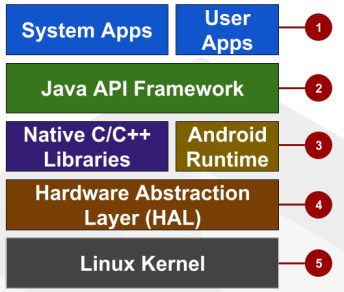
Embedded Systems Laboratory (ESL) – Faculty of Engineering (STI)

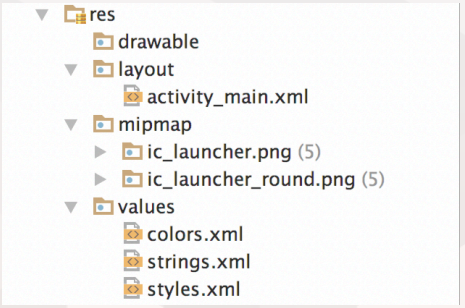
Before starting…

- Teams and projects: first assignment of people to teams
  - Check in Moodle! → I will drop by your desks to confirm you're ok with it!
- Solutions to the labs on Moodle and "git"!
  - When do you want solutions to be available?
    - Fill-in the form!
- Update! Update! Update!
  - **Update your SDK** to the latest version
  - If your tablet/watch ask you to update… **Please do so!**

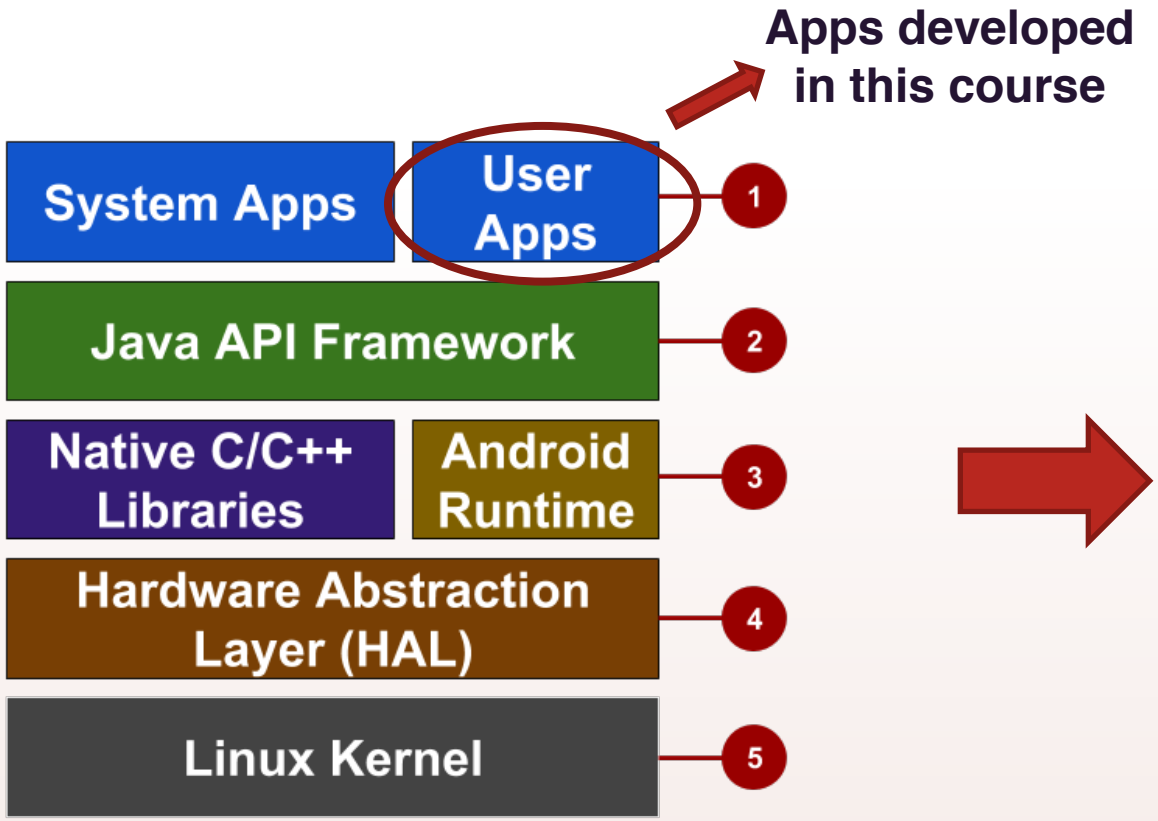
Class outline

- **The Android platform architecture**
  - The Android Stack: More in-depth information
  - Android components
- Activities
- Application resources
- Views and Layouts
  - Event handling
- Lab of Today

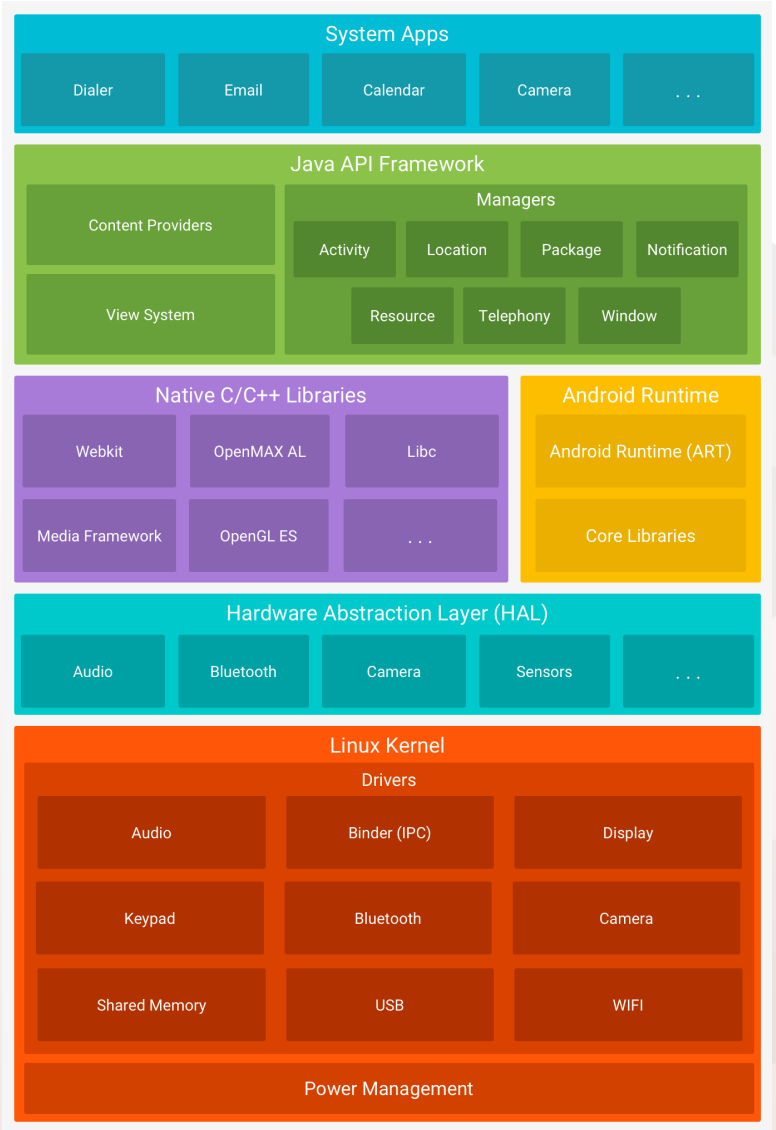




The (expanded) Android stack



5-layer stack introduced last week

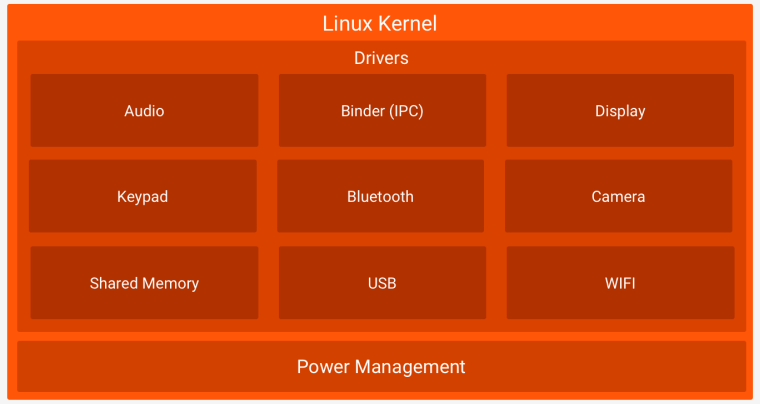


Platform Architecture: https://developer.android.com/guide/platform/index.html

5.Linux Kernel

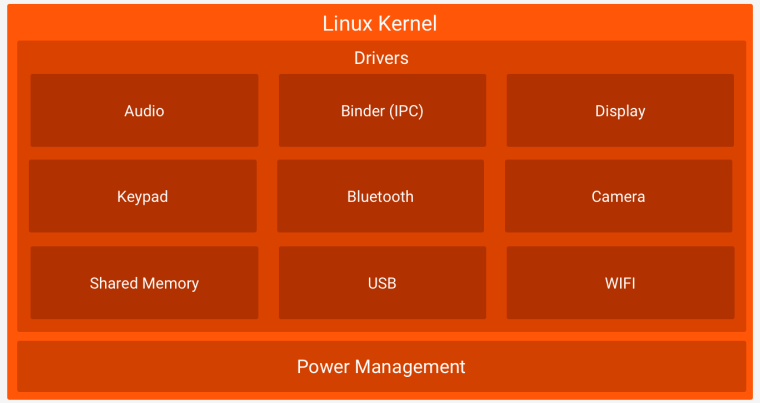
- Android is built on the Linux kernel
- No native windowing system
- No glibc support
- Does not include the full set of standard Linux utilities

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.



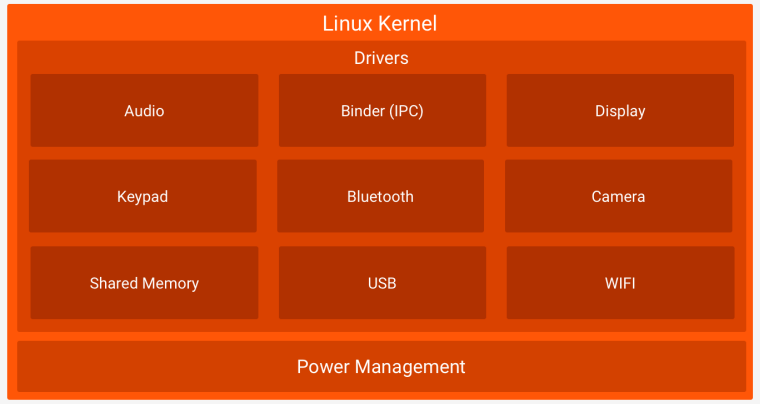
5.Linux Kernel

- Why Linux kernel:
  - Great memory and process management
  - Permission-based security model
  - Proven driver model
  - Support for shared library
  - Open source!



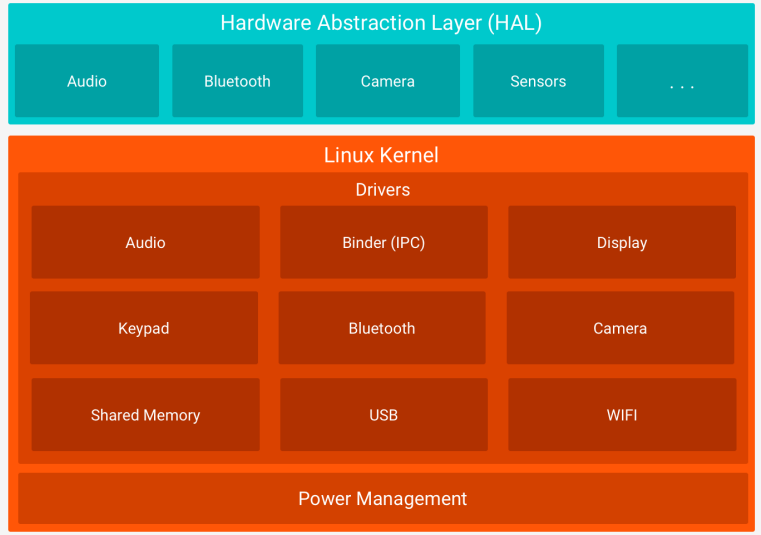
5.Linux Kernel

- Kernel Enhancements:
  - Alarm
  - Low Memory Killer
  - Kernel Debugger
  - Logger
  - Binder (IPC)
  - Power Management



4.Hardware Abstraction Layer (HAL)

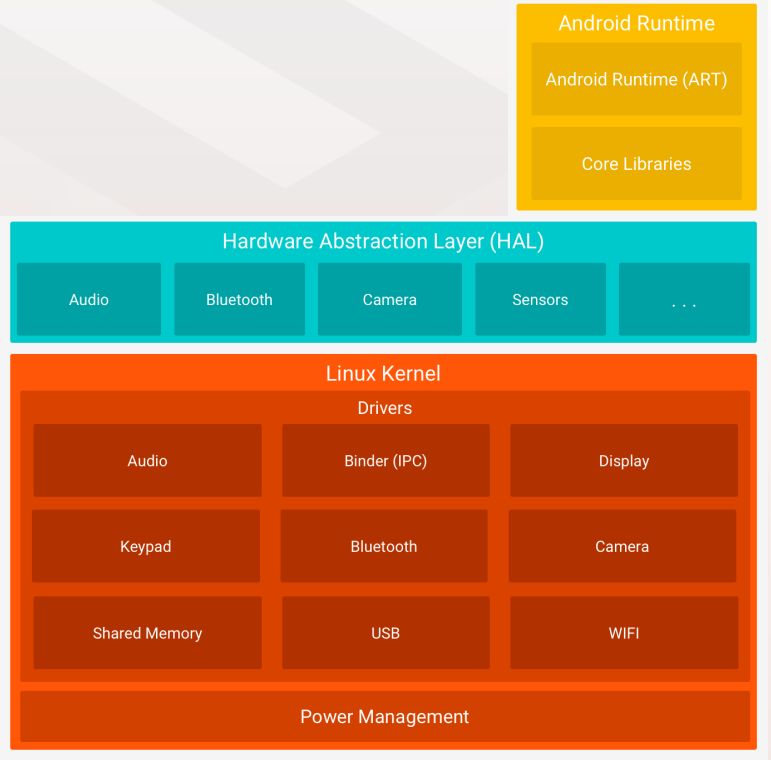
- Provides standard interfaces between the hardware capabilities (Linux kernel) and the high-level Java API framework
- One HAL library module for each hardware component
- User space C/C++ shared library (built as ".so")
- To add new hardware you need to implement the driver + the HAL



3(a).Android Runtime (ART)

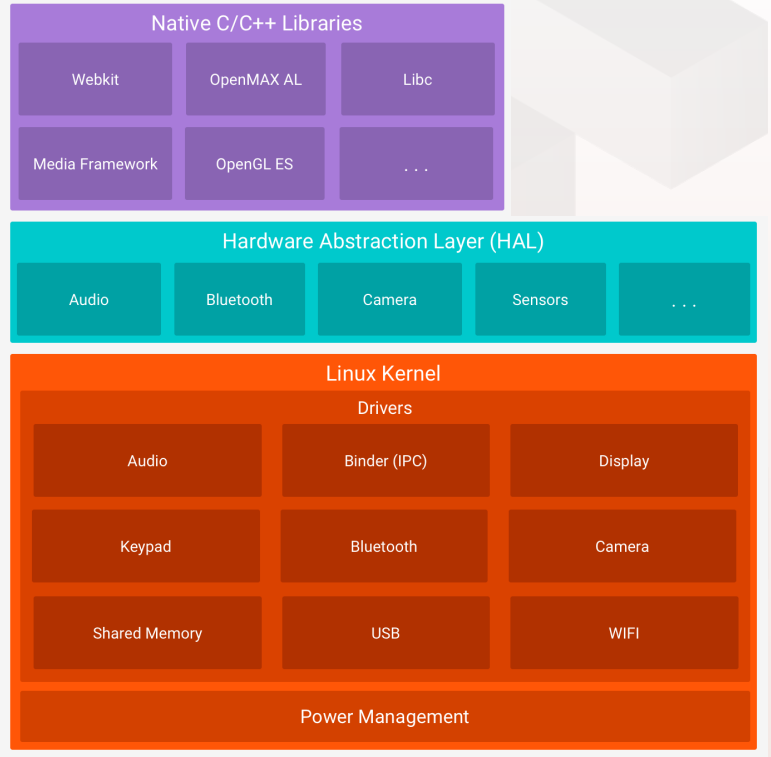
- For devices running Android version 5.0 (API level 21) or higher, each **app runs in its own process and with its own instance** of the Android Runtime (ART)
  - Prior to version 5.0 (API 21), Dalvik was the Android Runtime

- ART runs multiple virtual machines (VMs), by executing an Android bytecode (DEX files).
  - Ahead-of-time (AOT) and just-in-time (JIT) compilation
  - Optimized garbage collector (GC)
  - Better debugging support



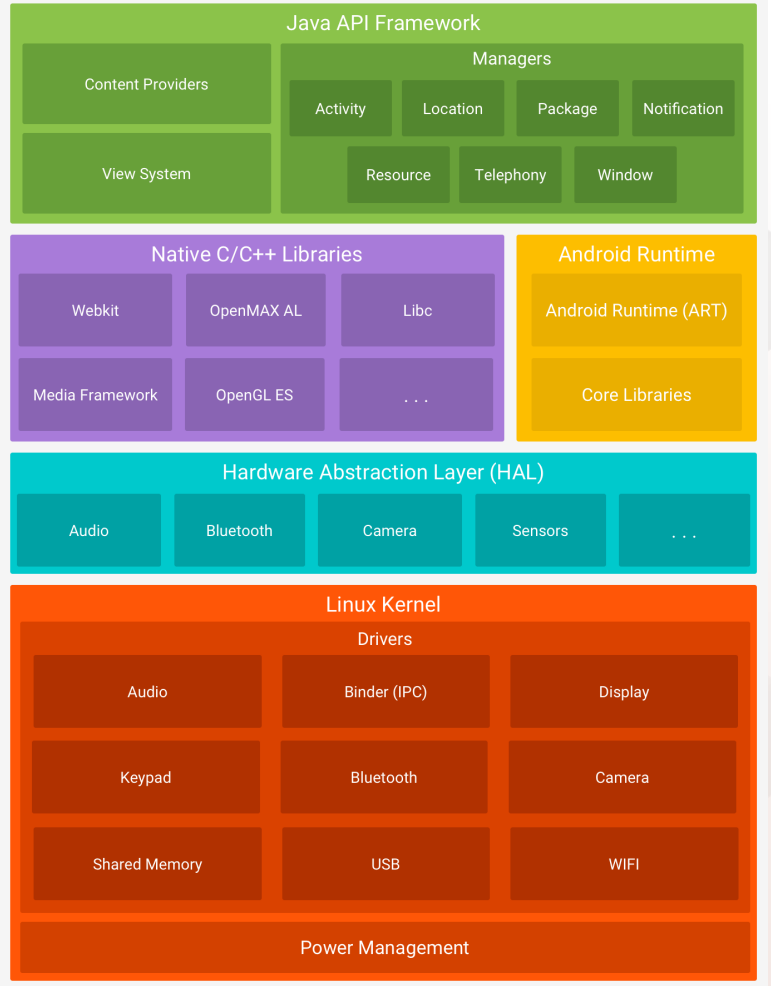
3(b).Native Libraries

- Many Android core components (such as ART and HAL) are built in C/C++ code that require libraries.
- Examples of native libraries include:
  - OpenGL ES: 2D/3D graphics engine
  - SQLite: relational database engine
  - Libc: standard C library
  - Webkit: tools for browsing web
  - …



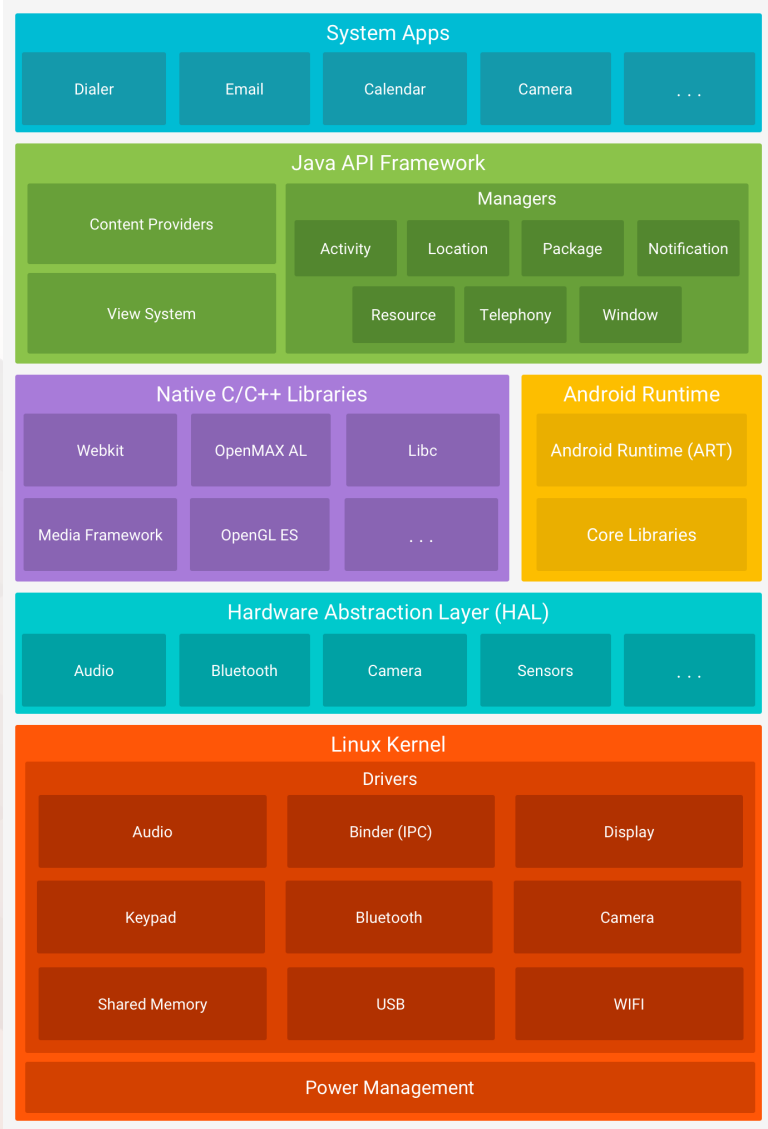
2.Java API Framework

- The entire feature-set of the Android OS is available to you through Java APIs
- Developers have full access to the same framework APIs used by the core apps.
  - The application architecture is designed to simplify the reuse of components:
- Includes:
  - View system: to build the app User Interface (UI)
  - Content provider: access data from other apps.
  - Managers:
    - Resources: access to non-code structures
    - Notifications: custom alerts on status bar
    - Activity: lifecycle of applications



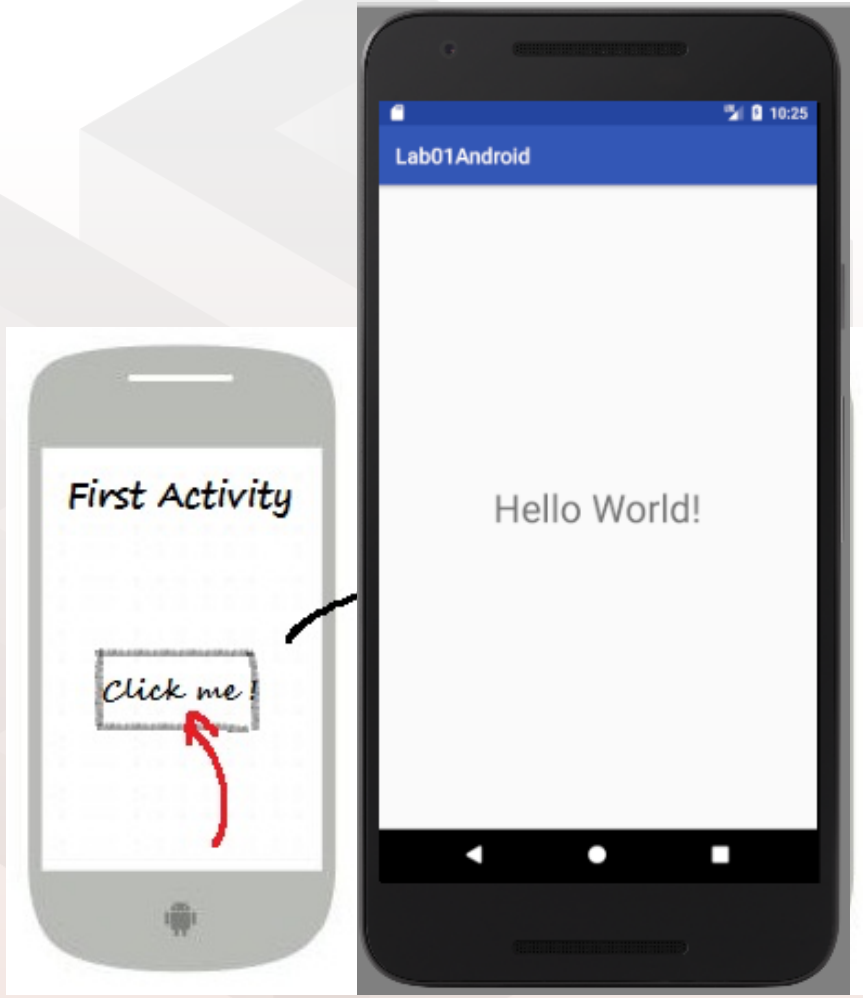
1.System apps

- Set of core applications included in the platform:
  - E-mail, calendar, camera…
- Provide key capabilities for developers
  - No need to re-write functionalities



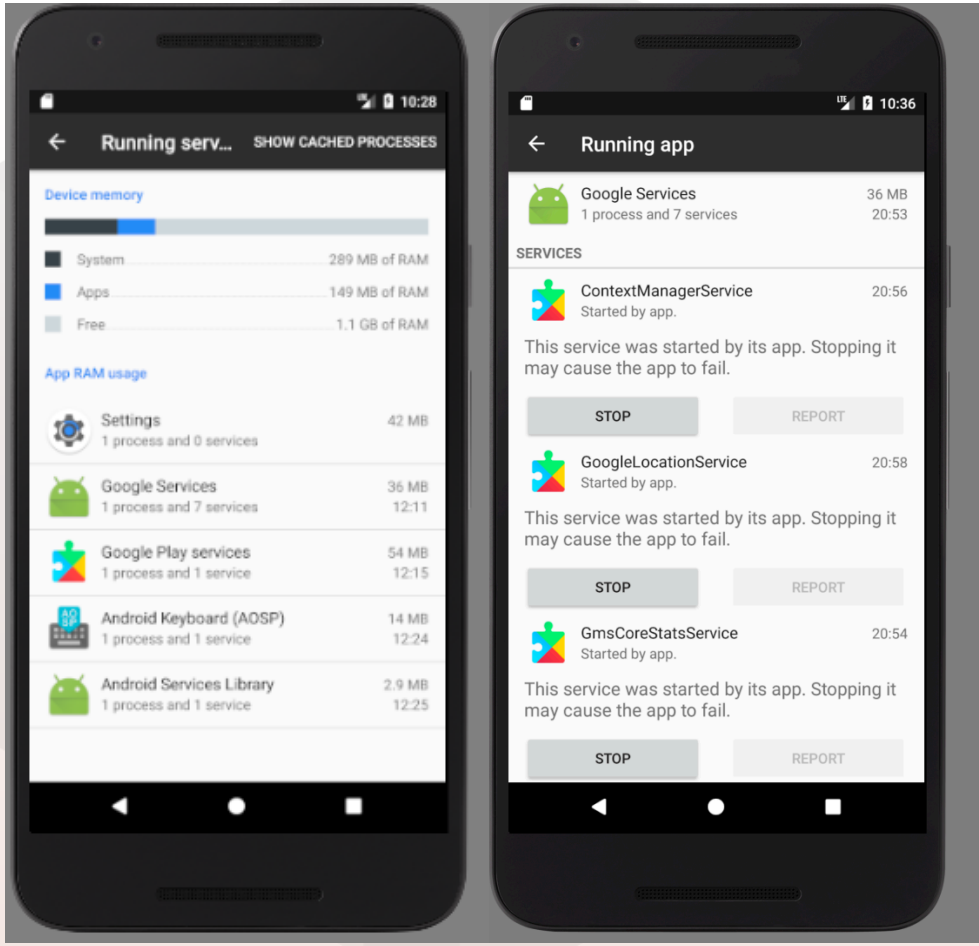
Android Component types

- **Activity** is a single screen with a user interface
  - Last week you created your first Activity (MainActivity → HelloWorld)
- **Service** performs long-running tasks in background
- **Content provider** manages shared set of data
- **Broadcast receiver** responds to system-wide announcements



Android Component types

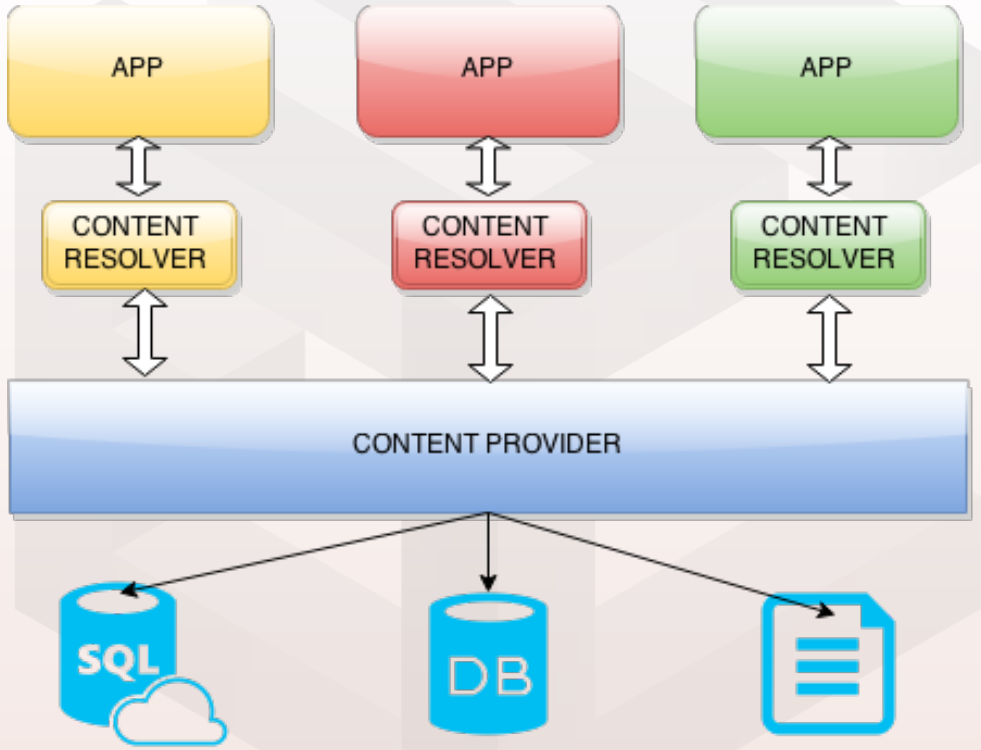
- **Activity** is a single screen with a user interface
  - Today you'll be creating your first Activity (MainActivity → HelloWorld)
- **Service** performs long-running tasks in background
- **Content provider** manages shared set of data
- **Broadcast receiver** responds to system-wide announcements



We'll go in depth in future lectures!

Component types

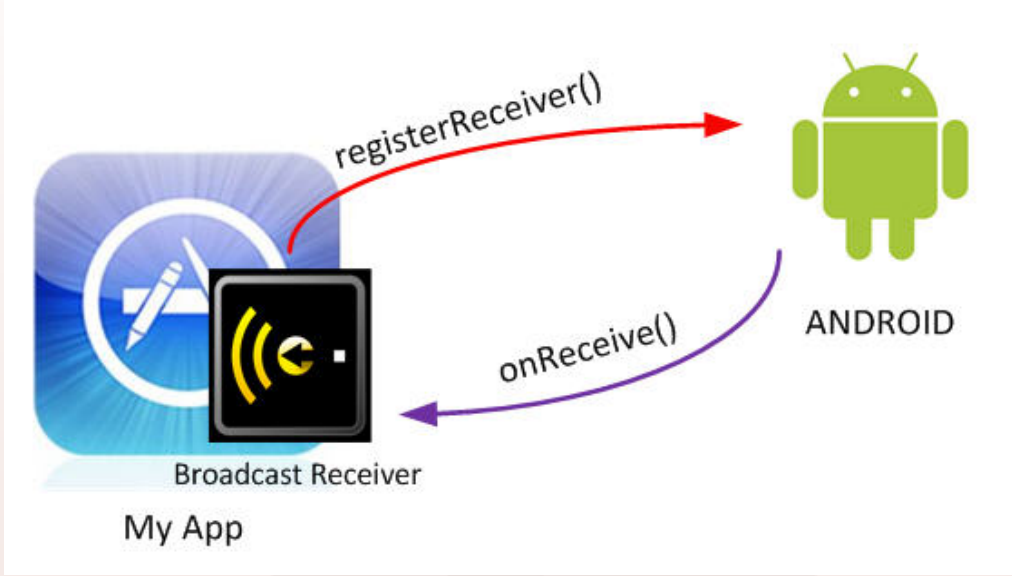
- **Activity** is a single screen with a user interface
  - Today you'll be creating your first Activity (MainActivity → HelloWorld)
- **Service** performs long-running tasks in background
- **Content provider** manages shared set of data
- **Broadcast receiver** responds to system-wide announcements



We'll go in depth in future lectures!

Component types

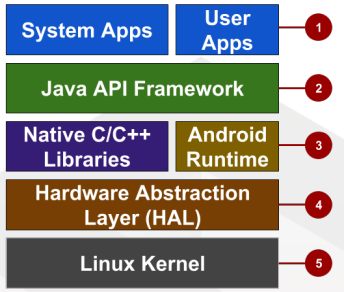
- **Activity** is a single screen with a user interface
  - Today you'll be creating your first Activity (MainActivity → HelloWorld)
- **Service** performs long-running tasks in background
- **Content provider** manages shared set of data
- **Broadcast receiver** responds to system-wide announcements
  - Low battery, turned-off screen…

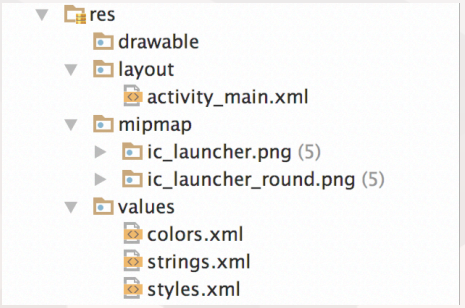


We'll go in depth in future lectures!

Class outline

- The Android platform architecture
  - The Android Stack: More in-depth information
  - Android components
- **Activities**
- Application resources
- Views and Layouts
  - Event handling
- Lab of Today





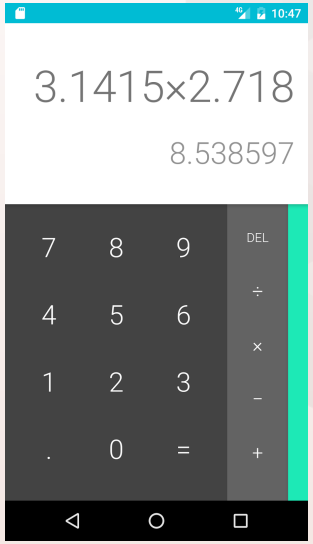
What is an activity?

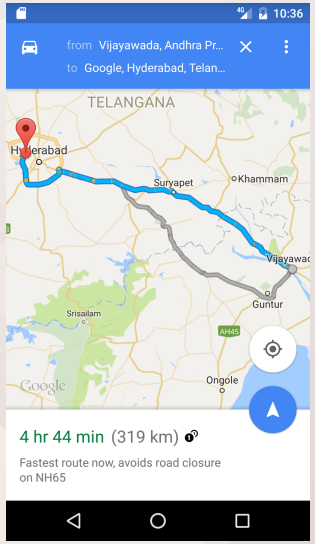
- One of the 4 component types of Android
  - Application components:
    - Activities, Services, Content Providers, Broadcast Receivers
- Provides a visual interface for user interaction
  - Represents one window, one hierarchy of views
- Typically supports one thing a user can do
- Java class, typically one activity in one file
- An activity typically has a UI layout, defined in an XML file

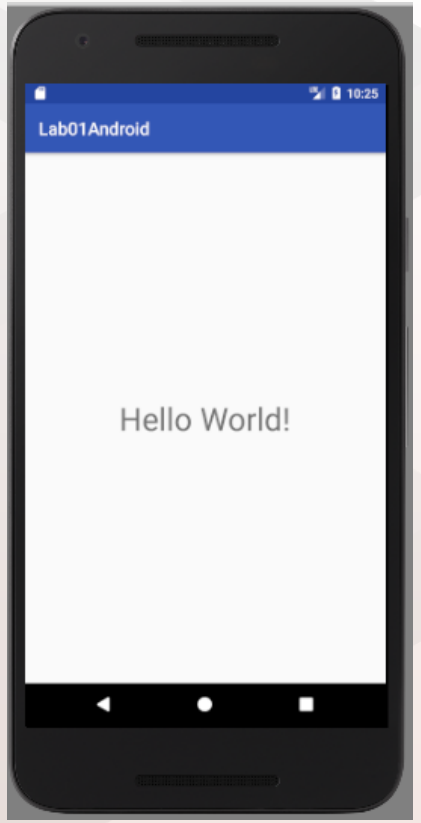
- We worked with one activity (**MainActivity**) for our HelloWorld app
  - MainActivity.java → Activity
  - activity_main.xml → The UI of our activity.

Apps and Activities

- Activities handle user interactions (button clicks, text entries…)
- Activities are tied together to make up an app
  - They can start other activities in the same or other apps
- Activities can be organized in a parent-child relationship
- Activities have a life cycle:
  - are created, started, run, paused, resumed, stopped destroyed…
  - **We'll see this next week!**







The Cookbook for Implementing Activities

1. Define layout in XML
2. Define Activity Java class
3. Connect Activity with Layout
   - Set content view in onCreate()
   - setContentView(R.layout.activity_main)
4. Declare activity in the Android manifest

1.

activity_main.xml × | AndroidManifest.xml × | Log.class × | strings.xml ×

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://sche
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.zapater.lab01android.MainActivity">

    <TextView
        android:id="@+id/my_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="36sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

2.

activity_main.xml × | MainActivity.java × | AndroidManifest.xml × | Log.class ×

```
package com.example.zapater.lab01android;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView myview1 = (TextView) findViewById(R.id.my_text);
        String s = "This is my first app!";
        myview1.setText(s);
    }
}
```

3.

The Cookbook for Implementing Activities

1. Define layout in XML
2. Define Activity Java class
3. Connect Activity with Layout
   - Set content view in onCreate()
   - setContentView(R.layout.activity_main)
4. Declare activity in the Android manifest

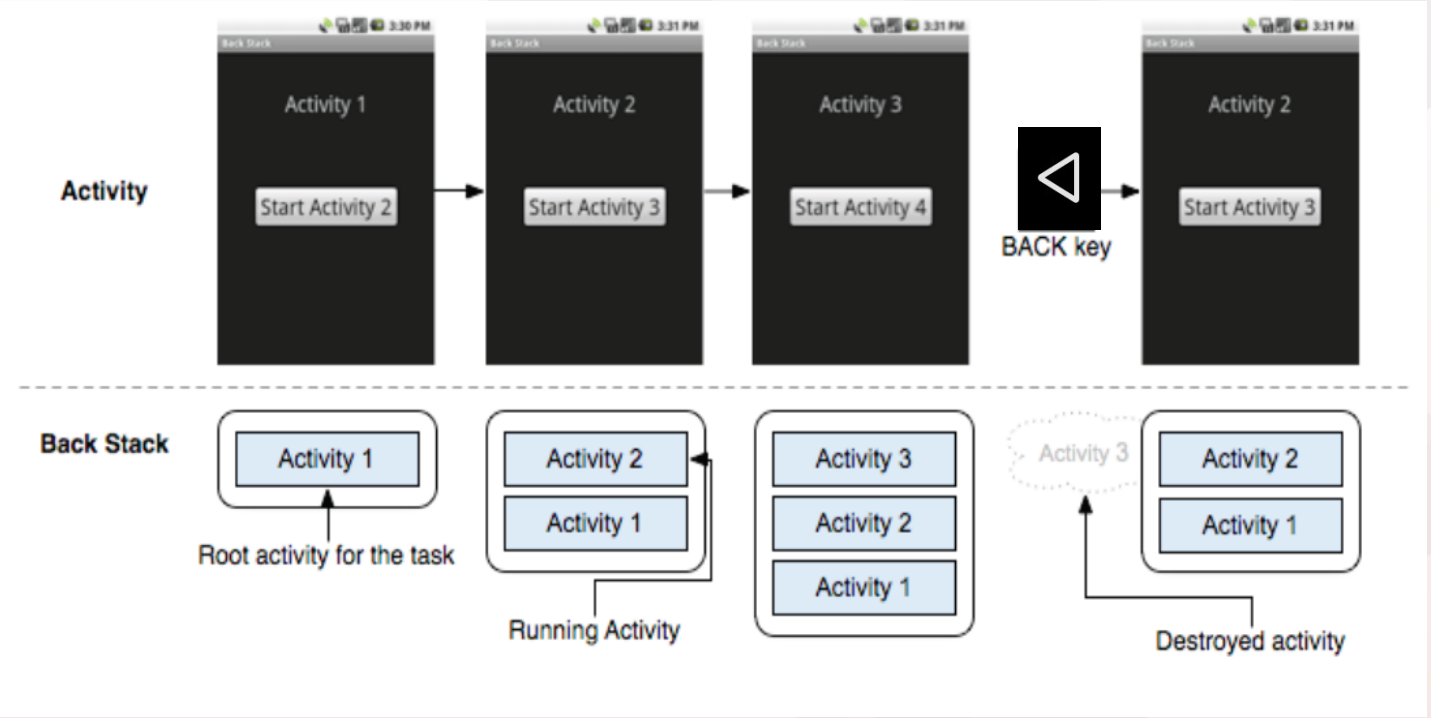
4.

activity_main.xml × | MainActivity.java × | AndroidManifest.xml × | Log.class × | strings

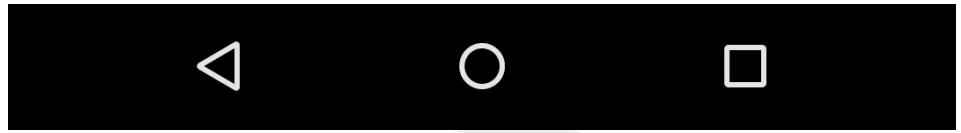
```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.zapater.lab01android">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="Lab01Android"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportsRtl="true"
11         android:theme="@style/AppTheme">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19     </application>
20
21 </manifest>
```

The Activity Stack

- When a new activity is started, the previous one is stopped and pushed on the activity back stack
- Last-in-first-out (LIFO) stack: when the current activity ends, or the user presses the ”back” button, the activity is popped from the stack and the previous resumes



Two ways of navigation



Temporal or “back” navigation

- Provided by the device’s back button
- Controlled by the Android system’s back stack
- Switching between tasks activates the task’s back stack
- Launching an activity from the home screen ○ starts a new task
- Navigate between tasks □ with the overview of recent tasks screens

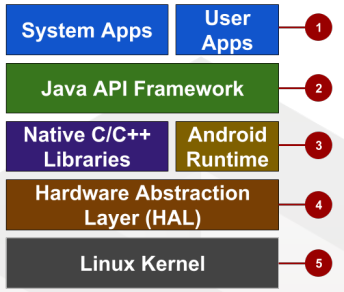
Ancestral or “up” navigation

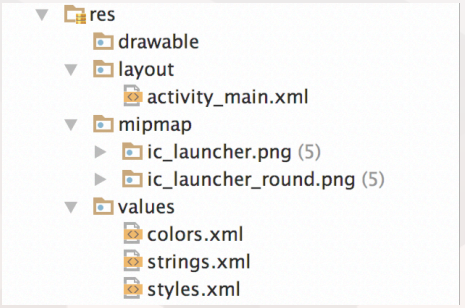
- Provided by the app’s action bar
- Controlled by defining parent-child relationships in the Android manifest

```
<activity
    android:name=".ShowDinnerActivity"
    android:parentActivityName=".MainActivity" >
</activity>
```

Class outline

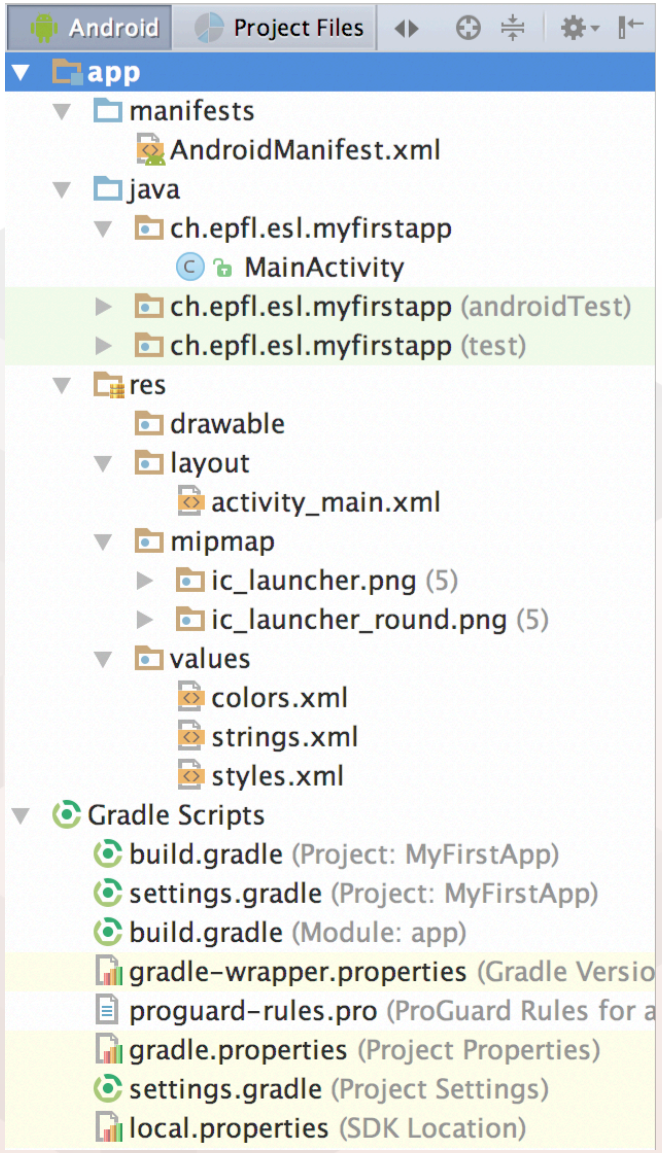
- The Android platform architecture
  - The Android Stack: More in-depth information
  - Android components
- Activities
- **Application resources**
- Views and Layouts
  - Event handling
- Lab of Today





Android application building blocks

- Manifest:
  - Characteristics of the app and component definition
- Components → ”java” folder
  - Java code (functionality) of your app
  - activities, services, …, and helper classes
- **Resources** → ”res” folder
  - layouts, images, strings, colors, XML and media files
- Gradle Scripts:
  - Scripts used to build the application
  - APK versions in Gradle config files

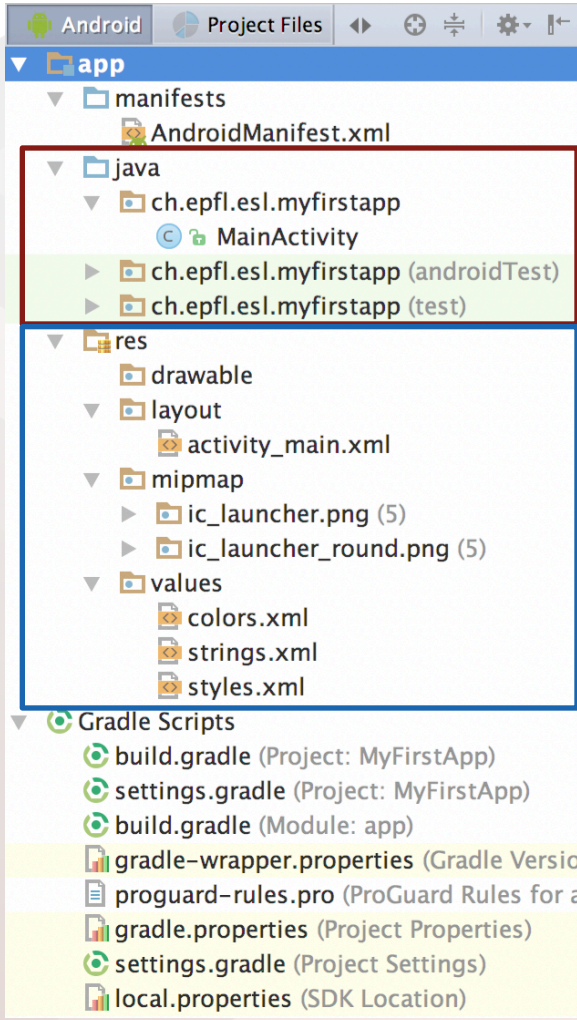


Last week we saw this slide!

Application Resources Definition

- An Application is composed of: code and resources.

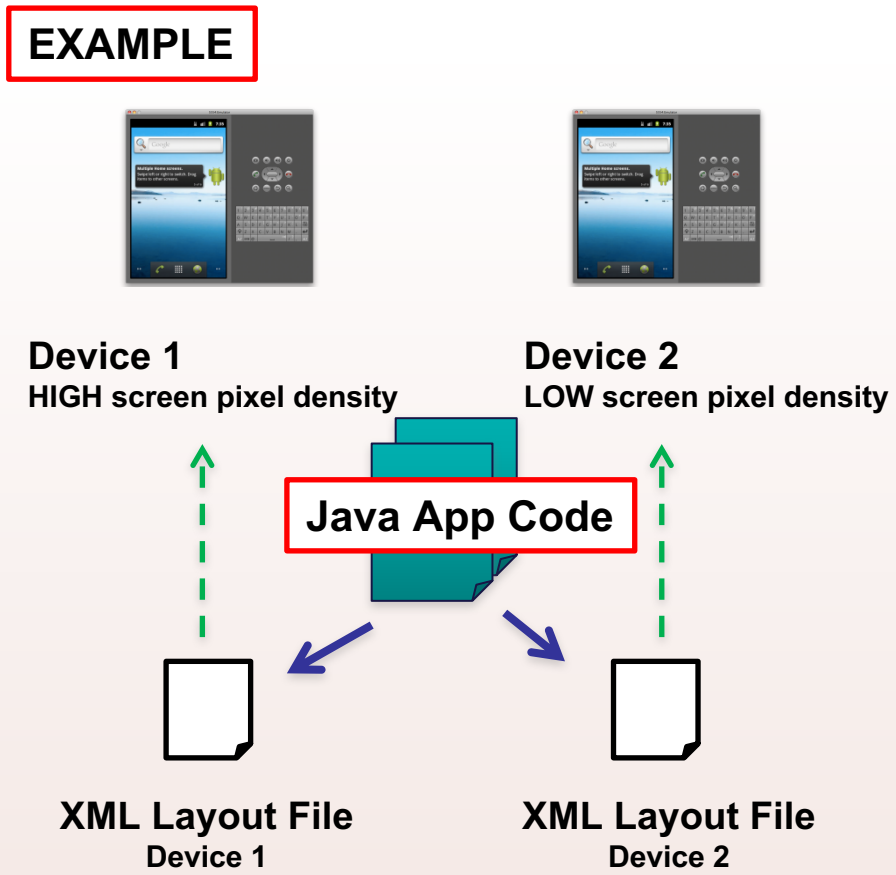
DEF. Resources are <u>everything that is not code</u>
(including: XML layout files, images, audio/video files...)



Utilization of Resources… why?

- **Separate** data presentation (layout) from data management
- **Provide** alternative resources to support specific device configurations (e.g. different language packs)
- **Re-compile** <u>only when strictly needed</u>!

Application Resources Definition



- Build the **application layout** through XML files (like HTML)
- Define **two** different XML **layouts** for two different devices
- At **runtime**, Android detects the current device configuration and loads the appropriate resources for the application
- **No need to recompile**!
- Just add a new XML file if you need to support a new device

Application Resources Definition

Resource Type	Resource contained
res/animator	*XML files that define property animations.*
res/anim	*XML files that define tween animations.*
res/colors	*XML files that define a state list of colors.*
res/drawable	*Bitmap files (.png, .9.png, .jpg, .gif) or XML files that are compiled into other resources.*
res/layout	*XML files that define a user interface layout.*
res/menu	*XML files that define application menus.*
res/raw	*Arbitrary files to save in their raw form.*
res/values	*XML files that contain simple values, such as strings, integers, array.*
res/xml	*Arbitrary XML files.*

Definition of Application Resources

- Resources are defined in a **declarative** way through **XML**.
- Each resource has a name/identifier (see details later).

Example: **string.xml** contains all the text that the application uses. For example, the name of buttons, labels. default text, etc

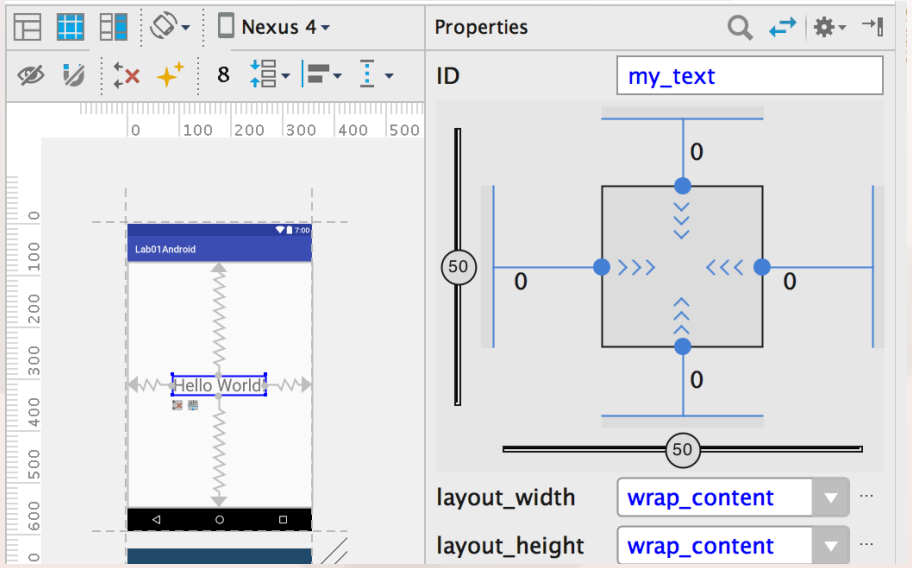
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="hello"> Hello world! </string>
   <string name="labelButton"> Insert your username </string>

</resources>
```

Resource type (string)

Access to Application Resources

- Each Resource is associated with an **Identifier** (ID), that is composed of two parts:
  - The resource **type**: Each resource is grouped into a "type," (e.g. string, color, menu, drawable, layout, etc.)
  - The resource **name**, which is either: the filename, excluding the extension; or the value in the XML *<android:name>* attribute.
  - Identifiers must be unique!

- **Two ways** to access resources:
  - From the **Java** Code
  - From the **XML** files



Acess to Application Resources via Java

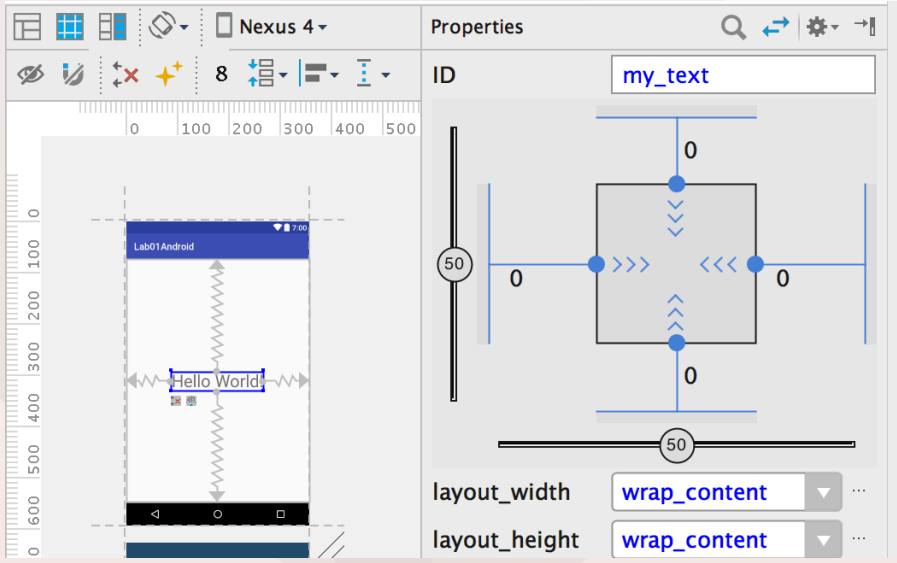
- Resource can be accessed in the **Java** code through the **R class**, that works as a **glue** between the world of java and the world of resources.

R.<*resource_type*>.<*resource_name*>

  - **<resource_type>** is the **R** subclass for the resource type
  - **<resource_name>** is either the resource filename or the android:name attribute

- Exactly as we did in Lab1:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TextView myview1 = (TextView) findViewById(R.id.my_text);
    String s = "This is my first app!";
    myview1.setText(s);
    demo_logcat();

}
```



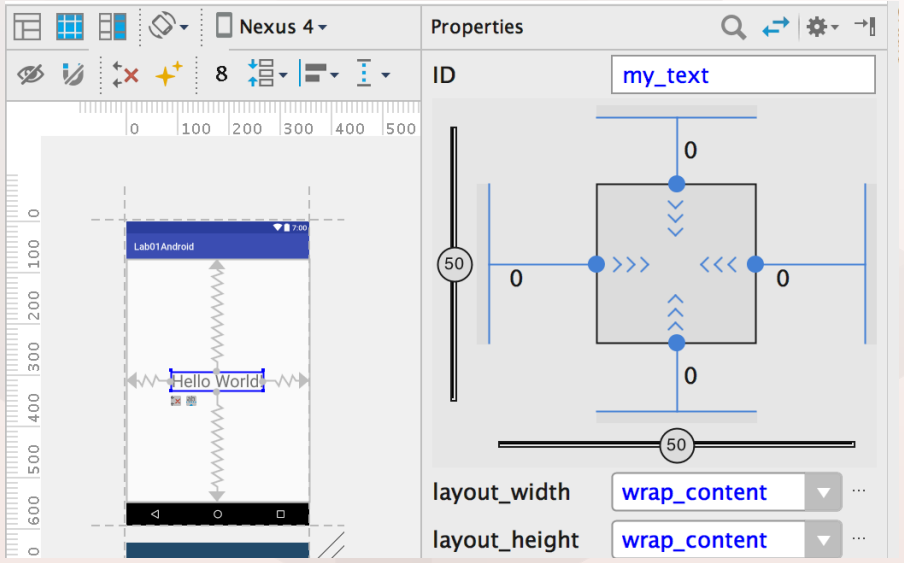
Access to Application Resources via XML

@[+][<*package_name*>:]<*resource_type*>/<*resource_name*>

- **[+]** indicates this is a new resource name that must be created and added
- <**package_name**> is the name of the package in which the resource is located (not required when referencing resources from the same package)
- <**resource_type**> is the **R** subclass for the resource type
- <**resource_name**> is either the resource filename without the extension or the android:name attribute value in the XML element.

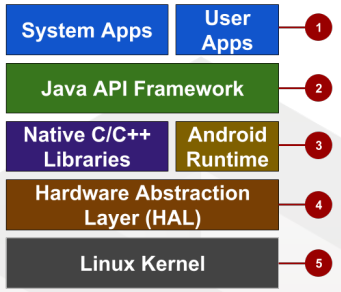
- Again, exactly as we did in Lab1:

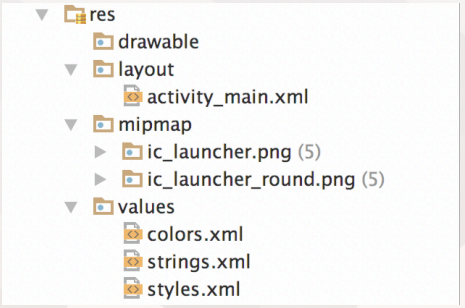
```
<TextView
    android:id="@+id/my_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
```



Class outline

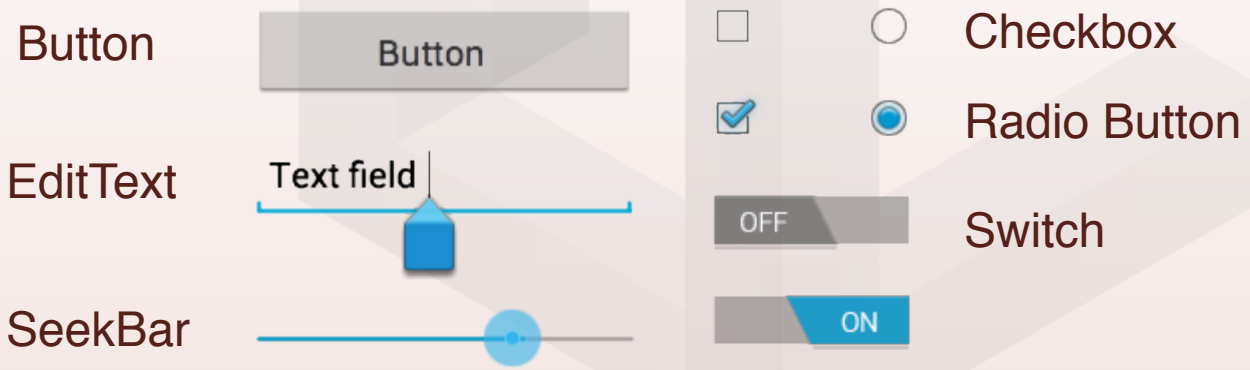
- The Android platform architecture
  - The Android Stack: More in-depth information
  - Android components
- Activities
- Application resources
- **Views and Layouts**
  - Event handling
- Lab of Today





Everything you see, is a View

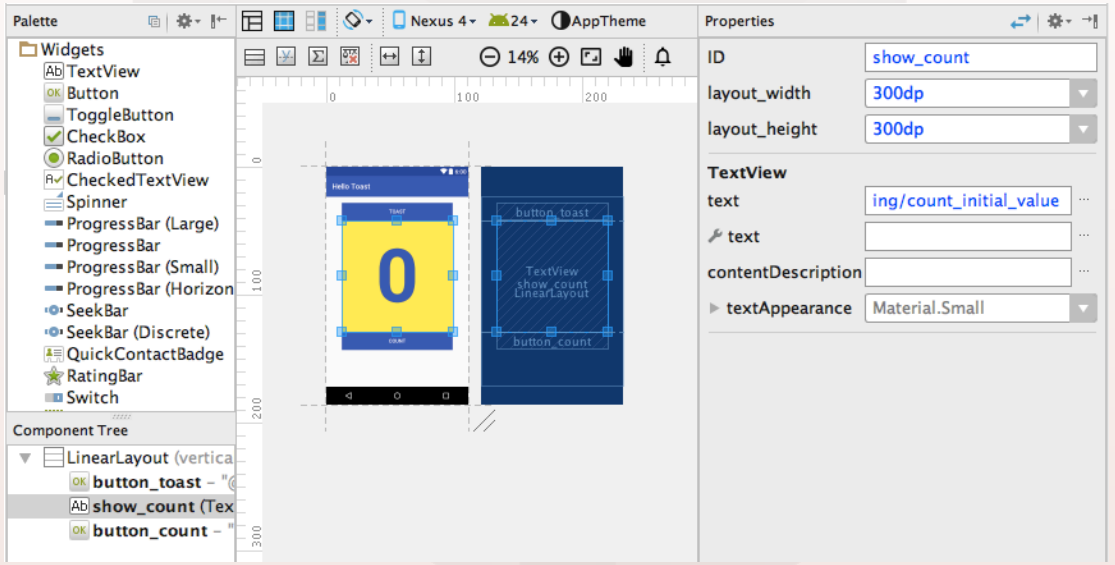
- **View** is the basic building block to create user interfaces
- All elements are subclasses of the View class:
  - display text (TextView class), edit text (EditText class)
  - buttons (Button class), menus, other controls
  - scrollable (ScrollView, RecyclerView)
  - show images (ImageView)



Views properties and creation

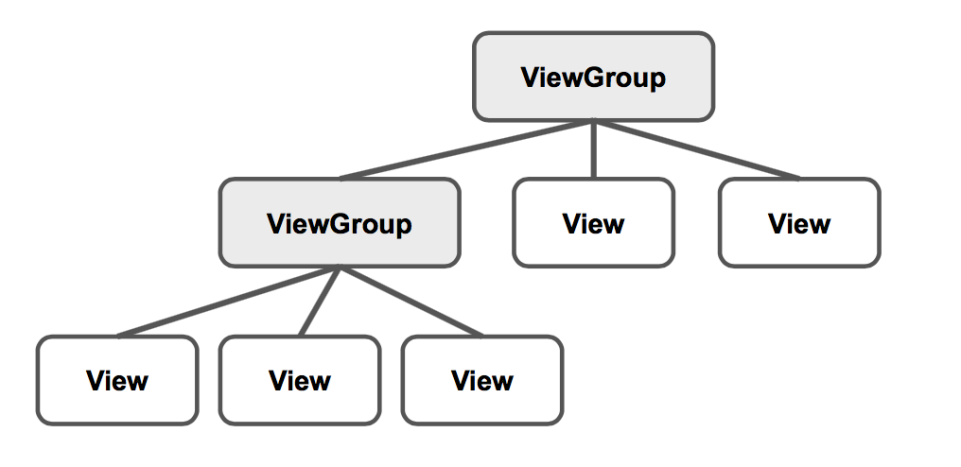
- Views have properties
  - color, dimensions
- They may be interactive
  - respond to clicks
- Have relationships with other views
- Creating views:
  - Via XML files
  - Graphically with Android Studio
    - Graphical representation of XML
  - Programmatically
    - TextView txt = new TextView (this);

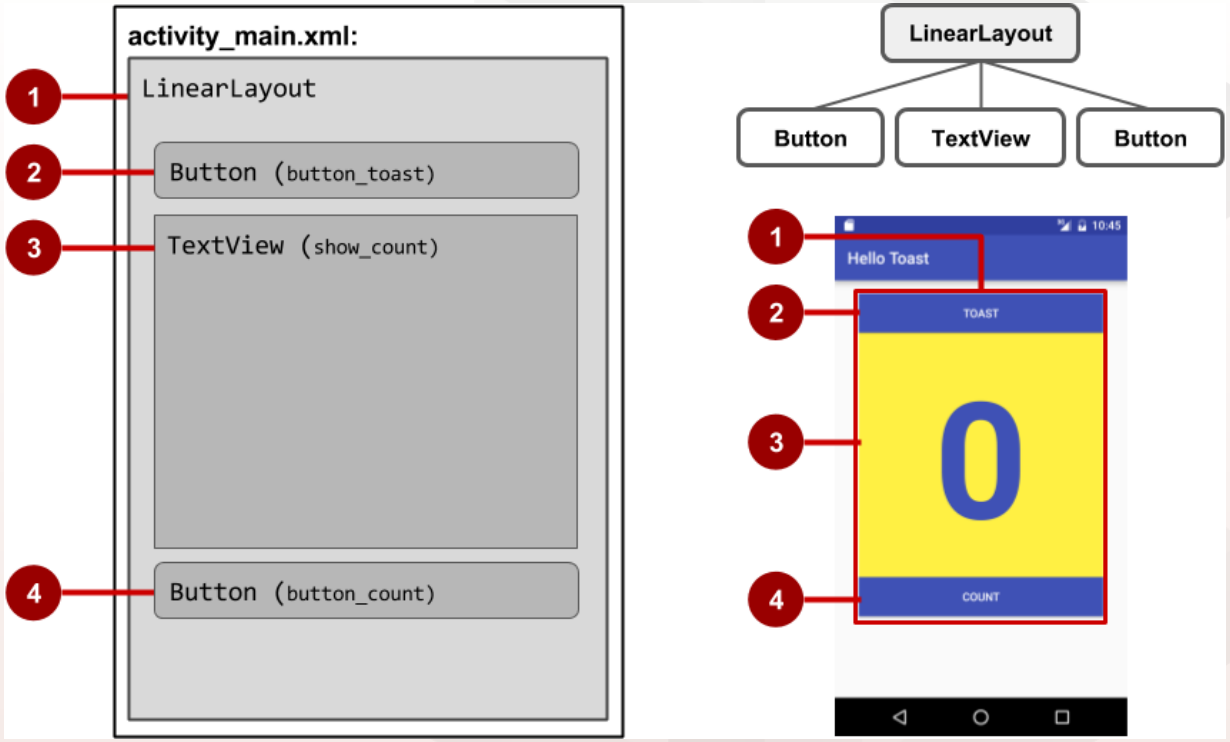
```
<TextView
    android:id="@+id/my_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="36sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



ViewGroup views

- A ViewGroup (parent) is a type of View that can contain other views (children)





Layouts

- Layouts are specific types of ViewGroups
  - Subclasses of ViewGroups

- Several types of layouts exist:
  - ConstraintLayout - connect views with constraints
  - LinearLayout - horizontal or vertical row
  - RelativeLayout - child views relative to each other
  - TableLayout - rows and columns
  - FrameLayout - shows one child of a stack of children

LinearLayout

RelativeLayout

Recap: Referring to Views and Layouts in code

- Same than referring to the application resources
- Layout:
  - R.layout.activity_main
  - setContentView(R.layout.activity_main);
- View:
  - R.id.recyclerView
  - Rv = (RecyclerView) findViewById(R.id.recyclerview);
- String:
  - In Java: R.string.title
  - In XML: android:text="@string/title"

Event handling

- Events (something that happens) are noticed by Android
  - In the UI: Click, tap, drag
- A method, called event handler, does something in response to an event (for example, a click).
- This is done programmatically in XML and Java:

Attach handler to view in layout:

```
android:onClick="showToast"
```

Implement handler in activity:

```
public void showToast(View view) {
  String msg = "Hello Toast!";
  Toast toast = Toast.makeText(
        this, msg, duration);
  toast.show();
  }
}
```

Today's lab (Lab2)

- Building a simple user interface
  - View, ViewGroup,…
- Creating a LinearLayout and adding elements to it.
- Reacting to taps (clicks)
- Using a custom Android Wear Layout
- Using Android Wear Idle display

Questions?