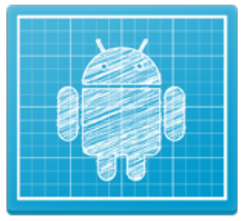
Android Development

Lecture 2
Android Platform



Lecture Summary

- The Android Platform
- Dalvik Virtual Machine
- Application Sandbox
- Security and Permissions
- Traditional Programming compared to Android
- Activities
- Services
- Content Providers
- Intents
- Broadcast Receivers
- Android Application Structure
- Android Application Manifest
- Model View Controller
- User Interface

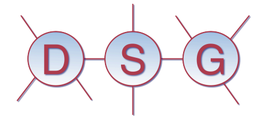
The Android Platform

- Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.
- It is a Linux-based operating system for mobile devices such as smartphones and tablet computers. It is developed by the Open Handset Alliance led by Google.
- The Linux 2.6 kernel handles core system services and acts as hardware abstraction layer (HAL) between the physical hardware and the Android Software Stack.

- Kernel handles:
  - Application permissions and security
  - Low-level energy management
  - Process management and Threading
  - Networking
  - Display, keypad input, camera, Flash memory, audio and binder (IPC) driver access

Google
android

Source: Wikipedia, AWA, http://developer.android.com/guide/developing/tools/aidl.html



Android Application A

Written Using Android Java Framework

DALVIK Virtual Machine

Android Application A

Written Using Android Java Framework

DALVIK Virtual Machine

Android Application A

Written Using Android Java Framework

DALVIK Virtual Machine

Linux User A

Linux User B

Linux User C

Linux 2.6 Operating System
(Hardware Abstraction Layer)

Security

Memory Management

Process Management

Binder IPC

Network Stack

I/O

Display, Keyboard, Touchscreen

Power Management

Other Devices
WiFi, Bluetooth, Camera, Audio, Telephony, Flash, Device Sensors

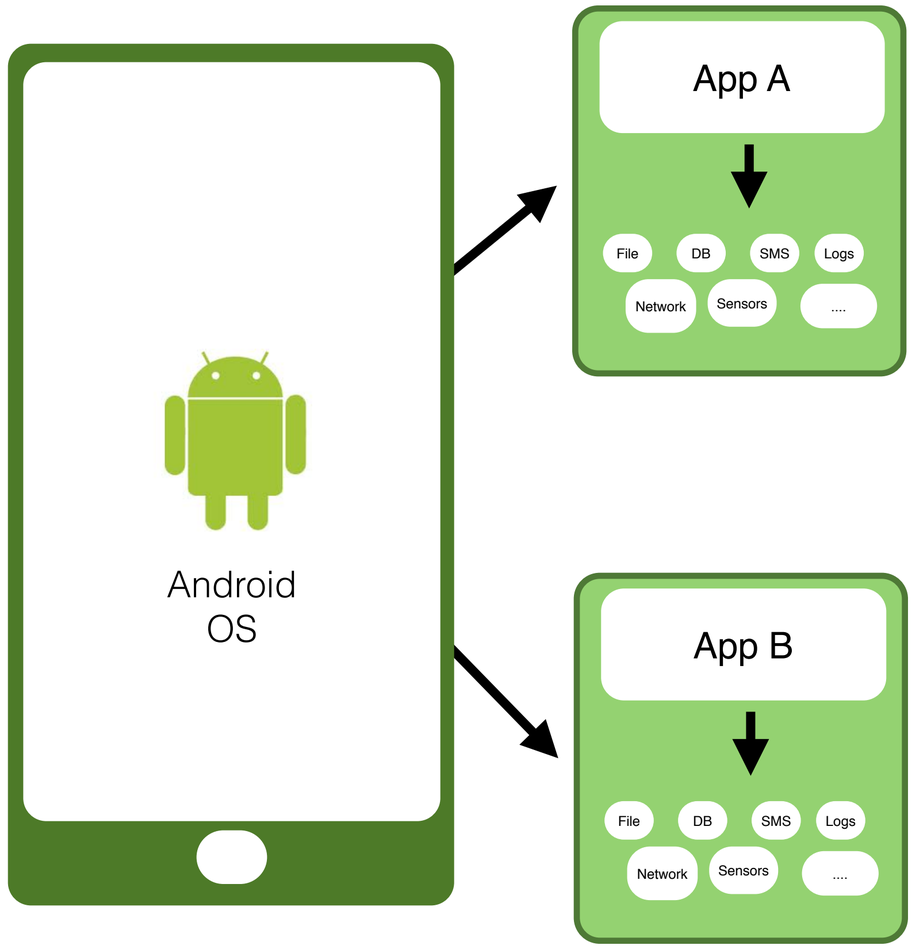
Physical Hardware

Dalvik Virtual Machine

- It is the Android Virtual Machine, based on the Java VM but optimized for mobile devices.
- Has a small memory footprint, and multiple instances of Dalvik VM can run concurrently on the device.
- Programs are mainly written in a dialect of Java and compiled to bytecode. They are converted from Java Virtual Machine-compatible .class files to Dalvik-compatible .dex (Dalvik Executable) files before installation on a device. The compact Dalvik Executable format is designed to be suitable for systems that are constrained in terms of memory and processor speed.
- Each Android application runs in a separate process, with its own instance of the Dalvik virtual machine.
- Android application life cycle (next lecture) enables the OS to enhance the performance of the garbage collection and how it manages memory recovery across multiple heaps.

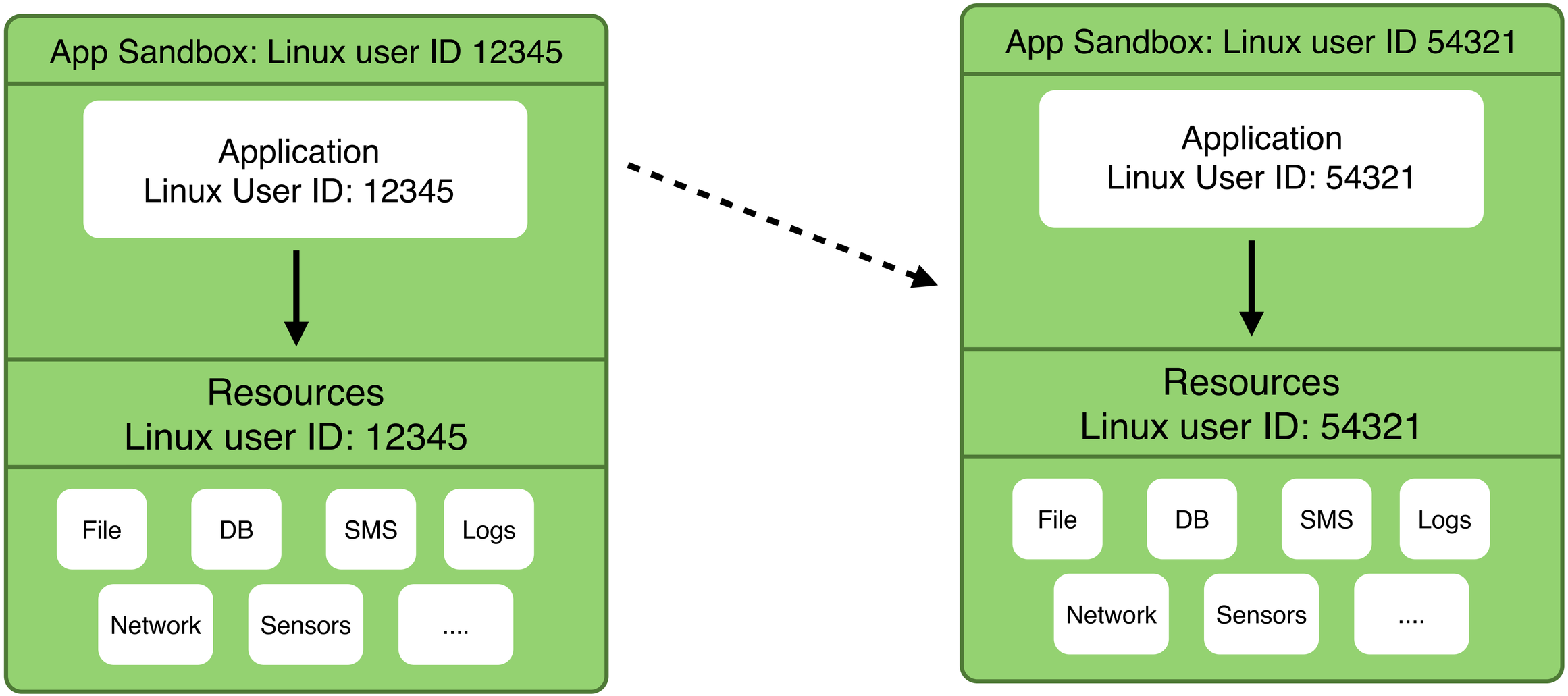
Application Sandbox

- Android security is strictly related to restriction and security levels of the Linux operating system and in particular on process and user-level boundaries.
- Android creates a new user for each application vendor. Each application it is executed with different user privileges (expect for those signed by the same vendor).
- Files owned by one application are, by default, inaccessible by other applications.
- The SDK provides **Content Provider** and **Intent** to allows applications to communicate and exchange data.
- This solution has been adopted to provide an high security in a world of numerous third part small applications from different vendors.
- The sandbox approach is a common solution for mobile operating systems such as iOS.



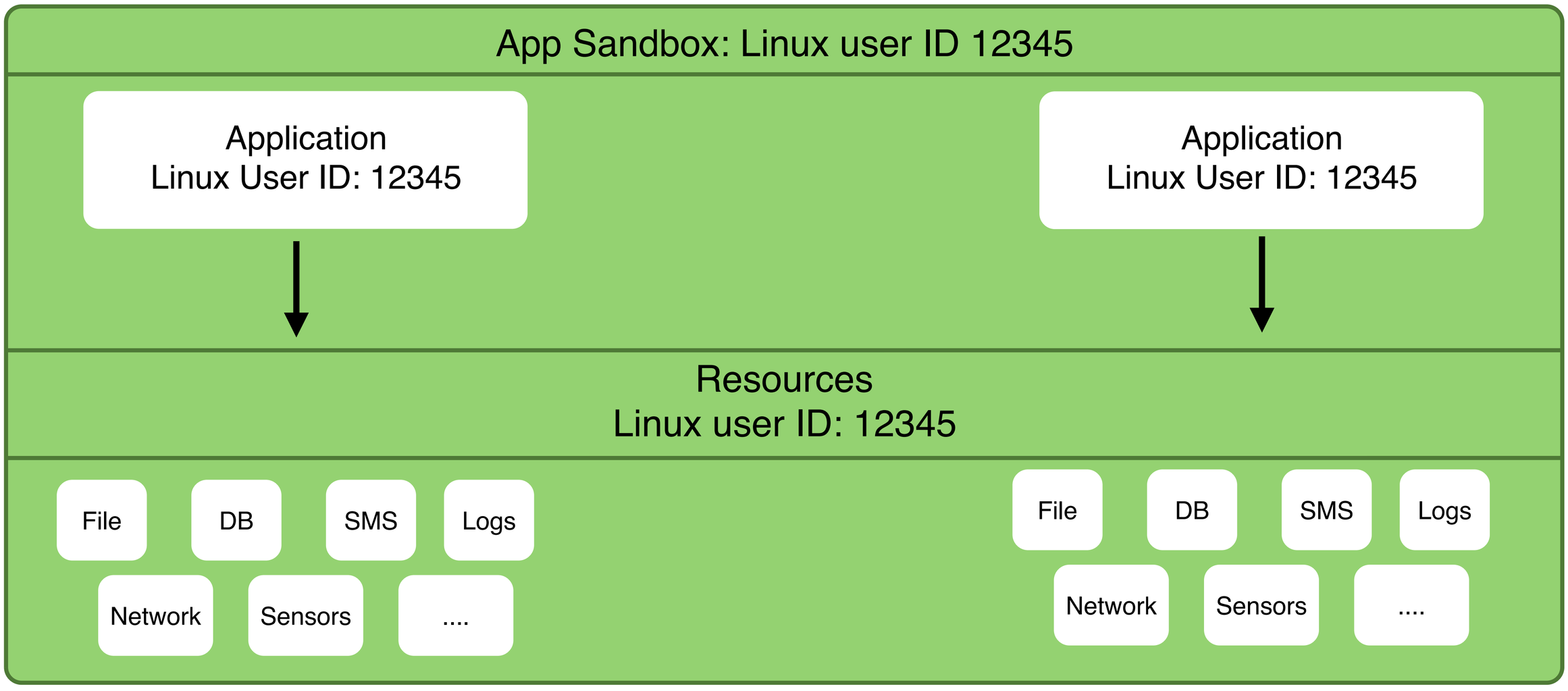
Source: PA-Chapter 3 - http://www.ibm.com/developerworks/xml/library/x-androidsecurity/ - http://source.android.com/tech/security/index.html#the-application-sandbox

Application Sandbox - Same User ID



Source: PA-Chapter 3 - http://www.ibm.com/developerworks/xml/library/x-androidsecurity/ - http://source.android.com/tech/security/index.html#the-application-sandbox

Application Sandbox - Same User ID



Source: PA-Chapter 3 - http://www.ibm.com/developerworks/xml/library/x-androidsecurity/ - http://source.android.com/tech/security/index.html#the-application-sandbox

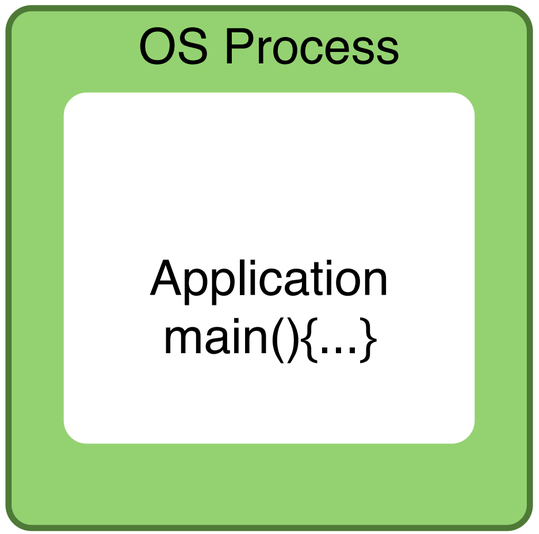
Security and Permissions

- Android platform adopts several security and integrity measures to ensure that the user's data is secure and that the device is note subject to malware.
- Application as Operating System Users (Dalvik instance and sandboxes)
- Application Permissions
  - Each application registers for specific privileges to access shared resources. Some of these privileges are related for example to phone functionalities, make calls, access network file and system, control the camera or the other hardware sensors.
  - Additional permission are dedicated to access private and personal information, such as preferences, user's contacts or location.
- Limited Ad-Hoc Permissions
  - Applications that use a **Content Provider** to openly share specific information to other applications can define some on-the-fly permission to grant o revoke access to a resource.
- Application Signing
  - All application are signed with a certificate to allow the users to know that the application is authentic.

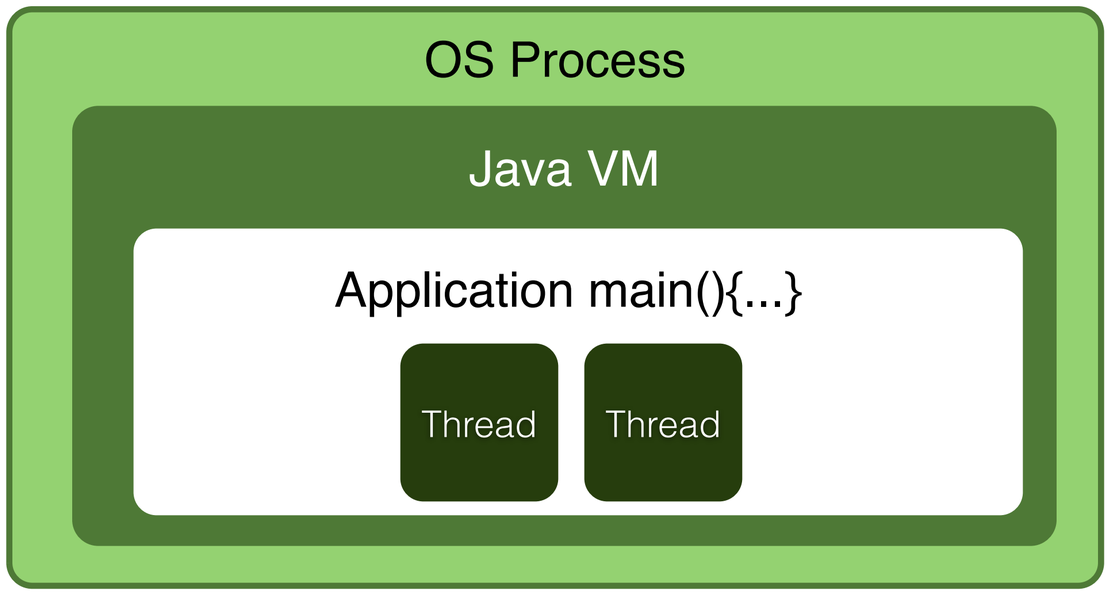
Source: http://developer.android.com/guide/topics/security/security.html

Traditional Programming compared to Android

- Traditional OS applications:
  - use a single entry point (traditionally called main). The OS loads the program into a process and then start executing it (Fig. A).



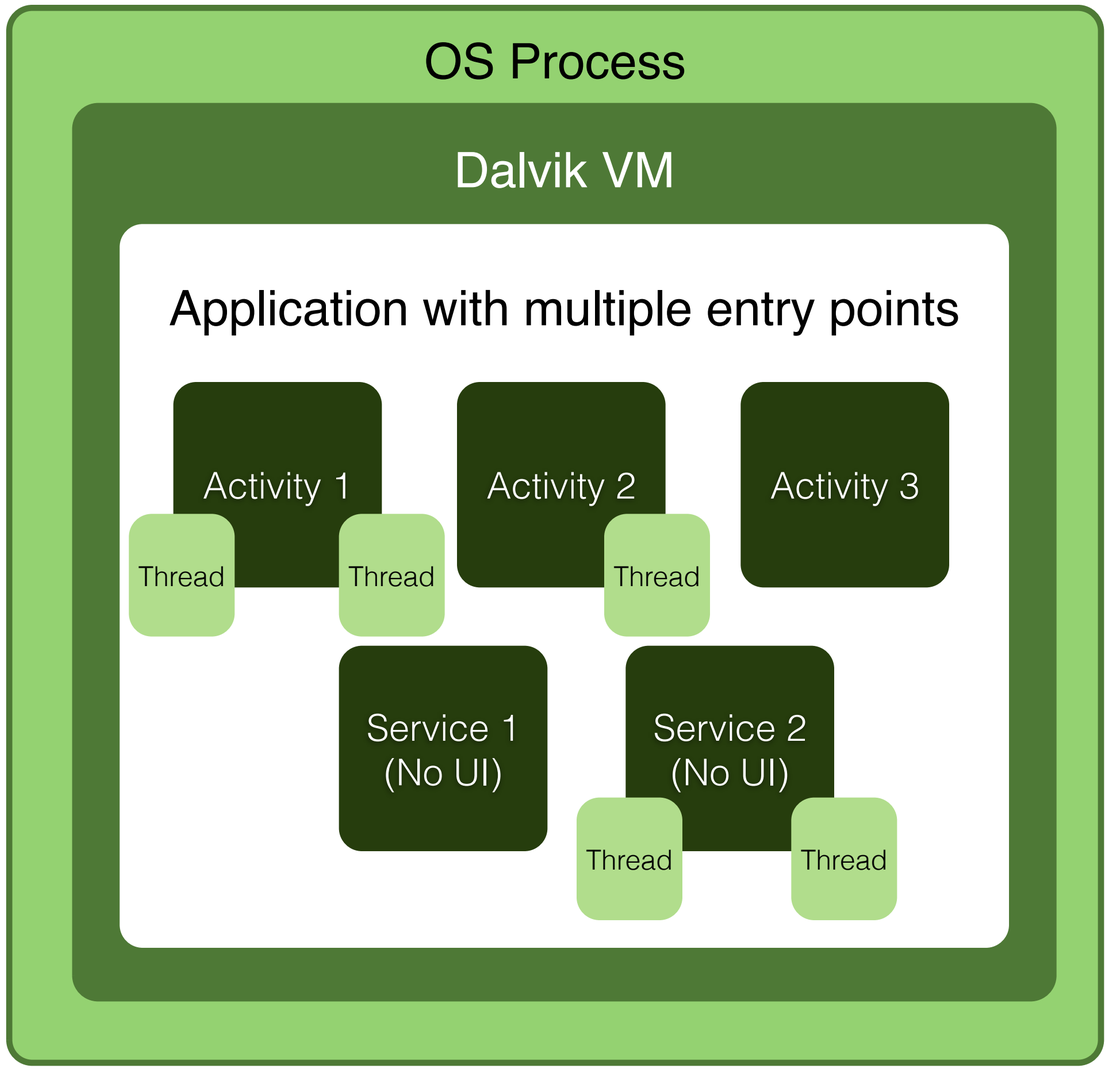
- Java based applications:
  - are managed in a different way. A Java VM is instantiated in a dedicated process to load all classes used by the application and execute it.



Source: http://developer.android.com/guide/topics/fundamentals.html

Traditional Programming compared to Android

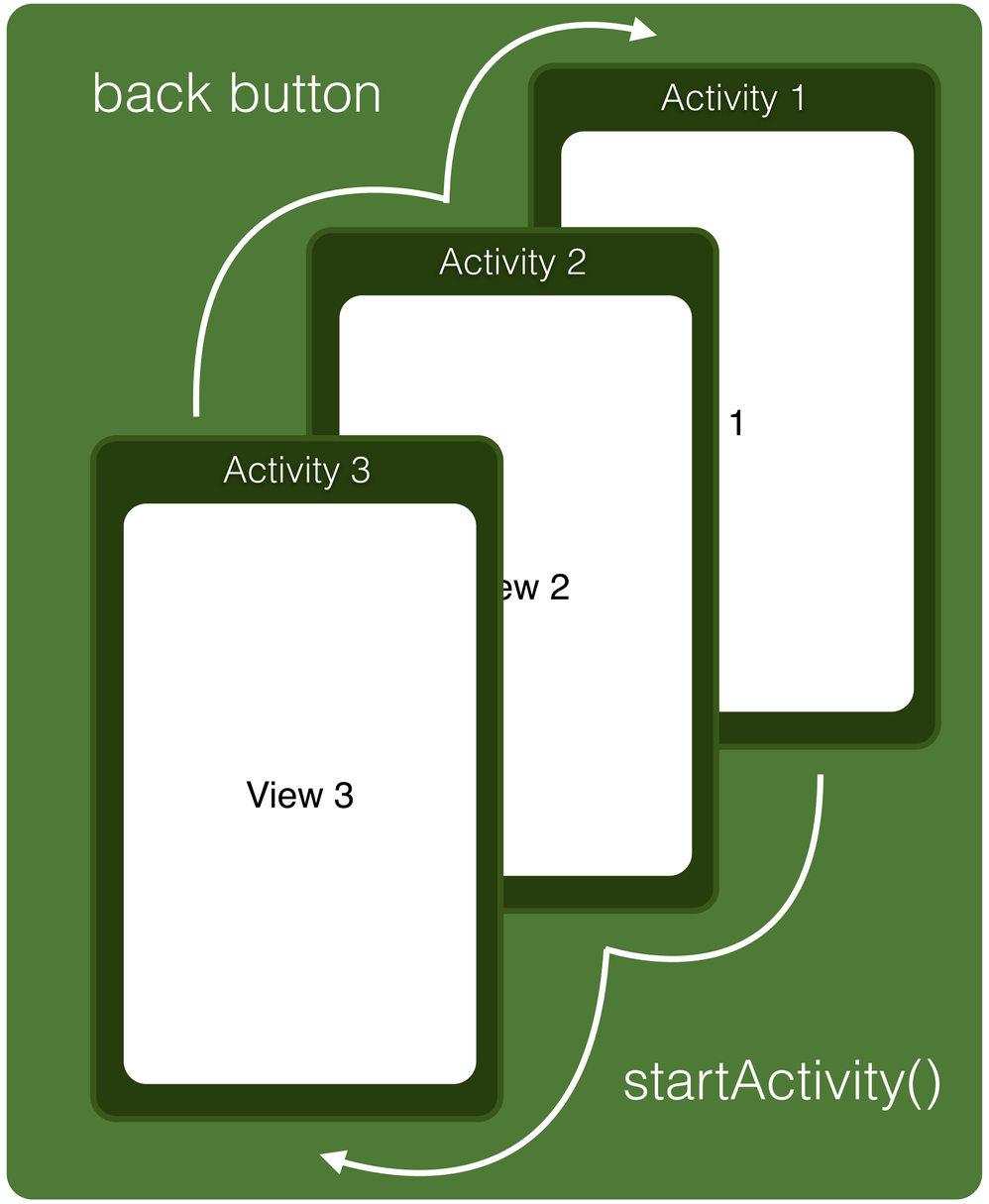
- Android introduces a more complex system supporting multiple application entry points. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role each one is a unique building block that helps define your application's overall behavior.
- There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. Main application entry points are:
  - Activities
  - Services
  - Content Providers
  - Broadcast Receivers



Source: http://developer.android.com/guide/topics/fundamentals.html

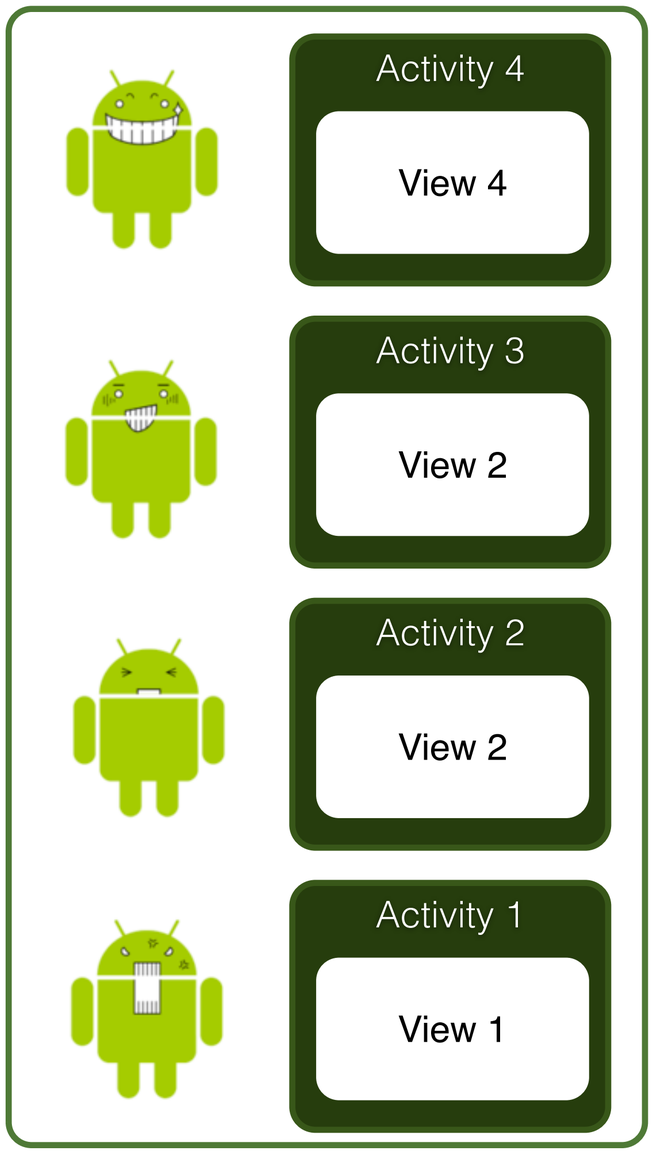
Activities

- Is both a unit of user interaction (typically associated to a View) and a unit of execution.
- To create an activity, you must create a subclass of Activity (or a subclass of it such as MapActivity).
- Multiple Activities (related also to different applications) are organized in a stack structure where new elements can be pushed or removed after specific user actions (UI interaction or back button).
- Although activities work together to form a cohesive user experience in application, each one is independent of the others. As such, a different application can start any one of these activities. For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.



Source: http://developer.android.com/guide/topics/fundamentals.html

Activity Life Cycle



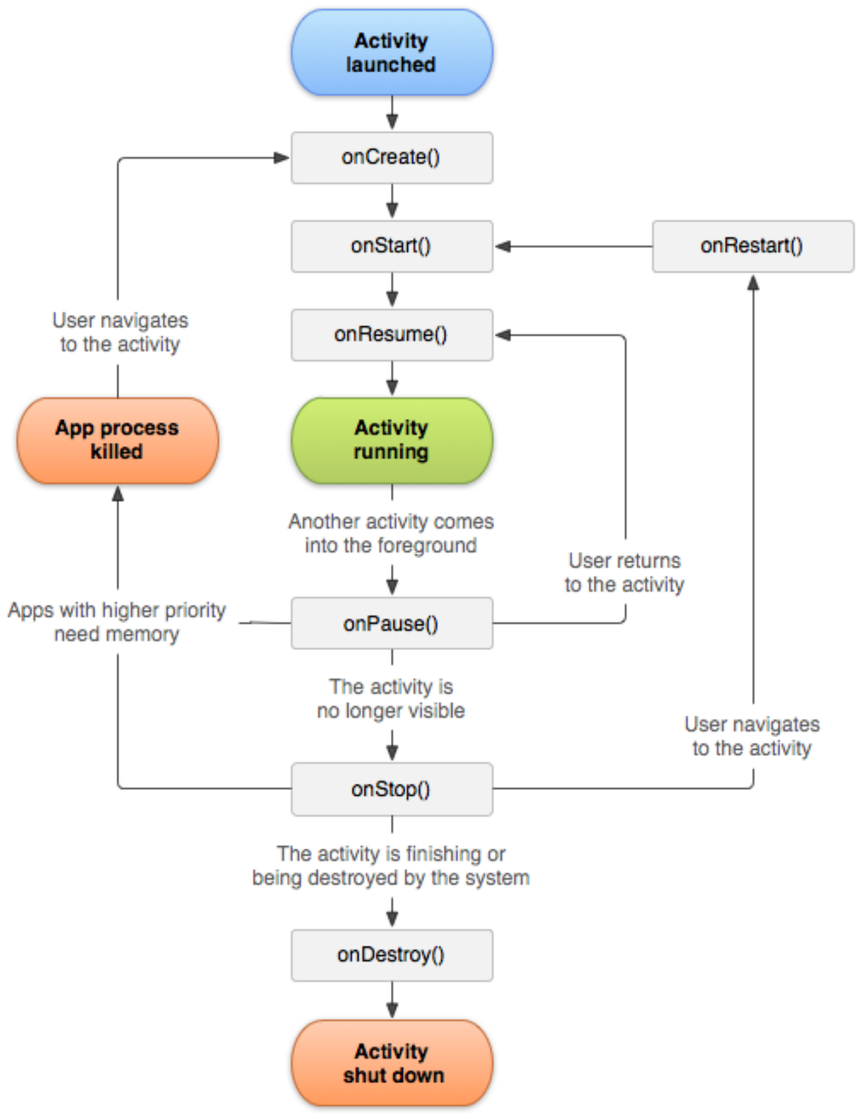
Top Activity.
User can interact with it !

Second Activity in the stack.
If the user clicks back or the Top Activity is destroyed the user can see and interact with it again !

Activity in the middle of the stack.
User cannot interact with it until every activity above it is destroyed.

Activity at the bottom of the stack.
If Activities above it use too many resources it will be destroyed.

Activity Life Cycle

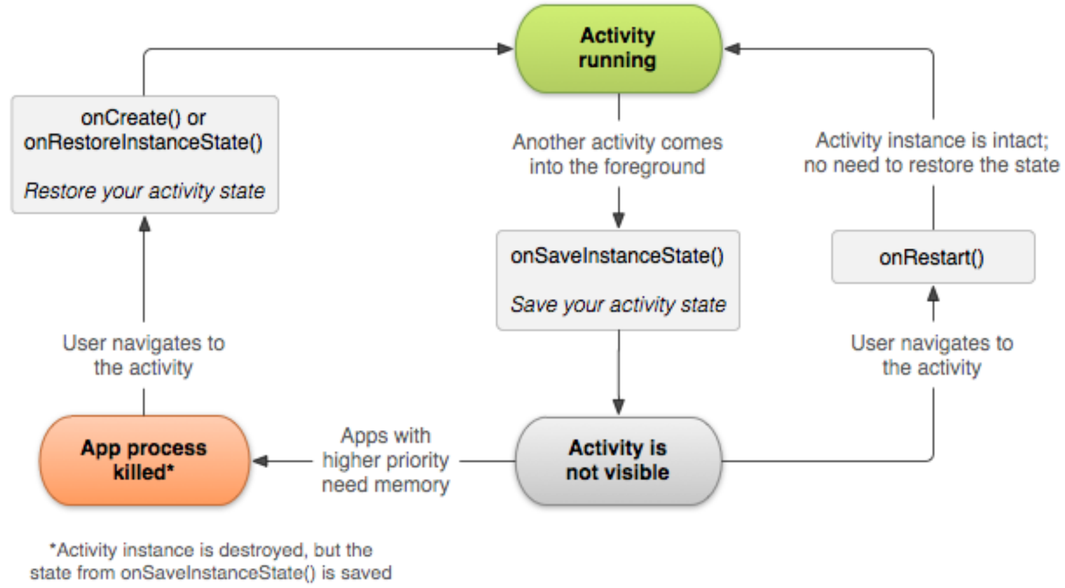


Method	Description	Killable after?	Next
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up — create views, bind data to lists, and so on.	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it being started again.	No	onStart()
onStart()	Called just before the activity becomes visible to the user.		onResume() or onStop()
onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it.	No	onPause()
onPause()	Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns.	Yes	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.	Yes	onRestart() or onDestroy()
onDestroy()	Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing, or because the system is temporarily destroying this instance of the activity to save space.	Yes	nothing

Source: http://developer.android.com/guide/topics/fundamentals/activities.html

Saving Activity State

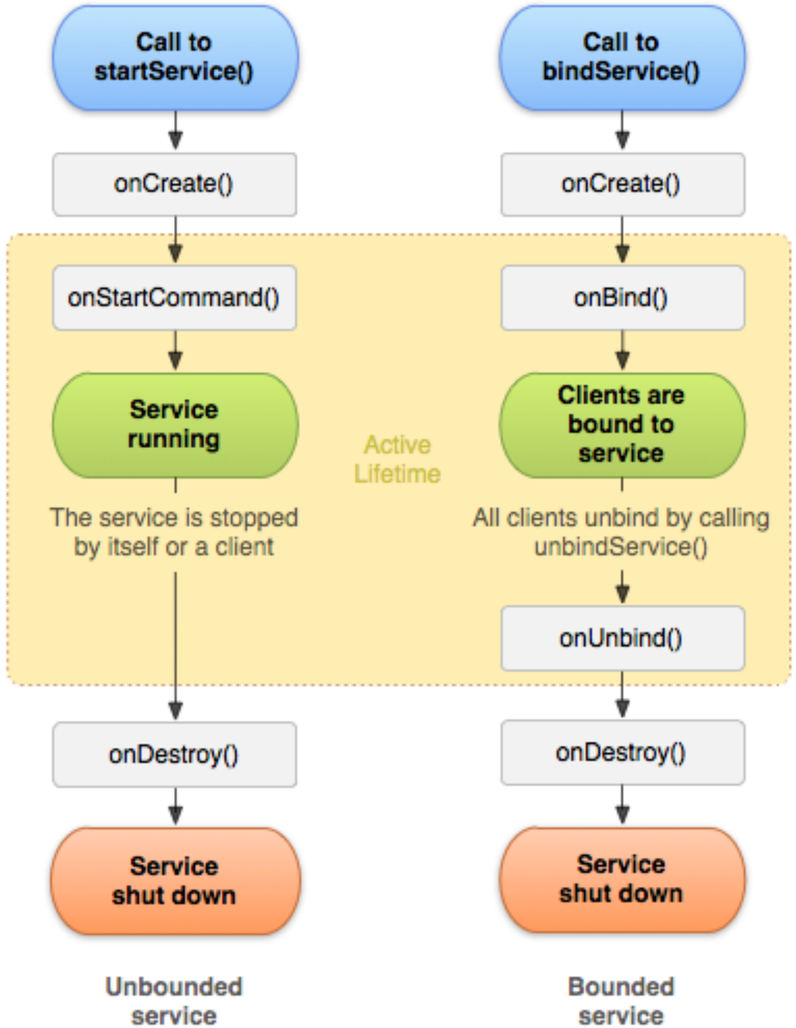
- When the system destroys an activity in order to recover memory, the Activity object is destroyed, so the system cannot simply resume it with its state intact.
- If the user navigates back to the destroyed Activity the system needs to recreate the object. The user is unaware that the system destroyed the activity and recreated it and, thus, probably expects the activity to be exactly as it was.
- In this situation Android OS provides a method called onSaveInstanceState() ensure that important information about the activity state is preserved.



Source: http://developer.android.com/guide/topics/fundamentals/activities.html

Service

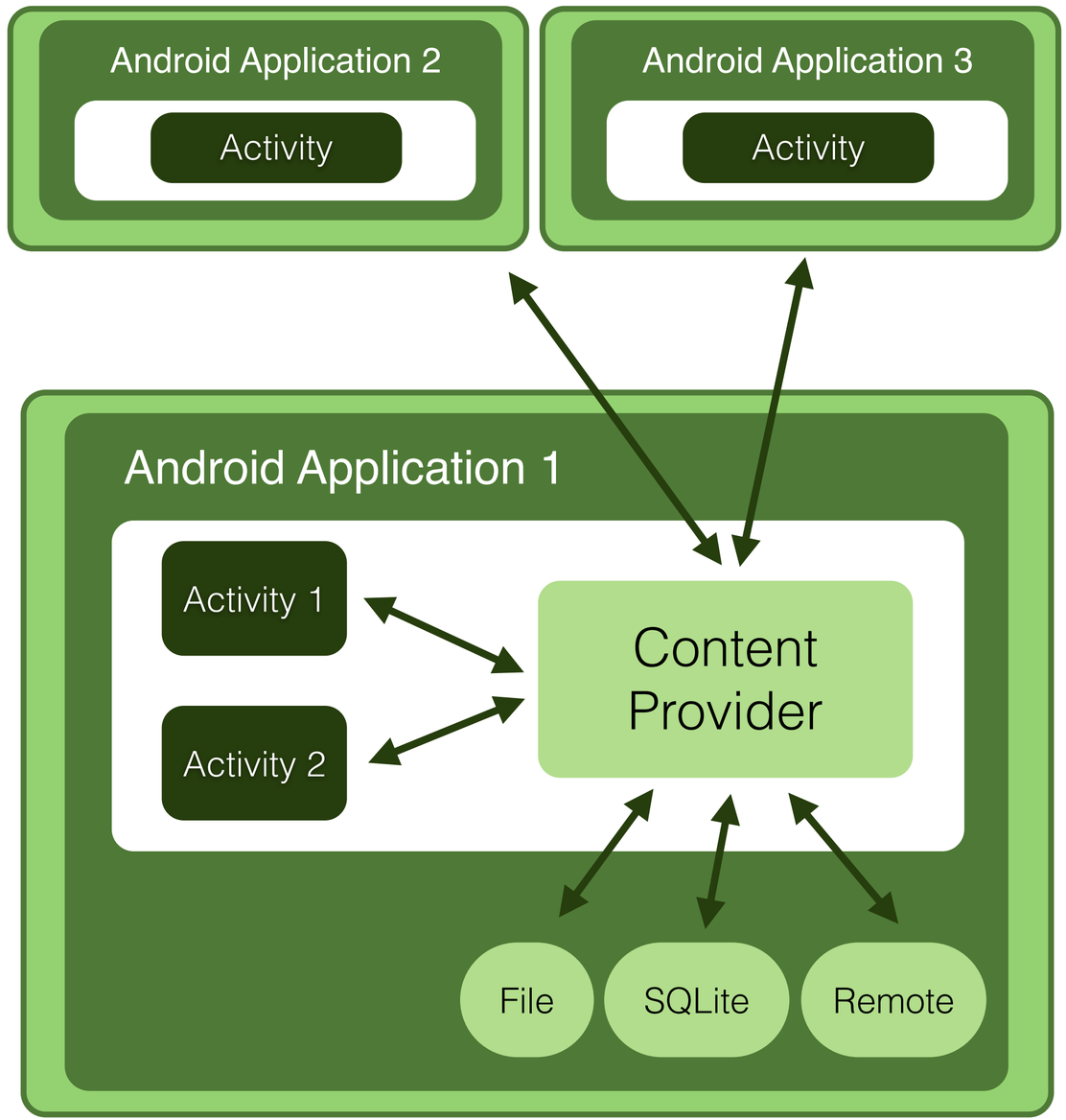
- The Android Service class is for background operations that are active but not visible to the user.
- It is used too perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use (chat service, HTTP fetcher, GPS Location tracking).
- 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.
- Has a life cycle similar to the one presented for the Activity and has two main method to control its life such as start, stop and restart.



Source: http://developer.android.com/guide/topics/fundamentals/services.html

Content Providers

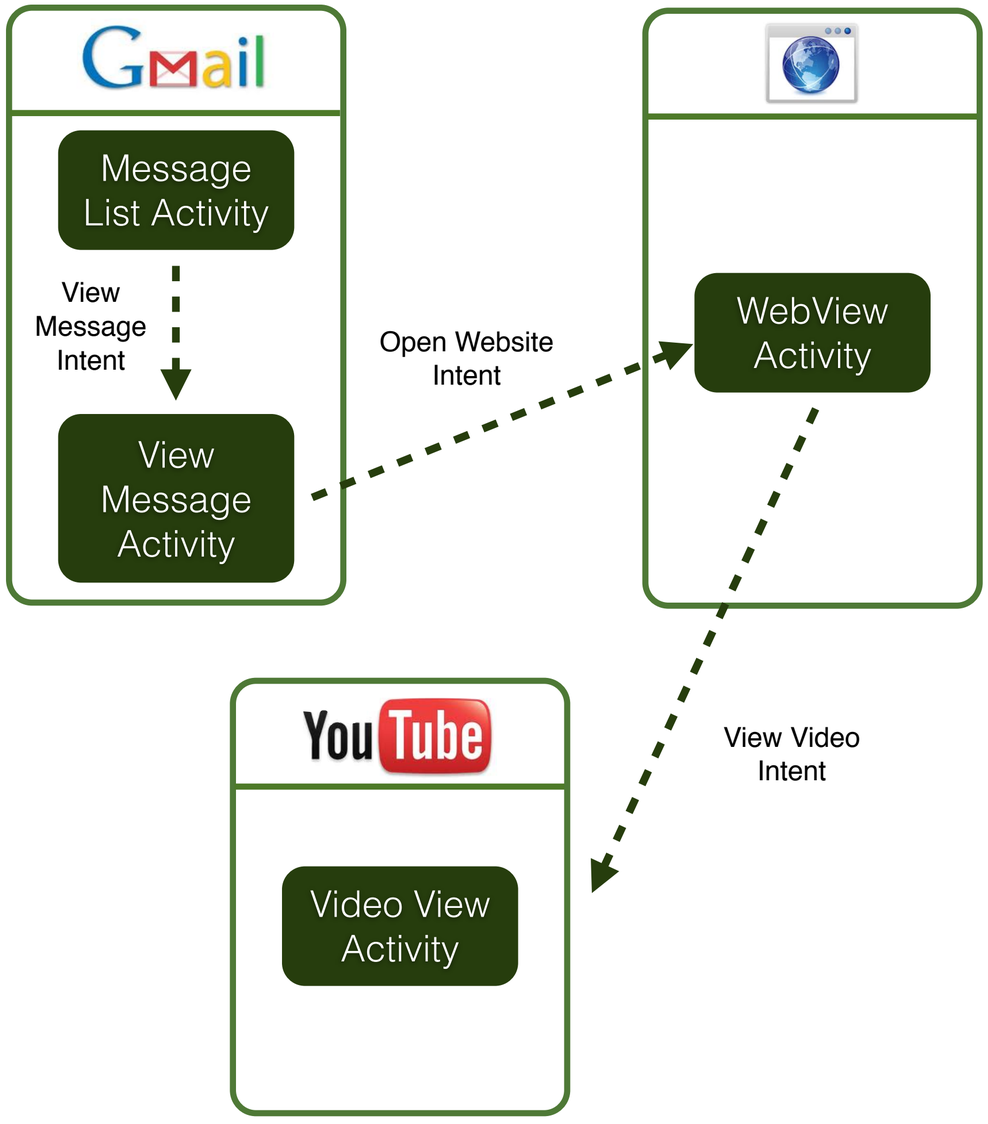
- Content Providers (CPs) manage access to a structured set of data (Files or Databases).
- They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process.
- Other applications can access information of a content provider through a specific URI (*content://*).
- Typical provided operations are:
  - Create [Insert]
  - Read [Query]
  - Update
  - Delete
- Android OS provides some built-in CPs for Browser, Calendar, Contacts, Call Log, Media and Setting allowing the access to those data from other active applications.



Source: PA-Chapter3 - http://developer.android.com/guide/topics/providers/content-providers.html

Intents

- Intents are asynchronous messages allowing OS components to request functionality from other system elements. For example an Activity can send an Intent to the Android system to starts another Activity.
- Intents can be used to signal to the Android system that a certain event has occurred. Other components in Android can register to this event and will get notified.
- Intents are send to the Android system. Depending on how the Intent was constructed the Android system will run an receiver determination and determine what to do.
- An Intent can also contain data. This data can be used by the receiving component. For example your application can calls via an Intent a browser component. As data is it may send the URL to the browser component.
- Android supports explicit and implicit Intents.



Source: http://developer.android.com/guide/topics/intents/intents-filters.html / http://www.vogella.de/articles/AndroidIntent/article.html

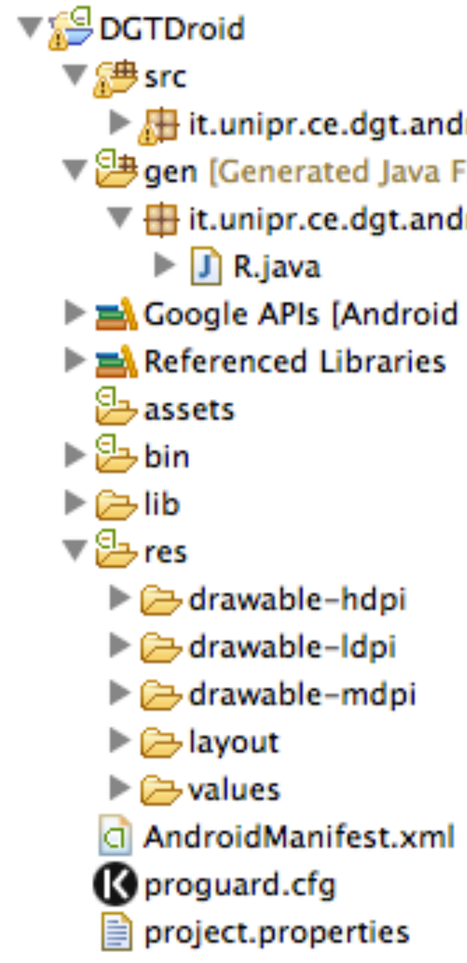
Broadcast Receiver

- Represent a variant of communication between different process based on Intent objects.
- An application or directly the system
- 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.
- They do not display a user interface, but may create a status bar notification to alert the user when a broadcast event occurs.
- To simplify a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

Source: PA-Chapter3 / http://developer.android.com/guide/topics/fundamentals.html

Android Application Structure

- *Android Manifest.xml*
- *res/*
  - *layout/* [application layout files]
  - *drawable/* [images, patches, drawable, xml]
  - *raw/* [data files that can be loaded as streams]
  - *values/* [xml files with strings, number values used in the code for example to localize the application in difference languages]
- *src/*
  - *java/package/directories*
- *gen/* [directory generated by Eclipse and Android SDK]



Android Application Context

- The Application Context is central entity accessible by all top applications containing global information about an application and the environment.
- It is an abstract class provided by Android OS allowing to access application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents.
- Methods in the Context can be directly called by Activity and Service classes because they directly extend Context class.
- Some examples of Context features are:
  - retrieving application resources
  - accessing application preferences
  - lunch activities
  - request a system service (for example Location Service)
  - manage private application files, directories and databases
  - inspect and enforce application permissions

Source: Wikipedia, AWA, http://developer.android.com/guide/developing/tools/aidl.html

Android Application Manifest

- Android Application describes their content in an XML file called *AndroidManifest.xml*.
- Using this file the developer can declare the presence of Activities, Services, Content Provides, Broadcast Receiver, Permissions and other elements such as code version, app name, SDK requirements and .
- Manifest files all applications are shared with the operating system allowing to load and execute them in a managed environment.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="it.unipr.ce.dgt.android"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GPS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
    <uses-library android:name="com.google.android.maps" />

    <activity android:name=".WarningMessageActivity"></activity>
    <activity android:name=".IncomingWarningMessageActivity"></activity>
    <activity android:name=".PeerInfoActivity"></activity>

        <activity android:name=".DGTDroidActivity"
                  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>
```

R Object

- Eclipse and Android SDK work together to create the directory *gen*. It contains a specific class called **R**, which is located inside the Java application package named in the Android Manifest.
- This class contains fields that uniquely identify all resources in the application package structure.
- Using the **Context** object (**Context.getResources**) it is possible to obtain an instance of **android.content.res.Resources** that directly contains application resources.
- R object is also used to retrieve UI layout files and related UI elements.

```
/* 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 it.unipr.ce.dgt.android;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int accident_marker=0x7f020000;
         ...
    }
    public static final class id {
        public static final int accidentButton=0x7f050018;
        ...
    }
    public static final class layout {
        public static final int alert_message_dialog=0x7f030000;
         ...
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        ....
    }
}
```

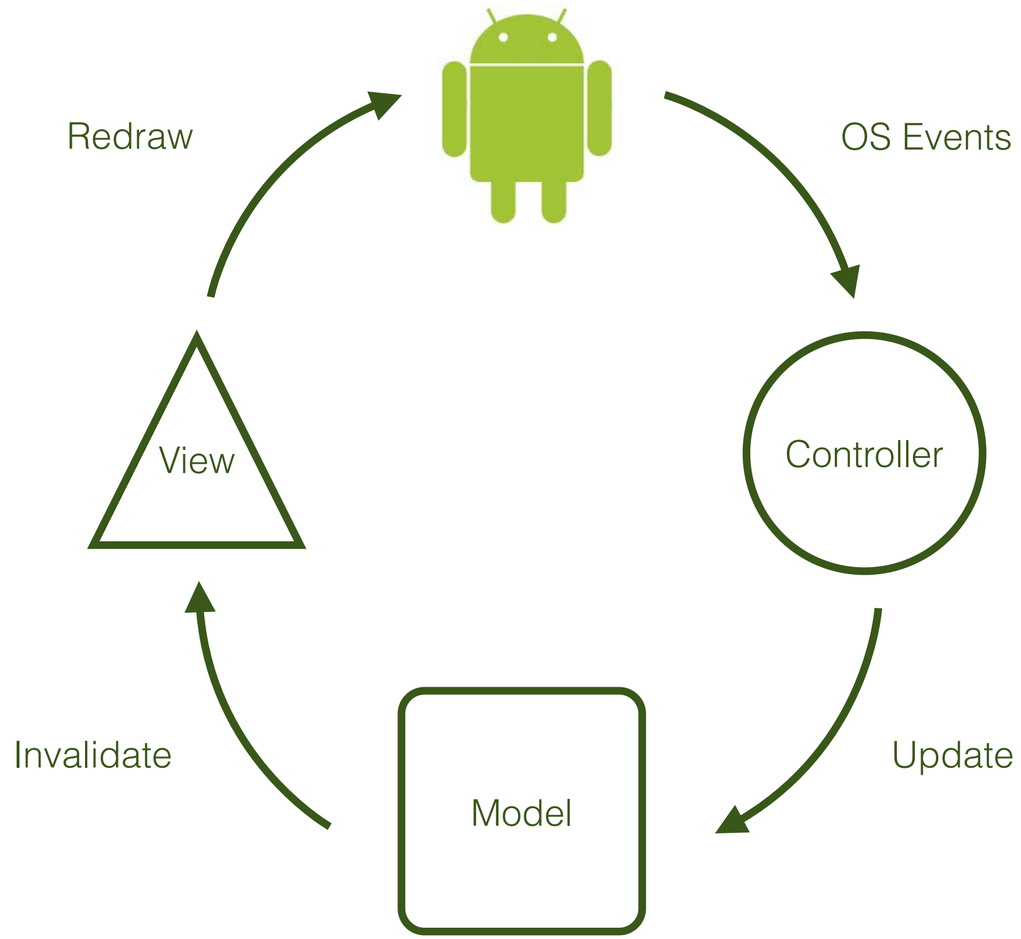
Application Logs

- Android framework provides a structured way to allows developer to log event related to his application using the class called **Log**.
- Log class provides five static log method with the same structure:
  - Log.v(TAG,”<log>”) - VERBOSE
  - Log.d(TAG,”<log>”) - DEBUG
  - Log.i(TAG,”<log>”) - INFO
  - Log.w(TAG,”<log>”) - WARNING
  - Log.e(TAG,”<log>”) - ERROR
- The order in terms of verbosity, from least to most is ERROR, WARN, INFO, DEBUG, VERBOSE. Verbose should never be compiled into an application except during development. Debug logs are compiled in but stripped at runtime. Error, warning and info logs are always kept.
- A good approach is to declare a TAG constant in you class and use it in your log calls.

```
private static final String TAG = "MyActivity";
```

Model View Controller

- Model-View-Controller (MVC) is a software architectural pattern used to isolates the user interface (input and presentation) from the "logic" of an application (the application logic for the user).
- The use of MVC permitting independent development, testing and maintenance of different component.
- Android UI Framework is, like other Java framework, organized around a common MVC pattern.
- It provides structure and tools for building a Controller that handle use input (key pressed and screen taps) and a View that renders graphical information to the screen.

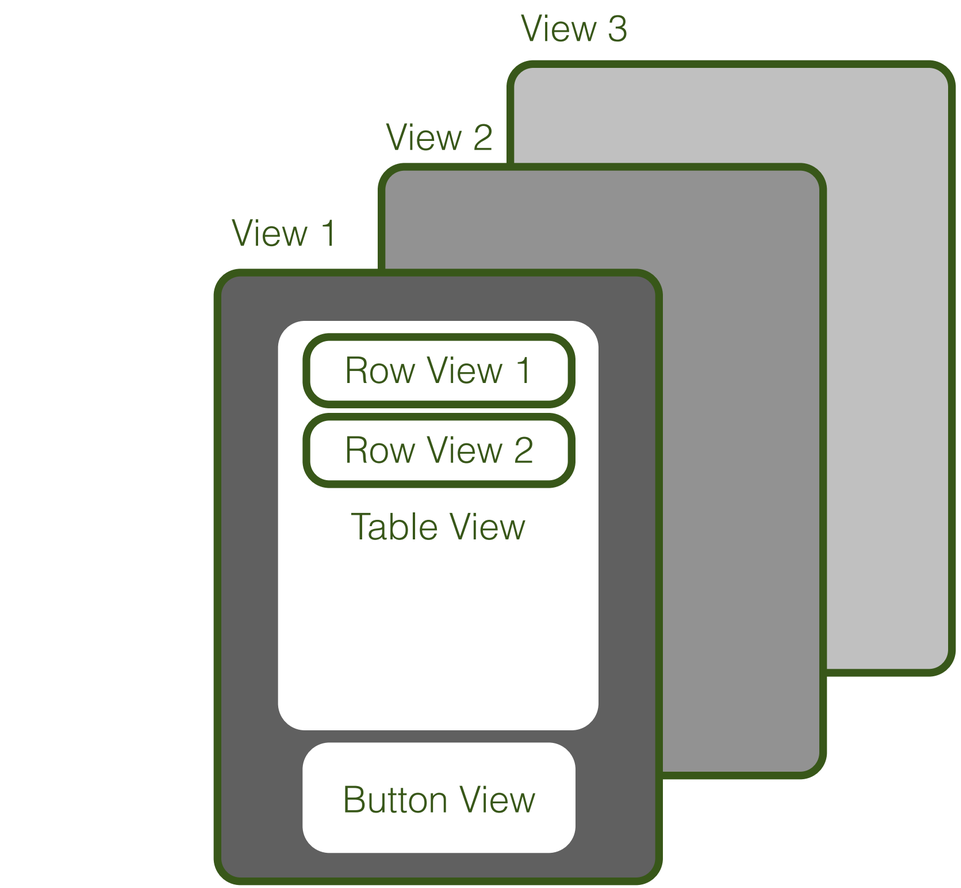


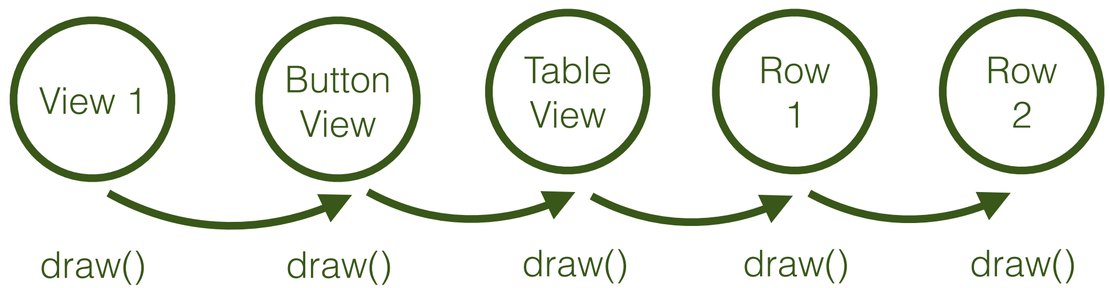
Model

- The model manages the behavior and data of the application. It responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).
- In event-driven systems, the model notifies observers (usually views) when the information changes so that they can react.
- The model is all about application data. It provides all methods needed to access application data related contained for example in a database, a file or a generic data structure.
- While particular application's View and Controller will necessary reflect the Model they manipulate, a single Model might be used by several different application.
- Simple examples could be the list of contact addresses or the database of music on your device. Applications such as a MP3 Player and MP3 Convert have different functionalities but are both based on the same model related to MP3 file format.

View

- The View is the visualization of the model. Generally speaking a view is an application's component responsible for rendering the display, sending audio to the speaker, generate tactile feedbacks, and so on.
- In Android framework, like other existing platforms, the framework paints the screen by walking the view tree, asking each component to draw itself in a preorder traversal. Each View draw itself and then asks each of its children to do the same.
- A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A View object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides.
- As an object in the user interface, a View is also a point of interaction for the user and the receiver of the interaction events.

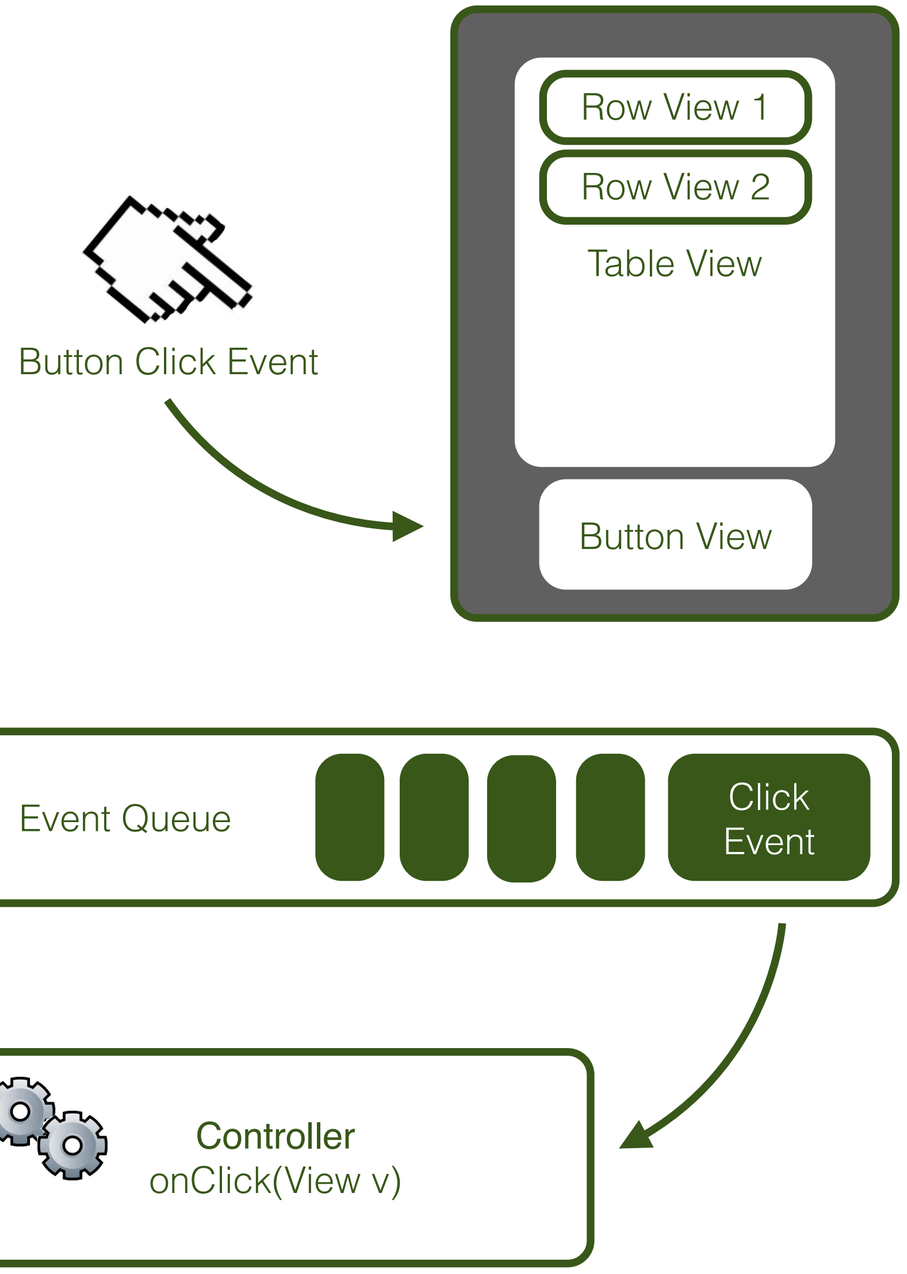




Source: http://developer.android.com/guide/topics/ui/how-android-draws.html

Controller

- The Controller is the application's portion that responds to external events: screen tap, key pressed, incoming call, etc.
- It is implemented as an event queue. Each external action is represented as a unique event in the queue. The framework removes each event from the queue and dispatch it calling the right handler method of the view related to the event.
- For example in an MP3 player application, for instance, when the user taps a Play/Pause button, the related event is dispatched to that button object and its handler can update the Model to resume playing some previously selected song.



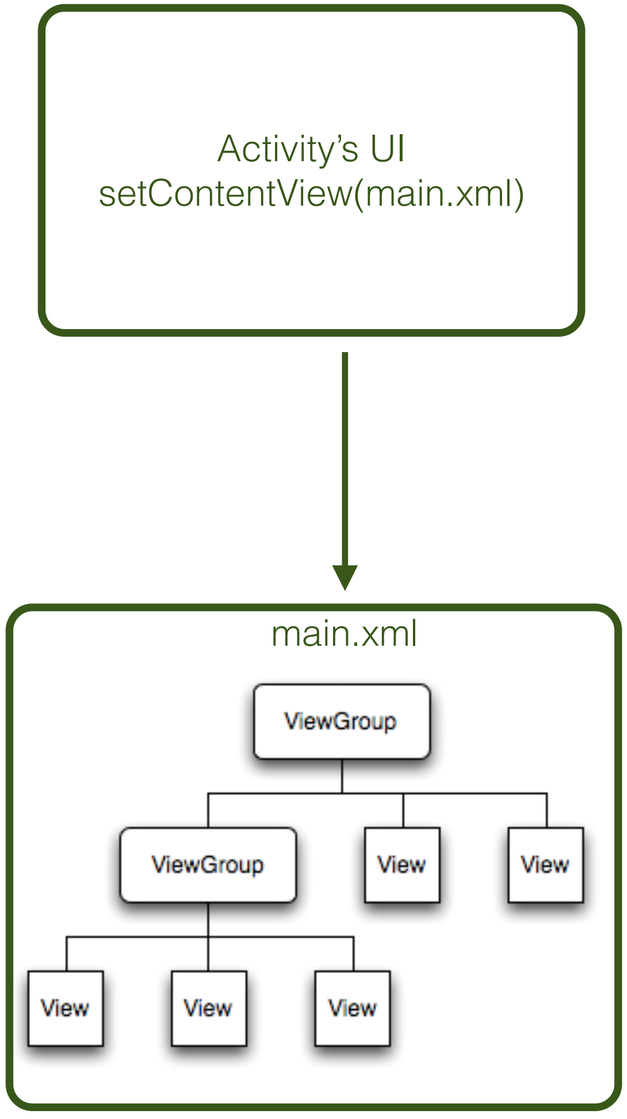
Android User Interface

- The user interface of Android application is built using View and ViewGroup objects. The framework provides several types of views and view groups, each of which is a descendant of the View class.
- The View class is the base for subclasses called "widgets," which offer fully implemented UI objects, like text fields and buttons.
- The ViewGroup class serves as the base for subclasses called "layouts," which offer different kinds of layout architecture, like linear, tabular and relative.
- A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A View object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides. As an object in the user interface, a View is also a point of interaction for the user and the receiver of the interaction events.

Source: http://developer.android.com/guide/topics/ui/index.html

View Hierarchy

- On the Android platform, you define an Activity's UI using a hierarchy of View and ViewGroup nodes. This hierarchy tree can be as simple or complex as you need it to be, and you can build it up using Android's set of predefined widgets and layouts, or with custom Views that you create yourself.
- View objects are leaves in the tree, ViewGroup objects are branches in the tree (see the View Hierarchy figure above).
- In order to attach the view hierarchy tree to the screen for rendering, your Activity must call the setContentView() method and pass a reference to the root node object (main layout xml).
- As previously described The Android system receives this reference and uses it to invalidate, measure, and draw the tree. The root node of the hierarchy requests that its child nodes draw themselves — in turn, each view group node is responsible for calling upon each of its own child views to draw themselves.
- Because these are drawn in-order, if there are elements that overlap positions, the last one to be drawn will lie on top of others previously drawn to that space.



Source: http://developer.android.com/guide/topics/ui/index.html

Layout

```
<?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:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

- The most common way to define your layout and express the view hierarchy is with an XML layout file. XML offers a human-readable structure for the layout, much like HTML.
- Each element in XML is either a View or ViewGroup object (or descendant thereof).
- There are a variety of ways in which you can layout your views. Using more and different kinds of view groups, you can structure child views and view groups in an infinite number of ways. Some pre-defined view groups offered by Android (called layouts) include LinearLayout, RelativeLayout, TableLayout, GridLayout and others. Each offers a unique set of layout parameters that are used to define the positions of child views and layout structure.

Layout Attributes

- Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object (for example, TextView supports the textSize attribute), but these attributes are also inherited by any View objects that may extend this class.
- Some are common to all View objects, because they are inherited from the root View class (like the id attribute).
- Other attributes are considered "layout parameters," which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object.
- Layout attributes can be changed directly in the XML file or using the dedicated framework API when the View object reference has been retrieved in the code.

Layout ID Attribute

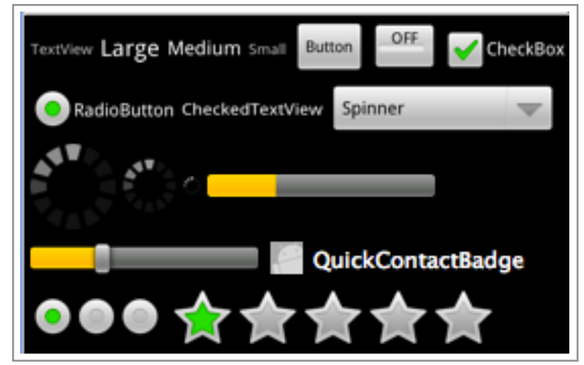
- Any View object may have an integer **ID** associated with it, to **uniquely identify the View within the tree**.
- When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute.
- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource. The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the R.java file).
- When the resource is added to the R object it can be used to retrieve the View element in an Activity using the method context.findViewById(R.<resource_id>);

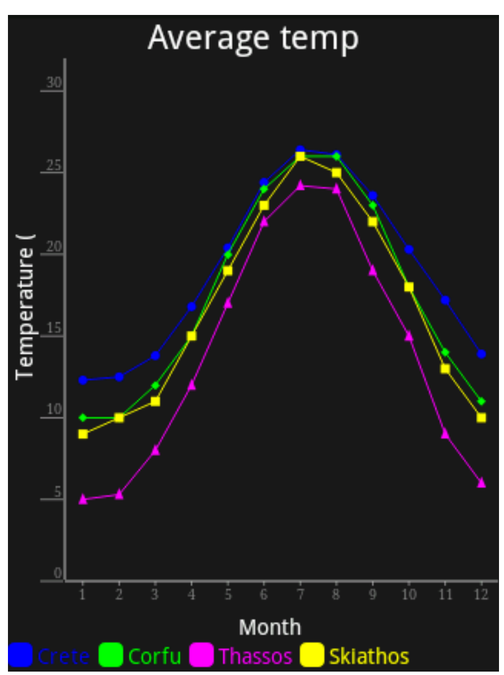
```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

```
Button myButton = (Button) findViewById(R.id.my_button);
```

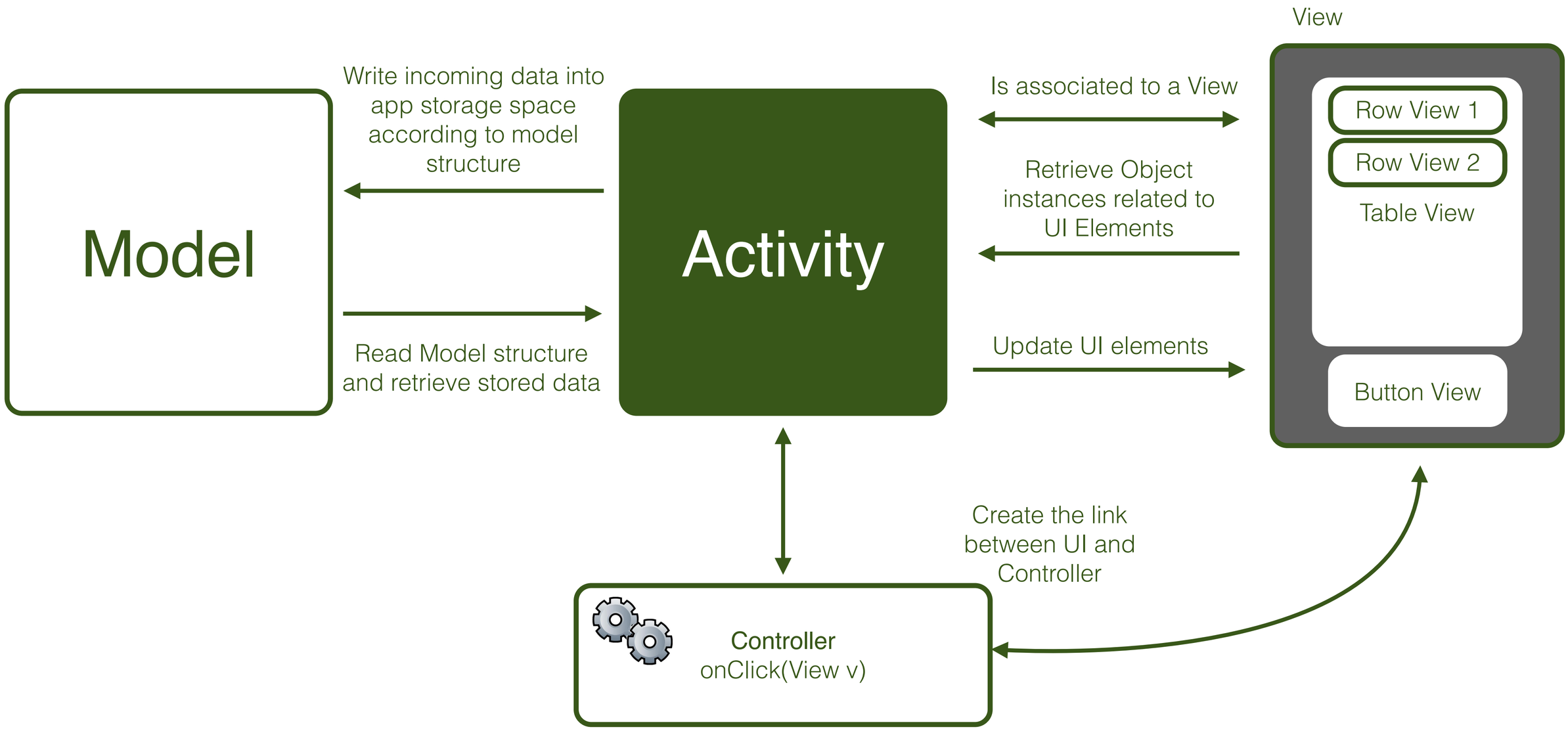
Widgets

- A widget is a View object that serves as an interface for interaction with the user.
- Android provides a set of fully implemented widgets, like buttons, checkboxes, and text-entry fields.
- Some widgets provided by the platform are more complex, like a date picker, a clock, and zoom controls.
- There is also the opportunity to do something more customized creating a user defined actionable elements. The developer can define he/she own View object or can extend and combine existing widgets.
- An example of custom components is user defined Custom Buttons, Progress Bar, Chart View, Map Marker etc ....





Source: Custom Components (http://developer.android.com/guide/topics/ui/custom-components.html) / ChartDroid(http://code.google.com/p/chartdroid/wiki/Screenshots), / achartEngine (http://code.google.com/p/achartengine/)


Model
Write incoming data into app storage space according to model structure
Read Model structure and retrieve stored data
Activity
Is associated to a View
Retrieve Object instances related to UI Elements
Update UI elements
View
Row View 1
Row View 2
Table View
Button View
Controller
onClick(View v)
Create the link between UI and Controller

Activity

- An Activity is both a unit of user interaction (typically associated to a View) and a unit of execution.
- Inside the override method **onCreate(...)** the developer can set the layout View associated to the Activity calling **setContentView(R.layout.<layout_file>)** using the shared R object to specify the resource file containing the layout.

```
<?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:id="@+id/mainTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

</LinearLayout>
```

- When the root view has been set it is possible to retrieve objects associated to the UI elements through the method **findViewById(R.id.<resource_id>).**
- A View object like TextView can be used to updated the user interface, to get values from input fields or to set Event listeners using provided get and set methods.

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

```
 TextView mainTextView =
(TextView)findViewById(R.id.mainTextView);
 mainTextView.setText("Hello World ! :D");
```

Input Events

- Views/widgets and generally UI are used not only to show contents and information but have the important role to allow the user to interact with the application performing actions. To be informed of user input events, the developer needs to do one of two things:
  - **Define an event listener and register it with the View.** The View class contains a collection of nested interfaces named On<something>Listener, each with a callback method called On<something>().
    - For example, View.OnClickListener (for handling "clicks" on a View), View.OnTouchListener (for handling touch screen events in a View), and View.OnKeyListener (for handling device key presses within a View).
    - In order to be notified when the view is "clicked" (such as when a button is selected), implement OnClickListener and define its onClick() callback method (where you perform the action upon click), and register it to the View with setOnClickListener().
  - **Override an existing callback method for the View.** This is what you should do when you've implemented your own View class and want to listen for specific events that occur within it.

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};
protected void onCreate(Bundle savedValues) {
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
}
```

Button Events

```
public class MyActivity extends Activity {
     protected void onCreate(Bundle icicle) {
         super.onCreate(icicle);

         setContentView(R.layout.content_layout_id);

         final Button button = (Button) findViewById(R.id.button_id);
         button.setOnClickListener(new View.OnClickListener() {
             public void onClick(View v) {
                 // Perform action on click
             }
         });
     }
}
```

Launch / Finish Activity

- The **startActivity(Intent)** method is used to start a new activity, which will be placed at the top of the activity stack. It takes a single argument, an Intent, which describes the activity to be executed

```
startActivity(new Intent(getApplicationContext(), MyActivity.class));
```

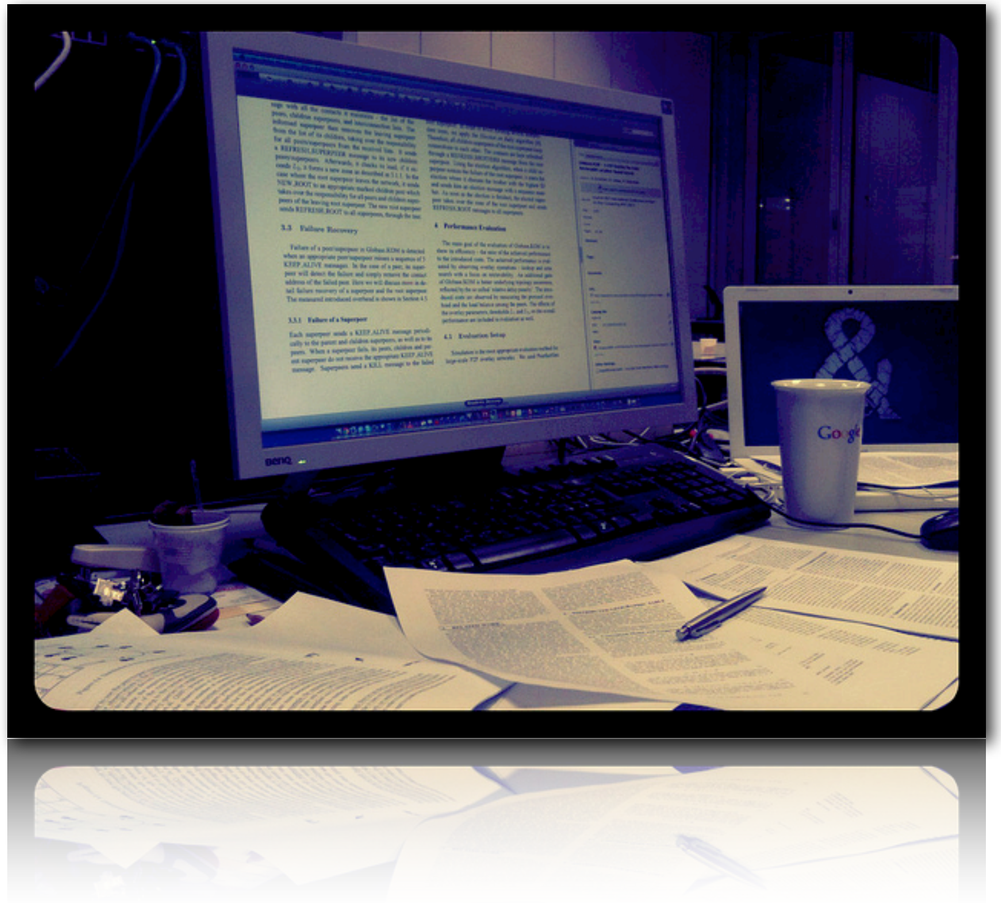
- Sometimes you want to get a result back from an activity when it ends. For example, you may start an activity that lets the user pick a person in a list of contacts; when it ends, it returns the person that was selected. To do this, you call the **startActivityForResult(Intent, int)** version with a second integer parameter identifying the call. The result will come back through your onActivityResult(int, int, Intent) method.

- When an activity exits, it can call setResult(int) to return data back to its parent. It must always supply a result code, which can be the standard results RESULT_CANCELED, RESULT_OK, or any custom values starting at RESULT_FIRST_USER. In addition, it can optionally return back an Intent containing any additional data it wants. All of this information appears back on the parent's Activity.onActivityResult(), along with the integer identifier it originally supplied.

- When the activity is done the developer can call the method **finish()** to close it. The ActivityResult is propagated back to whoever launched you via onActivityResult().

Coming Up

- Next Lecture
  - Android Graphical User Interface - 1
- Homework
  - Review Hello Android Application
  - ....



Android Development

Lecture 2
Android Platform