Java

http://www.android.com/

Android framework overview

Activity lifecycle and states

Major framework terms 1

- All Android apps contains at least one of the 4 components
- Activity (1)
  - Building block of the UI. Every screen in your application will be an extension of the Activity super class
  - You can think of an Activity as being analogous to a window or dialog in a desktop environment
- Service (2)
  - Headless (non-UI) application that runs in the background
  - They are designed to keep running independent of any Activity
- ContentProvider (3)
  - A Content Provider do not store data, but provide the interface for other applications to access the data (in some cases store data)
  - Enable multiple applications to share data with each other
  - Access to the components (Content Provider) data is handled via a Content Resolver as:

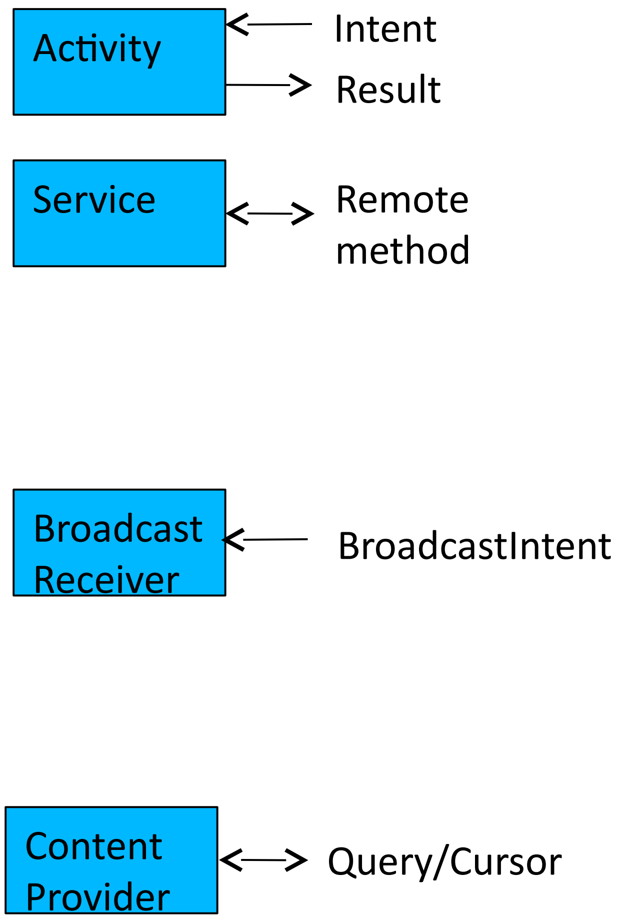
```
Uri allCalls = android.provider.CallLog.Calls.CONTENT_URI;
```

Major framework terms 2

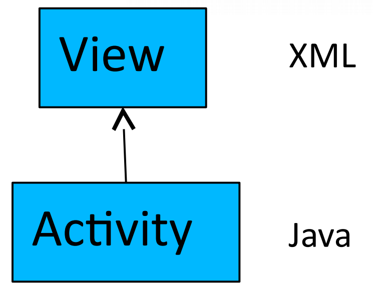
- Intent
  - An asynchrony message passing framework. Using an intent you can broadcast a message **implicit** system-wide or **explicit** to a target Activity, Service or BroadcastReceiver
- BroadcastReceiver (4)
  - By registering a broadcast receiver in the AndroidManifest or in the source code the application can listen and respond to broadcast Intents that match a specific filter criteria
- Alert Dialogs, Notifications and Toasts
  - The user notification framework lets you signal users in the status bar without interrupting their current activity
  - For instance an incoming call can alert you with flashing lights, making sounds, or showing a dialog/toast
- Views, widgets
  - Views and widgets are all the graphical components we use in our layout

Android components

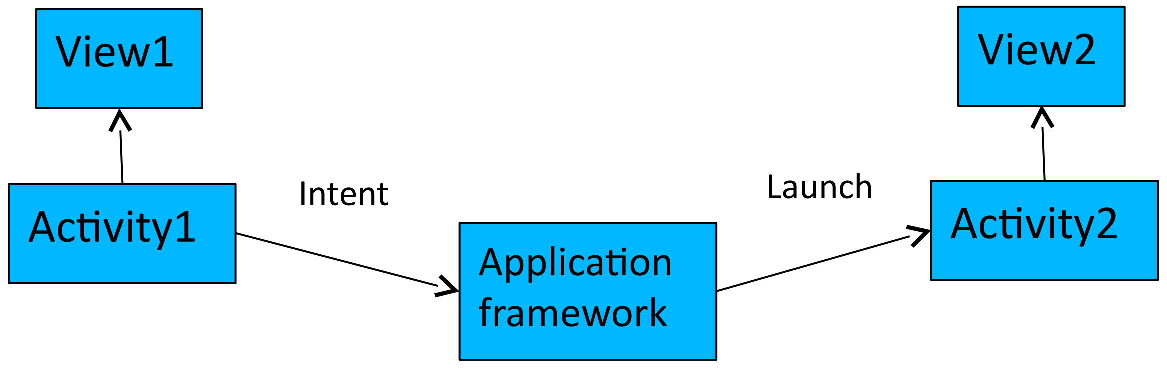
- **Activity** (Presentation layer: what user sees)
- **Service** (runs on background and has no view)
- **Intents** (asynchronous messages that may be followed by actions)
- **Broadcast Receivers** (receive broadcast announcements and react accordingly)
- **Content Providers** (support sharing and accessing of data among applications)



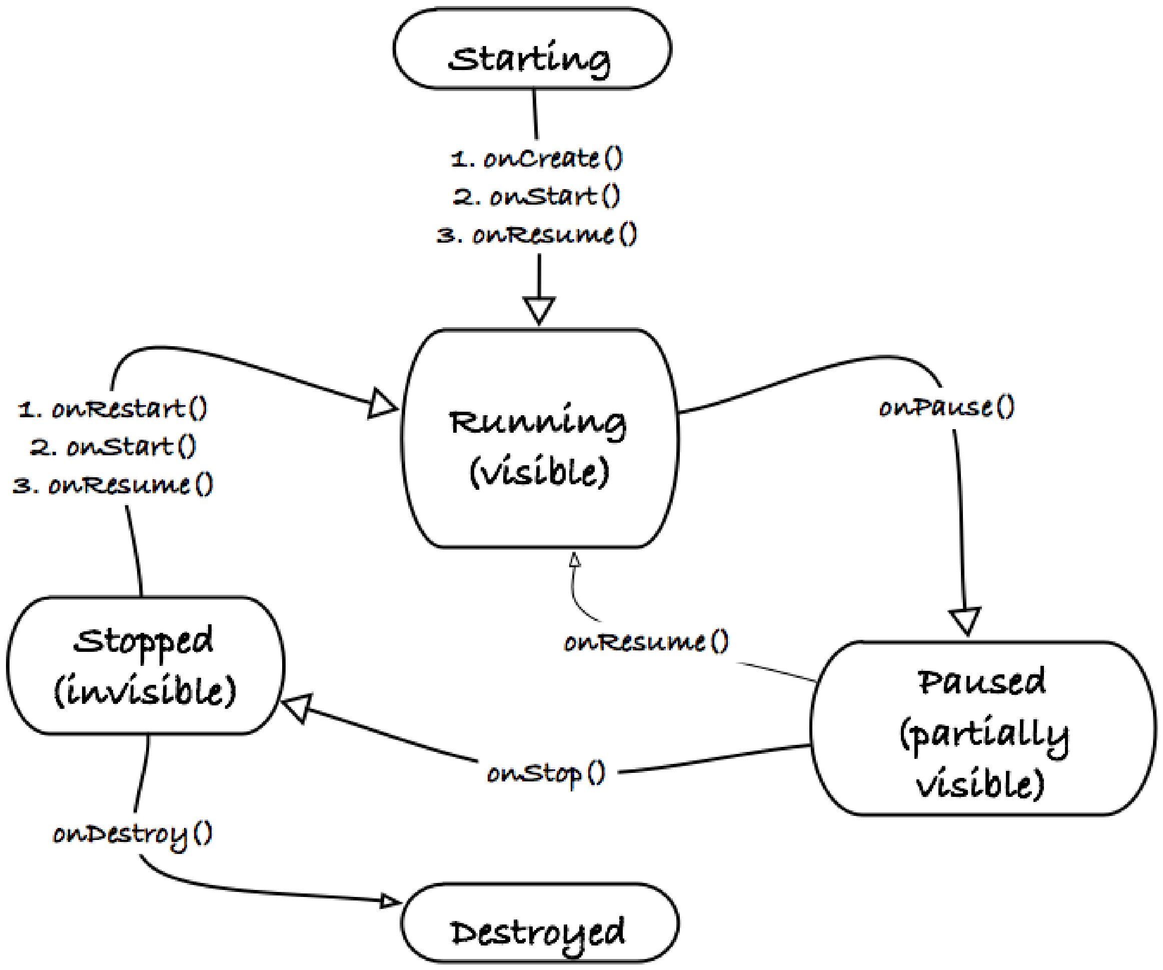
Activity and Intent



- Each activity is a single screen
- It allows user interaction
- The main activity is launched when an app is launched
- Activities launch other activities by sending intents to the application framework



Activity lifecycle



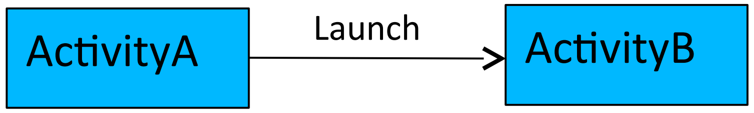
https://thenewcircle.com/s/post/1178/architecting_android_apps

Example of Intent (1)

```
//Activity A
@Override
protected void onCreate(Bundle
savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_layout);
Button b = (Button) findViewById(R.id.button1);
b.setOnClickListener(new OnClickListener() {
   public void onClick(View v) {
      Intent intent = new
Intent(ActivityA.this, ActivityB.class);
      startActivity(intent);
   }
});
}
```

```
//Activity B
@Override
protected void onCreate(Bundle
    savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_layout);
final TextView t =
    (TextView)findViewById(R.id.textView1);
t.setText("This is activity B");
}
```

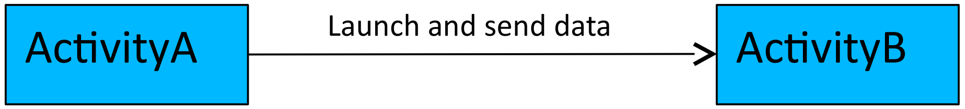
Note: both activities should be declared in *AndroidManifest.xml*



Example of Intent (2)

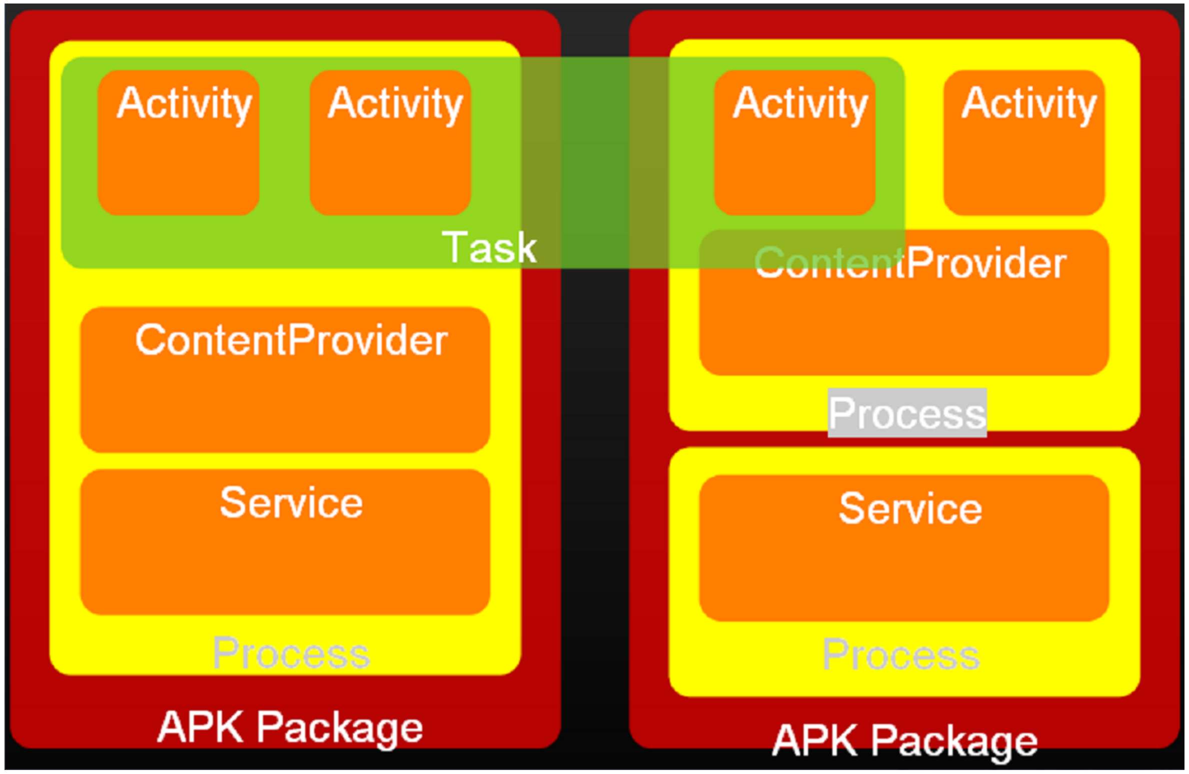
```java
// Activity A
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_layout);
Button b = (Button) findViewById(R.id.button1);
t = (TextView)findViewById(R.id.textView1);
t.setText("This is Activity A");
b.setOnClickListener(new OnClickListener() {
   public void onClick(View v) {
      Intent intent = new Intent(ActivityA.this,
ActivityB.class);
      Bundle b = new Bundle();
      b.putString("greeting", "Hello");
      intent.putExtra("p_greetingBundle", b);
      startActivity(intent);
   }
});
}
```

```java
// Activity B
@Override
protected void onCreate(Bundle
    savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_layout);
Intent intent = getIntent();
Bundle bundle =
    intent.getBundleExtra("p_greetingBundle");
TextView t = (TextView)
    findViewById(R.id.textView1);
t.setText("This is activity B: " +
    bundle.getString("greeting"));
}
```



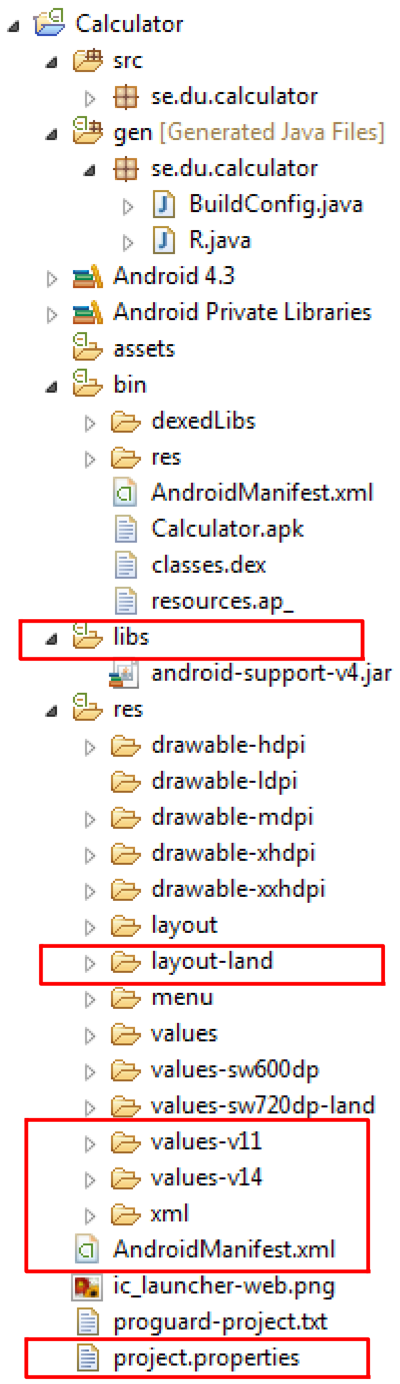
Android application model

- A task is what the user see as an application
  - The borders between executable, process and app icon are blurry
  - Default 1 thread/process (GUI), additional threads are created only if the app itself create them
  - http://developer.android.com/guide/topics/fundamentals.html
  - http://developer.android.com/guide/components/processes-and-threads.html



More about the Android project folders/files

- An Android application is described in the file AndroidManifest.xml
  - This file contains all Intents, Activities, Services or ContentProviders it can handle. Application name, icons and version number plus the Android versions (API) it can run on
  - Required permissions for the application as using the network, access the camera etc. Also other special feature permissions and meta data values as API_KEYs etc.
  - A LOT of other possible configuration settings!
- The project.properties file
  - Project build target API (must be installed in SDK)
  - Enable ProGuard if a release build is done (File > Export...)
  - Usually manage via Project Properties
- libs folder for private libraries/referenced libraries (non SDK libraries)
  - It is possible to use ordinary Java libraries but for robustness it is recommended to use libraries made for Android
- res/xml
  - For miscellaneous XML files that configure application components. For example, an XML file that defines a PreferenceScreen (settings)

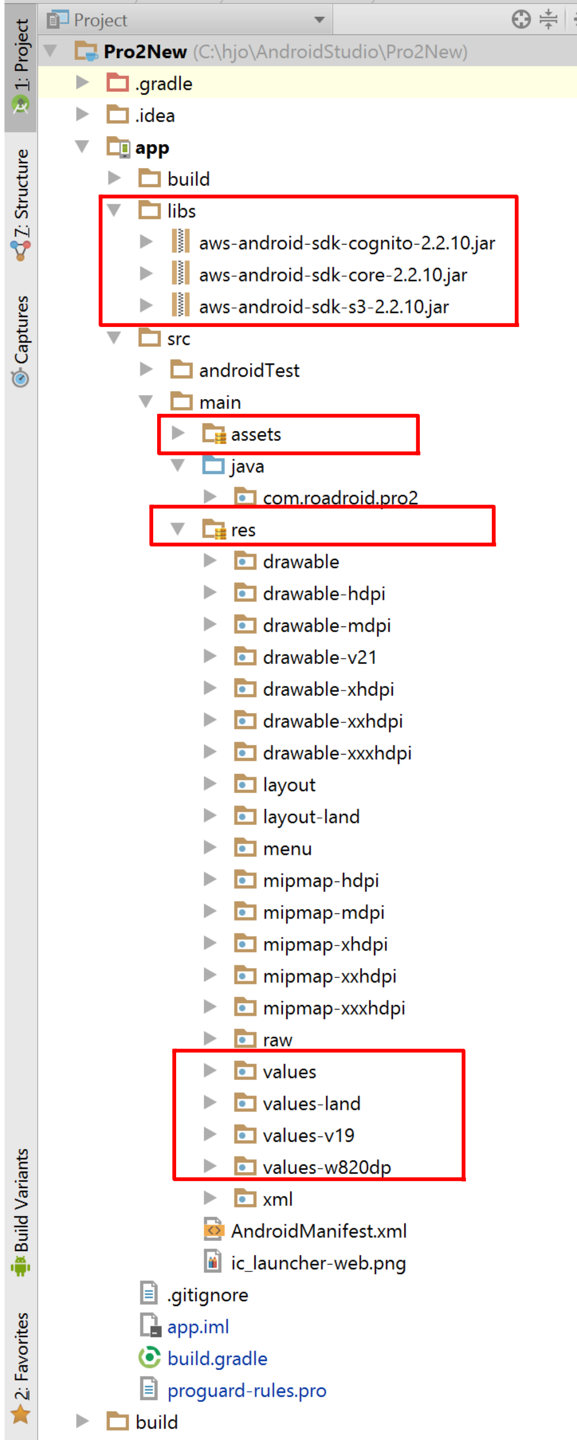


```
Calculator
  src
    se.du.calculator
  gen [Generated Java Files]
    se.du.calculator
      BuildConfig.java
      R.java
  Android 4.3
  Android Private Libraries
  assets
  bin
    dexedLibs
    res
    AndroidManifest.xml
    Calculator.apk
    classes.dex
    resources.ap_
  libs
    android-support-v4.jar
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    drawable-xhdpi
    drawable-xxhdpi
    layout
    layout-land
    menu
    values
    values-sw600dp
    values-sw720dp-land
    values-v11
    values-v14
    xml
  AndroidManifest.xml
  ic_launcher-web.png
  proguard-project.txt
  project.properties
```

More about the AS project folders/files

- Assets
  - Contains ”as is” file resources as images, html docs etc.
  - Using resources here is like open a file on external storage
- res/values-***
  - Contains xml declarations for a multitude of things
  - Arrays, colors, dimens, strings, styles, themes, etc.
  - Values for different screen sizes, APIs, languages etc.
- res/layout and layout-land
  - Different layouts for activities and fragments
- res/menu
  - Contains xml definitions of different menus
  - Overflow menu, app drawer menu etc.
- res/drawable-xxx
  - Contains app icons for different screen densities and APIs
- res/drawable
  - Icons which is not screen density managed
- res/mipmap-xxx
  - Contains launcher icons for different screen densities

https://developer.android.com/studio/intro/index.html



```
Pro2New (C:\hjo\AndroidStudio\Pro2New)
  .gradle
  .idea
  app
    build
    libs
      aws-android-sdk-cognito-2.2.10.jar
      aws-android-sdk-core-2.2.10.jar
      aws-android-sdk-s3-2.2.10.jar
    src
      androidTest
      main
        assets
        java
          com.roadroid.pro2
        res
          drawable
          drawable-hdpi
          drawable-mdpi
          drawable-v21
          drawable-xhdpi
          drawable-xxhdpi
          drawable-xxxhdpi
          layout
          layout-land
          menu
          mipmap-hdpi
          mipmap-mdpi
          mipmap-xhdpi
          mipmap-xxhdpi
          mipmap-xxxhdpi
          raw
          values
          values-land
          values-v19
          values-w820dp
          xml
        AndroidManifest.xml
        ic_launcher-web.png
    .gitignore
    app.iml
    build.gradle
    proguard-rules.pro
  build
```

Simple AndroidManifest.xml

http://developer.android.com/guide/topics/manifest/manifest-intro.html

- The Application node properties include icon and application label in the home screen
- The Activity node name is abbreviated to .HttpDownload, it tells Android which Java class to load. The activity label is the title for that Activity in the titlebar. **To be seen every Activity needs to be specified!**
- Intent-filter tags specify which Intents that launch the Activity. In order for an application to be available from the launcher (app drawer) it must include an intent-filter listening for the MAIN action and the LAUNCHER category

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.android.httpdownload"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".HttpDownload"
                  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>
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Intent and intent-filter

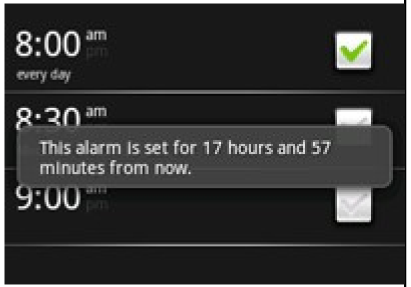
- Allows the application to request and/or provide services i.e. start Activities, Services or Broadcast Receivers
- Intents are system messages that notify applications of various events/actions, transfer various data etc.
  - Activity events (launch app, start activity, pressing widgets, etc.)
  - Hardware state changes (battery status, screen off, etc.)
  - Incoming data (call received, SMS received, etc.)
- Applications are registered via an intent-filter allowing to create loosely coupled applications
- You can create your own to launch applications
  - In the AndroidManifest.xml below the SmsReceiver class which extends BroadCastReceiver can intercept incoming SMS action

```
<receiver android:name=".SmsReceiver">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

BroadcastReceiver

- By registering a broadcast receiver firm in the AndroidManifest or dynamic in the source code, the application can listen and respond to broadcast Intents that match a specific filter criteria
- By calling batteryLevel() a Toast will show the battery level when onReceive() is called by the system
- When onRecive() is done the lifecycle has ended for a broadcast receiver

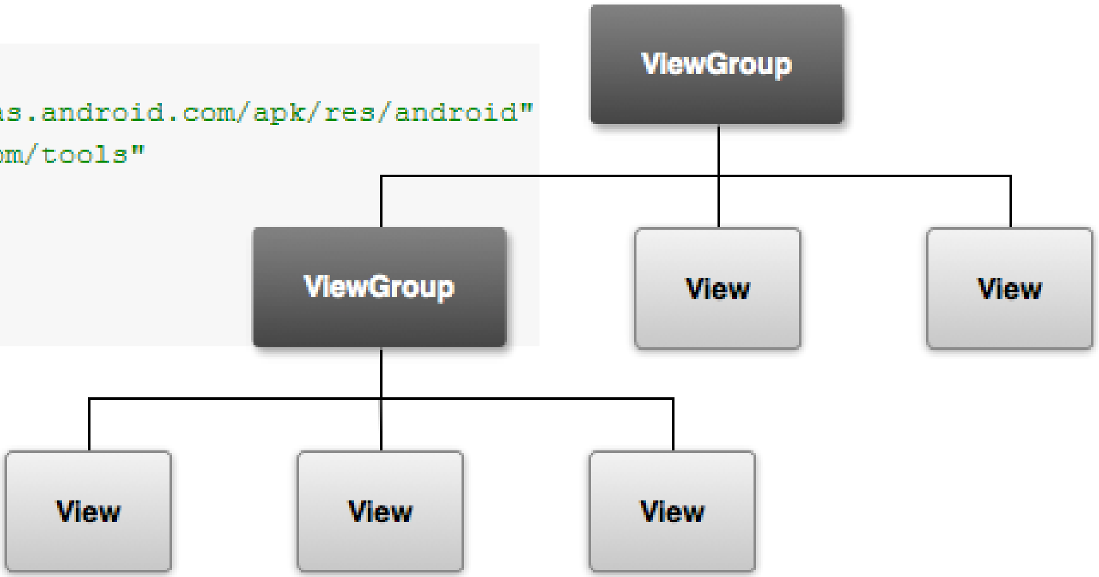
```
private void batteryLevel()
{
    BroadcastReceiver batteryLevelReceiver = new BroadcastReceiver()
    {
        @Override
        public void onReceive(Context context, Intent intent)
        {
             // unregistration of the reciever
            context.unregisterReceiver(this);
            int rawlevel = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
            int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
            int level = -1;
            if (rawlevel >= 0 && scale > 0) {
                level = (rawlevel * 100) / scale;
           }
           Toast.makeText(getApplicationContext(), "Battery Level Remaining: " + level + "%",
                 Toast.LENGTH_SHORT).show();
        }
    };

    // Intent and a dynamic registration of a receiver via registerReciver
    IntentFilter batteryLevelFilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
    registerReceiver(batteryLevelReceiver, batteryLevelFilter);
}
```


8:00 am pm
every day
8:30 am
This alarm is set for 17 hours and 57 minutes from now.
9:00 am pm

GUI for an Android app

- The graphical user interface for an Android app is built using a hierarchy of View and ViewGroup objects
  - View objects are usually UI widgets such as buttons or text fields and ViewGroup objects are invisible view containers that define how the child views are laid out, such as in a grid or a vertical list
  - Android provides an XML vocabulary that corresponds to the subclasses of View and ViewGroup so you can define your UI in XML using a hierarchy of UI elements

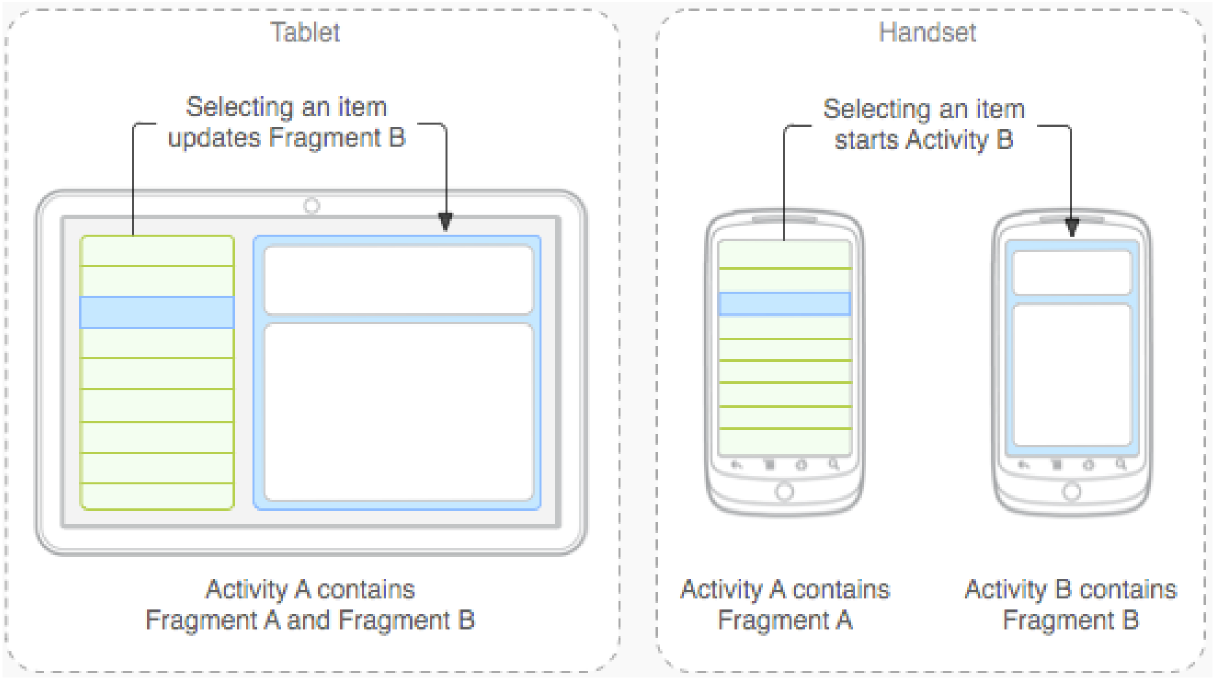
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
</LinearLayout>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



Major framework terms 3

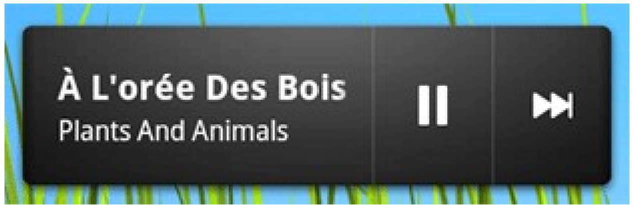
- A **fragment** is an independent component which can be used (embedded) by an activity
  - A fragment encapsulate functionality so that it is easier to reuse within activities and layouts
  - It can be added dynamically (code) or statically (xml) to an activity
- You can think of a fragment as a modular 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).

An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated in 2 activities for a handset design.


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Major framework terms 4

- Loaders
  - Make it easy to asynchronously load data in an activity or fragment
  - They monitor the source of their data and deliver new results when the content changes
- App Widgets
  - App widgets are home screen mini apps that recive periodic updates as current weather or song played
  - The user may interact with the app widget as scroll thru the events in a calendar



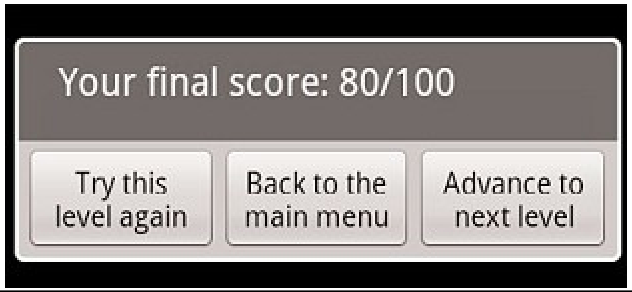
- Adapter
  - An Adapter object acts as a bridge between an AdapterView and the underlying data for that view providing access to the items
  - The Adapter is also responsible for making a View for each item in the data set. ArrayAdapter and SimpleCursorAdapter are common adapters
- Action Bar and Menus (Menu button deprecated from API 11)
  - Enables a consistent way for implementing actions and navigation

Using an Alert Dialog box

Up to three buttons with possible actions

```
private void showAlertDialog()
{
	AlertDialog dialog = new AlertDialog.Builder(this).create();

	dialog.setMessage("Your final score: " + mScore + "/" + PERFECT_SCORE);
	dialog.setButton(DialogInterface.BUTTON_POSITIVE, "Try this level again",
			new DialogInterface.OnClickListener() {
		public void onClick(DialogInterface dialog, int which) {
			mScore = 0;
			start_level();
		}
	});

	dialog.setButton(DialogInterface.BUTTON_NEGATIVE, "Advance to next level",
			new DialogInterface.OnClickListener() {
		public void onClick(DialogInterface dialog, int which) {
			mLevel++;
			start_level();
		}
	});

	dialog.setButton(DialogInterface.BUTTON_NEUTRAL, "Back to the main menu",
			new DialogInterface.OnClickListener() {
		public void onClick(DialogInterface dialog, int which) {
			mLevel = 0;
			finish(); // or dismiss(); to just remove dialog
		}
	});

	dialog.show();
}
```

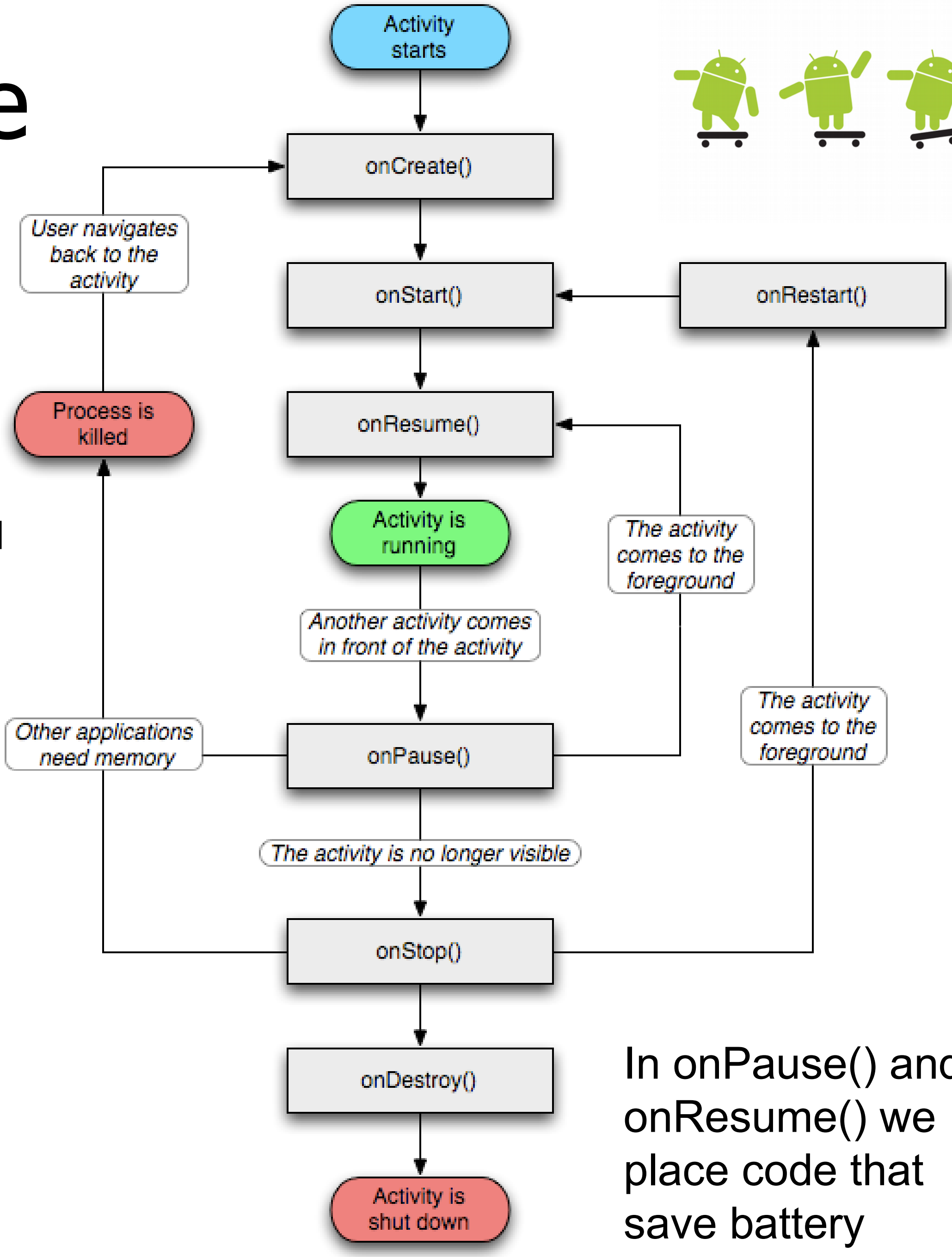
The showDialog / dismissDialog methods in Activity are being deprecated in favor of DialogFragments


Your final score: 80/100

Fragments sample code: http://android-developers.blogspot.in/2012/05/using-dialogfragments.html

Activity Lifecycle

- Activities in the system are managed as an activity stack (”back stack”)
- In onCreate() we init data for the whole lifecycle
- The onStart() and onResume() methods are run when we are placed at top of the stack
- If an activity is in the foreground of the screen (at the top of the stack), it is active or running
- If an activity has lost focus but is still ”visible” it is paused
- If an activity is completely obscured by another activity, it is stopped
- If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

In onPause() and onResume() we place code that save battery

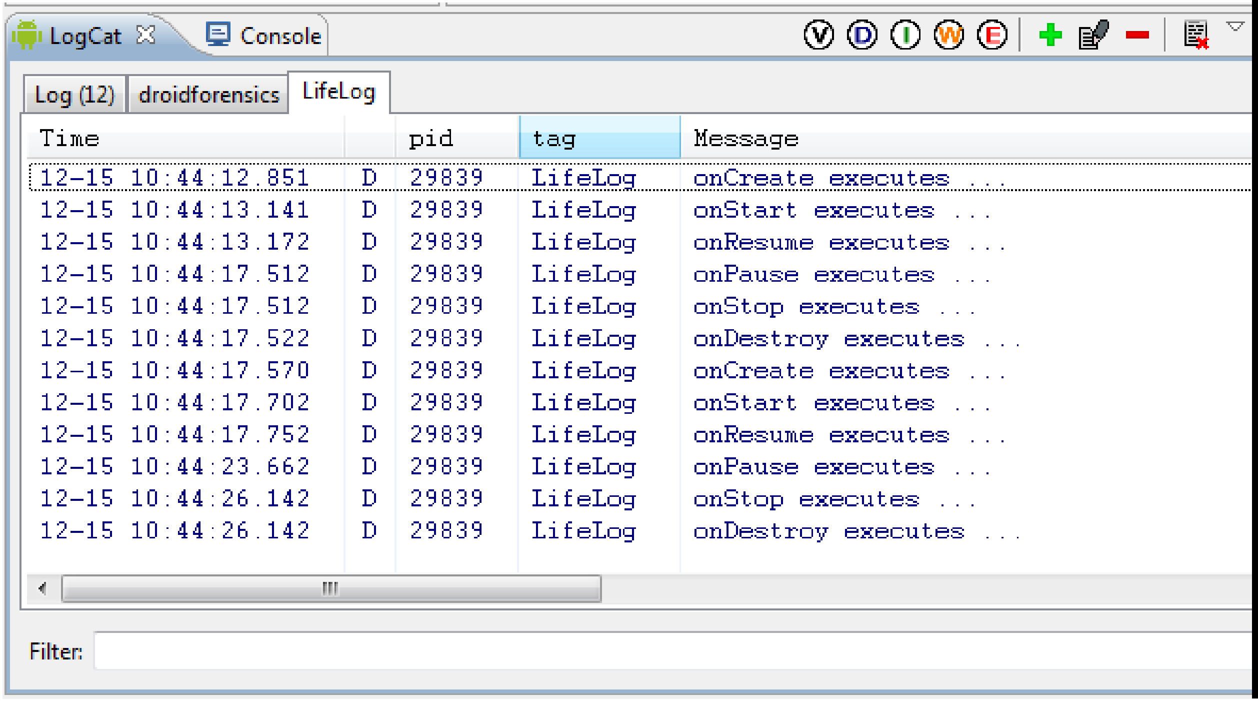
LifecyleTest with implicit Intent and LogCat debug 1

Note! The button handler for openBrowser() is declared in the XML layout file

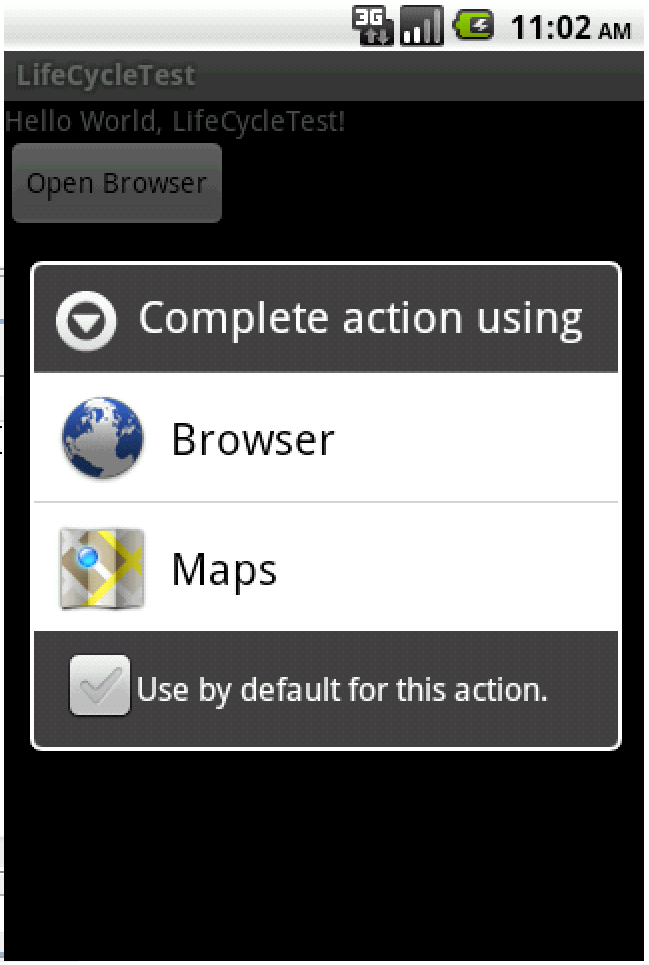
```
package se.du.LifeCycleTest;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;

public class LifeCycleTest extends Activity {

    private static final String DEBUG_TAG = "LifeLog";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(DEBUG_TAG, "onCreate executes ...");
        setContentView(R.layout.main);
    }

    public void openBrowser(View view) {
        Uri uri = Uri.parse("http://maps.google.se/........");
        /* Uri.parse("content://contacts/people/");
        Uri.parse("tel:023778000"); */
        Intent it = new Intent(Intent.ACTION_VIEW, uri);
        startActivity(it);
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d(DEBUG_TAG, "onRestart executes ...");
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.d(DEBUG_TAG, "onStart executes ...");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.d(DEBUG_TAG, "onResume executes ...");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.d(DEBUG_TAG, "onPause executes ...");
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.d(DEBUG_TAG, "onStop executes ...");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(DEBUG_TAG, "onDestroy executes ...");
    }
} // end class LifeCycleTest
```

LifecyleTest with implicit Intent and LogCat debug 2

- Log after start, rotate screen and stop of the LifeLog Activity
- Logfilter with with tag ”LifeLog”
- Pressing the button ”Open Browser” two apps receives the Intent
- Screen capture by DDMS



```
Time                 pid    tag      Message
12-15 10:44:12.851 D 29839 LifeLog  onCreate executes ...
12-15 10:44:13.141 D 29839 LifeLog  onStart executes ...
12-15 10:44:13.172 D 29839 LifeLog  onResume executes ...
12-15 10:44:17.512 D 29839 LifeLog  onPause executes ...
12-15 10:44:17.512 D 29839 LifeLog  onStop executes ...
12-15 10:44:17.522 D 29839 LifeLog  onDestroy executes ...
12-15 10:44:17.570 D 29839 LifeLog  onCreate executes ...
12-15 10:44:17.702 D 29839 LifeLog  onStart executes ...
12-15 10:44:17.752 D 29839 LifeLog  onResume executes ...
12-15 10:44:23.662 D 29839 LifeLog  onPause executes ...
12-15 10:44:26.142 D 29839 LifeLog  onStop executes ...
12-15 10:44:26.142 D 29839 LifeLog  onDestroy executes ...
```



Activities and Lifecyle

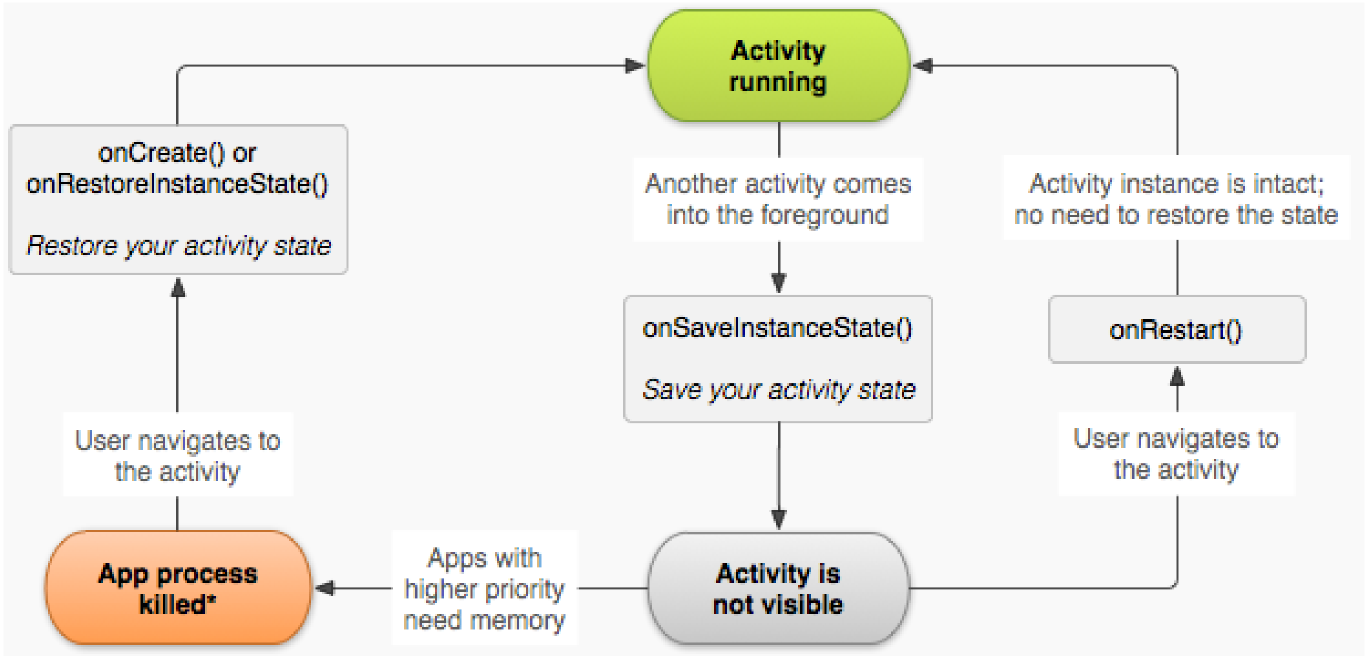
- The operating system controls the life cycle of your application
  - Remember rotating the phone (going from Portrait mode to Landscape mode) will destroy the activity and recreate it from scratch
  - At any time the Android system may pause, stop or destroy your application, e.g. because of an incoming call etc.
    - This will call the proper lifcycle methods
  - In order for the Android system to restore the state of the views in your activity, each view must have a unique ID, supplied by the **android:id** attribute in your layouts
- A Bundle is a structure that can store the applications state - a Bundle (via an intenet) is also used to send data between different Activities
  - The Bundle stores this information in Name – Value pair format (HashMap)
- New Activities can be started with startActivity methods as
  - startActivity(Intent) or startActivityForResult(Intent, int_code)

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(DEBUG_TAG, "onCreate executes ...");
    setContentView(R.layout.main);
}
```

```java
private static final int RESULT_SETTINGS = 1;

Intent otherActivityIntent = new Intent(this, OtherActivity.class);
startActivity(otherActivityIntent);

//otherActivityIntent.putExtra(String name, String value);
//startActivityForResult(otherActivityIntent, RESULT_SETTINGS);
```

Save and restore your Activity state 1

- To save additional data about the activity state, you must override the **onSaveInstanceState()** callback method. The system calls this method when the user is leaving your activity (before the onPause() method) and passes it the Bundle object that will be saved in the event that your activity is destroyed unexpectedly
- If the system must recreate the activity instance later, it passes the same Bundle object to both the **onRestoreInstanceState()** and **onCreate()** methods


Activity running
onCreate() or onRestoreInstanceState()
Restore your activity state
Another activity comes into the foreground
Activity instance is intact; no need to restore the state
onSaveInstanceState()
Save your activity state
onRestart()
User navigates to the activity
User navigates to the activity
App process killed*
Apps with higher priority need memory
Activity is not visible

Save and restore your Activity state 2

```
static final String STATE_NAME = "playerName";
...
@Override
public void onSaveInstanceState(Bundle outState) {
    // Save the user's current game state
    outState.putString(STATE_NAME, mCurrentName);
    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(outState);
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState);
    // Restore state members from saved instance
    mCurrentName = savedInstanceState.getString(STATE_NAME);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first
    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentName = savedInstanceState.getString(STATE_NAME);
    } else {
        // Probably initialize members with default values for a new instance. Or get the intent with
          parameters who was sent to this activity from a parent activity (remember check for null!)
        mCurrentName = getIntent().getData().toString(); // most simple way to get the intent data
    }                                                    // there are many get*** methods available
    ...
}
```

Instead of restoring the state during onCreate() you may choose to implement onRestoreInstanceState(), which the system calls after the onStart() method.

The system calls onRestoreInstanceState() only if there is a saved state to restore, so you do not need to check whether the Bundle is null.

Save and restore your Activity state 3

- See the Flashlight1 test project

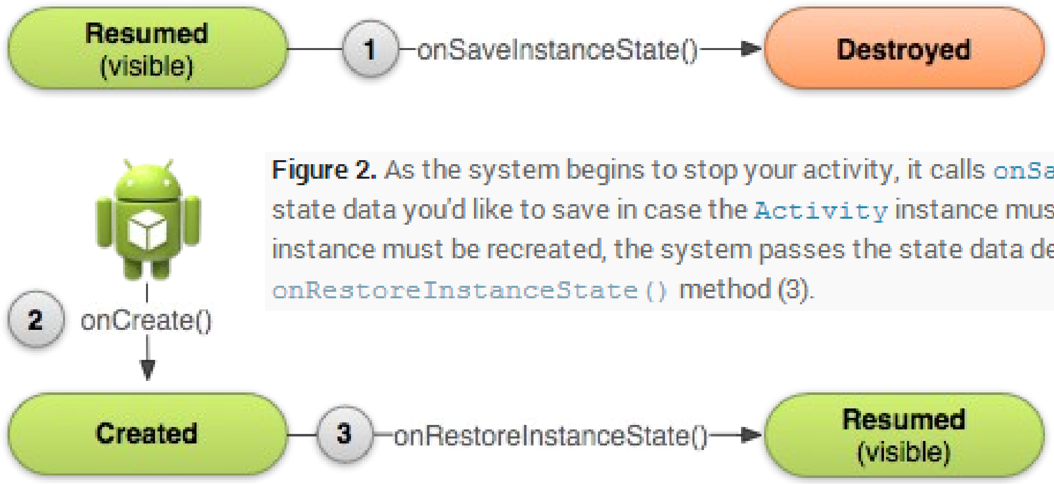


Figure 2. As the system begins to stop your activity, it calls `onSaveInstanceState()` (1) so you can specify additional state data you'd like to save in case the `Activity` instance must be recreated. If the activity is destroyed and the same instance must be recreated, the system passes the state data defined at (1) to both the `onCreate()` method (2) and the `onRestoreInstanceState()` method (3).

```
int red=255; int green= 255; int blue= 255; int alpha = 255;
@Override
protected void onSaveInstanceState(Bundle outState) {
      // saving currently selected color in a bundle
      outState.putInt("redValue", red);  outState.putInt("greenValue", green); outState.putInt("blueValue", blue);
      outState.putInt("alphaValue", alpha);
      super.onSaveInstanceState(outState);
}
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
      super.onRestoreInstanceState(savedInstanceState);
      // coming back to life. setting background to values set before restoring
      red =  savedInstanceState.getInt("redValue"); green = savedInstanceState.getInt("greenValue");
      blue = savedInstanceState.getInt("blueValue"); alpha = savedInstanceState.getInt("alphaValue");
      int myBackColor = android.graphics.Color.argb(alpha, red, green, blue); tvLight.setBackgroundColor(myBackColor);
}
```

Tasks and the back stack

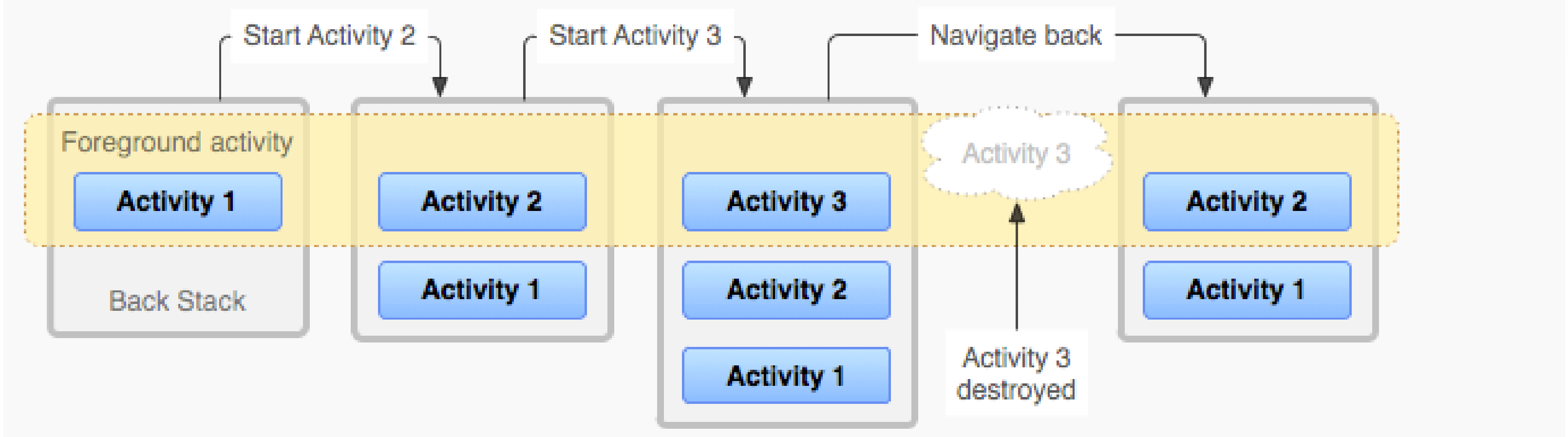


Figure 1. A representation of how each new activity in a task adds an item to the back stack. When the user presses the *Back* button, the current activity is destroyed and the previous activity resumes.

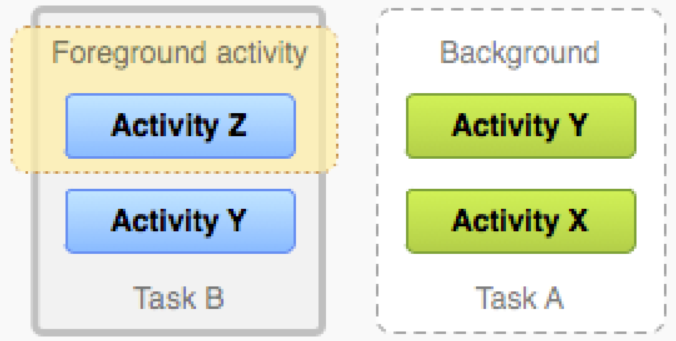


Figure 2. Two tasks: Task B receives user interaction in the foreground, while Task A is in the background, waiting to be resumed.

Check the ActivityLifecycle Demo SingleTask for howto avoiding multiple instances of an Activity

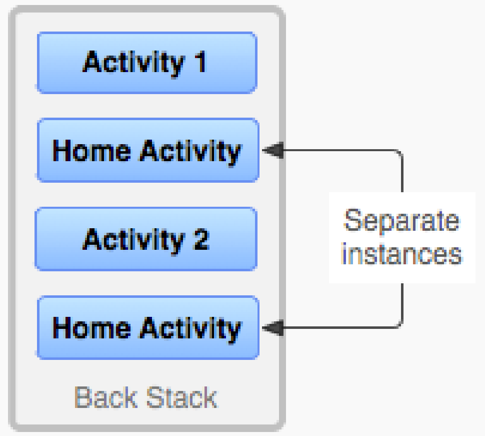


Figure 3. A single activity is instantiated multiple times.

```
In the AndroidManifest Activity tag put android:launchMode="singleTask"
```

Lab review - Android Lab1

- **List with topics you need to understand before next laboration**
- **You must be able to**
  - Use and understand Android Studio and the Android SDK
  - Produce simple (following instructions) Android programs on a basic level
  - Handle the Android emulator
  - Do simple changes to existing Android programs
  - Understand how Android programs works on a basic level
  - View LogCat output from your program with ADM (Android Device Monitor), know how to debug
  - Understand Gradle, AndroidManifest.xml and the other resource xml files on a basic level