Android Programming Basics and User Interfaces

TABLE OF CONTENTS (HIDE)

1. Introduction

Android is an *Operating System for mobile devices* developed by Google, which is built upon Linux kernel. Android competes with Apple's iOS (for iPhone/iPad), RIM's Blackberry, Microsoft's Windows Phone (previously called Windows Mobile), Sambian OS, and many other proprietary mobile OSes.

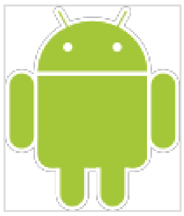
Brief History and Versions

- In 2005, Google acquired a start-up called Android Inc. and started to work on the Android platform, in particular the Dalvik Java Virtual Machine (JVM).
- In 2007, a group of mobile handset producers and computing companies (including Google, Samsung, Motorola, Sony Ericsson, Intel, T-mobile, and Vodaphone) formed the *"Open Handset Alliance"* (OHA) to develop an open and non-proprietary mobile operating platform. The Android beta version was released in November 2007.
- In September 2008, Google released Android SDK 1.0, and placed it under open-source licence. Android 1.x did not support soft keyboard and required physical keypad. Android 1.5 (codenamed Cupcake - Android versions are codenamed after desserts) fixed this problem and including new features such as madia recording and playback, widgets, and added supports to native codes. Android 1.6 (Donut) introduced support for different screen resolutions.
- In October 2009, Google released Android SDK 2.0 (Éclair), which supports HTML 5, flash and multi-touch screens. Android 2.2 (Froyo) added just-in-time (JIT) compilation to the Dalvik virtual machine, which greatly improves the performance. Android 2.3 (Gingerbread) added supports for the OpenGL ES 2.0, and a new concurrent garbage collector to the Dalvik VM.
- In March 2011, Google released Android SDK 3.0 (Honeycomb) - the first *tablet-only* Android platform. Android 3.0 supports wide screen and multi-core processors, and USB peripherals. Android 3.0 does not run on smart phones.
- Android 4.0 (Ice Cream Sandwich) was released in October 2011, which is the result of merging Honeycomb (3.1) and Gingerbread (2.3) to support both smart phones and tablets.
- Android 4.1 (Jelly Bean) was released in June 2012. It introduces the new features such as new Google search experience.

Android platform is identified by two numbers: a version (`x.y.z`) and an API level (a running integer starts from 1). The running integer is used in the Android Market/Google Play to identify new version.

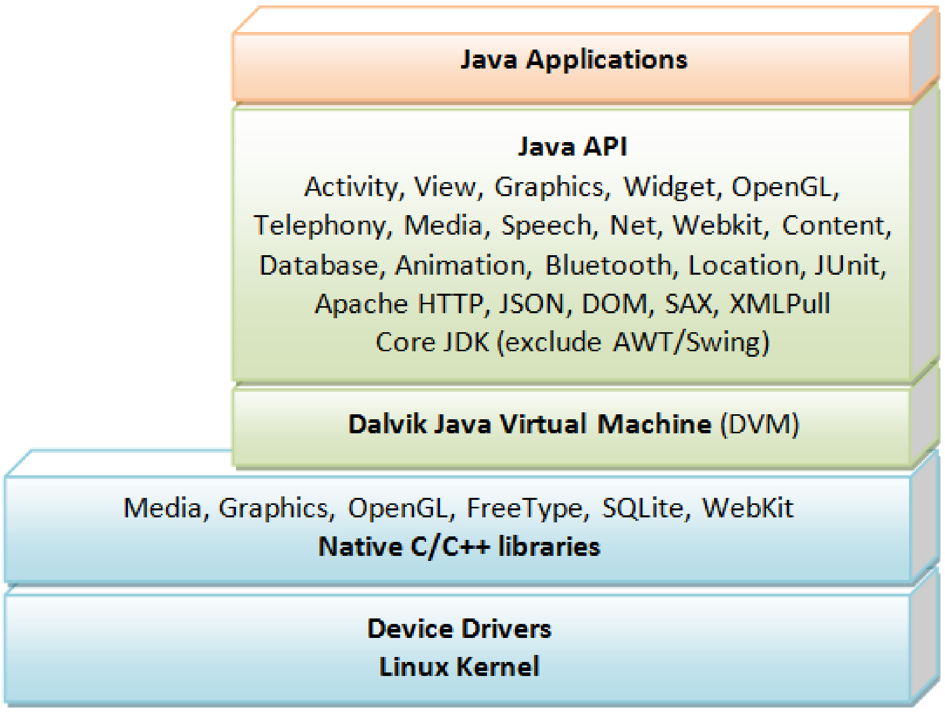
Year	Version	Codename	API Level
2009, Apr 30	1.5	Cupcake	3
2009, Sep 15	1.6	Donut	4
2009, Oct 26	2.0 - 2.1	Eclair	7
2010, Mar 20	2.2	Froyo	8
2010, Dec 6	2.3 - 2.3.2	Ginderbread	9
	2.3.3 - 2.3.7		10
2011, Feb 22	3.0	Honeycomb	11

	3.1		12
	3.2		13
2011, Oct 19	4.0 - 4.0.2	Ice Cream Sandwich	14
	4.0.3 - 4.0.4		15
2012, Jun 28	4.1	Jelly Bean	16



The *mascot* for Android is a little green robot, who is still *nameless*.

2. The Android Platform



Java Applications
Java API
Activity, View, Graphics, Widget, OpenGL, Telephony, Media, Speech, Net, Webkit, Content, Database, Animation, Bluetooth, Location, JUnit, Apache HTTP, JSON, DOM, SAX, XMLPull
Core JDK (exclude AWT/Swing)
Dalvik Java Virtual Machine (DVM)
Media, Graphics, OpenGL, FreeType, SQLite, WebKit
Native C/C++ libraries
Device Drivers
Linux Kernel

Linux Kernel and Device Drivers

The Linux kernel is responsible for the OS functions, such as processor, memory, file, power management. The device drivers includes display, camera, keypad, flash memory, communications (GSM telephony, 3G, WiFi, bluetooth, Near-Field Contact-less communication), accessories (GPS, compass, accelerator), and etc.

Dalvik Virtual Machine

Dalvik VM (DVM) was developed by Google, led by Dan Bornstein, to optimize the performance of Java applications on mobile devices with limited capabilities (Dalvik is the name of a town in Iceland). DVM takes the traditional Java classes ("`.class`") and combines them into one or more Dalvik Executable ("`.dex`") files. By removing duplicate information among the Java classes, it reduces the resultant file size compared with the traditional JAR file. However, as a result, the DVM is not binary-compatible with Java Virtual Machine (JVM). You need to convert the "`.class`" into "`.dex`".

Android 2.2 (Froyo) added a JIT (Just-in-time) compiler, and Android 2.3 (Gingerbread) added a concurrent garbage collector to improve the runtime performance.

Android SDK uses XML file extensively. All the XML files are compiled into binary files to improve the performance.

Native C/C++ Libraries

The native C/C++ libraries include:

- Google's Skia: for 2D graphics support (also used in Google's Chrome).

- Khronos Group's OpenGL ES: for 3D graphics support.
- WebKit: for broswer support (also used in Google's Chrome).
- FreeType: for font support.
- SSL (Secure Socket Layer): for secure communications.
- SQLite: a lightweight relational database system.
- PacketVideo's OpenCore: video/audio record and playback in various format.
- Others.

Android Java API Packages

The API reference is available @ http://developer.android.com/reference/packages.html.

- `android.app`:
- `android.view`, `android.widget`:
- `andrioid.graphics`, `android.animation`:
- `android.media`, `android.speech`, `android.telephony`:
- `android.net`:
- `android.bluetooth`, `android.location`:
- `android.provider`, `android.content`, `android.database`:
- `andriod.gesture`:
- `android.sax`:
- `android.security`:
- `android.test`, `junit.framework`, `junit.runner`:
- `android.text`, `android.util`:
- `java.lang`, `java.io`, `java.nio`, `java.text`, `java.util`
- `java.net`, `javax.net`:
- `java.security`, `java.crypto`:
- `java.sql`, `javax.sql`:
- `javax.xml`, `org.json`, `org,w3c.dom`, `org.xml.sax`, `org.xmlpull`:
- `org.apache.http`:

Android supports a subset of JavaSE (no AWT and Swing). There is no JavaME library in Android.

3. Android Application Framework

Android application framework is radically different from traditional programming frameworks, such as C/C++ and Java. There is no single entry point (you can't find the `main()`?!). This is because the Android apps are targeted at mobile devices with limited capabilities. "Reusing" components from other applications is curial to reduce their footprint.

Android applicatons consist of components, and can communicate and use components of other applications seamlessly (e.g., making a phone call, or taking a photo), without including the codes of the other applications or linking to it. An Android app can simply start other application's component by instantiate Java objects for that component. An Android application does not have a single entry point (there is no `main()` method), but consists of components that can be instantiated and run as the need arises.

There are four main types of application components:

1. Activity: An activity has a *single screen*, which usually composes of one of more views. An activity *interacts* with the user to perform one task.
2. Service: Background processes (similar to Windows' service or Unix's daemon), e.g., playing music.
3. Broadcast Receiver: Receives and reacts to system messages, e.g., low battery life.
4. Content Provider: Android defines a *content provider* mechanism for applications to share data without exposing the underlying implementation. Via content provider, your application can share data with other applications, or use data from another application (e.g., from SQLite Database).

4. Android Programming Basics

4.1 Android Developer Website

The Android developer website @ http://developer.android.com/index.html, is curial for developing Android app. It has three main items: Design, Develop, and Distribute. Begin with the "Develop", which contains Android Training, API Guides, Reference and Tools. In this article, I shall follow closely with "Android Training" and "API Guides", but explain in my perspective.

4.2 Revisit Hello-world

Let us revisit the Hello-world program, reproduced as follow:

```
package com.example.helloandroid;

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

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView textView = new TextView(this); // Construct a TextView UI component
        textView.setText("Hello, world!");      // Set text for TextView
        setContentView(textView);               // This Activity sets content to the TextView
    }
}
```

Activity

An Android application typically comprises several activities. An activity has a *single screen*, which usually composes of one of more views. An activity, as its name implied, *interacts* with the user to do ONE (and only ONE) thing, such as viewing data, creating data, or editing data.

View

Views are UI *components* (or widget, or control) (such as button, label, text field) as well as *containers* (of components), that could be used to build the user interface. In the Hello-world, we use a view called `TextView` to display the message.

Android supports many core JDK packages, except graphics packages AWT and Swing. It provides its own 2D graphics supports, via views and widgets. It supports 3D graphics via OpenGL ES.

Fragment

Fragment was introduced in Android 3.0 to support wider screen, which could be difficult to put all the functions in a single activity. Fragments are like *sub-activities*. An activity can display one or more fragments on the screen at the same time. For a smaller screen, an activity is more likely to contain just one fragment.

AndroidManifest.xml

Each application has a manifest, or *application descriptor*, to describe its contents, resources and behaviors, such as the list of activities, services, intents and permissions. For example, the hello-world's manifest is as follow:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.helloandroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".HelloActivity"
            android:label="@string/title_activity_hello" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- The root element `<manifest>` defines the XML name space, Java package name, version-code and version-name. The version-code is an integer uses by the Android Market/Google Play to keep track of new version, usually starts at 1. The version-name is a string for your own identification. It is interesting and confusing that two numbers are used to identify a version, e.g., Android platform 4.1 has API Level 16.
- The `<use-sdk>` specifies the target and minimum Android SDK API levels.
- The `<application>` element specifies the *icon*, *label* and *theme* for this app. It contains one ore more `<activity>` elements.
- This application has one activity. The `<activity>` element declares its *Java class name* ("`.HelloAndroid`" where `'.'` is relative to the package `com.example.helloandroid`); and *label* (to be displayed as the screen title). It may contain `<intent-filter>` to declare the intent for this activity.
- The `<intent-filter>` declares that this activity is the entry point (`android.intent.action.MAIN`) of the application. This activity is to be added to the application launcher (`android.intent.category.LAUNCHER`).

(The Android Application Descriptor file `AndroidManifest.xml` is similar to Web Application Deployment Descriptor `web.xml` in a JavaEE web application.)

Intent

An intent, as its name implied, declares an *intention* to do something, such as launching an activity, broadcasting a message, starting a service, display a web page, dialing or answering a phone call. An intent could be initiated by your application, or by the system to notify your application about a specific event (such as an incoming phone call or text message). Intents are like "glue" that enable different activities from different applications to work together.

For example, the `<intent-filter>` in the above manifest declares that this activity is the entry point (`android.intent.action.MAIN`) of the application, and it shall be added to be application launcher (`android.intent.category.LAUNCHER`).

5. Activities and their Life Cycle

As mentioned, an *activity* is usually presented in a single screen composing of views (UI components, widget or control), which interacts with the users to perform a single, focused task (e.g., viewing information, editing data, or entering data).

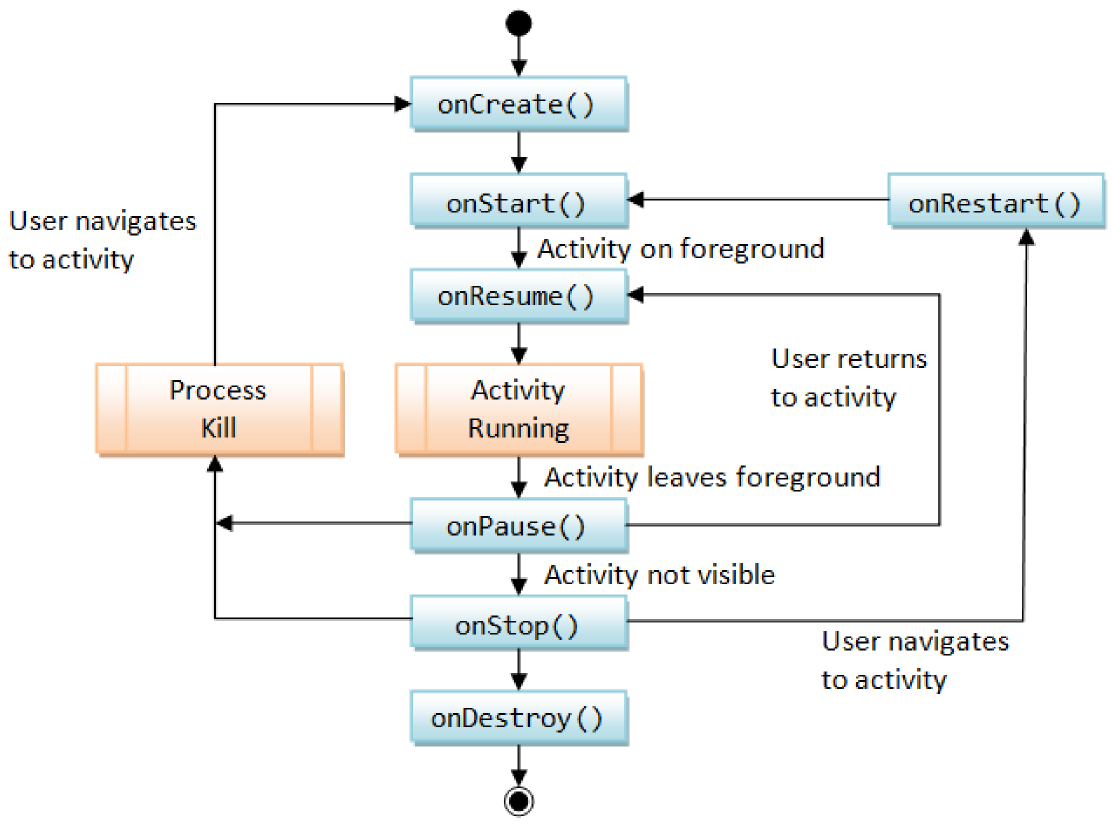
An Android app typically has one or more activities (multiple screens). One of the activities is marked as the startup activity, which in turn starts the next activity, via the *intent*.

To create an activity, we extend the `android.app.Activity` class, and override some of its methods, in particular, `onCreate()`, to provide our own implementation. A typical template of an `Activity` is as follows:

```
package ......;

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

public class ..... extends Activity {
    /** Called back when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ......
        setContentView(......); // Set the content-view of this Activity's screen

        // To use XML Layout, use
        //   setContentView(R.layout.xxx);
        // to map layout specified in res/layout/xxx.xml
    }
}
```

You can set this activity's content-view programatically or via a XML layout file.

The life cycle of an activity is managed via the following *call-back* methods, defined in the `Activity` base class:



- `onCreate()`, `onDestroy()`: Called back when the `Activity` is created/destroyed.
- `onStart()`, `onStop()`, `onRestart()`: Called back when the `Activity` is starting, stopping and re-starting.
- `onPause()`, `onResume()`: Called when the `Activity` is leaving the foreground and back to the foreground, can be used to release resources or initialize states.

Refer to Android API on `Activity` (http://developer.android.com/reference/android/app/Activity.html) for more details.

Example - Activity's Life Cycle

To illustrate the activity's life cycle, create a new Android application named "`Life Cycle`", with project name "`ActivityLifeCycle`" and package name "`com.mytest`". Creat a activity called "`LifeCycleActivity`" with title "`Life Cycle Test`".

Modify the "`LifeCycleActitivity.java`" as follows:

```
package com.mytest;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;
import android.view.Menu;

public class LifeCycleActivity extends Activity {
   private static final String TAG = "LifeCycleActivity: ";

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(TAG, "in onCreate()");
   }

   @Override
   public void onStart() {
      super.onStart();
      Log.d(TAG, "in onStart()");
   }

   @Override
   public void onRestart() {
      super.onRestart();
      Log.d(TAG, "in onReStart()");
   }

   @Override
   public void onResume() {
      super.onResume();
      Log.d(TAG, "in onResume()");
   }

   @Override
   public void onPause() {
      super.onPause();
      Log.d(TAG, "in onPause()");
   }

   @Override
   public void onStop() {
      super.onStop();
      Log.d(TAG, "in onStop()");
   }

   @Override
   public void onDestroy() {
      super.onDestroy();
      Log.d(TAG, "in onDestroy()");
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.activity_main, menu);
      return true;
   }
}
```

"LogCat" & DDMS

In the `LifeCycleActivity`, we use `static` method `Log.d()` to write a "debug" message to the "LogCat" view - this is the standard way of writing log messages in Android, instead of using `System.out.println()`.

LogCat is the Android Logging Utility that allows system and applications to output logging information at various logging levels. Each log entry consists of a time stamp, a logging level (V - verbose, D - debug, I - informational, W - warning , E - error), the process ID (`pid`) from which the log came, a tag defined by the logging application itself, and the actual logging message.

You could use the various static methods in the `java.Android.Log` class to log a message at various logging levels:

```
public static void v(String tag, String message);  // Verbose
public static void d(String tag, String message);  // Debug
public static void i(String tag, String message);  // Informational
public static void w(String tag, String message);  // Warning
public static void e(String tag, String message);  // Error
  // where tag identifies the applicatoin
```

From the LogCat view, you can select a particular logging level, or filter based on pid, tag and log level.

To get finer-grained control and more options, you can switch to the DDMS (Dalvik Debugging Monitor Server) perspective (via "Window" menu ⇒ Open Perspective ⇒ Other ⇒ DDMS). You can get information about all the processes, their VMs and threads, the current state of the heap and class allocation, etc.

Run the App

Run the application (right-click on the project ⇒ Run As ⇒ Android Application), and observe the message on the "LogCat" view.

```
07-14 03:45:48.698: D/LifeCycleActivity:(752): in onCreate()
07-14 03:45:48.698: D/LifeCycleActivity:(752): in onStart()
```

```
07-14 03:45:48.708: D/LifeCycleActivity:(752): in onResume()
```

Click the "BACK" button:

```
07-14 03:46:29.559: D/LifeCycleActivity:(752): in onPause()
07-14 03:46:31.849: D/LifeCycleActivity:(752): in onStop()
07-14 03:46:31.899: D/LifeCycleActivity:(752): in onDestroy()
```

Run the application again from the "App Menu":

```
07-14 03:47:24.749: D/LifeCycleActivity:(752): in onCreate()
07-14 03:47:24.749: D/LifeCycleActivity:(752): in onStart()
07-14 03:47:24.759: D/LifeCycleActivity:(752): in onResume()
```

Click the "HOME" button to send the activity to the background, then run the app from the "App Menu" again. Notice that `onDestroy()` is not called in this case.

```
07-14 03:48:06.369: D/LifeCycleActivity:(752): in onPause()
07-14 03:48:08.258: D/LifeCycleActivity:(752): in onStop()

07-14 03:49:14.650: D/LifeCycleActivity:(752): in onReStart()
07-14 03:49:14.650: D/LifeCycleActivity:(752): in onStart()
07-14 03:49:14.659: D/LifeCycleActivity:(752): in onResume()
```

Click the "PHONE" button, and then "BACK":

```
07-14 03:51:31.829: D/LifeCycleActivity:(752): in onPause()
07-14 03:51:36.700: D/LifeCycleActivity:(752): in onStop()
07-14 03:51:38.308: D/LifeCycleActivity:(752): in onReStart()

07-14 03:51:38.308: D/LifeCycleActivity:(752): in onStart()
07-14 03:51:38.329: D/LifeCycleActivity:(752): in onResume()
```

Take note that clicking the "BACK" button destroys the activitiy, but "HOME" and "PHONE" button does not.

6. User Interface - Views & Layouts

An `Activity` interacts with the user, via a visual UI on a screen. The UI is placed on the `Activity` via the `Activity.setContentView()`. In Android, the UI composes of `View` (UI components, widget or control, such as button, label and text field) and `ViewGroup` (layout) objects, organized in a single view-tree structure.

A `View` is an interactive UI component, such as button and text field. It controls a *rectangular area* on the screen. It is responsible for drawing itself and handling events (such as clicking, entering texts). Android provides many ready-to-use `View`s such as `TextView`, `EditText`, `Button`, `RadioButton`, etc, in package `android.widget`. You can also create your custom `View` by extending `android.view.View`.

A `ViewGroup` is an *invisible container* used to *layout* the `View` components. Android provides many ready-to-use `ViewGroup`s such as `LinearLayout`, `RelativeLayout`, `TableLayout` and `GridLayout` in package `android.widget`. You can also create your custom `ViewGroup` by extending from `android.view.ViewGroup`.

`View`s and `ViewGroup`s are organized in a single tree structure called *view-tree*. You can create a view-tree either using programming codes or describing it in a XML layout file. XML layout is recommended as it separates the presentation view from the controlling logic, which provides modularity and flexibility in your program design. Once a view-tree is constructed, you can add the *root* of the view-tree to the `Activity` as the content view via `Activity.setContentView()` method.

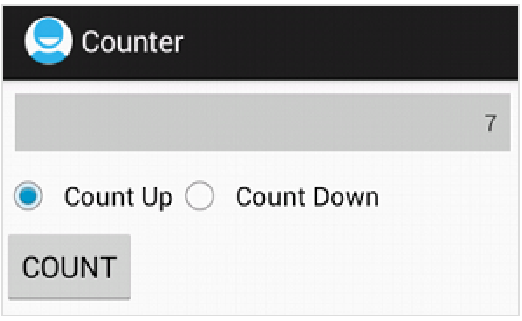
There are two approaches in constructing the UI:

1. Via the XML Layout file: For example, in "Hello-world in XML Layout", we layout all the UI components in an XML file. This approach is more flexible, with the view separated from the business logic, and therefore recommended.
2. Via the programming codes, as in the above Hello-world example. Use this approach only if absolutely necessary.

6.1 Example 1: `Views` (`TextView`, `RadioButton`, `Button`) and `ViewGroup` (`LinearLayout`)

Let's illustrate the view-tree with a simple "Counter" example as illustrated.

The UI consists of a text field (`android.widget.TextView`), two radio buttons (`android.widget.RadioButton`) and a button (`android.widget.Button`), arranged linearly in vertical orientation (`android.widget.LinearLayout`).



First, create an Android project named "`Counter`" with application name of "`Counter`" and package "`com.mytest`". Create an activity "`CounterActivity`" with layout name "`activity_counter`" and title "`Counter`".

Layout `res\layout\activity_counter.xml` and Strings `res\values\strings.xml`

Expand the project node, and replace the "`res\layout\activity_counter.xml`" with the following codes:

```
<LinearLayout xmlns:tools="http://schemas.android.com/tools"
              xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtCountId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/padding_small"
        android:background="#cccccc"
        android:padding="@dimen/padding_medium"
        android:gravity="right" />

    <RadioGroup
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <RadioButton
            android:id="@+id/rBtnUpId"
            android:text="@string/rBtnUpLabel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:checked="true" />
        <RadioButton
            android:id="@+id/rBtnDownId"
            android:text="@string/rBtnDownLabel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </RadioGroup>

    <Button
        android:id="@+id/btnCountId"
        android:text="@string/btnCountLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

This XML layout file describes a view-tree, composing of `View`s (UI components) and `ViewGroup`s (layouts) objects. The root of this view-tree is a `LinearLayout`, which has 3 children: `TextView`, `RadioGroup` and `Button`. `RadioGroup` is also a layout `ViewGroup` which has two children of `RadioButton`.

Each object has its own set of properties (or attributes). The common attributes are:

- `id`: an integer used for referencing the object. Instead of hardcoding an integer, we use a special syntax `@+id/`*`idName`*, with a `'+'` sign to ask the Android SDK to generate a new ID resource. To reference an element, we use syntax `@id/`*`idName`*, without the `'+'` sign.
  For example, in the `Button` declaration, we generate a new `id` called `btnCountId` as follow:

  ```
  <Button android:id="@+id/btnCountId" ...... />
  ```

  Once the `id` is set up, we can use this `id` to reference the object and manipulate the object in our programming codes. For example, in our `CounterActivity`, we can retrieve a reference to the button, via method `findViewById()`, and attach an on-click event handler:

```
// Retrieve a reference to the Button by finding it by its id - in the form of R.id.idname
Button btnCount = (Button)findViewById(R.id.btnCountId);
btnCount.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) { ...... }
});
```

(This approach is very similar to JavaScript programming, which uses `document.getElementById("idname")` to retrieve an element, and process the element or attach an event handler such as `onclick`.)

- `layout_width` and `layout_height`: specifying the rectangular area of the `View` or `ViewGroup`. It is recommended to use a *relative* measurement, such as `MATCH_PARENT` (as big as its parent, called `FILL_PARENT` before API level 8) or `WRAP_CONTENT` (big enough to enclose its content). You can also provide an *absolute* measurement by specifying a floating-point number with a unit, e.g., 12.5px. The available measurement units are px (pixels), dp or dip (density-independent pixels), sp (scaled pixels based on preferred font size), in (inches), and mm (millimeters).
  In our example, the `LinearLayout` fills the screen width and height (`layout_width="match_parent"`, `layout_height="match_parent"`). The `TextView` fill the screen's width (`layout_width="match_parent"`), with a height big enough to enclose its content (`layout_height="wrap_content"`). The `RadioGroup`, `RadioButton` and `Button` have width and height big enough to hold their contents (`layout_width="wrap_content"`, `layout_height="wrap_content"`).
- `text`: specifying an initial text for a text field, or a label for a button. Instead of hardcoding the text in the XML layout file, it is recommended to provide a string reference in the form of `@string/stringName` in the layout file, and place the actual text in `"res\values\strings.xml"`. In this way, you could provide different strings for different locales (languages and countries) for internationalization (i18n) support.
  For this example, we used string references `rBtnUpLabel` and `rBtnDownLabel` (for `RadioButton`) and `btnCountLabel` (for `Button`).
  Next, open "`res\values\strings.xml`" to add the actual strings, as follows:

```
<resources>
    <string name="btnCountLabel">Count</string>
    <string name="rBtnUpLabel"  >Count Up</string>
    <string name="rBtnDownLabel">Count Down</string>
    <string name="app_name">Counter</string>
    <string name="title_activity_counter">Counter</string>
    <string name="menu_settings">settings</string>
</resources>
```

  Take note that the reference `app_name` keeps the application name, `title_activity_counter` keeps the title of the activity, which we configured while creating the Eclipse project.
- `orientation="horizontal|vertical"`: For layouts such as `LinearLayout` and `RadioGroup`, you can specify whether to arrange its children in horizontal or vertical orientation. In our example, the `LinearLayout` uses vertical orientation, while `RadioGroup` uses horizontal.
- others: There are many other attributes. Some attributes are applicable to a certain object only. For example, we use `checked="true"` to set the initial state of a `RadioButton`. For the `TextView`, we use `"background="#cccccc"` to set its background to lightgrey (`#RRGGBB`), `gravity="right"` to right-align the text, `layout_margin="@dimen/padding_small"` to set the margin of the layout element, and `padding="@dimen/padding_medium"` to set the padding between the text and the borders.

Besides using the XML codes, you can also set these attributes graphically, via the "Properties" panel.

`CounterActivity.java`

Let's complete our example by writing the main `Activity` class to process the button-clicks:

```
package com.mytest;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.RadioButton;
```

```
public class CounterActivity extends Activity {

   // References to UI views
   TextView txtCount;
   RadioButton rBtnUp, rBtnDown;
   Button btnCount;

   int count = 0; // counter value

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_counter);

      // Retrieve references to UI views by their id in XML layout
      rBtnUp = (RadioButton) findViewById(R.id.rBtnUpId);
      rBtnDown = (RadioButton) findViewById(R.id.rBtnDownId);
      txtCount = (TextView) findViewById(R.id.txtCountId);
      txtCount.setText(String.valueOf(count)); // Display initial count

      btnCount = (Button) findViewById(R.id.btnCountId);
      // Process the button on-click event
      btnCount.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            if (rBtnUp.isChecked()) { // Counting up
               count++;
            } else if (rBtnDown.isChecked()) { // Counting down
               count--;
            }
            txtCount.setText(String.valueOf(count));
         }
      });
   }
}
```

Dissecting `CounterActivity.java`

- We set the content-view of this application to `R.layout.activity_counter` (`res\layout\activity_counter.xml`).
- We retrieve references to the `View` components `TextView`, 2 `RadioButton`s and `Button`, via `findViewById()` given the respective resource ID.
- For the `Button`, we set the on-click handler via `setOnClickListener()`. We use an inner class implementing `OnClickListener` and override the `onClick()` handler.

NOTES: If you encounter an error on the `@Override onClick()` statement, you can simply remove the `@Override`, or change the Java Compiler level to 1.6 (Right-click on the project ⇒ Properties ⇒ Java Compiler ⇒ set the compiler compliance level to 1.6 (1.7 is not yet supported in Android at the time of writing)). This is because the `@Override` annotation, which was introduced in JDK 1.5 for classes, is applicable for interface in JDK 1.6.

Run the Application

Now, you can run the application.

Generated `AndroidManifest.xml`

Each Android application has a *manifest* file (or *application descriptor*) named `AndroidManifest.xml`, in the project's root directory. It describes the application components. The manifest is generated automatically by the Eclipse ADT:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mytest"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <application
```

```
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".CounterActivity"
            android:label="@string/title_activity_counter" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

In our example,

- The `<manifest>` element specifies the Java package name and program version. It contains one `<application>` element.
- The `<application>` element specifies the icon and label used in mobile device's "apps" menu. It contains one ore more `<activity>` elements.
- This application has one activity named `CounterActivity`. The `<activity>` element declares its program name (`.CounterActivity` where `'.'` is relative to the package `com.mytest`) and label (string reference `title_activity_counter`, where actual text in `res\values\strings.xml`). It may contain `<intent-filter>`.
- The `<intent-filter>` declares that this activity is the entry point (`intent.action.MAIN`) of the application and is included in the app launcher (`intent.category.LAUNCHER`).

Generated "`gen\com.test.R.java`"

```
package com.mytest;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
        public static final int btnCountId=0x7f080003;
        public static final int menu_settings=0x7f080004;
        public static final int rBtnDownId=0x7f080002;
        public static final int rBtnUpId=0x7f080001;
        public static final int txtCountId=0x7f080000;
    }
    public static final class layout {
        public static final int activity_counter=0x7f030000;
    }
    public static final class menu {
        public static final int activity_counter=0x7f070000;
    }
    public static final class string {
        public static final int app_name=0x7f050000;
        public static final int btnCountLabel=0x7f050001;
        public static final int menu_settings=0x7f050005;
        public static final int rBtnDownLabel=0x7f050003;
        public static final int rBtnUpLabel=0x7f050002;
        public static final int title_activity_counter=0x7f050004;
    }
    public static final class style {
        public static final int AppTheme=0x7f060000;
    }
}
```

Eclipse ADT automatically generates a `R.java` ("`R`" for resources) to keep track and instantiate all the resources (icons, string, view and layout) used in the application. The resource IDs are kept in inner classes `R.id`, `R.drawable`, `R.layout`, `R.string`, which maps to the respective resource folders. For example, `R.layout.activity_counter` maps to `res\layout\activity_counter.xml`; `R.string` maps to `res\values\strings.xml`.

Specifying the onClick Handler in the Layout XML File

Instead of using `Button`'s `setOnClickListener()` to register a listener object, we could also specify the `onClick` handler directly in the layout XML file. For example,

```
<Button
   android:id="@+id/btnCountId"
   android:text="@string/btnCountLabel"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:onClick="btnCountOnClickHandler" />
```

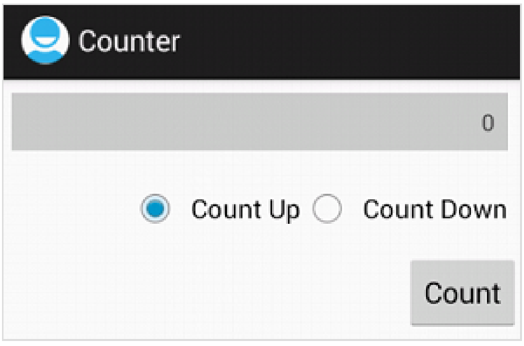
In the activity, you need to program the event handler, which has the same signaure as the `onClick()` method, as follows:

```
public void btnCountOnClickHandler(View v) {  // same signature as onClick()
   if (rBtnUp.isChecked()) { // Counting up
      count++;
   } else if (rBtnDown.isChecked()) { // Counting down
      count--;
   }
   txtCount.setText(String.valueOf(count));
}
```

Try it out! (I decided to show the Java listener code for the programmers, but you should use the `onClick` handler, if feasible, as it is much simpler.)

6.2 Example 2: Relative Layout

Let's modify our program to use `RelativeLayout`, instead of `LinearLayout`. In relative layout, the UI are arranged relative to each others.



Layout `res\layout\activity_counter_relative.xml`

We shall use the same project (as the previous example), but specifying a new layout file.

Create a XML layout file called `activity_counter_relative.xml` in `res\layout` folder (right-click on the `res\layout` node ⇒ new ⇒ file) as follows:

```
<RelativeLayout xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/txtCountId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/padding_small"
        android:background="#cccccc"
        android:padding="@dimen/padding_medium"
        android:gravity="right" />

    <RadioGroup
        android:id="@+id/radioGroupId"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/padding_small"
        android:orientation="horizontal"
        android:layout_below="@id/txtCountId"
        android:layout_alignParentRight="true">
        <RadioButton
            android:id="@+id/rBtnUpId"
            android:text="@string/rBtnUpLabel"
```

```
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:checked="true" />
          <RadioButton
              android:id="@+id/rBtnDownId"
              android:text="@string/rBtnDownLabel"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content" />
      </RadioGroup>

      <Button
          android:id="@+id/btnCountId"
          android:text="@string/btnCountLabel"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_below="@id/radioGroupId"
          android:layout_alignParentRight="true" />
</RelativeLayout>
```

We use the `RelativeLayout` as the root element. In relative layout, we need to assign an `id` to elements that would be used for reference. As mentioned, to create an `id`, we use a special syntax `@+id/`*`idName`*, with a `'+'` sign to ask the Eclipse ADT to generate an integer `id`. To reference an element, we use the syntax `@id/`*`idName`*.

In relative layout, we could use `layout_below`, `layout_toLeftOf`, `layout_toRightOf`, `layout_alignParentRight`, `layout_alignTop`, etc, (defined in `android.widget.RelativeLayout.LayoutParams`) to arrange a UI view relative to another view.

In this example, the `RadioGroup` is placed below the `EditText` and right-aligned with its parent `RelativeLayout`. The `Button` is placed below the `RadioGroup`, and also right-aligned with its parent `RelativeLayout`.

CounterActivity.java

To use the new XML layout file, simply change the `Activity.setContentView()` as follows:

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.counter_activity_relative);  // Use res\layout\counter_activity_relative.xml
   ......
}
```

6.3 More Layouts

Study the various UI layouts @ http://developer.android.com/guide/topics/ui/declaring-layout.html:

- `LinearLayout`: A `ViewGroup` that displays its `View` children in linear direction, either horizontally or vertically.
- `RelativeLayout`: A `ViewGroup` that display its `View` children in relative position to each others or to the parent. e.g., below, align (top, left, right, bottom), margin (top, left, right, bottom).
- `ListView`: A scrollable list of items.
- `GridView`: A scrollable grid of rows and columns.

6.4 Example 3: Simple Calculator

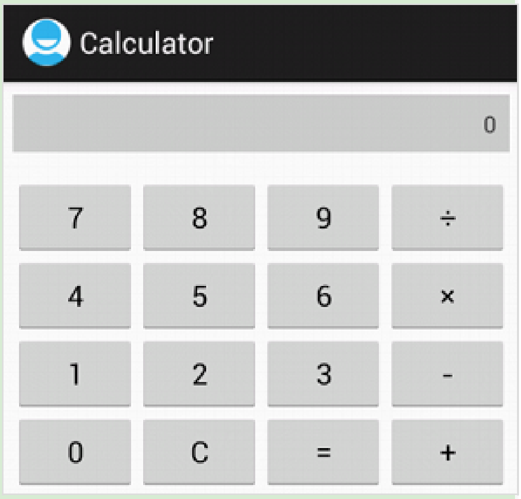
Create a project called `SimpleCalculator`, with application name of "`Simple Calculator`", package "`com.mytest`", main activity "`CalculatorActivity`" with layout name "`calculator_activity`" and title of "`Calculator`".

Layout res\layout\calculator_activity.xml

The UI consists of an `TextView` for displaying the result and 16 `Button`s. We use `TableLayout` to layout the buttons in rows and columns as illustrated.

```
<RelativeLayout 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" >

    <TextView
```

```
        android:id="@+id/txtResultId"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/padding_small"
        android:background="#cccccc"
        android:padding="@dimen/padding_medium"
        android:gravity="right" />

    <TableLayout
        android:id="@+id/tableId"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/padding_small"
        android:layout_below="@id/txtResultId" >
        <TableRow>
            <Button
                android:id="@+id/btnNum7Id"
                android:text="7"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnNum8Id"
                android:text="8"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnNum9Id"
                android:text="9"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnDivId"
                android:text=""
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
        </TableRow>
        <TableRow>
            <Button
                android:id="@+id/btnNum4Id"
                android:text="4"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnNum5Id"
                android:text="5"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnNum6Id"
                android:text="6"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnMulId"
                android:text=""
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
        </TableRow>
        <TableRow>
            <Button
                android:id="@+id/btnNum1Id"
                android:text="1"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
```



```
            <Button
                android:id="@+id/btnNum2Id"
                android:text="2"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnNum3Id"
                android:text="3"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnSubId"
                android:text="-"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
        </TableRow>
        <TableRow>
            <Button
                android:id="@+id/btnNum0Id"
                android:text="0"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnClearId"
                android:text="C"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnEqualId"
                android:text="="
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
            <Button
                android:id="@+id/btnAddId"
                android:text="+"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1" />
        </TableRow>
    </TableLayout>
</RelativeLayout>
```

CalculatorActivity.java

```
package com.mytest;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.TextView;
import android.widget.Button;

public class CalculatorActivity extends Activity {

   private TextView txtResult; // Reference to EditText of result
   private int result = 0;     // Result of computation
   private String inStr = "0"; // Current input string
   // Previous operator: '+', '-', '*', '/', '=' or ' ' (no operator)
   private char lastOperator = ' ';

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_calculator);
```

```java
      // Retrieve a reference to the EditText field for displaying the result.
      txtResult = (TextView) findViewById(R.id.txtResultId);
      txtResult.setText("0");

      // Register listener (this class) for all the buttons
      BtnListener listener = new BtnListener();
      ((Button) findViewById(R.id.btnNum0Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum1Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum2Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum3Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum4Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum5Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum6Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum7Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum8Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnNum9Id)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnAddId)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnSubId)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnMulId)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnDivId)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnClearId)).setOnClickListener(listener);
      ((Button) findViewById(R.id.btnEqualId)).setOnClickListener(listener);
   }

   private class BtnListener implements OnClickListener {
      // On-click event handler for all the buttons
      @Override
      public void onClick(View view) {
         switch (view.getId()) {
         // Number buttons: '0' to '9'
         case R.id.btnNum0Id:
         case R.id.btnNum1Id:
         case R.id.btnNum2Id:
         case R.id.btnNum3Id:
         case R.id.btnNum4Id:
         case R.id.btnNum5Id:
         case R.id.btnNum6Id:
         case R.id.btnNum7Id:
         case R.id.btnNum8Id:
         case R.id.btnNum9Id:
            String inDigit = ((Button) view).getText().toString();
            if (inStr.equals("0")) {
               inStr = inDigit; // no leading zero
            } else {
               inStr += inDigit; // accumulate input digit
            }
            txtResult.setText(inStr);
            // Clear buffer if last operator is '='
            if (lastOperator == '=') {
               result = 0;
               lastOperator = ' ';
            }
            break;

         // Operator buttons: '+', '-', '*', '/' and '='
         case R.id.btnAddId:
            compute();
            lastOperator = '+';
            break;
         case R.id.btnSubId:
            compute();
            lastOperator = '-';
            break;
         case R.id.btnMulId:
            compute();
            lastOperator = '*';
            break;
         case R.id.btnDivId:
            compute();
            lastOperator = '/';
            break;
         case R.id.btnEqualId:
            compute();
```

```
                  lastOperator = '=';
                  break;

               // Clear button
               case R.id.btnClearId:
                  result = 0;
                  inStr = "0";
                  lastOperator = ' ';
                  txtResult.setText("0");
                  break;
               }
            }

            // User pushes '+', '-', '*', '/' or '=' button.
            // Perform computation on the previous result and the current input number,
            // based on the previous operator.
            private void compute() {
               int inNum = Integer.parseInt(inStr);
               inStr = "0";
               if (lastOperator == ' ') {
                  result = inNum;
               } else if (lastOperator == '+') {
                  result += inNum;
               } else if (lastOperator == '-') {
                  result -= inNum;
               } else if (lastOperator == '*') {
                  result *= inNum;
               } else if (lastOperator == '/') {
                  result /= inNum;
               } else if (lastOperator == '=') {
                  // Keep the result for the next operation
               }
               txtResult.setText(String.valueOf(result));
            }
         }
      }
```

Dissecting `CalculatorActivity.java`

We define an inner class called `BtnListener` which implements the `OnClickListener`. We then create a `BtnListener` object and register the same object to all the 16 `Button`s. In the on-click event handler, we inspect the `View` object's `id` to dispatch the appropriate action.

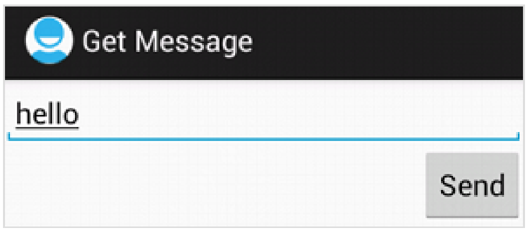
[TODO more]

Specifying the onClick handler in the Layout XML file

The program could be simpler, if we specify the `Button`'s `onClick` handler in the Layout XML file (I decided to show the Java listener code for the programmers):

```
<Button
   ......
   android:onClick="onClick" />
```

6.5 Example 4: Using an Intent to Start a Second Activity



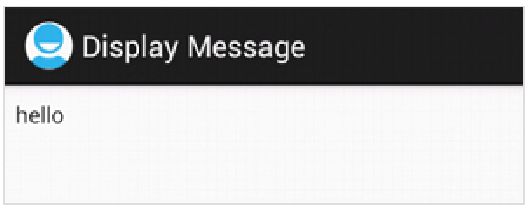
In this example, we shall illustrate how to write an app with two activities (two screens), and use an `Intent` to start the second activity from the first one. As mentioned, an `Intent`, as its name implies, is an intention to do something - in most cases, start another activity. `Intents` are like "glue" that enables different activities from different application to work together.

The first activity prompts user to enter a message (into a `EditText`) and launch the second activity to display the message, when "Send" `Button` is pressed.

Create a new Project wtih the First Activity "`GetMessageActivity.java`"

Create a new Android project called "`TestIntent`", with application name "`Test Intent`" and package name "`com.mytest`".

Create the activity named "GetMessageActivity" with layout name "activity_get_message" and title "Get Message".



hello

Layout of the First Activity - "res\layout\activity_get_message.xml"

A "RelativeLayout" is used with two views: an EditText (a text field for user to enter text) and a Button. The EditText has a "hint" string, which will be shown when the field is empty. Button has a "onClick" hander called sendMessage, to be written in the activity.

```
<RelativeLayout 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" >

    <EditText
        android:id="@+id/txtMessageID"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="@dimen/padding_medium"
        android:hint="@string/message" />

    <Button
        android:id="@+id/btnSend"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/txtMessageID"
        android:layout_alignParentRight="true"
        android:text="@string/btnSendLabel"
        android:onClick="sendMessage" />
</RelativeLayout>
```

Strings "res\values\string.xml"

Include the following string references, which was used in the above layout:

```
<string name="btnSendLabel">Send</string>
<string name="message">Enter your text here</string>
```

First Activity - "GetMessageActivity.java"

```
package com.mytest;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class GetMessageActivity extends Activity {
   public final static String EXTRA_MESSAGE = "com.mytest.MESSAGE";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_get_message);
    }

    /** Called when the user selects the Send button */
    public void sendMessage(View view) {
        // Create an Intent to start the second activity
        Intent intent = new Intent(this, DisplayMessageActivity.class);
        // Retrieve the message entered and put into the Intent
        EditText txtMessage = (EditText) findViewById(R.id.txtMessageID);
        String message = txtMessage.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);  // key-value pair
        // Start the intended activity
        startActivity(intent);
    }
}
```

This activity includes the "Send" `Button`'s `onClick` handler `sendMessage()`. It creates an `Intent` object for launching the second activity called "`DisplayMessageActivity`". It then retrieves the message entered into the `EditText`, puts the message into the `Intent` object, with a key "`EXTRA_MESSAGE`", and starts the intended activity.

The Second Activity

We need to call our second activity "`DisplayMessageActivity`", as declared in the `Intent`. We shall use the layout "`activity_display_message.xml`".

Layout of the Second Activity - "`res\layout\activity_display_message`"

Create the following XML layout file under the "`res\layout`" folder (right-click on the folder ⇒ New ⇒ File). This layout contain one view - a `TextView` to display the message received from the first activity.

```
<RelativeLayout 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" >

    <TextView
        android:id="@+id/txtDisplayID"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="@dimen/padding_medium" />

</RelativeLayout>
```

`DisplayMessageActivity.java`

Create the Java class for the second activity called "`DisplayMessageActivity.java`" under the same package "`com.mytest`" (right-click on the package ⇒ New ⇒ Class).

```
package com.mytest;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class DisplayMessageActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_display_message);

        // Get the message entered in the first activity from the intent
        Intent intent = getIntent();
        String message = intent.getStringExtra(GetMessageActivity.EXTRA_MESSAGE);  // get value via key

        // Get the display TextView for this activity and display the message
        TextView txtDisplay = (TextView) findViewById(R.id.txtDisplayID);
        txtDisplay.setText(message);
    }
}
```

This activity gets the `Intent` object that started it via `getInent()`, and retrieves the message put into the `Intent` via its key. It then puts the message on the `TextView`.

Take note that the content-view is set to "`R.layout.activity_display_message`" which corresponds to the "`res\layout\activity_display_message.xml`" we have created earlier.

`AndroidManifest.xml`

Finally, we need to declare the second activity in the "`AndroidManifest.xml`" by adding the following `<application>` element:

```
<activity
   android:name=".DisplayMessageActivity"
   android:label="@string/title_activity_display_message" >
```

```
</activity>
```

The name of the activity is "DisplayMessageActivity" (relative to the current package "."). There is no need for the <intent-filter> as this activity is launched from the first activity.

We also define its label uses a string reference. Hence, we need to include this string reference in "res\values\string.xml":

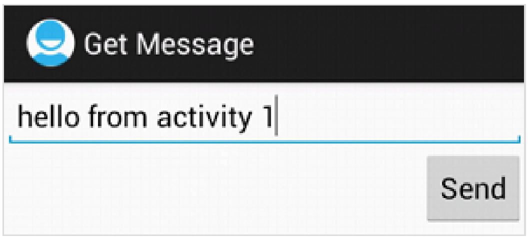
```
<string name="title_activity_display_message">Display Message</string>
```

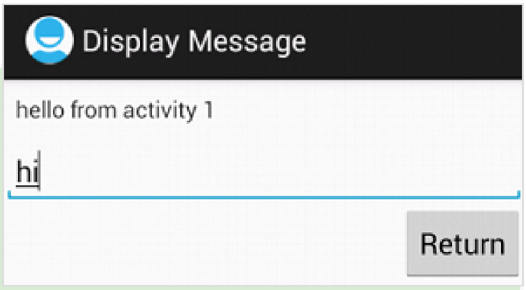
Run the App

Now, you can run the app.

6.6 Example 5: Returning Result from Second Activity

In this example, we shall illustrate how the second activity can return a result back to the first activity.





First Activity - "GetMessageActivity.java"

Modify the first activity "GetMessageActivity.java" to use startActivityForResult() to start the second activity and receive the result (instead of startActivity()). Also include an onActivityResult() handler to handle the returned result.

```
package com.mytest;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class GetMessageActivity extends Activity {
   public final static String EXTRA_MESSAGE = "com.mytest.MESSAGE";
   private int requestCode = 1;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_get_message);
   }

   /** Called when the user clicks the "Send" button */
   public void sendMessage(View view) {
      // Retrieve the message entered
      EditText txtMessage = (EditText) findViewById(R.id.txtMessageID);
      String message = txtMessage.getText().toString();

      // Create an Intent to start the second activity
      Intent intent = new Intent(this, DisplayMessageActivity.class);
      intent.putExtra(EXTRA_MESSAGE, message);
      startActivityForResult(intent, requestCode);
   }

   /** Called when the second activity returns a result */
   public void onActivityResult(int resultRequestCode, int resultCode, Intent result) {
      if (resultRequestCode == requestCode) {
         if (resultCode == RESULT_OK) {
            Toast.makeText(this, result.getData().toString(), Toast.LENGTH_LONG).show();
         }
      }
   }
}
```

On the onActivityResult() handler, we use the Intent's getData() to retrieve the return string, and dispaly on a Toast (a floating view) for Toast.LENGTH_LONG of times.

Layout of Second Activity - "res\layout\display_message_activity.xml"

Add an `EditView` and a `Button` in the layout as follows:

```
<RelativeLayout 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" >

    <TextView
        android:id="@+id/txtDisplayID"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="@dimen/padding_medium" />

    <EditText
        android:id="@+id/txtReturnedID"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/txtDisplayID"
        android:padding="@dimen/padding_medium"
        android:hint="@string/message" />

    <Button
        android:id="@+id/btnReturn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/txtReturnedID"
        android:layout_alignParentRight="true"
        android:text="@string/btnReturnLabel"
        android:onClick="returnMessage" />
</RelativeLayout>
```

We specify the `onClick` handler for the `Button` as `returnMessage()`.

Include this string reference in res\values\string.xml:

```
<string name="btnReturnLabel">Return</string>
```

Second Activity - "DisplayMessageActivity.java"

Modify the second activity to return a message to the first activity that started it.

```
package com.mytest;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class DisplayMessageActivity extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_display_message);

      // Get the message entered in the first activity from the intent
      Intent intent = getIntent();
      String message = intent.getStringExtra(GetMessageActivity.EXTRA_MESSAGE);

      // Get the display TextView for this activity and display the message
      TextView txtDisplay = (TextView) findViewById(R.id.txtDisplayID);
      txtDisplay.setText(message);
   }

   // onClick handler for Button "Return"
   public void returnMessage(View view) {
      Intent result = new Intent();
```

```
        // Retrieve the string entered in the EditView
        EditText txtReturn = (EditText) findViewById(R.id.txtReturnedID);
        String returnMessage = txtReturn.getText().toString();
        // Set the message back
        result.setData(Uri.parse(returnMessage));
        setResult(RESULT_OK, result);
        // Close the activity
        finish();
     }
  }
```

Upon clicking of "Return" button, the handler retrieves the `Intent` object, set the return data (via `setData()`). It then sets the result code (via `setResult()`) and closes this activity (via `finish()`).

6.7 Example 6: Fragment

[TODO] Convert the above example to fragment

7. Handling Input Events

Like most of the programming languages (such as Java), Android adopts the event-handing model to process input events. (Read "Java Event Handling" for the basic concepts.)

An `android.view.View` (UI component, widget or control) can register various event-listener objects via the `setOnXxxListener()`. The event-listener object shall implement the appropriate `OnXxxListener` interface and provide the various event handling methods.

```
public void setOnClickListener (View.OnClickListener l)
public void setOnKeyListener (View.OnKeyListener l)
public void setOnTouchListener (View.OnTouchListener l)

public void setOnCreateContextMenuListener (View.OnCreateContextMenuListener l)
public void setOnDragListener (View.OnDragListener l)
public void setOnFocusChangeListener (View.OnFocusChangeListener l)
public void setOnGenericMotionListener (View.OnGenericMotionListener l)
public void setOnHoverListener (View.OnHoverListener l)
public void setOnLongClickListener (View.OnLongClickListener l)
public void setOnSystemUiVisibilityChangeListener (View.OnSystemUiVisibilityChangeListener l)
```

The `OnXxxListener`s are defined as static nested class in `android.view.View`.

7.1 Click Event

7.2 Key Event

7.3 Touch Event

7.4 Multi-Touch Event

8. Handling Sensors

8.1 Accelerator Sensor

8.2 Compass Sensor

8.3 GPS Sensor

Link to Android's References and Resources

Latest version tested: Android SDK r20, Android 4.1 (API 1.6), Eclipse 4.2 (Juno)
Last modified: July, 2012