Android By Example

Introduction:

In this text I aim to give you a good introduction to the fundamentals of android programming. I will also give an introduction to things like sensors and GPS. The reason why I consider this an introductory text is because there is a vast amount of things that are possible with android devices and I could not possibly cover it all.

In this text I will deal with the fundamental skills of creating and managing the android UI along with generating your own custom controls. I will also touch on things like Google Maps, customising already provided widgets, network communication etc.

My style of teaching is with full code examples. If you wish you can copy paste these examples into your development environment. However, I think this is the worst way that you could use these examples. I would highly recommend writing out the examples line by line yourself. This will enable you to notice the coding patterns that appear frequently in Android development. It will also develop your muscle memory such that you will experience less cognitive load when you are building your own Android projects. Those of you who come from a Java Swing or JavaFX programming background will also find multiple aspects of Android development familiar and reassuring.

Above all else I hope you enjoy this small foray into the world of Android :)

My Development Environment

For all of the examples in this book I am using the Eclipse IDE (Kepler 4.3 with Java 8 support) with the Android Developer Tools (ADT) plugin (version 23.0.2). The android SDK (version 23.0.2) I have installed has support for API 20 (Android KitKat 4.4 Wear) and I will be using this for the target environment.

The minimum API version that these examples will run on is API 14 (Android 4.0 Ice Cream Sandwich) this was the last major change to the API and devices using a previous API I consider vastly out of date.

All screenshots that you see in this text are taken from a Nexus 7 running Android 4.4.4

Differences with the first edition.

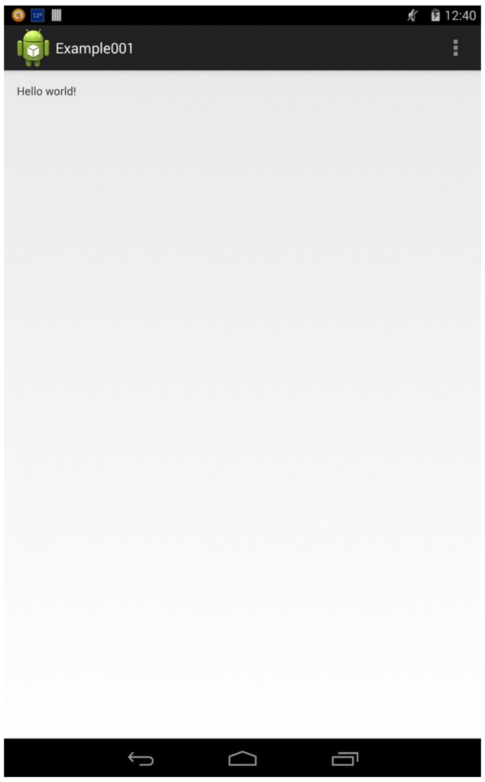
The previous edition of this text I considered to be a scattershot approach there was no real organisation to the examples plus it also had some incomplete examples. I also spent far to long introducing things like string resources, colour resources etc. These have been merged and cut down to give me more space for examples that deal with networking, etc.

I've also given more concrete examples with sensors as the previous edition only introducted them but never gave much use to them.

I will also give more information as to where and when you should expect an application to compile. This way you will be able to see the application building before your eyes and watch how the various components of code will modify and enhance the application.

Example 001: Hello world in android

In this example we will use the default code that is generated for an android project by the Eclipse ADT plugin. As with any programming language or API tutorial we will start with the hello world example. You can see this in the screenshot below. I will also introduce the code and the basic structure of an android application as well.



01) Generate a new android project with a blank activity.

02) In your project navigate to res/layout/activity_main.xml and you should see the following XML code if you are using a different development environment you should put this code in that file

```xml
<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"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.example001.MainActivity" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world" />
```

```
</RelativeLayout>
```

Explaination:

- The preferred method to constructing activity layouts is to use XML layout files. While it is possible to construct layouts through Java code alone, I can tell you from experience that it is messy and is less powerful than the options that the XML versions of layouts will provide. The layout we see here is a relative layout which we will cover in a lot more detail later. It is considered one of the most versitile of layouts
- All activity layouts must have a single layout as the root of the layout file. In this layout the root tags are considerd to be <RelativeLayout> and </RelativeLayout>
- All tags will have associated attributes that are used to change the behaviour of the control we will go through some of these attributes for the Relative layout later when we come to layouts. However we will detail some of the most important ones now
    - The xmlns:android attribute this is a standard attribute that will appear in the root node of all layouts. It is required for layouts to work. This is the same for all layouts and mandatory in the root layout.
    - android:layout_width, android:layout_height. This determines how much space the layout will use in terms of width and height. You may specify either match_parent or wrap_content here. match_parent specifies that the contents of this layout should take the full width (or full height for android:layout_height) of the parent layout. The root layout should always specify match_parent for both the width and height as activities in android are by default full screen.
        - wrap_content tells the layout or view that it should take the minimum amount of space (in either width or height) to display it's contents. Textviews are normally set to wrap_content in both directions as they will not use any extra space.
- If you look inside the RelativeLayout tags you will see another single tag called a TextView. Widgets in Android are called Views. The TextView simply displays a piece of non interactive text. All views are written as a single tag whereas layouts are written as a pair of tags. This is a convention used throughout android to make it easier to recognise views from layouts.
    - if you look inside the TextView tag you will find an attribute called android:text. this sets the text that will show on the TextView.
    - Here you will see that it is set to android:text="@string/hello_world". Android applications are split into two sections: code resources (where all your Java files are held) and non-code resources (strings, colours, images, static files etc). In fact the XML layout file you are looking at

now is considered a non-code resource.

- the @string/ part states that we are looking for a string resource. while hello_world specifies the name of the string resource we are looking for. If you take a look in res/values/strings.xml you will see the string resource for hello_world
- we will go into depth on string resources soon.

03) In your project navigate to src/<package_name>/MainActivity.java and you should see the following Java code. If you are using a different development environment you should put this code in that file.

```
package com.example.example001;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;


public class MainActivity extends Activity {

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


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

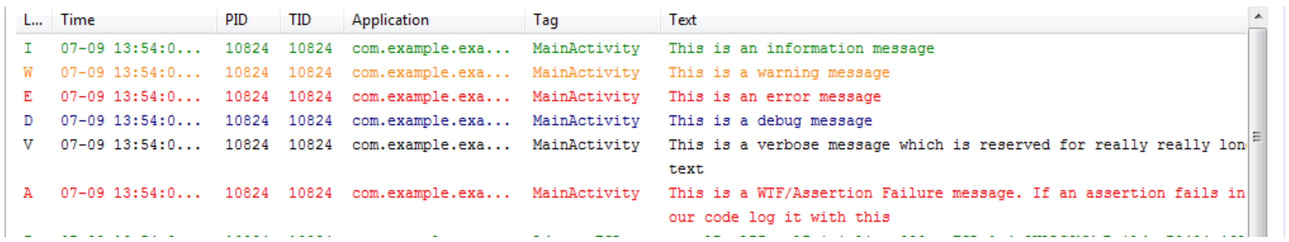
Explaination

- Here we see the Java side of Android programming. You might be surprised to find that there is no main() method to be seen. Android does not permit the standard java main() method. The reason for this is that Android will dictate what actions your application will take at any given time. This is enforced by requiring all Activities to extend the Activity class. When you create an activity you are required to have the three methods that you see above. The code above is considered standard boilerplate that will work in all cases.

- All Activities must implement the onCreate() method. We will ignore the supplied Bundle for now as we will deal with them later. But you must be aware that it is a rule in android that the first line of your onCreate() method must be a call to the onCreate() method of the superclass. If you fail to do this Android will raise an exception and will crash your application.
- The second line retrives the XML layout file that we have in step 02 and will set it to be displayed as the content for this activity by using the setContentView() method.
    - you may be wondering where the R.layout.activity_main bit is coming from. R is a class that is auto generated by android. DO NOT under any circumstances try to edit or modify this file as your changes will be lost. R contains a list of unique identifiers for each non code resource that is part of your application. It is also divided into sections. In this case we are looking at the set of identifiers that belong to XML layout files. In this case we are looking for the identifier for activity_main.xml. Android will automatically generate id names for layouts by taking the full layout filename and removing the ".xml" part.
- The onCreateOptionsMenu() method is mandatory in activities and is responsible for adding menus to the activity. We will ignore this for now as menus will be covered later in the examples.
- The onOptionsItemSelected() method is also mandatory in activities as it is responsible for responding to any clicks in the menus that are added to the activities. Again we will ignore this for now until the examples on menus but the boilerplate code above will work in all cases.
- Anytime I ask you in further examples to initialse a new activity I will require you to use the boilerplate code in 03 as a starting point

Example 002: introducing the Android logging facilities

Some of you may be wondering where you can output string messages that you can use to log the execution of your application. Android maintains a general log all applications are free to write to. In this example we will demonstrate the six different types of logging messages and their associated purpose. If you are developing an Android application through Eclipse and look at the Logcat view you will be able to see the entire contents of the log. After running this example you should find something similar to this below



```
L...  Time               PID    TID    Application         Tag           Text
I     07-09 13:54:0...   10824  10824  com.example.exa...  MainActivity  This is an information message
W     07-09 13:54:0...   10824  10824  com.example.exa...  MainActivity  This is a warning message
E     07-09 13:54:0...   10824  10824  com.example.exa...  MainActivity  This is an error message
D     07-09 13:54:0...   10824  10824  com.example.exa...  MainActivity  This is a debug message
V     07-09 13:54:0...   10824  10824  com.example.exa...  MainActivity  This is a verbose message which is reserved for really really lon
                                                                         text
A     07-09 13:54:0...   10824  10824  com.example.exa...  MainActivity  This is a WTF/Assertion Failure message. If an assertion fails in
                                                                         our code log it with this
```

01) create a new android project and use all of the code from Example 001.

02) In the onCreate() method of the MainActivity class add in the following lines of code after the call to setContentView()

```
// put some messages into the log
Log.i("MainActivity", "This is an information message");
Log.w("MainActivity", "This is a warning message");
Log.e("MainActivity", "This is an error message");
Log.d("MainActivity", "This is a debug message");
Log.v("MainActivity", "This is a verbose message which is reserved for really really long text");

Log.wtf("MainActivity", "This is a WTF/Assertion Failure message. If an assertion fails in your
code log it with this");
```

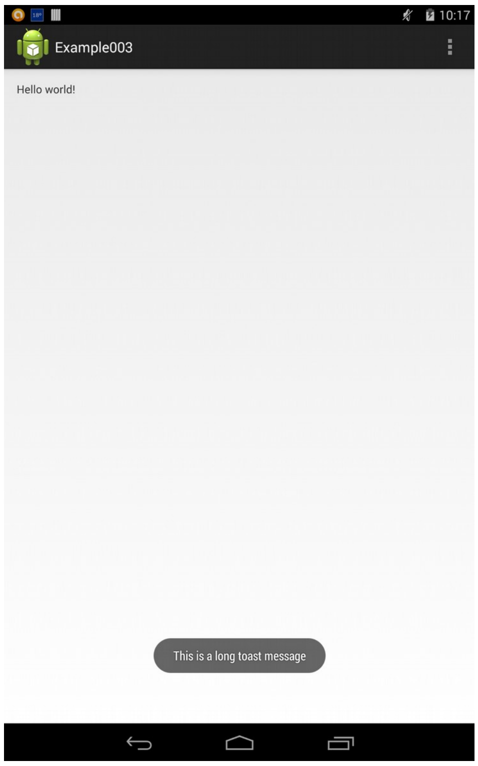
Explaination

- in order to use the log class you must import android.util.log,
- All calls to the logging methods require two arguments. The first is a tag to identify the message (usually this is a class name, to identify what code it came from). the second argument is the text that should be written to the log.
- when you open the LogCat window for the first time you will notice a set of saved filters on the left and a main window full of coloured text
- when you run your app a filter with your app name will be automatically generated for you. this only shows the messages that are emitted from your app.
- you will see 6 fields at the top of the list of messages they

  - Level: how severe is the message. A red "E" is an error or failed assertion, an orange "W" is a warning, a green "I" is information, a blue "D" is debugging, and a black "V" is a verbose message.
  - PID: the linux process ID for this application. Most apps will only have one process.
  - TID: the linux thread ID for this application. Apps will generally use threads if they need to take advantage of multiple cores or are downloading something in the background.
  - Application: the fully qualified java package name that contains the application
  - Tag: usually a class name this gives an idea as to where the message has come from
  - Text: the actual text that was written to the log
- Error messages are for catastrophic failures that prevent an action from completing. Good android apps should be able to handle these failures without crashing and offer alternatives or advice to the user. Error messages will in your applications will usually show up in the form of exception
- Assertion failures are extremly catastrophic. Assertions are conditions that are assumed to be true at a given point of app execution. If an assertion fails the application will require careful inspection by the developers as this usually indicates a bad bug being present in code (Hence why it's called a WTF message)
- Warnings are indications of possible bad behaviour but behaviour that is recoverable. These should be observed with care as they may indicate possible bugs
- Information messages are indications as to what the program is doing at any given time. Good apps will indicate important actions like file saving, saving of settings etc in the log to show that significant actions have been completed without failure.
- Debug messages: these messages are used for development purposes. you should use these to query the behaviour of variables or code during development. These are also useful to leave in your program in case a user needs to submit a debug report to you to help fix your app.
- Verbose messages: if your message is really long then use one of these messages.

Example 003: Showing a temporary message to the user informing them of application activity.

Sometimes during your application you will want to show a message indicating something that has just happened or that an error has occured. Most of the time this will be in the form of a simple piece of text that will show up temprorarily. In Android we use the Toast class to do this (I've no idea why it it called Toast but it's one of Android's quirks) You will end up seeing a screen like that below for the first few seconds that you run this application



01) create a new Android project.

02) modify the onCreate() method of MainActivity.java add these line of code after the setting of the content view

```
Toast.makeText(this, "This is a long toast message", Toast.LENGTH_LONG).show();
Toast.makeText(this, "This is a short toast message", Toast.LENGTH_SHORT).show();
```

when you run your app, you will see two temporary messages appear and will then disappear soon after. this is useful for alerting users that settings have been saved or there is no network access.

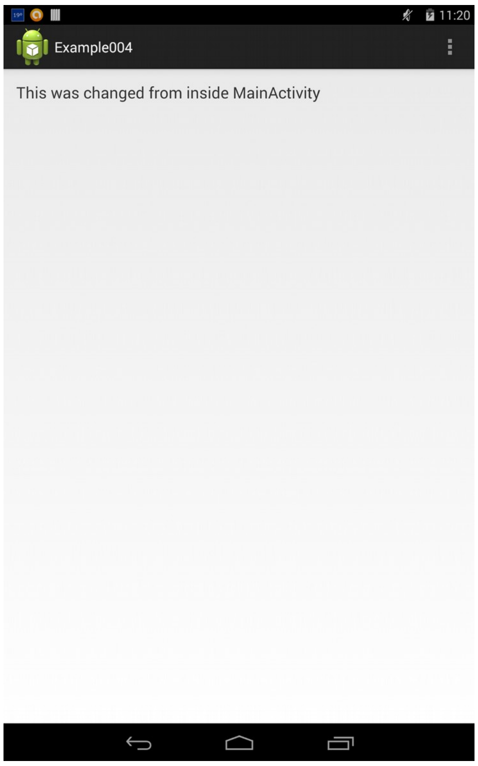
things to note here

- makeText() is responsible for generating these toast messages. it is entirelyp possible to generate all your toast messages on application start up and store them somewhere for later use. it accepts three parameters

- the first parameter is the owning activity or context. most of the time you will use the keyword this.
- the second parameter is the string that is to be displayed. This can also be a string resource from the R class.
- The third parameter is the length of time that the message will be displayed. Toast provides two values by default LENGTH_LONG and LENGTH_SHORT. LONG will only give you approximately 5 seconds while short is about 2, thus your messages have to be short and informative
- makeText().show() we call the show method to show this message immedately. if you do not wish to show the message immediately you will need to store this in a Toast variable and call the show method on it later

Example 004: Pulling a view from XML into an Activity and then modifying it there.

One of the most common tasks that is performed in Android is modifying views that appear in an XML dynamically from Java code. In this example we will take the boilerplate code and XML from Example 001 as our base and we will modify it enable us to change the text that is displayed on screen. By the time you are finished with this example you will get the screenshot seen below.



This was changed from inside MainActivity

01) start a new android project with all of the boilerplate code from Example001

02) modify this section of activity_main.xml from this

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

to this

```
<TextView
    android:id="@+id/textview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

Explaination

- here we have added in the android:id property. This specifies the id that belongs to the view. If we wish to pull a view into java code there must be a unique id attached to it. Luckily Android provides a neat shorthand for generating a new id and give it a name. This is the "@+id/textview" part
  - @+id/ specifies that the text string that follows the / is going to be the name of a new identifier and it will be assigned to this view.
  - all of these ids are accessible in Java code through the R class in the id section. You will see this in the next step

03) in the onCreate() method of MainActivity.java after the call to setContentView() add in the following lines of code (this is a good place to compile and run code)

```
// pull the textview from the XML layout that was set on this activity
TextView tv = (TextView) findViewById(R.id.textview);

// change the text and size of the text view
tv.setText("This was changed from inside MainActivity");
tv.setTextSize(20);
```

Explaination:

- The first line is where we are requesting android to get a view by its identifier from the layout that was set on this activity. The TextView class is the java representation of the textview XML element that you saw in 02. Note the typecast. You will have to typecast every view that you pull from XML.
- The findViewById() method accepts a single argument which is a unique identifier for the view. Android will then find that view and will return it to you.
  - Note the use of R.id, anytime you add a new identifer using @+id/ it will show up under R.id, also the string that you defined after the / is the exact name that is given to that identifier.
- The following lines then change the text displayed on the textview and also changes the text size as well

Example 005: The string resource types that are provided by android.

In android there are a lot of resources that are kept seperate from code to make portability and translation amongst languages easier. In this example we will introduce a few of these resources as you will be frequently using them during your own application development. The first of these resources we will explore is the string resources. This is a single xml file that contains a list of strings that may be used by your application

01) create an empty android project

02) navigate to res/values/strings.xml and observe the following code that is there

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

    <string name="app_name">Example005</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

</resources>
```

We will explore the format of this file before making any modifications to it:

- the first line declares the version of xml that is used to represent this file. All android xml files are standardised to use version 1.0 of the xml standard. We also use the unicode standard for encoding the file as this is a character set that is understood by most of the world.
- <resources></resources> this is how we indicate to android that we are declaring and defining non-code resources. For the sake of compression and translation all non code resources (static strings, layouts, images) are seperated from compiled java byte code. Usually a text compression algorithm can be used on these resources to reduce their size by upto 90%. All files that declare resources must have one pair of these tags as the root tags of the resources file. If they are not the root then your resources may not be added correctly.
- <string name="hello_world">Hello World!</string>. This is how we define a single string. The contents of the string are defined between the pair of tags. When you define a new string resource you are required to provide a unique name that identifies that string. This name must be set in the name attribute. When you ask for the string by this name (either in Java code or XML) the contents of the string will replace the identifier.
- If you remember from the previous example in the XML layout you saw this code
    - android:text="@string/hello_world"

- here android was directed to look for the string resource named hello_world and replace this identifer with the text named hello_world

03) add in the following lines to strings.xml below the line declaring hello world

```
<string name="my_string_one">my string one</string>
<string name="my_string_two">my string two</string>
```

Explaination:

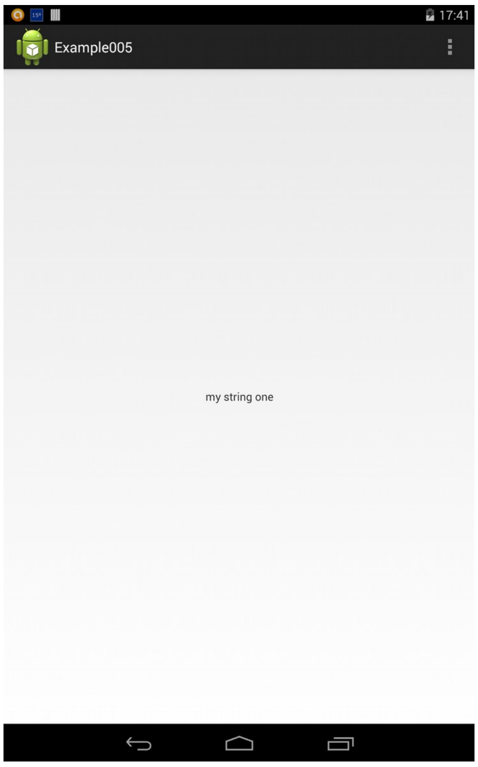
- Here we are defining a few of our own strings which we will use to modify the layout

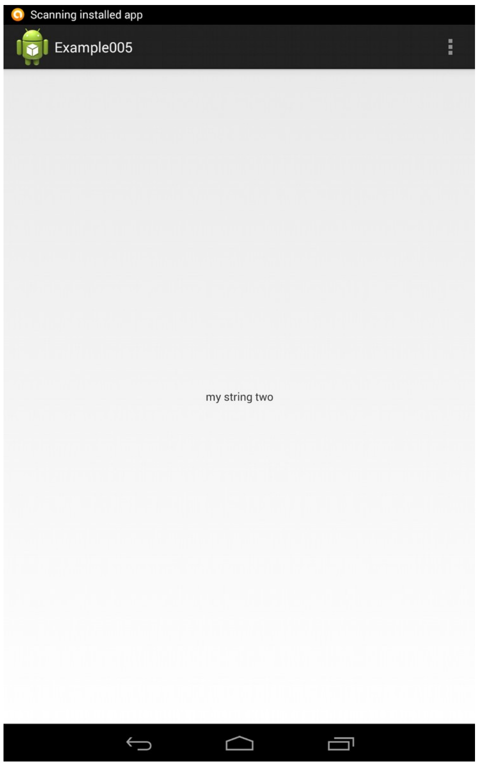
04) clear out activity_main.xml and replace it with the following code (good place to compile and run code)

```
<FrameLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>
      <TextView
      android:id="@+id/textview"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:gravity="center"
      android:text="@string/my_string_one"
      />

</FrameLayout>
```

Explaination:

- Here we have the same layout as the previous example. The only exception is that we have changed android:text to reference "@string/my_string_one" instead of hello_world. If you now run this application you will see the string you defined in the previous step take the place of hello world. Change "@string/my_string_one" to "@string/my_string_two" to see the other string in its place.
- Running both should give you the two screenshots that we see below


17:41
Example005
my string one


Scanning installed app
Example005
my string two

05) go back to strings.xml and add in the following code after the declaration of of my_string_two but before the </resources> tag

```
<plurals name="dollar_plural">
    <item quantity="one">1 Dollar</item>
    <item quantity="other">%d Dollars</item>
</plurals>
```

Explaination:

- Here we are introducing the plural resource type. The plural resource type is there to give you gramatically correct quantities in strings. In this particular case we are giving the correct spellings for when we have a single dollar or mulitple dollars (I would use Euro here but officially you are not ment to say Euros as plural you are meant to use Euro still.)
- There are about six different quanties that can be set for words but the majority of time one and other will do. One of course specifying that we have a single item and other specifying what to use in all other cases. Each quantity must be delared in a pair of item tags. Note that for the singular we explicitly mention the quantity in the string of text (as there is only one value for the singular) whereas in the other we specify "%d". Those of you from a C background will recognise this as a format specifier. This is a placeholder in which we will substitue in a value before rendering the text. We will do this in the next step

06) in MainActivity.java inside the onCreate() method add the following lines after the call to setContentView() (good place to compile and run code)

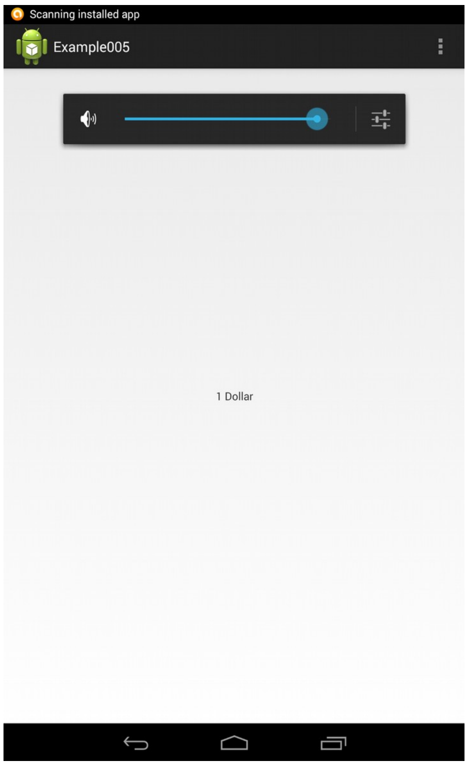
```
// get the plural from the resources and apply it to a string for testing
Resources resources = getResources();
String s = resources.getQuantityString(R.plurals.dollar_plural, 1, 1);

// pull the textview from the layout and set the string
TextView tv = (TextView) findViewById(R.id.textview);
tv.setText(s);
```

Explaination:

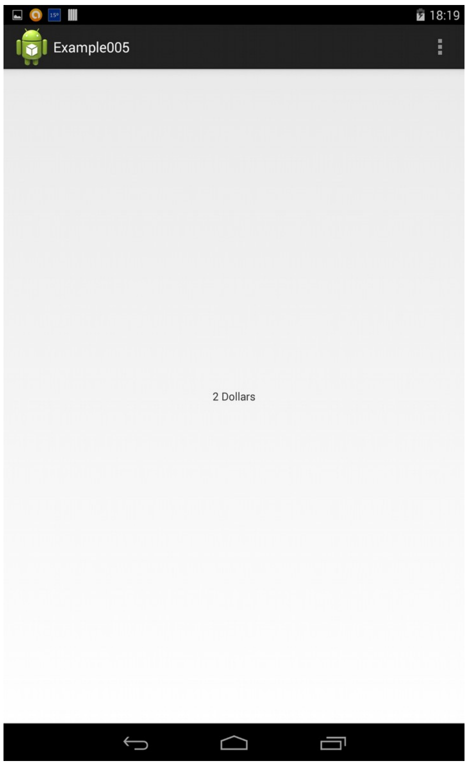
- The first line declares a resources object which is attached to every activity. The resources object is there to parse and pull more complex resources from XML into Java code. One such complex resource are plurals. We will use the resources object to handle these for us.
- On the second line is where we are actually generating the plural itself by using the getQuantityString() method of the resources object. This method takes three arguments. The first is an identifier to the plural resource that we are using. All plurals have their identifiers stored under R.plurals. The second argument specifies what quantity string you wish to use, while the third argument specifies what value you want to substitue in for the format specifier in the string itself. Generally we will keep the second and third values the same.
- Below this we pull the text view from the XML as before and and set the

value of the quantity string as its text. This will overwrite the string that was set for the textview in the XML layout.

- If you change the second and third values of the call to getQuantityString() to 2 and run again you will see the quantity string changing. After running both times you should get the screenshots that you see below



1 Dollar



2 Dollars

Example 006: introducing string arrays and general arrays

In the previous example we introduced the concept of strings and plurals as resources. In this example we will introduce the array types that can be stored as resources. There are two types of arrays that are provided in XML. They are string arrays for a collection of related strings and also general arrays that are used for related collections of numbers. We will show both types in this example.

01) create a new android project

02) in strings.xml add in the following XML after the definition of the hello world string.

```
    <string-array name="my_string_array">
        <item>First string</item>
        <item>Second string</item>
        <item>Third string</item>
    </string-array>
```

Explaination:

- this defines an array of strings that are all connected. Note how this follows a similar format to the plurals example that you saw earlier.
- All string arrays have to be declared with the <string-array></string-array> tag pair. All string arrays must have the name attribute set to a unique name that is not used by any other array.
- Each item in the string array must be enclosed in <item></item> tags. The first item will have an index of zero in the string array.

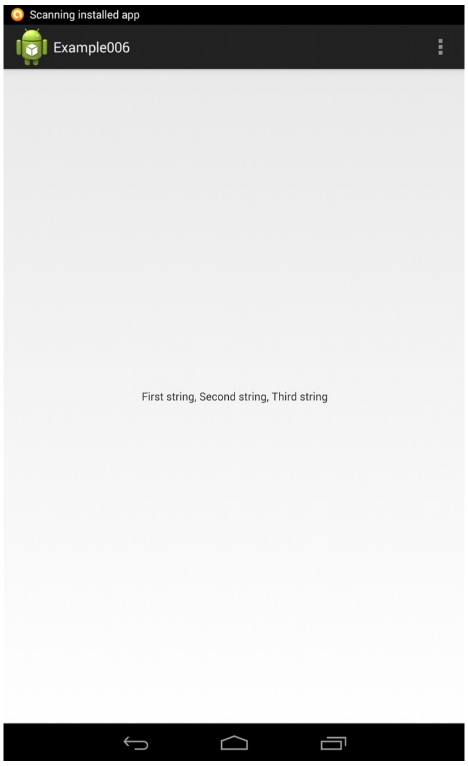
03) In MainActivity.java add the following code after the call to setContentView() in the onCreate() method. (good point to compile and run code)

```
// get access to the string array and construct a string from all three of
// its elements
      Resources resources = getResources();
      TypedArray ta_string_array =
resources.obtainTypedArray(R.array.my_string_array);

      String s = ta_string_array.getString(0) + ", " +
      ta_string_array.getString(1) + ", " + ta_string_array.getString(2);

      // get the textview and set the string
      TextView tv = (TextView) findViewById(R.id.textview);
      tv.setText(s);
```

Explaination

- Like the plurals example the string array is considered a more complex resource thus we need to use the resources object to convert the string array XML representation into a java string array representation. To do this the first line like before is get access to the resources object. The second line is how we get access to our string array. Android has its own TypedArray class for handling all arrays that are declared in XML. Thus if we want to get access to our strings we need to use the typed array.

- We use the obtainTypedArray() method of the resources object to construct and return our required typed array. The only argument you need to provide to this method is the ID of the array that you wish to retrieve from XML. Note that all ids for arrays are stored in R.array
- The following line constructs a single string by taking all three strings that compose the string array and joining them together. As the typed array can hold different types for each individual element of the array we have to declare the type we are retrieving at an index. This is why we use the call to getString() with a single index. This tells android that we know there is a string at the given index. Like before we take the constructed string and set it on the text view.
- If you run this code you will get the screenshot below



First string, Second string, Third string

04) create a new file in res/values/ called arrays.xml and provide it with the following XML code

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="int_array">
        <item>1</item>
        <item>2</item>
        <item>3</item>
    </array>
    <array name="float_array">
        <item>1.5</item>
        <item>1.618</item>
        <item>3.14159</item>
    </array>
</resources>
```

Explaination:

- In res/values/ you can declare as many different XML files for holding your resources as you wish. You are free to break up your arrays as you see fit. But like all resource files there must be a tag declaring which version and encoding of XML you are using and also a root tag pair of <resources></resources>
- general arrays are declared with the <array></array> tags they work in the same way as string-arrays.

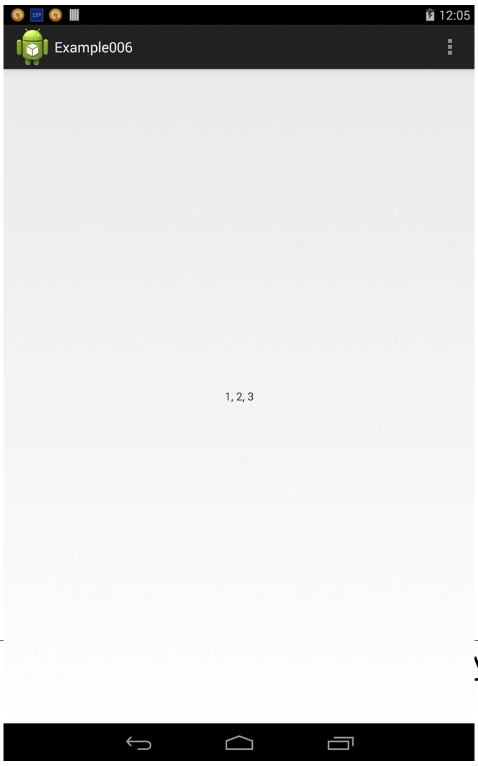
05) in MainActivity.java onCreate() method remove the code that was added earlier and replace it with the following code (good place to compile and run

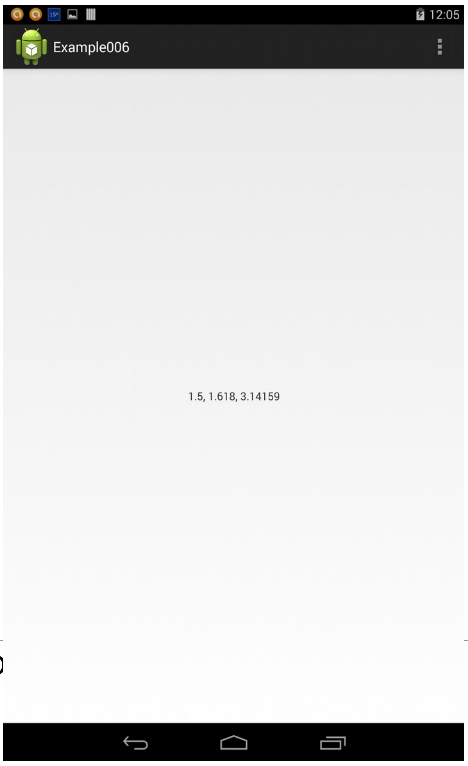
code)

```
// get access to the string array and construct a pair of strings
// from all three of its elements
Resources resources = getResources();
TypedArray ta_int_array = resources.obtainTypedArray(R.array.int_array);
TypedArray ta_float_array = resources.obtainTypedArray(R.array.float_array);

// construct strings for displaying arrray contents
String s1 = ta_int_array.getInt(0, 0) + ", " +
            ta_int_array.getInt(1, 0) + ", " +
            ta_int_array.getInt(2, 0);
String s2 = ta_float_array.getFloat(0, 0) + ", " +
            ta_float_array.getFloat(1, 0) + ", " +
            ta_float_array.getFloat(2, 0);

// get the textview and set the string
TextView tv = (TextView) findViewById(R.id.textview);
tv.setText(s1);
```

Explaination:

- Like before we have to use the resources object to obtain and generate our typed arrays.
- The only difference here between the string arrays earlier is that we are using different methods getInt() and getFloat() to retrieve the values from the arrays.
- Note that these two functions take two arguments. The first argument is an index into the array. The second argument is a default value that will be returned if the parsing of the number fails or the element does not exist in the array.
- If you run the code as is you will get the screenshot on the left but if you change the last line to setText(s2) you will get the screenshot on the right.


12:05
Example006
1, 2, 3


12:05
Example006
1.5, 1.618, 3.14159

Example 007: introducing the colour resources

There are yet more resources that are considered non code resources in android. Another one of these are colour resources. Colours may be stored in XML for in your application. This can be a good way to modify the look and feel of an application without manually going through code by hand replacing all instances of the colour. Please note that because this is an American produced system that colour is spelled as color (because of US English)

01) start with a new android project

02) replace all XML in main_activity.xml with the following XML.

```
<FrameLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>
    <TextView
            android:id="@+id/textview"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:gravity="center"
      android:text="@string/hello_world"
    />
</FrameLayout>
```

03) create a new file in res/values/ called colours.xml and fill it with the following XML

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="my_red">#ffff0000</color>
    <color name="my_green">#ff00ff00</color>
    <color name="my_blue">#ff0000ff</color>
</resources>
```

Explaination:

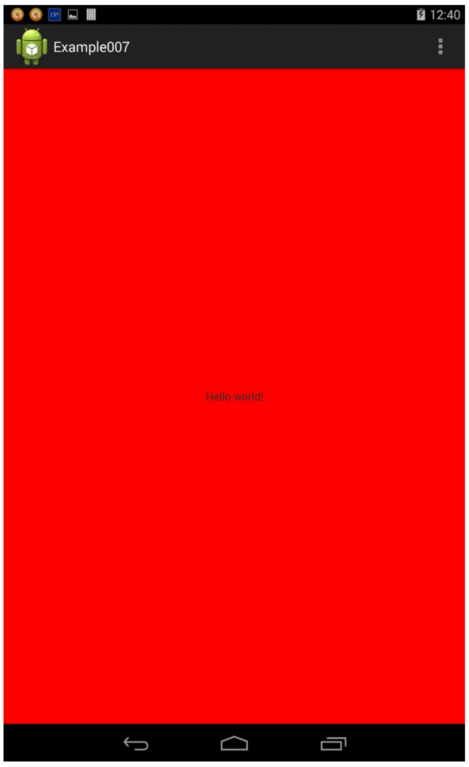
- again as with previous resource files we need to declare the xml version and also the root resources tag pair.
- Colours are declared with the <color></color> tag pair (note the spelling) all colours are required to have a unique name.
- Colours are specified in HTML hexadecimal style. Colours are 32-bit depths and are of the form #AARRGGBB which represents 8-bits for Alpha, Red, Green, and Blue channels.

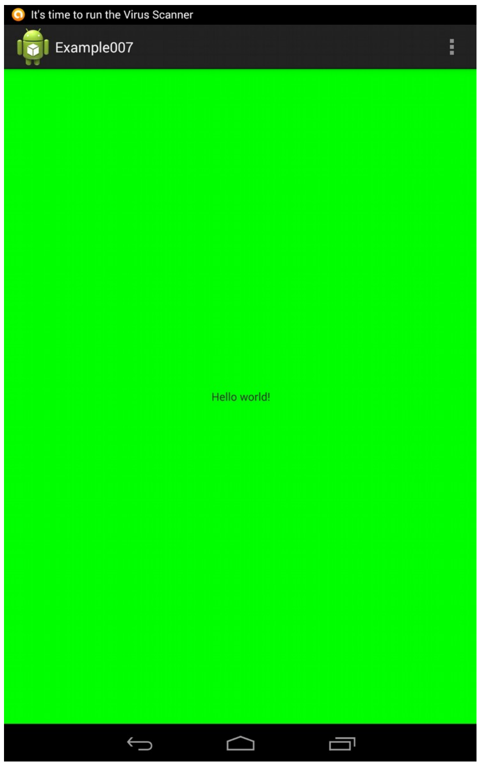
04) modify the onCreate() method of MainActivity.java. Add in the following lines after the call to setContentView() (good place to compile and run code)

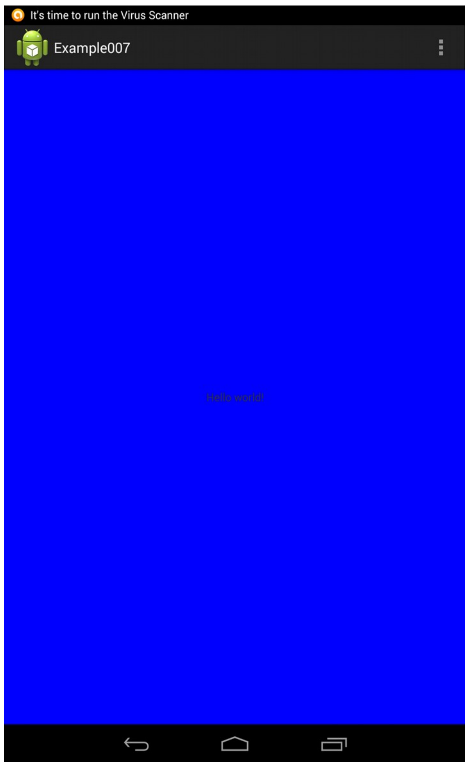
```
// get the textview and set its background colour
Resources resources = getResources();
TextView tv = (TextView) findViewById(R.id.textview);
tv.setBackgroundColor(resources.getColor(R.color.my_red));
```

Explaination:

- Like before we need access to the resources object in order to convert the colours from html style to actual color objects using the getColor() method and by providing a colour ID to it. Not surprisingly there is a section in the resources (R.color) dedicated for storing colours.
- And in the final line we set the background colour of the textview to be red. If you change this to green and blue yourself you will get all three screenshots that you see below.







Example008: Introducing Drawables and how to set them in XML and code

The last of the resources that we will observe is that of Drawables. Drawables are anything that is an image and can be rendered on screen. These can be raster based and vector based images that can be displayed on screen. In this case we will use an image that is provided by default with every android applicaton, and will use this as the basis for our experiments.

01) start with a new android project

02) add in the following line to strings.xml

```
<string name="iv_description">Random image here</string>
```

Explaination

- Whenever an image view is added in any android layout there has to be an alternate text present in case the image does not load or show (similar to the alt attribute of an img tag in html)
- The string can have any content you wish but ideally something short and descriptive

02) replace all XML in activity_main.xml with the following XML (good place to compile and run)

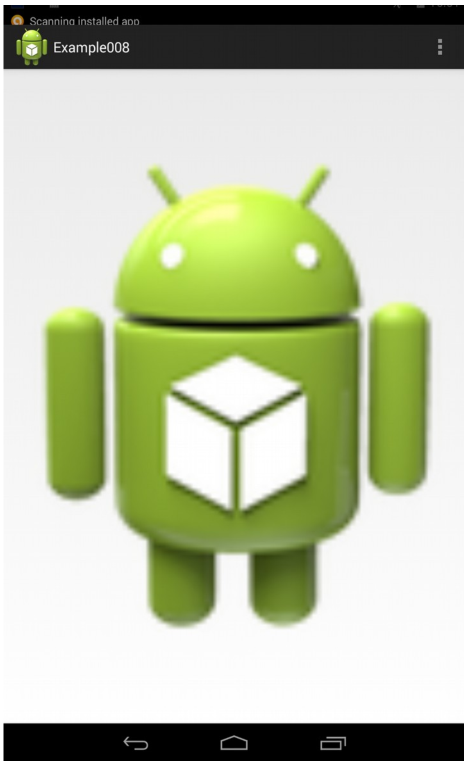
```
<FrameLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>
    <ImageView
      android:id="@+id/imageview"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:gravity="center"
      android:src="@drawable/ic_launcher"
      android:contentDescription="@string/iv_description"
    />
</FrameLayout>
```

Explaination:

- Here we have a frame layout as before. Except instead of a textview we have an image view. Here we have given the image view an ID and told it to use the drawable called ic_launcher as the image to be displayed.
- The contentDescription attribute states the text that will be displayed in place of an image if an image fails to load or is non existant. Android will emit warnings if you do not have a contentDescription attached to an image view
- You may be wondering how and where android finds the drawable called ic_launcher. If you look in the res/ directory you will find four

subdirectories called drawable-ldpi/, drawable-mdpi/, drawable-hdpi/, and drawable-xhdpi/ These directories represent the images that should be used on low DPI (Dots Per Inch) medium DPI, high DPI and extra high DPI screens. In at least three of these directories you will find an image called ic_launcher.png.

- To make identifiers for drawables android will take the name of the image and remove the file format identifier at the end of the image (in this case the .png will be removed) thus this means that all of the names for your drawables must be unique and they will be found in R.drawable
- There is good reasons for why we have four directories for raster images. To scale a raster image up or down requires interpolation which is expensive and reduces image quality. If a raster image is scaled it will become blurry. Thus good android apps will provide three or four copies of images each of which have already been scaled for the appropriate DPI and they will be stored in their respective directories.
- When you run this code you will see the image below and it is likely that it will appear blurry on your screen.



03) remove the attribute android:src from the XML layout and add in the following code after the call to setContentView() in the onCreate() method of MainActivity.java

```
Resources resources = getResources();
```

```
ImageView iv = (ImageView) findViewById(R.id.imageview);
iv.setImageDrawable(resources.getDrawable(R.drawable.ic_launcher));
```

Explaination

- this is how you load an image and display it in android from your resources in java code. Like before you need a resources object to convert the raster image into a Drawable object that can be used with the ImageView's setImageDrawable() method.

Example009: using the android provided resources.

In many applications there are common attributes or strings that are used by the majority of applications. Some of these common attributes are provided as common resources by android that your application is permitted to use. In this example we will show how to pull a string and a colour from those android provided resources for use in our own applications. Another benefit of this with regards to strings is that they will be automatically translated for you depending on the user's language of choice.

01) start with a new android project.

02) clean out the XML from activity_main.xml and replace it with the following XML (good place to compile and run code

```
<FrameLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>
    <TextView
      android:id="@+id/textview"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:gravity="center"
      android:text="@android:string/emptyPhoneNumber"
      android:background="@android:color/holo_blue_dark"
    />
</FrameLayout>
```

Explaination:

- Note how before the desired resources we prepend "@android:" this is to state that the resources we are pulling should come from the android common resources. In this case we are setting the string to a message stating that there is no phone number while the background will be one of the standard android holo theme colours.

03) remove the android:text and android:background attributes from the TextView and add the following code after the call to setContentView() in the onCreate() method of MainActivity.java (good place to compile and run code

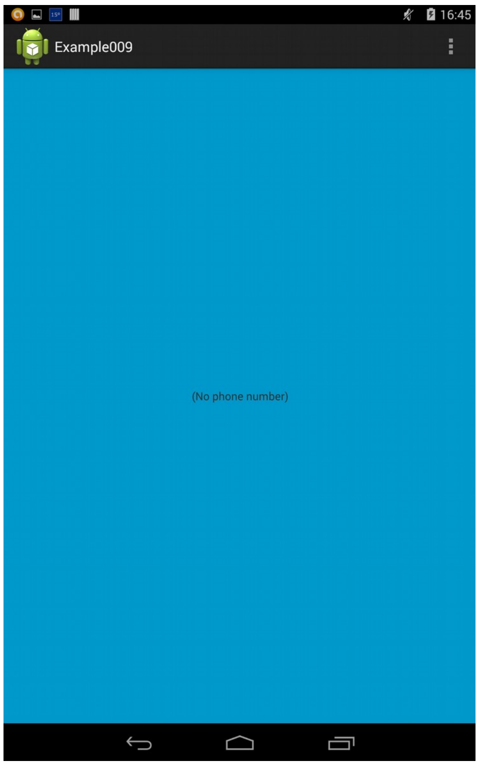
```
      TextView tv = (TextView) findViewById(R.id.textview);
```

```
tv.setText(android.R.string.emptyPhoneNumber);
tv.setBackgroundResource(android.R.color.holo_blue_dark);
```

Explaination

- this produces the same window as the screenshot below but through Java code instead of XML.
- Note that in order to get access to the android provided resources we had to prefix R with android.

- Every single package that you define will get its own autogenerated R file. Thus if you define multiple packages and you wish to share their resources you can simply prefix R with <package name>. Where package name is the name of your package.



Example010: Introducing the frame layout.

In the following examples we will explore the layouts that are provided by android and that are used to arrange elements within activities. There are multiple useful layouts that are available. We will first explore in a bit more detail the frame layout as this is the layout that we have been using thus far.

01) start with a new android project

02) remove the XML from activity_main.XML and replace it with the following XML. (good place to compile and run)

```
<FrameLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
    />

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/app_name"
    />

</FrameLayout>
```

Explaination:

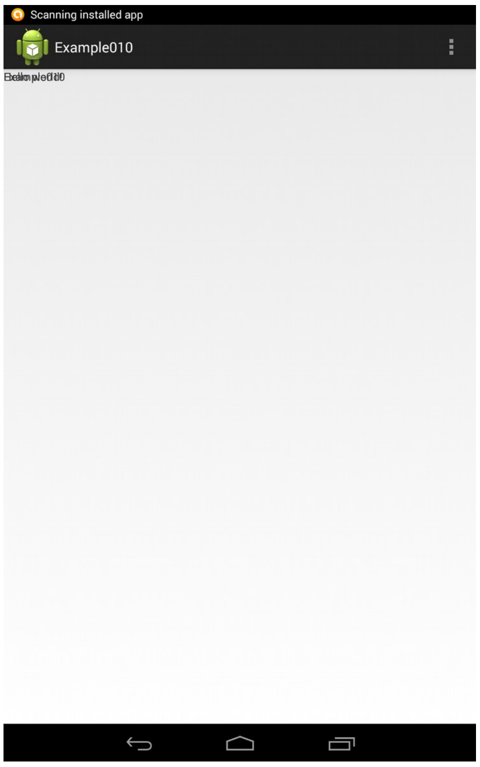
- The FrameLayout is not the most useful of layouts as it only permits positioning of elements in one of 9 positions (positions are divided into three rows and three columns)
- The FrameLayout does not handle overlap between views. (run the code now and you will get the screenshot on the left) by default every view will be placed in the top left of the FrameLayout.
- This layout should be used with one or two views at most or as a place holder for other things.

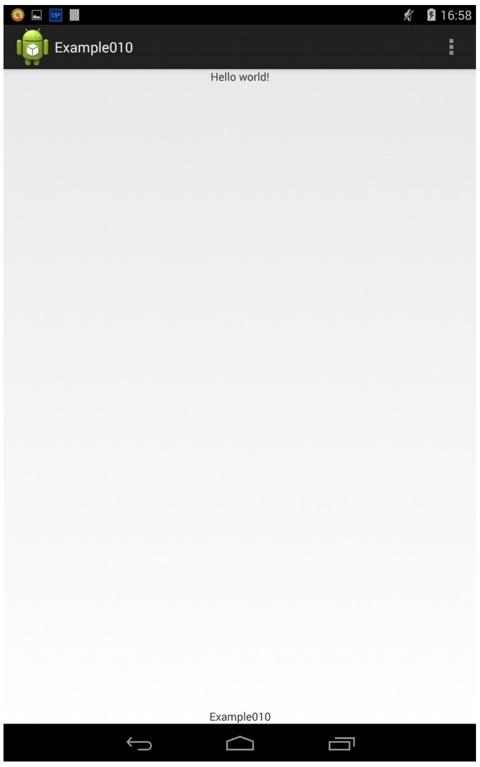
03) modify our text views to have the following layout_gravity properties.

```
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="top|center_horizontal"
      android:text="@string/hello_world"
    />

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="bottom|center_horizontal"
      android:text="@string/app_name"
    />
```

Explaination:

- Here we are telling the framelayout where to position both of our text views. In the first textview we state that we want it to be on the top row and in the middle column. In the second textview we state that we want it to be on the bottom row and in the middle column.
- This will yield you the screenshot on the right





Example011: The vertical LinearLayout

In the previous example we saw the FrameLayout and discovered that it is not particularly useful. In this example we will introduce the LinearLayout in its vertical form and will show the horizontal form in the next example. Here we will also introduce the android button (for display purposes only, however we will do event handling later) This example will produce three buttons arranged in a single column.

01) start with a new android project

02) add in the following strings to strings.xml

```
    <string name="button_one">Button one</string>
    <string name="button_two">Button two</string>
    <string name="button_three">Button three</string>
```

03) replace all XML in activity_main.xml with the following XML (good place to compile and run)

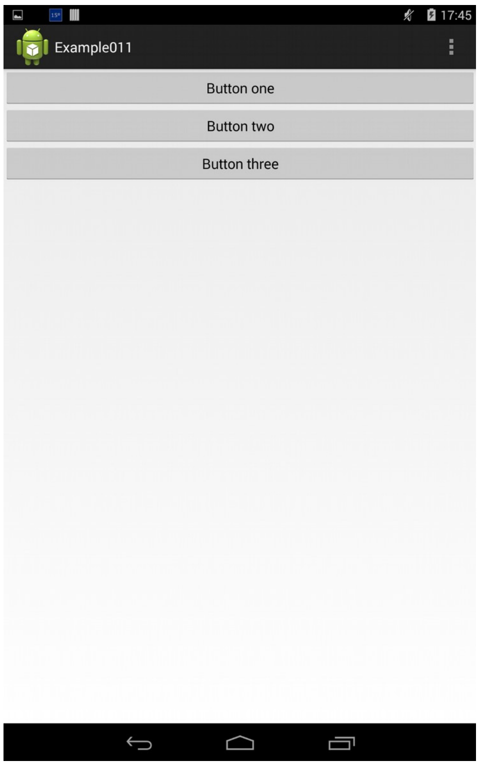
```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/button_one"
    />

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/button_two"
    />

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/button_three"
    />
</LinearLayout>
```

Explaination:

- This is how a vertical linear layout is specified. We use the <LinearLayout></LinearLayout> tag pair to define the vertical linear layout. As this is the root layout of this activity we specify it to take the full width and height of the activity. However, there is a third attribute listed in the LinearLayout tag android:oreientation which in this case is set to vertical.
- The LinearLayout interprets the orientation attribute to mean the

direction this linear layout is working in. Vertical states that we want a vertical linear layout and as you will see in the next example setting this to horizontal will generate a horizontal linear layout.

- <Button /> introduces the Button view which implements a standard push button. Here we only use if for display purposes and nothing else. Note that we have set the button to take the maximum width possible but only the minimum required to display the button vertically. Buttons like text views require text to be set for display through the android:text property.
- Running this code will yield you the screenshot below



Example012: The Horizontal LinearLayout

In the previous example we saw the vertical linear layout. In this example we will show the horizontal linear layout and a couple of potential issues you may run into when you are using it for the first time. As before the buttons are here for show only and do nothing.

01) start with a new android project

02) add the following strings to strings.xml

```
    <string name="button_one">Button one</string>
    <string name="button_two">Button two</string>
    <string name="button_three">Button three</string>
```

03) replace all XML in activity_main.xml with the following XML (good place to compile and run)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="horizontal"
>
    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/button_one"
    />

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/button_two"
    />

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/button_three"
    />
</LinearLayout>
```

Explaination:

- This is essentially the XML from the previous example except with the oreintation changed from vertical to horizontal. Those of you from a GUI background would expect that the layout would give equal weight to each of these buttons by default, however you would be wrong. All buttons are given the full width of the parent so when this code is run you will only see the first of the three buttons as the second and third is missing from view. (as seen in the screenshot on the left)
- To get around this problem we have to use the weighting system to assign weights to each view in the horizontal layout and to determine

how much space each view will get.

04) modify the XML of the buttons in activity_main.xml to look like the following (good place to compile and run)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="horizontal"
>
    <Button
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/button_one"
    />

    <Button
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/button_two"
    />

    <Button
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/button_three"
    />
</LinearLayout>
```

Explaination:

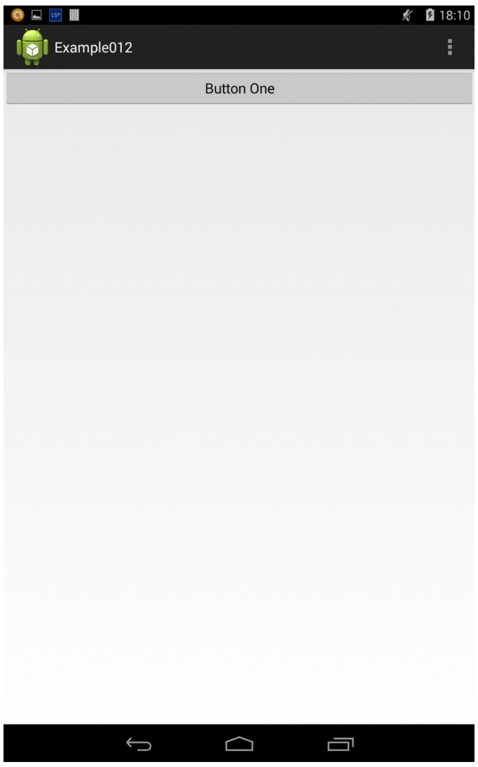
- here we introduce the weighting system for the horizontal layout. In this case we set the width of each of the buttons to be zero. The “dp” after the zero stands for Display independant Pixels. A display independant pixel will take up the same amount of screen space regardless of the DPI of the device screen.
- An extra attribute has been added to each button called layout_weight and it has been given a value of 1 for all buttons. The linear layout will add up the weights to get a sum of all weights in the horizontal layout in this case that total is 3. Each of the viewss has its own individual weight divided by the total to determine how much horizontal space it gets. In this case all viewss have a weight of 1 meaning all views will get 1/3rd of the horizontal space.
- This code produces the screenshot that you see in the middle
- At this point you may have noticed that android complains that you have a button bar and states that all buttons should be borderless in a button bar. This is a design decision of android's and if you wish to respect it then modify the layout to look like the XML in step 05.

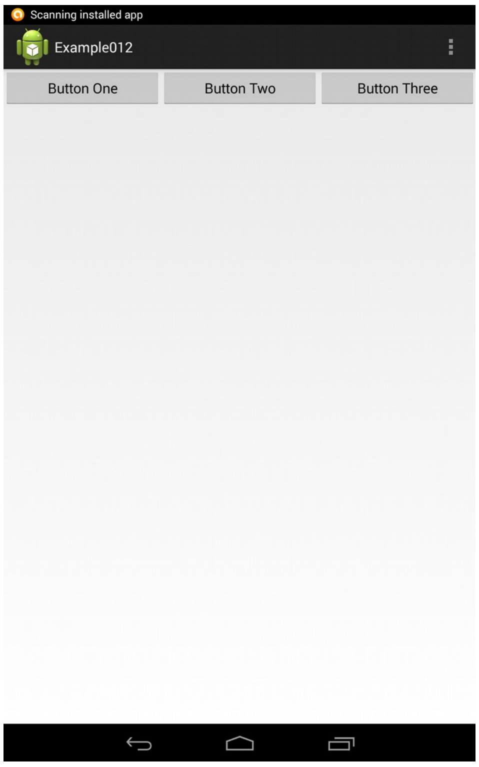
05) modify the entire XML of activity_main.xml to look like the following

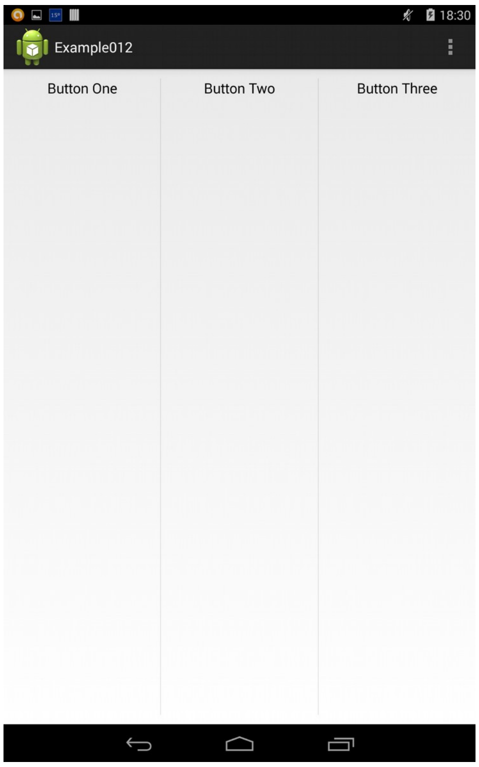
```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="horizontal"
      style="?android:attr/buttonBarStyle"
>
    <Button
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/button_one"
      style="?android:attr/buttonBarButtonStyle"
    />

    <Button
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/button_two"
      style="?android:attr/buttonBarButtonStyle"
    />

    <Button
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:text="@string/button_three"
      style="?android:attr/buttonBarButtonStyle"
    />
</LinearLayout>
```

Explaination:

- When buttons are arranged in a horizontal layout it is assumed by android that you are making a button bar. In this case you will be warned to use the approrpate styling by adding in a style to the layout itself and also to each of the individual buttons. The format of the style may be a bit confusing but when broken down it can be explained.
    - ?android:attr is asking android to query the current set of style attributes for whatever is listed after the /
    - these attributes are set by themes thus it enables you to adjust your application to fit in with the theme that the user has set on their android device. The screenshot that you see on the right is the default button bar style on a nexus 7


18:10
Example012
Button One


Scanning installed app
Example012
Button One
Button Two
Button Three


18:30
Example012
Button One
Button Two
Button Three

Example013: LinearLayout within a LinearLayout

Sometimes to achieve the desired layout effect it may be necessary to embed one layout within another. In this example we will use two linear layouts to arrange four buttons. We will have a vertical linear layout as the main layout that will contain a single button, then a horizontal linear layout containing two buttons then another single button to finish. It is common practice to combine multiple layouts to achieve a more complex layout of elements in an activity.

01) start with a fresh android project

02) add the following strings into strings.xml

```
    <string name="button_one">Button One</string>
    <string name="button_two">Button Two</string>
    <string name="button_three">Button Three</string>
    <string name="button_four">Button Four</string>
```

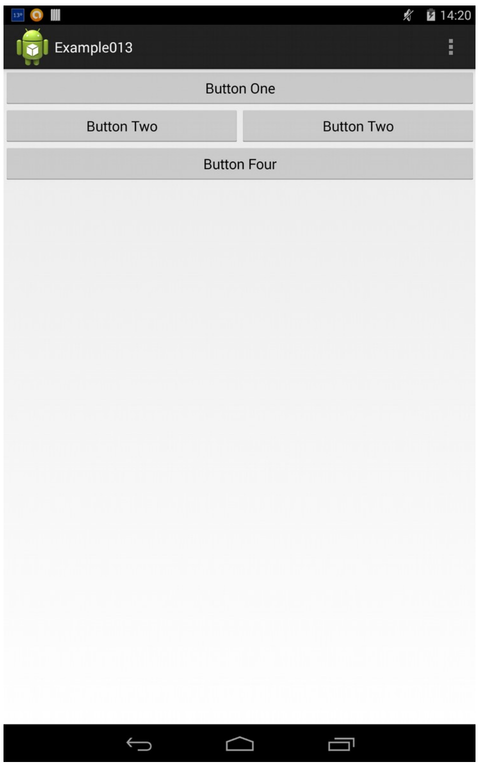
03) replace all XML in activity_main.xml with the following XML (good place to compile and run code)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/button_one"
    />

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal"
    >
        <Button
                  android:layout_width="0dp"
                  android:layout_height="wrap_content"
                  android:layout_weight="1"
                  android:text="@string/button_two"
        />

        <Button
                  android:layout_width="0dp"
                  android:layout_height="wrap_content"
                  android:layout_weight="1"
                  android:text="@string/button_two"
        />
    </LinearLayout>

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
```

```
        android:text="@string/button_four"
    />
</LinearLayout>
```

Explaination:

- Here we combine a vertical linear with a horizontal linear layout. Because the tags for the horizontal linear layout are contained within the tags for the vertical linear layout it is assumed that the horizontal layout is a child of the vertical layout.
- The vertical layout is set to take the full width and height of the activity as it is the root layout of this activity. We start by placing a single button as the first item in the vertical linear layout.
- Directly under this we have another linear layout that this time is set to take the full width of the parent (the vertical linear layout) and only as much space as necessary in the vertical direction. If this was set to match parent some unpredicatable results would occur.
- The two buttons inside the horizontal linear layout are then set to share the horizontal space by using a weighting system as per the previous example.
- Once the linear layout is finished then a final button is added to the main vertical layout.
- this will produce the screenshot that you see below



Example014: Introducing the RelativeLayout

The relative layout is one of the most common layouts that is used in android, in fact it is the default layout for every generated layout file. What the relative layout permits you to do is to position views in relation to the parent and also the other views that are present in the layout. To demonstrate this we will create a set of four buttons. One will be in the center of the layout and will take the maximum width possible but only the necessary height. While the other three will be on the bottom of the parent and defined relative to each other.

01) start with a new android project

02) add the following strings to strings.xml

```
    <string name="button_one">Button One</string>
    <string name="button_two">Button Two</string>
    <string name="button_three">Button Three</string>
    <string name="button_four">Button Four</string>
```

03) replace all XML in activity_main.xml with the following XML (good time to compile and run)

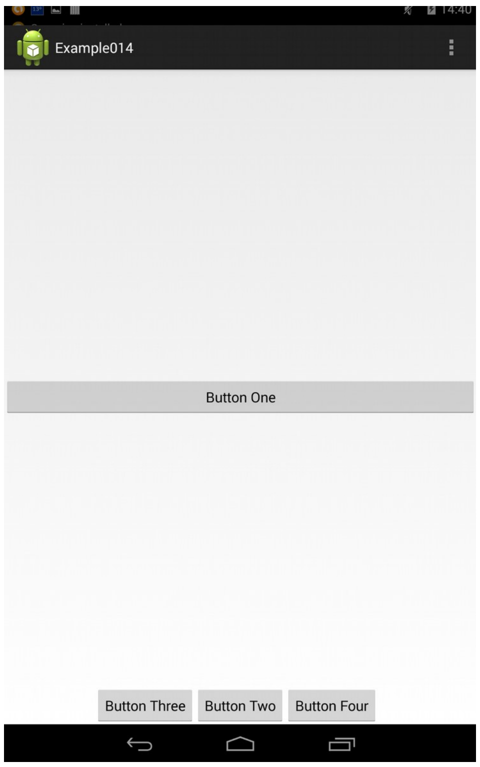
```
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/button_one"
      android:layout_centerInParent="true"
    />

    <Button
      android:id="@+id/button_two"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/button_two"
      android:layout_centerHorizontal="true"
      android:layout_alignParentBottom="true"
    />

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/button_three"
      android:layout_alignBaseline="@id/button_two"
      android:layout_toLeftOf="@id/button_two"
    />

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/button_four"
```

```
        android:layout_alignBaseline="@id/button_two"
        android:layout_toRightOf="@id/button_two"
    />

</RelativeLayout>
```

Explaination

- To declare a RelativeLayout in Android we use the <RelativeLayout></RelativeLayout> xml tag pair. As before if this is the root layout of an activity it must have the xmlns:android attribute and the width and height of the layout must be set to match_parent.
- Look at the layout parameters for the first button. You will see that we set the dimensions and the text as before. However, the layout parameters have changed. In this case we have a new attribute called android:layout_centerInParent which is set to true. This attribute only works for the RelativeLayout and asks the relative layout to place this view in the center of the layout
- Look at the layout parameters for the second button. Like the previous button the dimesions and text are set in the same way. However, we have added an id to this button. If you wish to define a view relative to another view you must have a unique identifier for that view. In this case we give the id of “button_two” which will be used later in the layout file. Although we already have a string with an id of “button_two” we can reuse the name because ids for strings and views are held separately in the R class.
    - The last two parameters layout_alignParentBottom=”true” states that we want this view to appear at the bottom of the relative layout, while layout_centerHorizontal=”true” states that we want this view to appear in the middle horizontally.
- The third button has the dimensions and text set as before but we would like this to appear to the left of button two. We have two attributes here. The first layout_toLeftOf states that we want this view to appear to the left of another view. The other view must be specified using a unique id. In this case we use the id that we added to button two. The second attribute layout_alignBaseline also uses the same id. If this attribute was omitted then button three would appear to the left of button two but would be at the top of the screen. By aligning the baselines we can guareente that the views will appear together.
- The fourth button is similar to the third button but instead of toLeftOf we have toRightOf
- Be aware that if you write this code that you will get warning stating that from API 17 onwards we should be using toStartOf and toEndOf instead. As this book is aimed at API 14 devices onwards we will stick with toLeftOf and toRightOf for now.

- this should produce the following screenshot



Example015: Using the table layout to arrange items in a grid like manner

Although android provides a GridLayout in practice it is extremly restrictive as you cannot assign elements in a grid layout any kind of weight. Also the GridLayout cannot be expanded to take the full size of a parent it will always produce “wrap_content” behaviour. Thus it is highly recommended that to get grid like behaviour that you use the TableLayout instead. In this example we will create a 3x3 grid of buttons using the table layout and they will all be assigned equal weight.

01) start with a new android project

02) add the following strings into strings.xml

```
    <string name="button_one">Button One</string>
    <string name="button_two">Button Two</string>
    <string name="button_three">Button Three</string>
    <string name="button_four">Button Four</string>
    <string name="button_five">Button Five</string>
    <string name="button_six">Button Six</string>
    <string name="button_seven">Button Seven</string>
    <string name="button_eight">Button Eight</string>
    <string name="button_nine">Button Nine</string>
```

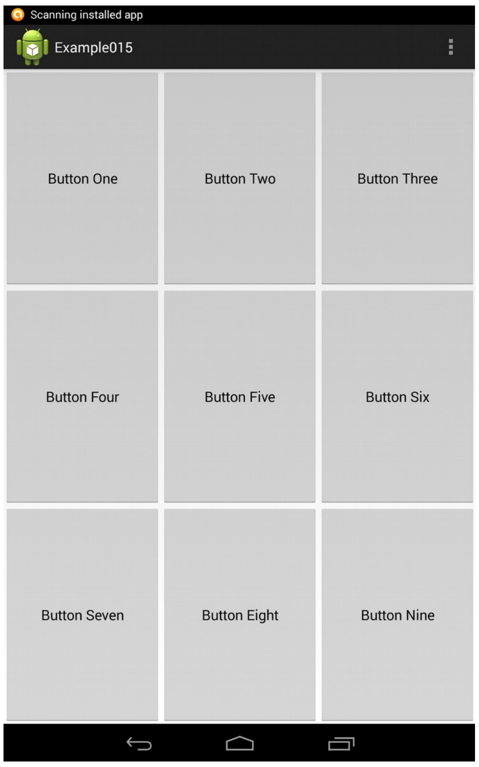
03) replace all XML in activity_main.xml with the following XML (good place to compile and run code)

```
<TableLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>

      <TableRow
            android:layout_height="0dp"
            android:layout_weight="1"
      >
          <Button
            android:text="@string/button_one"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
          />

          <Button
            android:text="@string/button_two"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
          />

          <Button
            android:text="@string/button_three"
            android:layout_width="0dp"
```

```
        android:layout_height="match_parent"
        android:layout_weight="1"
      />

    </TableRow>

    <TableRow
        android:layout_height="0dp"
        android:layout_weight="1"
    >
      <Button
        android:text="@string/button_four"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
      />

      <Button
        android:text="@string/button_five"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
      />

      <Button
        android:text="@string/button_six"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
      />

    </TableRow>

    <TableRow
        android:layout_height="0dp"
        android:layout_weight="1"
    >
      <Button
        android:text="@string/button_seven"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
      />

      <Button
        android:text="@string/button_eight"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
      />

      <Button
        android:text="@string/button_nine"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
      />
    </TableRow>

</TableLayout>
```

Explaination:

- This may seem like a lot of code to begin with but there is plenty of repetition in here. As before to define a TableLayout you must use the <TableLayout></TableLayout> tag pair. If this is the root of a activity then it must have the xmlns:android tag and also take up the space of the entire activity.
- Table layouts are fairly similar to tables in HTML. You have to define the beginning and end of each table row. This gives you versatility in that you can have different numbers of elements on each row and you can assign different weights to each row as necessary. The price you pay for this versatility is that you must write more code.
- TableRows are defined by the <TableRow></TableRow> tag pair
- If you wish to use the weighting system for table rows then all table rows must have their layout_height set to 0 and a layout_weight must be assigned. In this case we want all rows to recieve the same height thus they all have a weight of one.
- Inside the TableRows we define our elements as normal in this case we define three buttons for each row. Note that the width has a size of zero and that a weight is set to determine the amount of horizontal space that each button will recieve. In this case all buttons are set to have a horizontal weight of 1 such that they will all recieve equal horizontal space. However the height is set to match_parent with states that each button will take the full vertical space provided by the table row
- running this code will yield the screenshot below


Scanning installed app
Example015
Button One
Button Two
Button Three
Button Four
Button Five
Button Six
Button Seven
Button Eight
Button Nine

Example016: introducing the merge tag as a way of splitting up layouts

In the apps you have seen thus far we have used a single XML file to define a full layout of an activity. If you find this unmanageable or that you find that you are using common bits of layout between one or more activities then you may want to split your XML layouts into multiple files and take advantage of the include and merge tags. In this example we will recreate the layout of Example014 by splitting out the horizontal layout into a separate file.

01) start with a new android project

02) add the following strings in strings.xml

```
    <string name="button_one">Button One</string>
    <string name="button_two">Button Two</string>
    <string name="button_three">Button Three</string>
    <string name="button_four">Button Four</string>
```

03) create a new layout file in res/layout/ called merge_layout.xml and give it the following XML

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

        <Button
            android:text="@string/button_two"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
        />

        <Button
            android:text="@string/button_three"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
        />

    </LinearLayout>
</merge>
```

Explaination:

- Here we have split out the horizontal linear layout into a seperate file. The only additions that we have to make is to enclose this layout in a pair of <merge></merge> tags. Part of the reason for this is to ensure that there will not be multiple definitions of the xmlns:android attribute as this will be contained in the merge tag which simply indicates that this layout information should be merged in wherever it is included. The final layout will have all merge tags stripped and removed.

04) replace the XML in activity_main.xml with the following XML (good time to compile and run)

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

    <Button
      android:text="@string/button_one"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
    />

    <include layout="@layout/merge_layout" />

    <Button
      android:text="@string/button_four"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
    />

</LinearLayout>
```

Explaination:

- Here we have the linear layout as before that you saw in Example014. Note that as the horizontal linear layout has been split out into a seperate file we have put an <include/> tag in its place. This tells android that it must load a layout file from elsewhere and merge its contents inplace. When the final layout file is produced by android all <include/> tags will be stripped and removed.
    - Include tags only understand a single attribute called layout. This accepts an identifier for a layout XML file as its parameter. All layouts are given unique ids that are stored in the layout section of the R class and can be found under @layout. The name of the identifier is generated by searching for the file in res/layout and stripping ".xml" from the file name.
- this will produce the same layout as you saw in Example014

Example017: Introducing the Button View and its associated event handling

Up to this point we have only introduced different types of resources and the different forms of layouts that can be applied to group, place and size views. We have yet to do any event handling and reaction to user input. In the first example of this we will take the Button class the we have been using for the last while and react to clicks of that button. However, there are two methods that can be used to handle events from the button. It is entirely up to yourself which method you use. The second method can be used in all cases but the first method can be a convenient shortcut if you know a layout will only be attached to a Activity. If the layout is to be attached to a Fragment (which will be covered later) you must use the second method. Both methods will produce the same result but it is useful to recognise both.

01) start with a new android project

02) replace the XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv_display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="30sp"
      android:text="@string/tv_start"
    />

    <Button
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/btn_text"
      android:onClick="buttonPress"
    />
</LinearLayout>
```

Explaination:

- A standard vertical linear layout with a textview and a button. Note that for the TextView we have specified android:textSize which gives us full control of the size of the text that is displayed. Note that we provide a value of 30sp here. Which is 30 scale-independant pixels. This is basically display independant pixels with a scale factor added to reflect the android system's global text size setting.
- We use sp such that our application will display text in a similar size to the user's preference and to fit in with the rest of their android system.

- We have assigned an id to the TextView because we are going to manipulate it in java code

03) In MainActivity.java add the following fields to the MainActivity class

```
private int clicks;
private TextView tv_display;
```

Explaination:

- the first field is to track how many times the button has been clicked. The second field is to give us access to the text view so we can update its contents.

04) in the onCreate() method of MainActivity.java add in the following code after the call to setContentView()

```
// set the initial clicks to zero and get the textview from the layout
clicks = 0;
tv_display = (TextView) findViewById(R.id.tv_display);
```

Explaination:

- sets the initial count of clicks to zero and gets a reference to the textview so it can be updated later.
- At this point we will now divert into both methods of reacting to a button press and updating the textview. The first method we will look at is linking the button to a method by using an XML attribute.

Method 1: 05) add in the following method to the MainActivity class.

```
// method for reacting to a button press from the button
public void buttonPress(View arg0) {
      clicks++;
      tv_display.setText("Button has been clicked " + clicks + " times");
}
```

Explaination:

- This is the method that we will link our button press to. All buttons that are linked through XML must reference a method that is public, returns nothing (void) and accepts a single argument of type View. If you do not adhere to this then your method will not be found by android.
- All this method does is increment the click count and update the display with the new clicks value

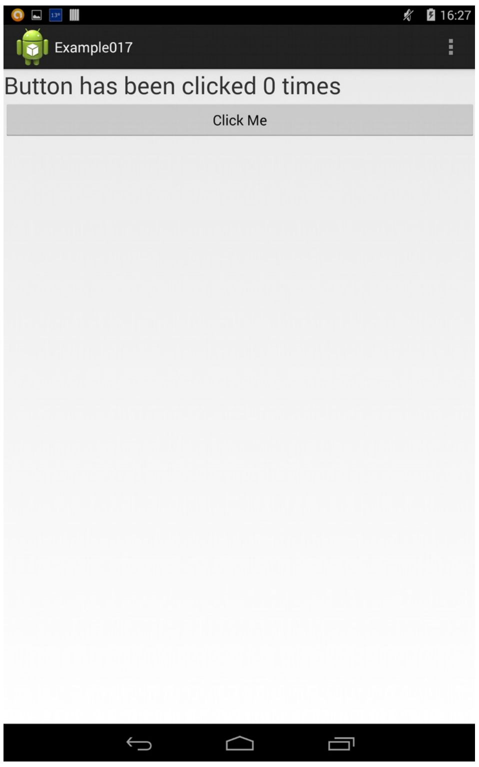
Method 1: 06) add the following attribute to the Button in the layout (good place to compile and run)

```
android:onClick="buttonPress"
```

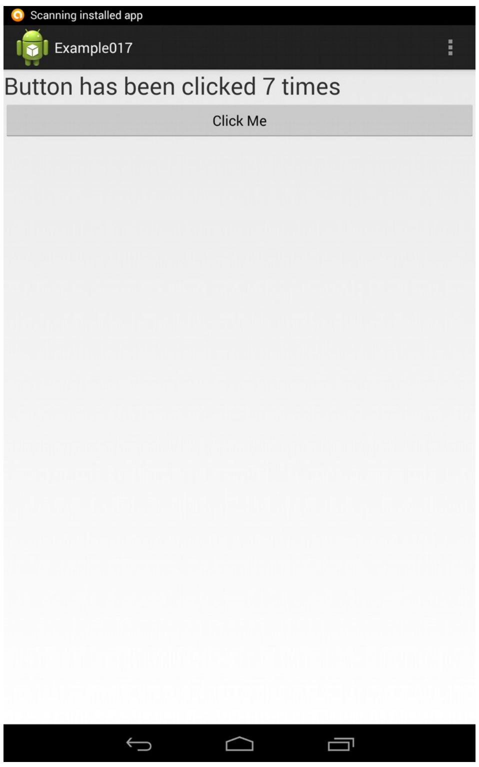
Explaination:

- This will tell android that whatever class this layout is set on it will contain a method called buttonPress that is public, returns nothing and accepts a single argument of type View that will handle the event for this button

- when this code is run you will obtain the screenshots below where the left screenshot is the initial state of the program and the right screenshot is after the button has been clicked seven times
- After this you should revert your code to that of step 04 and we will implement the second method of handling events
- for the rest of this book we will use the second method for handling events but we will also mention approriate XML attributes that can be used to replicate the same effect



Button has been clicked 0 times



Button has been clicked 7 times

Method 2: 05) add the following field to the MainActivity class

```
        private Button btn_clickme;
```

Explaination:

- provides us with a reference that we can use to manipulate the button in java code

Method 2: 06) add the following code after 04 in the onCreate() method of the MainActivity class

```
btn_clickme = (Button) findViewById(R.id.btn_clickme);

// add a click listener to the button
btn_clickme.setOnClickListener(new OnClickListener() {
      public void onClick(View arg0) {
            clicks++;
            tv_display.setText("Button has been clicked " + clicks + " times");
```

```
    }
});
```

Explaination:

- this is how you set a click listener on a button in java code. This is guarenteed to work for all cases.
- To set an onClickListener we must first get the button from the layout by using findViewById(). After this we call the setOnClickListener() method and inside the method call we create a new OnClickListener anonymous class that implements the onClick() method
- All onClickListener classes must implement the onClick() method that you see above and it must have the exact same method signature.
- Like before this will update the text on the text view every time the button is clicked and will yield the same screenshots above.

Example018: introducing the EditText and its event handling

In this example we introduce the EditText which is a control for accepting text entry from a user. In this example we will show how you can specify hints on an edit text to inform the user what they should enter in any given edit text and also how to react to a standard press of the enter key on an edit text. We will use the entered text to update a textview to display the text that the user has entered.

01) start with a new android project

02) add the following strings into strings.xml

```
      <string name="et_hint">Enter text here.</string>
      <string name="tv_initial">You have not entered any text</string>
```

03) replace the XML in activity_main.xml with the following XML (good time to run and compile)

```
<LinearLayout
      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/tv_display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/tv_initial"
      android:textSize="30sp"
    />

    <EditText
      android:id="@+id/et_input"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/et_hint"
      android:inputType="text"
    />

</LinearLayout>
```

Explaination:

- like the previous example we have a vertical linear layout with a text view and an edit text immediately underneath. If you were to run this code right now you would get the first of the screenshots at the end of this example.
- Edit Texts are defined with the <EditText/> tag and usually you will attach an id to these controls in order to get input from them in java

code. The two most important attributes however are the hint and the inputType. The hint is a piece of text that will be displayed in the EditText if the user has not entered any input. A good use for this is to tell the user what you want them to input in a particular EditText. In this case we are asking for general text to be input.

- The second attribute inputType specifies the kind of information that will be entered into the edit text. This does two important functions. First it provides a bit of validation on your input (not complete validation) and it will also choose the appropriate display and keyboard types depending on the type of information. e.g. if you ask for a password it will blank out the characters or if you ask for a number only digits can be inserted.

04) add in the following fields to the MainActivity class

```
// fields for the textview and the edittext
private TextView tv_display;
private EditText et_input;
```

Explaination:

- as before needed to get access to these elements in XML and to use them throughout the class

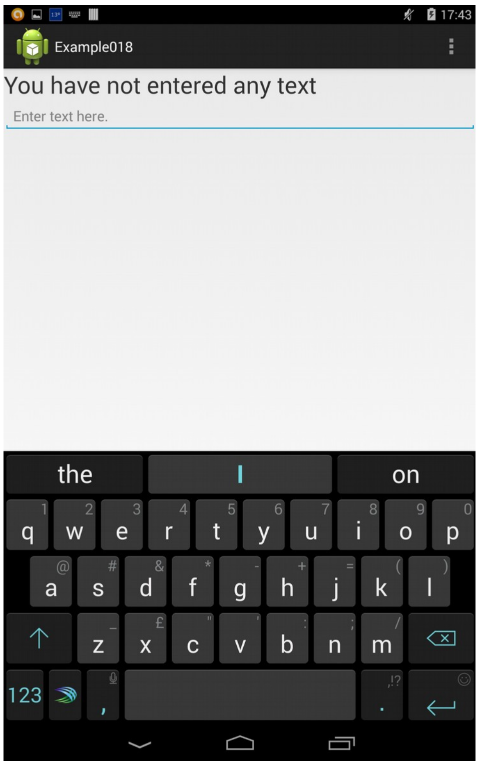
05) add in the following code after the call to setContentView in the MainActivity class (good place to compile and run)

```
// pull the textview and the edittext from the xml
tv_display = (TextView) findViewById(R.id.tv_display);
et_input = (EditText) findViewById(R.id.et_input);

// set an editor action listener on the edit text to respond to input
// events on the return key
et_input.setOnEditorActionListener(new OnEditorActionListener() {
      public boolean onEditorAction(TextView view, int actionid, KeyEvent event)
{
            // if the enter button has been clicked then update the text view
with
            // the text that the user has entered.
            if(actionid == EditorInfo.IME_ACTION_DONE){
                  tv_display.setText("Entered: " + et_input.getText());
                  return true;
            }

            // if this event has not been handled then return false
            return false;
      }
});
```

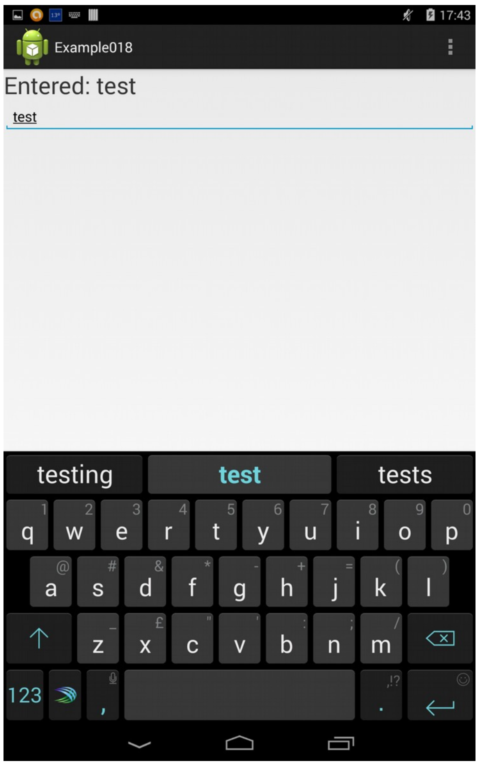
Explaination:

- In the first two lines we are pulling the views from the layout file.
- In the following lines we are adding an OnEditorActionListener to our EditText to listen out for input events on the EditText.

- Every EditorActionListener must implement the onEditorAction method and must provide the exact method signature that you see above. It must be public and return a boolean and it must accept three arguments the first of which is a text view, the second is an integer and the third is a key event.
  - Aside: the method is defined this way to account for both hardware and software keyboards, early devices with hardware keyboards just used keyevents whereas later software keyboard started defining general events that have a better meaning and context to a user action.
- In the if statement we check the value of actionid. This is the action that the user has performed on the edit text. In this case IME_ACTION_DONE states that the user has pressed the enter key on the edit text in question. After hitting the enter key we take the text that was entered and display it in the textview. Which is what you see in the second screenshot.
- Note that we have to return either true or false. These values have specific meaning in android. True from this method tells android that this event has been handled and consumed and thus no further processing is needed on that event. False states that the event has not been handled and android may provide further processing on that event if necessary.
- The general accepted convention is that if you handle and event you should return true. If you do not handle an event you should return false.
- Running this code will net you the screenshots below



You have not entered any text
Enter text here.



Entered: test
test

Example019: introducing the checkbox and its event handling

Another particularly useful widget provided by default in android is that of the checkbox. This is a two state widget that is either on or off and is generally used to switch options quickly. It should only be used for options that exist in a binary state. In this example we will show a simple check box and how it capture events from it.

01) start with a new android project

02) add in the following strings to strings.xml

```
    <string name="tv_off">CheckBox is unchecked</string>
    <string name="tv_on">CheckBox is checked</string>
    <string name="cb_text">Click Me</string>
```

03) add in the following fields to the MainActivity class

```
      // private fields for the checkbox and textview
      private TextView tv_display;
      private CheckBox cb_clickme;
```

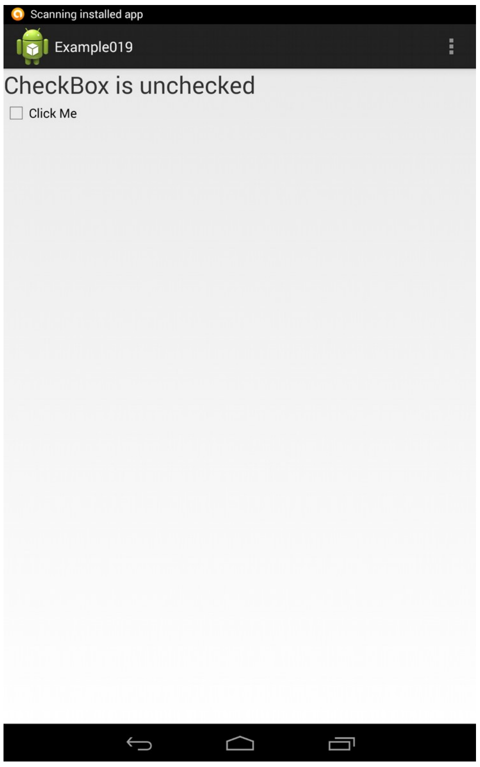
04) add in the following code after the call to setContentView() in the onCreate() method of the MainActivity class (good time to compile and run)

```
// pull the textview and the checkbox from the XML
tv_display = (TextView) findViewById(R.id.tv_display);
cb_clickme = (CheckBox) findViewById(R.id.cb_clickme);

// add an on click listener to the check box
cb_clickme.setOnCheckedChangeListener(new OnCheckedChangeListener() {
      // implementation of on checked change function that is required
      public void onCheckedChanged(CompoundButton button, boolean state) {
            if(state == true)
                  tv_display.setText(R.string.tv_on);
            else
                  tv_display.setText(R.string.tv_off);
      }
});
```

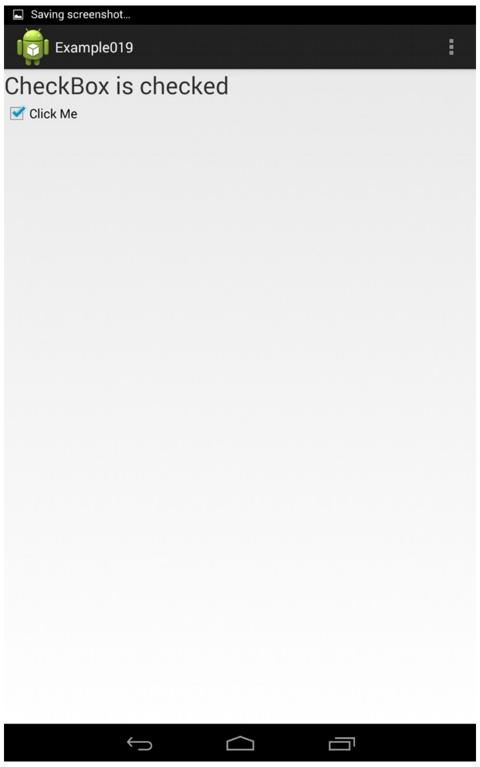
Explaination:

- like before we pull the two view from the XML first so we are capable of manipulating it.
- In the next block of code we are adding an OnCheckedChangeListener to the checkbox which is responsible for capturing any changes in the checked state of the checkbox. This listener requires that you implement the onCheckedChanged method which must have the exact signature that you see above. The first argument is used if you wish to have the same listener attached to multiple checkboxes and provides a means for you to identify which checkbox generated the event

  - generally we will create a listener for each control. From a performance standpoint this is faster as we eliminate costly if-else-if ladders from code. The only downside is that it requires more memory
- In the method we use the state variable to determine what state the checkbox is in. True means that a check is present, false means the check is absent. And we update the text view to reflect this directly but instead of using hardcoded strings we pull them from the XML files (generally recommended for translation purposes)
- running the code at this point will yield you the screenshots below



CheckBox is unchecked



CheckBox is checked

Example020: Introducing the Radio Button and their event handling.

Blarg

01) start with a fresh android project

02) add the following strings into strings.xml

```
    <string name="tv_display_rb1">RadioButton 1 selected</string>
    <string name="tv_display_rb2">RadioButton 2 selected</string>
    <string name="tv_display_rb3">RadioButton 3 selected</string>
    <string name="rb1">Radio Button 1</string>
    <string name="rb2">Radio Button 2</string>
    <string name="rb3">Radio Button 3</string>
```

03) replace the XML in activity_main.xml with the following XML (good place to compile and run)

```
<LinearLayout
      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/tv_display"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/tv_display_rb1"
        android:textSize="30sp"
      />

    <RadioGroup
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    >

        <RadioButton
            android:id="@+id/rb1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/rb1"
            android:checked="true"
        />

        <RadioButton
            android:id="@+id/rb2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/rb2"
        />

        <RadioButton
            android:id="@+id/rb3"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/rb3"
        />

    </RadioGroup>
</LinearLayout>
```

Explaination:

- as before we are using a vertical linear layout to hold all of our widgets. Note that in order to use RadioButtons they must be assigned to a RadioGroup. The RadioGroup will ensure that if one RadioButton is clicked then the last RadioButton that was selected by the user will be deselected. This is to ensure mutual exclusion (i.e. force the user to make a single choice)
- RadioGroups although they appear invisible are required to have a width and a height property. Usually these will be set to wrap_content but as this is embedded in a vertical linear layout we can take the maximum width of the parent layout.
- RadioButtons are declared with the <RadioButton /> class and have similar setup to normal buttons in that we must provide dimensions, text (which appears to the right of the button) and an id for event handling.
- Usually it is a good idea to default one of the radio buttons to be selected we do this by adding the android:checked property to one of the radio buttons and setting it to true. This property can also be used with checkboxes as well to give it a check when it is first initialised.

04) add the following fields into the MainActivity class

```
      // fields the the text view and the radiobuttons
      private TextView tv_display;
      private RadioButton rb1, rb2, rb3;
```

05) add the following code into the onCreate() method of the MainActivity class (good place to compile and run)

```
// get the widgets from the XML
tv_display = (TextView) findViewById(R.id.tv_display);
rb1 = (RadioButton) findViewById(R.id.rb1);
rb2 = (RadioButton) findViewById(R.id.rb2);
rb3 = (RadioButton) findViewById(R.id.rb3);

// attach a listener to the first radio button
rb1.setOnClickListener(new OnClickListener() {
      // method that must be implemented in an onClick listener
      public void onClick(View arg0) {
            // change the text to state that radio button one was clicked
            tv_display.setText(R.string.tv_display_rb1);
      }
});
```

Explaination:

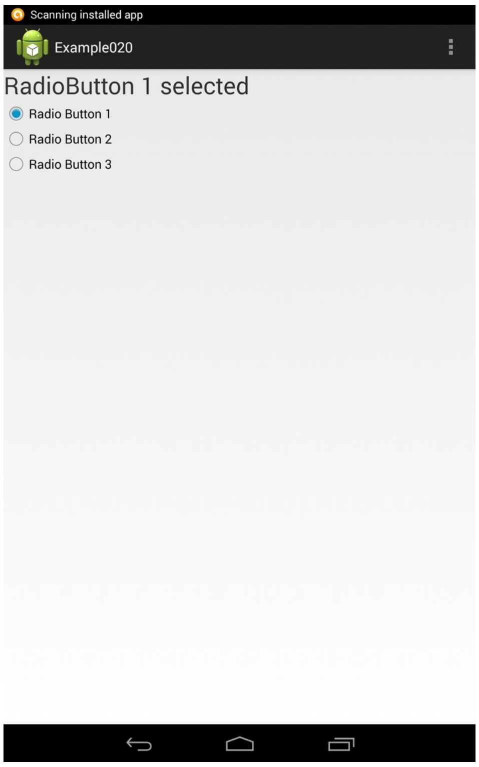
- As before the first block of code is used to get all of the controls from the XML layout into Java code.
- The second block shows an example of adding a listener to a radiobutton. For RadioButtons we must use the OnClickListener (same as the Button class) to listen for click events. Although the onClick() method provides a View we have no need to use it here as all we are doing is modifying the text that is displayed at the top of the layout.

06) add the following code into the onCreate() method of the MainActivity class immediately after the code for 05 above (good place to compile and run)

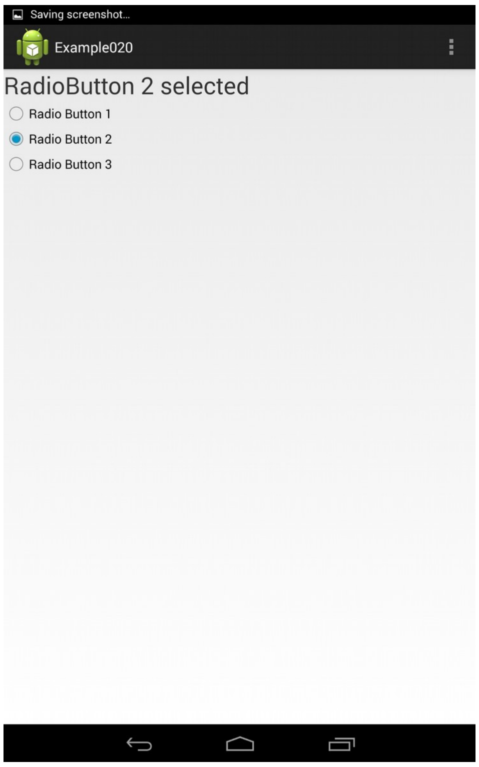
```
// attach a listener to the second radio button
rb2.setOnClickListener(new OnClickListener() {
      // method that must be implemented in an onClick listener
      public void onClick(View arg0) {
            // change the text to state that radio button one was clicked
            tv_display.setText(R.string.tv_display_rb2);
      }
});

// attach a listener to the third radio button
rb3.setOnClickListener(new OnClickListener() {
      // method that must be implemented in an onClick listener
      public void onClick(View arg0) {
            // change the text to state that radio button one was clicked
            tv_display.setText(R.string.tv_display_rb3);
      }
});
```

Explaination:

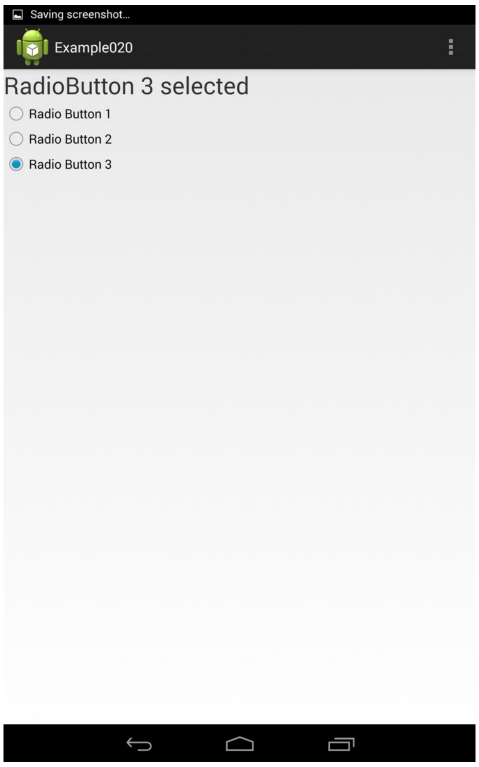
- Similar event handlers to 05 above but for the other RadioButtons
- running this code will yield all three screenshots below showing each radio button in its checked state



RadioButton 1 selected
- [✓ Radio Button 1]
- Radio Button 2
- Radio Button 3



RadioButton 2 selected
- Radio Button 1
- [✓ Radio Button 2]
- Radio Button 3



RadioButton 3 selected
- Radio Button 1
- Radio Button 2
- [✓ Radio Button 3]

Example021: Introducing the Spinner and its event handling.

Most applications will use some form of dropdown box to give a user a selection of choices. When the User clicks on the box a dropdown menu is presented with a list of choices for the user to select from. When the user selects a choice the drop down menu will disappear and will display the choice that the user has selected. Other names for this type of widget include Comboboxes, choiceboxes etc. In Android they are called spinners. In this example we will show how to construct a spinner for a list of strings and also their event handling. It is possible to use custom layouts and views with the spinner (similar to a ListView) but we will wrap that discussion inside the ListView later.

01) start with a new Android

02) add in the following strings to strings.xml

```
    <string name="tv_display_init">Spinner value selected: This</string>
    <string-array name="spinner_strings">
        <item>This</item>
        <item>is</item>
        <item>a</item>
        <item>list</item>
        <item>of</item>
        <item>spinner</item>
        <item>items</item>
    </string-array>
```

Explaination:

- Spinners for strings can accept string-array resources. For simplicity we will use a string array to set the values in our spinner

03) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv_display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/tv_display_init"
      android:textSize="30sp"
    />

    <Spinner
      android:id="@+id/spinner"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
    />
```

```
</LinearLayout>
```

Explaination:

- all spinners are declared with the <Spinner /> tag. Spinners must be initialised in Java code however in the XML you must provide an id for the spinner as well as layout parameters.

04) add in the following private fields to the MainActivity class

```
// private fields to give us access to the spinner and the text view
private TextView tv_display;
private Spinner spinner;
```

05) add in the following code at the end of the onCreate() method of the MainActivity class (good time to compile and run)

```
// pull the TextView and the Spinner from the xml
tv_display = (TextView) findViewById(R.id.tv_display);
spinner = (Spinner) findViewById(R.id.spinner);

// we have to pull the string array from the xml resources and link it up
// to an array adapter such that it can be used with the spinner
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
            this,
            R.array.spinner_strings,
            android.R.layout.simple_spinner_item);

// specify the layout that will be used on the list of choices when the spinner
// is clicked
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

// set the adapter on the spinner
spinner.setAdapter(adapter);
```

Explaination:

- The first two lines are standard pulling of views from XML. The third line generates an ArrayAdapter for the for the Spinner. By default Spinners interpret strings as CharSequence types (Android's own representation of strings). On the right hand side of the assignment statement we are asking the ArrayAdapter to take a string array from our resources and generate an ArrayAdapter out of it.
  - The first argument specifies the context. The majority of time the keyword this is used here.
  - The second argument specifies which string array resource will be used to provide the values for the spinner.
  - The third argument specifies a layout file that will provide the layout for an individual item in the list generated by the spinner. Android provides a good default for the strings in a spinner thus you should use it when possible

- The next line then sets the overall style of the dropdown list that the spinner generates. Again Android provides a good layout for spinners of strings which you should use whenever possible.
- The final line then links the adapter to the spinner. If there is any items added or removed from the array adapter it will then immedately notify the spinner to update itself to display the new list of items in the spinner.
- Running this code now will give you a working spinner however you will not be able to handle events from it.

06) add in the following code at the end of the onCreate() method of the MainActivity class (good time to compile and run)

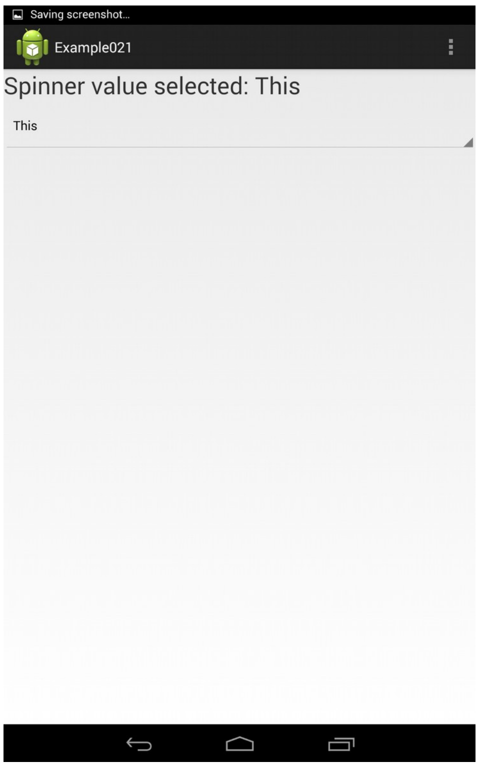
```
// set a listener on the spinner
spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
      // overridden method that will be called when an item has been
      // selected in the spinner
      public void onItemSelected(AdapterView<?> parent, View view, int pos, long
id) {
            // here we will query what item was selected and then set
            // its text on the textview
            tv_display.setText("Spinner value selected: " +
parent.getItemAtPosition(pos));
      }

      // overridden method that will be called when no item has been
      // selected in the spinner
      public void onNothingSelected(AdapterView<?> parent) {
            // here is where you would react if no item was selected
      }
});
```

Explaination:

- To react to item selection in the spinner you must provide an implementation of the OnItemSelectedListener interface. Generally we will do this through the use of an anonymous class. You are required to provide implementations for two methods onItemSelected() for when the user selects an item in the spinner and onNothingSelected() for when the user has not selected an item. In most cases you will not need to do anything with onNothingSelected()
- onItemSelected() takes four arguments.
  - The first is the spinner that generated the event. This is particularly useful if you want to query the exact item that the user has selected. In this case we will get the text associated with the item and will display it on the TextView.
  - The second argument is the actual view that corresponds to this item. This is provided as a means of modifying the view in response to the selection. Examples where this could be used is to change the background colour of the selected item in the list.

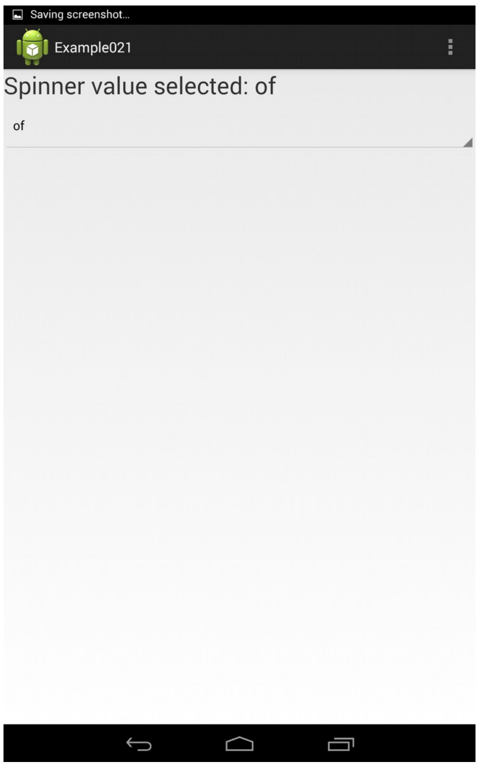
   - The third argument provides the position of the item in the adapter. While the fourth arguement provides the id of the item. When array types are used these two arguments will have the same value however for list and hashmap types ids should be provided for better identification of items.

- Running this code will get you the screenshot on the right after you have selected an item in the list



Spinner value selected: This

This



Spinner value selected: of

of

Example022: introducing the number picker and its event handling.

Number pickers are simple controls that enable users to pick from a well defined small range of integer values. If you have such a range then it is highly advisable to use a NumberPicker as it will provide you with automatic validation of values. In this example we will show how to initialise a number picker and also how to listen for changes in the value of a number picker. Note that to initialise the number picker fully you must do part of the initialisation in XML and the rest in Java code.

01) start with a new Android project.

02) add in the following string to strings.xml

```
<string name="tv_display_init">Current value selected is: 0</string>
```

03) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv_display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/tv_display_init"
      android:textSize="30sp"
    />

    <NumberPicker
      android:id="@+id/np_picker"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
    />

</LinearLayout>
```

Explaination:

- It is generally recommended that when initialising a number picker that you set the dimensions of the width and height to wrap content. Number pickers realistically do not need more space as it will just be wasted.
- Also note that if you try to set a maximum and minimum value here that it will not be possible and you will be met will multiple compiler errors.

04) add in the following private fields to the MainActivity class

```
      // private fields of the class
      private TextView tv_display;
```

```
	private NumberPicker np_picker;
```

05) add the following code to the end of the onCreate() method of the MainActivity class (good time to compile and run)

```
	// pull the number picker and the textview from the xml
	tv_display = (TextView) findViewById(R.id.tv_display);
	np_picker = (NumberPicker) findViewById(R.id.np_picker);

	// set a minimum and a maximum value on the number picker
	np_picker.setMinValue(0);
	np_picker.setMaxValue(100);
```

Explaination:

- here is the rest of the initialisation of the number picker. You are required to call setMinValue() and setMaxValue() which define an integer range. Failure to do so will result in a number picker that will not handle any events nor will be able to move up or down in value.
- Running the code at this point will get you the first screenshot on the left. You can move the number picker about however you will not see any reaction in the textview

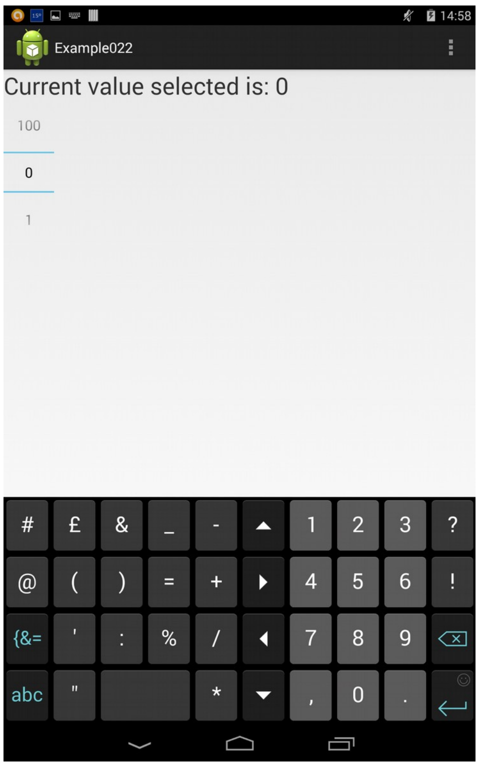
06) add the following code to the end of the onCreate() method of the MainActivity class (good time to compile and run)

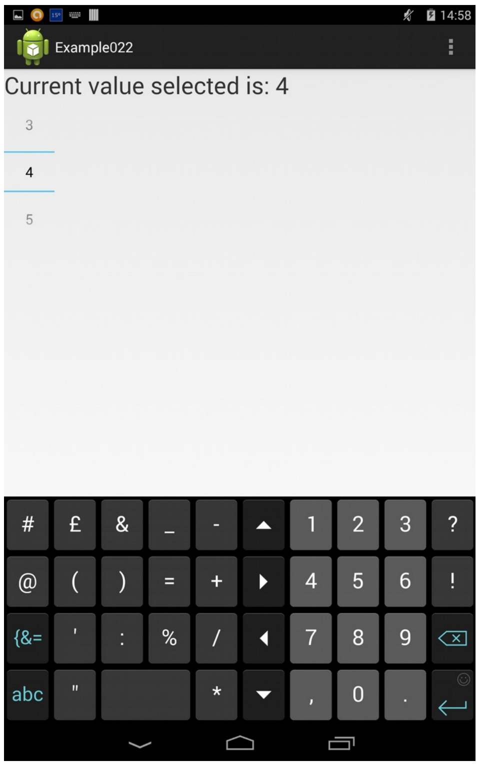
```
// add in a listener to the number picker to listen for changes in numbers
np_picker.setOnValueChangedListener(new OnValueChangeListener() {
	// overridden method that we must provide for when the value changes
	public void onValueChange(NumberPicker picker, int old_val, int new_val) {
		// take the value and apply it to the text view
		tv_display.setText("Current value selected is: " + new_val);
	}
});
```

Explaination:

- If you wish to listen for a change in the value of the number picker you need to implement the OnValueChangeListener that is specific to the NumberPicker class.
- You are required by this interface to implement the onValueChange method that takes three arguments
    - The first is the NumberPicker that the event has come from. There are two reasons for providing this value. The first is that the same listener is being shared amongst multiple number pickers (good for saving memory but leads to inefficient performance) and the second is that you may wish to make changes to the number picker depending on the value that was selected.
    - The second argument is the old value that was selected by the user
    - The third argument is the new value that was selected by the user

- Running this code will get you the screenshot on the right where the user has changed the value of the number picker has been changed and consequently the textview has been updated because the listener has issued a change to the TextView.





Example023: Introducing the CalendarView and its event handling

In this example we introduce a widget that will enable your users to pick dates from a calendar. We will also show how to handle events from this view. Take note of the few caveats that are mentioned in the code below. If you do not take these into account then your CalendarView will not display properly.

01) start a new android project

02) add the following line to strings.xml

```
<string name="tv_display_init">No date has been selected</string>
```

03) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv_display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/tv_display_init"
      android:textSize="30sp"
    />

    <CalendarView
      android:id="@+id/cv_calender"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:maxDate="12/31/2014"
      android:minDate="01/01/2014"
    />
</LinearLayout>
```

Explaination:

- CalendarViews are delcared with the <CalendarView /> tag. At the very minimum you must provide the layout parameters for the view. Anything else is optional.
- Note that is it very important to set the height of the calendar view to be match_parent. If you set this to wrap_content instead you will only see the header of the calendar view which mentions the current month and has letters for the individual days of the week. You will not see any dates displayed. If you need to restrict the space that the calendar view uses you have two options
    - set a defined size on the calendar view by giving direct pixel width and height to the view

  - or embedded the calendar view in a frame layout and set the calendar view to take the max width and height of the frame layout.

- By default the CalendarView will select and display the current date upon initialisation and will allow selection of any date in any year. If however you wish to restrict the calendar to a defined period of time you can set the minDate and maxDate attributes. Note that as this is a US toolkit dates are in the format MM/DD/YYYY even though the rest of the world uses DD/MM/YYYY.

04) add the following fields to the MainActivity class

```
      // private fields for the class
      private TextView tv_display;
      private CalendarView cv_calender;
```

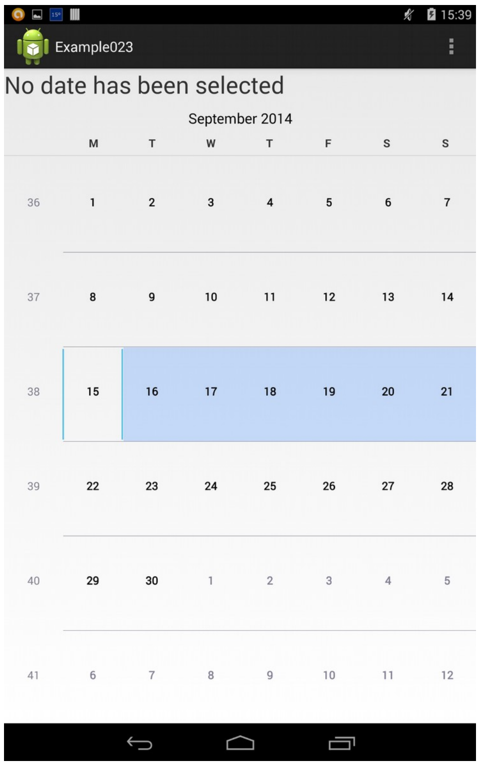
05) add the following code to the end of the onCreate() method of the MainActivity class. (good time to compile and run)

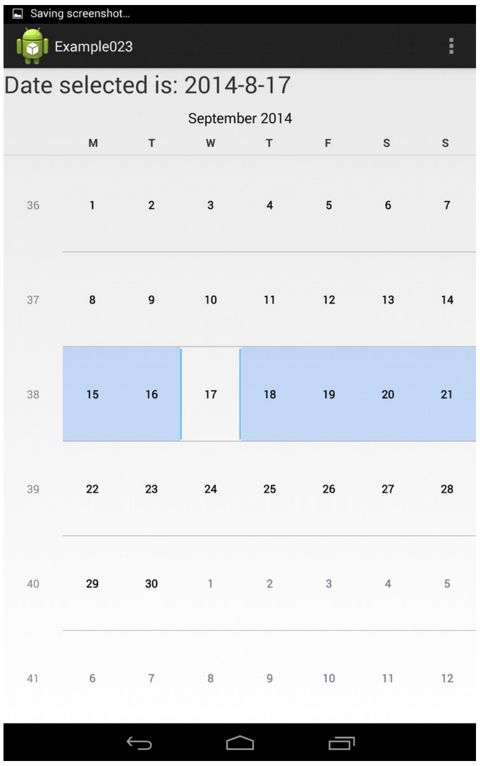
```
// pull the views from the XML
tv_display = (TextView) findViewById(R.id.tv_display);
cv_calender = (CalendarView) findViewById(R.id.cv_calender);

// set a listener on the calender view to listen for changes in the date
cv_calender.setOnDateChangeListener(new OnDateChangeListener() {
      // overridden method that will be called whenever the user changes the
date
      public void onSelectedDayChange(CalendarView view, int year, int month,
int day) {
            // display the new date on the text view
            tv_display.setText("Date selected is: " + year + "-" + month + "-" +
day);
      }
});
```

Explaination:

- This is how you set a listener on the CalendarView. You must implement the OnDateChangeListener that is specific to the CalendarView. This interface requires that you implement the onSelectedDayChange() method which will be called every time a user clicks on a date in the CalendarView. The method requires four arguments the first of which is the CalendarView that generated the event (again useful if multiple CalendarViews share the same listener) while the last three arguments provide the full date that the user selected.
- In this implementation we update the text view to reflect the date that the user has chosen.
- Running this code will get you the two screenshots below which show the app in its initial state (left) and where the user has selected a date (right)

- pay close attention to the value of the month as it has a range of [0,11] you must add 1 to this value to get the correct month





Example024: introducing the ListView and its event handling.

On of the most useful of the default widget set that is provided by the android toolkit is that of the ListView. The reason it is particularly useful is that the layout of each item in the list is capable of being modified and it can be adapted to work with any kind of class be it a standard class or user defined. In this example we will show the standard list view that works with the basic String class. Here we will show how to add in items to a list and also how to detect short and long clicks on items in the list. In the next example we will look at customising the layout of each item in the list.

01) start with a new android project.

02) add the following strings into strings.xml

```
    <string name="tv_display_init">Nothing selected</string>
    <string name="et_hint">Enter text here</string>
```

03) replace all XML in activity_main.xml with the following XML

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

    <TextView
      android:id="@+id/tv_display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:text="@string/tv_display_init"
      android:textSize="30sp"
    />

          <ListView
            android:id="@+id/lv_mainlist"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_below="@id/tv_display"
            android:layout_above="@+id/et_new_strings"
          />

    <EditText
      android:id="@id/et_new_strings"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:inputType="text"
      android:hint="@string/et_hint"
    />

</RelativeLayout>
```

Explaination:

- here we have used a relative layout instead of a LinearLayout. The reason for this is that we want our textview to appear on the top and our edit text to appear on bottom with the list view in between.
- With the RelativeLayout if you use match_parent for the width and height but specify a layout_above of the text view and layout_below of the edittext parameters the ListView will expand to take all available space in the middle of these two components.
- Also note how the identifier for the edit text is first defined in the list view. This is because XML compilation is done in a single pass, meaning that your ids must be defined before you use them. If you were to declare the id in the EditText yet still reference it in the ListView you would get compiler errors.

04) Add the following fields into the MainActivity class

```
// private fields of the class
private TextView tv_display;
private ListView lv_mainlist;
private EditText et_new_strings;
private ArrayList<String> al_strings;
private ArrayAdapter<String> aa_strings;
```

Explaination:

- To make a listview work we need three components. We need a ListView object that is responsible for displaying the list. We also need a datastructure for holding our data for the ListView (in this case al_strings) and we also require an ArrayAdapter that will map the data to the view (in this case aa_strings)

05) add the following code to the end of the onCreate() method of the MainActivity class (good place to compile and run)

```
// pull the list view and the edit text from the xml
tv_display = (TextView) findViewById(R.id.tv_display);
lv_mainlist = (ListView) findViewById(R.id.lv_mainlist);
et_new_strings = (EditText) findViewById(R.id.et_new_strings);

// generate an array list with some simple strings
al_strings = new ArrayList<String>();
al_strings.add("first string");
al_strings.add("second string");
al_strings.add("third string");

// create an array adapter for al_strings and set it on the listview
aa_strings = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
al_strings);
lv_mainlist.setAdapter(aa_strings);
```

Explaination:

- The first block of code obtains all of our views from XML.

- The second block of code initialises our data storage for our ListView. In this case we use an ArrayList which is a List type datastructure that is understood by the default ArrayAdapter class. Once initialised we add some data to the ArrayList.
- Finally in the last block of code we connect the adapter to the ArrayList and also to the ListView. The first line creates the adapter and the link to the ArrayList. The constructor for the ArrayList takes three arguments
  - The first is the context. Most of the time this will be the keyword this
  - The second is the layout used to generate each item in the list. For displaying strings it is recommended that you use simple_list_item_1. In the next example we will use our own layout here.
  - Finally the last argument is the list that the adapter should map to the ListView
- The last line then establishes the connection between the adapter and the ListView
- If you run this code as is you will get the initial screenshot on the left where no event handling has been implemented

06) add in the following code to the end of the onCreate() method of the MainActivity class

```
// add in a listener for the edit text to create new items in our list view
et_new_strings.setOnEditorActionListener(new OnEditorActionListener() {
	public boolean onEditorAction(TextView view, int actionid, KeyEvent event)
{
		// if the user is done entering in a new string then add it to
		// the array list. this then notifies the adapter that the data has
		// changed and that the list view needs to be updated
		if(actionid == EditorInfo.IME_ACTION_DONE) {
			al_strings.add(et_new_strings.getText().toString());
			aa_strings.notifyDataSetChanged();
			tv_display.setText("added item: " + et_new_strings.getText());
			return true;
		}

		// if we get to this point then the event has not been handled thus
		// return false
		return false;
	}
});
```

Explaination:

- Here we are adding a listener to the EditText. As you have seen this before we will only focus on the code inside the if statement.
- The first line inside the if statement takes the string that was entered by the user into the EditText and adds it as a new item on the end of the ArrayList. The second line then tells the ArrayAdapter that the data inside the ArrayList has changed. This will prompt the ArrayAdapter to update

the ListView display such that the new data is displayed to the user.

07) add in the following code to the end of the onCreate() method of the MainActivity class

```
// add in a listener that listens for short clicks on our list items
lv_mainlist.setOnItemClickListener(new OnItemClickListener() {
      // overridden method that we must implement to get access to short clicks
      public void onItemClick(AdapterView<?> adapterview, View view, int pos, 
long id) {
            // update the text view to state that a short click was made here
            tv_display.setText("item " + lv_mainlist.getItemAtPosition(pos) + " 
selected");
      }
});
```

Explaination:

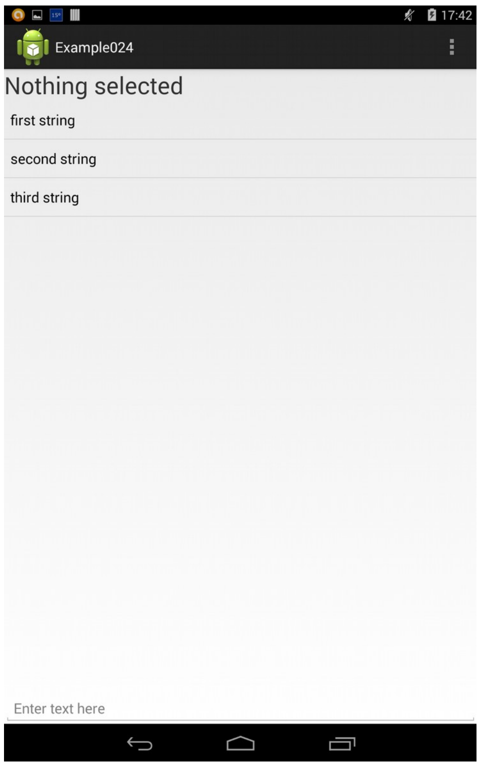
- This is how we detect short clicks on items in the list. Hopefully you can see that this follows a similar format to the selected listener on the spinner. In this case you must have a class that implements the OnItemClickListener interface.
- Any class that implements this interface must provide an implementation of the onItemClick method that you see above.
  - The first argument is the ListView that the event is coming from.
  - The second argument is the View that generated the event in the list in case that you wish to modify it in any way or retrieve data from it.
  - While the last arguments you can use to identify which item the event came from. Generally because we are using a list type generally the position will be good enough to identify which item the event came from. If you are using anything other than an ordered list (a hashmap for example) you may need to use the id to identify the item that generated the event.

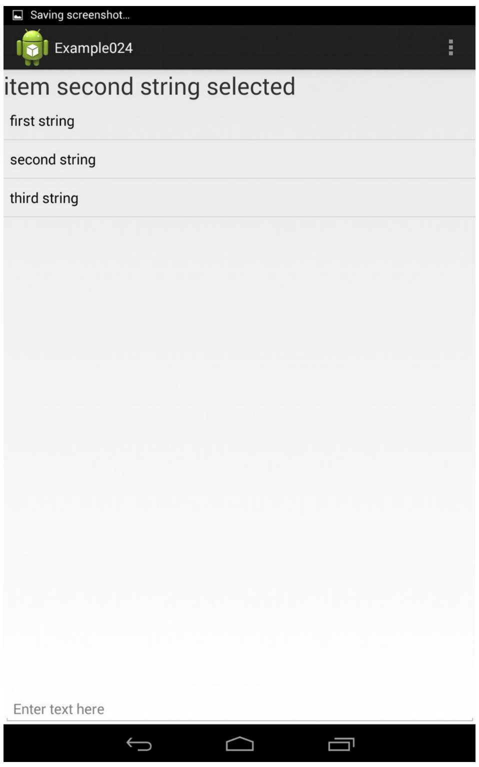
08) add in the following code to the end of the onCreate() method of the MainActivity class

```
// add in a listener that listens for long clicks on our list items
lv_mainlist.setOnItemLongClickListener(new OnItemLongClickListener() {
      // overridden method that we must implement to get access to long clicks
      public boolean onItemLongClick(AdapterView<?> adapterview, View view, int 
pos, long id) {
            // update the display with what we have just long clicked
            tv_display.setText("item " + lv_mainlist.getItemAtPosition(pos) + " 
long clicked");

            // as we are going to change the textview anyway we can 
automatically
            // return true;
            return true;

      }
});
```

Explaination:

- Listener for the long clicks. A long click in the list view is defined as the user holding down on the list item for a few seconds or more until the Android device gives a small vibration to denote that the click has been registered.
- In order to listen for long clicks on a list you must have a class that implements the OnItemLongClickListener interface. Your class should implement the onItemLongClick method which has the same arguments as the onItemClick() method that you see in 07 above.
    - This method is required to return a boolean however, true states that you have handled the event whereas false states that the application should handle the event instead as it has not been consumed by this listener.
- Running this code will get you the screenshot you see on the right where the user is interacting with the listview by adding a few items and clicking a few items.


Nothing selected
first string
second string
third string


item second string selected
first string
second string
third string

Example025: Writing a custom adapter and a custom item for the ListView.

In the previous example we saw the standard list view when used with strings. However in this example we will show the full power of the ListView by getting it to display our custom data in the list view in a custom format. Of course in order to take advantage of this power you are required to write a far amount of code. In this case we will define an XML custom layout for describing how all items fit together in the view. We will also create a custom array adapter to convert items in our array list into views for the ListView.

01) start a new android project

02) add the following strings into strings.xml

```
    <string name="tv_display_init">Nothing selected</string>
    <string name="et_hint">Enter text here</string>
```

03) replace all XML in activity_main.xml with the following XML

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

    <TextView
      android:id="@+id/tv_display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:text="@string/tv_display_init"
      android:textSize="30sp"
    />

          <ListView
            android:id="@+id/lv_mainlist"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_below="@id/tv_display"
            android:layout_above="@+id/et_new_strings"
          />

    <EditText
      android:id="@id/et_new_strings"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_alignParentBottom="true"
      android:inputType="text"
      android:hint="@string/et_hint"
    />

</RelativeLayout>
```

Explaination:

- same layout as the previous example.

04) create a new layout file in res/layout called custom_item_layout.xml and give it the following XML

```
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
>
      <ImageView
            android:id="@+id/iv_image"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:contentDescription="@string/hello_world"
      />
      <TextView
            android:id="@+id/tv_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@id/iv_image"
            android:layout_toEndOf="@id/iv_image"
      />
      <TextView
            android:id="@+id/tv_date"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toRightOf="@id/iv_image"
            android:layout_toEndOf="@id/iv_image"
            android:layout_alignParentBottom="true"
      />

</RelativeLayout>
```

Explaination:

- Here we are creating the layout that will be used for every item in our list view. This layout comprises three items, an ImageView and two TextViews
  - The Image is set to have a square size of 50 density independant pixels. 50 density independant pixels is 50 pixels on a 160 DPI device. If the image is displayed on a 320 DPI device android will use double the pixels to 100 make the image take the same physical space on the 320 DPI device. If however it was displayed on an 80 DPI device then half the pixels (25) will be used to display the image in the same physcial space. It is recommended that you use density independant pixels wherever possible.
    - The image is also set to appear at the very left of the layout. Note that there are two settings for this: alignParentLeft is how it is defined for API 14 (Ice Cream Sandwich) devices whereas alignParentStart is how it is defined for API 17 (Jelly Bean 4.2) devices, we put both in here to ensure forward compatibility.

- The Textviews are set to take as little space as possible to display. Ideally we should take as little space as possible to display as many items in the list. However, it should not be made too small as to render it unreadable to the user.
    - One of the TextViews is set to appear at the top of the layout and the other at the bottom. Note how we have two attributes in the first Textview called toRightOf and toEndOf. This is for similar reasons as alignParentLeft and alignParentStart for the imageview. We do this for forward compatibility reasons.

05) create a source file called CustomItem.java and give it the following code

```
package com.example.example025;

// simple class that contains a name and a date

// imports
import java.text.SimpleDateFormat;
import java.util.Date;

// class definition
public class CustomItem {
      // constructor for the class
      CustomItem(String name, long time) {
            // take a copy of the name and convert the time into a simple date
format string
            this.name = name;
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
            Date d = new Date(time);
            date = sdf.format(d);
      }

      // getter methods for both fields
      public String getName() { return name; }
      public String getDate() { return date; }

      // private fields of the class
      private String name;
      private String date;
}
```

Explaination:

- simple class that contains a name and a date. We pass in a long to the constructor as we will use the current system time to create a string representation of the date for us.

06) create a new source file called CustomArrayAdapter.java and give it the following shell code

```
package com.example.example025;

// class that implements an array adapter for our custom list items

// imports
```

```
import java.util.ArrayList;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

// class definition
public class CustomArrayAdapter extends BaseAdapter {
      // constructor for the class that takes in references to a context and
      // an array list
      public CustomArrayAdapter(Context c, ArrayList<CustomItem> al) {
            context = c;
            al_items = al;
      }

      // overridden method that will construct a View for the listview out of
the
      // item at the given position
      public View getView(int position, View convert_view, ViewGroup parent) {


      }

      // overridden method that will tell the listview how many items of data
there is
      // to be displayed
      public int getCount() { return al_items.size(); }

      // returns the rowid of the item at the given position. Given that we are
using an
      // array list the rowid will be equal to the index of the item
      public long getItemId(int position) { return position; }

      // overridden method that will return the item at the given position in
the list
      public Object getItem(int position) { return al_items.get(position); }

}
```

Explaination:

- This is an extremly important class when defining a custom list view. The custom array adapter is responsible for transforming your custom datatypes into Views that can be inserted into the list view. Every custom adapter should extend the BaseAdapter class
- You are required to provide 4 methods whenever you subclass the BaseAdapter class.
- getView() is responsible for generating the Views from the data for display inside the list view. It takes three arguments which will be discussed here

    - The first argument is the position of the item in the list. Because we are using an array list with the listview the position will have a 1:1 mapping with the indicies of our listview items. Thus the position will be used to query the arraylist for the data to be displayed in this list item.
    - The second argument is the View object that you must use to setup your display of this list item. If this has a value of null it means you are required to create a view object and return it to the list. However, if you are presented with a View object it means that the ListView is requesting you to reuse this ListItem and just reset the data. You should not generate a new view and return it. This is done for scrolling efficiency reasons. If you do not use this efficiently then android through the ADT eclipse plugin will register warnings on your code until you fix this.
    - The third argument is the parent that owns the view. In this case it will always be the ListView that contains the View that is being generated here.
- getCount() is responsible for telling the ListView how many items in the array are to be given an item in the ListView. If you are using an ArrayList as the datastructure here you simply need to return the number of items in the ArrayList as the count
- getItem() is responsible for returning the Item in the ArrayList to the caller. You will be provided with a position value which in the case of an ArrayList has a 1:1 mapping to the indicies of the ArrayList. Note that we return our item as an Object as the BaseAdapter class is not a generic class. This means we have to know what kind of object we have and to typecast it correctly when it is returned from this method.
- getItemId() is responsible for returning the row id of the given row. Again because this is an ArrayList we are using here the position will always map to the row id so we can just return this value directly.

07) add the following private fields and static class to the CustomArrayAdapter class

```
// private fields of the class we need a copy of the context in order to
// update the list view properly and a link to the array list in order to
// provide data for generating list items
private Context context;
private ArrayList<CustomItem> al_items;

static class ViewHolder {
      public TextView tv_name;
      public TextView tv_date;
      public ImageView iv_image;
}
```

Explaination:

- We need access to a context to get access to the LayoutInflator service which is responsible for converting an XML description of a layout into a View object. This context will come from the activity that contains the ListView. And we also need a reference to the array list such that we can populate Views inside the ListView with the appropriate data.
- Note how there is a second class defined here called the ViewHolder. This is designed to take advantage of the view holder pattern. The idea here is that as we create Views for the ListView we will store a reference to the items that we extract from the XML layout. The idea being is that we will only call findViewById() only once on each View the first time it is created. If we didn't have this pattern we would have to inflate a layout each time and call findViewById() for all items in the custom layout. This is designed to sacrifice a little bit of memory in order to save CPU cycles and consequently some battery power.

08) add in the following code to the getView() method of the CustomArrayAdapter class

```
// view holder to save us from requesting references to items over and over
again
ViewHolder holder;

// if we do not have a view to recycle then inflate the layout and fix up the
view holder
if(convert_view == null) {
      holder = new ViewHolder();

      // get access to the layout infaltor service
      LayoutInflater inflator = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

      // inflate the XML custom item layout into a view to which we can add data
      convert_view = inflator.inflate(R.layout.custom_item_layout, parent,
false);

      // pull all the items from the XML so we can modify them
      holder.tv_name = (TextView) convert_view.findViewById(R.id.tv_name);
      holder.tv_date = (TextView) convert_view.findViewById(R.id.tv_date);
      holder.iv_image = (ImageView) convert_view.findViewById(R.id.iv_image);

      // set the view holder as a tag on this convert view in case it needs to
be
      // recycled
      convert_view.setTag(holder);
}
```

Explaination:

- Here we deal with the case where the ListView is asking us to create a new View for the list. In the first line we are creating a new instance of the ViewHolder object. In the second line we are using the reference to the context to ask android to give us access to the LayoutInflator service.
- In the third line we ask the layout inflator to create a view out of an XML

layout file. This method takes three arguments

  - The first is the id of the layout file that you want to inflate. Ids are automatically generated by taking the full filename and stripping away the .xml part
  - The second is to tell the layout provider who is responsible for setting the layout of this view. In our case because we have layouts already specified in the XML file we can set this to null, however this will generate a warning so in this case we set it to parent to keep the compiler happy.
  - As the last argument determines wether this layout is attached to the second argument. If you have layouts specified in the XML file you should set this to false.

- In the following lines we get access to all views inside the XML layout using the findViewById method. As you can see these are stored within the holder object which will be reused later. In order to reuse these we use the setTag method of the view to attach an arbitrary object to it, which we will then retrieve upon reuse.

09) add in the following code to the end of the getView() method of the CustomArrayAdapter class

```
else {
      holder = (ViewHolder) convert_view.getTag();
}

// set all the data on the fields before returning it
holder.iv_image.setImageResource(R.drawable.ic_launcher);
holder.tv_name.setText(al_items.get(position).getName());
holder.tv_date.setText(al_items.get(position).getDate());

// return the constructed view
return convert_view;
```

Explaination:

- The else statement deals with the other case where the view is being recycled and we are retrieving the ViewHolder that was attached to that View. The lines below pull the references from the holder and update the data of that View. With the holder we would have to make a findViewById call for each item that is in the view.
- After this we return the view to state to the list view that it can be used for display in the list again.

10) Add the following fields to the MainActivity class

```
      // private fields of the class
      private TextView tv_display;
      private ListView lv_mainlist;
      private EditText et_new_strings;
      private ArrayList<CustomItem> al_items;
      private CustomArrayAdapter caa;
```

Explaination:

- similar to the previous example but instead of an ArrayAdapter we us our custom adapter instead.

11) add the following code to the end of the onCreate() method of MainActivity.java

```
      // pull the list view and the edit text from the xml
      tv_display = (TextView) findViewById(R.id.tv_display);
      lv_mainlist = (ListView) findViewById(R.id.lv_mainlist);
      et_new_strings = (EditText) findViewById(R.id.et_new_strings);

      // generate an array list with some simple strings
      al_items = new ArrayList<CustomItem>();

      // create an array adapter for al_strings and set it on the listview
      caa = new CustomArrayAdapter(this, al_items);
      lv_mainlist.setAdapter(caa);
```

Explaination:

- Similar ot the previous example except we dont create any initial objects in the list. However this time we link the arraylist to our custom adapter and then link the custom adapter to the list view.

12) add the following code to the end of the onCreate() method of MainActivity.java (good time to compile and run)

```
// add in a listener for the edit text to create new items in our list view
et_new_strings.setOnEditorActionListener(new OnEditorActionListener() {
      public boolean onEditorAction(TextView view, int actionid, KeyEvent event)
{
            // if the user is done entering in a new string then add it to
            // the array list. this then notifies the adapter that the data has
            // changed and that the list view needs to be updated
            if(actionid == EditorInfo.IME_ACTION_DONE) {
                  al_items.add(new
CustomItem(et_new_strings.getText().toString(), System.currentTimeMillis()));
                  caa.notifyDataSetChanged();
                  tv_display.setText("added item: " + et_new_strings.getText());
                  return true;
            }

            // if we get to this point then the event has not been handled thus
            // return false
            return false;
      }
});
```

Explaination:

- similar to the previous example but instead of adding a string to the array list we add one of our custom items instead.
- Like before we state to the array adapter that the data has changed and

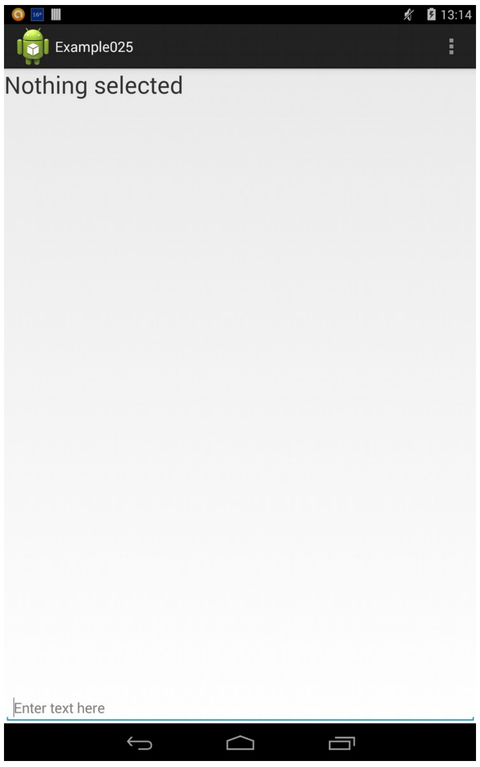
thus needs to be updated.

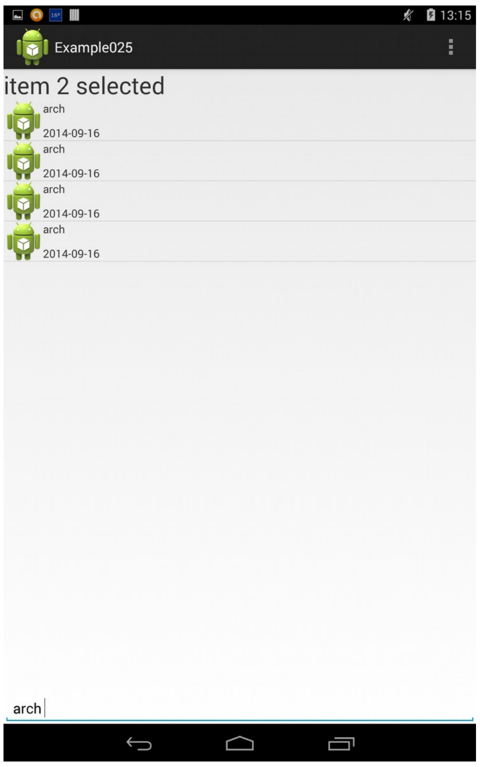
13) add the following code to the end of the onCreate() method of MainActivity.java

```
// add in a listener that listens for short clicks on our list items
lv_mainlist.setOnItemClickListener(new OnItemClickListener() {
      // overridden method that we must implement to get access to short clicks
      public void onItemClick(AdapterView<?> adapterview, View view, int pos,
long id) {
            // update the text view to state that a short click was made here
            tv_display.setText("item " + pos + " selected");
      }
});

// add in a listener that listens for long clicks on our list items
lv_mainlist.setOnItemLongClickListener(new OnItemLongClickListener() {
      // overridden method that we must implement to get access to long clicks
      public boolean onItemLongClick(AdapterView<?> adapterview, View view, int
pos, long id) {
            // update the display with what we have just long clicked
            tv_display.setText("item " + pos + " long clicked");

            // as we are going to change the textview anyway we can
automatically
            // return true;
            return true;

      }
});
```

Explaination:

- similar to the previous example but instead of accessing the string directly we show the position instead
- running this code will get you the screenshots below where we see the initial state (left) and the app after the user has entered a few items and interacted with them (right)


13:14
Example025
Nothing selected
Enter text here


13:15
Example025
item 2 selected
arch
2014-09-16
arch
2014-09-16
arch
2014-09-16
arch
2014-09-16
arch

Example026: introducing the activity lifecycle handler methods.

In this simple example you will be introduced to the lifecycle handler methods that android calls to manage applications and system resources. Control over when of your application runs is entirely the decision of the Android RunTime (ART) which is the replacement for the Dalvik system that was introduced in Android KitKat (4.4). While android has full control over when and how your application runs it does give you lifecycle handler methods that will be called before your application or activity changes state. In this example we will show the basic structure of these handlers and we will describe the purpose of each.

01) start a new android project

02) add this method to the MainActivity class

```
	// overridden destroy method that should be used to free up memory as this
	// activity is being destroyed
	protected void onDestroy() {
		super.onDestroy();

		Log.i("MainActivity", "onDestroy() called");
	}
```

Explaination:

- onDestroy() is the counterpart of the onCreate() method
- The onCreate() method is called for a newly instantiated Activity. The onCreate() method is responsible for initialising and setting up the layout and fields for the Activity.
- onDestroy() is responsible for freeing up resources (be it memory, files, sensors etc) that the activity is finished with. This is to prevent memory leaks in the application.
- Note that for the onDestroy() method that the first line you are required to have in the method is super.onDestroy(). Failure to make this method call will result in an exception that will terminate you application.

03) add the following methods to the MainActivity class

```
// overridden resume method that should be used to restart threads and
processing
// when the activity becomes fullscreen and unobstructed. this method should be
// quick at processing because it has the potential to be called many times in a
// single activity
protected void onResume() {
	super.onResume();

	Log.i("MainActivity", "onResume() called");
}

// overridden pause method that should be used to temporarily stop threads and
// processing when the activity has become partially obstructed (e.g. a dialog
// has popped up). This method should be quick at processing because it has the
// potential to be called many times in a single activity
protected void onPause() {
      super.onPause();

      Log.i("MainActivity", "onPause() called");
}
```

Explaination:

- The onResume() method is called by Android when your activity is shown on the screen without obstruction from a dialog or another activity. This method is usually used to resume threads or processing for the current activity.
- The onPause() method is the counterpart to onResume() this is called whenever the current activity becomes obstructed by a dialog or another activity is being made visible. It is expected that you will pause processing tasks here as it is assumed that the user wants the processing power available for the new activity or dialog that is being shown to them.
- In order to maintain application responsiveness you are required to keep this method as short and simple as possible as both of these methods have the potential to be called multiple times in the same activity particularly if the user is interacting with many dialogs.

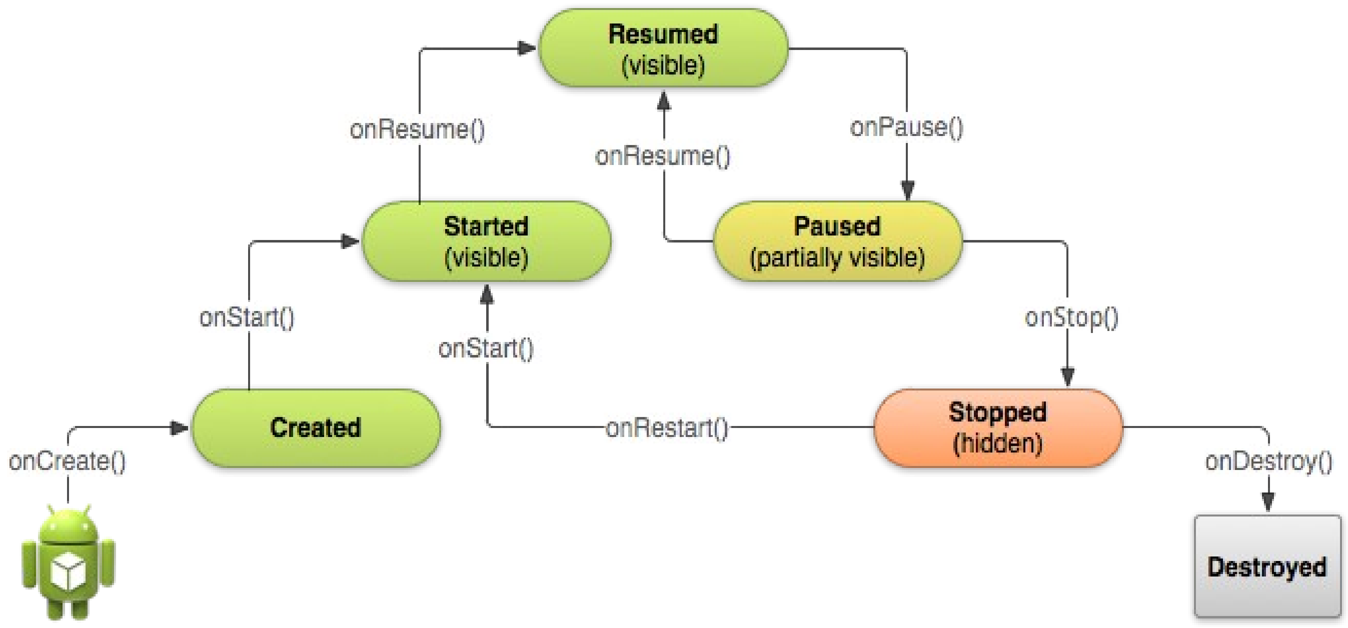
04) add the following methods to the MainActivity class

```
// overridden stop method that is used to temporarily free up resources for
other
// activities and applications. This is called whenever the activity becomes
// completely obscured or backgrounded.
protected void onStop() {
      super.onStop();

      Log.i("MainActivity", "onStop() called");
}

// overridden restart method that is used to restart an activity. most of the
time
// you can ignore this method as the on start method will be called imediately
// after this method. It remains for compatibility purposes
protected void onRestart() {
      super.onRestart();

      Log.i("MainActivity", "onRestart() called");
}

// overridden start method that is used to allocate resources to the activity
both
// when it is created or when it is being restarted from a stopped state. Any
// memory freed from onStop should be reallocated here
protected void onStart() {
      super.onStart();
```

```
        Log.i("MainActivity", "onStart() called");
}
```

Explaination:

- The onStart() and onStop() methods are considered counterparts to each other.
- The onRestart() handler was introduced as a means for reallocating resources prior to restarting the activity. However this is rarely used now because after the call to onRestart() finishes the call to onStart() follows immediately. Thus the official documentation recommends that you use the onStart() method instead of onRestart(). onRestart() is retained for application compatibility purposes.
    - Another reason why onStart() is preferred is that it also gets called immediately after the call to onCreate() meaning that resource allocation can be localised in one place.
- The onStop() handler is called when your activity becomes completely obscured by another activity or has been backgrounded by a user switching applications. It is expected that your activity will use this method to temporarily free up resources for the new activity that is becoming visible to the user. Resources do not just mean memory it can also include hardware such as the camera and sensors.
- The onStart() method performs the opposite job to onStop() it is there to reallocate resources that were temporarily freed for other activities because this activity is becoming visible again. It may be necessary to reinitialise hardware here.
- Upon running the program you will see messages appearing in Logcat when each of the methods are being called. If you switch applications, start it, kill it you will see sequences of these messages appearing in the Logcat to give you an idea of when each method is called.
- The diagram you see below is from the offical Android documentation showing how all the lifecycle handlers relate to each other and the states of each activity.


Resumed
(visible)
onResume()
onPause()
onResume()
Started
(visible)
Paused
(partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed

Example027: introducing the menu, how to add items and their event handling.

In this example you will be introduced to the menu that is attached to each and every activity and also how to pick up events that indicate selection of menu items.

01) start with a new android application

02) add the following strings to strings.xml

```
    <string name="tv_display_init">No menu item selected</string>
    <string name="mi_my_menu_item_string">My menu item</string>
```

03) replace the XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv_display"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="30sp"
      android:text="@string/tv_display_init"
    />
</LinearLayout>
```

04) open the file res/menu/menu.xml and you should see the following XML

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.example027.MainActivity" >

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/action_settings"/>

</menu>
```

Explaination:

- This menu is the one that is loaded and displayed on the MainActivity. The onCreateOptionsMenu() method of MainActivity uses a MenuInflater to inflate this file and attach it as the menu for this activity. All menu files must have a root <menu></menu> tag pair that contains the three attributes that you see above. Note that the tools:context attribute should always specify which Java file that this menu will be used in and follows the format of <package name>.<Java file name>

- The item tag is used to declare a new menu item. You are required to have at least four attributes in here which you see above. The id is necessary for event handling and is declared in the same way as ids for normal layouts.
    - The orderInCategory attribute is there to sort the order of the menuitems for display. The smaller the value placed here the further towards the top of the list it will appear. Increasing this value will move the menu item towards the bottom of the list.
    - The showAsAction attribute determines if the menu item is displayed in the menu or on the action bar itself. The action bar is the black bar you see at the top of an activity, with the applicaton icon, application name and also the three vertical dots that represent the menu. If this is set to never then the item will always be kept inside the menu. If you specify a value of ifRoom then the item will be displayed on the action bar itself beside the three dots and can be directly accessed there.
        - It is highly recommended that you only use this for a small number of actions that are accessed frequently. A good example of such an action is the compose message in the Gmail application.
    - The title attribute is the text that will be displayed for the item.

05) add the following item to menu.xml

```
<item
  android:id="@+id/mi_my_menu_item"
  android:orderInCategory="110"
  android:showAsAction="never"
  android:title="@string/mi_my_menu_item_string"
/>
```

Explaination:

- here we are defining our own menu item and adding it into the list. We give it an id and set it to have an order value of 110 meaning it will appear below the settings option. If this was set to a value less than 100 it would appear above the settings option. We also state that this will always remain in the menu and will never go on the action bar. As an experiment see what happens if you set this to “ifRoom”

06) add the following field to the MainActivity class

```
// the textview that is part of the layout
private TextView tv_display;
```

07) add the following code to the end of the onCreate() method of the MainActivity class

```
// pull the textview from the XML
tv_display = (TextView) findViewById(R.id.tv_display);
```

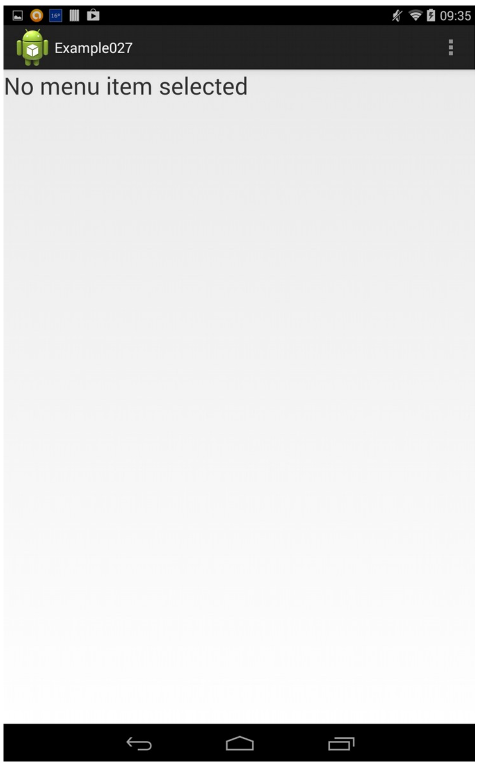
08) In the onOptionsItemSelected() method replace all code in that method

with the following code (good time to compile and run)

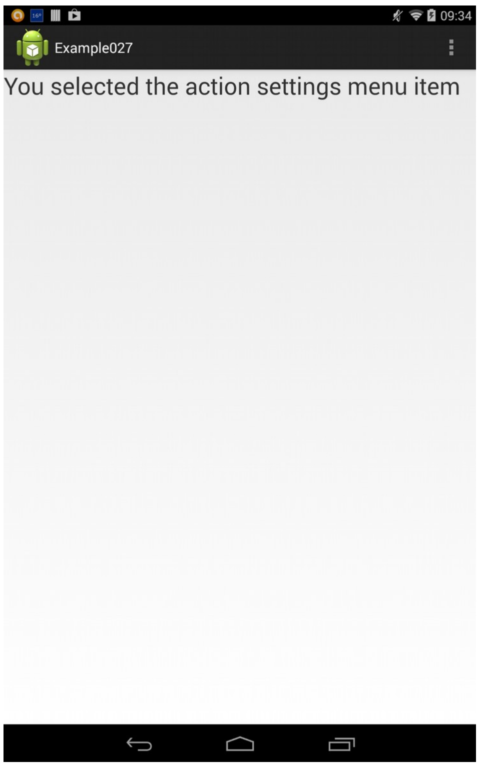
```java
// Handle action bar item clicks here. The action bar will
// automatically handle clicks on the Home/Up button, so long
// as you specify a parent activity in AndroidManifest.xml.
int id = item.getItemId();

// depending on the item that was selected change the text of the textview.
// we must return true to indicate that the menu item has been handled
if (id == R.id.action_settings) {
      tv_display.setText("You selected the action settings menu item");
      return true;
} else if (id == R.id.mi_my_menu_item) {
      tv_display.setText("You selected the my menu item menu item");
      return true;
}

return super.onOptionsItemSelected(item);
```

Explaination:

- This method is an event handler for the menu that is attached to this activity. Any time a user clicks on a menu item of an activity this method will be automatically called with the menu item that was clicked.
- The first line here extracts the id that was attached with to the menu item with the android:id attribute. As there are multiple items in the menu we must check against each menu item id to determine what item was selected. Generally I will use an if-else-if ladder here but it is also a good place to use a switch-case statement if you prefer to use that instead.
- When we find the id that matches the menu item we can then do the appropriate actions for that menu item. In this case we are simply changing the value of the text view to indicate which one was selected. Note that if you consume the menu item you are required to return true to indicate this to the android system.
- If you do no handle the menu item click it is generally good practice to call the superclass method and return the value from that.
- Running this code will get you the following screenshots where we see the initial state of the application (left) and where the user has clicked on a menu item (right) the menu on a nexus 7 is a software button displayed as 3 vertical dots in the top right. Note that some devices have a hardware menu key in which case these dots will not be shown. If this is the case with your device then press the menu key on your device to bring up the menu.



No menu item selected



You selected the action settings menu item

Example028: creating a full custom view and handling touch events

One of the most common tasks for Android development is the building of custom widgets. The SDK will not provide all the widgets you will need. It will only provide the most common widgets. There will be times where you will require a widget that you will have to construct yourself. This generally consists of two components. The first is drawing the widget itself and the second is handling touch events from the user. This example will show both integrated together and will also show how to deal with single touch and multi touch events.

There are no screenshots for this example as it is extremly difficult to touch the screen and also try and take a screen cap at the same time. What you should see though is a single blue square on the screen. When you place your first finger on the screen you should see that square turn red and be under your finger, and it will follow you wherever you move. If you place further fingers on the screen you will see additional green squares following those fingers. Should you remove a finger a square will disappear. If it is the red square that you remove the finger from the oldest touch will change to a red square. Releasing all fingers will leave you with a single blue square in the place where you removed your last finger.

This will show the general case for multitouch, you may use touch to implement swipe, scale, rotate operations or other gestures, meaning you can reduce the amount of views visible on the screen

01) start a new android project

02) create a new source file called CustomView.java and give it the following shell code.

```java
package com.example.example028;
// class that implements a custom view that will handle touch events

// imports
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;

// class definition
public class CustomView extends View {
      // default constructor for the class that takes in a context
      public CustomView(Context c) {
            super(c);
            init();
      }
```

```
	// constructor that takes in a context and also a list of attributes
	// that were set through XML
	public CustomView(Context c, AttributeSet as) {
		super(c, as);
		init();
	}

	// constructor that take in a context, attribute set and also a default
	// style in case the view is to be styled in a certian way
	public CustomView(Context c, AttributeSet as, int default_style) {
		super(c, as, default_style);
		init();
	}

	// refactored init method as most of this code is shared by all the
	// constructors
	private void init() {

	}

	// public method that needs to be overridden to draw the contents of this
	// widget
	public void onDraw(Canvas canvas) {
		// call the superclass method
		super.onDraw(canvas);
	}

	// public method that needs to be overridden to handle the touches from a
	// user
	public boolean onTouchEvent(MotionEvent event) {
		// if we get to this point they we have not handled the touch
		// ask the system to handle it instead
		return super.onTouchEvent(event);
	}

}
```

Explaination:

- Whenever you create a new custom view you are required to extend the View class. At the bare minimum you must provide implementations of the three constructors you see above and also an implementation of the onDraw() method. The implementation of the onTouchEvent method is optional but in most cases you will need this method if you wish to react to touch input from the user.
  - Note that in all three constructors there is a shared init method. The reason this should be added is that all three constructors will share a lot of the same initialisation code so it has been refactored out into a separate method.
- The first constructor will be used if you are initialising your custom view entirely through Java code with no XML interaction. In this case the widget only needs to know about the owning context (the context of the activity containing this view)

    - Note that all constructors require a call to the superclass constructor otherwise you will get errors and your custom view will not initialise properly
- The second constructor will be used if you declare your custom view in an XML layout file. The context will be automatically be provided by the layout however the attribute set is the list of attributes that appear after your view's name and before the /> of the tag in the XML layoutfile. Generally in this method you would parse any tags that are in the attribute set and adjust your widget accordingly. Any tags that are prefixed with android: you do not have to parse as these will be handled by the superclass constructor
- The third constructor will be used if the view has been declared in an XML file with a set of attributes and also a style attribute set. Generally you will use the style to query the color/text scheme that is being used by the device and you can adjust your view to fit in better with the user's device.
- The onDraw() method is where you will draw the contents of your widget. All drawing should be performed using the canvas object that is provided to you.
- The onTouchEvent() method is called whenever the user touches the screen or moves their touch along the screen. This is where you will react to user input. The MotionEvent object will give you information about what fingers are touching the device, how they are moving, and when fingers are added or removed from the device

03) remove all XML in activity_main.xml and replace it with the following XML

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
      <com.example.example028.CustomView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
      />

</LinearLayout>
```

Explaination:

- If you have a custom view defined and you wish to use it in XML its tag will be the fully qualified package name followed by the name of the java file itself minus the .java extension.
- Here we set this view to take the full size of the activity. This means our second constructor will be called and the superclass constructor will handle the layout_width and layout_height attributes set above.

04) add in the following fields to the CustomView class

```
// private fields that are necessary for rendering the view

// the colours of our squares
private Paint red, green, blue;
private Rect square;           // the square itself
private boolean touches[];     // which fingers providing input
private float touchx[];        // x position of each touch
private float touchy[];        // y position of each touch
private int first;             // the first touch to be rendered
private boolean touch;         // do we have at least on touch
```

Explaination:

- The Paint objects will be used to represent different colours. While the Rect will represent a rectangular shape. Both will be combined to draw rectangles of varying colours on the canvas.
- touches, touchx and touchy are arrays that represent which fingers are touching the display, their x coordinate, and y coordinate, of the touch.
- first keeps a track of which of the touches is the oldest and therefore the active touch. Depending on your application you may or may not need to keep track of the first touch but it is entirely dependant on the gestures you are planning to implement
- touch is a simple boolean that represents if we have a one or more touches on the device or if there are no touches on the device

05) add in the following code to the init method

```
// create the paint objects for rendering our rectangles
red = new Paint(Paint.ANTI_ALIAS_FLAG);
green = new Paint(Paint.ANTI_ALIAS_FLAG);
blue = new Paint(Paint.ANTI_ALIAS_FLAG);
red.setColor(0xFFFF0000);
green.setColor(0xFF00FF00);
blue.setColor(0xFF0000FF);

// initialise all the touch arrays to have 16 elements as we know no way to
// accurately determine how many pointers the device will handle. 16 is an
// overkill value for phones and tablets as it would require a minimum of
// four hands on a device to hit that pointer limit
touches = new boolean[16];
touchx = new float[16];
touchy = new float[16];

// initialise the first square that will be shown at all times
touchx[0] = 200.f;
touchy[0] = 200.f;

// initialise the rectangle
square = new Rect(-100, -100, 100, 100);

// we start off with nothing touching the view
touch = false;
```

Explaination:

- In the first block we are initialising the paint objects. It is recommended that all paint objects should have their anti aliasing flag set as this will make text and curved objects appear smooth. Note that the setting of the colours here is following the same format as the colour resources that you saw earlier (0xAARRGGBB format)
- The second block of code initialises support for 16 pointers on the device. This is an arbitrary value as there is no way of reliably querying an android device for the number of pointers the screen supports. 16 is considered an overkill value as this would require a minimum of four hands on the device to break this limit.
- The first touch is initialised to be at 200 pixels to the right in x and 200 pixels down in y.
- The next line defines a 200x200 pixel square that will represent the rectangle. By specifying a top left corner of (-100,-100) this means that the top left corner of the rectangle will be 100 pixels to the left, and 100 pixels up from the drawing origin.The bottom right corner is set to (100,100) which is 100 pixels to the right, and 100 pixels down from the drawing origin. What this means is that the center of the square will be the origin itself as it is at halfway in both dimensions.
- Finally we state that no touch has been made on this widget yet

06) add in the following code to the end of the onDraw() method of CustomView

```
// draw the rest of the squares in green to indicate multitouch
for(int i = 0; i < 16; i++) {
	if(touches[i]) {
		canvas.save();
		canvas.translate(touchx[i], touchy[i]);
		if(first == i)
			canvas.drawRect(square, red);
		else
			canvas.drawRect(square, green);
		canvas.restore();
	}
}
```

Explaination:

- Here is the code responsible for drawing all the active squares on the canvas. The if statement queries if the current touch is active. If not then there is not point rendering it.
- Inside the if statement we see how the squares are drawn. Android uses an OpenGL/DirectX like matrix stack to keep track of transformations (translate, scale, rotate) as a way of removing floating point error. Because we are using floating point numbers the more we translate to a square and back the more floating point error will creep in and will renderer our squares with an increasing offset from the user touch.
- The origin of drawing operations is (0,0) which is the top left of the screen. Calling canvas.save() will save this origin onto the matrix stack. Canvas.translate() takes in an x and y value which will move the origin of drawing to (x,y) thus any drawing operations made without providing an x and y value will be drawn at this point. If an x and y value are provided then the object will be drawn relative to the new origin. Calling canvas.restore() at the end restore our drawing origin to (0,0)
- The if else statement states that if this pointer is listed as the first pointer (i.e. the oldest) then it should be rendered in a red colour otherwise render it in a green colour.

07) add in the following code to the end of the onDraw() method of CustomView

```
// if there is no touches then just draw a single blue square in the last place
if(!touch) {
      canvas.save();
      canvas.translate(touchx[first], touchy[first]);
      canvas.drawRect(square, blue);
      canvas.restore();
}
```

Explaination:

- this is used to render a single blue square in the position of the oldest touch before all touches were removed from the screen.

08) add in the following code at the beginning of the onTouchEvent() method of CustomView

```
// determine what kind of touch event we have
if(event.getActionMasked() == MotionEvent.ACTION_DOWN) {
      // this indicates that the user has placed the first finger on the
      // screen what we will do here is enable the pointer, track its location
      // and indicate that the user is touching the screen right now
      // we also take a copy of the pointer id as the initial pointer for this
      // touch
      int pointer_id = event.getPointerId(event.getActionIndex());
      touches[pointer_id] = true;
      touchx[pointer_id] = event.getX();
      touchy[pointer_id] = event.getY();
      touch = true;
      first = pointer_id;
      invalidate();
      return true;
} else if(event.getActionMasked() == MotionEvent.ACTION_UP) {
      // this indicates that the user has removed the last finger from the
      // screen and has ended all touch events. here we just disable the
      // last touch.
      int pointer_id = event.getPointerId(event.getActionIndex());
      touches[pointer_id] = false;
      first = pointer_id;
      touch = false;
      invalidate();
      return true;
} else if(event.getActionMasked() == MotionEvent.ACTION_MOVE) {
      // indicates that one or more pointers has been moved. Android for
      // efficiency will batch multiple move events into one. thus you
      // have to check to see if all pointers have been moved.
      for(int i = 0; i < 16; i++) {
            int pointer_index = event.findPointerIndex(i);
            if(pointer_index != -1) {
                  touchx[i] = event.getX(pointer_index);
                  touchy[i] = event.getY(pointer_index);
            }
      }
      invalidate();
      return true;
}
```

Explaination:

- There is a lot of code here to begin with but a lot of it is repeated. However will go through it bit by bit.
- In each of the if statements we call event.getActionMasked() which will return the last event that was generated by one of the pointers. Generally it is a good idea to use getActionMasked() instead of getAction() because getAction() only supports a single pointer and will be depreciated soon enough. It only exists for compatibility reasons. In this case we are looking for the ACTION_DOWN event which indicates to us that the user has placed their first finger on the device and has initiated the first touch.
  - The first thing we must do is query which pointer was added. We do this by asking the MotionEvent to get the pointer id of the pointer that generated the event. The reason we need getActionIndex() is because android maintains a list of pointers that are currently active. This list can get bigger and smaller meaning there will not be a 1:1 mapping between pointers in the list and pointer ids. This is a necessary conversion that must be applied at all times if you wish to work with touch safely. In the rest of this code we enable the touch for that pointer and the event x and y values are copied so the square can be rendered where the user has touched the screen. Finally we state that a touch has started. The invalidate() method is called to tell android that the state of this View has changed and needs to be rerendered to show the changed state. We then return true to indicate that this event has been handled.
  - In all branches of the if-else-if we will invalidate and return true as the display is changing and the event has been handled.
  - The ACTION_UP event indicates that the user has removed their last finger from the device. Here we disable the square from being rendered. However we also register this as the oldest touch. The second block of code in the onDraw() method will use this value to show the user that this was the oldest touch before the user removed all fingers from the device
  - The ACTION_MOVE event is used to indicate that one or more pointers has moved on the screen. For efficiency reasons android will batch multiple move events into a single move event. Thus it is your responsibility to check if all pointers have moved as you cannot be guarenteed that you will get a separate move event for each pointer that has moved. Inside the loop the first thing we do is find the index of the pointer id in the list of pointers that are currently active. It is entirely possible that the pointer id does not exist in which case you will be returned an index value of -1. you should check for this before you try to query the x and y values for this pointer.

09) add in the following code after the 08 code in the onTouchEvent() method of CustomView (good time to compile and run)

```
else if(event.getActionMasked() == MotionEvent.ACTION_POINTER_DOWN) {
      // indicates that a new pointer has been added to the list
      // here we enable the new pointer and keep track of its position
      int pointer_id = event.getPointerId(event.getActionIndex());
      touches[pointer_id] = true;
      touchx[pointer_id] = event.getX(pointer_id);
      touchy[pointer_id] = event.getY(pointer_id);
      invalidate();
      return true;
} else if(event.getActionMasked() == MotionEvent.ACTION_POINTER_UP) {
      // indicates that a pointer has been removed from the list
      // note that is is possible for us to lose our initial pointer
      // in order to maintain some semblance of an active pointer
      // (this may be needed depending on the application) we set
      // the earliest pointer to be the new first pointer.
      int pointer_id = event.getPointerId(event.getActionIndex());
      touches[pointer_id] = false;
      if(pointer_id == first) {
            for(int i = 0; i < 16; i++)
                  if(touches[i]) {
                        first = i;
                        break;
                  }
      }
      invalidate();
      return true;
}
```

Explaination:

- The ACTION_POINTER_DOWN event indicates that a second or subsequent pointer has been added to the screen. In this case we enable the touch for the pointer id and by providing the pointer id to the getX and getY we can obtain the x and y value associated with this pointer. If you do not provide a value here you will get the x and y value for the first pointer instead.
- The ACTION_POINTER_UP indicates that one of the additional touches has been removed from the device. Here we disable the square associated with that touch. We also need to query if this touch was the oldest. If it was then we must chose another pointer as the oldest (usually the one with the lowest id) to act as the active pointer of this touch.
- When you run this code you should be able to place your fingers on the screen and get full visual feed back on the application following your fingers at all times.

Example029: Creating a custom view by combining one or more views.

Sometimes you may find that it the custom view you need can be made by combining a few of the views that are already provided by the android sdk. In this example we will build a custom edit text that consists of an edit text and a button alongside it to clear the text in the edit text.

01) start with a new android project

02) add the following strings to strings.xml

```
    <string name="et_hint">Enter text here</string>
    <string name="btn_text">Clear</string>
```

03) create a new layout file in res/layout called compound_view_layout.xml and give it the following XML

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="horizontal"
>
      <EditText
          android:id="@+id/et_entry"
            android:layout_width="0dp"
            android:layout_weight="4"
            android:layout_height="wrap_content"
            android:hint="@string/et_hint"
            android:inputType="text"
      />
      <Button
          android:id="@+id/btn_clear"
          android:layout_width="0dp"
          android:layout_weight="1"
          android:layout_height="wrap_content"
          android:text="@string/btn_text"
      />
</LinearLayout>
```

04) create a new source file called CustomEditText.java and give it the following shell code

```
package com.example.example029;

// simple class that links an edit text and a button together
// the button will clear all text out of the edit text

// imports
import android.content.Context;
import android.util.AttributeSet;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;

// class definition
public class CustomEditText extends LinearLayout {
	// required constructor that takes in a context
	public CustomEditText(Context c) {
		super(c);
		init();
	}

	// required constructor that takes in an attribute set
	public CustomEditText(Context c, AttributeSet as) {
		super(c, as);
		init();
	}

	// required constructor that takes in a style
	public CustomEditText(Context c, AttributeSet as, int default_style) {
		super(c, as, default_style);
		init();
	}

	// private method that will share initialisation between all three
	// constructors
	private void init() {

	}

	// private fields of the class
	private EditText et_edit;
	private Button btn_clear;
}
```

Explaination:

- almost the same shell as the custom widget in the previous example. Note that this edit text extends the LinearLayout. The reason for this is that the root node of the XML file that we will use to control the layout is a LinearLayout. If this was using a RelativeLayout in the XML we would extend the RelativeLayout class instead.

- Note that we have fields in place for the EditText and the Button in case any other methods need to use or access them in the class.

05) add in the following code to the init() method of the CustomEditText class

```
// get access to the layout inflator service and inflate the XML file
LayoutInflater li = (LayoutInflater)
getContext().getSystemService(Context.LAYOUT_INFLATER_SERVICE);
li.inflate(R.layout.compound_view_layout, this, true);

// get access to the controls and add a listener
et_edit = (EditText) findViewById(R.id.et_entry);
btn_clear = (Button) findViewById(R.id.btn_clear);
btn_clear.setOnClickListener(new OnClickListener() {
      // overridden method that will clear the text in the edit text
      // when the button is clicked
      public void onClick(View v) {
            et_edit.setText("");
      }
});
```

Explaination:

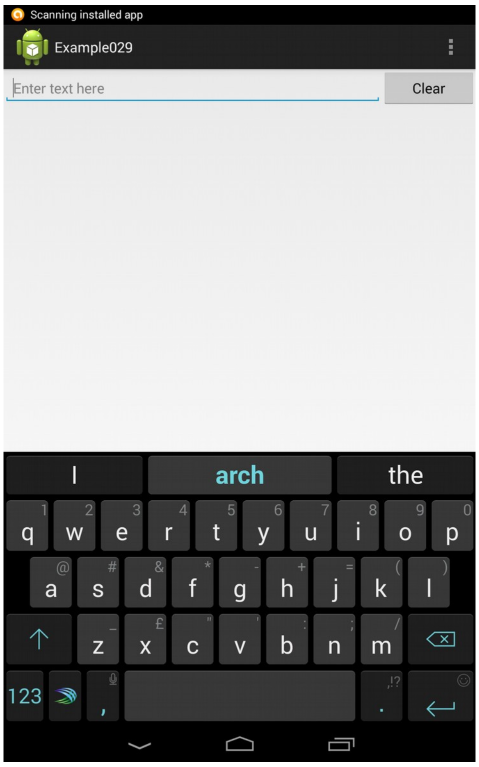
- here we are using the layout inflater to take our XML, notice how the last two arguments here have different values to those that were used in the custom ListView item. Because we are extending a Layout class this class will set the layout parameters for every view within the XML layout. Thus it will be attached to this class by using a combination of the “this” and “true” keywords.
- We then set a simple click listener on the button that will clear the edit text when it is clicked.

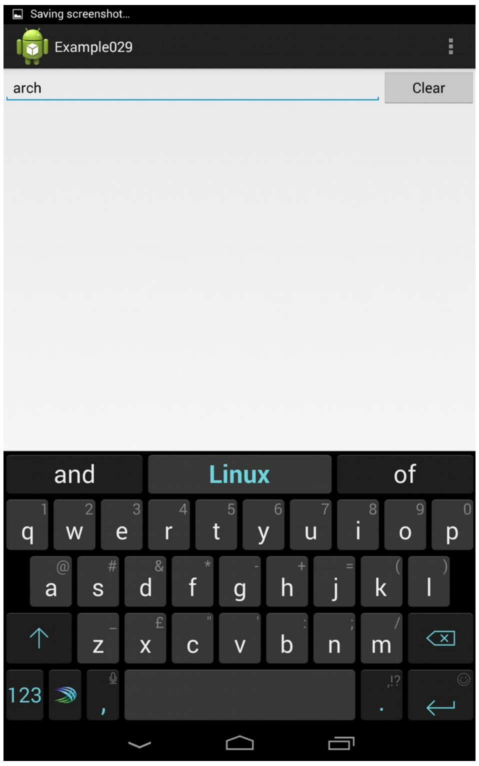
06) replace all XML in activity_main.xml with the following XML (good time to compile and run)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>
    <com.example.example029.CustomEditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
    />
</LinearLayout>
```

Explaination:

- similar declaration to the previous example for putting your custom view in code.
- Running this code will get you the following screenshots. The initial state (left) is also the state you will see when the clear button is clicked. The state on the right shows some text being entered into the edit text.


Scanning installed app
Example029
Enter text here
Clear
I
arch
the


Saving screenshot...
Example029
arch
Clear
and
Linux
of

Example030: Creating adaptable components by permitting attributes to be set on them.

In the previous examples of custom widgets we only showed how to create and make them interactive. In this example we will make a widget reusable by making it accept custom attributes. Our example will take the number picker class that does not accept a minimum or maximum value through XML and we will add code to permit these values to be set in XML.

01) start a new android project.

02) add a new source file called CustomNumberPicker.java and give it the following shell code.

```java
package com.example.example030;

// extension of the number picker class that will accept a max and min
// attribute

// imports
import android.content.Context;
import android.util.AttributeSet;
import android.widget.NumberPicker;

// class definition
public class CustomNumberPicker extends NumberPicker {
	// required constructor that accepts a context
	public CustomNumberPicker(Context c) {
		super(c);
	}

	// required constructor that accepts an attribute set
	public CustomNumberPicker(Context c, AttributeSet as) {
		super(c, as);

	}

	// required constructor that accepts a default style
	public CustomNumberPicker(Context c, AttributeSet as, int default_style) {
		super(c, as, default_style);
	}

}
```

Explaination:

- similar shell to the previous two examples. Note that there is no shared init method because we are only going to modify the second constructor.

03) add the following code to the Constructor that takes in a Context and an AttributeSet of the CustomNumberPicker class

```
// pick out our attributes for min and max and set their values on the number
// picker (assumes that values are entered correctly you should do validation
// here)
int max = as.getAttributeIntValue(null, "max", 0);
int min = as.getAttributeIntValue(null, "min", 0);
setMaxValue(max);
setMinValue(min);
```

Explaination:

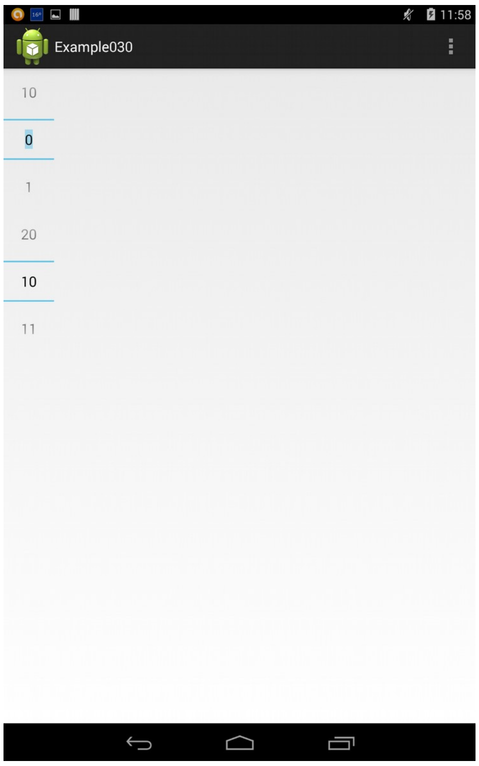
- here we are asking for the attribute called “max” and to get the integer value of that attribute. The last last value in this call is a default value that is provided should the attribute not exist in the XML or if the attribute does not parse correctly. The first parameter that is set to “null” is the namespace for the attribute. For example if we wanted to pick up the attribute for “android:layout_width” where “android” is the namespace and layout_width is the attribute name then we would provide android as the first argument and layout_width as the second argument.
- After the values have been retrieved we set the minimum and maximum values on the number picker.

04) replace all XML in activity_main.xml with the following XML (good time to compile and run)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
    <com.example.example030.CustomNumberPicker
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      max="10"
      min="0"
    />
    <com.example.example030.CustomNumberPicker
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      max="20"
      min="10"
    />
</LinearLayout>
```

Explaination:

- here we have a simple vertical layout with two number pickers using the new attributes we have defined.
- Running this code will net you the following screenshot where the first number picker has a range of 0 to 10 and the second number picker has a range from 10 to 20



Example031: Starting a new activity with an explicit intent and an introduction to the Manifest file

All of the examples you have done to this point have focused on apps that have a single activity. In most useful android applications however there are normally multiple activities that will compose the full application. The idea is that an activity should focus on a certian task that a user should perform (e.g. change app settings, send an email, read an email etc.). In order to transition from one activity to another we have to use Android's intent system. In this application we will define two activities. The first will contain a single button that when clicked will pause then stop the first activity (and call the appropriate life cycle handlers). It will then send a request to the android system that asks android to start a new activity. Android will then construct that activity (through the onCreate() method) and will show that activity to the user. This example will show how to use a simple explicit intent to connect two activities together

01) start a new android project

02) in strings.xml add the following strings

```
    <string name="second">This was started by an intent</string>
    <string name="btn_text">Start Second Activity</string>
```

03) in activity_main.xml replace all XML with the following XML

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
    <Button
        android:id="@+id/btn"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/btn_text"
    />
</LinearLayout>
```

04) in res/layout add a new file called activity_second.xml and give it the following XML

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/second"
      android:textSize="30sp"
    />
</LinearLayout>
```

Explaination:

- Different activities will require different layout files. A recommended practice with android is the following. Every time you define a new activity you should also define a new XML layout file.
- By calling this file activity_second.xml it will automatically be given an identifier name of “activity_second”. We will use this identifier name inside the second activity to get this layout and set it on the second activity.

05) add in a new source file called SecondActivity.java and give it the following shell code

```
package com.example.example031;
// second activity class that will be launched from Main Activity

// imports
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

// class definition
public class SecondActivity extends Activity {
      @Override
      protected void onCreate(Bundle savedInstanceState) {
            // call the super class method and set the content for this activity
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_second);

      }

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

```
	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		// Handle action bar item clicks here. The action bar will
		// automatically handle clicks on the Home/Up button, so long
		// as you specify a parent activity in AndroidManifest.xml.
		int id = item.getItemId();
		if (id == R.id.action_settings) {
			return true;
		}
		return super.onOptionsItemSelected(item);
	}
}
```

Explaination:

- same as the shell code that you have used for the past 30 examples. The only difference here is the call to setContentView(). In this call instead of passing R.layout.activity_main we now pass R.layout.activity_second
  - this will tell android to get the file res/layout/activity_second.xml, inflate it, and attach it to as the display for this activity.

06) add in this code to the end of the onCreate() method of the MainActivity class

```
// get the button from the layout and add a listener to it
btn = (Button) findViewById(R.id.btn);
btn.setOnClickListener(new OnClickListener() {
	// overridden method to handle a button click
	public void onClick(View v) {
		Intent intent = new Intent(MainActivity.this, SecondActivity.class);
		startActivity(intent);
	}
});
```

Explaination:

- Adding an event handler to the button However note the code in the onClick() method
- This is how you describe to android how to start a new activity. For an explicit intent we generate an intent object that links two subclasses of the Activity class together. In the creation of the Intent object the first argument is the class that the intent is launched from, and the second argument is the activity class that we wish to start now.
  - Note that the first argument uses MainActivity.this if we only said “this” here it would reference the anonymous class that we have just defined. By Saying MainActivity.this we are asking java to reference the outer class that contains this anonymous class which is the MainActivity class.
  - Once the intent has been setup we call startActivity() with the intent to start the new activity
- Note that you should not run this code now. Android requires that all

activities in an Application must be declared in the Manifest file. We will take a look at the manifest file, explain what it does and show you how to modify it to reference the new activity

07) open up the AndroidManifest.xml and you will see the following XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.example031"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="14"
        android:targetSdkVersion="20" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Explaination:

- This is the default manifest file that is generated for you automatically by Eclipse. This tells the android device vital information about the package that is being installed.
- The Manifest file is also responsible for covering telling the android system what list of permissions an application requires in order to function properly. However this will be covered in a later example. For now we will explain what we see in the current manifest file
- The first line you have seen before. This is declaring the version of XML that this file is written in.
- The second line introduces the manifest tag. The android manifest file requires that the root tag pair is <manifest> </manifest> everything defined between these tags are considered to be the contents of the manifest file
    - We have four properties as part of the manifest file. The first is the xmlns:android property that you should be well used to from all the layout files that you have written.

    - The second property declares a Java package that contains the majority of source files for our activity classes. By declaring the package here we can give relative paths to each of our source files for activities instead of the fully qualified java name.
    - versonCode is an internal version number that is used to keep track of which version of the project is being worked on. This number is here for developers only. It is never exposed to the user.
    - versionName is a version number that will be exposed to the user. This should be updated every time you release a new version of your application. This version name will also be displayed for your application in the play store.
- The uses-sdk tag specifies to an android device which android versions that this application will run on.
    - The minSdkVersion specifies the minimum API level that is necessary for this application to run. All applications in this book should have a minimum API level of 14 (Android 4.0 Ice Cream Sandwich) an Android device that has an API level greater than or equal to this will run the application. Any android device that has a lower API level will refuse to run the application.
    - The targetSdkVersion attribute specifies the version of the android sdk that this application was built with.
- The <application></application> tags specify properties and the structure of the android application.
    - By default four attributes are present in the application tag. The first is the allowBackup tag. If this is set to true the user data for this application can be backed up to a user's google account in the cloud.
    - The icon tag specifies the icon that is displayed in the launcher for this application. This icon will also be displayed in the top left of the action bar as well.
    - The label property states the name that will appear in the launcher for this application. This can be completely different to the name of your project. It does not have to be the same.
    - The theme property states what them is to be used for this application. You can choose a theme explicitly but for the sake of fitting in with other applications on an android device we use “@style/AppTheme” to use the currently set system theme.
- Every activity that can be shown to the user must be specified in <activity></activity> tags. If you try to launch an activity without it being declared in the manifest then android will throw an exception and will kill your application immediately.
    - All activities must have two properties. The first is the name property

which specifies the name of the java class that implements this activity. Here we use ".MainActivity" to specify the MainActivity class. Note that because we declared the package attribute earlier to be "com.example.example031" this name will be appended to the package to generate the fully qualified java name of "com.example.example031.MainActivity" with gives Android the full path to find and use the class that implements the activity.

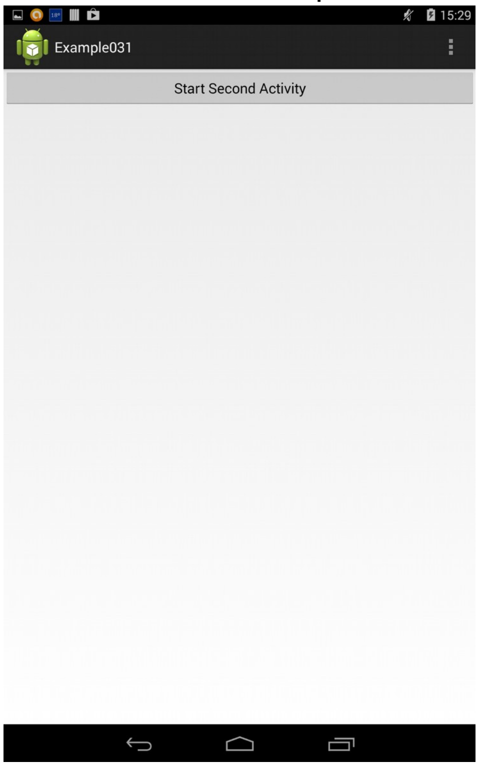
- The label attribute is a string that will appear to the right of the icon in the action bar at the top of the activity. This label can be different for each activity but the suggestion is that you keep this short and informative. It should give a clear idea for what the current activity represents and the task it is performing.
- The intent filter tags we will ignore for now. Take note that one and only one activity must have the intent filter for the "android.intent.action.MAIN" coming from the launcher. This specifies to android the activity that should be constructed and started when the application is launched for the first time.

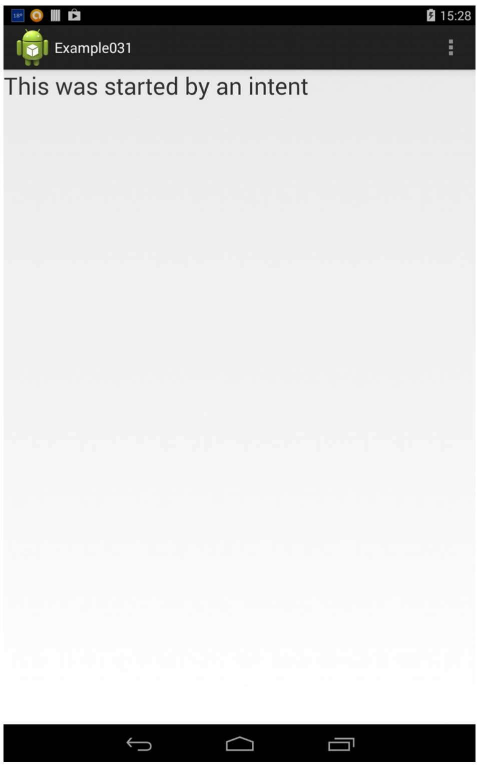
08) add these activity tags into AndroidManifest.xml after the activity tags for the main activity (good place to compile and run)

```
<activity
    android:name=".SecondActivity"
    android:label="@string/app_name"
>
</activity>
```

Explaination:

- Here we are adding our second activity into the application tags.
- All activities should have a name and a label at a bare minimum. Adding in these tags for the SecondActivity class means that when the intent is fired for starting the second activity android will see that it has been declared in the manifest and will subsequently launch it.
- Running this code will get you the screenshots you see below, where the application is in the initial state (left) and the button has been pressed and has launched our second activity (right) if you hit the back key on the second activity it will return you to the first activity. This is because of the link that we set up in our intent





This was started by an intent

object all the way back in step 06.

Example032: using an explicit intent to return a result from an activity you just started

Sometimes when you wish to start another activity you may want to retrieve one or more results from it. In this example we will explore how to return results through the intent system. The reason this is done is that it is possible that an activity may have been started by one of many different activities. In this example we will show the simplest case in which our first activity will start the second on the click of a button. The first activity will attach a single integer value to an intent which is received by the second activity. Should the user click the button on the second activity then the counter will be incremented and passed back to the first activity wherein the new value will be copied across.

01) start a new android project

02) add the following strings to strings.xml

```
    <string name="btn_text">Click for new activity</string>
    <string name="btn_accept">Dismiss and update counter</string>
```

03) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
      <Button
          android:id="@+id/btn"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/btn_text"
      />
</LinearLayout>
```

04) create a new file in res/layout called activity_second.xml and give it the following XML

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical"
>
    <TextView
        android:id="@+id/tv"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:textSize="30sp"
      android:text="@string/hello_world"
    />
    <Button
      android:id="@+id/btn_accept"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/btn_accept"
    />
</LinearLayout>
```

05) create a new source file called SecondActivity.java and give it the following shell code

```
package com.example.example032;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class SecondActivity extends Activity {

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

      @Override
      public boolean onCreateOptionsMenu(Menu menu) {
            // Inflate the menu; this adds items to the action bar if it is
            // present.
            getMenuInflater().inflate(R.menu.main, menu);
            return true;
      }

      @Override
      public boolean onOptionsItemSelected(MenuItem item) {
            // Handle action bar item clicks here. The action bar will
            // automatically handle clicks on the Home/Up button, so long
            // as you specify a parent activity in AndroidManifest.xml.
            int id = item.getItemId();
            if (id == R.id.action_settings) {
                  return true;
            }
            return super.onOptionsItemSelected(item);
      }
}
```

06) add this activity section to the AndroidManifest.xml file to enable the second activity

```
        <activity
            android:name=".SecondActivity"
            android:label="@string/app_name"
        >
        </activity>
```

07) add the following private fields to the MainActivity class

```
      // private fields of the class
      private Button btn;
      private int count;
```

Explaination:

- Even though we are passing data between two activities we will still maintain a full copy of the count value here for subsequent starts of the second activity.

08) add the following code to the end of the onCreate() method of the MainActivity

```
// initialise the count to zero
count = 0;

// add a listener to the button that will launch the second activity and
// will attach the count to the intent
btn = (Button) findViewById(R.id.btn);
btn.setOnClickListener(new OnClickListener() {
      // overridden method to handle the button click
      public void onClick(View v) {
            Intent intent = new Intent(MainActivity.this, SecondActivity.class);
            intent.putExtra("count", count);
            startActivityForResult(intent, 16);
      }
});
```

Explaination:

- The counter is initialised to zero when the application is first started. But the main interest here is the onClick() method of the click listener. In this method we create an explicit intent as we did in the previous example making an explicit connectio between MainActivity and SecondActivity. The difference here however, is that a value has been added to the intent through the use of the putExtra() command.
    - Extras in intents work on a key-value system (like a hash table) The key (first argument to putExtra) can be an arbitrary string, while the value is any number/string/serialisable object
- Finally instead of calling startActivity() as we did in the previous example we call startActivityForResult() which states to Android that when the started activity finishes we expect to recieve an intent from the finished activity with data attached to it.
    - Note that this takes two arguments. The first is the intent object which will state which activity is being started. While the second is a request code. As an activity may start multiple different activities the request code is used to identify where the result came from.

09) add the following method into the MainActivity class

```
// overridden method that will be called whenever an intent has been returned
from
// an activity that was started by this activity.
protected void onActivityResult(int request, int result, Intent data) {

      // check the request code for the intent and if the result was ok. if both
      // are good then take a copy of the updated count variable
      if(request == 16 && result == RESULT_OK) {
            count = data.getIntExtra("count", 0);
            Log.i("MainActivity", "count is " + count);
      }
}
```

Explaination:

- The onActivityResult() method is called for you automatically by Android whenever an activity that was started by startActivityForResult() has finished and is returning a result to which ever activity started it.
- As an activity can start more than one new activity you will need to distinguish the request code for each activity. It is highly recommended that you have a unique request code for each and every activity in the application.
- This method takes three arguments
  - The first is the request code that was attached to the activity that was started. You must compare this value to known request codes to determine which activity has returned a result.
  - The second argument is the result code. Generally you will be looking for one of two values here: RESULT_OK or RESULT_CANCELED. RESULT_OK is used whenever the user hits an accept button on an activity. It is usually ment as the user is happy with the changes or the data they have entered and your application should update itself to reflect those changes. In the case of RESULT_CANCELED you should revert your application to its previous state or not carry out the action that you were expecting to carry out. This value is usually returned when the user hits a cancel/no button or hits the hardware/software back key on their device.
  - The last argument is then the intent with associated data that is returned from the activity.
  - Most of the time this method will be written in the form of an if-else-if ladder that will do comparisons on both the request code and the result code.
- In this example we are checking if the result came from second activity. If it has and the result is ok then we will take a copy of the count value that is associated with the intent of data that we have recieved.

10) add the following fields to the SecondActivity class

```
// private fields of the class
private int count;
private TextView tv;
private Button btn_accept;
```

Explaination:

- keeping a copy of the count as we will need to recieve it in the onCreate() method but it will be updated in the listener.

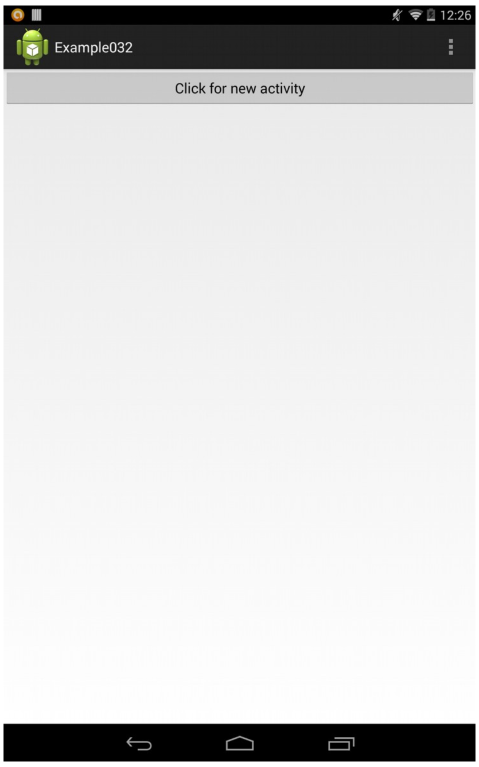
11) add the following code to the end of the onCreate() method of SecondActivity (good time to compile and run)

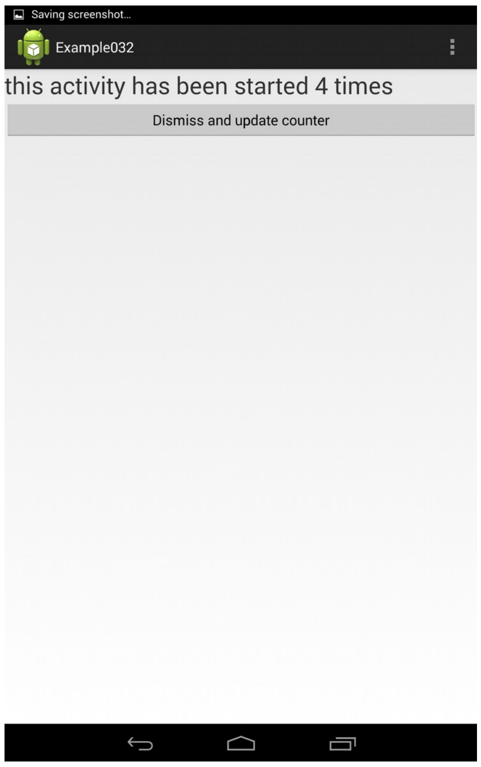
```
// get the intent that started this activity and extract the count from
// it
Intent intent = getIntent();
count = intent.getIntExtra("count", 0);

// grab the textview from the layout and update the text to show the new
// count value
tv = (TextView) findViewById(R.id.tv);
tv.setText("this activity has been started " + count + " times");

// get the button and attach a listener that will update the counter and
// will dismiss this activity
btn_accept = (Button) findViewById(R.id.btn_accept);
btn_accept.setOnClickListener(new OnClickListener() {
	// overridden on click method to return a result to the starter of this
	// activity
	public void onClick(View v) {
		Intent result = new Intent(Intent.ACTION_VIEW);
		count++;
		result.putExtra("count", count);
		setResult(RESULT_OK, result);
		finish();
	}
});
```

Explaination:

- In the first line the getIntent() method will return us the intent object that was used to start this activity. From this we can pull our count value by using the getIntExtra() method.
    - The first argument is the key of the value we wish to retrieve. The second argument is a default value to return should the key not exist in the intent.
    - The onClick() method of the click listener is interesting however. In this block of code we generate a new intent but instead of specifying classes like an explicit intent we are specifying an action. The action usually represents what should be done with the data that is attached to the intent. Intent.ACTION_VIEW is considered a catch all case that you should default to if there is no action that matches your data.

   - We then increment the value of the counter before adding it to the intent object that we have just created.
   - Finally using the setResult() method we state to Android that when this activity finishes it should return a result code of RESULT_OK, and the data for this return is stored in the result object.
      - Note that you do not need to specify a request code as part of the result. Android will handle this automatically.
- Finally we will call the finish method that will push the activity into its paused, stopped, and destroyed states before returning to the activity that started this activity. The result will be returned to the activity that started this activity through the onActivityResult method.
- If you run this code you will see the two screenshots below where we see the main Activity (left) and the second activity after it has been started and dismissed four times





this activity has been started 4 times

Example033: Using an implicit intent to ask Android to show a URL and also a GPS location on a map.

Up to this point we have used explicit intents that will link from one activity to another within the same application. This is one use of the intent system. There is a more powerful version of intents in the form of implicit intents. Say for example your application may require users to visit a web site, or you may want to show them a point on a map of an interesting place. In normal cases your application would implement or embed its own web browser or representation of a map. But this is repetative and wasteful as there are already quite good web browsers and map applications installed on a user's device, wouldn't it be better to use those instead?

With an implict intent we can ask Android to use an application for rendering websites or showing a map. The power of this system is that you can take advantage of the applications on the user's device for handling different forms of data. In this example we will show how an implict intent can be used to launch a web browser to show the griffith website and also to launch the maps application to show the location of griffith college dublin itself.

01) start a new android project

02) add the following strings into strings.xml

```
    <string name="btn_url_text">Load Griffith website</string>
    <string name="btn_maps_text">Show Griffith on map</string>
```

03) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
    <Button
      android:id="@+id/btn_url"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/btn_url_text"
    />
    <Button
      android:id="@+id/btn_maps"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/btn_maps_text"
    />
</LinearLayout>
```

04) add the following fields to the MainActivity class

```
      // private fields for the class
      private Button btn_url;
      private Button btn_maps;
```

05) add the following code to the end of the onCreate() method of MainActivity (good time to compile and run

```
// pull all the buttons from the XML
btn_url = (Button) findViewById(R.id.btn_url);
btn_maps = (Button) findViewById(R.id.btn_maps);

// attach a listener to the url button that will launch the griffith website
btn_url.setOnClickListener(new OnClickListener() {
      // overridden method that will launch an implicit intent that will load
      // up the griffith website in a browser
      public void onClick(View v) {
            Intent intent = new Intent(Intent.ACTION_VIEW, 
Uri.parse("http://www.griffith.ie/"));
            startActivity(intent);
      }
});
```

Explaination

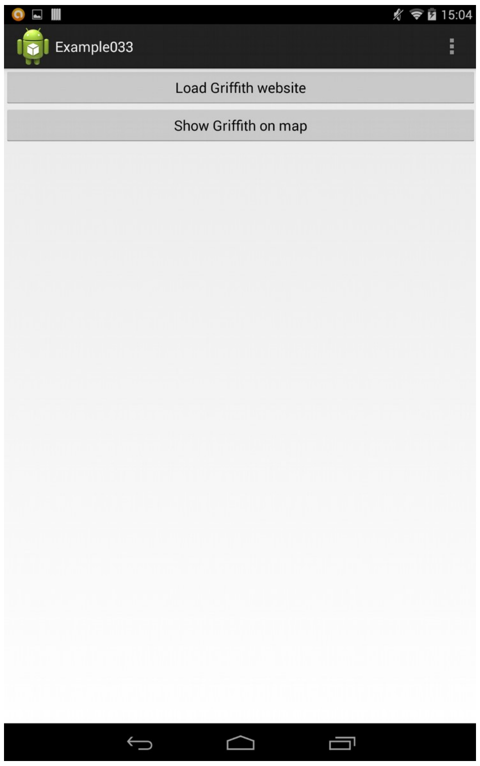
- This onclick listener will launch the default web browser on an Android device and will provide it the URL to the Griffith website
- Running the application now will get you the first two screenshots where we show the initial state of the application (left) and the result of clicking the website button (middle) where my firefox application has opened up the griffith website.
- An implicit intent states to Android that the user has a particular action to perform on a piece of data. Android will then do its best to find an application that can handle that request.
    - Note that not all actions and data will resolve. As good practice you should check if the intent will resolve before starting it
    - However, in the case of websites and maps there is no need to check as all devices must have both installed by default.
- This form of intent takes two parameters. The first is the type of action that you wish to perform on the data. As before in most cases ACTION_VIEW will suffice for the majority of cases.
- The second parameter is the data that we wish to resovlve. All data must be parsed into a URI object, hence we use the Uri.parse method which will take the data and will convert it to a URI object. This is to provide a unified representation of data attached to actions within an intent.
- When the button is clicked one of two things will happen. If a default web browser has been set on the device (firefox in my case) it will launch the browser immediately and will show the URL. If there is no default selected but there are multiple browsers installed then a small dialog box will pop up asking you to select a browser and if you wish set it as the default. Once the browser is selected then it is launched with the URL and shown to the user.

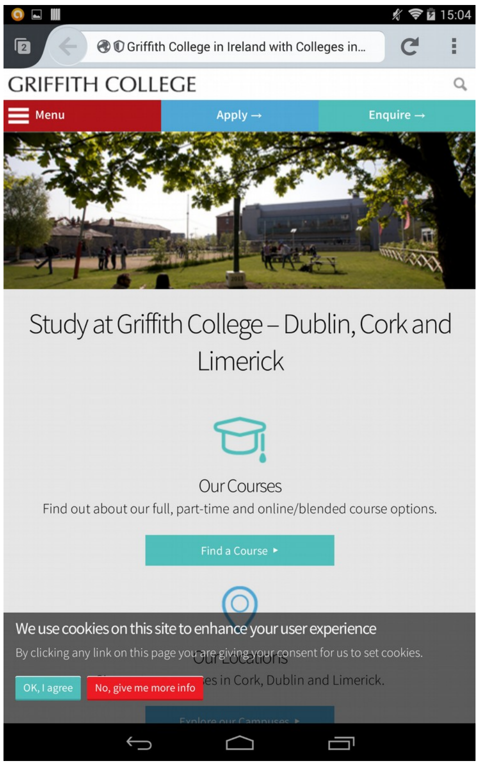
06) add the following code to the end of the onCreate() method of MainActivity (good time to compile and run)

```
// attach a listener to the maps button that will launch google maps and will
// place a pin on griffith college dublin
btn_maps.setOnClickListener(new OnClickListener() {
      // overridden method that will launch an implicit intent that will load
      // up google maps and place a pin for griffith college
      public void onClick(View v) {
            Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("geo:0,0?
q=53.33086,-6.27880"));
            startActivity(intent);
      }
});
```

Explaination

- Here we have a similar listener to the one in 05 above except in this case we are asking android to show us a latitude, longitude point on a map.
- Note the format of the information in the URI.
  - Geo:0,0 specifies that the map should be centered around the point that is provided at the end of this URI.
  - ?q=lat,long specifies that we wish to query the map for this specific latitude and longitute on the map. All maps will understand this specification of a point and will therefore support it and show it correctly.
    - It is possible to specify a string representation of a place name however you will not be guarentted that this will resolve for you.
- If you run the application now you will get the screenshot on the right showing the pin placed on Griffith college in google maps





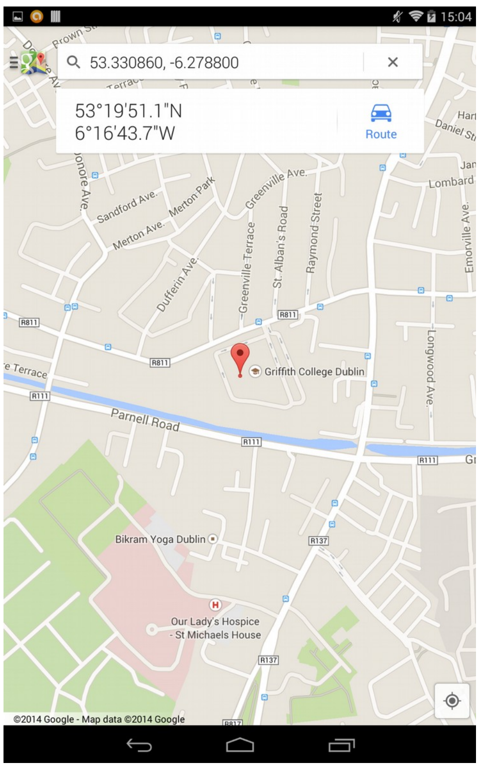
Study at Griffith College – Dublin, Cork and Limerick

Our Courses

Find out about our full, part-time and online/blended course options.

We use cookies on this site to enhance your user experience

By clicking any link on this page you are giving your consent for us to set cookies.



Example034: using an implicit intent to launch the camera to take a picture and use it in your application.

In the previous example we showed how to use an implict intent to view a website and also launch the google maps application. In this example we will show how to use an implicit intent to launch the camera application and take a photo for us. When the photo comes back we will set it on an image view inside of our main activity. This shows us another very powerful aspect of the implict intent system. It is entirely possible to launch activities from other applications and recieve results back from them. In this example we will make a simple main activity that will have a button for launching the camera and taking a photo. If the user accepts the photo then it will be displayed on our activity.

01) start a new android project

02) add in the following strings to strings.xml

```
    <string name="btn_get_pic_text">Take a picture</string>
    <string name="iv_view_text">Pic taken by camera</string>
```

03) replace all XML in activity_main.xml with the following XML

```
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>
      <Button
            android:id="@+id/btn_get_pic"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/btn_get_pic_text"
            android:layout_alignParentTop="true"
      />
    <ImageView
            android:id="@+id/iv_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_below="@id/btn_get_pic"
            android:src="@drawable/ic_launcher"
            android:contentDescription="@string/hello_world"
            android:adjustViewBounds="true"
      />
</RelativeLayout>
```

Explaination:

- note in the ImageView there is an attribute called adjustViewBounds which is set to true. This is to tell the ImageView to adjust the size of the image such that it will fit within the boundaries of the ImageView as defined by the RelativeLayout.

04) add the following fields to the MainActivity class

```
      // private fields of the class
      private Button btn_get_pic;
      private ImageView iv_view;
      private Uri uri_new_image;
```

Explaination:

- fields for the button, image view and a URI. We need a URI here to give us a full reference to the image that will be saved by the camera onto the SD card. We will use this URI to set the image view

05) add the following code to the end of the onCreate() method of the MainActivity class

```
// pull the button from the layout and attach a listener that will launch
// an intent for the camera when it is clicked
btn_get_pic = (Button) findViewById(R.id.btn_get_pic);
iv_view = (ImageView) findViewById(R.id.iv_view);
btn_get_pic.setOnClickListener(new OnClickListener() {
      // overridden method that will launch an intent for the camera
      public void onClick(View v) {
            Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
            File file_new = new
File(MainActivity.this.getExternalFilesDir(null), "test.jpg");
            uri_new_image = Uri.fromFile(file_new);
            intent.putExtra(MediaStore.EXTRA_OUTPUT, uri_new_image);
            startActivityForResult(intent, 100);
      }
});
```

Explaination:

- As before this is the standard adding a click listener to the button that is in our layout. The interesting part is the setup of the intent for the camera activity.
- The first line shows another way of initialising an intent. Here we are providing a string that lists an action that the user wants to perform. In this case we are asking the android system to capture an image from the camera. As before with implicit intents Android may ask the user which camera app they wish to use to capture the image.
- In the second line we are declaring the path to the external data storage maintained on the SD card for this app. Every application will have an external data storage directory on an android device. If you look at the SD card filesystem of your android device you will see a directory in there called Android with a data directory inside, in that data directory you will

see directories with fully qualified Java package names. Each of these directories can have one or more subdirectories. In this case my package is called com.example.example034 which means when the image is saved it will have a full path of /Android/data/com.example.example034/files/test.jpg

  - The first call to the method for getting the external files dir will return the string "/Android/data/com.example.example034/files/" while the second argument will be appended to this to make the full absolute path for the file.

- In the third line we will generate a URI from the file defined on the second line. We then attach this URI to the intent in the fourth line using the putExtra() method note that we use a constant defined in the MediaStore class. Using this constant will enable the camera application to find the URI attached to the intent and thus will use that to determine where to store the file on disk.
- After this we start an activity for a result like we did in the previous examples.

06) add the following method to the MainActivity class

```
// overridden method that will get a result from the activity that we started
// for the camera
public void onActivityResult(int request_code, int result_code, Intent data) {
      // if we get an ok result from the camera we should display the image in
      // the image view
      if(request_code == 100 && result_code == RESULT_OK) {
            BitmapFactory.Options options = new BitmapFactory.Options();
            Bitmap image = BitmapFactory.decodeFile(uri_new_image.getPath(),
options);
            if(image == null) {
                  Log.d("MainActivity::onActivityResult", "failed to load
image");
                  return;
            }
            iv_view.setImageBitmap(image);
      }
}
```

Explaination:

- As before we use the onActivityResult method to pick up the result from the camera activity that we started in the click listener from the button. As before we also check the request code and the result code to ensure that we got the correct result. In this case RESULT_OK means that the camera has captured an image and has written it to the file that was earlier specified in the Uri of step 05.
- Once the image has been written we have to generate a bitmap object for display on the image view. We do this by using a BitmapFactory object with BitmapFactory.Options class. The factory object will do all the heavy lifting of converting the JPEG image into a full RGB image for display on

the image view. Interacting with the Bitmap class directly is not recommended.

- Note that we do not set any explicit options on the bitmap as in most cases the defaults will suffice.
- PS if the image does not show on your imageview it may be because you have an older device that cannot handle large image sizes. In this case you should downsample the image by setting options.inSampleSize = 2 which means that only every second pixel of the image will be sampled (reduces the image to a quarter of its size)

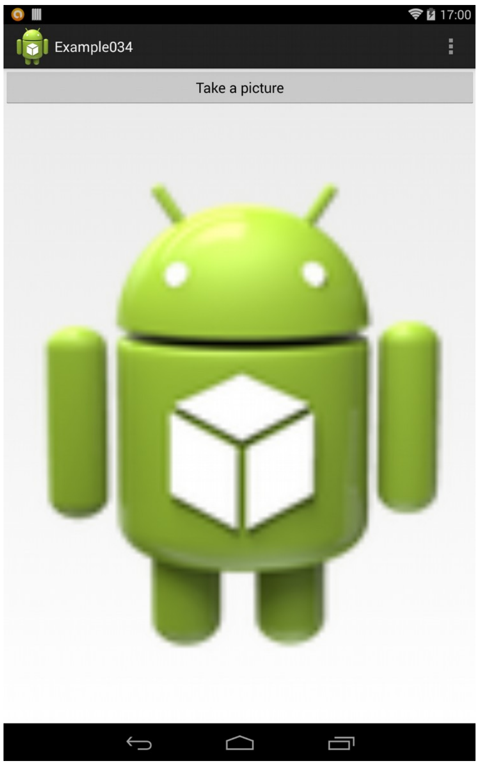
- After this we use the decodeFile method of the BitmapFactory with the Uri and the Options to tell the BitmapFactory class where to find the file and how to decode it.
  - There is a possiblilty that the image may not load in which case the following if statement will print a log message and will exit the method immediately.
- Finally we set the bitmap on the image view so it is displayed.

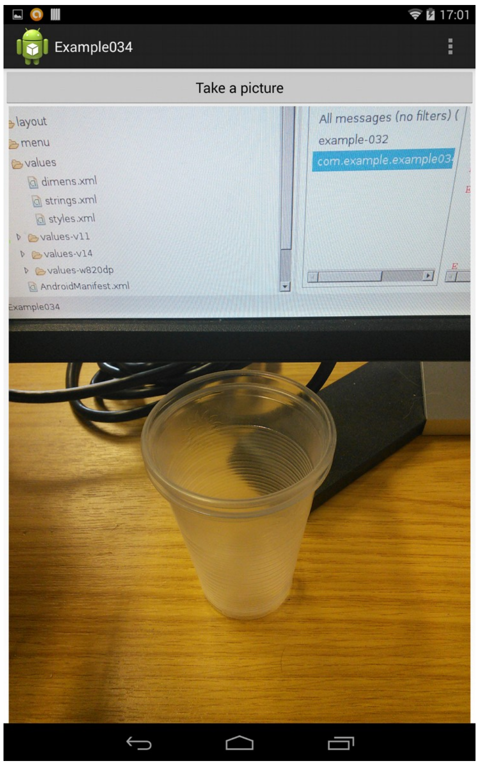
07) add in the following permissions to the MainActivity class (good time to compile and run)

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Explaination:

- These give your application the ability to read and right to the external storage (SD card) if you fail to include these permissions then your application will throw an exception as soon as it tries to read or write to/from external storage.
- Running this application will get you the screenshot below showing the application in its initial state (left) and after an image has been captured by the camera (right)


17:00
Example034
Take a picture


17:01
Example034
Take a picture

Example035: How to determine if the device is in portrait mode or landscape mode.

One task that you will find common to Android applications is rearranging the layout depending on whether the user's device is in portrait or landscape mode. Android provides a mechanisms for you in which you can detect if the device is in either mode. In this simple example we will show how to query the device for this information and adjust our display based on it.

01) start a new android application

02) replace all XML in activity_main.xml with the following XML

```
<FrameLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
>
      <TextView
          android:id="@+id/tv"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="@string/hello_world"
            android:textSize="30sp"
      />
</FrameLayout>
```

03) add the following fields into the MainActivity class

```
      // private fields of the class
      private TextView tv;
```

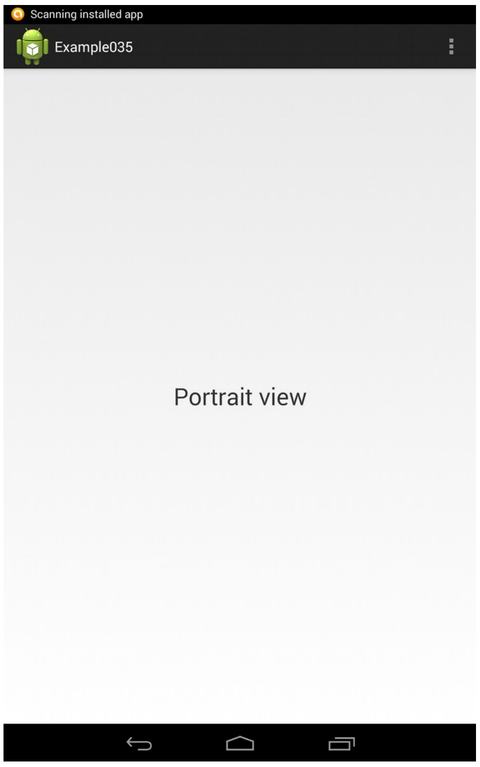
04) add the following code to the end of the onCreate() method of the MainActivity class (good time to compile and run)

```
            // pull the textview from the xml
            tv = (TextView) findViewById(R.id.tv);

            // query the configuration of the device to get the orientation
            // from this report to the user what orientation the device is
            // in
            Configuration c = getResources().getConfiguration();
            if(c.orientation == Configuration.ORIENTATION_PORTRAIT)
                  tv.setText("Portrait view");
            else if (c.orientation == Configuration.ORIENTATION_LANDSCAPE)
                  tv.setText("Landscape view");
```

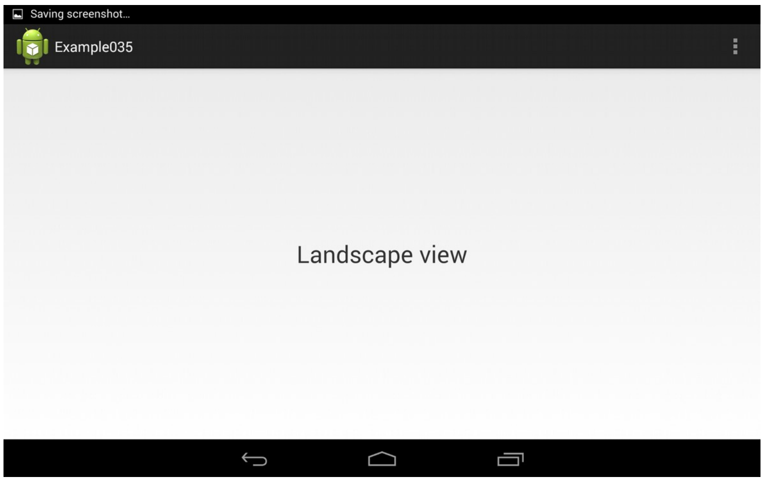
Explaination

- Here we pull the text view from the XML but afterwards we pull a Configuration object from the Resources that are attached to this activity. The Configuration object contains a lot of information about the dimensions and the hardware of the device. But for the purposes of this application we are only interested in the orientation.

- The orientation will return one of many values for the orientation of the device however we are only interested in a couple of these values.
    - Configuration.ORIENTATION_PORTRAIT states that the device is in portrait mode (i.e. the longest dimension of the device is the vertical dimension)
    - Configuration.ORIENTATION_LANDSCAPE states that the device is in landscape mode (i.e. the longest dimension of the device is the horizontal dimension)
- By querying these values you can then adjust your layout to reflect the new orientation
- NOTE that when a device switches from portrait to landscape and vice versa the entire activity will be paused, stopped, destroyed, and then recreated in that order. Thus it is recommended that you have seperate layout files for landscape and portrait mode to make your application handle both cases nicely.
- Running this application with get you the screenshots you see below where the device is in portrait and landscape mode respectively.



Portrait view



Landscape view

Example036: using the GPS sensor to determine where the device is located in the world.

One of the most useful aspects of any android device is that they come with an array of sensors that you can use to collect data about the environment. One of these sensors is the GPS sensor which will help you to locate the user's device on earth's surface. Uses for the GPS include navigation from one place to another, Geotagging photos, social media posts, and making traces for open source maps amongst others. In this example we will show how to register a listener for GPS coordinates and also how to react to the GPS information that comes into the device.

01) start with a new android project

02) add the following strings to strings.xml

```
    <string name="tv_lat_text">Latitude: N/A</string>
    <string name="tv_long_text">Longitude: N/A</string>
```

03) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
    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/tv_lat"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="30sp"
            android:text="@string/tv_lat_text"
      />
      <TextView
            android:id="@+id/tv_long"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="30sp"
            android:text="@string/tv_long_text"
      />
</LinearLayout>
```

04) add the following fields to the MainActivity class

```
      // private fields of the class
      private TextView tv_lat, tv_long;
      private LocationManager lm;
```

Explaination:

- The LocationManager class is necessary for obtaining any kind of information from the GPS device. There are multiple ways in which a location can be provided either by using a GPS sensor (the GPS_PROVIDER) or by using the location of the nearest wifi point or cell tower (the NETWORK_PROVIDER)

- The LocationManager class abstracts these details and will only provide us with Location objects (latitude, longitude based) that will indicate the position of the device regardless of which provider is used.

05) add the following method to the MainActivity class

```
// private method that will add a location listener to the location manager
private void addLocationListener() {
      lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 10000, 5, new
LocationListener() {

            @Override
            public void onLocationChanged(Location location) {
                  // the location of the device has changed so update the
                  // textviews to reflect this
                  tv_lat.setText("Latitude: " + location.getLatitude());
                  tv_long.setText("Longitude: " + location.getLongitude());
            }

            @Override
            public void onProviderDisabled(String provider) {
                  // if GPS has been disabled then update the textviews to
reflect
                  // this
                  if(provider == LocationManager.GPS_PROVIDER) {
                        tv_lat.setText(R.string.tv_lat_text);
                        tv_long.setText(R.string.tv_long_text);
                  }
            }

            @Override
            public void onProviderEnabled(String provider) {
                  // if there is a last known location then set it on the
                  //textviews
                  if(provider == LocationManager.GPS_PROVIDER) {
                        Location l =
lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);
                        if(l != null) {
                              tv_lat.setText("Latitude: " + l.getLatitude());
                              tv_long.setText("Longitude: " + l.getLongitude();

                        }
                  }
            }

            @Override
            public void onStatusChanged(String provider, int status, Bundle
extras) {
            }

      });
}
```

Explaination:

- As the code for adding in a location listener is a bit long we have split it out into a seperate method so we can discuss it in detail.
- If you wish to add a location listener to your application you have to use the requestLocationUpdates() method of the LocationManager class. This takes four arguments
    - The first argument specifies which provider you are using to get your network locations. In this case we are telling the location manager to use the GPS sensor as the source for location data for our listener.
    - The second argument specifies the time in milliseconds between GPS updates to this application. 10,000 specifies that if there has been no update made in 10 seconds then send another location value to this listener. The shorter you make this value the quicker you will recieve more updates improving accuracy at the cost of shorter battery life. If you make this longer you will get the opposite effects
    - The third argument specifies in the minimum distance between updates. In this case a value of 5 specifies that if the user has moved more than 5 meters then update the GPS value of this listener. As before the smaller you make this number the accuracy increases at the cost of battery life.
    - The final argument specifies the listener that is to be registered with the LocationManager. We define an anonymous class here that will implement our listener
- When you implement a LocationListener you are required to provide implementations for four methods that are defined by the LocationListener interface.
    - The first of these methods is onLocationChanged(): This takes in a single Location object, which you can query to get the latitude and longitude values of the current location of the device. In this example we get the raw latitude and longitude and display their values on the two text fields that we have defined in our layout.
    - The second of these methods is onProviderDisabled(): This method is called whenever the user has disabled either GPS location or network location. When this happens this method will be called for you with the name of the provider that was disabled. The general use for this method is to switch to gathering location from the network if GPS is disabled and to go to some default behaviour if all location is disabled. In this example when the GPS is disabled we show the user a message saying that the latitude and longitude are not available.
    - The third of these methods is onProviderEnabled(): This does the opposite of onProviderDisabled(). It is called whenever the user has enabled a provider. The general use here is to switch from no location to network location and then to GPS location if the user enables them in that way. In the example above we ask for the last known location

of the device if there is one. If there is no last known location a value of null will be returned. This is why we have an if statement to check for this condition. If there is a last known location we display it on the text views.

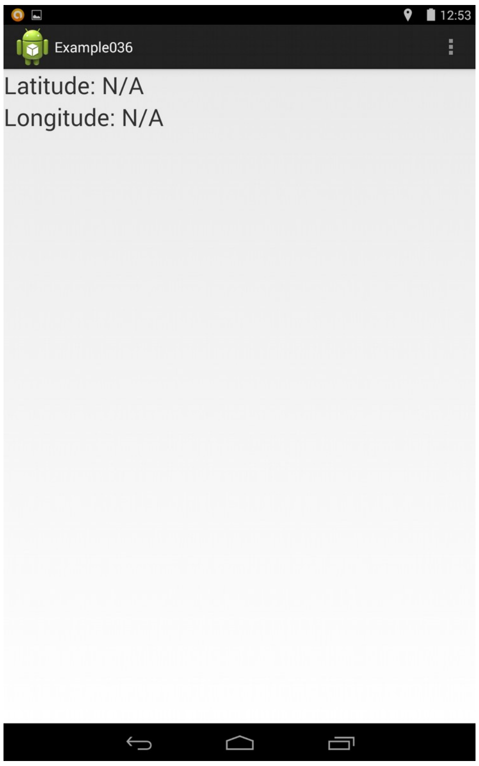
  - The last of these methods is the onStatusChanged(): This will be called whenever the location is temporarily unavailable or becomes available again. The bundle that is the third argument can be queried to determine the number of GPS satellites that are being used to locate the device's position.

06) add the following code to the end of the onCreate() method (good time to compile and run)

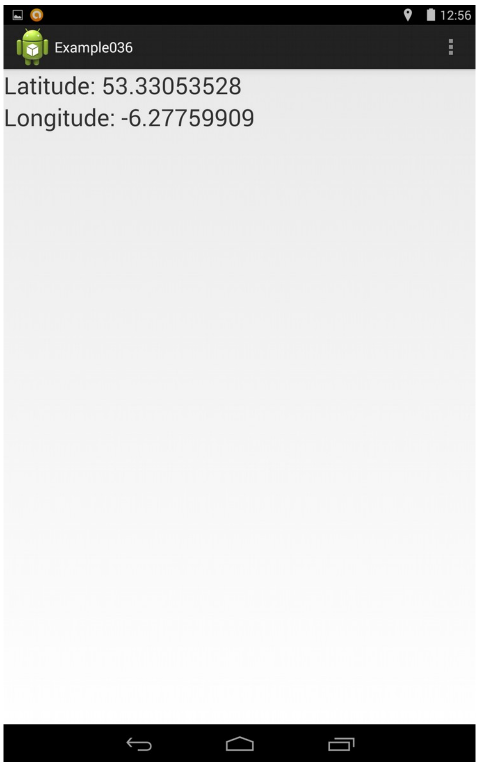
```
// pull the textviews from the xml and get access to the location manager
tv_lat = (TextView) findViewById(R.id.tv_lat);
tv_long = (TextView) findViewById(R.id.tv_long);
lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

// add in the location listener
addLocationListener();
```

Explaination

- To pull the location manager from the Android system so we can attach a listener we must call the getSystemService() method with the location serivce string (defined as a constant in the Context class) this will always be available thus there is no need to check for a null reference.
- The final line calls our method that we defined in 05 above to add a location listener to the location manager class.
- Running this application now will net you the screenshots below. This shows the application after it has been started with no GPS location (left), and after the GPS has obtained a fix and identified the device's position in the world (right).



Latitude: N/A
Longitude: N/A



Latitude: 53.33053528
Longitude: -6.27759909

Example037: Introducing single dimension sensors. How to access them and how to get values from them.

In this example we will introduce the first set of sensors on an android device. Here we will introduce the single dimension sensors (i.e. they provide a single scalar value) Part of the power of an android system is that by using such sensors a smartphone can be turned into a powerful data collection and analysis tool. In this example we will introduce the ambient sensor, light sensor, proximity sensor and athmospheric sensor.

01) start with a new android project

02) replace all the XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv_temperature"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="30sp"
    />
    <TextView
      android:id="@+id/tv_proximity"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="30sp"
    />
    <TextView
      android:id="@+id/tv_light"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="30sp"
    />
    <TextView
      android:id="@+id/tv_pressure"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="30sp"
    />
</LinearLayout>
```

03) In the MainActivity class add in the following fields

```
// private fields for the class
```

```
private TextView tv_temperature, tv_pressure;
private TextView tv_light, tv_proximity;
private SensorManager sm;
```

Explaination:

- The only thing that is new here is the sensor manager.
- The sensor manager is responsible for enumerating all the sensors on an android device and making them available to you in a single neat class. It supports multiple sensors of the same type and is responsible for updating applications that wish to use sensors about any sensor values that have changed and if their accuracy has changed

04) in the MainActivity class add this code to the end of the onCreate() method

```
// get the sensor manager for this activity
sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

// add in a sensor listener for the light sensor
sm.registerListener(new SensorEventListener() {
      // will be called when the accuracy of the sensor has been changed
      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      // will be called when the value of the sensor has changed
      public void onSensorChanged(SensorEvent event) {
      `// the value of the light sensor has changed so update the textview
      `tv_light.setText("Light Sensor: " + event.values[0] + " lux");
      }

}, sm.getDefaultSensor(Sensor.TYPE_LIGHT), SensorManager.SENSOR_DELAY_UI);
```

Explaination:

- The first line is necessary for all applications that wish to get access to the sensors as the sensor manager is the only place where you can get access to the sensors
- The second line is an example of how to attach a listener to a sensor and register for interest in sensor values
- The registerListener() method of the SensorManager object takes three arguments
  - The first is the SensorEventListener object that will listen for events on the given sensor. In this example we follow the general example with all listeners by generating an anonymous class that will implement the SensorEventListener interface
    - The interface requires two methods to be implemented. The first is onAccuracyChanged() which is used should the accuracy of a sensor change this usually this method will be used to switch to a different sensor of the same type (if available) or change sensing

        method or disable the sensor entirely. It is entirely dependant on the application in question

        - onSensorChanged is the more interesting of the two methods. It provides a SensorEvent object which tells you what sensor and sensor type that the event has come from along with its updated data. If you look at the code in this method you will see that the sensor values are stored in an array of floating point numbers which you have direct access to. The reason for this is that it will eliminate a method call (if you access sensor values a lot in a second then eliminating those method calls will save you time and power) and because some sensors are scalar dimensions and others are vector dimensions it provides a consistent way of accessing the values regardless of the sensor type and how many values it provides.

    - The second argument is the type of sensor that you wish to attach this listener to. In this case (and in the other sensor examples) we will usually ask the sensor manager to provide us with the default sensor which will usually be the best one. If there are multiple sensors available of the same type it is possible to query the sensor manager for those sensors and choose between them. One use for using multiple sensors of the same type is to correlate them and reduce error but for most applications this is not necessary.
    - The final argument is the time delay that you wish to set on the sensor. Generally you will use one of four provided constants depending on your situation.
        - SENSOR_DELAY_UI: a sensor refresh rate that is suitable for updating a UI. UI's are not expected to update much in a second so this is a suitable rate for UI displays
        - SENSOR_DELAY_GAME: a sensor refresh rate that is sutable for updating a game. This is a bit faster than the UI rate and will provide about 40 to 50 updates a second
        - SENSOR_DELAY_FASTEST: use with caution. This will provide updates as fast as the sensor will allow but at the cost of battery power. How fast the updates will appear will vary from sensor to sensor.
        - SENSOR_DELAY_NORMAL: The default update rate for sensors that can be used in most cases.

- In this Sensor listener we are using the light sensor. A light sensor is one of the few sensors that you can be pretty sure that all android devices will have. It is used to detect the light levels in the surrounding environment and is measured in a unit called lux. The higher the value indicates a brigher environment while the lower value is used to indicate darkness, one of the most common uses of this sensor is to adjust screen brightness to match the environment

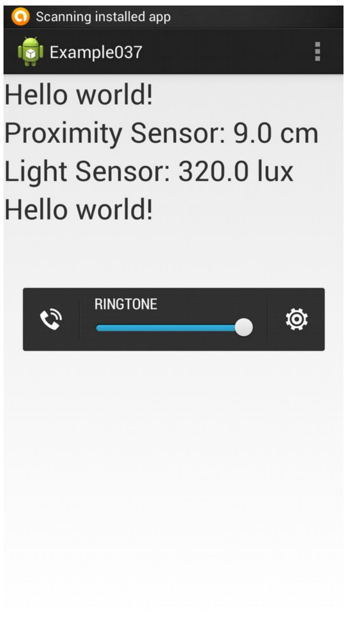
05) add the following code to the end of the onCreate() method in the MainActivity class

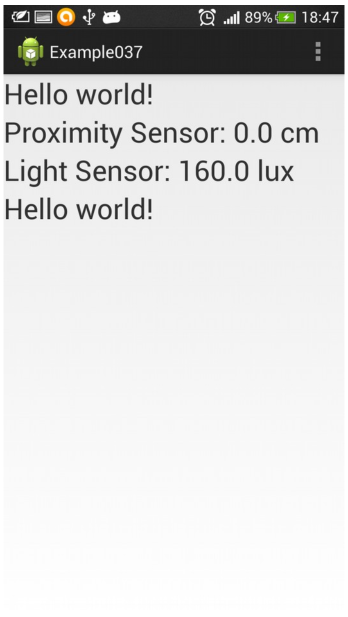
```
// add in a sensor listener for the temperature sensor
sm.registerListener(new SensorEventListener() {
      // will be called when the accuracy of the sensor has been changed
      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      // will be called when the value of the sensor has changed
      public void onSensorChanged(SensorEvent event) {
            // the value of the light sensor has changed so update the textview
            tv_temperature.setText("Temperature Sensor: " + event.values[0] + "
C");
      }

}, sm.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE),
SensorManager.SENSOR_DELAY_UI);

// add in a sensor listener for the temperature sensor
sm.registerListener(new SensorEventListener() {
      // will be called when the accuracy of the sensor has been changed
      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      // will be called when the value of the sensor has changed
      public void onSensorChanged(SensorEvent event) {
            // the value of the light sensor has changed so update the textview
            tv_proximity.setText("Proximity Sensor: " + event.values[0] + "
cm");
      }

}, sm.getDefaultSensor(Sensor.TYPE_PROXIMITY), SensorManager.SENSOR_DELAY_UI);

// add in a sensor listener for the temperature sensor
sm.registerListener(new SensorEventListener() {
      // will be called when the accuracy of the sensor has been changed
      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      // will be called when the value of the sensor has changed
      public void onSensorChanged(SensorEvent event) {
            // the value of the light sensor has changed so update the textview
            tv_pressure.setText("Pressure Sensor: " + event.values[0] + "
millibar");
      }

}, sm.getDefaultSensor(Sensor.TYPE_PRESSURE), SensorManager.SENSOR_DELAY_UI);
```

Explaination:

- Here we are adding in sensors for the other three types that are available. The other sensors that are here are an ambient temperature sensor which measures temperature in decrees celcius. A proximity sensor that measures the distance to an object in from of the device in centimeters and a pressure sensor that will measure athmospheric temperature in millibars.

- Note that not all of these sensors will be present on a device. There are ways and means of querying for the availablility of sensors but they are not covered in these examples. For example if you run this code on a nexus 7 you will only get information back from the light sensor but on a htc one x you will get the light sensor and proximity sensor. The pressure and ambient sensors are rare and not found on many devices
- Note that with android 5 and android wear there will be sensors that will track a persons heart rate and also how many steps they have made. There has been a big push by android devices to implement fitness tracking in recent years.
- Running this code will net you the screenshots below which were taken on a htc one x. As you can see the light sensor and proximity sensors are pretty basic in that they only show one of two values. If you run this on a nexus 7 you will only get the light sensor which in constant light is erratic in the values it presents.


Hello world!
Proximity Sensor: 9.0 cm
Light Sensor: 320.0 lux
Hello world!


Hello world!
Proximity Sensor: 0.0 cm
Light Sensor: 160.0 lux
Hello world!

Example038: Introducing multiple dimension sensors. How to access them and how to get values from them.

In the previous application we discovered the sensors that only provided a single scalar value. However in this example we will show the accelerometer, gyroscope and the magnetometer. Here we will also introduce the concept of virtual sensors which are filtered versions of real sensors. The two virtual sensors we will show here are the gravity sensor and the linear acceleration sensor.

01) start a new android project

02) replace all the XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv_1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="20sp"
    />

    <TextView
      android:id="@+id/tv_2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="20sp"
    />

    <TextView
      android:id="@+id/tv_3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="20sp"
    />

    <TextView
      android:id="@+id/tv_4"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="20sp"
    />
    <TextView
      android:id="@+id/tv_5"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="20sp"
    />

</LinearLayout>
```

03) add the following fields to the MainActivity class

```
// private fields of the class
private TextView tv1, tv2, tv3, tv4, tv5;
private SensorManager sm;
```

04) add the following code to the end of the onCreate() method of the MainActivity class

```
// pull the text views from the xml
tv1 = (TextView) findViewById(R.id.tv_1);
tv2 = (TextView) findViewById(R.id.tv_2);
tv3 = (TextView) findViewById(R.id.tv_3);
tv4 = (TextView) findViewById(R.id.tv_4);
tv5 = (TextView) findViewById(R.id.tv_5);

// get access to the sensor manager
sm = (SensorManager) getSystemService(SENSOR_SERVICE);

// attach a sensor listener to the accelerometer
sm.registerListener(new SensorEventListener() {

      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      public void onSensorChanged(SensorEvent event) {
            // print out the values associated with the sensor
            tv1.setText("accelerometer (ms-2) \n x: " + event.values[0] + " y: "
+ event.values[1] + " z: " + event.values[2]);
      }

}, sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
SensorManager.SENSOR_DELAY_UI);
```

Explaination:

- As in the previous example we need to get access to the SensorManager class to get access to all of the sensors on the device and also to register listeners for individual sensors.
- Note that for scalar and vector sensors the addition of a listener is the same as is the querying of values. In the vector sensor however there will be three values instead of one to query. In most cases values[0] will

represent the X direction, values[1] will represent the Y direction, and values[2] will represent the Z direction

- The first sensor that you see here is an unfiltered accelerometer that will present three acceleration values in each direction. This indicates how quickly the device is chainging its speed. The units here is meters per second squared (ms-2) However when you pull the values from this sensor you may notice that one of the directions will hover around a value of 9.81 even though the device is not moving. Do not worry about this value as it represents the constant force of gravity that pulls everything to earth.
- Most of the time you will not use the accelerometer directly but you will instead use a filtered version that we will see later.

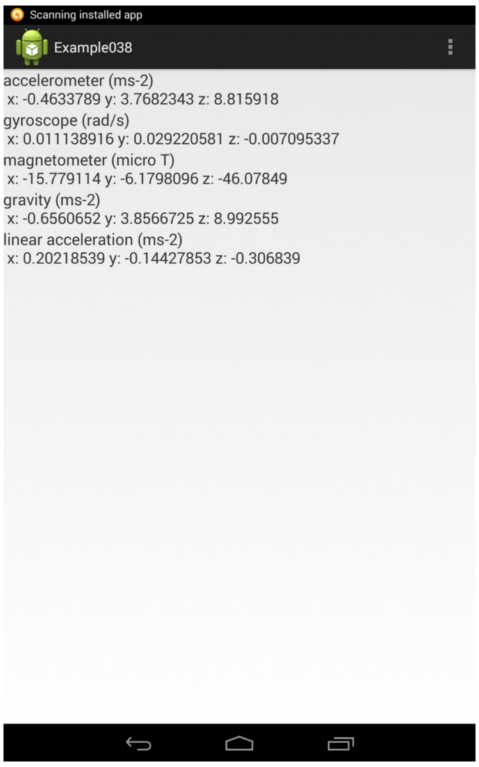
05) add the following code to the end of the onCreate() method of the MainActivity class

```
// attach a sensor listener to the gyroscope
sm.registerListener(new SensorEventListener() {

      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      public void onSensorChanged(SensorEvent event) {
            // print out the values associated with the sensor
            tv2.setText("gyroscope (rad/s) \n x: " + event.values[0] + " y: " +
event.values[1] + " z: " + event.values[2]);
      }

}, sm.getDefaultSensor(Sensor.TYPE_GYROSCOPE), SensorManager.SENSOR_DELAY_UI);

// attach a sensor listener to the magnetic field sensor
sm.registerListener(new SensorEventListener() {

      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      public void onSensorChanged(SensorEvent event) {
            // print out the values associated with the sensor
            tv3.setText("magnetometer (micro T) \n x: " + event.values[0] + " y:
" + event.values[1] + " z: " + event.values[2]);
      }

}, sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
SensorManager.SENSOR_DELAY_UI);

// attach a sensor listener to the gravity sensor
sm.registerListener(new SensorEventListener() {

      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }
      public void onSensorChanged(SensorEvent event) {
            // print out the values associated with the sensor
            tv4.setText("gravity (ms-2) \n x: " + event.values[0] + " y: " +
event.values[1] + " z: " + event.values[2]);
      }

}, sm.getDefaultSensor(Sensor.TYPE_GRAVITY), SensorManager.SENSOR_DELAY_UI);

// attach a sensor listener to the linear acceleration sensor
sm.registerListener(new SensorEventListener() {

      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      public void onSensorChanged(SensorEvent event) {
            // print out the values associated with the sensor
            tv5.setText("linear acceleration (ms-2) \n x: " + event.values[0] +
" y: " + event.values[1] + " z: " + event.values[2]);
      }

}, sm.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION),
SensorManager.SENSOR_DELAY_UI);
```

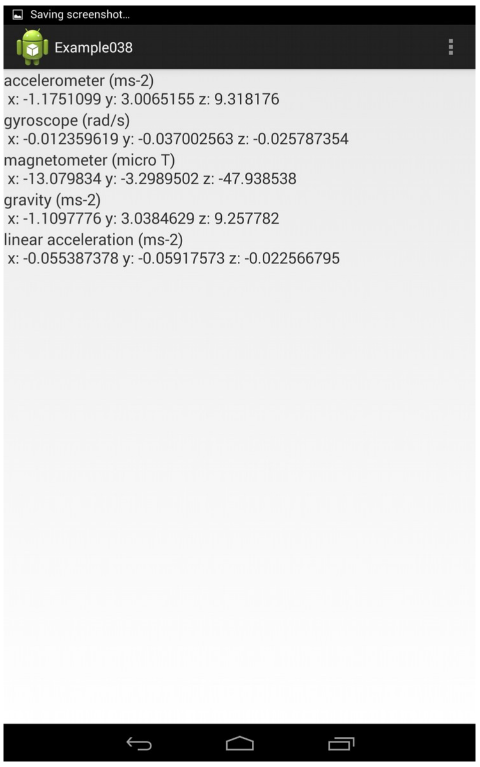
Explaination:

- Here we introduce the other two sensors of the gyroscope and the magetometer, we also introduce the virtual gravity and linear acceleration sensors which are filtered versions of the raw accelerometer.
- The gyroscope is there to measure rotation in the device. Two uses for this is to tell you how fast the device is rotating and also from this we can derive the direction it is facing. It measures rotation around all three axes and the measurements are in radians per second.
- The second sensor you see here is the magnetometer which measures the strength of the earths magnetic field in micro teslas. This is an alternative way in which the direction of the device can be determined and like the other sensors it presents values in all three directions.
- The third sensor you see here is the virtual gravity sensor. In the background what this sensor does is apply a low pass filter over the sensor values to eliminate high frequency changes (i.e rapidly changing accelerations). This is done because gravity is constant and never changes. this takes on the same units as the accelerometer and is presented as meters per second squared (ms-2)
- The fourth sensor you see here is the virtual linear acceleration sensor. In the background this sensor applies a high pass filter to the accelerometer to eliminate low freqency changes (slow changes or constant values) and will eliminate gravity from the accelerometer. This will leave you with the movement of the device as caused by the user. Again this takes on the same values as the accelerometer and will use the same units (ms-2)

- running this code will get you the screenshots that you see below.



```
Scanning installed app
Example038
accelerometer (ms-2)
 x: -0.4633789 y: 3.7682343 z: 8.815918
gyroscope (rad/s)
 x: 0.011138916 y: 0.029220581 z: -0.007095337
magnetometer (micro T)
 x: -15.779114 y: -6.1798096 z: -46.07849
gravity (ms-2)
 x: -0.6560652 y: 3.8566725 z: 8.992555
linear acceleration (ms-2)
 x: 0.20218539 y: -0.14427853 z: -0.306839
```



```
Saving screenshot...
Example038
accelerometer (ms-2)
 x: -1.1751099 y: 3.0065155 z: 9.318176
gyroscope (rad/s)
 x: -0.012359619 y: -0.037002563 z: -0.025787354
magnetometer (micro T)
 x: -13.079834 y: -3.2989502 z: -47.938538
gravity (ms-2)
 x: -1.1097776 y: 3.0384629 z: 9.257782
linear acceleration (ms-2)
 x: -0.055387378 y: -0.05917573 z: -0.022566795
```

Example039: Introducing the rotation vector composite sensor to determine the overall rotation of an android device in the world

So far we have come across sensors and virtual sensors, there is also a third class of sensor called a composite sensor which takes data from two or more different sensor types and combines them together. In this case we will introduce the rotation vector which is a composite sensor that takes a combination of the accelerometer and the gyroscope to figure out a full orientation of an android device in the world. One of the biggest uses for this is to enable compass functionality as well as pitch and roll information.

01) start with a new android project

02) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="20sp"
    />

</LinearLayout>
```

03) add the following fields to the MainActivity class

```
      // private fields of the class
      private TextView tv;
      private SensorManager sm;
      private float rotation_matrix[] = new float[16];
      private float orientation_values[] = new float[4];
```

Explaination:

- Here we have a text view and a sensor manager as before but we have also added in two arrays. The single dimension array of 16 elements describes a 4x4 rotation matrix that is compatible with OpenGL. This will be used to acquire the full rotation of the device from the rotation vector. The second array is a set of orientation values that will be obtained from a conversion of the rotation matrix. The first value in this array will describe a compass bearing, the second will describe an angle of pitch and the third will describe an angle of roll.

- Because we know the size of these arrays well in advance we can initialise them immediately before we ever hit the onCreate() method

04) add the following code to the end of the onCreate() method of the MainActivity class

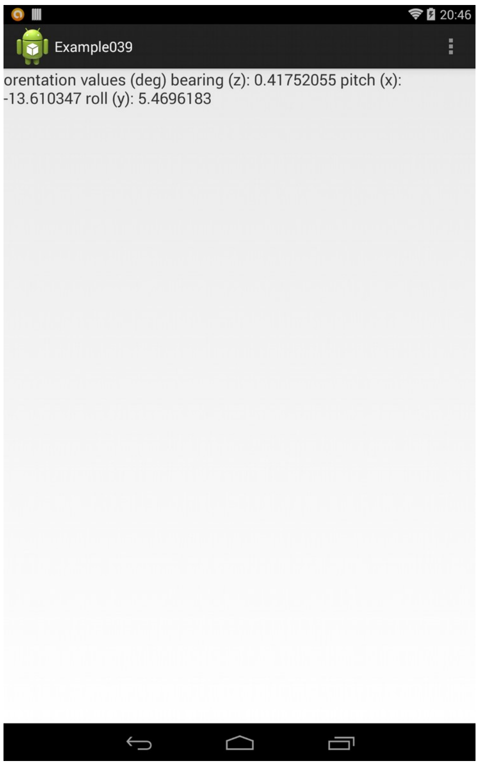
```
// get access to the text view and the sensor manager
tv = (TextView) findViewById(R.id.tv);
sm = (SensorManager) getSystemService(SENSOR_SERVICE);

// attach a sensor to the rotation vector
sm.registerListener(new SensorEventListener() {

      public void onAccuracyChanged(Sensor sensor, int accuracy) {

      }

      public void onSensorChanged(SensorEvent event) {
            // get the new rotation matrix and convert them into orientation
            //values
            SensorManager.getRotationMatrixFromVector(rotation_matrix,
event.values);
            SensorManager.getOrientation(rotation_matrix, orientation_values);

            // as the orientation values are unitless we need to convert them
back
            // into degrees
            orientation_values[0] = (float)
Math.toDegrees(orientation_values[0]);
            orientation_values[1] = (float)
Math.toDegrees(orientation_values[1]);
            orientation_values[2] = (float)
Math.toDegrees(orientation_values[2]);

            // print out the values to the text view
            tv.setText("orentation values (deg) bearing (z): " +
orientation_values[0]
                              + " pitch (x): " + orientation_values[1]
                              + " roll (y): " + orientation_values[2]);
      }

}, sm.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR),
SensorManager.SENSOR_DELAY_UI);
```

Explaination:

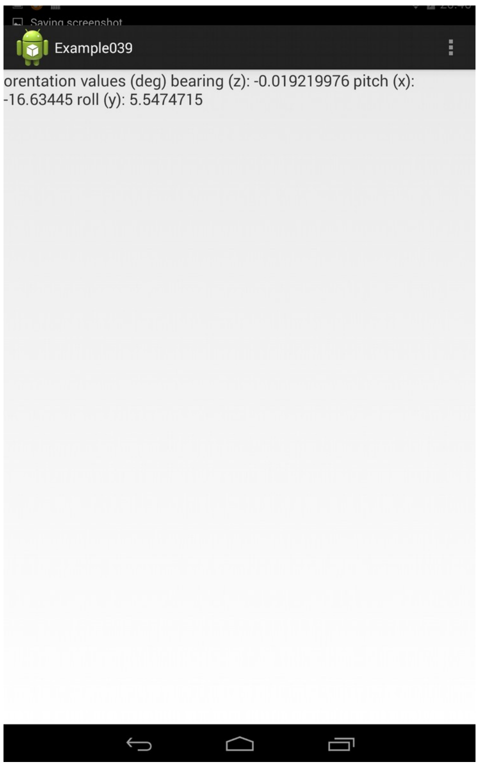
- Here we are adding a listener to the rotation vector as before by using the same method as the last two examples. Except in this case we are listening for the rotation vector composite sensor.
  - The first line of this listener takes the rotation vector and passes it to a helper method that will generate a 4x4 rotation matrix that represents that rotation vector and will store it in the array marked by the variable rotation_matrix. The second line then takes the rotation matrix and converts it into our orientation values.

    - The orentation values represent angles however these values are unitless. Hence why we need the conversion of the Math class to convert them to degrees.

- The way to interpret these values is as follows: imagine your device is lying flat on a table. The z-axis comes from the screen towards the sky and the compass bearing is the rotation around this axis. The second value is the pitch around the x-axis. The x axis emerges from the right side of the device parallel to the table and the ground. The y-axis represents roll of the device and the y-axis comes from the top of the device and moves forward parallel to the table and the ground. Using this it is possible to get the full orientation of the device in the world and the compass is particularly useful if you wish to implement assisted GPS applications where you combine the GPS sensor with the compass to give a more accurate direction.
- Running the code above will net you the following screenshots which will show you all three values in action



orentation values (deg) bearing (z): 0.41752055 pitch (x): -13.610347 roll (y): 5.4696183



orentation values (deg) bearing (z): -0.019219976 pitch (x): -16.63445 roll (y): 5.5474715

Example040: Introducing the SharedPreferences framework. How you persist small pieces of data with the SharedPreferences

In the previous applications we have seen we have manily worked with applications that always initliase to the same state. But in order to make an application truly useful you have to have a way of persisting application state between subsequent runs of the application and to go a step further subsequent runs of various activities. To get around this problem we will introduce the SharedPreferences framework. This gives you access to a simple key-value pair system (similar to that which you used with intents) to store data on the device itself.

01) start with a new android project

02) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="30sp"
    />

    <TextView
      android:id="@+id/tv2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="30sp"
    />

    <TextView
      android:id="@+id/tv3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="30sp"
    />

</LinearLayout>
```

03) add the following fields to the MainActivity class

```
      // private fields of the class
      private SharedPreferences sp_prefs;
```

```
	private boolean some_boolean;
	private int num_starts;
	private float some_float;
```

Explaination

- here we can see the declaration of a few basic data types but also the declaration of a SharedPreferences object.
- The shared preferences object is responsible for giving you access to the preferences for your application and is also responsible for updating them.

04) add the following method to the MainActivity class

```
	protected void onDestroy() {
		super.onDestroy();
	}
```

Explaination:

- We will use this method to save all of our preferences before the activity is killed by android.

05) add the following code to the end of the onCreate() method

```
	// get access to the shared preferences for this application
	sp_prefs = getSharedPreferences("com.example.example040",
Activity.MODE_PRIVATE);

	// pull the values from the shared preferences. here is a good time to
	// use teh defaults  because if the application starts
	// for the first time there will be no preferences there
	// so the default value will be substituted instead
	some_boolean = sp_prefs.getBoolean("some_boolean", false);
	num_starts = sp_prefs.getInt("num_starts", 0);
	some_float = sp_prefs.getFloat("some_float", 0.f);

	// get the textviews from XML and set the values on each
	TextView tv1 = (TextView) findViewById(R.id.tv1);
	TextView tv2 = (TextView) findViewById(R.id.tv2);
	TextView tv3 = (TextView) findViewById(R.id.tv3);
	tv1.setText("some_boolean is: " + some_boolean);
	tv2.setText("num_starts is: " + num_starts);
	tv3.setText("some_float is: " + some_float);
```

Explaination:

- The first line is how we get access to the shared preferences through the getSharedPreferences method which accepts two arguments
    - The first parameter is the name of the preferences that you wish to access. You may have as many of these names as you wish provided they are unique and if at all possible does not clash with other applications.
    - The second parameter specifies the access control on the shared

preferences by stating that we want the shared preferences to have private access it states to android that these shared preferences should only be accessible to this applicaton and no other application. While it is possible to make shared preferences that are accesible to other applications generally you will not do this.

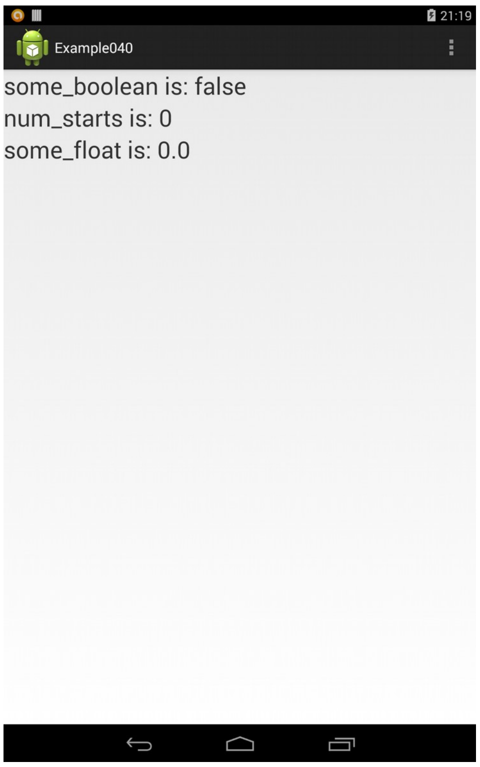
- The next three lines are dual purpose. Here we are pulling values from the shared preferences by their key name however there is a default value provided should that value not exist. Where this is useful is that when you start the application for the first time the shared preferences will not exist so the default values will be used in their place but on the second and subsequent runs it will use the values that are stored in the shared preferences instead.
- Finally we pull the text views from the XML and some messages stating what the values are in the shared preferences.

06) add the following code to the end of the onDestroy() method

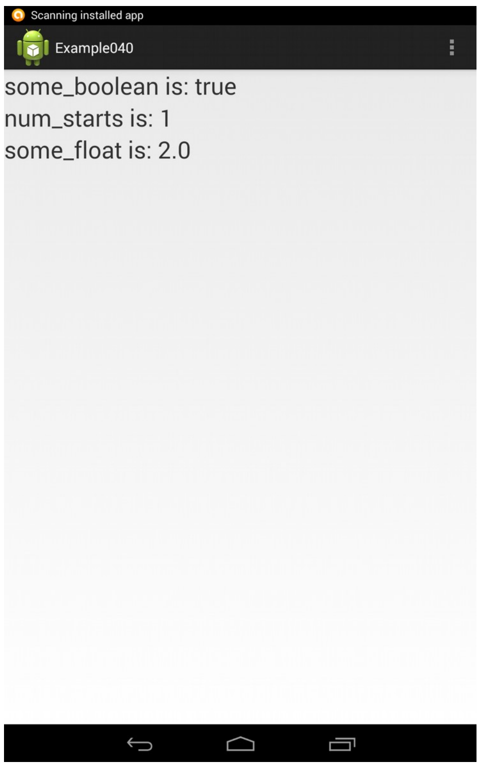
```
// update the values and save them with the shared preferences
some_boolean = !some_boolean;
num_starts++;
some_float += 2.f;

// get access to a shared preferences editor and save the values
SharedPreferences.Editor editor = sp_prefs.edit();
editor.putBoolean("some_boolean", some_boolean);
editor.putInt("num_starts", num_starts);
editor.putFloat("some_float", some_float);
editor.apply();
```

Explaination:

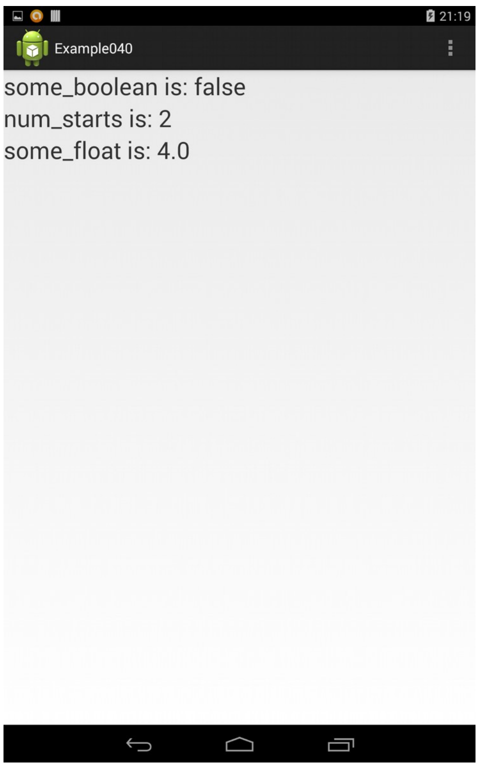
- This is how we are going to save data in our shared preferences. In the first three lines we are just modifying our variables to give them different values for each subsequent run to show that the data is being persisted.
- The next five lines are how you store data in the shared preferences. When you initially look at this you may notice that this follows a similar pattern to a transactioning system for databases. The shared preferences system works in the same way. It will collect all the data to be persisted first (everything between the .edit() command to the .apply() command) and will attempt to persist that data in a single go.
  - We don't have to worry about rollbacks because the shared preferences is always available and rarely fails
- A transaction is started with the edit() method which will give you a SharedPreferences.Editor object to which you have to attach all of your data to be persisted. The data will be persisted when the apply command is called.
- Running this application will give you the three screenshots below that show the first, second and third runs of the application on a device.



```
some_boolean is: false
num_starts is: 0
some_float is: 0.0
```



```
some_boolean is: true
num_starts is: 1
some_float is: 2.0
```



```
some_boolean is: false
num_starts is: 2
some_float is: 4.0
```

Example041: Creating and interacting with an SQLite database.

So far the only way we have seen to persist data in an android device is through the use of the shared preferences framework. However, while convenient the shared preferences is only really useful for saving small pieces of data like program settings. When larger sets of data wish to be stored it is more appropriate to use a file or a database. In this example we will use an SQLite database to store data. An SQLite database works the exact same way as a standard SQL database works, in that you can create and remove tables, add update search and remove data. The only major difference is that an SQLite database does not require a server to work. It is an in process database. The application that opens the database is responsible for running and maintaining that database until the database is closed. Fortunately Android provides a number of helper classes that will implement this functionallity for you. In this example we will show how to create a database and insert, update, and remove rows from the database.

01) start with a new android project

02) in activity_main.xml replace all XML with the following XML

```xml
<LinearLayout
      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/tv"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world"
      android:textSize="30sp"
    />
</LinearLayout>
```

03) create a new file called TestDBOpenHelper.java and give it the following code

```
package com.example.example041;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class TestDBOpenHelper extends SQLiteOpenHelper {
      // constructor for the class here we just map onto the constructor of the
      // super class
      public TestDBOpenHelper(Context context, String name, CursorFactory
factory,
                              int version) {
            super(context, name, factory, version);
      }

      // overridden method that is called when the database is to be created
      public void onCreate(SQLiteDatabase db) {
            // create the database
            db.execSQL(create_table);
      }

      // overridden method that is called when the database is to be upgraded
      // note in this example we simply reconstruct the database not caring for
      // data loss ideally you should have a method for storing the data while
you
      // are reconstructing the database
      public void onUpgrade(SQLiteDatabase db, int version_old, int version_new)
{
            // drop the tables and recreate them
            db.execSQL(drop_table);
            db.execSQL(create_table);
      }

      // a bunch of constant strings that will be needed to create and drop
      // databases
      private static final String create_table = "create table test(" +
                  "ID integer primary key autoincrement, " +
                  "FIRST_NAME string," +
                  "LAST_NAME string, " +
                  "COURSE string" +
                  ")";

      private static final String drop_table = "drop table test";
}
```

Explaination:

- Should you wish to interact with an SQLiteDatabase in android you are required to have a helper class that is responsible for opening/creating/updating the database. The standard way of making your own helper class is to extends the SQLiteOpenHelper class and define a few parameters and functions.

- Note that the static strings at the end of the class are SQL strings for creating and dropping a database
- The first of the methods that you are required to implement is the constructor. The standard way of doing this is to take the same four arguments that are required for the constructor of the SQLiteOpenHelper class and take them in for your constructor and simply pass them through to the constructor of the SQLiteOpenHelper.
    - The first argument is the owning context. Usually this will be the activity that the database is opened in.
    - The second argument is the name of the database that you are trying to open. In this case because we are using an sqlite database this will usually be a filename.
    - The thrid argument allows you to specify a custom CursorFactory class for creating Cursor objects. As you will see later Cursor objects are used to navigate the results of a query on a database. The majority of time the default cursor will suit your needs so the value null usually gets passed here to inidicate to Android that the default CursorFactory is to be used.
    - The final argument is the version of the database you wish to open. The version of the database usually only changes whenever the structure of a database is modified in an upgraded version of your application. All sqlite databases in Android will have a version number attached to them. If Android deems that there is an older version of the database then it will call the onUpgrade method of the SQLiteOpenHelper (discussed a little later) and will try and upgrade the old database to the new one.
- The second of the methods is the onCreate() method which gets called if the database is to be constructed from scratch as it does not already exist. You will be given and SQLiteDatabase object that represents an already open empty database and you are to populate this database with tables and initial data here.
- The third of the methods is the onUpgrade() metthod which gets called if you upgrade your database to a new version. What you are expected to define here is conversions from an earlier database type to a new database type. The conversions here should try and maintain all of the data that currently exists in the database. In the example here we drop all tables and create new ones without caring for the data that is lost.

04) Add the following fields to MainActivity.java

```
// private fields of the class
private TestDBOpenHelper tdb;
private SQLiteDatabase sdb;
```

Explaination

- here we have a reference to our database helper class. And when that helper opens a database for us we will take that database reference and store it in sdb. We will use sdb for performing queries later on

05) Add the following code to the end of the onCreate() method of the MainActivity class

```
// get access to an sqlite database
tdb = new TestDBOpenHelper(this, "test.db", null, 1);
sdb = tdb.getWritableDatabase();
```

Explaination:

- In the first line we are constructing a helper object that uses the activity as the owning context, test.db as the name of the file holding the sqlite database, we dont require a cursor factory for creating cursor objects so the default one will do and we also state that we are using the first version of the database.

06) Add the following code to the end of the onCreate() method of the MainActivity class

```
// add in a few rows to our database
ContentValues cv = new ContentValues();
cv.put("FIRST_NAME", "john");
cv.put("LAST_NAME", "doe");
cv.put("COURSE", "computing");
sdb.insert("test", null, cv);

cv.put("FIRST_NAME", "jane");
cv.put("LAST_NAME", "doe");
sdb.insert("test", null, cv);

cv.put("FIRST_NAME", "jim");
cv.put("LAST_NAME", "doe");
sdb.insert("test", null, cv);
```

Explaination

- This code here is inserting three rows of data into our sqlite database. The content values object works like the shared preferences and the intents that you saw earlier. It is a key-value pair system. It is expected that the keys you use here directly match the names of the columns in the table that you are adding data to. If you change the names to something else your data may not be entered into the database properly.
- To add data to a content values object you use the put method the first argument of which is the key (column name) and the value (value to be stored there) that is to be inserted into a new row of the database table.
- The insert function of the SQLiteDatabase object is then used to insert the new data into a new row in the table. It takes three arguments
  - The first is the name of the table that the row is to be inserted into

  - The second should always be set to null unless you are entering in an empty row into your database. If you are entering in an empty row this should be the name of a database field that can accept the null value to ensure that insertion of the empty row happens correctly
  - Finally the last argument is the content values object that contains all of the data that is to be added to the new row that is to be created.

07) Add the following code to the end of the onCreate() method of the MainActivity class

```
// name of the table to query
String table_name = "test";
// the columns that we wish to retrieve from the tables
String[] columns = {"ID", "FIRST_NAME"};
// where clause of the query. DO NOT WRITE WHERE IN THIS
String where = null;
// arguments to provide to the where clause
String where_args[] = null;
// group by clause of the query. DO NOT WRITE GROUP BY IN THIS
String group_by = null;
// having clause of the query. DO NOT WRITE HAVING IN THIS
String having = null;
// order by clause of the query. DO NOT WRITE ORDER BY IN THIS
String order_by = null;

// run the query. this will give us a cursor into the database
// that will enable us to change the table row that we are working with
Cursor c = sdb.query(table_name, columns, where, where_args, group_by,
                        having, order_by);

// print out some data from the cursor to the screen
TextView tv = (TextView) findViewById(R.id.tv);
String total_text = "total number of rows: " + c.getCount() + "\n";
c.moveToFirst();
for(int i = 0; i < c.getCount(); i++) {
      total_text += c.getInt(0) + " " + c.getString(1) + "\n";
      c.moveToNext();
}
tv.setText(total_text);
```

Explaination:

- Here we show an example of creating a query for an SQLite database in android. The seven variables that you see above are parameters that are required by the query function of the SQLiteDatabase object.
  - The first is the name of the table that you wish to run the query on
  - The second is the list of columns that you wish to select and retrieve from the table.
  - The third is the where clause of the query. Please note that you should not write the word WHERE in this string as it will be automatically added for you.

    - The fourth is the list of arguments that is to be substituted into the placeholders inside the where clause that was generated in the third argument
    - The last three argument are for specifing a group_by, having, and order_by clause in the query like the where clause you should not add in the words GROUP BY, HAVING, ORDER BY into the query as they will be automatically added for you.
    - Generally you will not require all components of the query any ones that you do not require you can give a value of null which will cause the query to ignore those parts entirely.
- Once you run the query command on the database you will get back a Cursor object that will contain a list of results from the database. The Cursor object allows random access so that you do not have to move through rows from start to finish you can visit in any order you wish. In the first line here we tell the Cursor to navigate to the very first row that is in the set of results.
- The for loop that immediately follows then queries how many rows are in the result set and uses that information to loop through each individual row in the results. All we do here is display the results on a text view to the user with the data that was retrieved. We use the moveToNext() method of the cursor to move the cursor to the next row that is in the set of results from the query.

08) Add the following onDestroy() method to the MainActivity class

```
// overridden method that will clear out the contents of the database
protected void onDestroy() {
      super.onDestroy();

      // run a query that will delete all the rows in our database
      String table = "test";
      String where = null;
      String where_args[] = null;
      sdb.delete(table, where, where_args);
}
```

Explaination:

- here we use the onDestroy() method to remove all data from the database. Usually this method would be used to commit all unsaved changes to the database to prevent dataloss before the activity is destroyed. To remove rows you are required to provide a table name, a where clause and where arguments which work in the same way as 07 above.
- If you specify a where clause of null then all rows in the table will be removed.

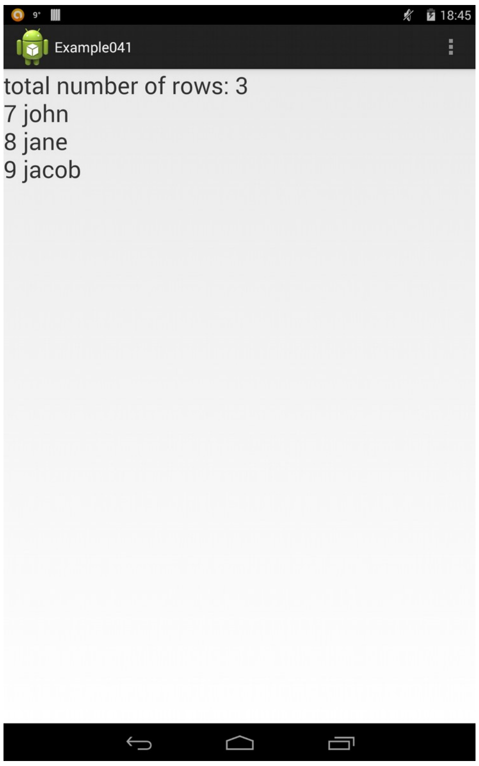
09) Add the following code between the end of part 06 and the start of part 07

```
// run a simple update to change a name
```

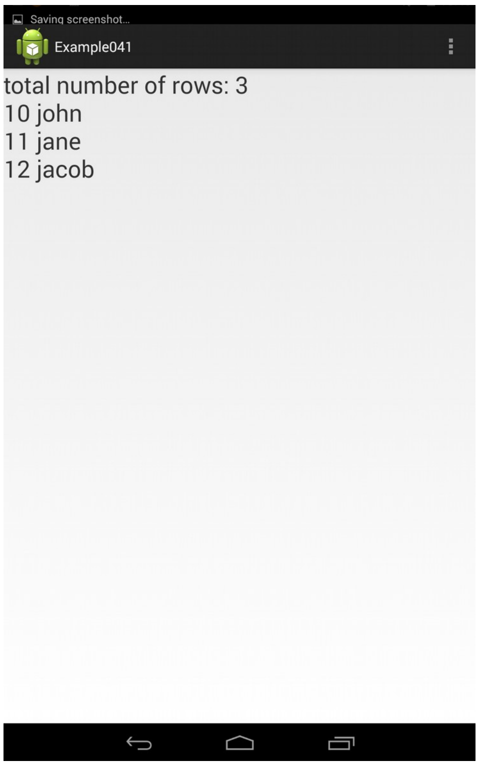
```
ContentValues u = new ContentValues();
u.put("FIRST_NAME", "jacob");
sdb.update("test", u, "FIRST_NAME = 'jim' ", null);
```

Explaination:

- Here is an example of updating a row in the database.
- When you are updating a row you need a ContentValues object that only contains data for the columns in the row that you wish to change. In this example we are only changing the first name of the person.
- Finally we call the update method on the database which requires four arguments
  - The first is the name of the table that you wish to update
  - The second is the content values that holds the updated data
  - The third is a where clause that will pick out the rows to be updated
  - The final argument is the where arguments that will fill the placeholders in the where clause.
- Running this code will get you the screenshots that you see below. Where the id field because it is set to auto increment will persist between runs of the application.



```
total number of rows: 3
7 john
8 jane
9 jacob
```



```
total number of rows: 3
10 john
11 jane
12 jacob
```

Example042: Creating/Writing/Reading/Deleting files

There will be times that the most appropriate way to store a set of data is in a file instead of the shared preferences or an SQLiteDatabase. You've already seen an example of this with the camera intent were we stored a file in external storage. Here we will create and write a file to internal private storage where it can only be accessed by the application itself and no other application can access it. With one small modification it is possible to write to the external storage which is shared and is accesible by all applications. Finally we will show how to read and delete a file from internal storage.

01) start with a new android project

02) remove all XML in activity_main.xml and replace with the following XML

```
<LinearLayout
      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/tv"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="30sp"
      android:text="@string/hello_world"
    />
</LinearLayout>
```

03) add the following code to the end of the onCreate() method in the MainActivity class

```
      // get access to the internal files directory and make a file there
      File f = new File(getFilesDir(), "test.txt");
      try {
            BufferedWriter bw = new BufferedWriter(new FileWriter(f));
            bw.write("hello world\nthis is an internal file");
            bw.close();
      } catch (IOException e) {
            e.printStackTrace();
      }
```

Explaination:

- Here is how you write a text file to internal storage. Note that the majority of code that you see here is standard Java code. The only major difference is in the declaration of the File object and how it is constructed.
- Note that the first argument of the constructor of the file is a call to the Activity's getFilesDir() method. This will return you the path of the internal directory that belongs to the application. If you change this to

getExternalFilesDir() you will get access to the external storage of the device.

04) add the following code to the end of the onCreate() method in the MainActivity class

```
// try to read the file back in using a buffered reader
String total = "";
try {
      BufferedReader br = new BufferedReader(new FileReader(f));
      String temp = br.readLine();
      while(temp != null) {
            total += temp + "\n";
            temp = br.readLine();
      }
      br.close();
} catch (IOException e) {
      e.printStackTrace();
}

// print the string on the text view
TextView tv = (TextView) findViewById(R.id.tv);
tv.setText(total);

// delete the file when we are finished with it
f.delete();
```

Explaination:

- reading the file using standard java code and setting the contents on a text view and finally deleting the file from internal storage. All of this is standard java code
- running this code will show the message “hello world, this is an internal file” when the application is started as that message is read from the constructed file.

Example043: Basic use of the notification manager

There will be times where your application is performing a task in the background and you may wish to notify your user that an action has completed or that a new message or error has occured. If you application is not visible then toast messages will not work. Thus the best way to get around this is use the notifications and the notification manager. In this example we will show how to create a notification and show it with a default sound and vibrate the device

01) start with a new android project

02) add in the following code to the end of the onCreate() method of MainActivity

```
// get access to the notification service
NotificationManager nm = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

// build a notification and show it on the device
int icon = R.drawable.ic_launcher;
Notification.Builder nb = new Notification.Builder(this);
nb.setContentTitle("Title of notification");
nb.setContentText("Description of notification");
nb.setSmallIcon(icon);
nb.setDefaults(Notification.DEFAULT_LIGHTS | Notification.DEFAULT_SOUND |
Notification.DEFAULT_VIBRATE);
Notification n = nb.build();
nm.notify(1, n);
```

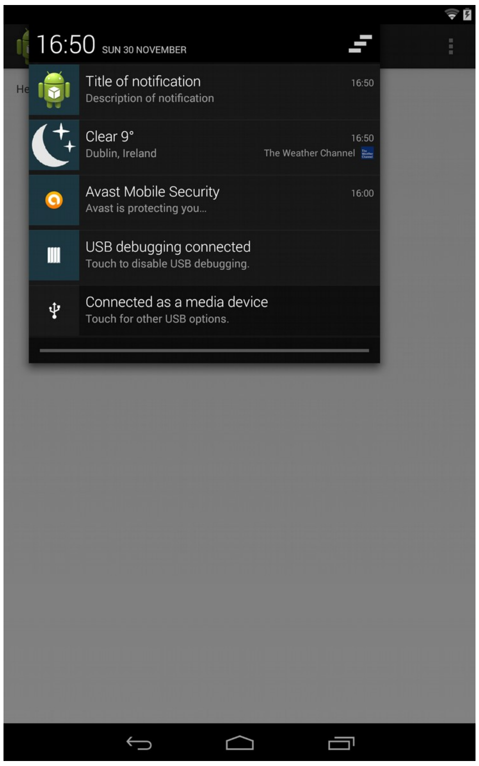
Explaination:

- The first line that is mentioned here is similar to the sensors and the GPS examples that you saw previously. In order to use notifications you need to get access to the notification service
- The next set of lines show you how to build a notification. Like Bitmaps that you saw with the camera it is easier to use the factory class (in this case called Builder) to generate the notification for you as doing so directly requires a lot of work.
  - The first line takes an id for a drawable that will be used for the icon of the notification to be displayed in the notification bar.
  - The second line constructs a notification builder. It requires a context to work so here we pass in a reference to our activity as the owning context.
  - The next four lines setup how the notification will look. The first line sets the title of the notification you should use this to give a short description of what the notification is. The second line sets the content of the notification essentially a longer description of why the notification is showing up in the notification area. The third line sets

the icon on the notification while th fourth line states that when the notification is shown it should show the default LED colour, sound, and vibration that is associated with a notification on the device.

- ◦ The last two lines build the notification object while the last line shows the notification in the notification area. The first argument is the id associated with the notification (in case you wish to react to it) while the second argument is the notification to show to the user.

- If you run the application you will see the screenshot below where the notification shows in the list of notifications.



16:50 SUN 30 NOVEMBER

Title of notification 16:50
Description of notification

Clear 9° 16:50
Dublin, Ireland The Weather Channel

Avast Mobile Security 16:00
Avast is protecting you…

USB debugging connected
Touch to disable USB debugging.

Connected as a media device
Touch for other USB options.

Example044: Creating simple dialogs

Sometimes you may wish inform users of warnings or errors in your application through the form of a dialog. Sometimes you may also want to give them a list of choices too. In this example we will show how to create three dialogs. One with a simple message and two buttons, A second that will give a user a list of choices and a final dialog that will take input from a user.

01) start with a new android project

02) replace all XML in activity_main.xml with the following XML

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
>
    <Button
      android:id="@+id/btn1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/btn1_text"
    />

    <Button
      android:id="@+id/btn2"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/btn2_text"
    />

    <Button
      android:id="@+id/btn3"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/btn3_text"
    />
</LinearLayout>
```

03) add the following empty methods into the MainActivity class

```
      // private method that will build a simple alert dialog
      private void simpleAlertDialog() {

      }

      // private method that will build a simple list dialog
      private void simpleListDialog() {

      }

      // private method that will build a simple input dialog
      private void simpleInputDialog() {
      }
```

Explaination:

- building dialogs takes a bit of work so it is better to separate that work into seperate methods instead of trying to crowd them into the listeners of the onCreate() method

04) add the following code to the end of the onCreate() method of the MainActivity class

```
// pull the buttons from the layout
btn1 = (Button) findViewById(R.id.btn1);
btn2 = (Button) findViewById(R.id.btn2);
btn3 = (Button) findViewById(R.id.btn3);

// add a listener to the first button to generate a simple dialog
btn1.setOnClickListener(new OnClickListener() {
      // overridden on click method
      public void onClick(View v) {
            simpleAlertDialog();
      }
});

// add a listener to the second button to generate a simple list dialog
btn2.setOnClickListener(new OnClickListener() {
      // overridden on click method
      public void onClick(View v) {
            simpleListDialog();
      }
});

// add a listener to the third button to generate a simple input dialog
btn3.setOnClickListener(new OnClickListener() {
      // overridden on click method
      public void onClick(View v) {
            simpleInputDialog();
      }
});
```

Explaination:

- as before putting the dialog code into seperate methods keeps the onCreate() method small and clean.

05) add the following code to the simpleAlertDialog() method of the MainActivity class

```
// we need a builder to create the dialog for us
AlertDialog.Builder builder = new AlertDialog.Builder(this);

// set the title and the message to be displayed on the dialog
builder.setTitle("Simple dialog");
builder.setMessage("Is this a simple dialog");

// add in a positive button here
```

```
builder.setPositiveButton("yes", new DialogInterface.OnClickListener() {

      @Override
      public void onClick(DialogInterface dialog, int which) {
            // we know for definite that the user has clicked the yes button
            Toast.makeText(MainActivity.this, "simple dialog yes", 
Toast.LENGTH_SHORT).show();
      }
});

// add in a negative button here
builder.setNegativeButton("no", new DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialog, int which) {
                  // we know for definite that the user has clicked the no 
button
                  Toast.makeText(MainActivity.this, "simple dialog no", 
Toast.LENGTH_SHORT).show();
      }
});

// create the dialog and display it
AlertDialog dialog = builder.create();
dialog.show();
```

Explaination:

- Like the notifications you saw in the previous example it is much easier to use a factory class to build your dialogs than to try and build on directly. For all of these dialogs we will use the AlertDialog.Builder factory to construct all of our dialogs. Like the notification build you saw in the previous example you need to provide an owning context for the dialog that you are about to builder
- The next two lines then set a title on the dialog and a message to be displayed to the user.
- Next we add a positive button and a listener to it for when the user accepts what the message is stating to them. The positive button should be used to perform an action that the user has selected. The negative button afterwards should not perform any action besides canceling the operation that the user was about to perform. It should make no change to the application.
- The final two lines then create the dialog and show it to the user.

06) add the following code to the simpleListDialog() method of the MainActivity class

```
// we need a builder to create the dialog for us
AlertDialog.Builder builder = new AlertDialog.Builder(this);

// set the title and the list of choices
builder.setTitle("Pick a colour");
builder.setItems(R.array.choices, new DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialog, int which) {
            // do a different thing depending on the choice
            if(which == 0)
                  Toast.makeText(MainActivity.this, "selected red",
Toast.LENGTH_SHORT).show();
            else if(which == 1)
                  Toast.makeText(MainActivity.this, "selected green",
Toast.LENGTH_SHORT).show();
            else if(which == 2)
                  Toast.makeText(MainActivity.this, "selected blue",
Toast.LENGTH_SHORT).show();
      }
});

// create the dialog and display it
AlertDialog dialog = builder.create();
dialog.show();
```

Explaination:

- Here we are building a list dialog that will give the user a choice between a few items. Note that we use the alert dialog builder again and we set a title on the dialog.
- The line after setting the title is responsible for getting a list of strings (in this case from an XML string-array) and setting each string as an individual choice in the list, the second argument here is a DialogInterface.OnClickListener() that will catch an event from the Dialog as soon as the user chooses an option. Be careful here do not use the standard OnClickListener we have used to this point. If you have an error with your listener and you don't know why this is a common cause of it.
    - The onClick() method of the listener accepts two arguments the first is the dialog that generated the event. While the second is the option that the user has chosen. Options are numbered from 0 at the top to N-1 at the bottom assuming your list has N items.
- As before the final lines create and show the dialog to the user.

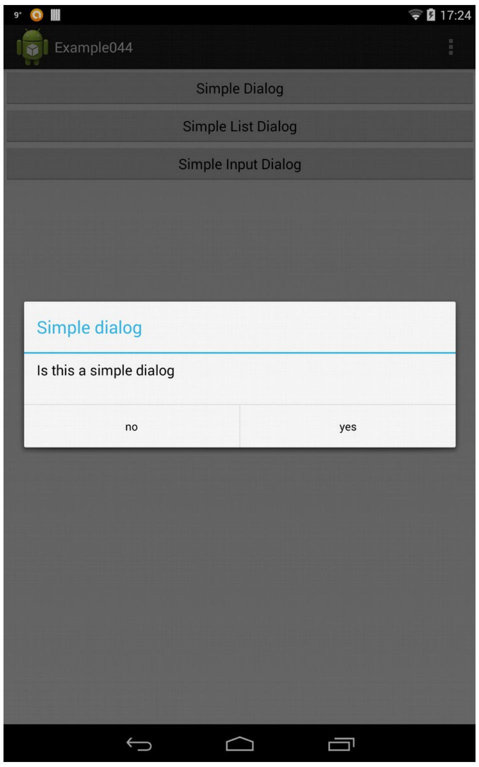
07) add the following code to the simpleInputDialog() method of the MainActivity class

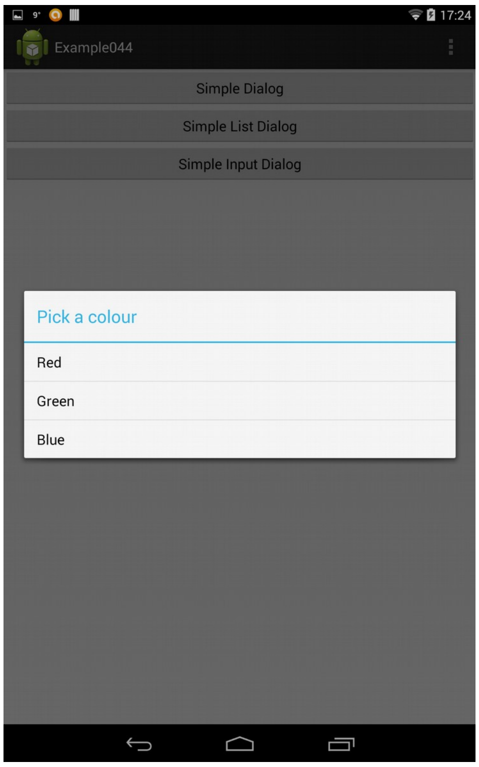
```
// we need a builder to create the dialog for us
AlertDialog.Builder builder = new AlertDialog.Builder(this);

// set the title on this dialog
builder.setTitle("Set a new name");

// it is possible to define your own layouts on a dialog but because we only
need
// a single edit text we
// will create it and add it here
final EditText et = new EditText(this);
et.setInputType(InputType.TYPE_CLASS_TEXT);
et.setHint("New name");
builder.setView(et);
```

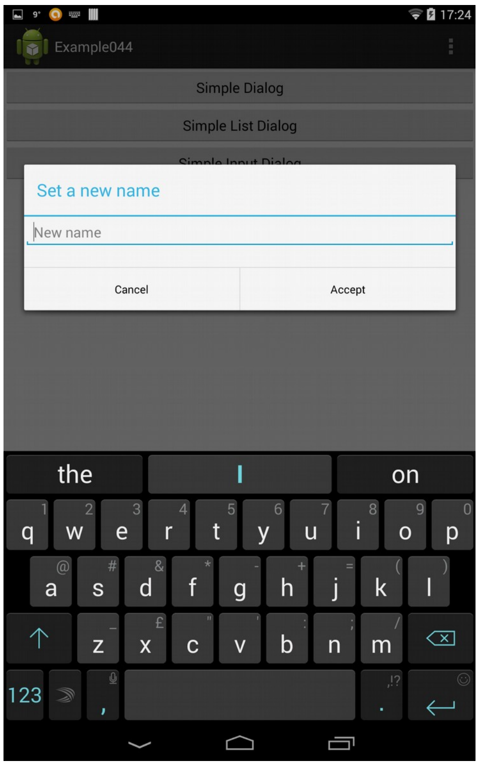
```
// add in the positive button
builder.setPositiveButton("Accept", new DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialog, int which) {
            // we know for definite that the user has clicked the yes button
            Toast.makeText(MainActivity.this, "user input is: " +
et.getText().toString(),
                        Toast.LENGTH_SHORT).show();
      }
});

// add in the negative button
builder.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialog, int which) {
            // we know for definite that the user has clicked the yes button
            Toast.makeText(MainActivity.this, "user cancelled input",
Toast.LENGTH_SHORT).show();
      }
});

// create the dialog and display it
AlertDialog dialog = builder.create();
dialog.show();
```

Explaination:

- Constructing an input dialog is similar to constructing a standard dialog. The only difference is that instead of using a standard layout we construct our own after the title is set.
- Here we just create a single edit text but it is just as easy to use a seperate layout XML file here and attach it. If you ever need to build a general dialog or something completely custom then this is the approach to use.
- Running this code will net you the screenshots below


Example044
Simple Dialog
Simple List Dialog
Simple Input Dialog
Simple dialog
Is this a simple dialog
no
yes


Example044
Simple Dialog
Simple List Dialog
Simple Input Dialog
Pick a colour
Red
Green
Blue


Example044
Simple Dialog
Simple List Dialog
Set a new name
New name
Cancel
Accept

Example045: Using a thread to perform actions in the background

While it may be tempting to perform all of your work on the main thread to simplify your application design but you have to be extremely careful here as you may find that if you do too much work on the main thread that your application will crash even though you are technically doing everything correctly. This is because android has a built in timer that measures the time between an event being delivered to an application and the time it recieves a response stating if the event has been handled.

If this time exceeds a predetermined limit (5 seconds) android will consider the application as unresponsive and will automatically terminate it as the user may be trying to perform other actions. The way around this is to introduce a thread that will do the work in the background for you and notify the UI whenever an update has been made. In this example we will create a simple timer that will count from 1 to 20 and will stop once the application reaches 20.

01) start with a new android project

02) replace all teh XML in activity_main.xml with the following XML

```
<LinearLayout
      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/tv"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="30sp"
      android:text="@string/hello_world"
    />

</LinearLayout>
```

03) create a new java file called ThreadHandler.java and give it the following code

```
package com.example.example045;

// imports
import android.os.Handler;
import android.os.Message;

// definition of the thread handler class
public class ThreadHandler extends Handler {
      // constructor for the class that takes in a reference to a main activity
      public ThreadHandler(MainActivity ma) {
            this.ma = ma;
      }

      // override this method so we can handle our own events
      public void handleMessage(Message msg) {
            if(msg.what == 0xDEADBEEF) {
                  ma.updateUI(msg.arg1);
            }
      }

      // a referenence to the main activity class so we can update the UI
      private MainActivity ma;
}
```

Explaination:

- Here we are creating a handler class that will pass messages from our background thread to the UI thread. The reason this needs to be done is that the UI thread has full control of the graphics context to render the UI. If any other thread tries to render than the UI thread it will cause an exception and the application will crash.
- All the handler does is listen for messages and react to them if the message is one that this application is interested in.
- To create a thread handler you are required to extend the standard Handler class as it contains a number of default message that are required for interaction with the android OS. All you will do here is add in your messages that you wish to handle. Here we state that if the what field of the message that we recieve is the hexadecimal integer 0xDEADBEEF then we will call the update UI method with the first integer argument of the message.
    - If at all possible if you only have a one or two arguements to send in a message try and make them integers and use arg1 and arg2 of the message. The reason for this is that the message handler pulls messages from a pool that is already allocated and it is much quicker to set a couple of values than use the key value pair system that takes a bit longer to setup and work. Arg1 in this case will be the time to be shown on the text view.
- Note that in the constructor we take a reference to our MainActivity so we can call the updateUI method with the updated time.

04) create a new java file called UpdateThread.java and give it the following code

```
package com.example.example045;

// imports
import android.os.Handler;
import android.os.Message;

// definition of the class
public class UpdateThread extends Thread {
	// constructor for the class that takes a reference to a handler
	public UpdateThread(ThreadHandler h) {
		this.h = h;
	}

	// overridden run method that will send an update message every second
	public void run() {
		// set the cycles to 0
		cycles = 0;

		// sleep for a full second and when we wake up update the cycles 
send
		// a message through the handler and sleep again
		while(cycles < 20) {
			cycles++;
			Message m = h.obtainMessage(0xDEADBEEF, cycles, 0);
			h.sendMessage(m);
			try {
				sleep(1000);
			} catch (InterruptedException ie) {

			}
		}
	}

	// private fields of the class
	private int cycles;
	private Handler h;
}
```

Explaination:

- Here we are creating our thread class by extending the standard Java Thread class.
- Note that the constructor takes a reference to a ThreadHandler so on each cycle it can get a message from the message pool and send it to the UI thread.
- The run method of the thread simply counts 20 seconds and sends a message every second to the UI thread. Note that we use the handler to obtain a message from the pool and set a message identifier and the two integer arguments immeidately. Thus upon obtaining the message it is ready for sending immediately.

05) add the following fields to the MainActivity class

```
// private fields of the class
private TextView tv;
private ThreadHandler th;
private UpdateThread ut;
```

Explaination:

- to setup the background thread fully we need to have access to a handler and the thread.

06) add the following methods to the MainActivity class

```
// method that will be called by the thread handler to update the UI
public void updateUI(int num) {
      // update the value that appears on the text view
      tv.setText("" + num);
}

// overridden resume method that will start the thread going
protected void onResume() {
      super.onResume();

      // start the thread
      ut.start();
}
```

Explaination:

- the updateUI method that will be called from the handler will simply update the textview with the new value that was found in the messages
- the onResume() method will start the thread working as soon as the activity is brought to the foreground. Ideally this thread would be paused as soon as the activity is moved to the paused state.

07) add the following code to the end of the onCreate() method of the MainActivity class

```
// pull the textview from the layout and set it to zero
tv = (TextView) findViewById(R.id.tv);
tv.setText("0");

// setup the update thread and the thread handler
th = new ThreadHandler(this);
ut = new UpdateThread(th);
```

Explaination:

- Here we get access to our textview and make the connection between the thread handler and MainActivity. Also the connection between the UpdateThread and the thread handler is made here as well. So when the application starts each message the thread generates goes through the thread handler and then to MainActivity.
- As soon as you create the handler it will be attached to the UI thread. The UI thread will always check for handler messages and as soon as one is recieved it will process it immediately.