Mobile Application Development - Part 2

Adding Interactivity, Layouts

In this section we will:

- Extend our application by actually making it do something!
- Introduce layouts

Making our app do something!

OK our app says 'Hello World'. And we've translated it into another language. But it doesn't yet actually ***do*** anything. In this exercise you will add some ***interactivity*** by developing a simple application to convert feet to metres. **Create a new project** and call the main activity **FeetToMetresActivity**. Replace the auto-generated code with this code (obviously change the package name to that appropriate in your case):

```
package whatever.your.package.is;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.TextView;
import android.widget.EditText;


public class FeetToMetresActivity extends AppCompatActivity implements OnClickListe
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b = (Button)findViewById(R.id.btn1);
        b.setOnClickListener(this);
    }

    public void onClick(View view)
    {
        TextView tv = (TextView)findViewById(R.id.tv1);
        EditText et = (EditText)findViewById(R.id.et1);
        double feet = Double.parseDouble(et.getText().toString());
        double metres = feet*0.305;
        tv.setText("In metres that is: " + metres);
    }
```

```
}
```

and also change your ***activity_main.xml*** layout file to look like this:

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

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="numberDecimal"
    android:id="@+id/et1"
    />

 <Button android:layout_width="wrap_content" android:layout_height="wrap_content"
 android:id="@+id/btn1" android:text="@string/convertBtn" />

 <TextView android:layout_width="match_parent" android:layout_height="wrap_content"
 android:id="@+id/tv1"/>

</LinearLayout>
```

and finally add the following line to your strings file:

```
<string name="convertBtn">Convert!</string>
```

So how is this working? Let's start with the layout file as that's probably the simplest. Note how we now have three components: an ***EditText*** (editable text field), a ***Button*** and a ***TextView***. Also note that each has been given an ID: ***et1***, ***btn1*** and ***tv1*** respectively. We will use these IDs to access each component from our Java code.

Also note how we have specified the layout for each component. In particular, note the different layout specifiers ***match_parent*** and ***wrap_content***. As seen above, the former specifies that the componen will fill the entire space of its parent component. Note how for the edit text and the text view, we set th width to ***match_parent*** so that the component fills the entire width of the parent (the screen in this case); however, note the height is set to ***wrap_content***. This means that the component will use up a much space is required, and if its content exceeds the space available to it, it will be wrapped to the next line.

Finally notice that the ***EditText*** has an ***inputType*** attribute. This specifies what type of data will be entered in the EditText. Here we have set it to ***numberDecimal*** to indicate that it can only be used for entering decimal numbers. Some other types:

- ***text*** - the default. Allows text and numeric entry.
- ***number*** - positive whole numbers.
- ***numberSigned*** - positive and negative whole numbers.

You can combine types with the "or" symbol |, e.g. ***numberDecimal|numberSigned*** would allow entry of positive and negative decimal numbers.

On to the Java code

OK, how about the actual Java code? First of all note how we access the user interface components from Java. We use code such as:

```
Button button = (Button)findViewById(R.id.btn1);
```

Note how the ***findViewById()*** method takes in an ID and returns the corresponding component. The ID matches the IDs that we specified in the XML file; for instance, our button was given an ID of ***btn1*** so we reference it in code with ***R.id.btn1***. If you look at the ***R.java*** file now, you'll notice that static attribues for each of our IDs have been generated, containing hex values.

Also note that ***findViewById()*** returns an object of type ***View*** (Buttons and other user interface elements are subtypes of ***View***) so we have to ***cast*** (convert) it to the appropriate type using code such as:

```
Button b = (Button)findViewById(R.id.btn1);
```

The ***(Button)*** is the cast (conversion) from View to Button. Later in the unit you will understand in more detail what is happening here.

Event handling

Obviously we need to make something happen when the user clicks the button! If you have worked with Java Swing or AWT, the approach taken by Android should be quite familiar. We attach a ***listener object*** to the button and then define an ***event handler*** method within the listener object to react to the event. The listener object is attached with this code:

```
b.setOnClickListener(this);
```

This indicates that the current object (***this***) will act as the listener object for the button. The listener object must implement the ***OnClickListener*** interface (hence the addition of ***implements OnClickListener*** to the class declaration for our main Activity) and provide a method called ***onClick()*** which takes one parameter: the View object which generated the event. The ***onClick()*** method runs when we click on the button. To reproduce the code:

```
public void onClick(View view)
{
   TextView tv = (TextView)findViewById(R.id.tv1);
   EditText et = (EditText)findViewById(R.id.et1);
   double feet = Double.parseDouble(et.getText().toString());
   double metres = feet*0.305;
   tv.setText("In metres that is: " + metres);
}
```

Hopefully this should be fairly straightforward. We obtain the TextView and EditText components using their ID, just like we did with the Button. We then convert the contents of the EditText to a ***double*** (high precision floating point number) so that we can perform the calculation on it:

```
double feet=Double.parseDouble(et.getText().toString());
```

Note that unlike the equivalent in Swing, the ***getText()*** method of EditText does not directly return a String, but returns an object of type ***Editable***. We then need to call ***toString()*** on the Editable to get

the actual text out of the text field, and finally convert it to a ***double*** using ***Double.parseDouble()***.

Then we convert the feet to metres and stick the result in the TextView.

Changing the layout

We're now going to experiment with some slightly more complex layouts. Firstly we're going to make the button and the TextView appear on the same line, as follows:

To do this we create a ***layout within a layout***, as follows:

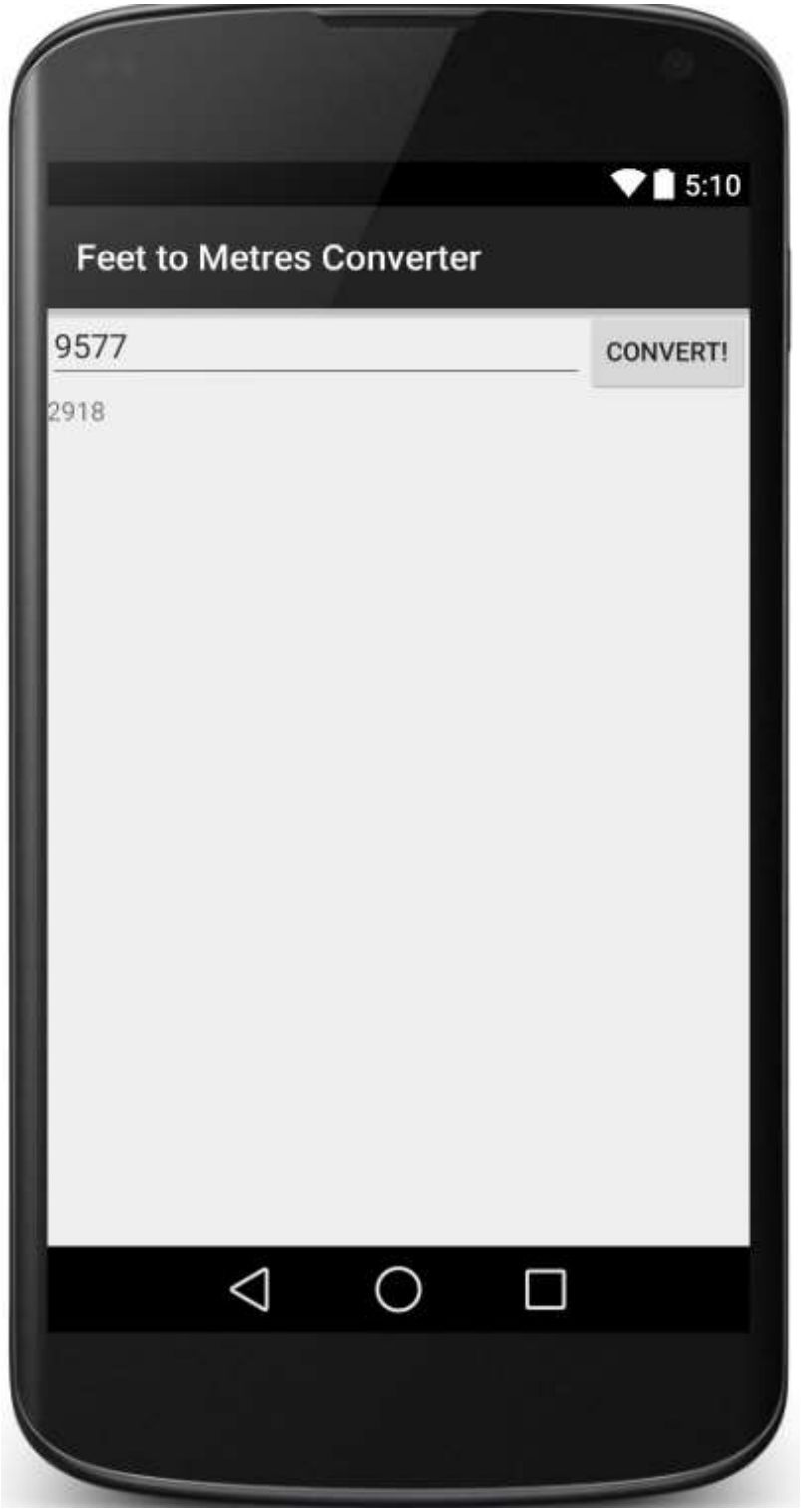
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
```

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
        android:inputType="numberDecimal"
    android:id="@+id/et1"
    />

 <LinearLayout android:orientation="horizontal"
  android:layout_width="match_parent" android:layout_height="wrap_content">

 <Button android:layout_width="wrap_content" android:layout_height="wrap_content"
 android:id="@+id/btn1" android:text="@string/convertBtn" />

 <TextView android:layout_width="match_parent" android:layout_height="wrap_content"
 android:id="@+id/tv1"/>

 </LinearLayout>
</LinearLayout>
```

Note how we have a second LinearLayout ***within*** the first LinearLayout, and the Button and TextView are within the second LinearLayout. Also note how the second LinearLayout has a ***horizontal*** orientation, so that components within it will appear next to each other. The net result will be that the button and TextView appear next to each other.

Relative Layouts

However, this probably isn't the nicest layout for this particular application. What would probably look best is to have the EditText and button next to each other, with the EditText occupying the majority of the row and the button the remainder. The TextView with the result would then appear on the next row In other words, we might want a layout such as:



So, you might be thinking, how about just creating another LinearLayout within a layout, as before, with a horizontal orientation? The problem, though, is how you get the EditText to occupy the majority of the row. If you set both to have an android:layout_width of ***wrap_content***, both EditText and Button will only occupy minimal space. If, on the other hand, you set the EditText to have an android:layout_width of ***match_parent***, it will occupy the ***whole*** of the row and overwrite the button!

A ***relative layout*** is the solution to this problem. With a relative layout, we can specify the position of components relative to each other and also specify whether components are aligned to the left, right, top or bottom of their parent component. What we have to do, as shown in the XML below, is to (the order doesn't matter, but doing it this way helps understand the concept better):

- Add the button ***first*** - even though it's on the right hand side of the row - and specify that it's aligned to the right of its parent, i.e. the RelativeLayout that it is within (with ***android:layout_alignParentRight="true"***). This will force the button to appear as far right as possible

- Add the EditText ***second***, specify that it's to appear to the left of the button (with ***android:layout_toLeftOf="@id/btn1"***), and set its android:layout_width to ***match_parent***. Because it's been forced to appear to the left of the button, it will only occupy the space availabl to it, in other words the space between the left hand side of the screen and the button.

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

 <RelativeLayout android:orientation="horizontal" android:layout_width="match_paren
 android:layout_height="wrap_content">

<Button android:layout_width="wrap_content" android:layout_height="wrap_content"
 android:id="@+id/btn1" android:text="@string/convertBtn"
 android:layout_alignParentRight="true"/>

 <EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/et1"
    android:layout_toLeftOf="@id/btn1"
        android:inputType="numberDecimal"
    />

 </RelativeLayout>

 <TextView android:layout_width="match_parent" android:layout_height="wrap_content"
 android:id="@+id/tv1" android:text="@string/hello"/>

</LinearLayout>
```

Advanced exercise

Try using a relative layout to create this layout:

Note that you can use the ***android:textSize*** attribute to set the text size, e.g give it a value of "12pt" for 12-point text.

Further reading

Layout Tutorials on Android site

Advanced Optional Topic: Layout weight

Imagine we wanted to write an app to report problems in the street (e.g broken lights, potholes etc). We might have a layout like this, where the user enters the problem in an EditText and can click either

"OK" to report it or "Clear" to clear the EditText:

Create a new project and try setting its layout to the layout above (hint: use a RelativeLayout for the top level layout). The problem is: how do you get the buttons to occupy the whole of their row, and occupy the same amount of space? By default you will get this as the buttons will only use the space

needed for them:

An easy way to do this is to give each button (this technique is not restricted to butttons, it can be use for other UI elements) a ***layout_weight***. The ***layout_weight*** indicates the relative proportion of each component to the others, so if each button has a ***layout_weight*** of 1, each button occupies the same amount of space. However, because layout_width is a compulsory attribute, we have to give it a value To avoid unnecessary computations which are then overridden by ***layout_weight***, we simply set it to 0 pixels rather than one of the usual values such as ***match_parent*** or ***wrap_content***.