**Lab 2: GUI, Layout & Simple interaction**

**Creating User Interfaces in Android**

In this practical session, you will create an Android app with a graphical user-interface and learn about implementing layout in Android development. You will probably wish to use some graphics files – some icons are on Moodle but please feel free to use your own.

Android Studio incorporates a user interface (UI) designer. Create a new android project as last week. When you start a new project the layout file is usually open and in Text mode, see the tab at the bottom of the code window. The layout file is an XML file that defines the layout (RelativeLayout by default) and any other components. In the XML you should see there is a TextView which is included as a default. Each component has attributes – for the default TextView they are the text, width and height. Identify these attributes in the code window.

To add another component you can add to the XML. For example to add a button you can add the XML below.

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="New Button"
  android:id="@+id/button"/>
```

The only problem with this is that it might appear on top of the TextView – we have not included position attributes. In the relativeLayout it is usually better to create a UI using drag & drop from a palette of components and widgets that is provided in the on-screen editor.

In the design view select the Hello World! and delete it. Do the same for the button. Check the xml that both have been removed. You will usually need to swap between the two windows when creating an interface.

Click on the Design Tab. Select the button from the Widgets part of the palette. Drag it over onto the middle of the phone screen. Skip back to the Text window to see that some positioning attributes have been added within the XML defining the button's relative position. You will see that the positioning is done with dp. You can edit these to move the button or drag the button around. Moving or adding components in the Design window generates XML.

The aim now is to get the button to activate a small pop up message. Double click the button and in the window that appears give the text for the button any name you want and give it an id *toastButton* – you will see why later. (Instead of double clicking the button you can use the properties window to the right of the IDE).

Now you need to edit the java file MainActivity to get the action to work. You can cut and paste the lines of code below but you must work out where they are to go.

1. Declare the button:
   ```
   Button myButton;
   ```

2. Connect the java button to the xml button
   ```
   myButton = (Button)findViewById(R.id.toastButton);
   ```

3. Make the activity implement View.OnClickListener
   ```
   implements View.OnClickListener
   ```

4. Attach the listener to the button
   ```
   myButton.setOnClickListener(this);
   ```

5. Provide an onClick method to deal with the click and pop up a message
   ```
   public void onClick(View view) {
           Toast.makeText(this, "Button has been clicked.", Toast.LENGTH_LONG).show();
   }
   ```

You will need to deal with some imports for toast, view and button. Scroll over the red places and Alt-enter should deal with this.

If your code is error free test your app on the AVD. If there are any problems check your code for errors – see any red boxes. If you are unable to see any problems ask for help.

Once you have this working, and you should not move on until you have, you have a way of testing whether your UI component is working.

The app uses Toasts to provide simple feedback – you do not need to know more now but you can look at the following link for further information
http://developer.android.com/guide/topics/ui/notifiers/toasts.html.

**Some other UI widgets**
The Design view gives access to a palette of many Widgets and other components. Let's look at a few now.

If you drag the widget onto the phone screen a short XML extract that defines the widget will appear in the layout file. You will then need to edit the java code to make the component work. Suitable java is provided below. Cut and paste these code extracts into your projects in the appropriate places.

You might like to start a new project for each component or you can add all the components to one project – it might be more confusing this way but you will learn more. The procedure for getting the component to work is essentially the same in each case.

**Checkbox**
In the design view drag a CheckBox onto the stage below the Hello World! textView – note that its default id is *checkBox*. You can see this by either double clicking on the component or scrolling down in the properties window to the right of the design window.

The procedure to make the check box active is similar to the button above:

```
CheckBox cb;                                    // Declaration of the checkbox
cb = (CheckBox)findViewById(R.id.checkBox);     // Connect the java checkbox to the xml checkbox

implements CompoundButton.OnCheckedChangeListener
cb.setOnCheckedChangeListener(this);            // Attach the listener to the checkbox
```

Provide an onCheckedChanged method to deal with the click and pop up a message:

```
public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
                cb.setText ("This checkbox is: checked");
        }
        else {
                cb.setText ("This checkbox is: unchecked");
        }
}
```

Notice how within this method the text of the checkbox is changed. You will need to add some import statements using Alt-enter as you go through this process.

**Radio Button/RadioGroup**

A single radio button can be added in a similar way. Try dragging one from the palette onto the device. A container is an object that you can put components into. The RadioGroup is an example of a container.

Usually we are interested in groups of radio buttons. On the palette scroll down to Containers and drag a RadioGroup onto the device. Now drag a number of radio buttons onto this. If you examine the xml you should see a number of Radio buttons (with separate ids) nested within the RadioGroup. If you run this on device you will see the usual radio button behaviour where only one button can be selected at any one time.

**Spinner Widget**

The spinner is a very useful component but slightly harder to set up. Drag one onto the device. The xml is set up but there is more configuration required in the java file. The TextView is there for output. You must make sure your textfield and spinner id's match the names in the layout xml file.

```
implements AdapterView.OnItemSelectedListener       // for class header
private TextView selection;                         // declaration of class variable
Spinner spin;
private static final String[] items = {"Nibali", "Froome", "Wiggins", "Evans", "Schleck*", "Contador",
"Sastre", "Pereiro"};

// code for onCreate() method
selection = (TextView)findViewById(R.id.textselection);
spin = (Spinner)findViewById(R.id.spinner);
spin.setOnItemSelectedListener(this);
ArrayAdapter<String> aa = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item,
items);
aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spin.setAdapter(aa);
```

```
// methods for class
public void onItemSelected(AdapterView<?> parent, View v, int position, long id) {
        selection.setText(items[position]);
}
public void onNothingSelected(AdapterView<?> parent) {
        selection.setText("");
}
```

**ImageView**
An ImageView is a simple way to include an image in an app. Find a small image or icon (preferably png format). Copy the file, select the *drawable* folder in the project and paste it in.

Whilst looking at a layout in Design view simply drag an imageView from the widgets palette onto the device screen. Double click the object and, to set your image as src, navigate to the images – you will need to scroll down to the drawable folder to select your image. This will be shown in the render.

**UI and Layout**
General link to Layout http://developer.android.com/guide/topics/ui/declaring-layout.html

Unless you have been very careful the screen is likely to look a mess now as you have added interface components in a fairly random way. The overall default layout is RelativeLayout. If you scan through the xml layout file you will see that the positioning of each component is relative to the other components that are already there. So moving one component can have an effect on other components. How the final interface looks can also depend very much on the dimensions of the device. So, relative layout is generally considered bad.

You can read more about RelativeLayout at
http://developer.android.com/guide/topics/ui/layout/relative.html.

A common layout is the LinearLayout. You can convert the RelativeLayout to linear by replacing RelativeLayout by LinearLayout at the beginning and end of the xml file (that is, just overwriting the text in the layout xml file). Do this to your current layout.

The result might not be quite what you expect. This is because the default LinearLayout is horizontal. This can be fixed by applying the following attribute:
```
android:orientation="vertical"
```

LinearLayout is a container into which you can put components and configure how they look. Android describes the linear layout at
http://developer.android.com/guide/topics/ui/layout/linear.html

**Building a simple interface**

Start a new project and cut and paste the example below, which contains 3 text boxes and a button, into the layout file for your project.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp"
   android:orientation="vertical" >
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="to" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="subject" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:gravity="top"
      android:hint="message" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:layout_gravity="right"
      android:text="send" />
</LinearLayout>
```

A point to note in the description at this link is the weight attribute that determines how much space an item gets. The Component Tree window is a useful way of keeping track of and manipulating layouts and UI components. The linear layout can be converted to horizontal (rather than vertical) in the properties. It ruins it!

Layouts can be dragged from the palette into the component tree. Drag a horizontal linear layout onto the current linear layout of the component tree. Drag a button from the palette into the new layout. Now drag the send Button onto this new layout. All layouts can be nested within each other – this might be a useful way of ensuring your layout looks just as you want it. The Outline window in the IDE is a great way of rearranging layouts and objects, where you can drag and drop items and layouts.

The android layout width and height properties can be set up in a number of ways. From the XML layout file you have seen this can be *match_parent* (take up as much room as the container it is in), *wrap_content* (give it all the space it needs and go into the next line if necessary) or, for example, a fixed size (such as 100dp).

Another useful layout is the grid layout. Drag a GridLayout into one of the linear layouts. In the properties window fill in 2 for the columnCount and the rowCount. You can now add widgets to the grid.

Drag an ImageView onto the gridlayout. You will need to select an image in the src property. You should add some images (preferably png) to the drawable folder. You can resize the images (but you might consider whether it is the original images you should be resizing).
Try adding an image button and other form widgets or text fields. There are a number of input controls or widgets http://developer.android.com/guide/topics/ui/controls.html

You can nest layers to many levels – but be warned that nesting layers of layouts can have an impact on render time.


Exercises:

1. Produce an interface for a weight watch app that allows users to enter weight and height and which calculates BMI when a button is pressed.

2. Extend the above app to allow entry of age, gender and level of activity that calculates the users recommended daily calorie intake.
Include a safety warning that the product is not medically approved!

**Appendix A Icons and Images**
Some icons etc. you can download
http://developer.android.com/design/downloads/index.html

**Appendix B: Switch**
The steps to implement a switch are the same as the checkbox. Notice that it needs to implement the same interface as the check box. This means that the onCheckedChanged() function needs to deal with the switch and the checkbox.

Be careful to distinguish between the component switch and the control statement switch.
What is the id of the switch component when you drag it onto the phone?

```
implements CompoundButton.OnCheckedChangeListener
Switch sw;

sw = (Switch) findViewById(R.id.switch1);
sw.setOnCheckedChangeListener(this);

public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
	if (isChecked) {
		sw.setText("This switch is: on");
	}
	else {
		sw.setText("This switch is: off");
	}
}
```

Note that the switch uses the same interface as the checkbox and if you have both on the same screen you need to have just one onCheckChanged method to deal with both "buttons". To make sure that only the compound button that is clicked responds you need to determine which button was clicked. You will need code like the following:

```
public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
	switch (buttonView.getId()) {
		case R.id.checkBox:
			if (isChecked) {
				cb.setText("This checkbox is: checked");
			}
			else {
				cb.setText("This checkbox is: unchecked");
			}
		break;
		case R.id.switch1:
			if (isChecked) {
				sw.setText("This switch is: on");
			}
			else {
				sw.setText("This switch is: off");
			}
		break;
	}
}
```