University of Stirling
Computing Science
Mobile App Development

Android Practical 1

Hello World

Warning!

As the industrial standard for Android development is very much in the "Android Studio camp", we thought it was important that this course reflected this reality. It does provide a useful user interface design capabilities and it is good at supporting a single development for a wide range of device types and screen sizes.

However Android studio is very much designed to work on a dedicated computer, and so when we use it the lab in a secure and flexible networked environment we may find it slow at times - **in particular for starting the emulator**. This is always slow, but can at times be particularly slow in the lab. So please **do have some patience**. You will often need to wait. It is best to think of starting the emulator as powering up your phone – it takes time so do it as little as possible. Start it at the beginning, and leave it running.

Android Studio in the lab has been configured to use your H: drive for the source files to provide flexible working in the lab. Each installation will also use the E: drive for working files, which links to the USB stick.

If you are working at home on your own computer you should find things rather faster!

Background

If you are familiar with the Eclipse or IntelliJ development environments you should find the user interface of Android Studio familiar. The course book *Hello, Android* is useful background reading on using Android Studio but in addition useful web references on Android include:

- https://www.javaworld.com/article/3340234/tutorial-series-android-studio-for-beginners.html which is a useful tutorial on getting started with Android studio.
- developer.android.com is a good Android development reference.

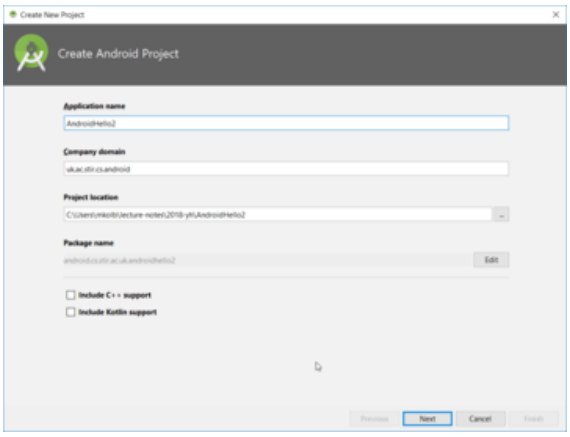
A screenshot at the start of each practical will show what your application should look like. Some of the practicals require you to create lengthy code. Rather than typing this from scratch, you can copy-and-paste it from the online PDF of the practical.

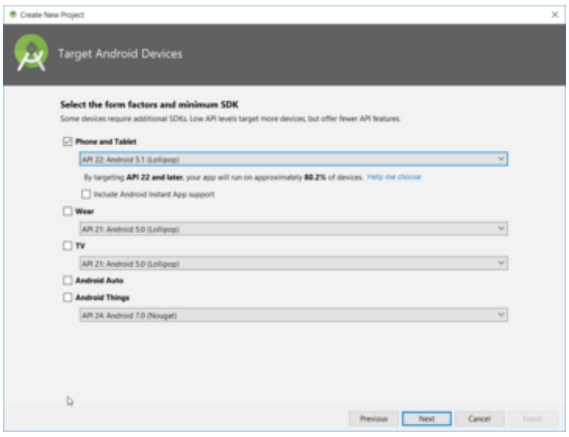
In this practical you will create a simple 'hello world' Android application.

Screen shots have been added as indicative guides but may differ slightly to what you see.

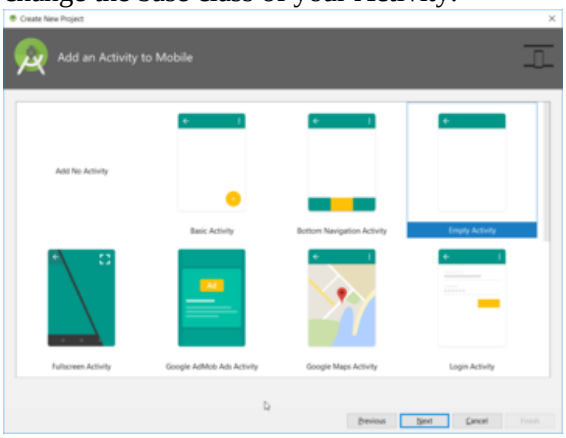
Create a New Android Project

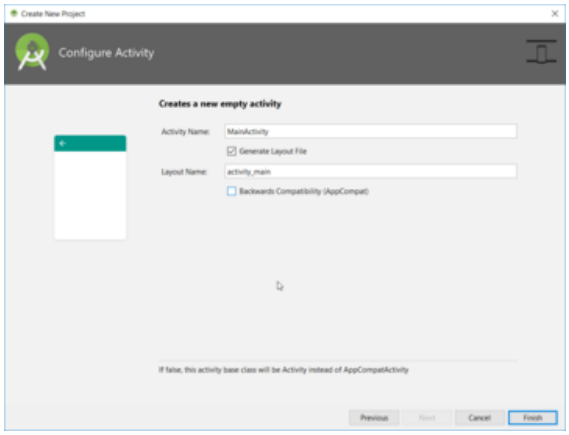
When you start up Android select a new project and then choose an application name, and then API 22:



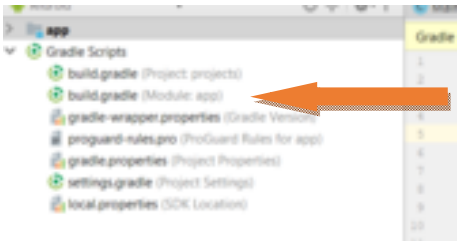


The screenshots are only indicative here – you may well want to use your own names. Now choose an empty activity and give it a meaningful name. Do not select Backward Compatibility as that will change the base class of your Activity.

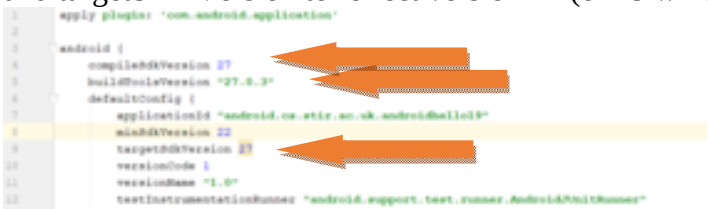




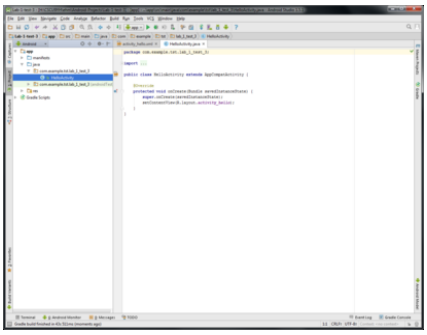
Important: Should you get an error message at this point saying that the required SDK and/or BuildTools version is not installed, DO NOT attempt to install the missing versions! It is unnecessary and will take a very long time! Instead open up the build.gradle file in the app module:



Then edit the lines: compileSDKVersion, buildToolsVersion (you may need to add this line!), and targetSDKVersion to reflect version 27 (or 29 which is also installed). See screenshot:



Now navigate your way to your activity class (you may have called it something else to this screenshot and indeed have a different package name!) but you will have something along the lines of the code given below:



```
package uk.ac.stir.cs.android;

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

public class HelloActivity extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);  }
}
```

Note that the class is based on the *Activity* class that performs actions. An application may have many separate activities, but the user interacts with one at a time. The *onCreate* method is called by the Android system when your activity starts. This is where you should perform all initialisation and user interface setup.

Construct the User Interface

Modify your code by making the changes shown below in colour (again you will need to adapt this to suit the names you have used):

```
package uk.ac.stir.cs.android;

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

public class HelloActivity extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // setContentView(R.layout.main);
    TextView textView = new TextView(this);
    textView.setText("Hello Android");
    setContentView(textView);
  }
}
```

An easy way to import missing packages into your project is to use Alt+Enter. This identifies missing imports based on your code and adds them for you.

An Android user interface uses a hierarchy of graphical objects called *Views*. A view is a drawable element in your UI layout such as a button, image or (in this case) a text field. Each of these is a subclass of the *View* class. The subclass that handles text is *TextView*.

A *TextView* is created with an Android *Context* instance as its parameter. A *Context* is a handle on the system that can resolve resources, can access databases and preferences, etc. *Activity* inherits from *Context* so it is also an instance of this class. The text content is defined with *setText*. Finally, the *TextView* is provided to *setContentView* to display it. If an activity does not call this method (directly or indirectly) then no UI is shown and the system displays a blank screen.

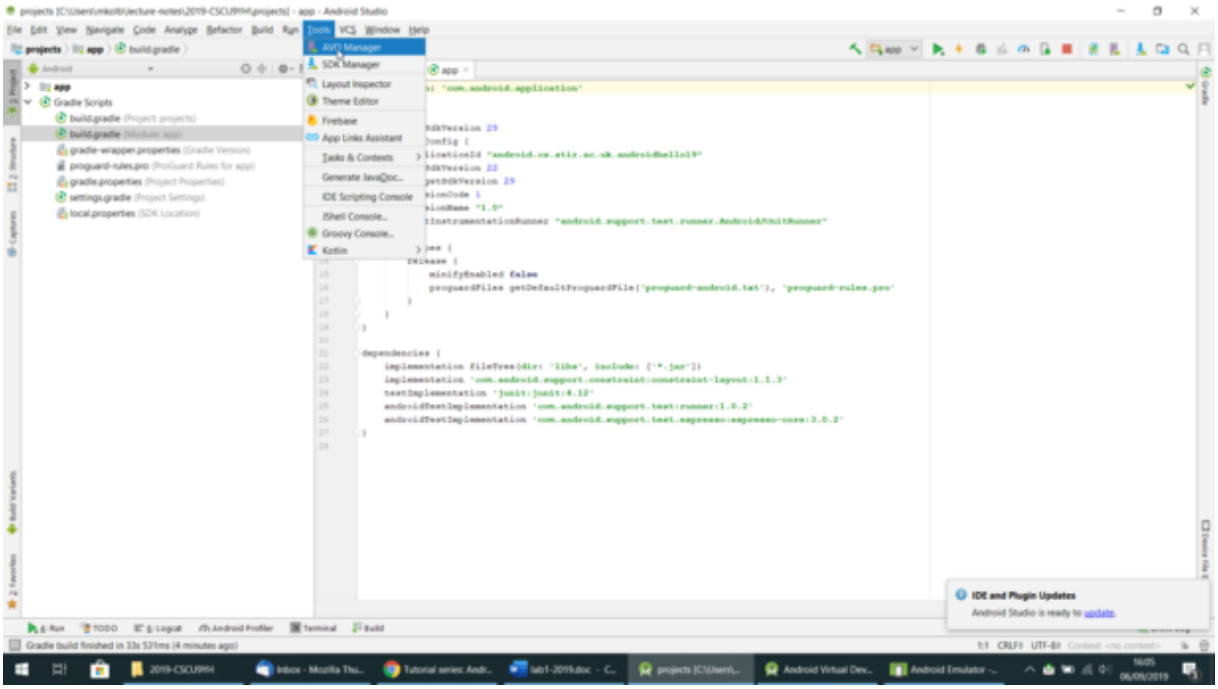
Create an Android Virtual Device

To try out your app you need to run it on a phone. Connecting a real phone is very time consuming and older (and very new) phones can prove problematic, so we will use a software

emulator. This imitates a phone and is very flexible as a wide variety of devices, screens, and Android versions can be accommodated.

Before running your app, it is a good idea to start up the emulator and keep in running. Treat the emulator as a phone and start it once in advance and leave it running, as (like a real phone) it will take some time for the phone to start – so please be patient. The emulator can take several minutes. The first time through you will need to set-up out some configuration parameters before you create the AVD (Android Virtual Device). This is in effect the emulator configuration you will run. This process defines the system image and device settings used by the emulator. To use an AVD from within Android studio:

- Choose *Tools* > *AVD Manager* :



- This allows you to create a virtual device, but there should be one already created for you.

- The first launch of a particular device configuration can be slow. Subsequent ones are relatively faster. When the emulator has finished booting, you will see the Android home screen.

In the future, do make sure you have an emulator running before you try to run an Android project.

Run the Application

Android Studio automatically creates a new run configuration for your project and then launches the Android Emulator if you have not already started one. However as mentioned earlier we recommend you treat the emulator as a phone and start it once in advance and leave it running, as it will take several minutes for the phone to start. When the emulator is booted, you will see the Android home screen. Android Studio then installs your application and launches the starting activity. You should then see your *Hello* application running.

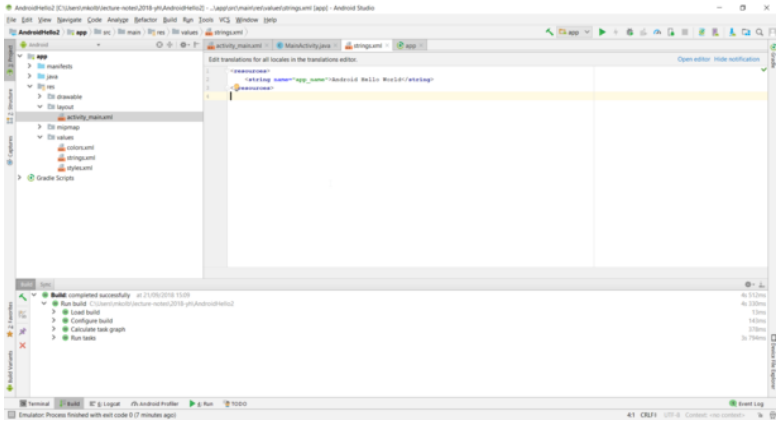
The 'AndroidHello2' you see in the grey bar is actually the application title. Android Studio creates this automatically (the *app_name* string is defined in *res\values\strings.xml* and referenced by *AndroidManifest.xml*). The text below the title is the actual text created in the *TextView* object.

Modifying the User Interface

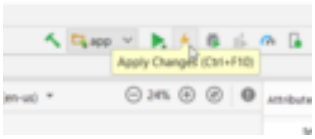
Modify your code to go back to what it originally said:

```
setContentView(R.layout.main);
  // TextView textView = new TextView(this);
  //  textView.setText("Hello Android");
  //  setContentView(textView);
```

Now open *strings.xml* under *res\values*. Your will see that the display string and *app_name* have been automatically generated. Click on these and alter their values, e.g. add a "**!**".

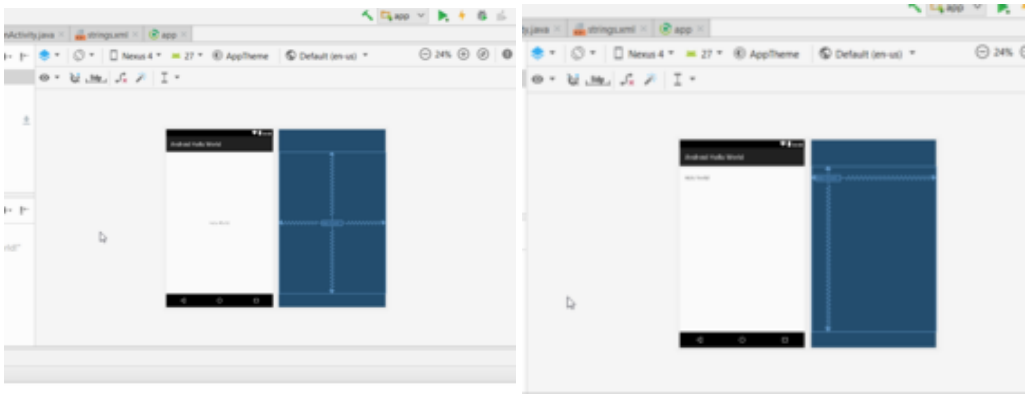


Now open activity_main.xml (you may have a different name) under *res\layout* and choose the *Text* tab from the bottom of the panel to see the XML that has been created (see figure below). Although you could edit

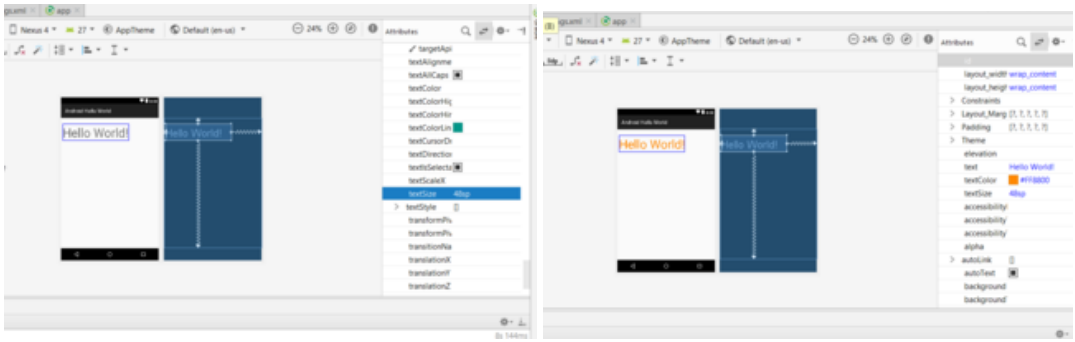


the XML directly, it can be easier to do this using the design interface provided by Android Studio. Switch to the graphical view by choosing the *Design* tab at the bottom of the edit panel to see the changes here too. Now save (shortcut Ctrl-S). Run the application again to see the result. Actually, Android Studio has a feature to push code changes to your app and see their effects on the fly without having to restart the app. Click on the Apply Changes icon rather than Run. It really is faster!

Stay in *main.xml* (you may have a different name) under *res\layout* in the *Design* tab. This allows to edit the user interface visually. You can edit existing elements, or can add new ones from the palette on the left. Right click on the text in the centre of the application window and drag it to the top left.



Confirm changes made to the textual (XML) representation of your layout (you should see some constraints being added). Then find the properties pane on the left hand-hand side and scroll down to *textSize* – change it to 48sp, where sp is scale independent pixels. Once you type in 48sp you will see that the graphical preview change. Choose the *Text* tab from the bottom of the panel to see the XML that has been created. Switch back to the graphical view by choosing the *Design* tab. Again save and run the application to check how it looks.



Staying in *main.xml* under *Design*, set the name *textColor* (in the *properties* panel) to the value *#FF8800* (red, green, blue values in hex). Your text should now appear in orange. Now switch to the *Text* tab and note the value of *textColor*. Again save and run the application to check how it looks.

Checkpoint!

You have now reached a checkpoint that you should show to a lab demonstrator.

If you have time, why not try more significant changes. Can you add another line of text near the bottom for example? What else, other than text can you add?