University of Stirling
Computing Science
Mobile App Development

Android Practical 1

Hello World

Warning!

Android Studio the development environment recommended for Android app development uses an Android emulator. This is the equivalent of running a real phones to experiment with your apps. It is best to think of starting the emulator as powering up your phone – it takes time so do it as infrequently as possible. Start it at the beginning, and leave it running.

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:

- developer.android.com is a good Android development reference.
- https://developer.android.com/codelabs/android-training-hello-world#0 contains a good tutorial to get you started.

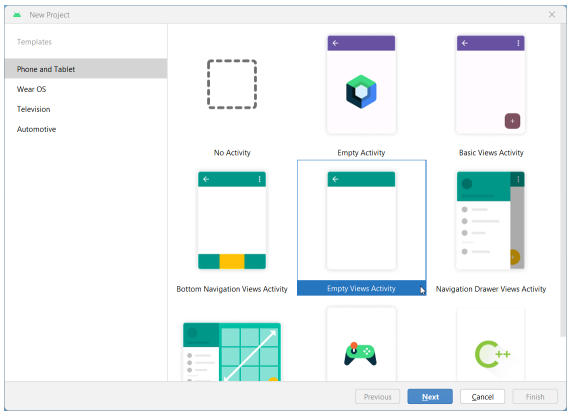
A screenshot at the start of each practical will show what your application should look like. Some of the practicals require you to create lengthy code. Rather than typing this from scratch, you can copy-and-paste it from the source provided.

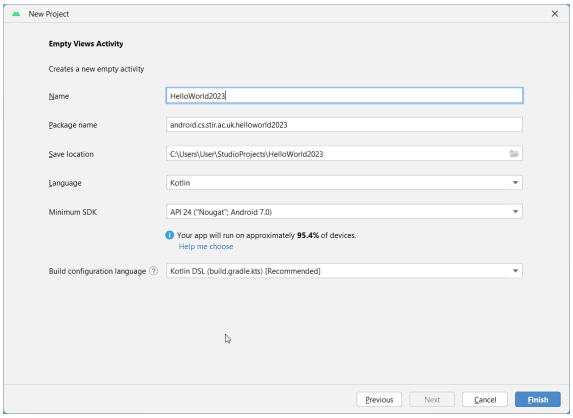
In this practical you will create a simple 'hello world' Android application.

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

Create a New Android Project

Choose an **Empty Views Activity** and give it a meaningful name. The language is set to Kotlin by default. All code provided for this module will be Kotlin. However, if you wish you may choose Java instead.





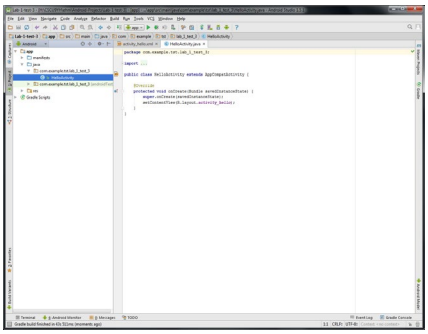
The screenshots are only indicative here – you may well want to use your own names.

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 android.cs.stir.ac.uk.helloworld2023

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```



Note that the class is based on the *AppCompatActivity* 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.

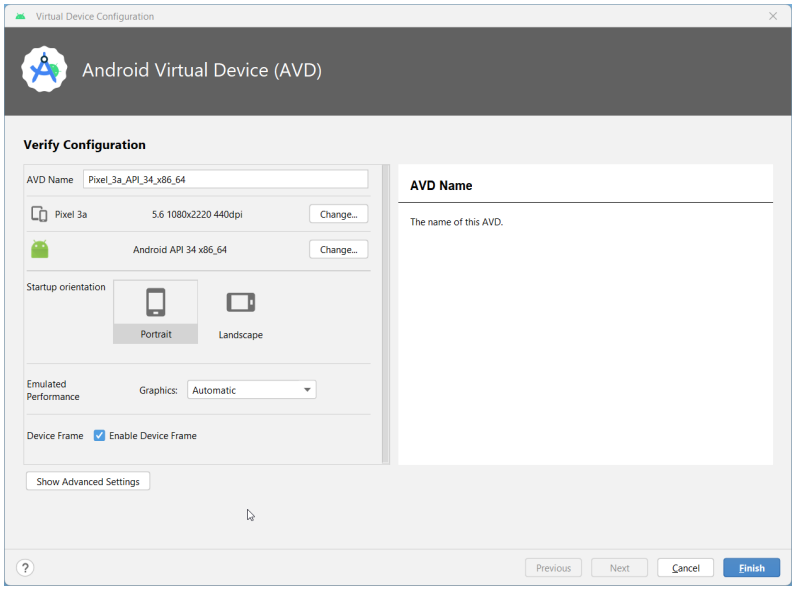
As the code provided for the empty activity is actually fully functional, lets try and execute this. For this to work, you will need to setup a virtual device of a phone.

Create an Android Virtual Device

To try out your app you need to run it on a phone. If you have an Android phone, you may be able to connect this to your computer using a USB cable (you will need to enable debug info on the phone). For the purposes of this exercise we will use an 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 then keep it 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 a few 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* > *Device Manager*
- This allows you to create a virtual device. Try and use similar settings to what is shown in the screenshot below (Pixel 3, API 34 (or 33)).



- 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.

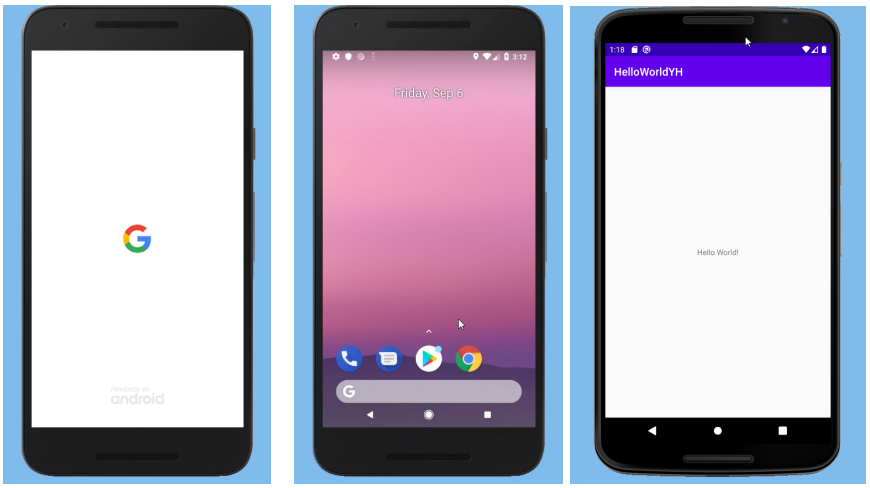
In the future, do make sure you have an emulator running before you try to run an Android project.

Running 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 above we recommend you treat the emulator as a phone and start it once in advance and leave it running, as it will take a while 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 'HelloWorld' you see in the blue bar is actually the application label. Android Studio creates this automatically (the *app_name* string is defined in *res\values\strings.xml* and referenced by *AndroidManifest.xml*).



Changing 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 android.cs.stir.ac.uk.helloworldyh

import android.os.Bundle
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val textView = TextView(this)
        textView.text = "Hello Android"
        setContentView(textView)
//        setContentView(R.layout.activity_main)
 }   }
```

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. Android Studio is pretty smart about them and may insert them automatically as you paste the code into the project.

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*.

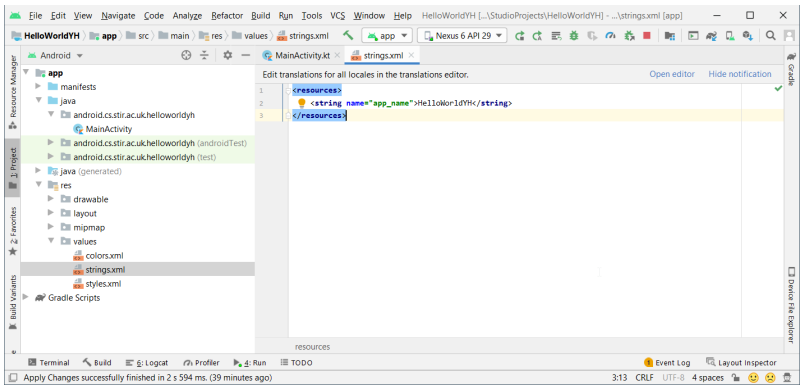
A *TextView* is created with an Android *Context* instance as its parameter. A *Context* is a handle on the system that can resolve resources, can access databases and preferences, etc. *AppCompatActivity* inherits from *Context* so it is also an instance of this class. The *TextView* is provided to *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.
Run your application again.

Modifying the User Interface

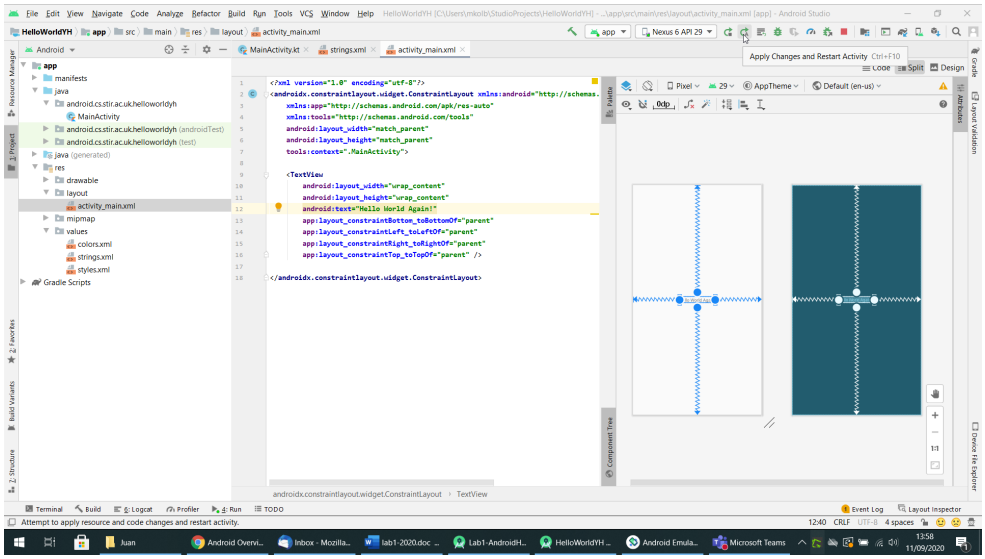
Modify your code to go back to what it originally said:

```
// val textView = TextView(this)
// textView.text = "Hello Android"
// setContentView(textView)
   setContentView(R.layout.activity_main)
```

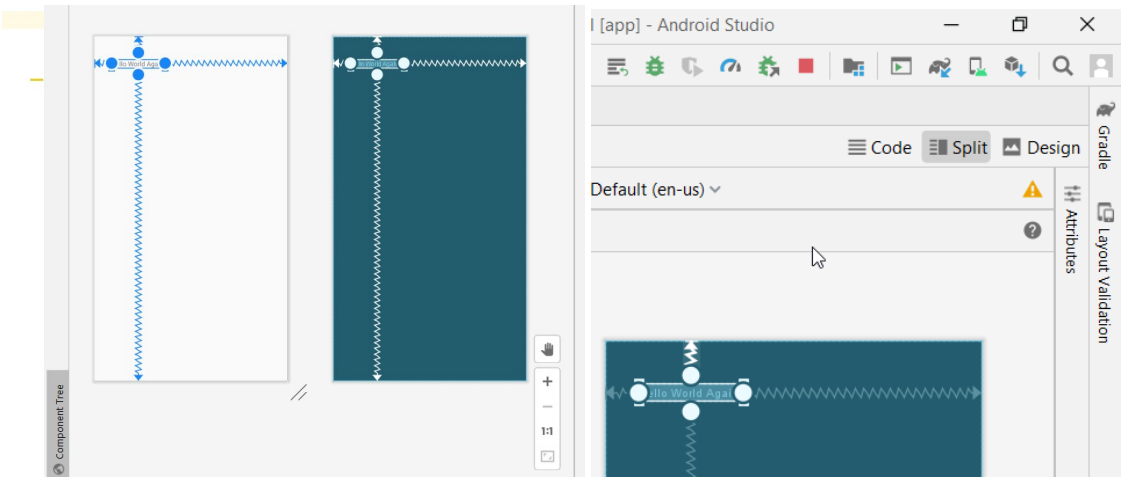
Now open *strings.xml* under *res\values* in the project hierarchy. Your will see that the string *app_name* has been automatically generated. Click on this and alter the values, e.g. add a "**!**".



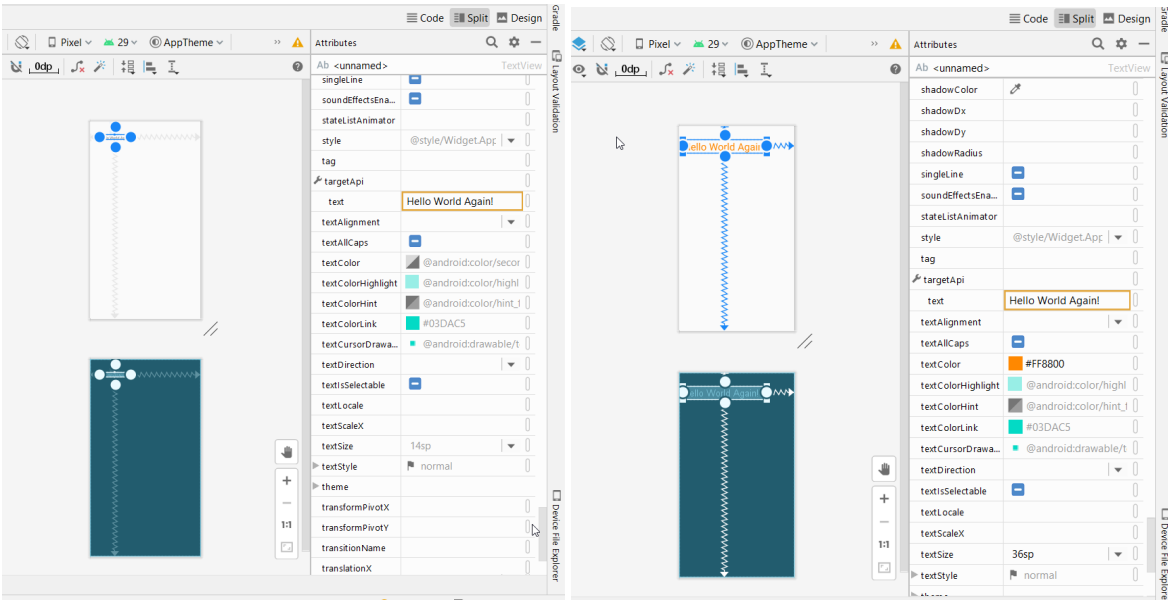
Now open activity_main.xml under *res\layout* and choose the *Split* option towards the right of the window to see the XML that has been created, but also the graphical view of the interface. Although you could edit the XML directly, it can be easier to do this using the design interface provided by Android Studio. Make a small change to the text string in the TextView. 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!



You can edit existing elements, or can add new ones from a palette. 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 changed). Click on the output text to select it, and then find the Attributes pane on the right hand-hand side and open up the All Attributes section. Scroll down to *textSize* – change it to 36sp, where sp is scale independent pixels. Once you changed it to 36sp you will see that the graphical preview changes. Check the *Text* section to see the XML that has been created. Again save and run the application to check how it looks.



Staying in *activity_main.xml*, make sure the output text is still selected. Set the attribute *textColor* (in the *attributes* panel) to the value *#FF8800* (red, green, blue values in hex). Your text should now appear in orange. Note the value of *textColor* in the XML. Again save and run the application to check how it looks.

Checkpoint!

You have now reached a checkpoint. Please record a very short video showing your app working and upload it to Canvas for Checkpoint 1.

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?