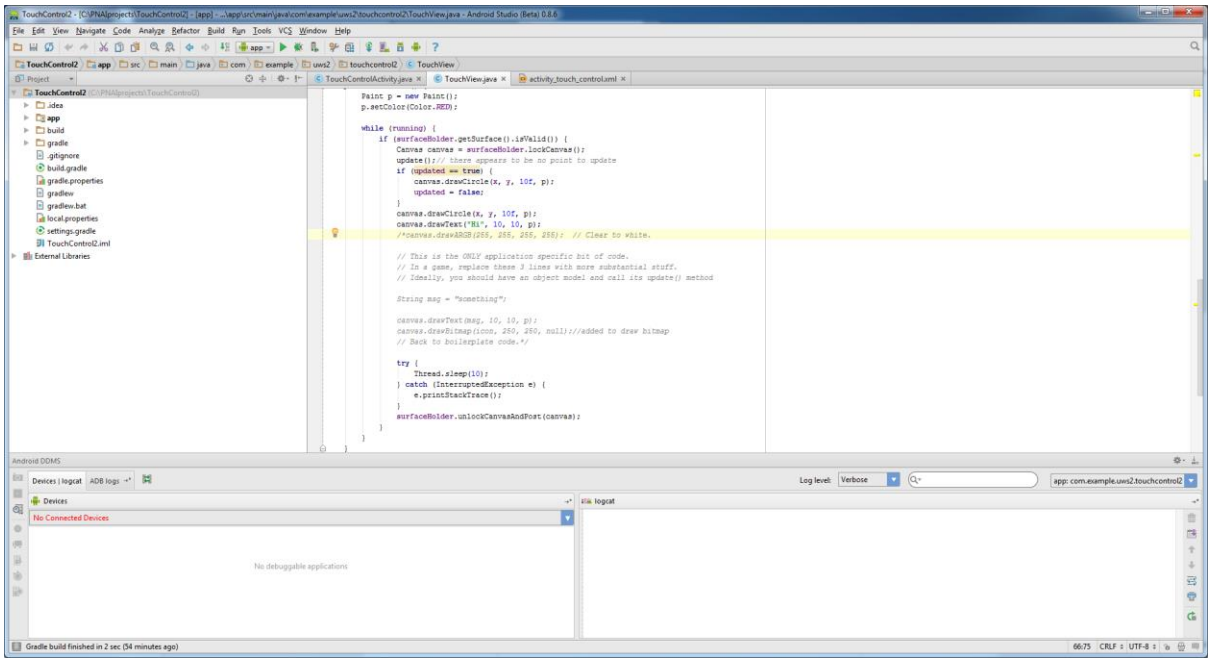
Lab 1: Introduction to Android Studio

Android Studio is the integrated development environment (IDE) that you will be using to develop and test apps for Android devices. It is the official Android IDE http://developer.android.com/sdk/index.html. In this practical session, you will:

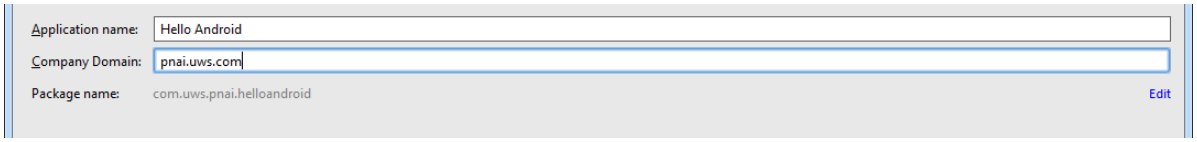
- Explore the Android IDE
- Create a simple application project for an Android device
- Create a new Android emulator and run the application in it
- Run the application on an Android device (if you have one and it is compatible)
- Implement a simple layout for a user-interface and manipulate this in program code

It is also recommended that after this class you install Android Studio on your own computer system (Windows, Mac OS-X or Linux) so that you can work on projects outside class.

Start-up Android Studio. What appears now will depend on whether the package has been used before. If it is the first time select “Start a new Android Studio Project” from the quick start menu. If you see a window like that below go for setting up a new project (File > New Project) from the menu system.

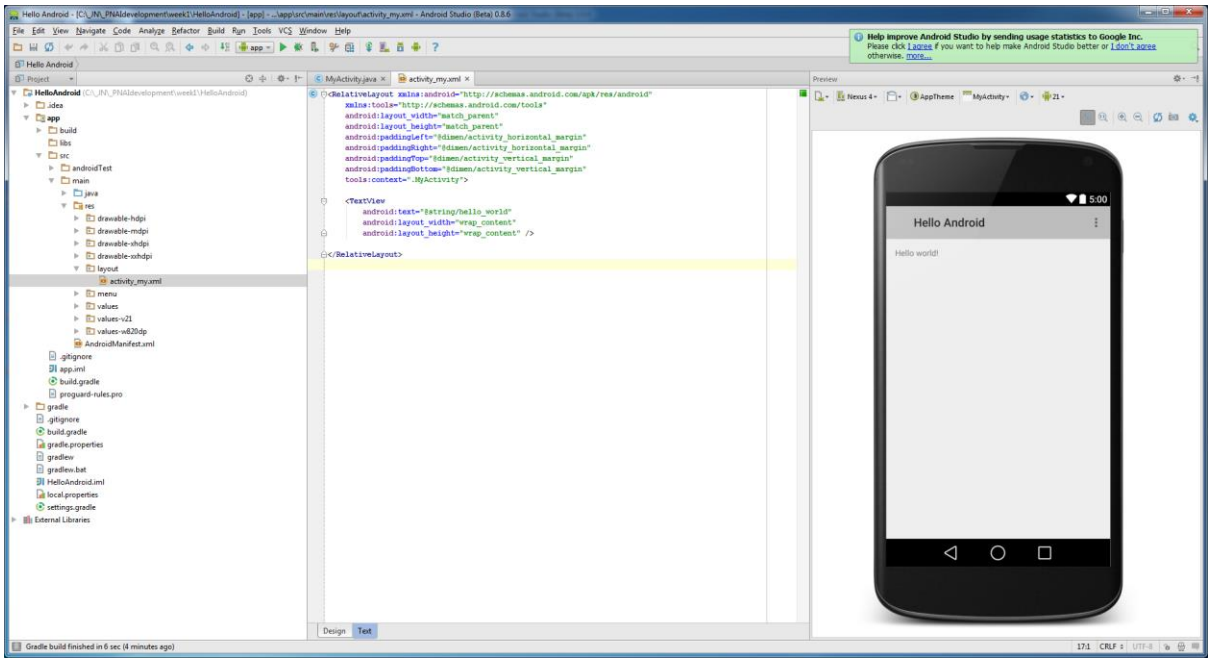


The following window will appear – enter details as shown for the Application name and Company Domain.



Press Next and in the next screen leave Phone and Tablet ticked and select a minimum SDK – choose 15 for the present and press next. The lower the level of API the more phones you will target (API 15 targets over 96.2% of phones, API 23 under 1%) but fewer new features you will be able to use.

Press Next and choose Empty Activity and press next again. The default activity names are okay for this example (but take a note of the layout name). Finally press Finish and the project will be created. This will take a while so be patient.



The window should look something like the one above although there might be variations depending on how the software is set up in the lab or which tabs are selected. If the project structure (on the left in the window above) is not showing View > Tool Windows > Project to make it appear.

In the central window there should be two "code" windows MyActivity.java and activity_my.xml each of which can be chosen by a tab at the top of the window. Usually activity_my.xml is open by default. There are two tabs at the bottom of the window – Design and Text. Learn to switch between these – the picture above shows when the Text tab is selected.

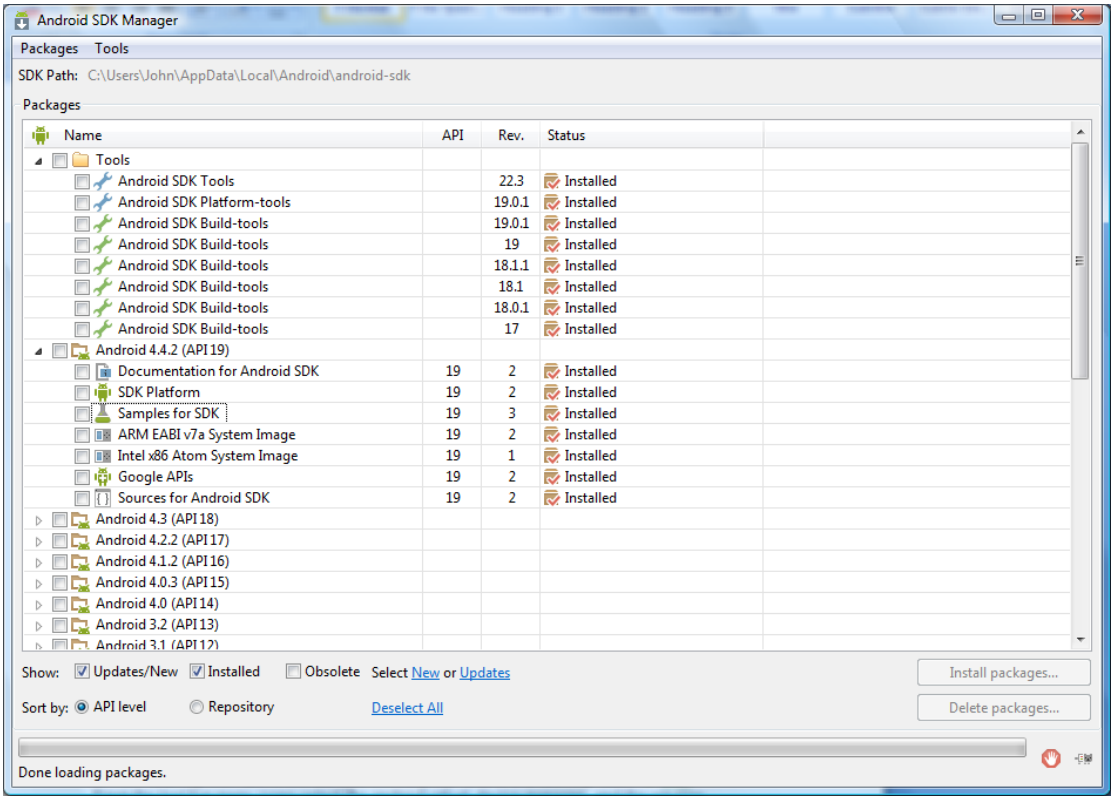
This preview window shows the initial layout of the app but does not provide any functionality. Before we look into doing this we will have a look at some more features of the IDE.

There are two pieces of software integrated into the IDE specific to Android development. They are the Android SDK Manager and the Android Virtual Device Manager (AVD). Locate these towards the right of the toolbar along the top of the development environment beneath the top menus. Tooltips should help you identify them. We will look at them now.

SDK Manager
Open up the SDK Manager (from the Window menu or tool bar icon). It will take a while to open up – so be patient. A window similar to the one below will open up (give it a while) showing the tools and APIs that have been installed.

In the lab most of the tools and APIs you need have been installed but on your own machine you will need to install them yourself. To load other versions of the APIs and tools select them and click Install packages – it can take a while. Take a note of the versions that have been installed so you can set up your own PC with the same versions (the lab should have at least 23 and 15).

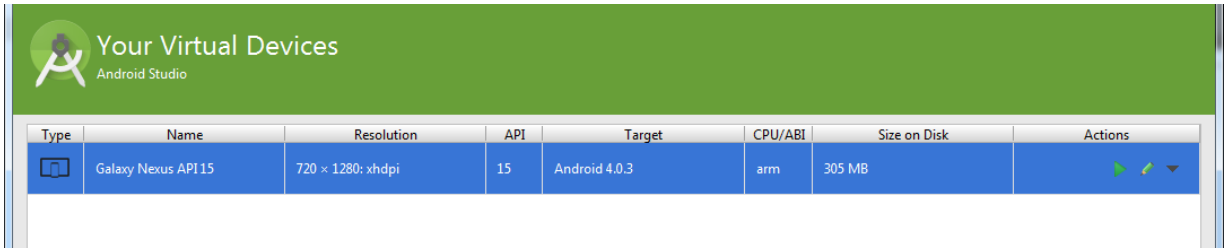
You should only need these two for this module. You would normally use the latest APIs for development but also have installed the version related to the oldest device you intend to target.



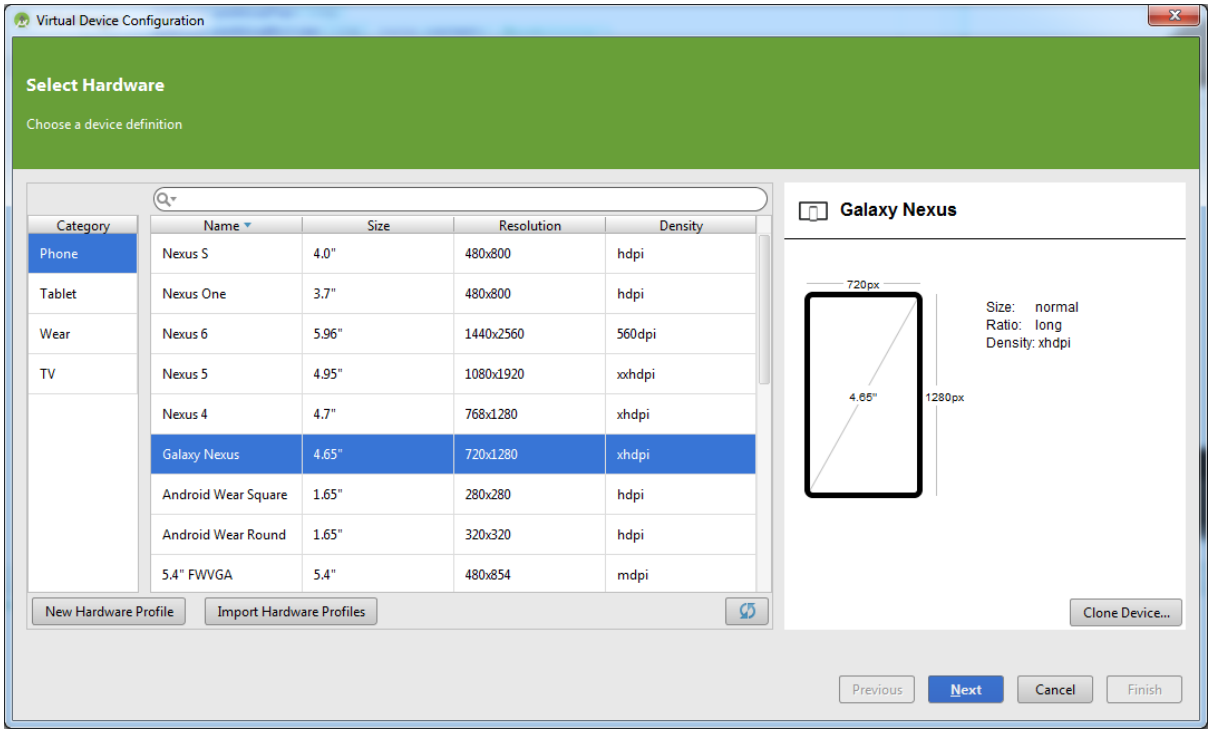
AVD Manager - Creating an Android Virtual Device (AVD)

Our next task is to create a new emulator that is software on your development machine that acts as an Android device. Each AVD emulates a different device and you can create a number for testing on different versions of Android.

From the tool bar icon start the AVD manager. Again if you wait a while a window similar to the one below will appear or you will be asked if you wish to create a device.



There might or might not be devices showing depending on the set up in the lab.
Press the Create Virtual Device button and select Galaxy Nexus.



Press next and scroll down to choose:

Release name: IceCreamSandwich
API level: 15
ABI: armeabi-v7a
Target: Android 4.0.3

Finish creating this device. This device is okay for us now and we will stick to it for this module. If you want to develop for a wide range of devices it is worth creating an emulator for an "old" phone – API 10 for Android 2.3.3.

Once you have created your emulator you can now close the AVD Manager window.

Before exploring the IDE further let's see if we can run the empty app the project created in the emulator and check the system works. In the tool bar click the green triangle, select the emulator in the Choose Device window you wish to

launch and click OK. Now wait and wait. You could read on while you wait for the emulator to start up because it can take a while. Check that the app runs in the emulator. Once the emulator is running do not close it down – it is much quicker to launch an app onto the running emulator rather than starting it up again.

The emulator view should look like the one on the left, although you might need to unlock the phone first.

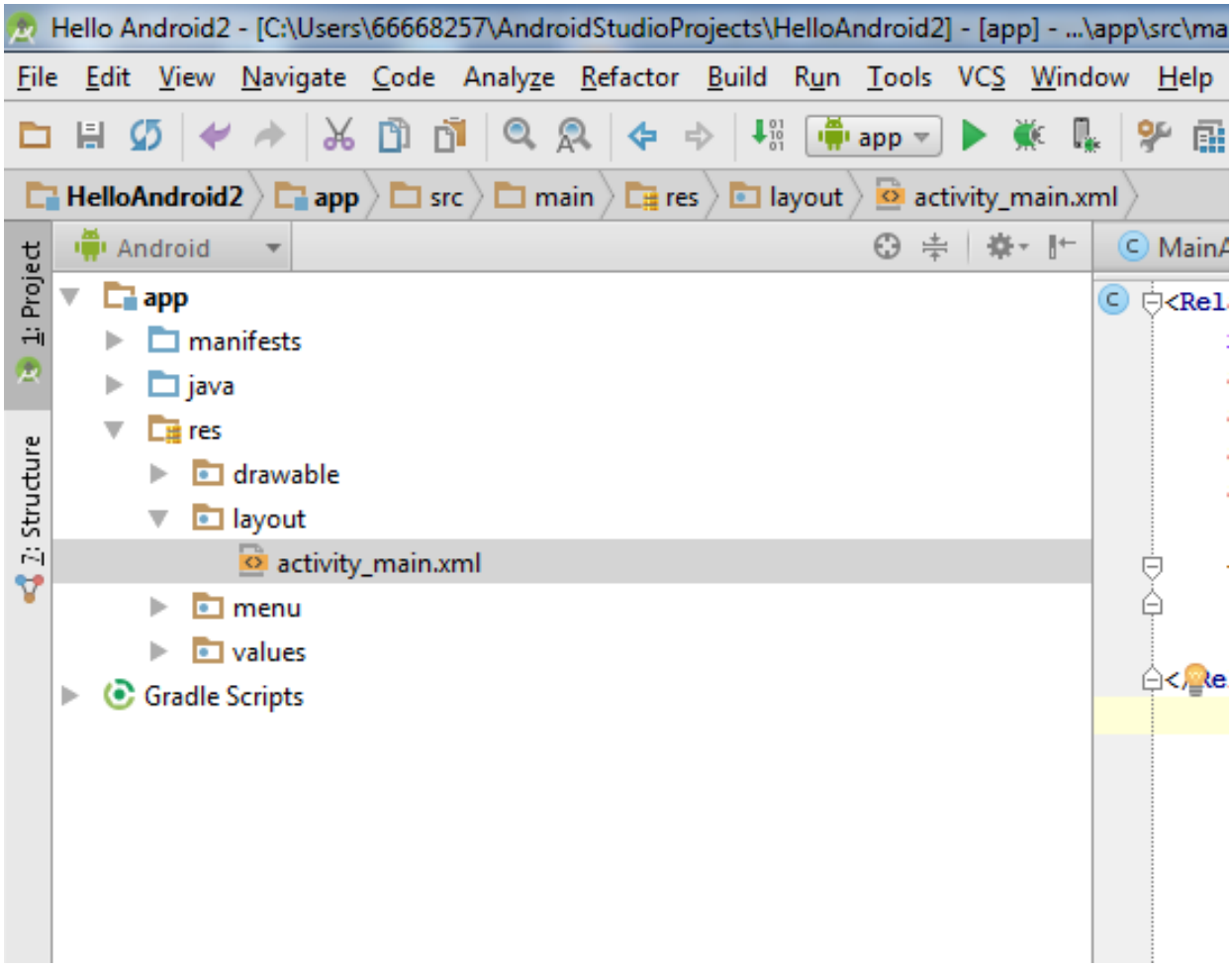
When you launch the emulator the first time the app does not always run and you may need to lunch it again from within the development environment.

You can stop the application by pressing the home button, which will return you to the emulator's home screen, or the back button which will return you to whatever screen the emulator was displaying before the app started. From here you can re-run your application by going to the Launcher screen, finding the app's icon and clicking on it.

Make sure you can run an app before moving on. The app you created has been "installed" on the emulator. If you go into settings and under apps you can see its installation details. How much space does it take up?

The Android Project

Now return to the project. You can see from the picture below that the application creation wizard has already created a structure to the project. There are over 400 folders and 1000 files. In time we will delve into more of this but for the moment locate the activity_main.xml file in the layout folder within res.



This is the file that is usually open in the code window and generates the output of the device. There are 3 important files at this stage

activity_main.xml in res > layout
MainActivity.java in java > com.example
AndroidManifest.xml in manifests

In the main code window select the java file MainActivity.java. As you might expect this file, created automatically when you formed the project, is written in java. You should be able to identify the

Package name …….
Class name …….
Class it extends ……..
The method that is created automatically ……

One of the frustrations of Android development is that there are frequent changes to the development environment which are often unexpected. In version 1.5.1 your MainActivity will extend AppCompatActivity. For this introduction to Android you would be as well just extending Activity. You could edit the file to make this change (and press Alt-Enter to import the appropriate library).

In Android the onCreate() method is called automatically when the app is run, so it is a good place to initialize any user-interface elements. Note that it is given a parameter of type Bundle – a class that defines an object which contains the state of the activity. This ensures that when an activity is suspended, it can be resumed into its previously saved state. More on this later.

Modifying the app
If you look at the code window there are two open files which make up the app. activity_main.xml which describes the layout and MainActivity.java which will determine the functionality. Select the xml file now which should have the following xml script (although it might be formatted in a slightly different way):

```
<?xml version="1.0" encoding="utf-8"?>
<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.uws.pnai.helloAndroid.MainActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!"
      android:id="@+id/tvHello" />
</RelativeLayout>
```

This is a nested set of instructions to create the layout – in this case a TextView created within a RelativeLayout. You should see where the *Hello World!* text is defined in the TextView via an attribute. Our short term aim is to be able to change what appears in that TextView from the java file. To do this we need to give the TextView an identifier so that we can refer to it.

Select the Design tab at the bottom of the window and select the *Hello World!* message on the screen layout. In the properties window scroll down to id and enter in the box to the right *tvHello* and press return. If you go back to the text view of activity_main.xml you will see the line added: android:id="@+id/tvHello"

This identifier is used to access the textView that shows the Hello message in code.
Return to the MyActivity.java file and alter the code in the onCreate() method of the class:

```
 @Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView tv = (TextView) findViewById(R.id.tvHello);
        tv.setText("Wecome to PNAI");
}
```

This links the TextView with id *tvHello* with a string that you define. Warning - If you cut and paste quotes from word documents you will sometimes find that you need to replace these in the editing environment.

You will notice that an error symbol appears next to the line beginning TextView and the text is highlighted in red. Hover the mouse pointer over the symbol and you should see the explanation "*Cannot resolve symbol textview"*. What this means is that the type TextView has not been recognized by the compiler. To correct this, we will need to add an Import statement to the top of the file. Right click on the error and the message to press Alt + Enter appears which should solve the error.

Now test the app in the emulator if all has gone well you should see the "Welcome to PNAI" message appear where "Hello Android" was. This might not seem like much but the message is now controlled in the java file rather than being fixed in the xml file.

Adding some Code to Hello Android
So far, there has been very little programming for this application, but we have covered a lot of Android development activities. Now we'll add some code to create a button:

```
public class MainActivity extends Activity implements View.OnClickListener{

        Button button;
        int touchCount;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                button = new Button(this);
                button.setText( "Touch me!" );
                button.setOnClickListener(this);
                setContentView(button);
        }

        public void onClick(View v) {
                touchCount++;
                button.setText("Touched me "+ touchCount +" times");
        }
}
```

Replace the main activity class by the code above. Remember the trick for importing libraries? You will need to use it now to remove any red highlighted errors. Test your app on an emulator and check that the button can count the number of times it has been touched.

Testing your app on a real Android Device
This is often fairly simple – just plug in your device to the machine you are developing on with a USB cable. You might need to install a device driver which windows will usually sort for you.

Enable USB debugging on your device. On most devices running Android 3.2 or older, you can find the option under **Settings > Applications > Development**.
On Android 4.0 and newer, it's in **Settings > Developer options**.

Note: On Android 4.2 and newer, **Developer options** is hidden by default. To make it available, go to **Settings > About phone** and tap **Build number** seven times. Return to the previous screen to find **Developer options**.

Homework
Install and configure Android studio on your own system. It is better to try this now and sort out any problems as soon as possible.

Appendix: Installing and running Android Studio on your own PC.
Android Studio Beta version is available for down load from the Android developer web site http://developer.android.com/sdk/installing/studio.html. The current version is 0.8.6. The download includes the development environment and the Android SDK.

You will also need to have a java development kit (JDK) installed on your machine as well as a java runtime environment (JRE). This can be downloaded from Oracle at http://www.oracle.com/technetwork/java/javase/downloads/index.html. It is recommended that you use a version of Java SE 7 (e.g. 7u79).

Within Android studio you might need to check that your projects are pointing to the correct folders for the Android SDK and the JDK. File > Project structure, select SDK location and navigate to the appropriate folders which will depend on where you installed the software.

Sometimes it can be tricky getting the initial set up correct for Android Studio. It is worth making sure you do this NOW so it is really ready when you need it.