

08309 Lab 1 Introduction to Android Virtual Devices

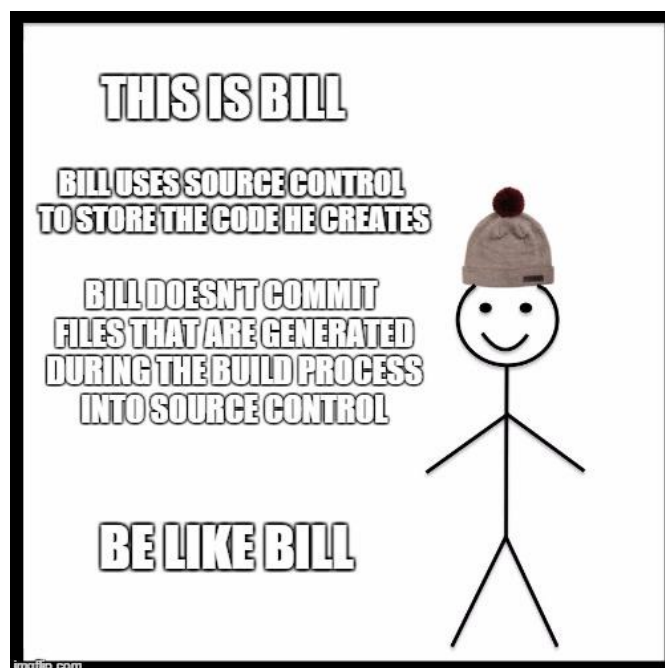
Goal

The goal of this lab is to introduce you to android studio and some of the development features it offers. By the end of this lab you should have gained experience in navigating an android studio project, creating an android virtual device (AVD) and using it to run your code, communicating with an AVD using telnet, debugging your code, and collecting and viewing debug output using logcat in the Dalvik Debug Monitor Service (DDMS).

Module Setup

Before we can begin you need to download all the lab code from Subversion. Create a folder for the module and then right click on it. Select the folder and check out your personal repository folder for the module. The URL of the repository is <https://visualsvn.net.dcs.hull.ac.uk/svn/08027-YYYY/XXXXXX> where XXXXXX is your six digit user ID and YYYY denotes the academic year (e.g. 1516, 1617 etc.). Inside this folder there is a folder for each week of labs. To avoid downloading the whole folder every week you may want to add /WeekZ where Z is the week number. It is likely you will want to download a new working copy each week because as there are issues compiling on a networked drive you may want to check out on the local drive instead. If you do this you must remember to delete your work before you log out.

Care has been taken to remove files that are generated as part of the build process. Please try not to add these back in to the repository as this can have a significant impact on the amount of data transferred to the repository. It also has the potential to cause source control conflicts that are best avoided if possible.



As a guide, the following files that are generated have been unversioned and added to the ignore list before your code was committed.

```
*.iml .gradle /local.properties /.idea/workspace.xml /.idea/libraries /build /captures .gitignore /app/.gitignore /app/build
```

To give you some idea of the significance of this the project folders for this lab include around 100 files taking up less than 500KB of disk space when they are first checked out from SVN. Once compiled the project folders used in this lab with contain over 3000 files taking up 40MB of disk space.

Lab 1.0 Android Virtual Devices

In order to run quickly Android Virtual Devices need to use Intel's HAXM virtualisation solution. This is incompatible with Microsoft's HyperV virtualisation solution. In the labs you *may* find that you have to turn off HyperV in order to run your AVD. If you do reboot the machine. When the machine reboots you should see an option to turn HyperV off. This is also the default option which will automatically be selected after 10 seconds.

Launch Android Studio. You may be asked to download the android SDK. All projects for this lab will be set up using API 16: Android 4.1 (Jelly Bean). This is a large download and install and may take some time. Feel free to read ahead while you wait. When android studio finishes loading select "open an existing Android Studio project", then navigate to the location of the code you downloaded and select "Lab1_1_HelloAndroid". This project is the classic Hello, World style project all ready to run, but before we can run this we need to create an AVD to run it on. Select Tools > Android > AVD Manager and then click Create Virtual Device in the bottom left corner. First we must select a device. Use the Nexus 5X. You will be asked what version of Android you want to emulate for your system image. Pick JellyBean 4.1 and Click Next. This will also require a download, but it take a lot less time than downloading the SDK. If you like you can take a look at some of the other settings available to you under Show Advanced Settings. However the default values are fine. Click finish.

Your new Nexus device should appear in the list of devices available. Click the green play icon on the left to launch your device. In the unlikely event that the emulator is too big you may wish to go back and adjust the scale settings. Occasionally the emulator is create "off screen". This issue seems to have been fixed in Android Studio 2 but just in case it is worth remembering some shortcut keys to move the emulator without using the mouse to grab the top bar (which is offscreen). The keyboard instructions are as follows:

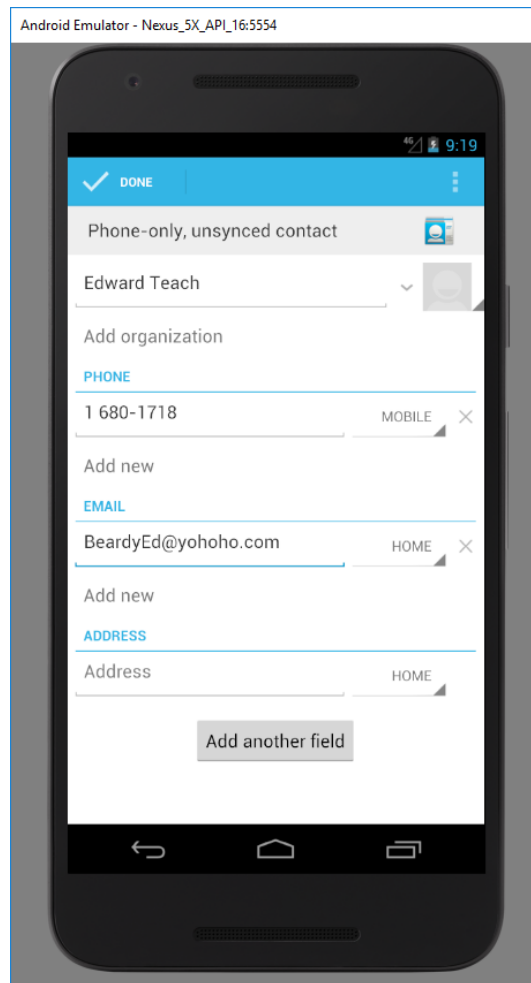
Alt-tab until the correct window has focus, then alt-space and M to select to move the window, then you can move the window using the arrow keys. Once the window has started moving you can also move it with the mouse.

A better solution would be to set the emulator up to scale itself so that it fits on your screen.



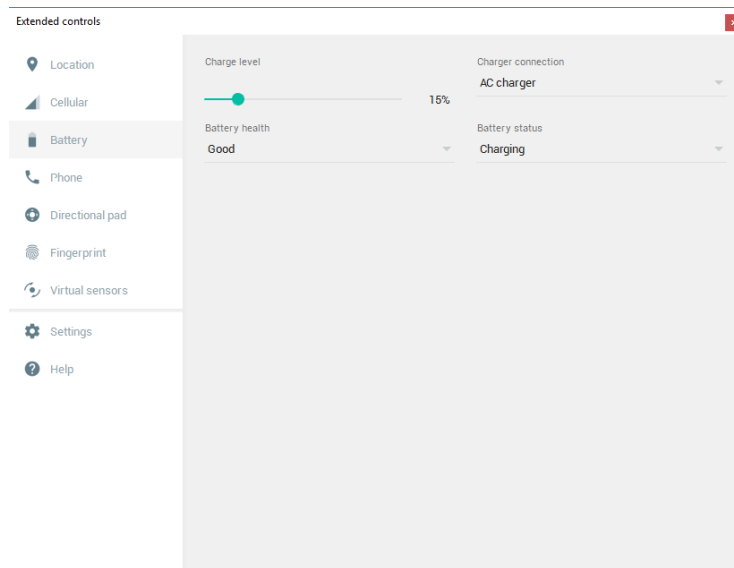
If your emulator launched correctly you should see something similar to the window above. The device's screen is shown on the left, and if the device has hardware buttons those should appear on

the right. There should also be buttons that allow you to do things like rotate the device. If you play around you will find that you can do many things that you would be able to do on a real device, like add a contact or go on the internet.

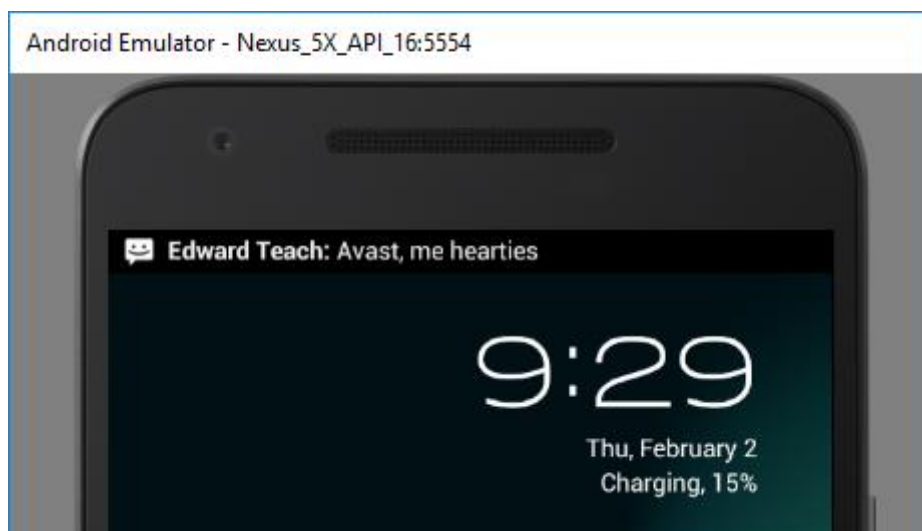


Note that if you do save contacts they will be retained even if you shut down and relaunch the AVD. You should also note that because of the way the labs have been set up data stored on emulators should not be considered private to you. Other users may be able to access this data. **Please do not populate any virtual devices with real data.**

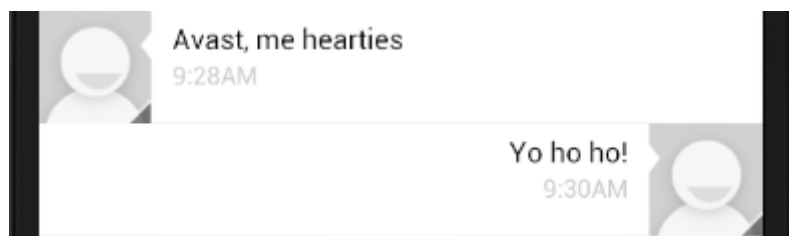
If you click on the ellipsis button (three dots ...) on the buttons control you should see more ways to interact with your virtual device through the extended controls menu. For example, under the battery tab you can change the properties of the battery in your virtual device.



You can even send text messages or phone calls to your AVD. You do this in the phone tab. If the phone recognises the number you sent the message from it will display the appropriate contact. Try it out for yourself!



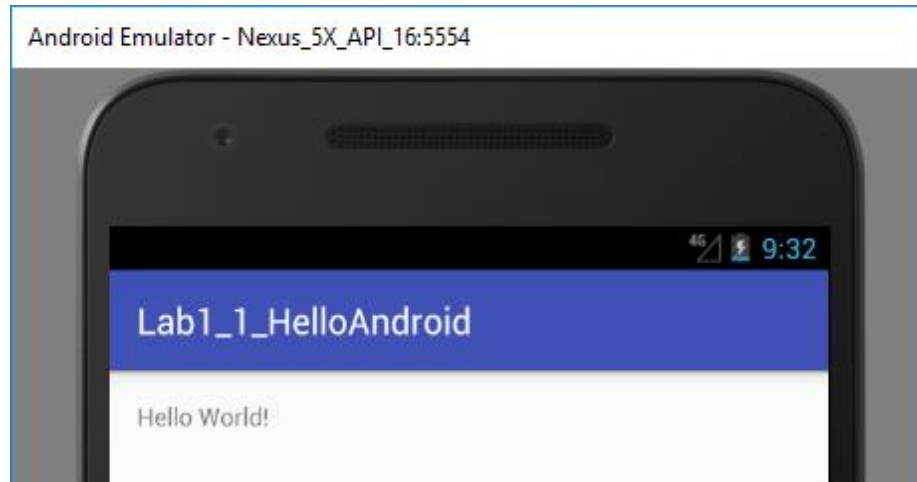
Just as you would expect on a real phone this message is stored on the device, and you can open your AVD's messaging app and see all the messages received by the device. You can even send messages back (although they won't go anywhere).



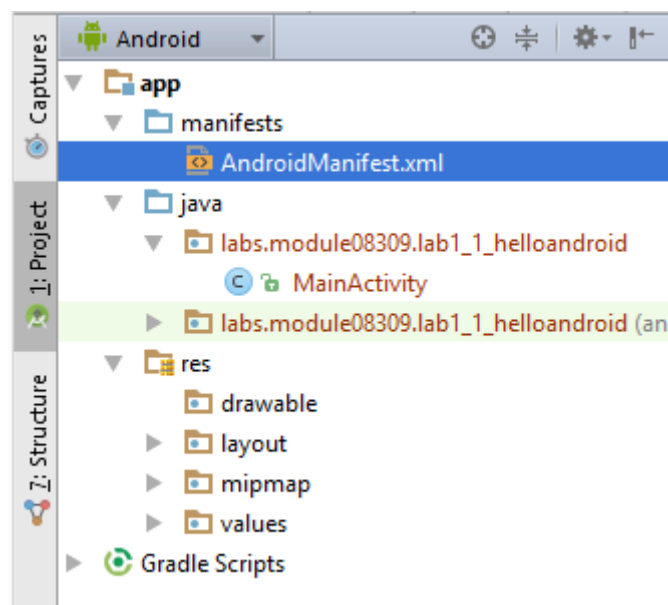
If you would like to know more about these tools please visit <http://developer.android.com/tools/devices/emulator.html>

Lab 1.1 Hello, Android

Now we've had a look at the emulator, let's take a closer look at the Lab1_1_HelloAndroid project. Click on the green play icon and you should be prompted to select a device. You can pick a device that's already running, or you can launch a new one from this window. Select the device that is already running and click OK.



Not very exciting, but let's look at the project files anyway. You may have to click on the project tab on the left. Beneath the app folder you should see a manifests folder, a java folder, a res (resources) folder.



Open the manifests folder and take a look at the AndroidManifest.xml file. The AndroidManifest.xml file contains details about the application and the activities within it. This is analogous to a project file in visual studio. As Lab1_1_HelloAndroid isn't a great name for an application let's change it.

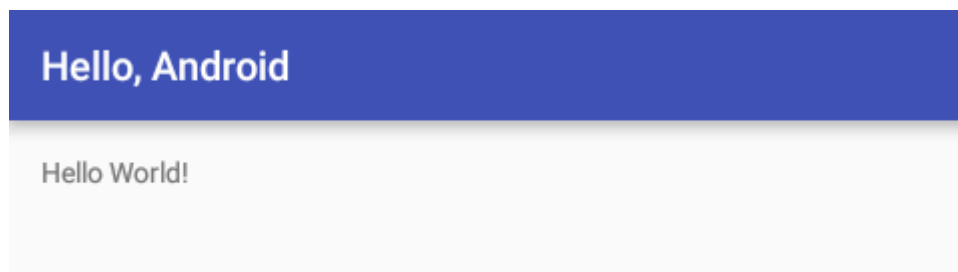
```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Lab1_1_HelloAndroid"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

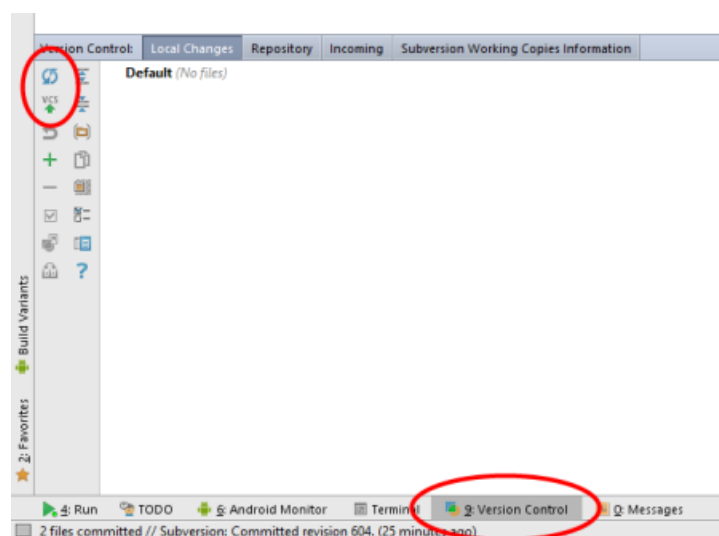
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

```

Change android:label="Lab1_1_HelloAndroid" to android:label="Hello, Android", as you try to do so you may notice that the grey "Lab1_1_HelloAndroid" turns green and changes to "@string/app_name". Go ahead and make the change anyway.



Commit your code into SVN. You can do this from within Android Studio by opening the version control tab on the bottom of the window and clicking the VCS arrow on the left to commit changes or through the VCS menu. Although it shouldn't matter, it's good practice to update your work whenever you start work. If you want to do an SVN Update through android studio use the refresh VCS changes icon that looks like a recycling symbol. Take care that the files being committed look sensible. Remember – be like Bill!

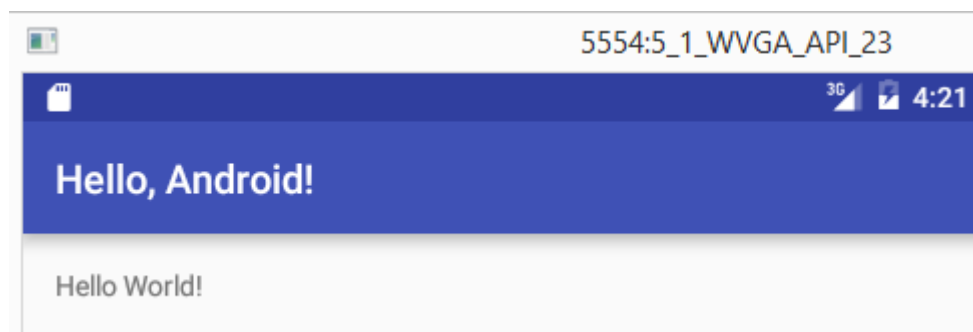


Alternatively you can also use tortoise svn through windows explorer to perform the usual commit function. Remember to leave an appropriate log message like *“Completed lab 1.1 task 1 changed app name to Hello, Android”*.

This looks much better, but it’s not good practice to put literal strings into these types of files. Let’s skip down to the res folder and open up the values folder. This is a single place that we can store values what we want to use again and again. Open the strings.xml file.

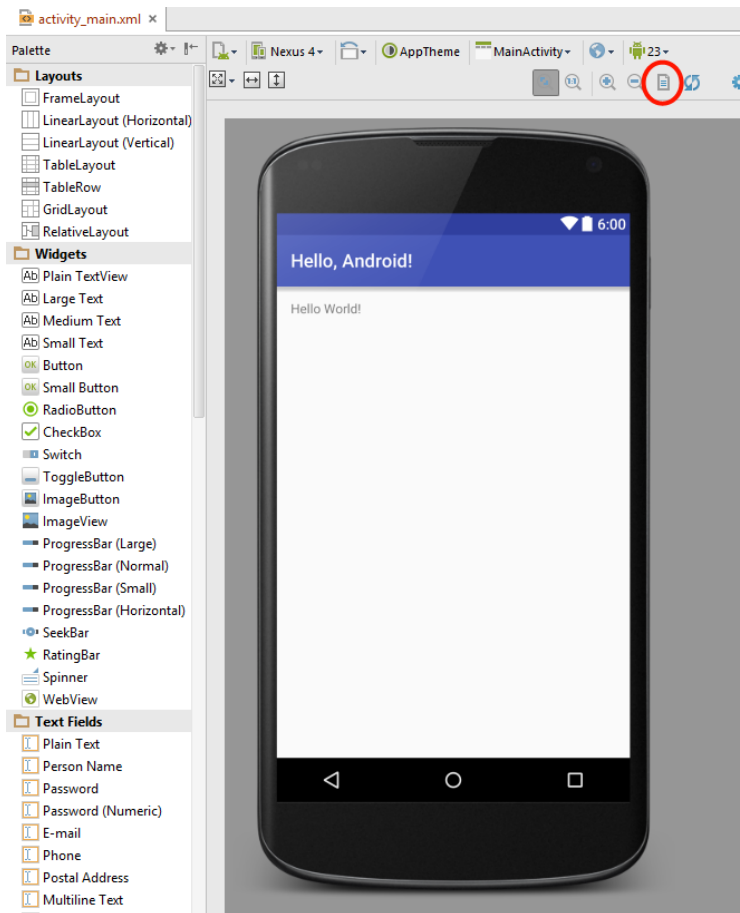
```
<resources>
  <string name="app_name">Lab1_1_HelloAndroid</string>
</resources>
```

You should see some xml with a string element with a name attribute of app_name. We’ve seen this somewhere before. Let’s change the value of the string element to Hello, Android! (with an exclamation mark so we can see a difference) then back in the AndroidManifest.xml file we can change the android:label attribute back to @string/app_name. Once you’ve done both of those things run your code on the emulator again.



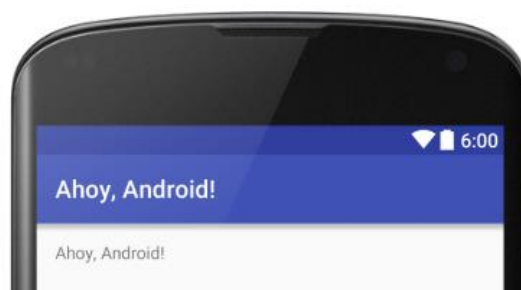
Commit your code to SVN with an appropriate message like *“Completed lab 1.1 task 2 changed the app name using an element in strings.xml”*

We’ve achieved something similar to what we did before, but now instead of having the string hard coded in the AndroidManifest.xml whenever we might want to use it we can reference the string in the strings.xml resource. To illustrate this point let’s change “Hello World!” to reuse our app_name string and also say “Hello, Android!”. To do this open the activity_main.xml file in the layout folder. This specifies the layout of our activity, but initially it probably won’t look very much like xml code. By default this file opens in a rich user interface editor, which writes your xml code for you. To get to the xml click on the page icon in the top right.



This should show you the xml that is generated by this user interface design tool. Note that there is a `TextView` that include and attribute `android:text="Hello World!"`. Change the value of this attribute to be `"@string/app_name"`. The change should be evident without even having to run your code on the emulator. Commit your code to SVN with an appropriate log message like *"Completed lab 1.1 task 3 reused string resource in text box"*

This is useful because now if we want to change the value of this string we can do it in one place, and our change will be propagated throughout our program. You may have noticed a pirate theme earlier in the lab. Let's get back to that by changing the value of the `app_name` string resource to "Ahoy, Android!". Go back to the layout and you can see your change right away in both the `AppBar` at the top, and also in the `TextView`.



Commit your code to SVN with an appropriate log message like *"Completed lab 1.1 task 4 changed app_name string resource – the single change was reflected in multiple points throughout the program"*

We will do more about layouts in a later lab. The goal of this lab is to get you comfortable with the using the IDE and AVDs.

The last folder to look at is the java folder. Inside that you will find the code for your projects. Find the file called MainActivity.java and take a look. There isn't much going on, but if you're not familiar with java then you might find some of the syntax confusing.

A screenshot of an IDE window showing the MainActivity.java file. The code is as follows:

```
package labs.module08309.lab1_1_helloandroid;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

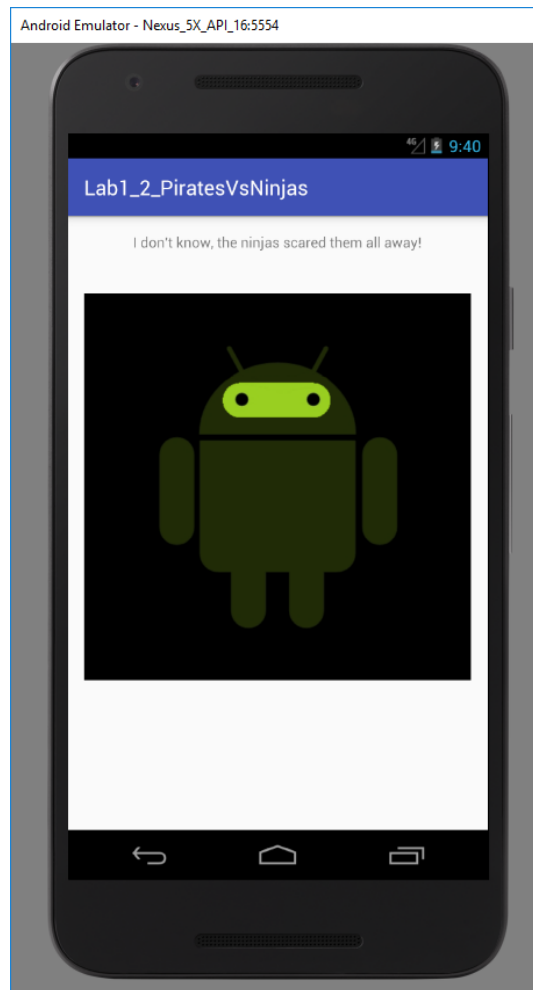
The IDE interface includes a tab at the top labeled 'MainActivity.java x', a left-hand sidebar with icons for package explorer, search, and run/debug, and a yellow highlight at the bottom of the code editor.

The line that begins with package is in some ways analogous to a C# namespace. It helps ensure that we don't have name clashes with classes with the same name, but in other namespaces. The import statements are very similar to C#'s using statements. Next we define a class. Notice the icon in the corner is a useful way to get to the associated layout file that we were just looking at. Extends is java's syntax for inherits. In C# this would just be MainActivity : AppCompatActivity, and the onCreate method we can see is decorated with @Override to show this is overriding a method in a parent. To the left of that the blue icon with the arrow pointing up allows you to jump to the method in the parent to see what is going on, although I doubt you will need to. Super.onCreate(savedInstanceState) calls the onCreate method in the parent class.

There's not very much to see here, so let's move on to a simple debugging exercise.

[Lab 1.2 Pirates vs Ninjas](#)

The purpose of this exercise is to give you experience of setting a breakpoint and running your code in debug mode to enable you to stop the program and step through it. Open the Lab1_2_PiratesVsNinjas project and run it in your emulator. You should see the following window.



We're trying to find out why pirates are called pirates, but we can't tell because this ninja has scared all the pirates away. Although the problem is trivial as is the solution we're going to go through the motions of setting a breakpoint, running our code in debug mode, and stopping and stepping through the code.

Open the MainActivity.java file and click in the grey border on the left to add a breakpoint at the start of the onCreate method. Then click on the bug logo (not the start arrow) or press shift + f9 to start debugging.

```
package labs.module08309.lab1_2_piratesvsninjas;

import ...

public class MainActivity extends AppCompatActivity {

    private final int mPirates = 1;
    private final int mNinjas = 2;

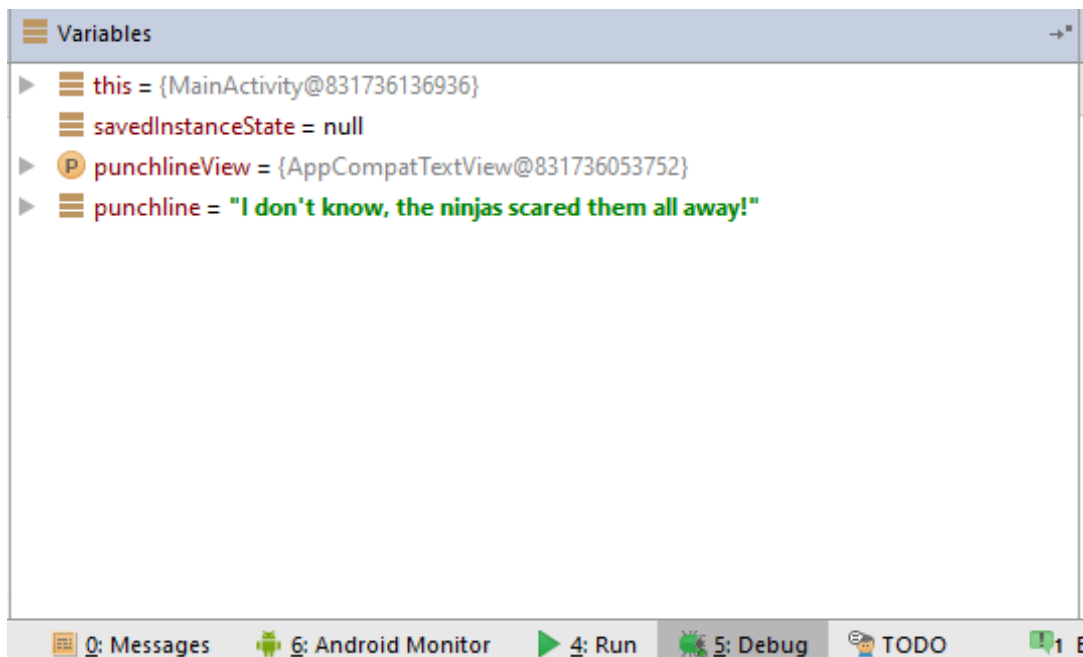
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView punchlineView = (TextView) findViewById(R.id.punchlineView);

        String punchline = GetPunchline();

        punchlineView.setText(punchline);
    }
}
```

Once the code has stopped look to the Debugger window. Keep an eye on the punchline variable.



Step through the code using the “Step Over” button or F8 key and note when the punchline variable changes.

Stop debugging and start debugging again. This time, when you reach the line that changes punchline use the “Step Into” button, or the F7 key. This should take you into the GetPunchline method. Step through this method, and see if you can find a solution to our (trivial) problem.

Once you’ve implemented your solution run your code again and you should be able to discover why pirates are called pirates.

Commit your code to SVN with an appropriate log message like *“Completed lab 1.2 task 1 debugged the code and found out why pirates are called pirates”*

Further Improvements

You might have noticed the strings that have been hard coded into the PiratesVsNinjas program. Use what you learnt in the previous lab to move the punchline strings out of the source code and into the resources file. To access a resource string from java code use the following syntax:

```
result = getString(R.string.punchline1);
```

This will retrieve the string with the name “punchline1”.

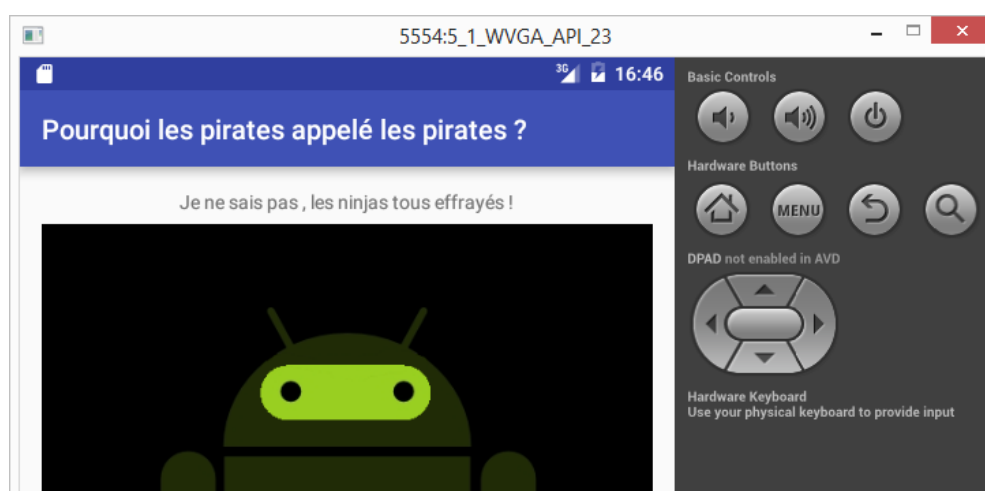
Commit your code to SVN with an appropriate log message like *“Completed lab 1.2 task 2 moved string literals out of the source code files and into the resource files”*

Even though these lines of code are used just once each there are other benefits from specifying string literals in this way. Mobile applications are often distributed through the internet and so can be made available in different countries all over the world. Let’s get this application ready for its French release. Open strings.xml and click Open editor in the top right. This will allow us to create translations for the strings we’re using. The appropriate strings will be selected at runtime according to the phone’s language settings.

Click on the world icon and select French, then use your favourite reliable online translation service to figure out what the French strings should be.

strings.xml x Translations Editor x			
+ Show only keys needing translations Order a translation...			
Key	Default Value	Untranslatable	French (fr)
app_name	Why are pirates called pirates?	<input type="checkbox"/>	Pourquoi les pirates appelé les pirates ?
punchline1	I don't know, the ninjas scared them all away!	<input type="checkbox"/>	Je ne sais pas , les ninjas tous effrayés !
punchline2	Because they Aarr!	<input type="checkbox"/>	

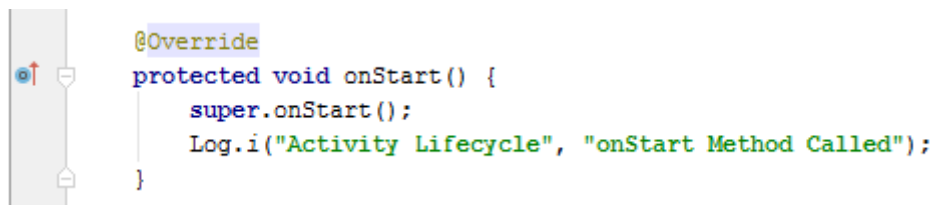
In your emulator find the settings menu and change the phone’s language to French. Then run your code.



Commit your code to SVN with an appropriate log message like *“Completed lab 1.2 task 3 added Spanish string resources and confirmed appropriate language is used for the phone settings”*

Lab 1.3 Activity Life Cycle

The final part of this week's lab concerns the activity life cycle. The aim is to demonstrate the activity lifecycle whilst also introducing you one of many debugging tools provided by the Dalvik Debugging Monitor System (DDMS). Open the project Lab_1_3_ActivityLifecycle and take a look at the code. The application is a simple Hello,World style application, but some code has been added to the MainActivity.java file overriding six parent methods. Those methods are onCreate, onStart, onResume, onRestart, onStop and onDestroy. In each case the method calls the parent method (using the super keyword) and then adds a Log message. Let's look at just one example.



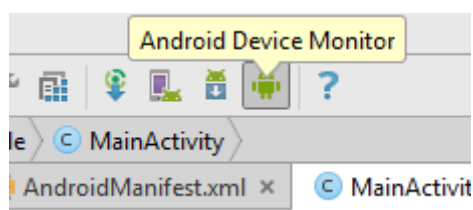
```
@Override
protected void onStart() {
    super.onStart();
    Log.i("Activity Lifecycle", "onStart Method Called");
}
```

The Log class allows us to send messages to the log output. Messages can be categorised as number of different ways for different purposes. In this case we are using I for INFO. To find out more about these categories please see <http://developer.android.com/reference/android/util/Log.html>

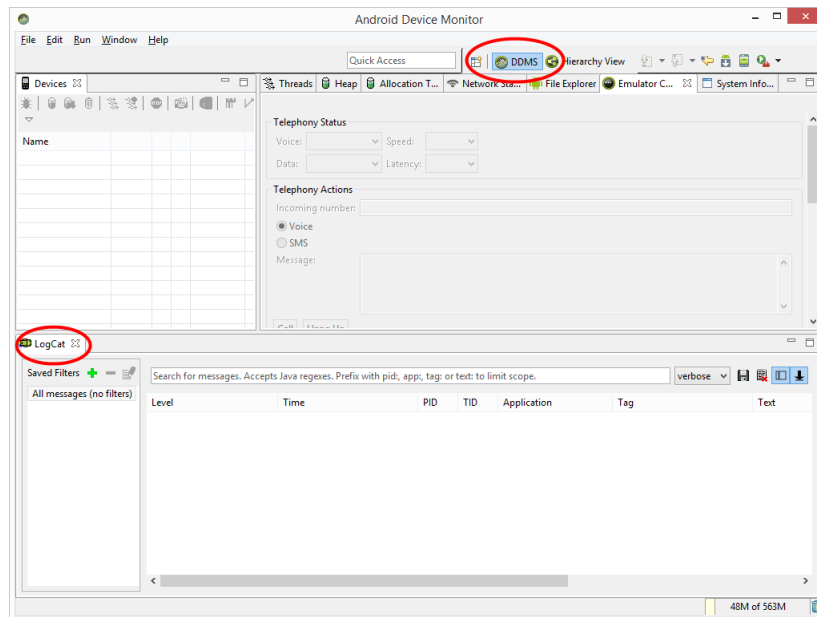
In this case we tag our message with "Activity Lifecycle". This will help us filter out all the other log messages so we can just see the ones we're interested. In this case the string we send to the log is "onStart Method Called" to tell us that the onStart method has been called.

One method is missing. Before we test this code to demonstrate the activity lifecycle we need to add an onPause method. Using other methods as a template add an onPause method, then commit your code to SVN. Remember to leave an appropriate log message like *"Completed lab 1.3 task 1 added an override for the onPause method that calls the parent's method, then sends a message to the log"*

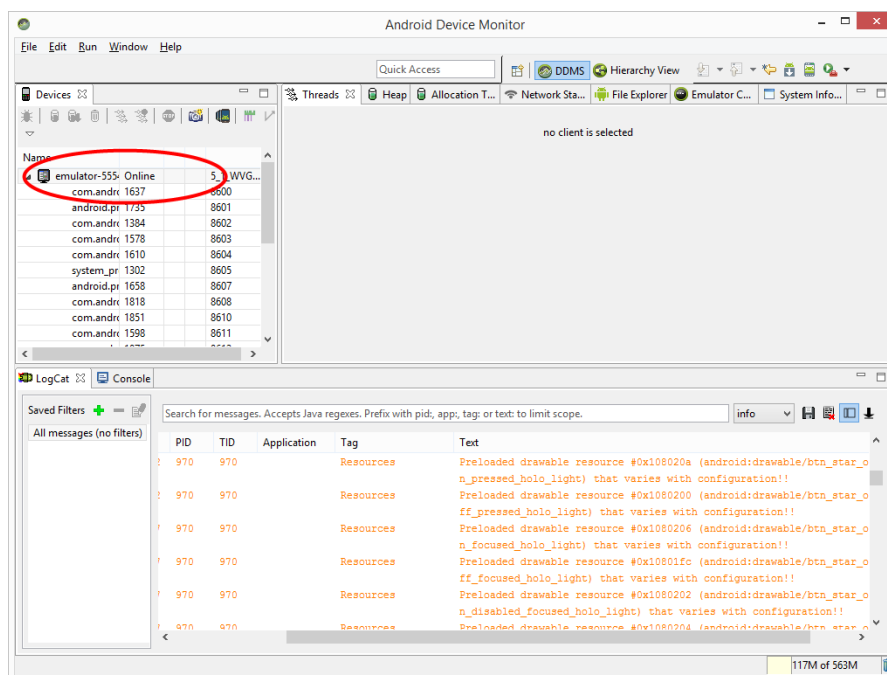
To run your code and read the log at the same time first we need to open the Android Device Monitor by clicking on the android at the top of the screen. You will also need to turn off hot swapping of code. To do this click File > Settings > Build, Execution, Deployment, Instant Run, and uncheck the first box. Remember to check this again later.



Once the Android Device Manager is open make sure that the Dalvik Debug Monitor Service (DDMS) is selected and that you can see the LogCat window.



Select the device that you want to monitor in the devices pane. You should see all sorts of messages in the LogCat window.

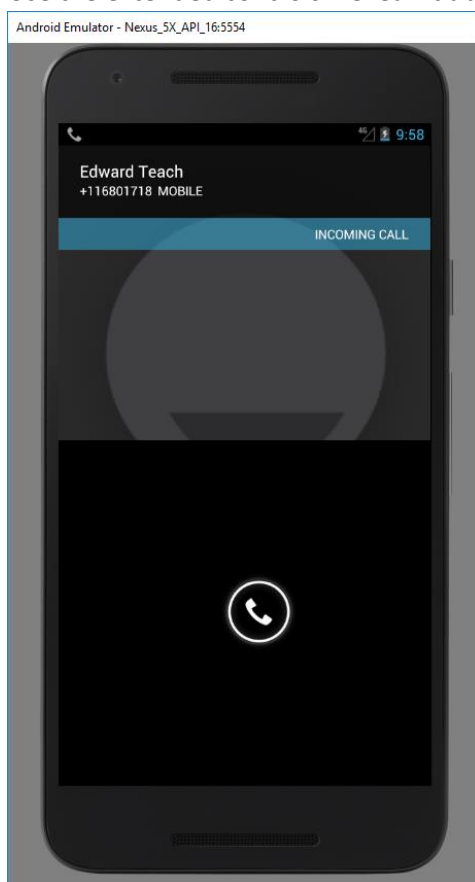


We categorized our log messages as “info” so make sure that is selected and in the filter box type tag:Lifecycle, which should only show us info messages with “Lifecycle” in their tags.

tag:Lifecycle							info				
Level	Time	PID	TID	Application	Tag	Text					
I	02-03 15:48:39.924	2137	2137	labs.module08...	Activity Lifecycle	onCreate Method Called					
I	02-03 15:48:39.924	2137	2137	labs.module08...	Activity Lifecycle	onStart Method Called					
I	02-03 15:48:39.924	2137	2137	labs.module08...	Activity Lifecycle	onResume Method Called					

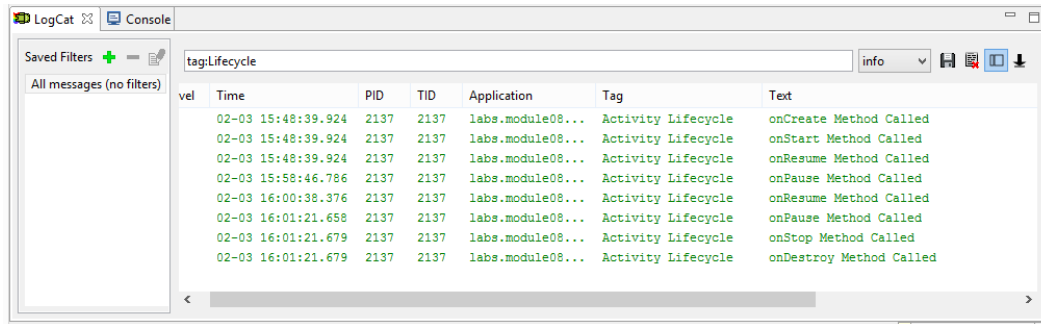
Here we can see that the onCreate, onStart and onResume methods have been called.

Use the extended controls we learnt about earlier to call the device.



Answer the call and note that the log in the device monitor indicates that the onPause method has now been called as the telephone application has come into focus. Hang up the fake call and note that now the onResume method has been called again.

Click the back button and notice that in our activity the onStop and onDestroy methods have been called.



Remember to turn hot swapping of code back on. To do this click File > Settings > Build, Execution, Deployment, Instant Run, and check the first box.

Summary

The purpose of this lab was to introduce you to android studio, launch an android virtual device, explore a simple android project and learn about some of the debugging tool that come with android studio. We've also taken a quick look at the activity class and the activity lifecycle. In the next lab we'll take a longer look at the activity class, how it works with the Android operation system and how it can interact with other activities.