CSCI 4100 Laboratory 1

Introduction to Android

Introduction

This lab covers the installation of the software required for Android development and two simple Android applications that will get you started. This ensures that you have everything that you need for the other Android labs in the course. By the end of this laboratory you will be able to construct simple single screen Android applications.

Android Installation

The Java programming language is used for Android development, so you must have a JDK installed on your laptop. I assume that you have already done this since Java is used in other CS courses. If not, do it now before going on to the next step. We will be using Eclipse as our development platform. You can use other IDEs and even the command line, but Android has been integrated into Eclipse and it provides a higher level of automation than the other IDEs.

I assume that you are already somewhat familiar with Eclipse and many of you will already have it installed on your laptop. You can use an existing Eclipse installation for Android development (the instructions for doing this can be found at http://developer.android.com/sdk/index.html), but I recommend using a separate Eclipse installation for Android. This reduces potential conflicts with other addons, and will probably result in a more stable development environment.

Google provides a single zip file that has most of the tools that you will need for Android development. You can download this zip file from http://developer.android.com/sdk/index.html . Once you have done that extract the zip file to a convenient folder, I used c:\CSCI4100\development. I am using the windows version of the development environment for this laboratory, but there are also versions for Linux and Mac OS X which behave in basically the same way. Note: this file will be available on one of the servers in the laboratory, but it is a good idea to download it and do the basic installation before the laboratory period.

After you have unzipped the file, drill down through the folders until you find the eclipse folder and then start the version of Eclipse in this folder. Once you have specified your workspace you will get a start screen, which you can dismiss. After this you will be in a close to standard Eclipse environment as shown in figure 1. I have circled two icons on the tool bar which are important for Android development. The one the left is the Android SDK Manager and the one on the right is the Android Virtual Device manager. Click the Android SDK Manager and you will get the dialogue shown in figure 2. Note that it already wants to install two packages and delete another. The default is to only include the latest version of Android, which is version 4.4W (or API level 20). The tablets that we will be using in the laboratory are on version 4.1.2, so you should also download this version of the platform as well. It is useful to have the SDK

documentation and the sample programs, so I have selected them as well. The selections that I made are shown in figure 3. If you have your own Android device you may want to install the packages for its version of Android as well. Once you have made the selections click the install button to install them.

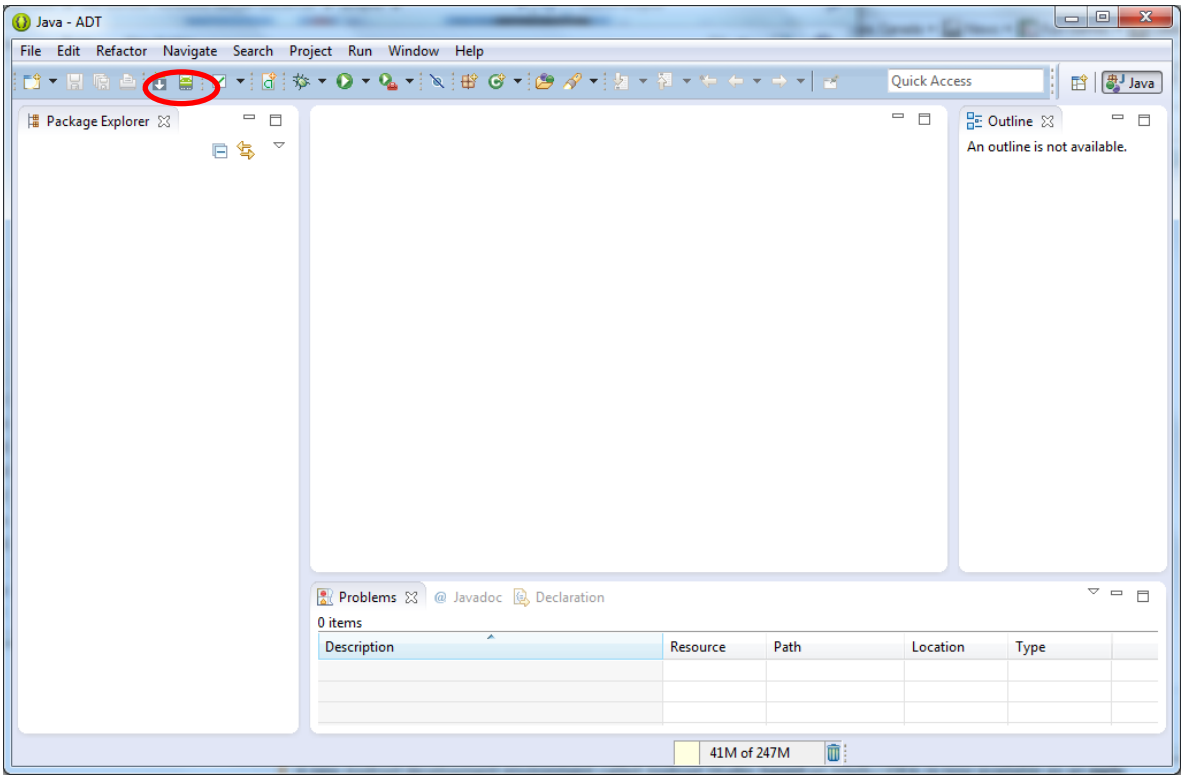

Figure 1 Android Eclipse environment

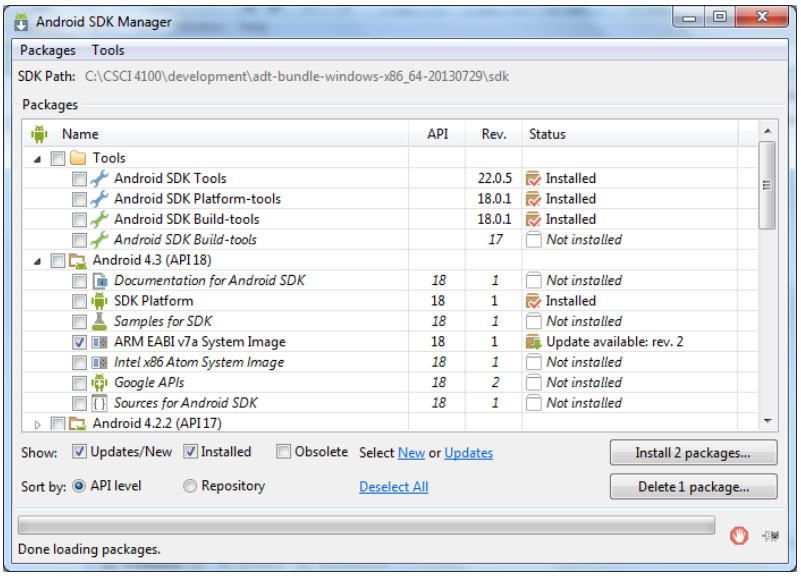

Figure 2 Android SDK Manager

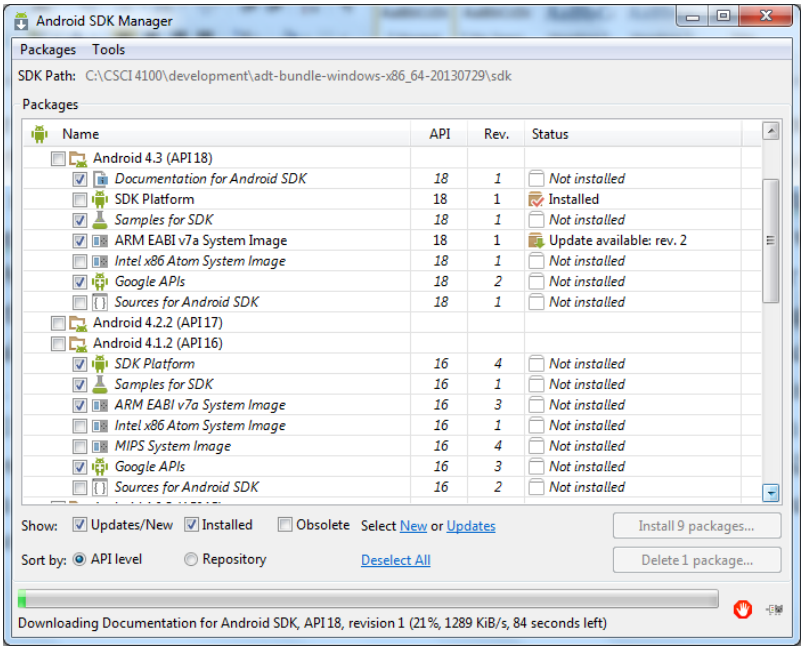

Figure 3 Selected packages for installation

The Android Virtual Device Manager is used to set up software emulators that you can use to test your applications. These emulators allow you to run your application on your laptop without having an Android device. Even if you do have an Android device you may find this useful for debugging, particularly when you start developing a new application. Click the Android Virtual Device Manager button and you will get the dialogue shown in figure 4.

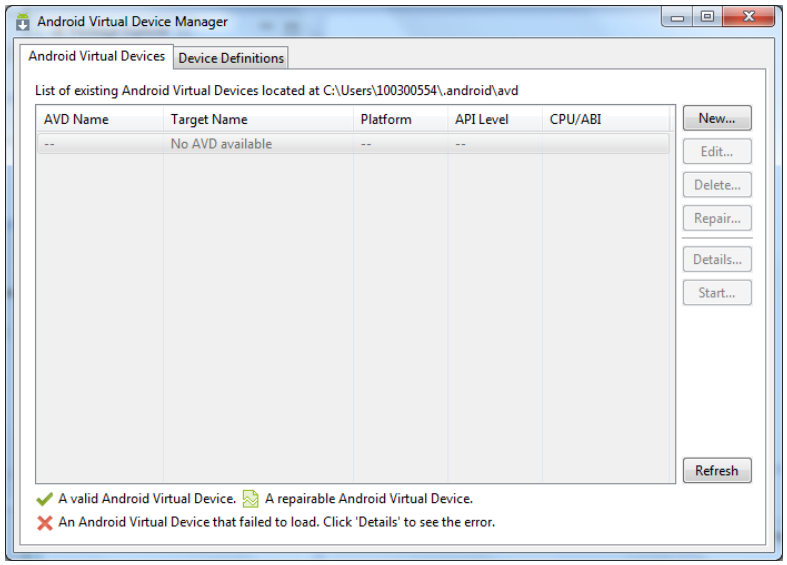

Figure 4 Virtual Device Manager

This dialogue has an empty list of devices, since we haven't defined any devices yet. We will need a virtual device later in this lab, so we will create one now. Press the new button and you will get the dialogue shown in figure 5.

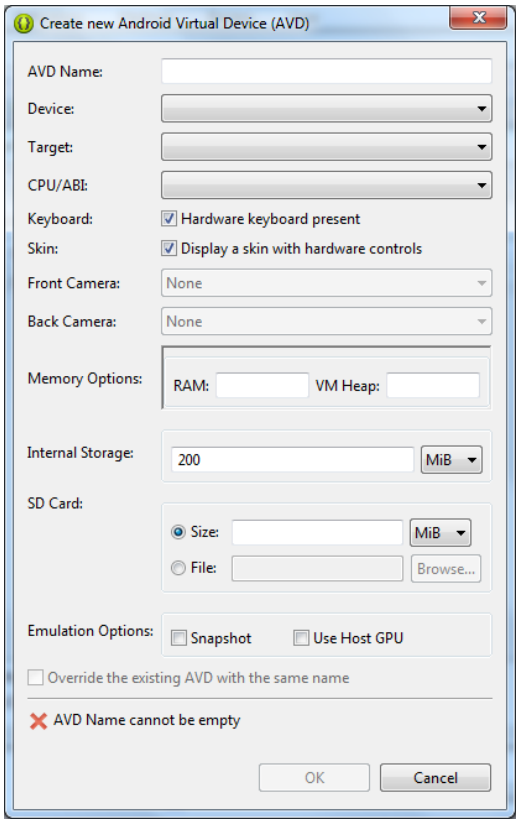

Figure 5 Dialogue for creating a new virtual device

I named my device Test_Device and from the device list I choose Nexus S which filled in most of the rest of the fields for me. When selecting a device for testing choose a device with a relatively small screen size, so it doesn't take up most of your laptop screen. This makes it easier to see the emulator and other windows at the same time. For the target I choose Android 4.3. At this point you can press the OK button to create the device. You can have more than one virtual device, but we only need one for this laboratory. We are now ready to build our first application.

First Android Application

To start building your first application go to the new icon (the left icon on the tool bar) and select Android Application. This will produce the dialogue shown in figure 6. I've changed the Compile With entry to version 4.3, since the default of 4.2 we haven't installed. Now we have the top three fields to fill out. The first of these fields in the application name. This is what the users of our application will see. We can pick any name that we like here, so we will use Lab One App. The second field is the name of the project, which will be used inside of Eclipse and as a directory name. The name is usually related to the application name, so we will use

LabOneApp (which Eclipse automatically fills in for us). The package name is a bit more complicated. This name must be unique on a device, since this is what Android uses to identify your application. If you only intend to run the application on an emulator or your own Android device you don't need to be too concerned about this. But, if you are doing a commercial application, you need to be careful that your application doesn't clash with that of another developer. This is usually solved by using the reverse domain name for the start of the package name. For example, you could use ca.uoit."*your name*".myfirstapp as the package name, where you substitute your real name into the package name in the appropriate place. The values that I used are shown in figure 7.

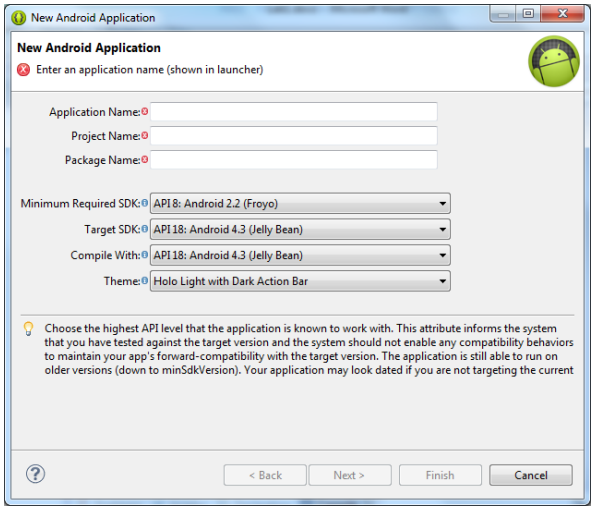

Figure 6 New application dialogue

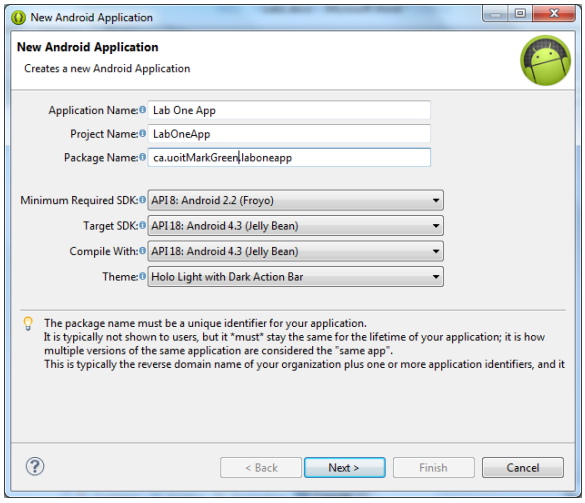

Figure 7 Filled out dialogue

Now click the next button, which gives a few options for the project that we don't need to change. Click the next button again and you will arrive at a dialogue where you can set your application's icon. We will just use the default icon for this example. Press the next button

again and you get to select the type of activity for your application; activities are the basic building blocks of Android applications. We want a blank activity which is the default, so you can press the next button again. The final dialogue allows us to change the name of our activity and a few other options, which we aren't interested in. You can now press the finish button, which creates the project.

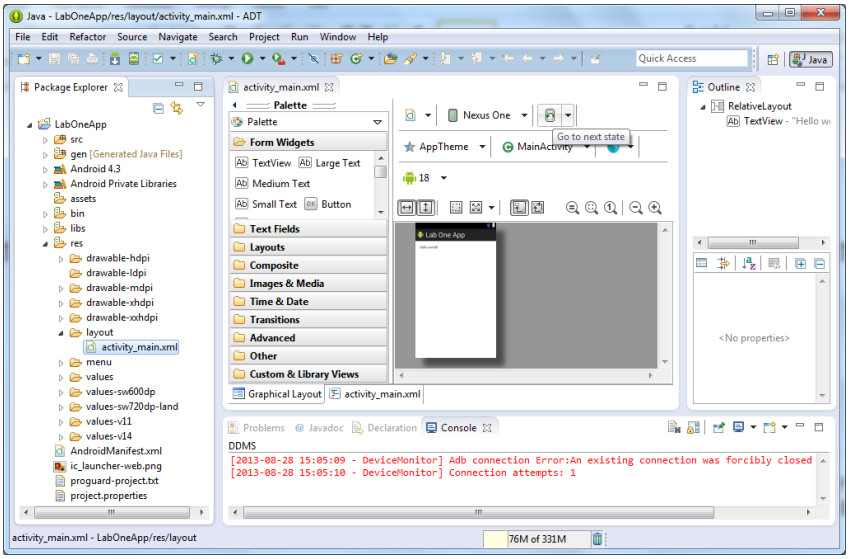

Figure 8 Our newly created project

The newly created project is shown in figure 8. Before we start exploring this project we can run the application by clicking the run button (the green and white right pointing arrow in the tool bar). The resulting application running in the emulator is shown in figure 9. Note that it takes a long time for the emulator to start, so be patient. During debugging don't close the emulator to avoid the start up time.

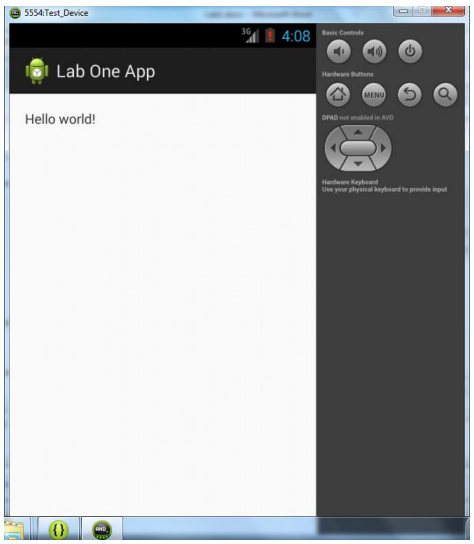

Figure 9 Our first application running in the emulator

The IDE creates a working application for us. It doesn't do anything interesting, but it shows that we have the basics right. Most of the smartphone platforms do this so we have a working starting point for our applications.

We can also run our application on a real device. As an example we will run it on one of the Android tablets in the lab. If you have an Android device, you can try running it on your own device as well. To do this we need to put the device in developer mode. This is done by going to the settings app and selecting the developer options tab. If your device has Android 4.2 or later you need to go to the about phone tab and then tap the build number 7 times before the developer options tab appears. In the developer options tab at the top of the screen first turn developer options on and the click USB Debugging (the tablets in the lab are already set up for this). Finally plug the tablet into your laptop using the USB cable. You are now ready to run your application on an Android device.

Now when you click the run icon the dialogue in figure 10 appears. The real device is shown in the top part of the dialogue, make sure that the device is selected and then press the OK button. This will start your application on the actual device. It should look basically the same as on the emulator.

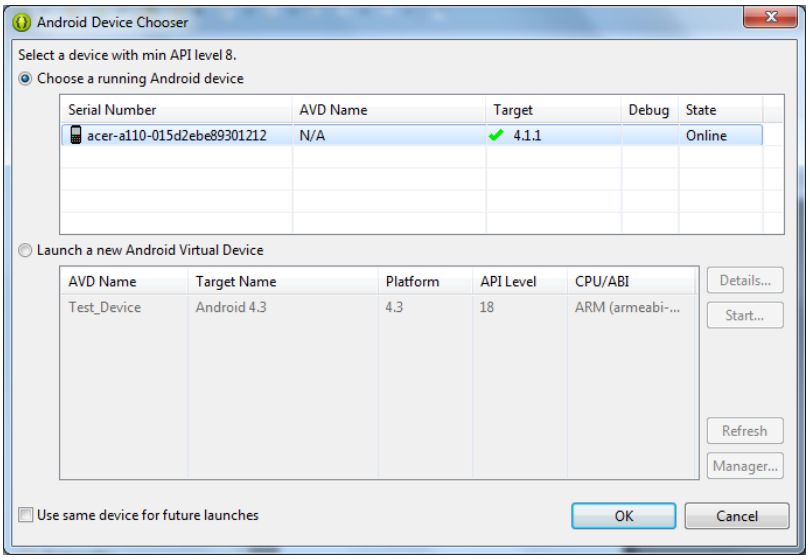

Figure 10 Selecting the device to run on

Now that we've had a chance to run our application, let's return to Eclipse and examine its major components. In figure 8 we see that Eclipse started in the UI layout editor. This editor gives us a view of the Android screen for our application. You may need to enlarge the middle pane a bit to get a clear view of the UI. The actual UI layout is stored in an XML file, which can be viewed by selecting the XML tab below the UI layout. Android uses a large number of XML files for configuration; this is just one of them. Take a quick look at the XML; we will come back to it later. Returning to the layout view, the panes on the right are used to edit the properties of the widgets in the layout. The upper pane is a hierarchical view of the widgets in the layout and the

bottom pane shows the properties of the widget that are selected in the top pane. Select TextView in the upper pane and examine its properties in the lower pane. Probably the only property of interest at this point is the Text property. On the left of the UI layout pane is another pane that contains widgets that can be added to the UI layout. Take a minute to explore this pane and the range of widgets that are available to you.

The pane on the far left is the Package Explorer that provides access to all the files and components in your project. When a new project is created we start in the layout folder of the res (for resources) folder, with the activity_main.xml file displayed. If we have multiple screens in our application there is a separate XML file for each screen in this folder. There are a number of other folders and file under res that we will come back to later.

At the top level you will find an AndroidManifest.xml file. This contains general properties for your application. Double click this file and an editor for the manifest file will appear in Eclipse's center pain. You can directly edit the manifest by going to the tab on the right, or use the other tabs for a friendlier editor for this file (I hate editing XML files). The first tab (on the left) of this editor has two important properties. The first is the package name, which you probably don't want to change. The other is the version code, which can cause some interesting debugging problems.

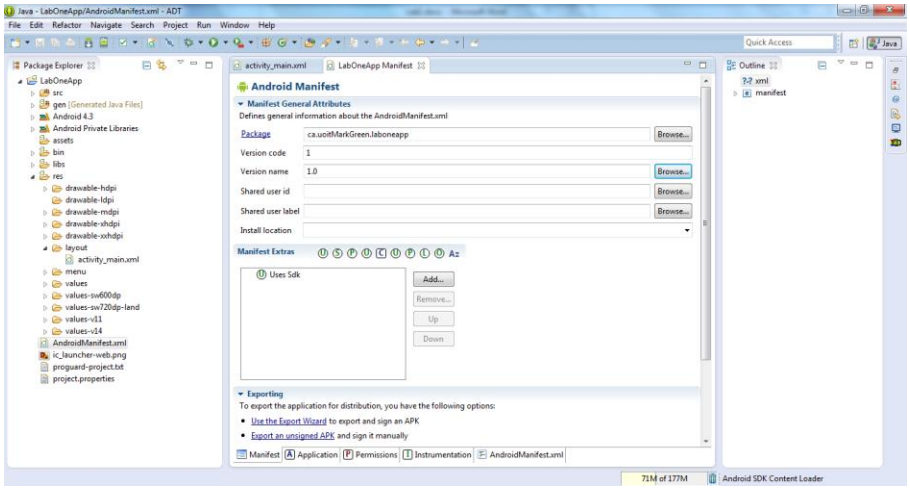

Figure 11 First tab of the manifest editor

Android does not re-install an app that has the same version code as an app with the same name that is currently on the device. If you are debugging using the emulator this isn't an issue, but it can be a problem on a real device. If you fix a bug in your application and then go to run it on the device, you will not be running the new version of the application, but the old one that is already installed on the device (this can be a very hard problem to find). There are two solutions to this problem. One is to remove the old version of the application before running the new version. This can be done through the settings app. The other is to change the version code before you run the application again.

The Java source code for our application is in the src folder. The complete code for the application is shown in figure 12. Since our application doesn't do very much we don't have very much source code. The only thing our application needs to do is construct the screen layout. The main class in our application, MainActivity, is a sub class of Activity and only overrides two methods.

```
package ca.uoitMarkGreen.laboneapp;

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

public class MainActivity extends Activity {

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


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

Figure 12 Application source code

The onCreate() method is called when the activity is created. It is responsible for initializing the application and drawing its initial screen. The setContentView() procedure converts our activity_main.xml file into the widgets and screen layout for our application. The parameter to this procedure is an integer identifier for the XML file, but where does this identifier come from? The parameter is R.layout.activity_main, so it must be part of the R class, but where does this class come from? It is automatically generated for us by the Android plugin for Eclipse based on the information in our layout files. If you are interested you can find the R.java file in the gen folder. It doesn't make for very interesting reading.

In Android development you will find that a number of things are automatically tied together for you by the SDK and the information that you need is quite often spread over multiple configuration files. This is typical of smartphone development.

Second Android Application

Our second Android application is slightly more complication and has some user interaction. This application is shown in figure 13. In this application you can enter our name, press the OK

button and a custom greeting is produced. The screen layout isn't very good, but that's something we can deal with later.

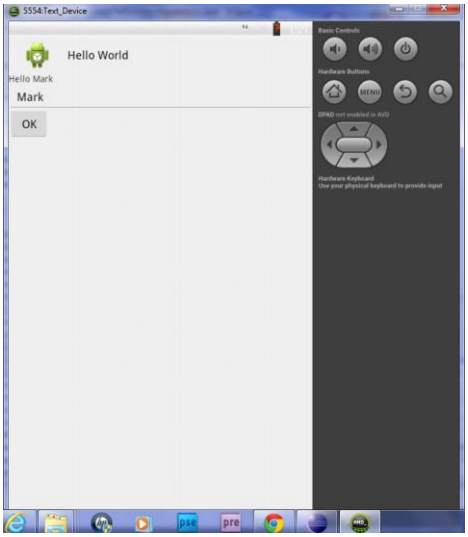

Figure 13 The second application

Start a new project for the application in the same way that we set up the project for the previous application. I called my application Hello World, but you can choose a different one if you like. When Eclipse starts go to activity_main.xml (this should already be displayed by Eclipse). By default Eclipse gives us a relative layout, but we want to use a linear layout instead. The easiest way to change the layout is to go to the XML tab and delete everything in it, so we can start over with a blank layout. In the left pane, click layouts and then drag a LinearLayout (Horizontal) to the middle pane. Now go to Form Widgets and drag a TextView to the middle pane, go to Text Fields and drag a Plain Text widget and place it below the TextView and finally go back to Form Widgets and drag a Button widget to the middle pane. This completes the basic layout of the UI, now we need to customize the widgets.

Start with the TextView, the first widget in our user interface. Click on it in the hierarchy view in the top right pane and its properties will be displayed in the lower right pane. We need to change the text that it's displaying. We could enter a text string for the Text property, but instead we are going to use a resource. For the Text value type @string/HelloWorld. This is a reference to a string resource called HelloWorld. Now we need to define this resource. In the package explorer on the far left go to the values folder in the res folder. There you will see a strings.xml file. Double click on this file and you will get a dialogue similar to the one shown in figure 14 (I've already defined the string resources, so the bottom two entries won't appear at this point). Click the add button, and then double click on the string icon and you will get the screen shown in figure 15. In this dialogue you can enter the name of the string resource, in this case HelloWorld and then whatever text string that you want displayed for this resource. The new string will now appear in the list of resources (it is initially displayed as String; just click it to get the updated value). Make sure to save the strings.xml file when you are finished.

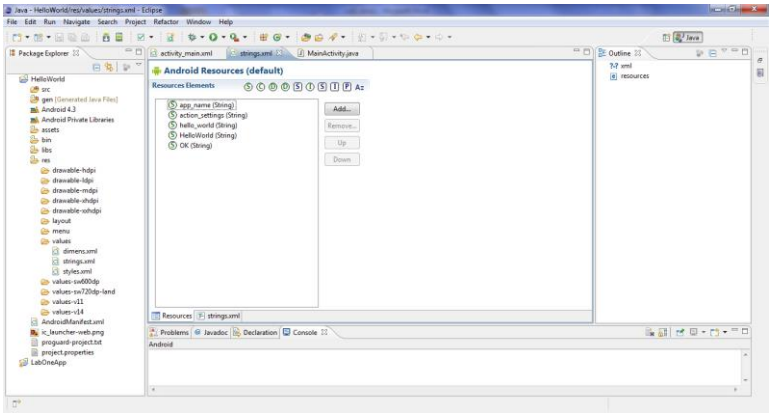

Figure 14 strings.xml file

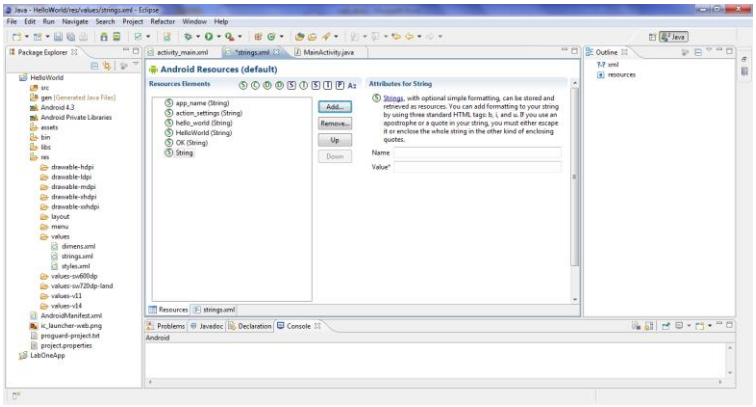

Figure 15 Editing a string resource

Now go to the next widget in the UI, editText1. We could leave this widget the way it is, but we get a warning message from Eclipse if we do this. Eclipse wants us to fill in the InputType Property. Pressing the button to the left of the property brings up a dialogue where we can select an input type; select textCapWords and then press the OK button. The button widget is the last widget we need to customize. We want to change the text displayed on the button to OK. Create a resource for this in the same way as the Text View widget. We want something to happen when the button is pressed; we want a method in our application to be called. We can set this in the UI editor, but the property that we need isn't displayed by default. We need to show the advanced properties for this widget and we can do this by pressing the button highlighted in figure 16. Once we have done this we can scroll down the list of properties until we get to the On Click property (it is close to the end of the list). The value of this property is the method in our activity that will be called when the button is pressed. Enter setName as the value for this property. This completes the design of the user interface and we are ready to go onto the program code.

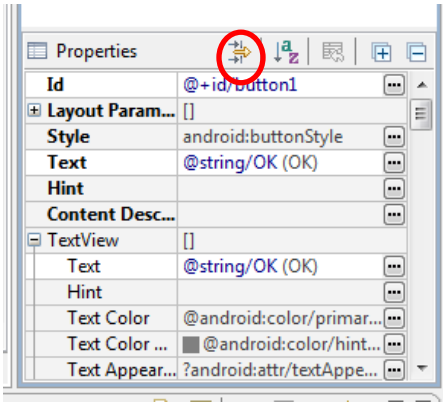

Figure 16 Button to show advanced properties

The program code for our second application is relatively simple; we just need to add one method to handle the button press. This code is shown in figure 17. When you enter this function you will get many errors. Many of them come from missing import statements, to solve this problem press cntl-shift-O and Eclipse will add the import statements for you. If there are still errors make sure that you have saved activity_main.xml and strings.xml, they should disappear when you run the application.

```
public void setName(View view) {
    EditText nameText = (EditText) findViewById(R.id.editText1);
    TextView label = (TextView) findViewById(R.id.textView1);
    String name = nameText.getText().toString();
    label.setText("Hello "+name);
}
```

Figure 17 The setName method

The first two statements in the setName method retrieve references to the EditText and TextView widgets in our UI layout. Both of these widgets have an Id property, which we can use in our program to retrieve a reference to them. The third statement retrieves the text string that the user entered and the fourth statement constructs the message to the user and sets it as the TextView's text.

This lab is relatively long, but you have learned most of what you need to know to build simple single screen Android applications.

Laboratory

Your task for this laboratory is to add a clear button to the second Android application. When this button is pressed the greeting to the user is cleared. Show your application and program code to the TA before the end of the laboratory, or zip the project and email it to the TA.