

CS 188 Android Introduction and Walkthrough Activities (multiple screens) and Buttons

Part 1. Activities and Android basics

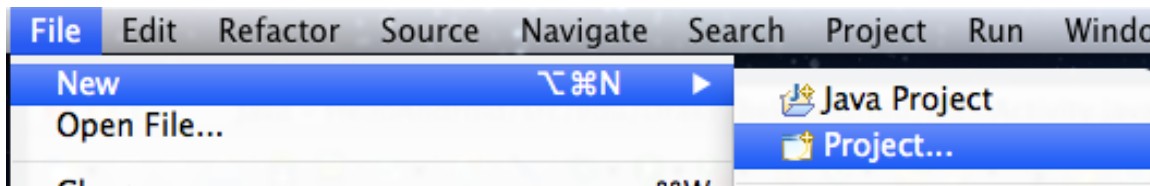
In Android, an **activity** is used to represent a single screen. Most applications have multiple activities to represent different screens.

1. If you haven't already, read the documentation on activities.

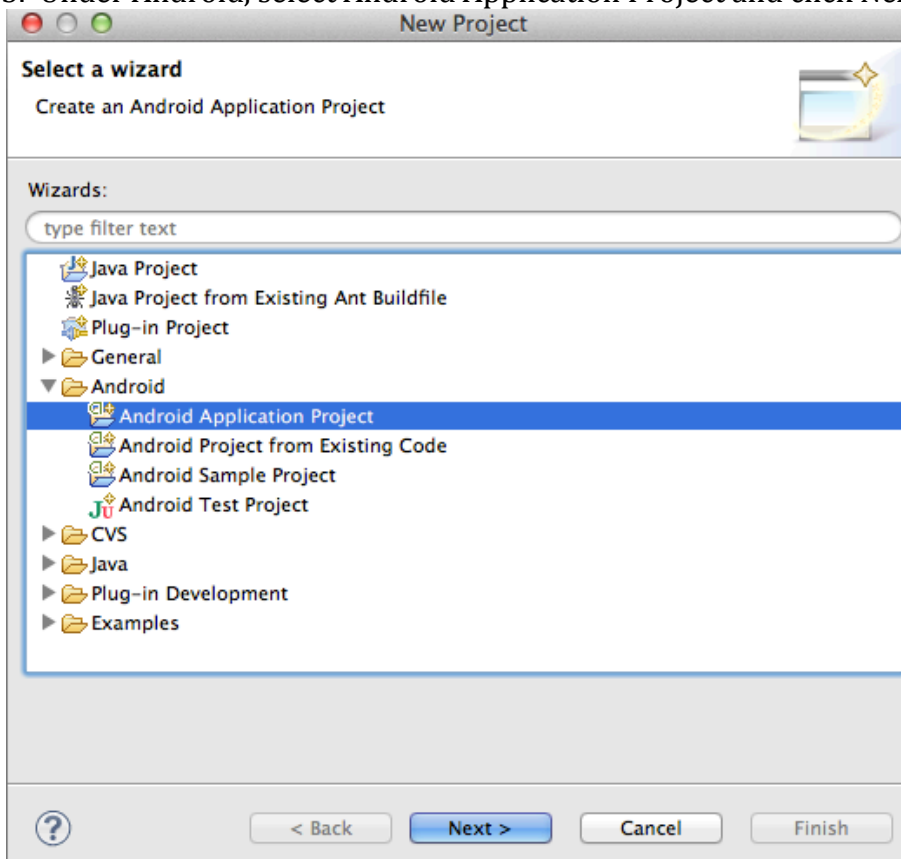
<http://developer.android.com/guide/components/activities.html>

Read the first four paragraphs very closely. Skim the rest of the document for now. It would be a good idea to return to this documentation for further in-depth reading.

2. In Eclipse, File > New > Project



3. Under Android, select Android Application Project and click Next



4. Fill in the following

New Android Application
Creates a new Android Application

Application Name:

Project Name:

Package Name:

Minimum Required SDK:

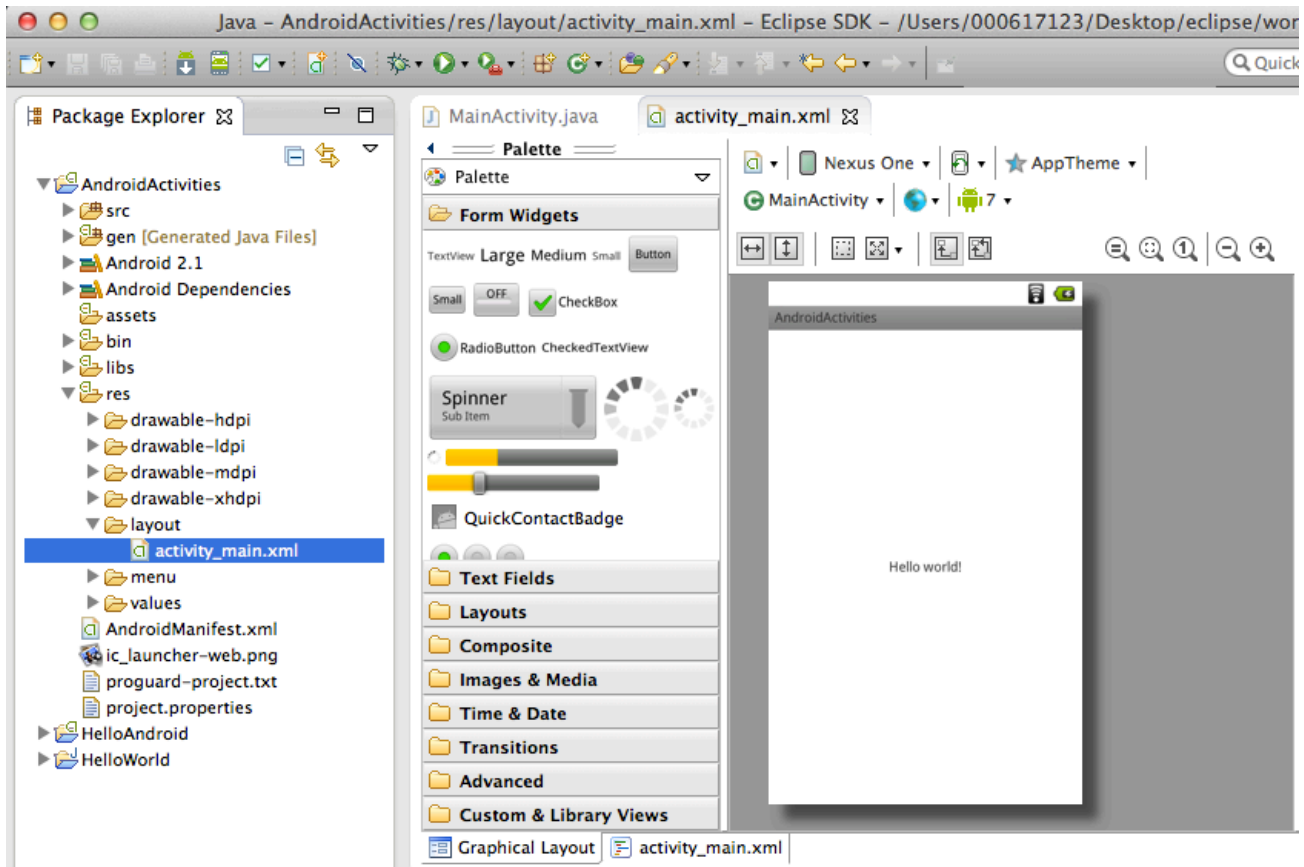
Target SDK:

Compile With:

Theme:

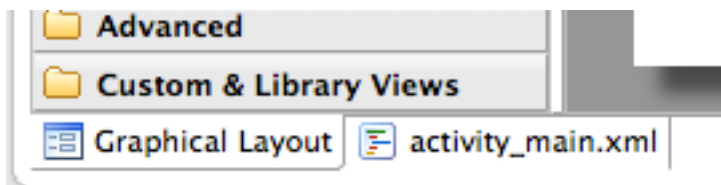
The package name must be a unique identifier for your application. It is typically not shown to users, but it *must* stay the same for the lifetime of your application; it is how multiple versions of the same application are considered the "same app". This is typically the reverse domain name of your organization plus one or more application identifiers, and it must be a valid Java package name.

4. Continue to click Next at the forthcoming dialog boxes, selecting the default values until you click Finish. This should create an app with a workspace that looks like this:



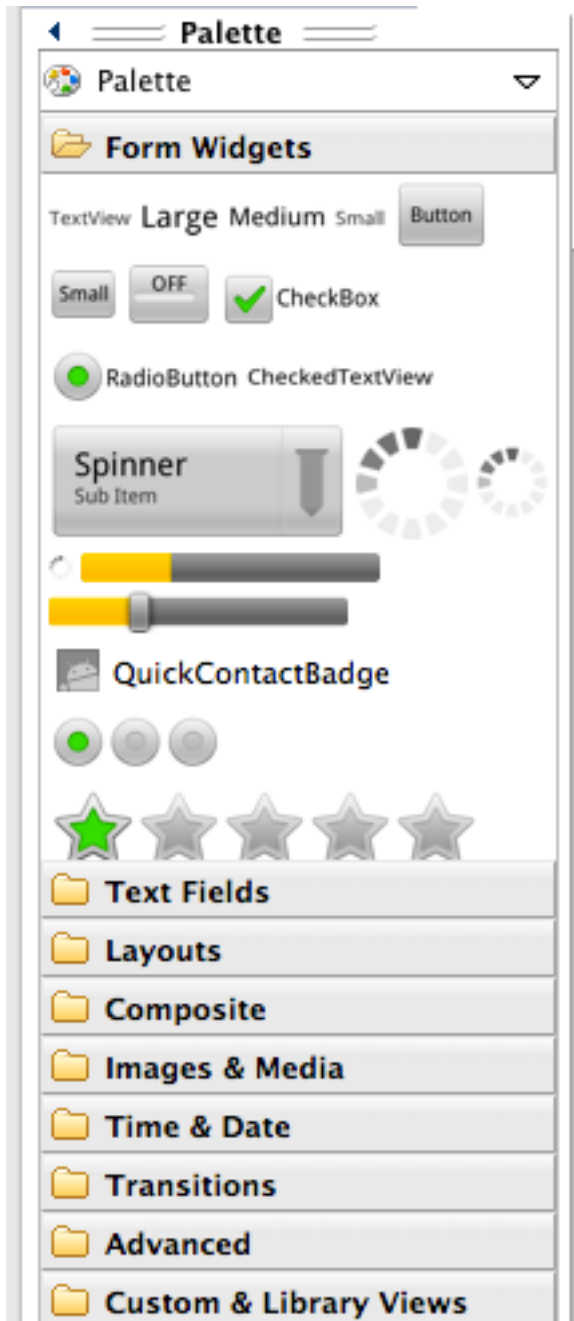
Notice the directory structure on the left hand side. Within the res/ (for “resources”) folder are application resources, such as raw asset files, colors, drawables, media or other files in the package, plus important device configuration details (orientation, input types, etc.) that affect how the application may behave.

Also Notice that res/layout folder contains a file called “activity_main.xml”. Eclipse is showing you a graphical view of this .xml file. Notice the tab on the lower left that allows you to toggle between this graphical layout and the actual xml text file.



You can work with either. It’s been my experience that you’ll have to be comfortable handling both the graphical layout and the .xml text view, as each has their advantages.

5. With activity_main.xml selected, and the graphical Layout selected, notice the “Palette” of GUI (graphical user interface) widgets that are available.

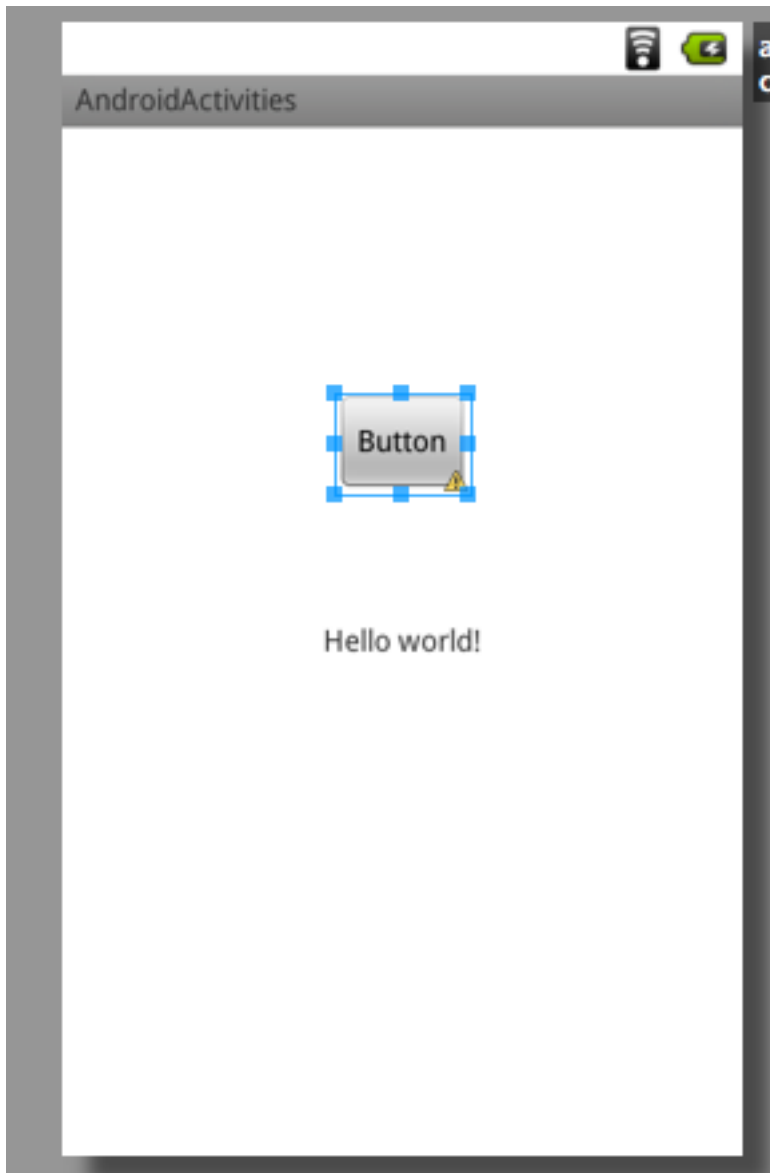


Take a few moments to look around at the various components.

6. Find the Button on the Palette.



7. Click and drag it onto your app screen so it looks something like this:



8. Now, switch to the activity_main.xml text view to see what has happened to the text.



9. You should see something like this added to your .xml file. Don't worry if your numbers are different, that is just specifying the location to put the button:

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="59dp"
    android:text="Button" />
```

10. Notice that the ID is “@+id/button1”. The rest of the fields are specifying to Android what size and where to place the button.

Also, the warning around the String “Button” can be disregarded at this time. It occurs because Android prefers all of your Strings to be placed in a “strings.xml” file (within the values/ directory) and then have all of the Strings in your GUI reference this file. One reason this is a good idea is that if you want your app’s text to change languages based on the preferred language set by the user, all of your Strings for the GUI would reference the same file – making it easy to adjust. For now, we will politely ignore this warning.

11. Change the “Button” text to say “Press Me” or something equally as clever.

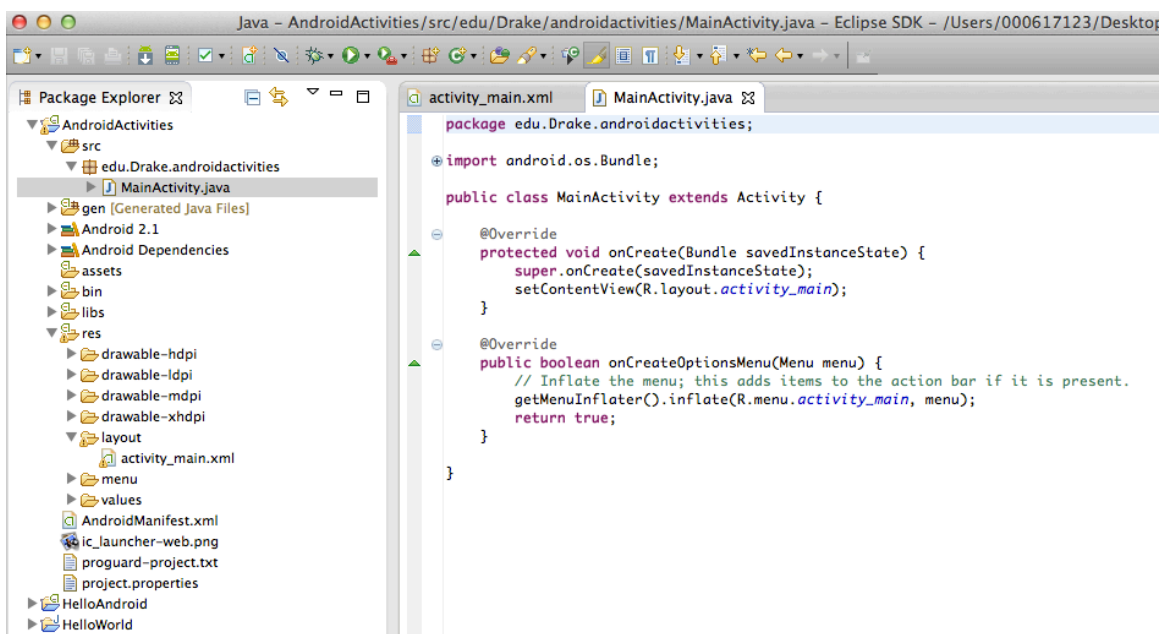
```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="59dp"
    android:text="Press Me" />
```



12. Go back and select the Graphical Layout.
The button text should now say “Press me”

13. **Make sure you save your file!!!** Command -> S or File > Save. Not saving your .xml files has been one of my biggest causes for frustration when working with Android – as it appeared my latest fixes didn’t take. Always save your files.

14. Now, let’s look at the source code. Go to the src/ directory within the edu.Drake.androidactivities package and select MainActivity.java

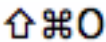


15. First, let's create a button object to correspond to the button we placed in the GUI as a global variable within this class as follows.

```
public class MainActivity extends Activity {  
  
    Button button;
```

Note that Eclipse is underlining the Button Class name. Why? We haven't yet imported the appropriate Android library for the Button class.

HERE IS AN AWESOME THING ABOUT ECLIPSE

On a mac: if you press  (shift+command+the letter O)

On windows: press Ctrl+Shift+O (control+shift+the letter O)

IT WILL AUTOMATICALLY ADD THE APPROPRIATE IMPORT LINES OF CODE.

This functionality is also available in Eclipse under Source > Organize Imports

16. Do the command in step 15 which will result in adding

```
import android.widget.Button;
```

to your code. And remove the syntax error from the Button declaration.

Part 2. Buttons

17. Read the documentation on Android Buttons.

<http://developer.android.com/guide/topics/ui/controls/button.html>

Read the top part of the page closely. Read the second half on "styling your button" briefly.

18. When using an Android activity, the onCreate method is similar to the main method – it's where everything gets started. To connect the button we declared in code with the button we created on the GUI we'll need to explicitly reference the ID that was identified in step 10 above ("@id/button1").

```
button = (Button) findViewById(R.id.button1);
```

19. Buttons in Android will respond to a special method called "onClick". Thus, we need to establish a listener for the button to call an "onClick" method once the button is pressed. This is typically all done in-line in one block of code. That is, we are going to declare the onClick method within the setting of the listener for the button.

Make your code look like the following:

```

Button button;

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

    button = (Button) findViewById(R.id.button1);
    button.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            //code put here will be executed when button is pressed
        }
    });
}
}

```

Why all of the syntax errors? We are missing import statements. Do the trick introduced in step 15 to add the appropriate import files. If given the choice, choose the `android.view.View.OnClickListener` option.

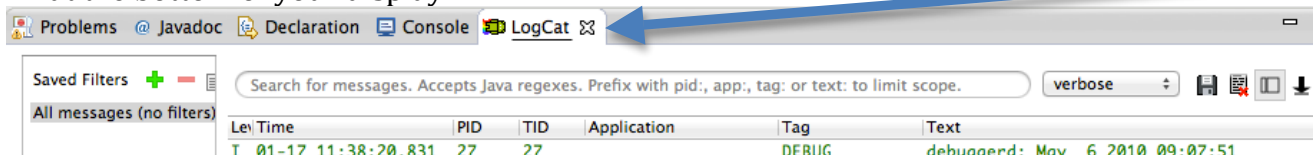
Part 3. Log Statements

Look at the documentation on Log

<http://developer.android.com/reference/android/util/Log.html>

Don't feel the need to read this in depth, just scan it for now.

20. Android does not respond to `System.out.println` statements to the Console. Instead, you can use the LogCat tool to get output to help you develop your program. LogCat is likely already at the bottom of your display



In the event that you accidentally close it, or cannot find it, goto Window > show View > Other > Android > Log Cat

There are several different “levels” at which you can log a message. They include: assert, debug, error, info, verbose, warn. For example, you could configure LogCat to just show you all of the verbose messages.

21. To print to the LogCat, you pass in two arguments, the first is a String called a tag. A good convention is to declare a TAG constant in your class. For example, enter the following


```
public class MainActivity extends Activity {


    private static final String TAG = "MainActivity";
```

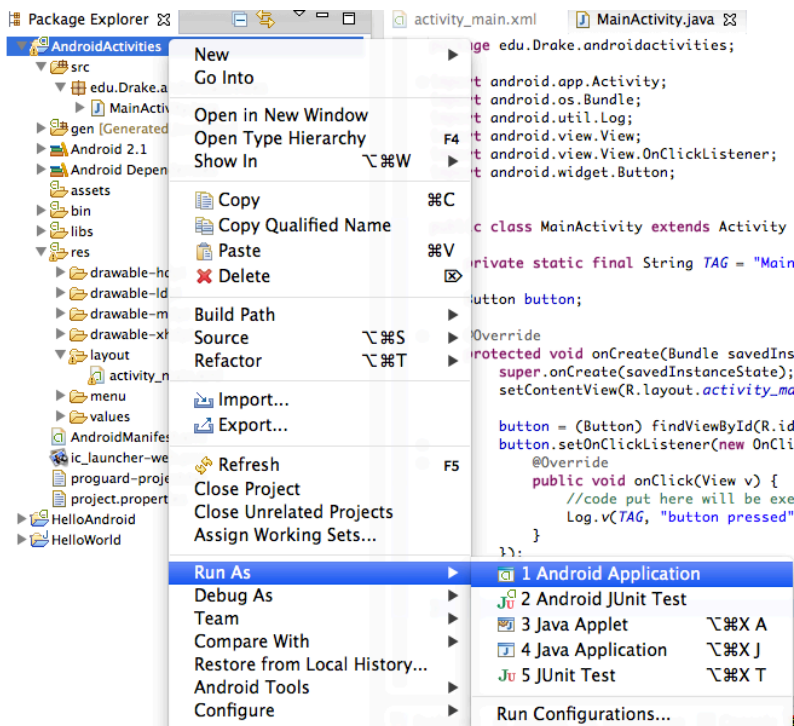
Then you can print out to LogCat using the verbose *verbose* level (using Log.v – v for verbose) when the button is pressed by doing the following:

```
button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //code put here will be executed when button is pressed
        Log.v(TAG, "button pressed");
    }
});
```

Again, do the step 15 trick to import the correct libraries for Log. (use android.util.Log).

22. Run the app.

Sometimes, pressing the  button may launch the wrong app – even though it is not selected in Eclipse. To make sure the appropriate app is run, you can right-click on the package explorer and select Run As> Android Application



If LogCat is unresponsive, you can close the LogCat Window and open it up again Window > show View > Other > Android > Log Cat

The final code should look like this:

```
package edu.Drake.androidactivities;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {

    private static final String TAG = "MainActivity";

    Button button;

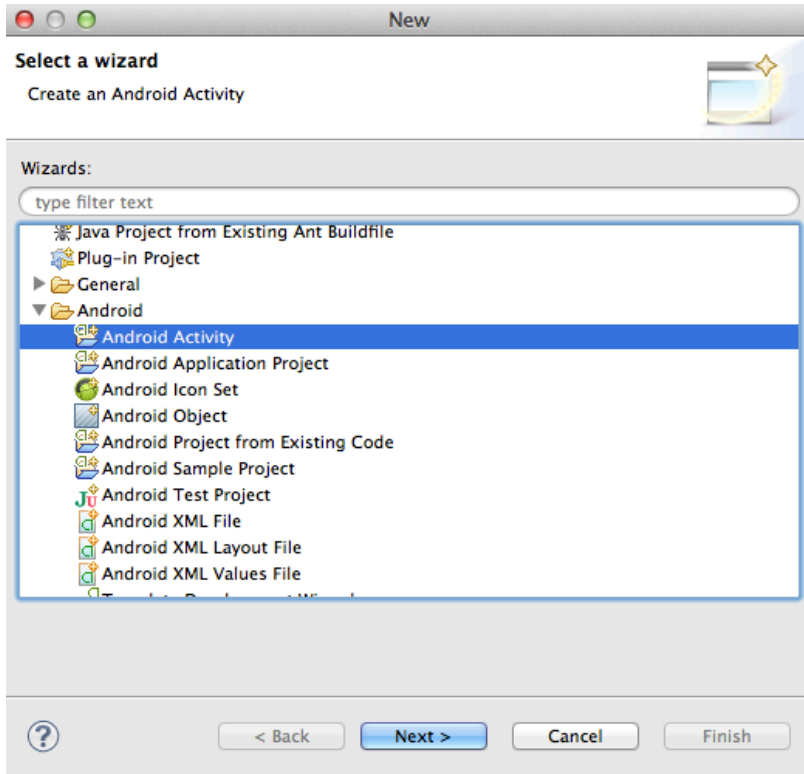
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button = (Button) findViewById(R.id.button1);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                //code put here will be executed when button is pressed
                Log.v(TAG, "button pressed");
            }
        });
    }
}
```

*Challenge: make an app that will output to the logCat the number of times the button was pressed.
Hint: use a variable.*

Part 4. An indepth look at activities

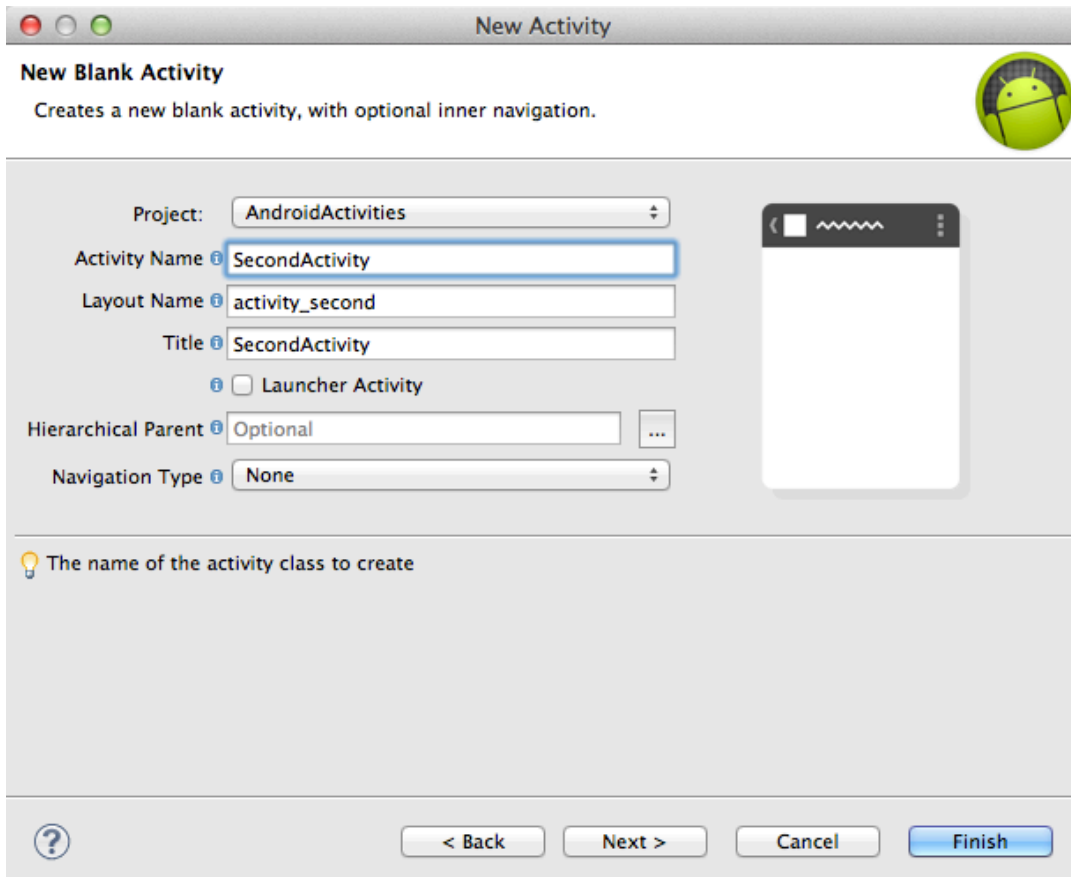
23. In order to have another screen, we must first define a new activity (including a .java src file and an .xml layout file). In Eclipse, select File > New > Other. Then Android > Android Activity



24. Create a blank activity and click next



25. Call it SecondActivity and click Finish



26. Notice that this creates a new file in the layout/ folder (activity_second.xml) as well as a new .java file in the src/ folder (SecondActivity.java).

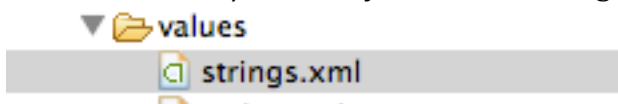
27. Select activity_second.xml in the layout/ folder and switch to the text .xml view. Let's change the text to read "Second Activity" by referencing a new String (which we will create next) in the values/ folder. Change the line

```
android:text="@string/hello_world" />
```

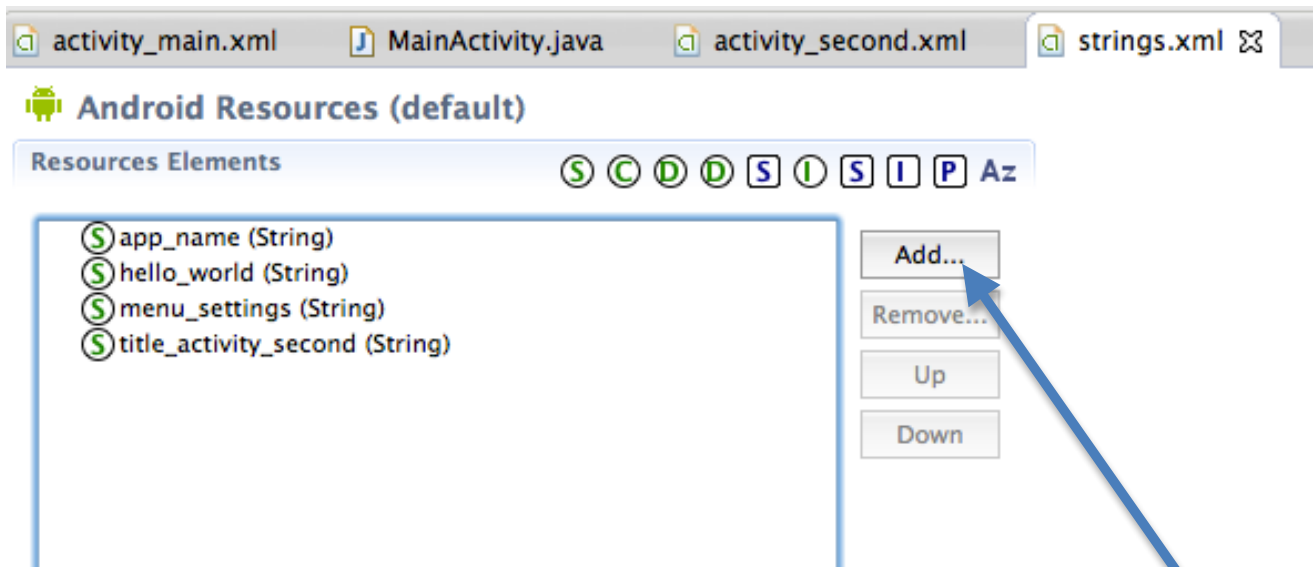
to

```
android:text="@string/second_screen" />
```

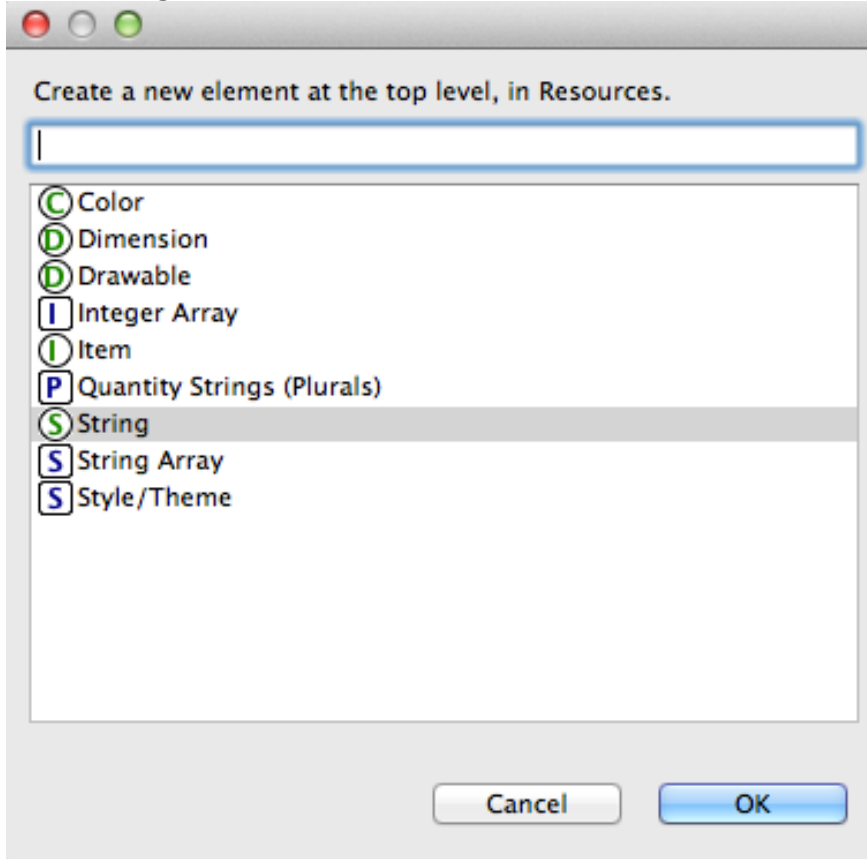
28. In the values/ directory, select the strings.xml file



and click the "Add" button



Select String



Add the values to your newly created String to look like the following:

Resources Elements
S C D D S I S I P Az

S app_name (String)
S hello_world (String)
S menu_settings (String)
S title_activity_second (String)
S second_screen (String)

Add...
Remove...
Up
Down

Attributes for second_screen (String)

@Strings@, with optional simple formatting, can be stored and retrieved as resources. You can add formatting to your string by using three standard HTML tags: b, i, and u. If you use an apostrophe or a quote in your string, you must either escape it or enclose the whole string in the other kind of enclosing quotes.

Name	second_screen
Value*	Second Screen

Intents are messages which allow Android components to request functionality from other components of the Android system. For example an *Activity* can send an *Intent* to the Android system which starts another *Activity*.

29. In the MainActivity.java file, we are going to use the button press to create an intent to cause the second screen appear. It's remarkably simple: Alter your code to the following:

```
button = (Button) findViewById(R.id.button1);
button.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //code put here will be executed when button is pressed
        Log.v(TAG, "button pressed");
        Intent intent = new Intent(v.getContext(), SecondActivity.class);
        startActivity(intent);
    }
});
```

Again use the Step 15 trick to add the appropriate import statements and make the syntax errors go away.

30. Run the app and notice the second screen should appear when the button is pressed.

Note, however, that the back buttons on the device don't work in the simulator. That is, once you've reached the second screen, there is no way to get back to the first screen. Activities are placed on a "stack." That is, if you'd like to return to a previous activity, you can pop the current activity off of the stack by

Activities are placed on a "stack". To pop an activity off of the stack you can use the method `finish()`.

Challenge: add a button to the Second Activity. When the button is pressed, the activity should be popped off of the stack and execution returned to the first activity.

The following code may be useful:

```
button.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        finish();  
    }  
});
```

31. Now go back and re-read the activities documentation (from step 1). Read the whole page to help you understand how Android programming works.

<http://developer.android.com/guide/components/activities.html>