

# ANDROID

Intro To Android App Creation  
Lesson 2





JESSE SCOTT

[jscott@langara.bc.ca](mailto:jscott@langara.bc.ca)





# FINALIZING OUR DEV ENVIRONMENT



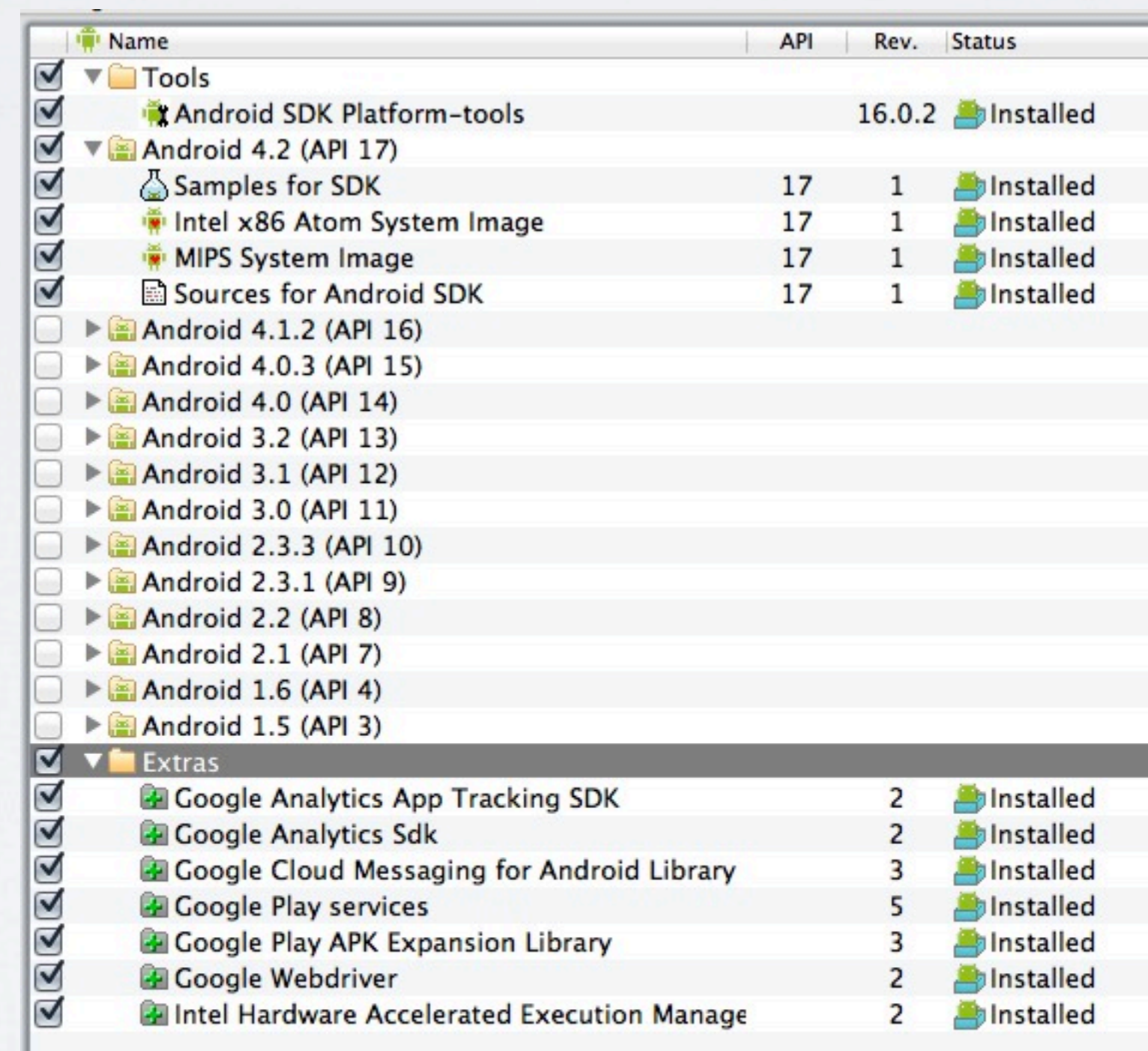


# SDK

Android SDK can be found at <http://developer.android.com/sdk/index.html>

Make sure you installed:

- USB Driver (if you are on Windows or Linux)
- Latest API (22)
- Platform Tools



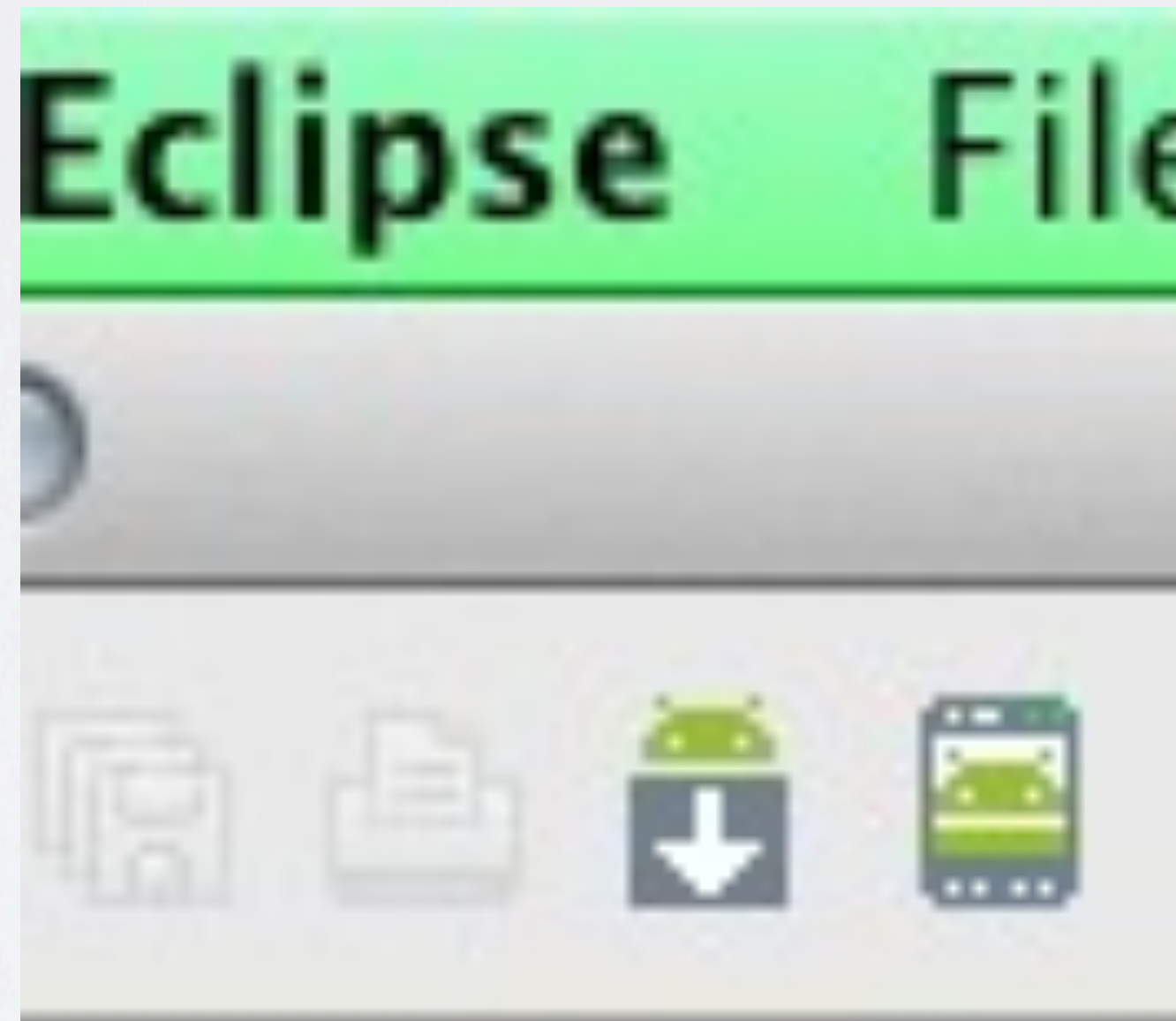
Name	API	Rev.	Status
Tools			
Android SDK Platform-tools		16.0.2	Installed
Android 4.2 (API 17)			
Samples for SDK	17	1	Installed
Intel x86 Atom System Image	17	1	Installed
MIPS System Image	17	1	Installed
Sources for Android SDK	17	1	Installed
Android 4.1.2 (API 16)			
Android 4.0.3 (API 15)			
Android 4.0 (API 14)			
Android 3.2 (API 13)			
Android 3.1 (API 12)			
Android 3.0 (API 11)			
Android 2.3.3 (API 10)			
Android 2.3.1 (API 9)			
Android 2.2 (API 8)			
Android 2.1 (API 7)			
Android 1.6 (API 4)			
Android 1.5 (API 3)			
Extras			
Google Analytics App Tracking SDK	2		Installed
Google Analytics Sdk	2		Installed
Google Cloud Messaging for Android Library	3		Installed
Google Play services	5		Installed
Google Play APK Expansion Library	3		Installed
Google Webdriver	2		Installed
Intel Hardware Accelerated Execution Manage	2		Installed



# ECLIPSE

Eclipse IDE can be found at <http://eclipse.org/mobile/>

You can make sure this works properly by launching the SDK / AVD managers  
- you should see them in your menu bar (see image)



# JDK

Java Development Kit can be found at

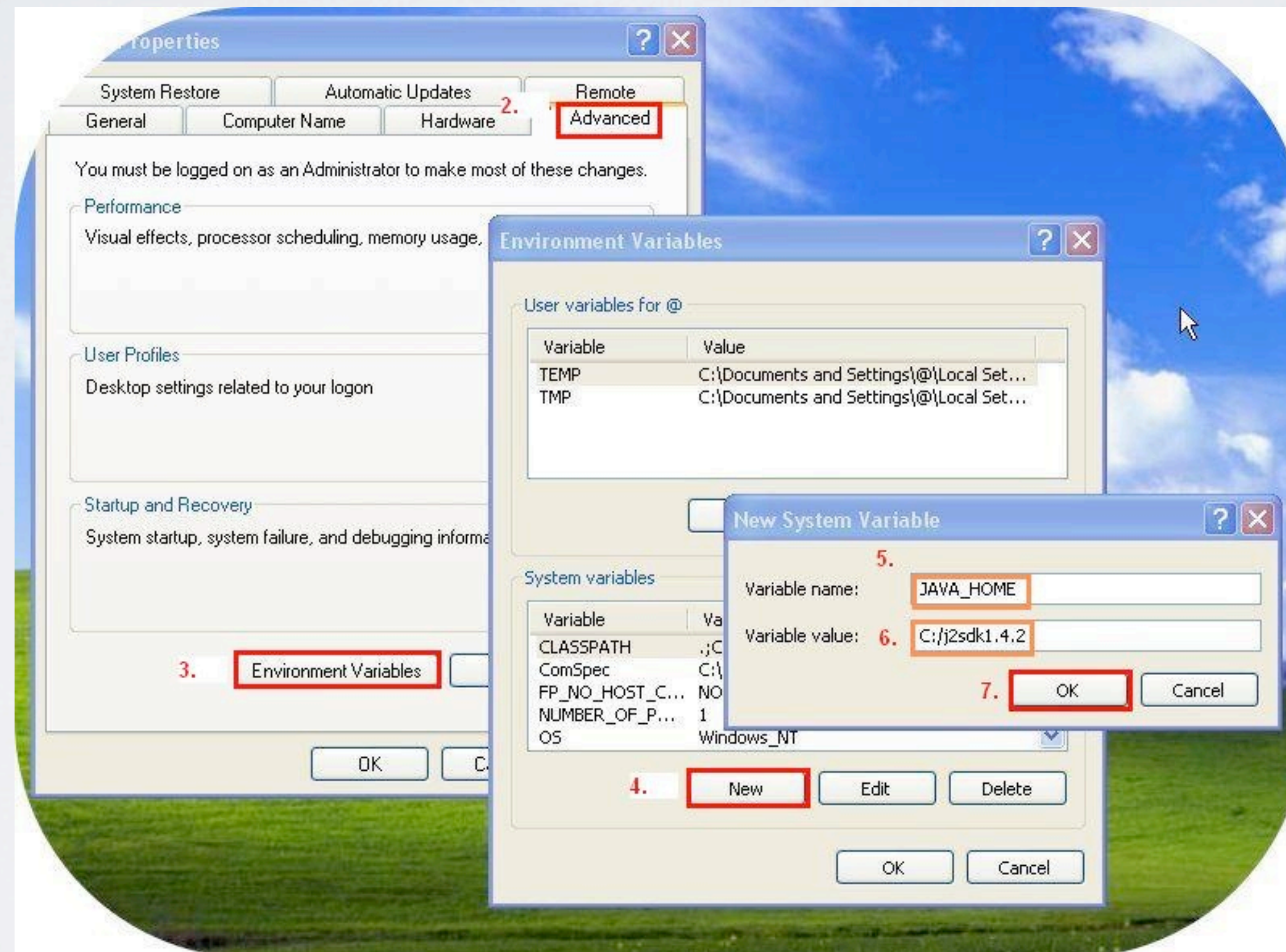
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

You can test the install by typing “java -version” from a Command Line program



# JAVA\_HOME

You may also need to create the JAVA\_HOME Environment Variable



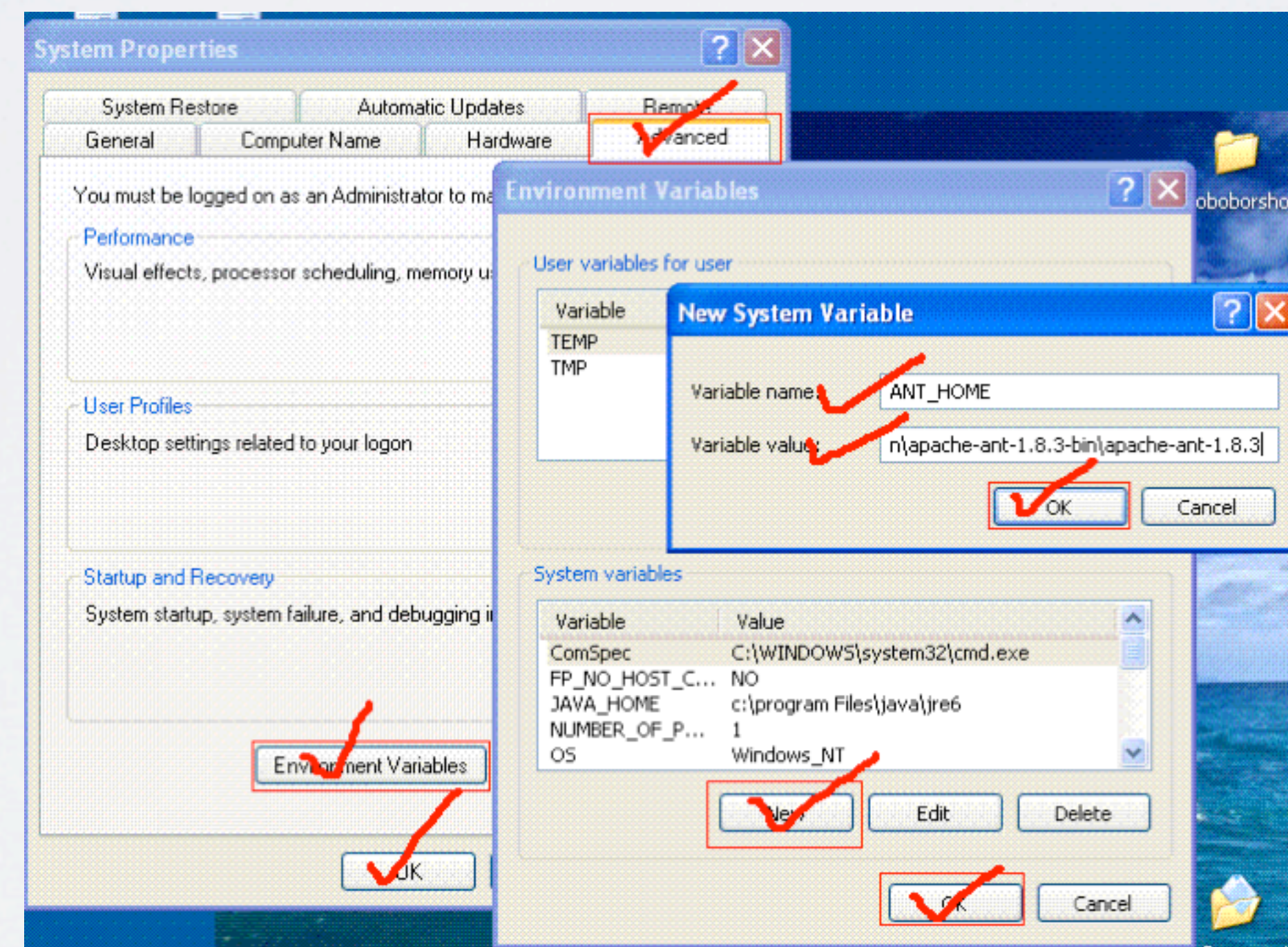


# ANT

ANT may need to be installed as well, and can be found at <http://ant.apache.org/>

You can test the install by typing “ant” from a Command Line program

You may also need to create the ANT\_HOME Environment Variable





# SDK PATH

If you are running your SDK from a different drive (partitioned or networked), then you may need to setup a `ANDROID_SDK_HOME` Environment Variable.

Follow the steps above, with the path to your SDK...

Within Eclipse, you need to edit the *config.ini* file under `/eclipse/configuration`



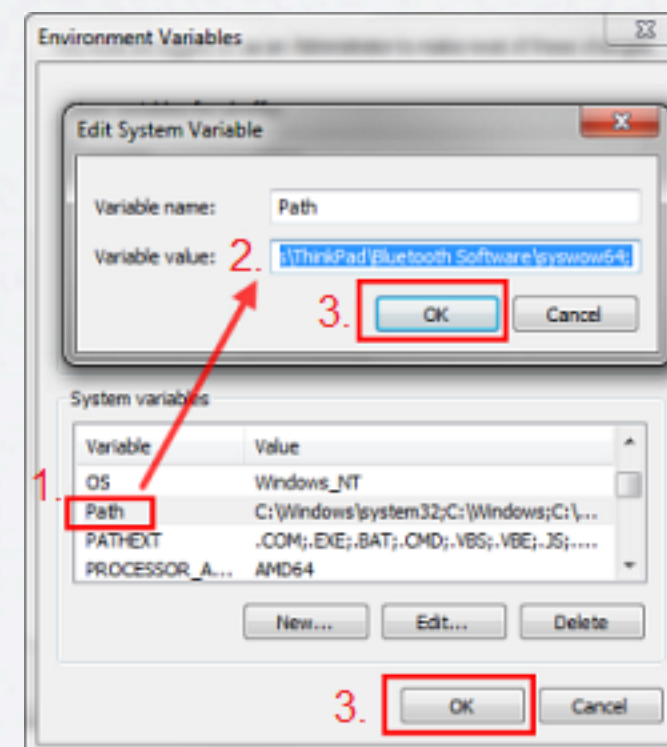
# PATH

All of the above edits need to be added to your PATH Environment Variable.

The PATH variable will already exist, so find it, and select Edit.

Navigate to the very end of it, and make sure to separate the additions with a semicolon... then the syntax is *%variable%/bin* ... so:

xxx;%ANT\_HOME%/bin;%JAVA\_HOME%/bin;%ANDROID\_SDK\_HOME%/bin









# ECLIPSE





# WORKSPACE

Eclipse uses the idea of a 'workspace' - essentially a dedicated folder for your projects. Set this up once, and you can set Eclipse to not ask you again.

You *can* work with multiple workspaces as well...



# NEW PROJECTS

To create a new Android Project ...

- > File > New > Project

- > Android > Android Application Project

Application Name : (what our app will be called, visible to the user)

Project Name: (what our project is called, for eclipse)

Package Name: (reverse domain name structure, i.e. *com.example.myApp* )



# NEW PROJECTS...

Minimum Required SDK : the lowest version of Android needed to run the app  
Target SDK : the highest version of Android that will run your app  
Compile With : should be the same as your Target SDK  
Theme : the base colour theme



# DIRECTORIES

MYPROJECTNAME : this is your main project directory

SRC : the directory where our package (our app name) and our Classes live

- if you create new Classes, they should be in here

GEN : generated Java files - DONT EDIT THESE !

- if you get an error about “R.java” , you edited it !

ANDROID X.X : this is the library for the Target SDK of Android

- you can change this by ctrl-clicking your project & editing the Properties

ANDROID DEPENDENCIES : this should be set by the above library...

ASSETS : this is where we put our assets that are NOT dependent on screen size

- such as sounds & video, text files, etc

RES : this is where we put our assets that ARE dependent on screen size

- skins, buttons, logos, etc



# DIRECTORIES...

## ANDROIDMANIFEST.XML :

this is the xml file that controls the behaviour of your application, such as your package name, the SDK version, your icon path, theme, your orientation, your permissions, and the name of your activities...

it is an important file to review and edit at certain points within your development.







# HELLO WORLD



# LAUNCHING

Option #1 : ctrl-click on project name > Run As > Android Application

Option #2 : Run Menu > Run

Option #3 : Run Menu > Run As > Android Application

Option #4 : Run Button

It *should* give you the option of launching a particular emulator



# EMULATOR

The Emulator takes a while to start up, so be patient!  
(and don't quit it - use the back button to 'quit' your app)

However, if you want to use a *different* emulator, quit the open one and either:

- a) run your app with the desired emulator, or
- b) launch your desired emulator first, then run your app







# TOUCH EVENT



# TEXT VIEW

In our activity\_main.xml file, let's drag and drop two text view's...

#1 is going to be an empty one with no text, and filling the whole screen (to receive a touch callback)

<TextView

```
    android:id="@+id/touchView"  
    android:background="#f00"  
    android:layout_weight="1"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"
```

/>



# TEXT VIEW...

#2 is going to have default text and sit at the bottom of our screen  
(to print our received touch events)

<TextView

```
    android:id="@+id/textView"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Hello World, TestActivity"
```

/>



# JAVA CLASS - IMPORTS

In our MainActivity.java file, let's add our imports...

```
import android.view.MotionEvent; // to get our motion event callback
import android.view.View; // to access the view (screen)
import android.widget.TextView; // to set our text programmatically
```



# JAVA CLASS - VIEWS

Then, let's add our TextView and TouchView

```
// this is the view on which you will set your text
final TextView textView = (TextView)findViewById(R.id.textView);
// this is the view on which you will listen for touch events
final View touchView = findViewById(R.id.touchView);
```

Note that we are 'unwrapping' these assets from our R.java file...



# JAVA CLASS - TOUCH LISTENER

We want to add a listener to our `TouchView`:

```
touchView.setOnTouchListener(new View.OnTouchListener() { ...
```

Note that *TouchView* `setOnTouchListener()` method *takes* a touch listener from our *View*



# JAVA CLASS - ONTOUCH

Inside our `setOnTouchListener()` method, we call an `onTouch()` method:

```
@Override  
public boolean onTouch(View v, MotionEvent event) {  
    return false;  
}
```

Note that we override the `onTouch` method, and we have to pass it our `View` (screen) and the `MotionEvent` (which we've imported already)



# JAVA CLASS - SET TEXT

Inside our onTouch() method we set the text of our textView object:

```
@Override
    public boolean onTouch(View v, MotionEvent event) {
        textView.setText("Touch coordinates :
        OUR X IS: " + String.valueOf(event.getX()) +
        " & OUR Y IS: " + String.valueOf(event.getY())
        );
        return false;
    }
```











# ACTIVITY

In our AndroidManifest.xml file, let's create a new activity

```
<activity
    android:name=".SecondActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.SECOND" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

Make sure this is within the <application> tag !

Also, note that the category is DEFAULT and not LAUNCHER anymore...



# MENU

Let's create a second menu xml file

/menu > ctrl-click > New > Other > Android > Android XML File

- resource type "MENU"
- file name "activity\_second"

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/menu_settings"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/menu_settings"/>
</menu>
```



# LAYOUT

Let's create a second layout xml file

/menu > ctrl-click > New > Other > Android > Android XML Layout File

- resource type "Relative Layout"
- file name "activity\_second"



# LAYOUT...

Let's add a text element

drag and drop a text label .. or just add it in the xml

```
<TextView
```

```
    android:id="@+id/textView1"
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
```

```
    android:gravity="center"
```

```
    android:text="I'm A New Screen!"
```

```
    android:textAppearance="?android:attr/textAppearanceLarge"
```

```
/>
```



# LAYOUT...

Let's add a button to our MainActivity layout...  
drag and drop it, or add the following

<Button

```
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:layout_centerVertical="true"  
    android:text="Click Me !"
```

/>



# MAIN ACTIVITY - IMPORTS

Let's add a button to our MainActivity Java File...

First, import the required API's

```
import android.content.Intent;  
import android.view.Menu;  
import android.view.View;  
import android.widget.Button;
```



# MAIN ACTIVITY - BUTTON OBJECT

Then, add the button code (inside our onCreate method)

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

We declare an object on the Button class (and cast it as a Button object), then inflate it from the R.java file (that stores our variables)



# MAIN ACTIVITY - CLICK LISTENER

Then we register the button object to a `onClick` listener:

```
button.setOnClickListener(new View.OnClickListener() {
```

Like the last project, it takes the listener of *our entire view* as an argument...  
... and this `View.OnClickListener` method calls the `onClick` method:

```
@Override  
public void onClick(View v) {  
    startActivity(new Intent("android.intent.action.SECOND"));  
}
```



# MAIN ACTIVITY - ON CREATE

Our entire onCreate() method should now look like this:

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

    Button button = (Button) findViewById(R.id.button1);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent("android.intent.action.SECOND"));
        }
    });
}
```



# SECOND ACTIVITY - CREATE

Now we need to make a Second Activity, which we do by making a new Java Class

- src > package > ctrl-click > New > Class > “SecondActivity”

Import our Bundle, Activity, and Menu Classes

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



# SECOND ACTIVITY - CREATE

Then make our class extend Activity, and add the onCreate and onCreateOptionsMenu methods...

```
public class SecondActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_second);  
    }  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        getMenuInflater().inflate(R.menu.activity_second, menu);  
        return true;  
    }  
}
```







FIN