**Midcities Programming Group Android Workshop by Jimmy Ruska**

Android Studio: Google’s latest IDE based on IntelliJ IDEA

Eclipse with ADT bundle: Eclipse open source IDE with Android Development Kit plugins

Requirements: Know some xml or html, know at least 1 other programming language at intermediate level.

**Make sure Android SDK has all dependencies**

In Eclipse or Android Studio there’s a button that looks like this somewhere in the toolbar: sdk-manager-studio.png

Make sure you have the following installed. Many are not installed by default. Android L or 20 is for watches.

Android SDK Manager:

Android SDK Tools 23

Android SDK Platform-tools 20

Android SDK build-tools 20

Android SDK build-tools 19.1

Android 4.4.2 (API 19)

SDK Platform

Intel x86 atom system image

Google APIs (x86 system image)

Extras

Android Support Repository

Android Support Library 20

(Optional: speeds up emulator) Intel x86 emulator accelerator

**Making a Working Emulator**

Look for an icon that looks like thistb-avd-manager.png. Then click “Create”. You don’t need a VM if you have a phone.

AVD Name: Nexus5

Device: Nexus S (4.0”, 480x800;hdpi)

Target: Google APIs…. API LVL 19

Keyboard: [X] Hardware keyboard present

Skin: Skin with dynamic hardware controls

Ram: 343

VM Heap:32

Internal Storage: 200

**Clean your project**

Cleaning can fix issues with upgraded versions and sometimes imported resources.

Android Studio: Build -> Clean Project

Eclipse: Project -> Clean...

**Quick Keys**

Eclipse: ctrl+shift+o = import everything needed. autocomplete = ctrl+space.

Android studio: Alt+Enter = import needed item (<http://stackoverflow.com/questions/16615038>). ctrl+space.

**Understanding Android Project Structure**

MyApp

|\_ app

|\_ build: This folder has code that’s generated at compile time. You do not need to modify anything here.

|\_ libs: You should put any additional java libraries your app uses inside this folder.

|\_ src

|\_ androidTest: Use a testing framework based on JUnit for tests at runtime.

|\_ main

|\_ java: This is where your main code is! Most programming will be here!

|\_ res

|\_ drawable\_\*: These are images. You can right click and add images to these folders.

|\_ layout: All the layouts and visual designs you make. They are XML but you can edit with a GUI.

|\_ menus: The layout of your menu when someone clicks the menu button

|\_ values: All the text in your application. Useful so you can make it multilingual eventually.

|\_ AndroidManifest.xml: Permissions, internet connectivity. Intents. Data about app.

If it were web programming the “java” folder would have your javascript and the “layout” folder would have your html. **void main()** equivalent would be the **onCreate** function specified as the main activity in the android manifest.

**What Happens when you Compile**

1. The XML files like in **layout** and **menus** folderare turned into java code then compiled and turned into a file called R.class, which you will often have to reference in your code.
2. Android Manifest is read and it looks for the activity to get the main activity. Eg **App -> Src -> main -> java -> MyActivity.java**
3. MyActivity.java is compiled. This is where you put all your application logic.
4. Either it prompts you to use an Emulator or it allows you to use a plugged in android phone. The emulator is slow and there’s no feedback on some things like vibration.

**Enable USB debugging on your phone**

1. On most devices running Android 3.2 or older, you can find the option under **Settings > Applications > Development**.
2. On Android 4.0 and newer, it's in **Settings > Developer options**.
3. **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**.

**Dissecting MyActivity.java: A Hello world with no fragments.**

// a namespace for all your classes and interfaces.

**package …;**

// all the imports of stuff you need from the android SDK

**import …;**

/\* **ActionBarActivity** is code google wrote to handle all the typical android events. It has all kinds of nifty

functions and information tied to it. We are creating our own class named MyActivity that takes all of the

functionality of the ActionBarActivity extends its functionality with our own functionality. \*/

**public class MyActivity extends ActionBarActivity {**

/\* When creating the activity do “...”. When the screen is rotated all the information is destroyed. For example

if I have a textbox that I click and type the word “foo” into, when I rotate the screen the text box will be blank

again. There are several things that can destroy the screen. savedInstanceState can be used to recover that

information. The @Override means that the function existed in ActionBarActivity but we’re overriding it. \*/

**@Override**

**protected void onCreate(Bundle savedInstanceState){ … }**

// this gets called when someone clicks the menu button on their android phone

// The options for menu items are specified in **res -> menu**

public boolean onCreateOptionsMenu(Menu menu){ … }

// This gets called when someone actually selects a menu item. You’ll notice it references R.id.action\_settings;

// Remember my.xml in **res -> menu** got compiled into java code as R.class. Here it is looking for an

// item element with attribute android:id=“@+id/action\_settings” which corresponds to a “Settings” menu item.

public boolean onOptionsItemsSelected(MenuItem item){ … }

}

That’s it for a hello world example!

**How do I print to console?**

Log.e(“Module Name”, “Error name”); is the typical way. It dumps errors to logcat. “e” stands for error. There’s multiple levels you can output. Log.v(...) is verbose, d is debug, w is warn, i is info.

**What’s the point of fragments?**

You have to handle onCreate, onDestroy, onPause, onResume in your app. People might get phone calls while using it, or a screen rotation may destroy the screen. Fragments net you create custom onPause, onCreate, onStart …, handlers for each view. This makes handling UI much more modular. You can also use them kind of like frames in html to have composite views.

**What is an intent?**

An intent is telling the operating system about an intent to do something. For example changing activities, accessing the webcam, accessing contacts or opening a web browser.

**What is an activity?**

If this were the internet the activity would be the different pages on your site. Activities are screens of your program you can switch between by using intents.

**What is a broadcast receiver?**

What you use to listen for different intents. If you want to make a web browser app you should listen to browser intents.

**What’s @+id/name vs @id/name**

+id/name is adding name as a new unique identifier. @id/name is referencing that identifier somewhere. You can reference anything, like @drawable/ic\_launcher for the ID of the main app icon, or things like @string/hello\_world. @string gives where to find that data once all the xml data is inflated.

**What’s inflator?**

It’s used to “inflate” the xml at runtime. That means the XML is scanned and turned into java objects representing UI components based on its data.

**What’s the difference between a service, asynctask, thread and handler.**

They are all for doing stuff in another thread and out of the UI thread. Services are for long running processes that can live as long as your app is running, thread is part of the java language for creating isolated processes, handler and async are build using threads and are more convenient. Use handler when you have multiple tasks to complete at once. Use async if you have a single task that you want to run and get back data from, outside the main UI thread.

**How would I make a 2d Game or animation in android?**

Research canvas + surfaceview. Views are typically UI components that are updated somewhat slowly. A surfaceview is updated as fast as it can possibly be updated. You will to make a new thread to update the surfaceview without locking the UI thread.

**What’s the UI thread vs separate thread?**

If you click a button that says download website and you do it from the UI thread, the UI thread would become frozen and unresponsive as it waits for the download to complete. It’s better to create a new thread that does it in the background. If you were a lumberjack and a client came in to order lumber and you ripoff your shirt and go choppin’, the next clients going to your store will not be able to buy lumber until you come back. It’s better if you tell bob the lumberjack, or steve the lumberjack to go chop lumber but leave yourself free to manage the store. A thread will create as many worker processes as you need.

**Things You Have to know about Java**

Example.java is what you have your code in. You run it through the compiler and it outputs Example.class. Android passes this to another compiler to output example.dex which android can interpret. ART is a new android runtime which takes a class and generates an Example.oat file. It is not yet ready for production. A class is just a module which contains functions and data. Think of it like a robot. Robot.java could contain the makeCoffee() function and dance() function. It could also know my name. The class would look like this.

class Robot {

public String creatorName = “jimmy”;

public String makeCoffee(){ … } // “...” is used to indicate more code which doesn’t matter in this example.

public String dance(){ … }

} // more information http://docs.oracle.com/javase/tutorial/java/javaOO/

You would create a new robot just by calling: Robot bobTheRobot = new Robot();

More info here: http://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html

**public** just means that anyone can access those fields or methods. **String** is the return type of the function. For example if I tell my robot **makeCoffee** it should talk to me by returning **“I’m done making coffee”** when it finishes making my coffee. String just means that text will be returned. String is capitalized because it’s a class just like Robot, and all classes are capitalized. Someone on the internet may have already written a robot with a lot of cool butler and even karaoke features. I can download that code and still add it to my robot by just changing 1 line and adding SomeOneElsesRobot.java to my src or library folder.

class Robot extends SomeoneElsesRobot {

Amazing! If SomeOneElsesRobot already has the ability to makeCoffee(), I would have to add @Override above the makeCoffee() function to show that my robot knows how to make my favorite coffee best and not to use the other guy’s makeCoffee() functionality. To access information in my robot from another other java code I would just say Robot.creatorName or Robot.makeCoffee(). Lets say I have an army of robots that I’m preparing for world domination. Leaving things **public** would be a bad idea because then anyone can use my robots. It would be better to set them as private, so that only other robots in my collection could make each other do things. There’s another feature called **private** for things each individual robot should know how to do, not no one else should make them do. For example my robots can take big clunky dumps of scrapmetal after eating their robot victims which are then refined into ores. No other robot should tell another robot when it’s time to refine copper ore nuggets. Such things are best known by the robot itself.

private CopperNugget takeDump(){ int nuggets = 10; while (nugget>0) nugget--; return new CopperNugget(); }

“int” is actually a primitive type but not a class. There are others like: char, double, float, long, byte. You could also say “Integer nuggets = 10;” and it would work exactly the same. “nuggets” is just a random name i picked to contain the number 10. It is not a class therefore it is not capitalized. It’s only capitalized when there’s a class somewhere by the same name, often sharing the same name as a java file. The difference is classes has extras like content or functions which can operate on the data you give it. For example calling nuggets.toString() would turn 10 into the String “10” only we declare nugget as “Integer nuggets = 10”. If we just use “int” then “nuggets.toString()” won’t work because only the Integer class has the method “toString”. **return** means we’re giving back a value of type CopperNugget, that’s why CopperNugget is right by private in the beginning. When you takeDump() it outputs a CopperNugget.

We can also create constructors which give information to the robot when creating it.

class Robot {

public String ownerName;

public Robot(String owner){

ownerName = owner

}

...

} // more examples http://docs.oracle.com/javase/tutorial/java/javaOO/constructors.html

If I sell a robot to bob, bob wants a way to start up the Robot knowing who his owner is. I could have also used “this.ownerName”. “this” keyword just means that we’re talking about the current class. Read this thread for examples of “this” keyword use: http://stackoverflow.com/questions/2411270

Using a constructor he can now use the code ‘new Robot(“Bob”);’ to create a new robot that knows his name. A constructor doesn’t have a return type and HAS TO have the same name as the class. There can also be multiple constructors with different numbers of arguments. If I extend “SomeoneElsesRobot” again and they also have a constructor, I can call that constructor using “super” before performing my own actions. If the other person already wrote code to construct and initialize a mecha laser weapon when you turn the robot, why not let it do its thing?

public Robot(String owner){

super(); // do cool stuff in whatever code we **extend**ed.

ownerName = creator;

} // more examples: http://docs.oracle.com/javase/tutorial/java/IandI/super.html

You will also often see things like:

private static final String TAG = "MyActivity";

static means that all new classes share the same information. final means that this is never changed once it is set. It is a constant. By convention we make constants all caps. The TAG is used in the Log.e(Tag, Msg) set of functions to indicate where the logged message is coming from. Use it in all your classes!

I know that’s a lot of information but that’s 90% of what you need. Java is a big language with a lot of quirks and features explainable only by understanding its history and evolution. You can still learn as you code!

**Rock Paper Scissors example!**

1. Eclipse: Android -> new -> project -> android application project

1. Android Studio: File -> new project.

1. Click next over and over until you see the option to select “blank activity”. Select it.

2. Let’s make the GUI. Go to activity\_my.xml in **main -> res -> layout.** Click the design tab. Click Hello World. Delete it using the DEL key. Drag and drop a LinearLayout (Vertical) onto the screen and make it take up the whole white part of the screen.. Drag and drop three Button Widgets into the linear layout. Click on the “text” tab. Change android:layout\_width=“wrap\_content” to “fill\_parent” instead for each button. Change the button android:text, to say “Rock”, “Paper”, “Scissors”. Change the IDs to “@+id/rock”, “@+id/paper”, “@+id/scissors”. Change LinearLayout’s layout\_height attribute to “wrap\_content”. Go back to design. Select and drag a LinearLayout (horizontal) under the buttons. Select and drag two image views into that horizontal LinearLayout. Go back to the “text” tab. Set the IDs for the ImageViews to “@+id/myAnswer” and “@+id/computerAnswer”.

3. Let’s find some images for our game! Go to images.google.com and search “rock”. Click search tools. click size -> exactly -> set it to 64 x 64. Find a pretty icon for rock, paper and scissors and save them to desktop.

5. Let’s import the images. In your **res -> drawable** folders, it doesn’t matter which one. Name the files rock, paper, scissors. Name the image resource name rock, paper, scissors.

**In android studio:** right click -> new -> image asset.

**In eclipse:** ctrl-c the file, then right click paste it into the folder. You can also drag and drop the file.

6. We’re ready to code! Go to **src -> main -> java.**

Add these lines where the other imports are.

**import android.content.DialogInterface;**

**import android.app.AlertDialog;**

**import android.util.Log;**

**import android.view.View;**

**import android.view.View.OnClickListener;**

**import android.widget.ImageView;**

Change “public class MyActivity extends Activity **implements OnClickListener** {”.

This means we can handle click events on our buttons. Now we’re using the interface “OnClickListener”, which just means that our class has an onClick method.

Lets add a do-nothing onClick method. You can add it under the closing bracket of the onCreate method.

@Override

public void onClick(View v) { }

Inside at the bottom of the onCreate() function, add:

findViewById(R.id.rock).setOnClickListener(this);

findViewById(R.id.paper).setOnClickListener(this);

findViewById(R.id.scissors).setOnClickListener(this);

R is the class the xml files generated. We’re asking android to find the view with ID rock, paper, and scissors as specified by the R xml, with the id attribute. That function returns the resulting UI object of class View. We then tell it that we want to handle mouse clicks and that the current class (this class), will be the one with an “onClick” function that is called when one of the buttons is touched. I know we created buttons, but buttons are just another type of view. We could have casted the view into a button object like this.

Button rock = (Button) findViewById(R.id.rock);

rock.setOnClickListener(this);

That is equivalent to the previous code. Since java is a typed language, findViewById(...) can only give back 1 type, in this case they chose View, which all widgets extend. Using (Button) casts the View into a Button type which has extra functionality. The android team could have made a function for each type you’d want to access, like using findButtonById(..) but they chose not to because anyone can create their own custom View extended UI widgets and still access them through that same function. You can see the Button object adds by going to the below pages. The additional functions Button supports will be under “Public Methods”

<http://developer.android.com/reference/android/view/View.html> (Button will have all these functions)

<http://developer.android.com/reference/android/widget/Button.html> (Fuctions button has but view doesn’t)

Let’s update the onClick to actually do something.

@Override

public void onClick(View v){

Integer[] images = {R.drawable.rock, R.drawable.paper, R.drawable.scissors};

ImageView computerAnswer = (ImageView) findViewById(R.id.computerAnswer);

computerAnswer.setImageResource(images[((int)(Math.random() \* 10)) % 3]);

ImageView myAnswer = (ImageView) findViewById(R.id.myAnswer);

int id = v.getId();

myAnswer.setImageResource(images[(id == R.id.rock) ? 0 : (id == R.id.paper) ? 1 : 2]);

}

7. Compile and view the program. It should display the image we choose as well as the image the computer chose. Now let’s make the logic for who won.

// ROCK = 0; PAPER = 1; Scissors = 2;

if (myChoice == computerChoice) Log.e("RPS","Draw!");

else if (myChoice == 2 && computerChoice == 0) Log.e("RPS","Lose!");

else if (myChoice == 1 && computerChoice == 2) Log.e("RPS","Lose!");

else if (myChoice == 0 && computerChoice == 1) Log.e("RPS","Lose!");

else Log.e("RPS", "Win!");

If the choices are equal it is a draw, then I set three possible conditions for losing, everything else that’s not a draw or a loss is a win. Instead of Log.e, we can use initialize a dialogue box, similar to an alert box in javascript or messagebox in windows api. The ending code looks like this.

public void alert(String msg){

AlertDialog.Builder builder = new AlertDialog.Builder(MyActivity.this);

// Set the alert box text.

builder.setMessage(msg);

// we need this otherwise we can't click "ok" on the dialogue.

builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface dialog, int which) {}

});

// Create an AlertDialog type then show() it

builder.create().show();

}

@Override

public void onClick(View v) {

final Integer[] images = {R.drawable.rock, R.drawable.paper, R.drawable.scissors};

ImageView computerAnswer = (ImageView) findViewById(R.id.computerAnswer);

int computerChoice = ((int)(Math.random() \* 10)) % 3;

computerAnswer.setImageResource(images[computerChoice]);

ImageView myAnswer = (ImageView) findViewById(R.id.myAnswer);

int id = v.getId();

int myChoice = (id == R.id.rock) ? 0 : (id == R.id.paper) ? 1 : 2;

myAnswer.setImageResource(images[myChoice]);

// ROCK = 0; PAPER = 1; Scissors = 2;

if (myChoice == computerChoice) alert("Draw!");

else if (myChoice == 2 && computerChoice == 0) alert("You Lose!");

else if (myChoice == 1 && computerChoice == 2) alert("You Lose!");

else if (myChoice == 0 && computerChoice == 1) alert("You Lose!");

else alert("You Win!");

}