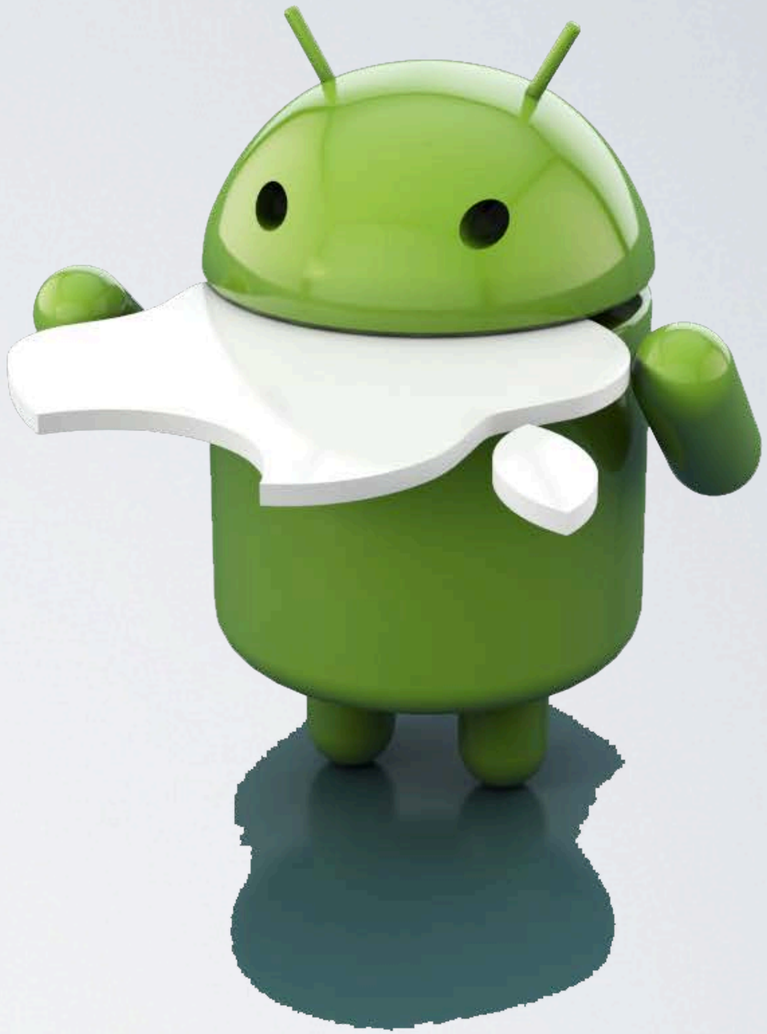
# ANDROID

## Intro To Android App Creation



# JESSE SCOTT

jscott@langara.bc.ca

# SENSORS

# SENSOR CLASS

The Android Sensor Class can be found at
http://developer.android.com/reference/android/hardware/Sensor.html

This is the base class that holds all the types of sensors available (Accelerometer, Orientation, Pressure, Gravity, etc), and the methods to query them.

# SENSOR MANAGER CLASS

The Android Sensor Manager Class can be found at
http://developer.android.com/reference/android/hardware/SensorManager.html

This is the class that allows you to access one of the sensors (of the types returned from the Sensor class).

# SENSOR EVENTLISTENER CLASS

The Android Sensor EventListener Class can be found at
http://developer.android.com/reference/android/hardware/SensorEventListener.html

This is the class that allows you to register / unregister a listener to get access to the current data (event) of the sensor type you are asking for.

# SENSOR EVENT CLASS

The Android Sensor Event Class can be found at
http://developer.android.com/reference/android/hardware/SensorEvent.html

This is the class that actually (finally!) gives you the current event from the sensor (i.e. a change in the XYZ position of the Accelerometer, etc.)

# ACCELEROMETER

# JAVA CLASS - IMPORTS

In our MainActivity.java file, let's add our imports... first for the Sensors...
(to allow us to access the Accelerometer)

```
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
```

# JAVA CLASS - IMPORTS

In our MainActivity.java file, let's add our imports... then for the Surface View...
(to allow us to create a virtual screen to draw to)

```
import android.view.Display;
import android.view.Surface;
import android.view.View;
import android.view.WindowManager;
```

# JAVA CLASS - IMPORTS

In our MainActivity.java file, let's add our imports... then for Canvas (to draw to the virtual screen)

```
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Color;
```

# JAVA CLASS - DECLARATIONS

Then, let's add our Class Declarations and Global Variables:

```
private SimulationView mSimulationView;
private SensorManager mSensorManager;
private WindowManager mWindowManager;
private Display     mDisplay;
```

# JAVA CLASS - INSTANTIATIONS

In our onCreate() method, let's instantiate our objects:

```
// Get an instance of the SensorManager
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

// Get an instance of the WindowManager
mWindowManager = (WindowManager) getSystemService(WINDOW_SERVICE);
mDisplay = mWindowManager.getDefaultDisplay();

// instantiate our simulation view and set it as the activity's content
mSimulationView = new SimulationView(this);
setContentView(mSimulationView); // note we're NOT using main.xml !!!
```

# JAVA CLASS - LIFE CYCLE

In our MainActivity Class, let's make the onResume() and onPause() methods to make sure we get/release our sensor! (note we're calling custom methods...)

```
@Override
protected void onResume() {
    super.onResume();
    mSimulationView.startSimulation();
}

@Override
protected void onPause() {
    super.onPause();
    mSimulationView.stopSimulation();
}
```

# JAVA CLASS - NEW CLASS

*Inside* our MainActivity Class, let's make a new class called SimulationView

```
class SimulationView extends View implements SensorEventListener {


}
```

NOTE that this is *inside* our existing class !!!

# JAVA CLASS - DECLARATIONS

Then, let's add our Class Declarations and Global Variables:

```
private Sensor mAccelerometer;
private float mSensorX;
private float mSensorY;
private float mSensorZ;

Paint paint = new Paint();
```

# JAVA CLASS - CONSTRUCTOR

Then, let's create our Class Constructor:

```
public SimulationView(Context context) {
    super(context);
    mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
}
```

# JAVA CLASS - START / STOP

Then, let's create our start/stop methods (which are called by onResume/onPause:

```
public void startSimulation() {
    mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_UI);
 }

 public void stopSimulation() {
    mSensorManager.unregisterListener(this);
}
```

This is done because it's better to make the *class that owns the sensor* do this...

# JAVA CLASS - SENSOR METHODS

For any sensor, we need to implement two methods - onAccuracyChanged() and onSensorChanged() ...

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
  //
 }

@Override
public void onSensorChanged(SensorEvent event) {
  //
}
```

# JAVA CLASS - ACCURACY

For onAccuracyChanged(), we don't need to do anything...

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
  //
}
```

# JAVA CLASS - SENSOR

For onSensorChanged(), we double check that we have the Accelerometer, then access the XYZ values from an array of floats :

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() != Sensor.TYPE_ACCELEROMETER) {
        return;
    }
    else {
        mSensorX = event.values[0];
        mSensorY = event.values[1];
        mSensorZ = event.values[2];
    }
}
```

# JAVA CLASS - PAINT

Then, let's setup a draw method for our canvas, and paint the values to the screen!

```
@Override
protected void onDraw(Canvas canvas) {
    // Set Our Stylr
     paint.setColor(Color.WHITE);
     paint.setTextSize(36);
    // Draw The Text
     canvas.drawText(String.valueOf("OUR X SENSOR IS: " + mSensorX), 100, 100, paint);
     canvas.drawText(String.valueOf("OUR Y SENSOR IS: " + mSensorY), 100, 200, paint);
     canvas.drawText(String.valueOf("OUR Z SENSOR IS: " + mSensorZ), 100, 300, paint);
    // Clear The Screen
    invalidate();
}
```

# LOCATION

# JAVA CLASS - IMPORTS

In our MainActivity.java file, let's add our imports... first for the Location...
(to allow us to access the GPS)

```
import android.location.Location;
import android.location.LocationManager;
import android.location.LocationListener;
import android.location.GpsStatus.Listener;
import android.location.GpsStatus.NmeaListener;
```

# JAVA CLASS - IMPORTS

In our MainActivity.java file, let's add our imports... then for the Surface View...
(to allow us to create a virtual screen to draw to)

```
import android.view.View;
import android.content.Context;
```

# JAVA CLASS - IMPORTS

In our MainActivity.java file, let's add our imports... then for Canvas (to draw to the virtual screen)

```
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Color;
```

# ANDROID MANIFEST - PERMISSIONS

In our AndroidManifest.cml file, let's add our permissions...

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

# JAVA CLASS - DECLARATIONS

Then, let's add our Class Declarations and Global Variables:

```
// Declarations
LocationManager locationManager;
MyLocationListener locationListener;
Paint paint = new Paint();

// Variables
float currentLatitude  = 0;
float currentLongitude = 0;
float currentAccuracy  = 0;
String currentProvider = "";
```

# JAVA CLASS - INSTANTIATIONS

In our onCreate() method, let's instantiate our objects:

```
// instantiate our location listener and set it as the activity's content
locationListener = new MyLocationListener(this);
setContentView(locationListener); // note we're NOT using main.xml !!!
```

# JAVA CLASS - LIFE CYCLE

In our MainActivity Class, let's make the onResume() method to make sure we get our sensor! (note we don't need to release it...)

```
@Override
protected void onResume() {
    super.onResume();
    // Acquire a reference to the system Location Manager
    locationManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
    // Register the listener with the Location Manager to receive location updates
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,   locationListener);
}
```

# JAVA CLASS - NEW CLASS

*Inside* our MainActivity Class, let's make a new class called MyLocationListener

```
class MyLocationListener extends View implements LocationListener {


}
```

NOTE that this is *inside* our existing class !!!

# JAVA CLASS - CONSTRUCTOR

Then, let's create our Class Constructor:

```
public MyLocationListener(Context context) {
    super(context);
}
```

# JAVA CLASS - SENSOR METHODS

For any sensor, we need to implement four methods

```
public void onLocationChanged(Location location) {}

public void onProviderDisabled (String provider) {}

public void onProviderEnabled (String provider) {}

public void onStatusChanged (String provider, int status, Bundle extras) {}
```

# JAVA CLASS - LOCATION

For onLocationChanged(), we set our global variables

```
public void onLocationChanged(Location location) {
   currentLatitude  = (float)location.getLatitude();
   currentLongitude = (float)location.getLongitude();
   currentAccuracy  = (float)location.getAccuracy();
   currentProvider  = location.getProvider();
}
```

# JAVA CLASS - PROVIDER

For onProviderDisabled() & onProviderDisabled(), we set/clear the provider:

```
public void onProviderDisabled (String provider) {
  currentProvider = "";
}
```

```
public void onProviderEnabled (String provider) {
  currentProvider = provider;
}
```

# JAVA CLASS - STATUS

For onStatusChanged(), we don't have to do anything...

```
public void onStatusChanged (String provider, int status, Bundle extras) {
  // Nothing...
}
```

# JAVA CLASS - PAINT

Then, let's setup a draw method for our canvas, and paint the values to the screen!

```
protected void onDraw(Canvas canvas) {
   paint.setColor(Color.BLUE);
   paint.setTextSize(36);
   canvas.drawText(String.valueOf("OUR LAT IS: " + currentLatitude),  100, 100, paint);
   canvas.drawText(String.valueOf("OUR LON IS: " + currentLongitude), 100, 200, paint);
   canvas.drawText(String.valueOf("OUR ACC IS: " + currentAccuracy),  100, 300, paint);
   canvas.drawText(String.valueOf("OUR PRO IS: " + currentProvider),  100, 400, paint);
    invalidate();
}
```