COM413 – Mobile App Development – Workbook Practical 3

Introduction

This Practical provides an opportunity for you to develop an application which uses the onboard sensors that are on **most** Android phones.

Most of the android devices have built-in sensors that measure motion, orientation, and various environmental condition. Android includes support for hardware sensors using the Android Sensor Framework.

The framework includes the following classes and interfaces:

- SensorManager
- Sensor
- SensorEventListener
- SensorEvent

Most Android devices include hardware sensors, but they vary greatly between different manufactures and models. If your application utilizes sensors, you have two choices:

- Specify the sensor in the Android Manifest
- Check for the sensor at runtime

The Android SDK provides support for the following sensor types:

Sensor	**Detects**	**Use**
TYPE_ACCELEROMETER	Motion detection including gravity	Used to determine shake, tilt, and so on
TYPE_AMBIENT_TEMPERATURE	Measures ambient room temperature	Used for determining local temperature
TYPE_GRAVITY	Measures the force of gravity on all three axes	Used for motion detection
TYPE_GYROSCOPE	Measures rotation on all three axes	Used to determine turn, spin, and so on
TYPE_LIGHT	Measures light level	Used for setting screen brightness
TYPE_LINEAR_ACCELERATION	Motion detection excluding gravity	Used to determine acceleration
TYPE_MAGNETIC_FIELD	Measures geomagnetic field	Used to create a compass or determine bearing
TYPE_PRESSURE	Measures air pressure	Used for barometer
TYPE_PROXIMITY	Measures object relative to the screen	Used to determine whether the device is being held against the ear during a phone call
TYPE_RELATIVE_HUMIDITY	Measures relative humidity	Used to determine dew point and humidity
TYPE_ROTATION_VECTOR	Measures device orientation	Used to detect motion and rotation

To specify your application uses a sensor, include the **<uses-feature>** declaration in the **Android Manifest**. Here is an example requiring a compass to be available:

```
<uses-feature android:name=”android.hardware.sensor.compass”
android:required=”true”>
```

If your application utilizes the compass, but does not require it to function, you should set **android:required="false”** instead, otherwise the application will not be available through Google Play.

Sensors are grouped into the following three categories.

- **Motion Sensors**
- **Environmental sensors**
- **Position sensors**

So in the basic aspect of this practical we will have 3 separate applications with various functionalities:

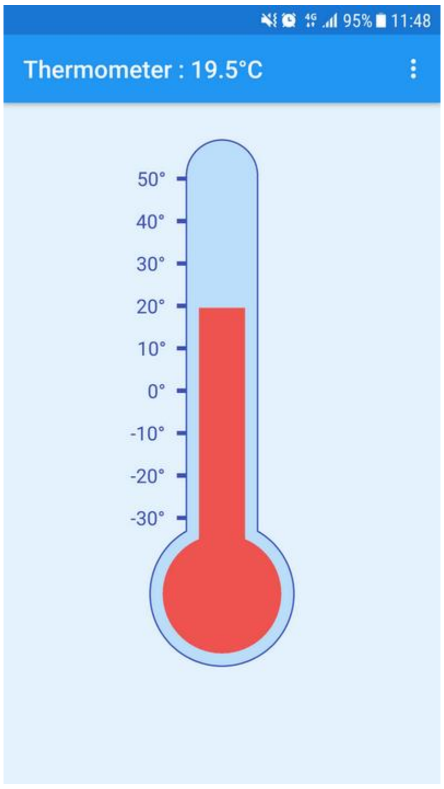
Basic:
Create an app:

- Which lists all sensors available on a phone
- Checks the temperature sensor and displays this in a text view
- Uses the Lux sensor to adjust the brightness of the background

Advanced:

- Create an app which detects the on-board accelerometer and changes the colour of a TextView based on if the user shakes the Phone.

Additional:
This is up to you, read in a sensor and display a graphic on the screen e.g. this temperature sensor:



Basic Functionality:

Part 1.

Now we will demonstrate retrieving a list of available sensors.

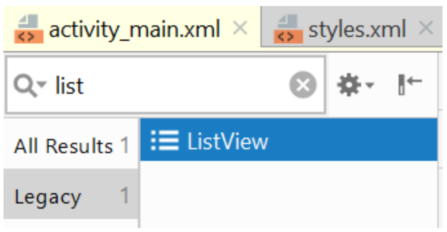
Getting ready

Create a new project in Android Studio and call it: List Device Sensors. Use the default **Phone & Tablet** options and select **Empty Activity** when prompted for the **Activity Type**.

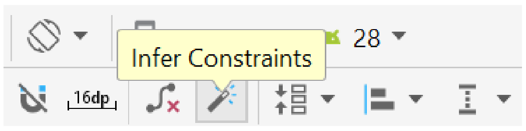
How to do it...

First, we'll query the list of sensors available, then display the results in a ListView. Here are the detailed steps:

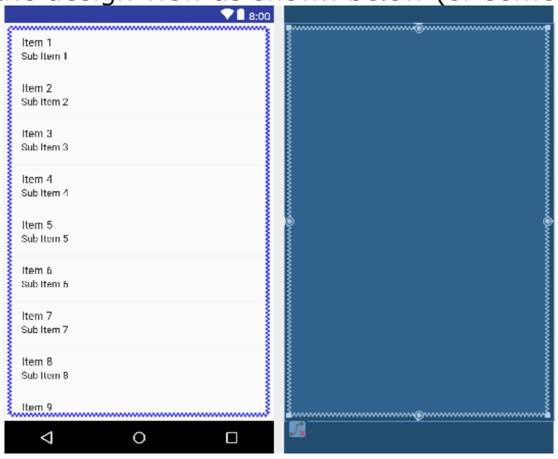
1. Open **activity_main.xml**, you need to remove what's already in the **layout**. So, click **TextView** in the **Component Tree** window, and then press **Delete**.
2. Now, we need to add a **ListView**.
    a. Search the word **list** in the **Palette**,



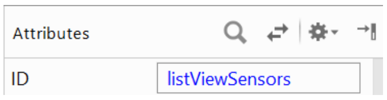
    b. Drag **ListView** into the design editor and drop it.
    c. Click on the **Infer Constraints**



    d. You will find the design view as shown below (or something similar).



e. Click the List View in the layout and, locate the **ID** in the Attributes window and provide **listViewSensors** as ID name.

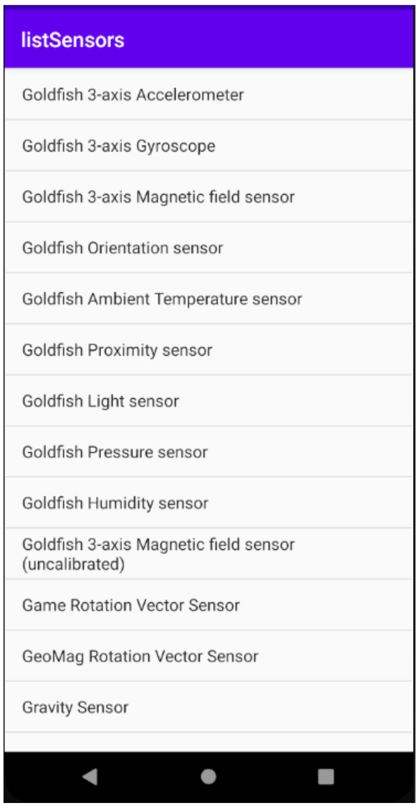


ID listViewSensors

3. Next, open **ActivityMain.java** and add the following code to the existing **onCreate()** method:

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

    ListView listView = (ListView)findViewById(R.id.listViewSensors  );
    List sensorList = new ArrayList<String>();
    List<Sensor> sensors = ((SensorManager) getSystemService(
            Context.SENSOR_SERVICE)).getSensorList(Sensor.TYPE_ALL);
    for (Sensor sensor : sensors ) {
        sensorList.add(sensor.getName());
    }
    ListAdapter sensorAdapter = new ArrayAdapter<String>( context: this,
            android.R.layout.simple_list_item_1, sensorList);
    listView.setAdapter(sensorAdapter);

}
```

4. Run the program on an emulator. The below screen-shot shows the list of sensors from a **Pixel3 API 29**.


listSensors

Goldfish 3-axis Accelerometer
Goldfish 3-axis Gyroscope
Goldfish 3-axis Magnetic field sensor
Goldfish Orientation sensor
Goldfish Ambient Temperature sensor
Goldfish Proximity sensor
Goldfish Light sensor
Goldfish Pressure sensor
Goldfish Humidity sensor
Goldfish 3-axis Magnetic field sensor (uncalibrated)
Game Rotation Vector Sensor
GeoMag Rotation Vector Sensor
Gravity Sensor

How it works...
The following line of code is responsible for getting the list of available sensors; the rest of the code is to populate the ListView:

```
    List<Sensor> sensors = ((SensorManager) getSystemService(
   Context.SENSOR_SERVICE)).getSensorList(Sensor.TYPE_ALL);
```

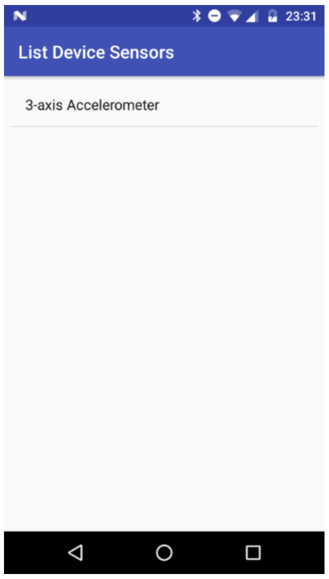
Notice that we get back a list of Sensor objects. We only get the sensor name to display in the ListView, but there are other properties available as well. See the link below for a complete list.

http://developer.android.com/reference/android/hardware/Sensor.html

A device can have multiple sensors of the same type. If you are looking for a specific sensor, you can pass in one of the constants from the table shown in page 1 or 2. In this case, if you wanted to see all the **Accelerometer** sensors available, you could use this call:

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

    ListView listView = (ListView)findViewById(R.id.listViewSensors  );
    List sensorList = new ArrayList<String>();

    SensorManager sensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
    List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
    for (Sensor sensor : sensors ) {
        sensorList.add(sensor.getName());
    }
    ListAdapter sensorAdapter = new ArrayAdapter<String>( context: this,
            android.R.layout.simple_list_item_1, sensorList);
    listView.setAdapter(sensorAdapter);
}
```

The below screen-shot shows the **Accelerometer sensor** from a **Motorola Moto G Play:**



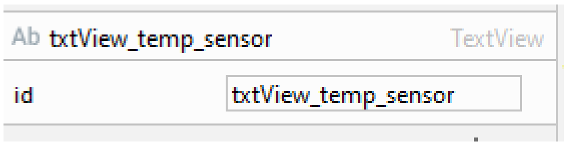
3-axis Accelerometer

Part 2:

Displaying the Temperature Sensor:

Start a new project.

Change the id of the Hello World TextView to txtView_temp_sensor.



id: txtView_temp_sensor

That’s the only thing we’ll change on the **activity_main.XML**.

Now in the MainActivity.java, we need to implement SensorEventListener as such:

```java
public class MainActivity extends AppCompatActivity implements SensorEventListener {
    TextView temp_output;
    SensorManager sensorManager;
    Sensor tempSensor;
    Boolean isTempAvailable;
```

We can also get all our variables out of the way.

Next, we link our temperature output to the XML and we’ll check if the temperature sensor is available on our device:

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

        temp_output = findViewById(R.id.txtView_temp_sensor);
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

        if(sensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE)!=null)
        {
            tempSensor = sensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE);
            isTempAvailable = true;
        }
        else {
            temp_output.setText("Temp is not available!");
            isTempAvailable = false;
        }
    }
```

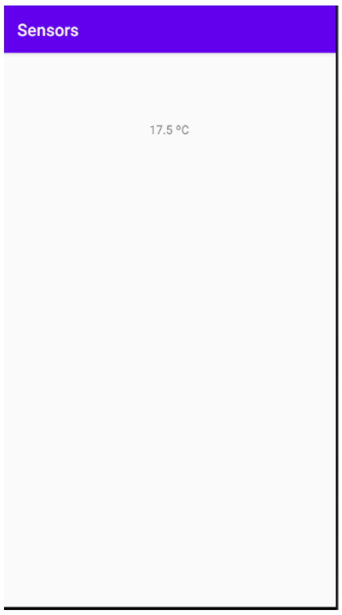
When we implement the previous bit of code we will get an error. It will tell us that we need to implement certain functions when using this sensor, we can click alt+enter and it will automatically create the following functions:

```
@Override
public void onSensorChanged(SensorEvent event) { temp_output.setText(event.values[0]+" ºC"); }

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

}
```

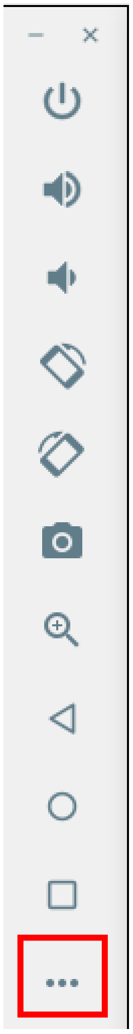
And finally, we need to create two methods to register and unregister the registerListener based on the activity lifecycle or current state of the app:

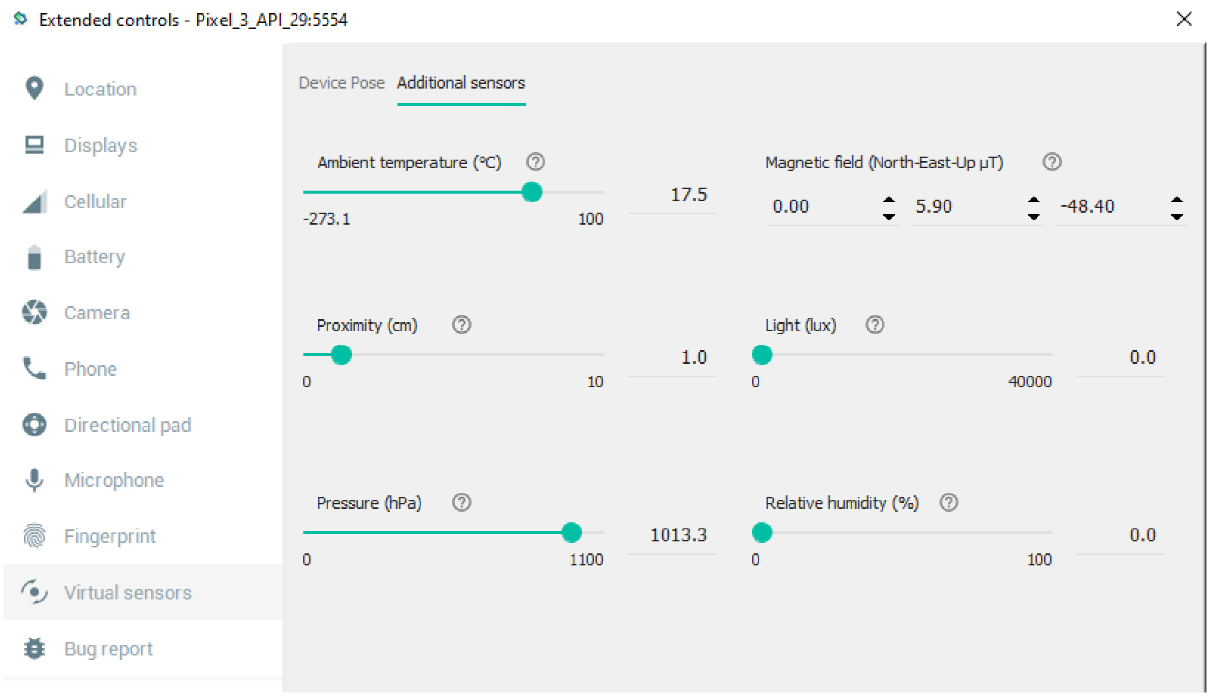
```
@Override
protected void onResume() {
    super.onResume();
    if(isTempAvailable)
    {
        sensorManager.registerListener( listener: this, tempSensor, SensorManager.SENSOR_DELAY_NORMAL);
    }
}

@Override
protected void onPause() {
    super.onPause();
    if(isTempAvailable)
    {
        sensorManager.unregisterListener(this);
    }
}
```

Run this and you should get the following output:



We can change this value by clicking on our emulator and choosing virtual sensors:




Extended controls - Pixel_3_API_29:5554

Ambient temperature (°C) -273.1 100 17.5

Magnetic field (North-East-Up μT) 0.00 5.90 -48.40

Proximity (cm) 0 10 1.0

Light (lux) 0 40000 0.0

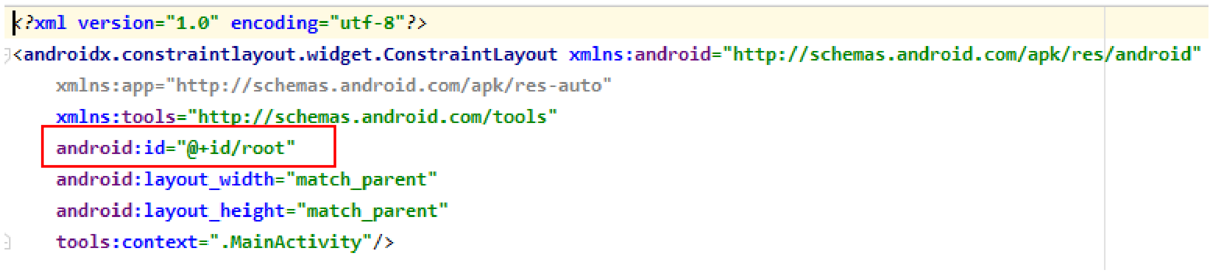
Pressure (hPa) 0 1100 1013.3

Relative humidity (%) 0 100 0.0

Part 3:

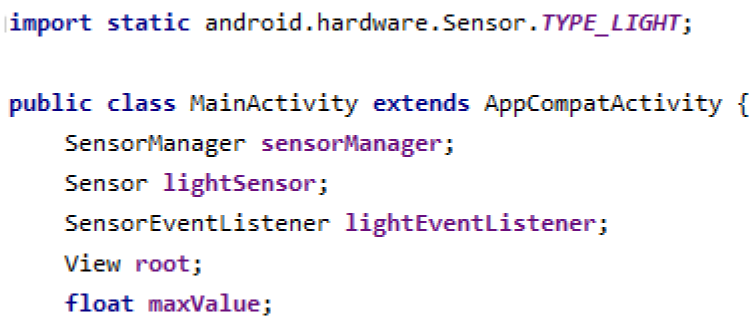
So, we can display the values of certain sensors but what if we Displaying the Lux Sensor and adjusting the brightness. This application is similar to how your screen adjusts brightness based on 2 factors: 1. External input from the sensor or camera, and 2. The User Input. For instance, you can override the sensor if you want the screen brightness to change.

This app needs a little bit of messing about in the XML, we need to open up the XML in programming view and add this one line:



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/root"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"/>
```

So, we are naming our background root. This makes it easier to manipulate later. Next, we add the following variables:



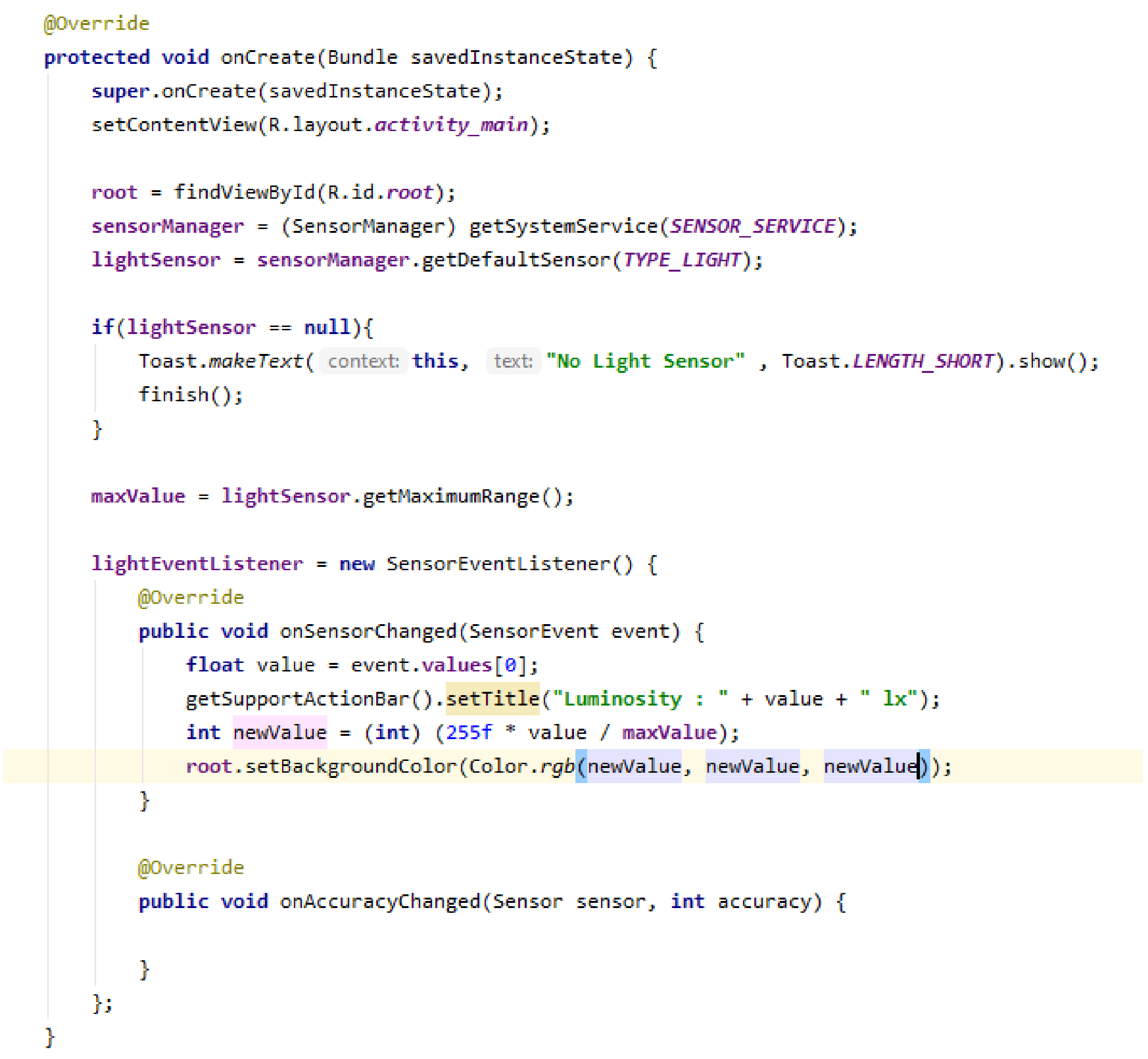
```java
import static android.hardware.Sensor.TYPE_LIGHT;

public class MainActivity extends AppCompatActivity {
    SensorManager sensorManager;
    Sensor lightSensor;
    SensorEventListener lightEventListener;
    View root;
    float maxValue;
```

In part 2 of the practical, you can change the Ambient_temp_sensor to light_sensor and it will display the value of the luminosity in Lux. However, we want to take that value and manipulate the background colour.

So, check first of all, does the device have a sensor, if not tell the user via a toast.

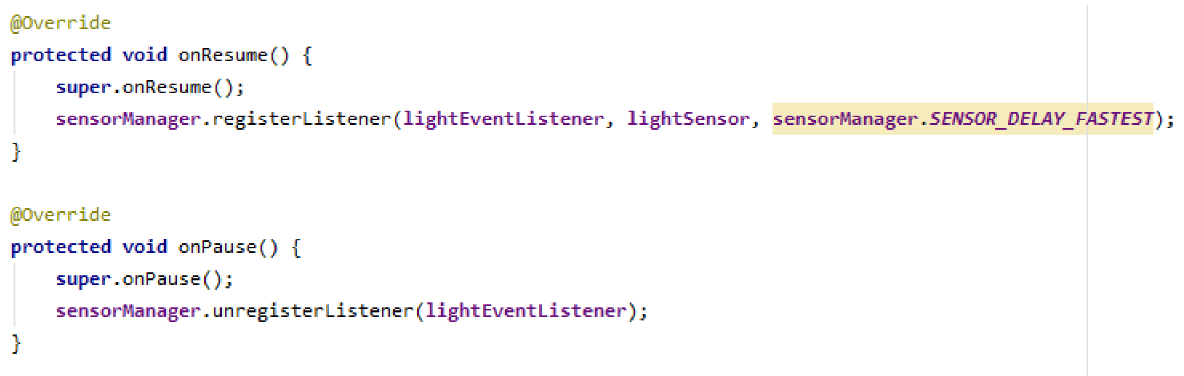
If the User does have a sensor, we create a sensor event listener and then we can set up a SupportActionBar, to display the information without being in the Background. The next part is to take the LUX value scale and normalise it: 255f is the colour of a single pixel, 0-255 is 256 bits of information. The LUX value is between 0 and 40k. So, we multiply our LUX value by 255f and divide it by maxValue every time we get a new value. This means that our scale of 0 to 40k is now a scale between 0 and 255.

RGB means 3 separate channels, Red, Green and Blue, but if we set all these values to the same value, we get a scale from black to white. This means that if all 3 channels are set to the same value it will mimic brightness.

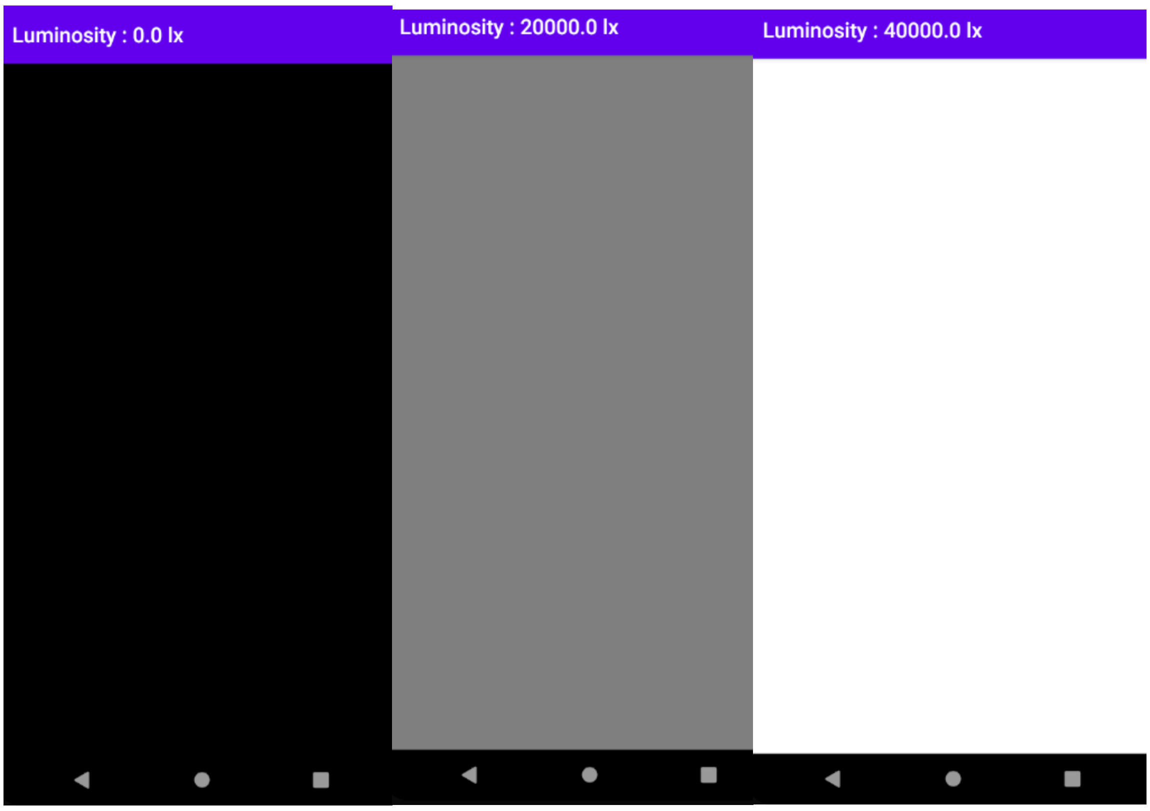


```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    root = findViewById(R.id.root);
    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    lightSensor = sensorManager.getDefaultSensor(TYPE_LIGHT);

    if(lightSensor == null){
        Toast.makeText( context: this,  text: "No Light Sensor" , Toast.LENGTH_SHORT).show();
        finish();
    }

    maxValue = lightSensor.getMaximumRange();

    lightEventListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            float value = event.values[0];
            getSupportActionBar().setTitle("Luminosity : " + value + " lx");
            int newValue = (int) (255f * value / maxValue);
            root.setBackgroundColor(Color.rgb(newValue, newValue, newValue));
        }

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

        }
    };
}
```

Then we add our onResume and onPause Methods, which will register the listeners when the app is open and unregister when the app is closed. This will stop apps overriding each other and the access to hardware.



```
@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(lightEventListener, lightSensor, sensorManager.SENSOR_DELAY_FASTEST);
}

@Override
protected void onPause() {
    super.onPause();
    sensorManager.unregisterListener(lightEventListener);
}
```


Luminosity : 0.0 lx

Luminosity : 20000.0 lx

Luminosity : 40000.0 lx

Advanced Functionality:

Read Accelerometer Sensor Data

In this practical we show how to read accelerometer sensor data on your device. Reading sensor data in Android is pretty straightforward as it involves implementing the SensorEventListener interface.

SensorManager

Android Sensors supports several sensors via the SensorManager, for example the accelerometer. You can access a SensorManager via getSystemService (SENSOR_SERVICE).

The Sensor class defines several constants for accessing the different sensors.

- Sensor.TYPE_GYROSCOPE
- Sensor.TYPE_MAGNETIC_FIELD
- Sensor.TYPE_ORIENTATION
- Sensor.TYPE_ACCELEROMETER

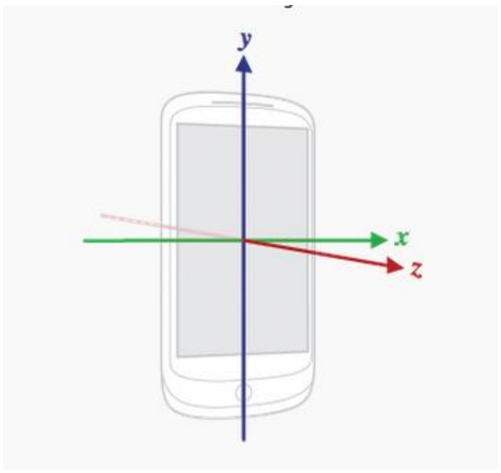
You can access the sensor via the sensorManager.getDefaultSensor() method.

Sensor listener

Once you acquired a sensor, you can register a **SensorEventListener** interface on it. This listener will get informed, if the sensor data changes. This interface is used for receiving notifications from the **SensorManager** when sensor values have changed. You will need to override a couple of methods – **onSensorChanged** and **onAccuracyChanged**. As you may have guessed, **onAccuracyChanged** is called when the accuracy of a sensor has changed and **onSensorChanged** is called when a sensor’s values have changed.

Accelerometer

The figure below shows the x, y and z axis of the accelerometer sensor with respect to the phone's orientation. The X axis is horizontal and points to the right, the Y axis is vertical and points up and the Z axis points towards the outside of the front face of the screen. In this system, coordinates behind the screen have negative Z values.



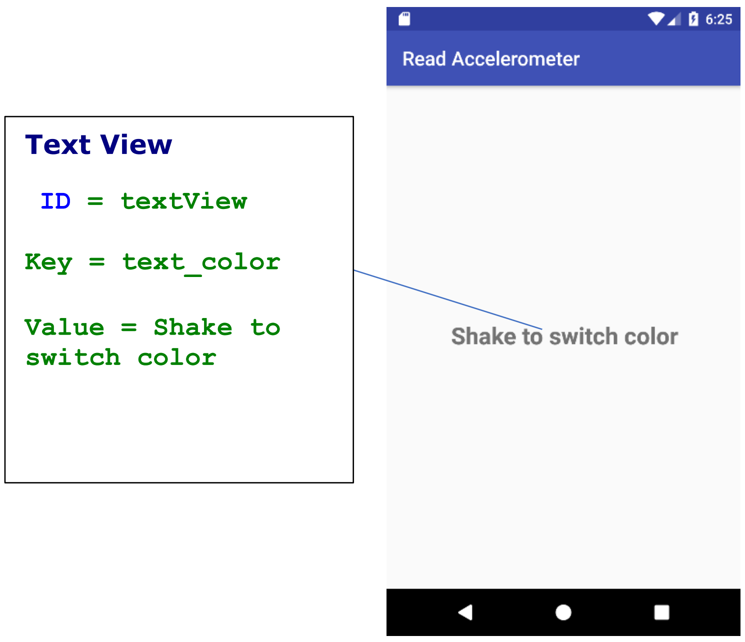
Let’s start by obtaining a reference to the SensorManager and the accelerometer. In order to get an instance of the default accelerometer sensor, we need to do the following:

```
private SensorManager sensorManager;
private Sensor sensor;
  ...
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

We will build an application which will change its background colour simply by shaking the phone or tablet.

Create a new project in Android Studio and call it: **Read Accelerometer**. Use the default **Phone & Tablet** options and select **Empty Activity** when prompted for the **Activity Type**.

1. Modify the activity_main.xml file located in the res/layout folder as shown here:



Text View

ID = textView

Key = text_color

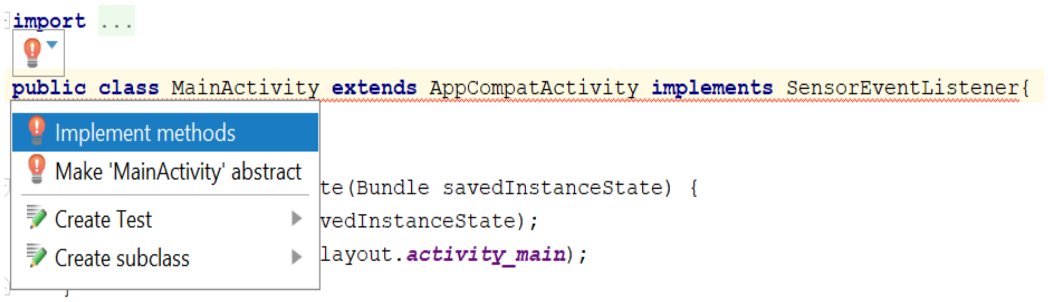
Value = Shake to switch color

2. Now open MainActivity.java and include the SensorEventListerner

```
public class MainActivity extends AppCompatActivity implements
SensorEventListener{
```

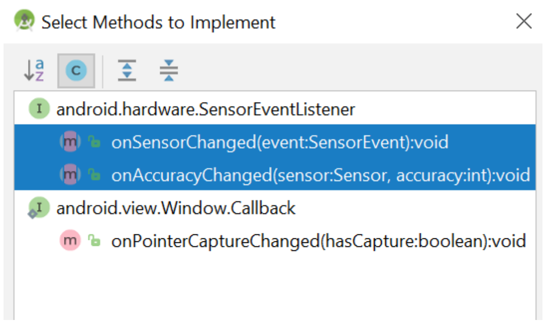
You'll see a few errors pop up. This isn't surprising since we need to implement two required methods of the SensorEventListener interface.

• Click anywhere on the public class MainActivity line. You will see a red bulb. Click on Implement methods option



```
import ...

public class MainActivity extends AppCompatActivity implements SensorEventListener{

te(Bundle savedInstanceState) {
vedInstanceState);
layout.activity_main);
```

- Now you will see a pop-up window as show below:



Select Methods to Implement

android.hardware.SensorEventListener
- onSensorChanged(event:SensorEvent):void
- onAccuracyChanged(sensor:Sensor, accuracy:int):void

android.view.Window.Callback
- onPointerCaptureChanged(hasCapture:boolean):void

- Click OK button. It will give the opportunity to create two new methods:

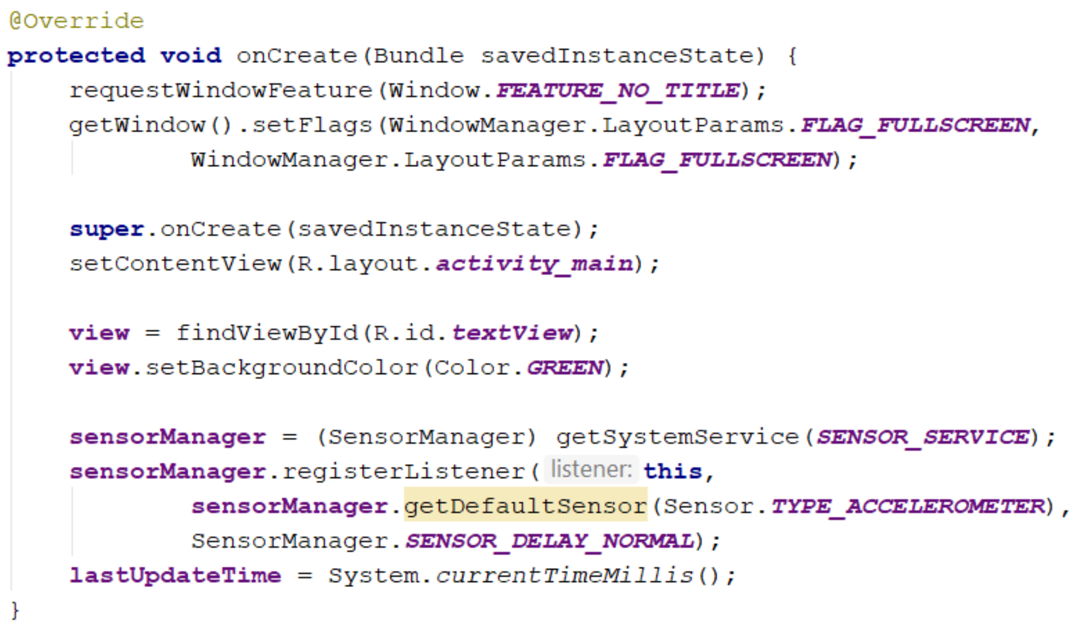
```
@Override
public void onSensorChanged(SensorEvent event) {
}

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

Let's take a look at the **onSensorChanged** method. We will be using this method to detect the shake gesture. The **onSensorChanged** method is invoked every time the built-in sensor detects a change. This method is invoked repeatedly whenever the device is in motion.

3. Add the following global variable declarations:

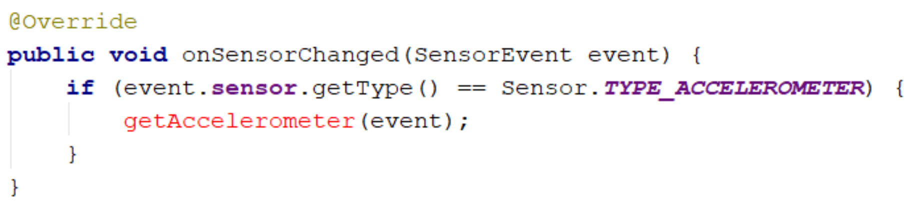
```
private SensorManager sensorManager;
private boolean color = false;
private View view;
private long lastUpdateTime;
private static float SHAKE_THRESHOLD_GRAVITY = 2;

@Override
protected void onCreate(Bundle savedInstanceState) {
```

4. Add the following code to the OnCreate():

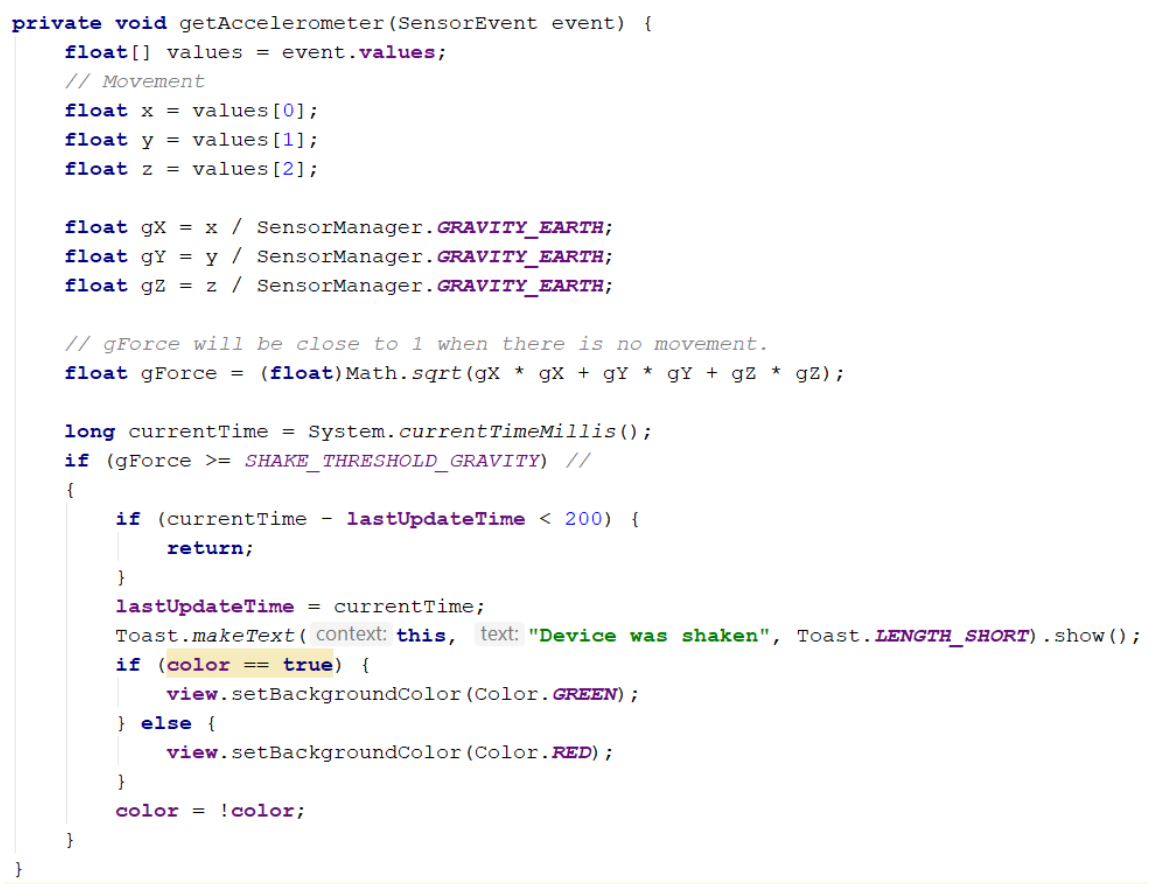

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    view = findViewById(R.id.textView);
    view.setBackgroundColor(Color.GREEN);

    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    sensorManager.registerListener( listener: this,
            sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
            SensorManager.SENSOR_DELAY_NORMAL);
    lastUpdateTime = System.currentTimeMillis();
}
```

To initialize the **SensorManager** instance, we invoke **getSystemService** to fetch the system's **SensorManager** instance. The **getSystemService** method is used to get a reference to a service of the system by passing the name of the service. We get a reference to the system's accelerometer by invoking **getDefaultSensor** on the sensor manager. We then register the sensor using one of the **SensorManager's** public methods, **registerListener**.

5. Add the following code to the **OnSensorChanged**():


```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        getAccelerometer(event);
    }
}
```

6. You will see the “**getAccelerometer**” in red colour. We need to create a new method **getAccelerometer**:



```java
private void getAccelerometer(SensorEvent event) {
    float[] values = event.values;
    // Movement
    float x = values[0];
    float y = values[1];
    float z = values[2];

    float gX = x / SensorManager.GRAVITY_EARTH;
    float gY = y / SensorManager.GRAVITY_EARTH;
    float gZ = z / SensorManager.GRAVITY_EARTH;

    // gForce will be close to 1 when there is no movement.
    float gForce = (float)Math.sqrt(gX * gX + gY * gY + gZ * gZ);

    long currentTime = System.currentTimeMillis();
    if (gForce >= SHAKE_THRESHOLD_GRAVITY) //
    {
        if (currentTime - lastUpdateTime < 200) {
            return;
        }
        lastUpdateTime = currentTime;
        Toast.makeText( context: this,  text: "Device was shaken", Toast.LENGTH_SHORT).show();
        if (color == true) {
            view.setBackgroundColor(Color.GREEN);
        } else {
            view.setBackgroundColor(Color.RED);
        }
        color = !color;
    }
}
```

We get the values of each axis as shown below to extract the device's position in space, the x, y, and z axis.

float x = values[0];

float y = values[1];

float z = values[2];

Then we need to find the “gForce” value from the values of each axis as below:

float gX = x / SensorManager.GRAVITY_EARTH;

float gY = y / SensorManager.GRAVITY_EARTH;

float gZ = z / SensorManager.GRAVITY_EARTH;

float gForce = (float)Math.sqrt(gX * gX + gY * gY + gZ * gZ);

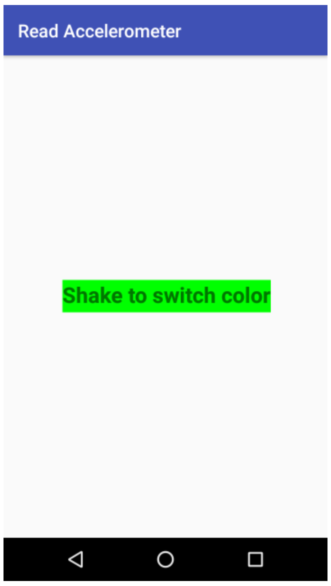
gForce will be close to 1 when there is no movement. Finally, we need to detect whether the device has been shaken or not. Therefore, a statistically declared **SHAKE_THRESHOLD_GRAVITY** variable is used to see whether a shake gesture has been detected or not, i.e. (**gForce** >= SHAKE_THRESHOLD_GRAVITY).

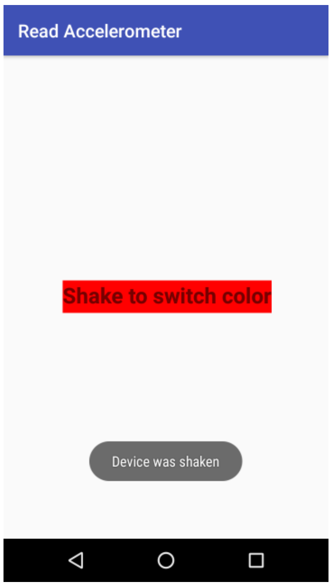
The system's sensors are incredibly sensitive. When holding a device in your hand, it is constantly in motion, no matter how steady your hand is. The result is that the **onSensorChanged** method is invoked several times per second. We don't need all these data. So, we need to make sure we only sample a subset of the data that we get from the device's accelerometer. We used two variables currentTime and lastUpdateTime to handle this.

7. We'll register and unregister the sensor events in the onResume() and onPause() as follows:

```
@Override
protected void onResume() {
    super.onResume();
    // register this class as a listener for the orientation and
    // accelerometer sensors
    sensorManager.registerListener( listener: this,
            sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
            SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onPause() {
    // unregister listener
    super.onPause();
    sensorManager.unregisterListener(this);
}
```

8. You can now run the application on a physical device.





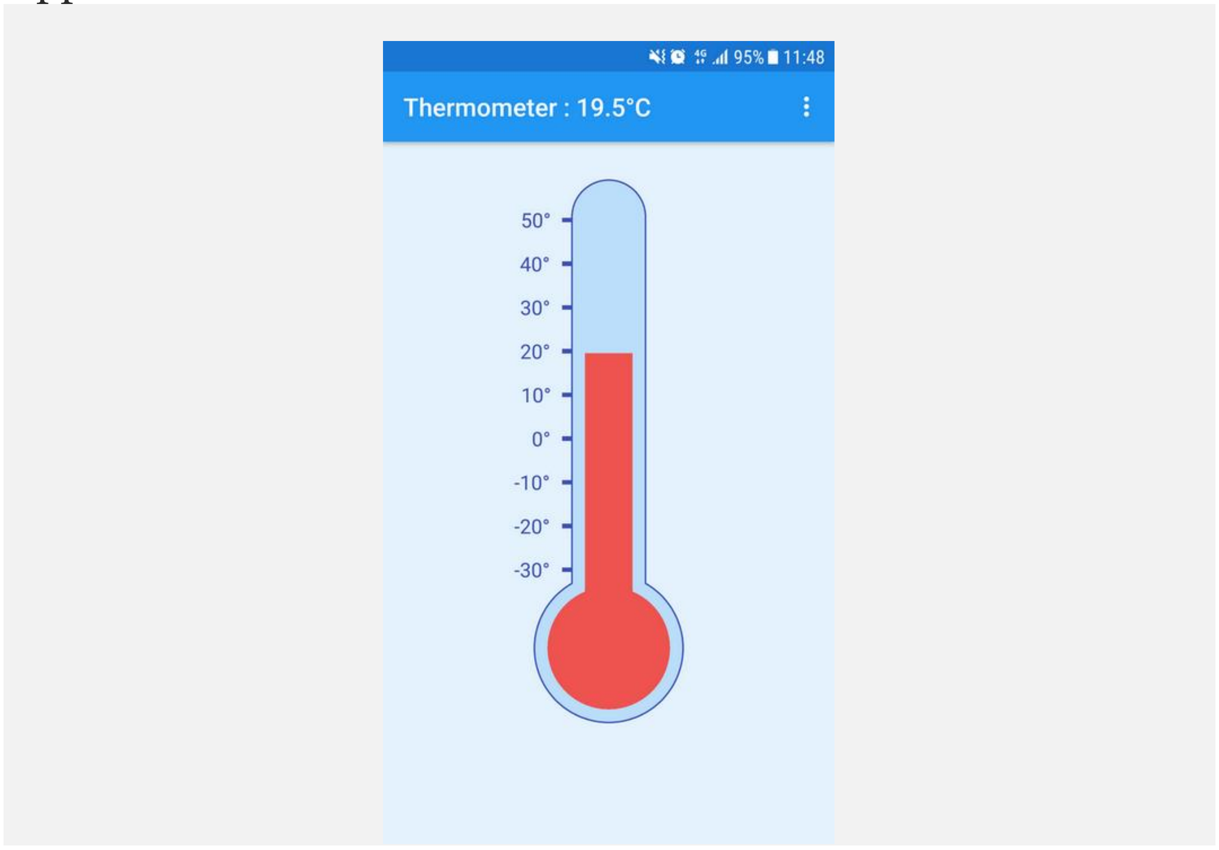
How it works…

Using the Android Sensor Framework starts with obtaining the Sensor, which we do in onCreate(). Here, we call getDefaultSensor(), requesting TYPE_ACCELEROMETER. We register the listener in onResume() and unregister again in onPause() to reduce battery consumption.

Additional Functionality:

Learn to create a Thermometer Application for Android

Android devices have a lot of interesting sensors letting you to make fun applications. Amongst the sensors supported by the Android SDK, the Ambient Temperature sensor lets us to know the temperature measured by an Android device. By using this sensor, we are going to be able to create a Thermometer Application for Android.



https://medium.com/@ssaurel/learn-to-create-a-thermometer-application-for-android-295d6611b4f9