LabW08 – Access to Sensors

Objectives:

1. Understand the motion sensors, environmental sensors and positional sensors
2. Explore the SensorEvent class and SensorEventListener interface
3. Learn to use Activity Recognition Transition API to detect changes in user's activity

Tasks:

1. Access to Accelerometer Sensor
2. Access to Step Counter Sensor
3. [Optional] Using Activity Recognition Transition API

Most mobile devices nowadays come with a varied set of embedded sensors; collectively, these sensors enable the creation of applications across a wide range of domains, such as gaming, healthcare, social networks, and so on. A sensor is simply a device that measures a physical quantity (e.g. the tilt of a device, or sudden movement) and converts it into a readable number that an application can interpret.

In this tutorial, we will focus on learning about the use of sensors. Both Accelerometer and Step Counter sensors are based on a common hardware sensor, which internally uses accelerometer, but Android still treats them as logically separate sensors. Both of these sensors are highly battery optimised and consume very low power.

The last optional part of this tutorial focuses on Activity Recognition Transition API. Developer is spending valuable engineering time to combine various signals like location and sensor data to determine when a user has started or ended an activity like walking or driving. When apps are independently and continuously checking for changes in user activity, battery life suffers. Activity Recognition Transition API helps in solving these problems by providing a simple API that does all the processing for you and just tells you what you actually care about: when a user's activity has changed. Your app subscribes to a transition in activities of interest and the API notifies your app only when needed. As an example, a parking detection app can ask: tell me when a user has exited his vehicle and started walking.

Task 1: Access to Accelerometer Sensor

In this task, we'll use a gesture that you find in quite a few mobile applications, the shake gesture. We'll use the shake gesture to randomly generate a number and display it on the screen using a pretty animation.

1. Create a new project "***COMP5216W08***". Since we're going to make use of a shake gesture, it's a good idea to lock the device's orientation. This will ensure that the application's user interface isn't constantly switching between portrait and landscape. Open the project's ***AndroidManifest.xml*** file and set the ***screenOrientation*** option to portrait.

```
<activity
    android:name=".MainActivity"
    android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

2. We will leverage the ***SensorEventListener*** interface, which is declared in the Android SDK. To use the ***SensorEventListener*** interface, the ***MainActivity*** class needs to implement it as shown in the code snippet below. If you take a look at the updated ***MainActivity*** class, you'll find that it uses the ***implements*** keyword to tell the compiler that the ***MainActivity*** class implements the ***SensorEventListener*** interface.

```
public class MainActivity extends AppCompatActivity implements SensorEventListener {
    /** Called when the activity is first created. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

3. Add ***onSensorChanged*** and ***onAccuracyChanged*** method within your ***MainActivity*** class. Please make sure they are outside of the ***onCreate*** method. Then declare some private variables in the ***MainActivity*** class, ***senSensorManager*** of type ***SensorManager*** and ***senAccelerometer*** of type ***Sensor*.**

```
private SensorManager senSensorManager;
private Sensor senAccelerometer;

private long lastUpdate = 0;
private float last_x, last_y, last_z;
private static final int SHAKE_THRESHOLD = 600;

...

@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    Sensor mySensor = sensorEvent.sensor;

    if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        float x = sensorEvent.values[0];
        float y = sensorEvent.values[1];
        float z = sensorEvent.values[2];

        long curTime = System.currentTimeMillis();

        if ((curTime - lastUpdate) > 100) {
            long diffTime = (curTime - lastUpdate);
            lastUpdate = curTime;

            float speed = Math.abs(x + y + z - last_x - last_y - last_z)/ diffTime *
10000;

            if (speed > SHAKE_THRESHOLD) {
                getRandomNumber();
            }

            last_x = x;
            last_y = y;
            last_z = z;
        }
    }
}

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

}
```

4. Update the onCreate function.

   To initialize the ***SensorManager*** instance, we invoke ***getSystemService*** to fetch the system's ***SensorManager*** instance, which we in turn use to access the system's sensors.

   The ***getSystemService*** method is used to get a reference to a service of the system by passing the name of the service. With the sensor manager at our disposal, we get a reference to the system's accelerometer by invoking ***getDefaultSensor*** on the sensor manager and passing the type of sensor we are interested in. We then register the sensor using one of the ***SensorManager*** 's public methods, ***registerListener***. This method accepts three arguments, the activity's context, a sensor, and the rate at which sensor events are delivered to us.

```
/** Called when the activity is first created. */
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    senSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    senAccelerometer = senSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    senSensorManager.registerListener(this, senAccelerometer ,
SensorManager.SENSOR_DELAY_NORMAL);
}
```

5. Create ***onPause*** and ***onResume*** functions to unregister the sensor when the app hibernates and register the sensor again when the app resumes.

```
protected void onPause() {
    super.onPause();
    senSensorManager.unregisterListener(this);
}

protected void onResume() {
    super.onResume();
    senSensorManager.registerListener(this, senAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
}
```

6. Setup the main layout file for your app.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.comp5216w08.MainActivity">

    <FrameLayout
        android:layout_height="400dp"
        android:layout_width="400dp"
        android:layout_weight="2"
        android:id="@+id/ball_1"
        android:background="@android:color/background_light"
        android:layout_alignParentEnd="true"
        android:layout_alignParentStart="true"
        android:longClickable="false">
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/number_1"
            android:gravity="center"
            android:layout_gravity="center_vertical"
            android:textColor="@android:color/holo_blue_light"
            android:textStyle="bold" />
    </FrameLayout>
</RelativeLayout>
```

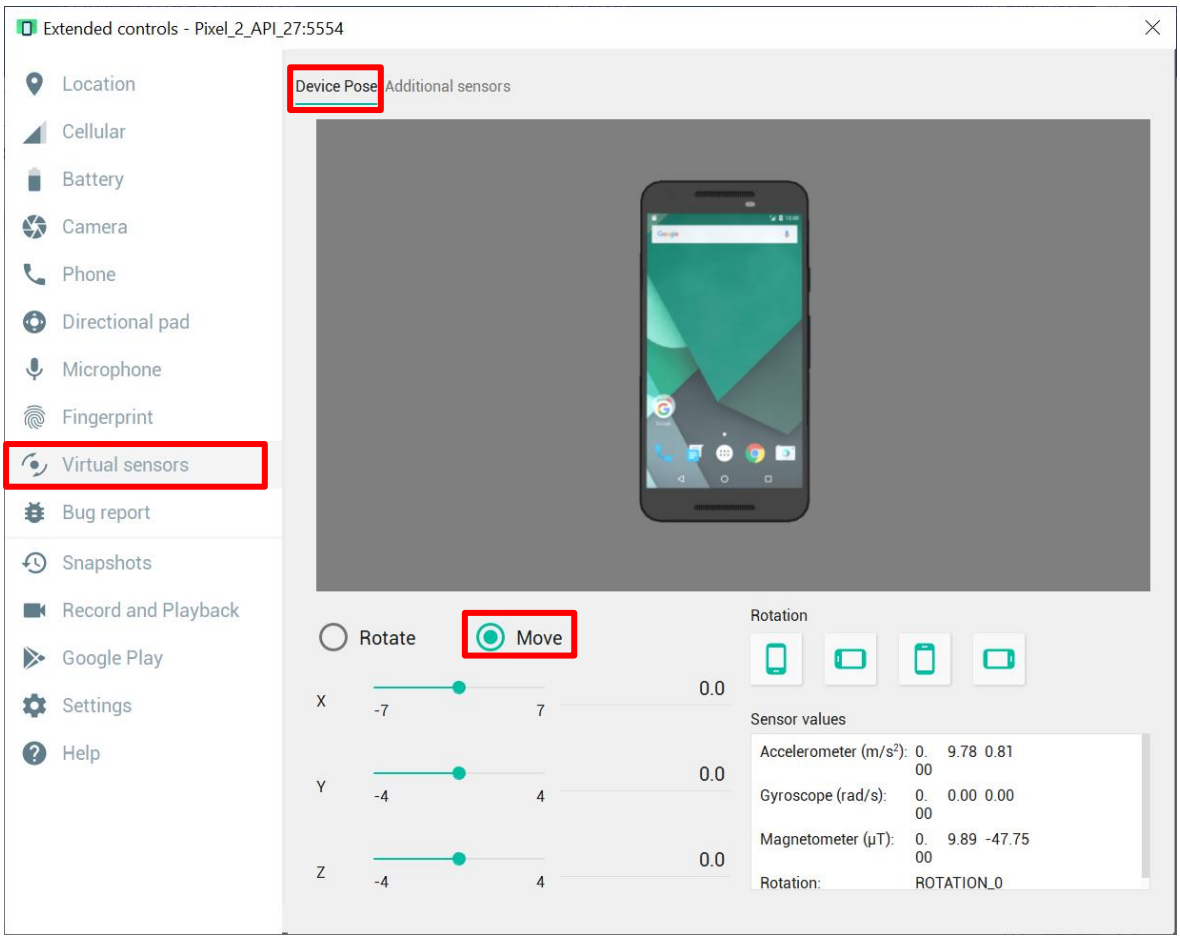
7. As for the animation, take a look at the contents of the animation file below. Note that you need to create an ***anim*** folder in your project's resources directory and name the animation file as ***move_down_ball_first.xml***. By adjusting the values of the scale element, you can modify the animation's duration and the position.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    android:fillAfter="true"
    android:interpolator="@android:anim/bounce_interpolator">

    <scale
        android:duration="1500"
        android:fromXScale="1.0"
        android:fromYScale="-10.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />
</set>
```

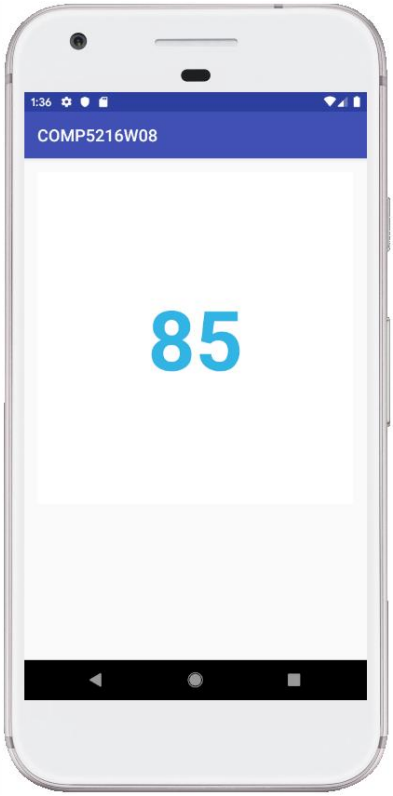
8. Create a ***getRandomNumber*** function in ***MainActivity*** so that every time you shake your mobile, this app will generate a random number between 1 and 100

```
private void getRandomNumber() {
    Random randNumber = new Random();
    int num = randNumber.nextInt(99) + 1;

    TextView text = (TextView)findViewById(R.id.number_1);
    text.setText(""+num);
    text.setTextSize(100);

    FrameLayout ball1 = (FrameLayout) findViewById(R.id.ball_1);
    ball1.setVisibility(View.INVISIBLE);

    Animation a = AnimationUtils.loadAnimation(this, R.anim.move_down_ball_first);

    ball1.setVisibility(View.VISIBLE);
    ball1.clearAnimation();
    ball1.startAnimation(a);
}
```

9. Run your code. If your SDK tools version is higher than 25.0.10, you may test it on the emulator. Launch your emulator and go to More -> Extended controls -> Virtual sensors -> Device Pose. Move around the virtual device on the control to simulate the move gestures on the emulator.

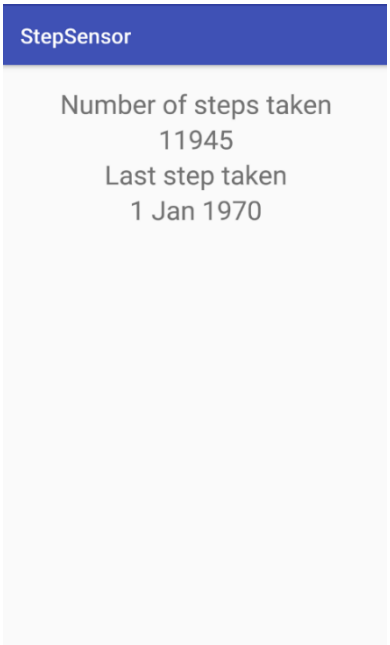


Finally, you should be able to see random numbers generated on the app using the shake gesture:



Task 2: Access to Step Counter Sensor

In this task, we will explore the use of the step sensors (step detector and counter). As shown in the figure below, when you launch this app, the app will display the number of steps taken as well as the time when your last step was taken.



1. Create a new project "***StepSensor***" and declare support for step counter sensor in ***AndroidManifest.xml***

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.comp5216.stepsensor">

    <uses-feature android:name="android.hardware.sensor.stepcounter"
android:required="true"/>
    <uses-feature android:name="android.hardware.sensor.stepdetector"
android:required="true"/>

    ...

</manifest>
```

2. Register the sensor event by adding a ***registerForSensorEvents*** function. Note that this function should be inside the ***MainActivity*** class but outside the ***onCreate*** function. We also need to declare some private variables in ***MainActivity***.

```
private long timestamp;
private TextView textViewStepCounter;
private TextView textViewStepDetector;
private Thread detectorTimeStampUpdaterThread;
private Handler handler;
private boolean isRunning = true;

public void registerForSensorEvents() {
    SensorManager sManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

    // Step Counter
    sManager.registerListener(new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
          float steps = event.values[0];
          textViewStepCounter.setText((int) steps + "");
        }

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

        }
    }, sManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER),
SensorManager.SENSOR_DELAY_UI);

    // Step Detector
    sManager.registerListener(new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            // Set the time when there is new sensor data
            timestamp = System.currentTimeMillis()
                    + (event.timestamp - SystemClock.elapsedRealtimeNanos()) / 1000000L;
        }

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

        }
    }, sManager.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR),
SensorManager.SENSOR_DELAY_UI);
}
```

3. Initialize thread for updating time stamp of your step count by adding a ***setupDetectorTimestampUpdaterThread*** function:

```
private void setupDetectorTimestampUpdaterThread() {
    handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            super.handleMessage(msg);

textViewStepDetector.setText(DateUtils.getRelativeTimeSpanString(timestamp));
        }
    };

    detectorTimeStampUpdaterThread = new Thread() {
        @Override
        public void run() {
            while (isRunning) {
                try {
                    Thread.sleep(5000);
                    handler.sendEmptyMessage(0);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    };

    detectorTimeStampUpdaterThread.start();
}
```

4. Create ***onPause*** function to unregister the sensor when the app hibernates.

```
@Override
protected void onPause() {
    super.onPause();
    isRunning = false;
    detectorTimeStampUpdaterThread.interrupt();
}
```

5. Setup the main layout for your app.

   For simplicity we use a ***LinearLayout*** which contains 4 ***TextViews*** to display the number of steps taken and the last step time.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="20dp" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="Number of steps taken"
        android:textSize="25sp" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="---"
        android:textSize="25sp" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="Last step taken"
        android:textSize="25sp" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="---"
        android:textSize="25sp" />

</LinearLayout>
```

6. Update the ***onCreate*** function in ***MainActivity***.

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

    textViewStepCounter = (TextView) findViewById(R.id.textView2);
    textViewStepDetector = (TextView) findViewById(R.id.textView4);

    registerForSensorEvents();
    setupDetectorTimestampUpdaterThread();
}
```

7. Run your code. It would be better to test the app on a real Android mobile device.

[Optional] Task 3: Using Activity Recognition Transition API

The Activity Recognition Transition API can be used to detect changes in the user's activity. Your app subscribes to a transition in activities of interest and the API notifies your app only when needed. This task shows how to use the Activity Recognition Transition API, also called the Transition API for short. We will create an app that would register for Activity transitions and list them on a Console activity when a transition occurs e.g. when a user starts or stops an activity like walking or running. In practice, there are a number of supported activities that your app can register for.

1. To use the Transition API in your app, you must declare a dependency to the Google Location and Activity Recognition API. To declare a dependency to the API, add a reference to the Google maven repository and add an implementation entry to com.google.android.gms:play-services-location:12.0.0 or higher to the dependencies section of your app build.gradle file. For more information, see Set Up Google Play Services

2. You must also specify the right permission by adding a <uses-permission> element in the app manifest:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.example.android.basicactivityrecognitiontransitionsample">

    <uses-permission
android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />

    ...

</manifest>
```

3. Follow the remaining steps as shown in the Activity Recognition Transition API Codelab tutorial to finish this task. If you manage to successfully build and run the app, you should be able to see the app activity with the cold start last activity transition reported by the device, as per Image 1 shown on the next page (this is the last activity that the device has transitioned). In practice, it could be any last known activity. Image 2 shows the activity when the user has started walking, and then became still again.

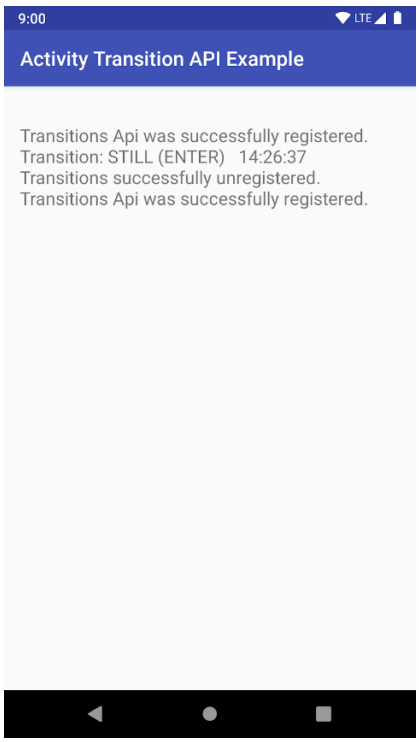


Image 1

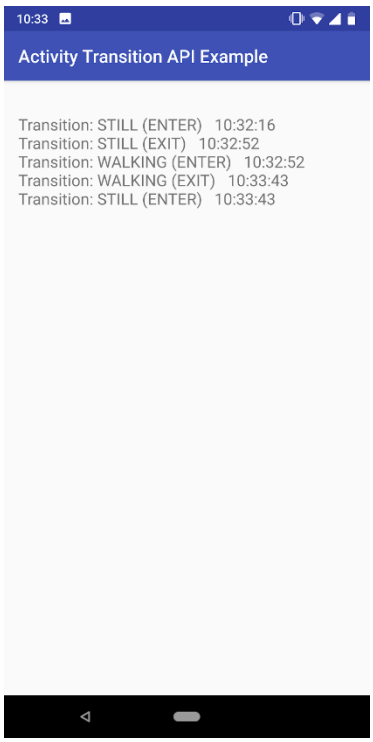


Image 2

For more information, please visit the following reference links :

1. Sample demonstrating how to set up SensorEventListeners for step detectors and step counters - https://github.com/android/sensors-samples
2. Creating a pedometer app that maintains the step history along with dates using the steps detector sensor - https://hub.packtpub.com/step-detector-and-step-counters-sensors/
3. Android Step Sensors Example - https://github.com/coomar2841/android-step-sensors
4. Accelerometer SensorEvent timestamp - https://stackoverflow.com/questions/5500765/accelerometer-sensorevent-timestamp
5. Detect when users start or end an activity - https://developer.android.com/guide/topics/location/transitions
6. Activity Recognition Transition API Codelab - https://codelabs.developers.google.com/codelabs/activity-recognition-transition/index.html