MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 9

Objectives:

- To understand Sensors
- To understand the working of Google Maps
- Practice Activities

OBJECTIVE 1: Understanding and using sensors

Most of the android devices have built-in sensors that measure motion, orientation, and various environmental condition. The android platform supports three broad categories of sensors.

- Motion Sensors
- Environmental sensors
- Position sensors

Sensor Framework

You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the android.hardware package and includes the following classes and interfaces:

SensorManager

You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

Sensor

You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

SensorEvent

The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

SensorEventListener

You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

Reference:
1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

Getting list of sensors supported

You can get a list of sensors supported by your device by calling the getSensorList method, which will return a list of sensors containing their name and version number and much more information. You can then iterate the list to get the information. Its syntax is given below –

```
sMgr = (SensorManager)this.getSystemService(SENSOR_SERVICE);
List<Sensor> list = sMgr.getSensorList(Sensor.TYPE_ALL);
for(Sensor sensor: list){
}
```

Sensor Manager

Android provides **SensorManager** and **Sensor** classes to use the sensors in our application. In order to use sensors, first thing you need to do is to instantiate the object of **SensorManager** class. It can be achieved as follows.

```
SensorManager sMgr;
sMgr = (SensorManager)this.getSystemService(SENSOR_SERVICE);
```

The next thing you need to do is to instantiate the object of Sensor class by calling the getDefaultSensor() method of the SensorManager class. Its syntax is given below –

```
Sensor light;
light = sMgr.getDefaultSensor(Sensor.TYPE_LIGHT);
```

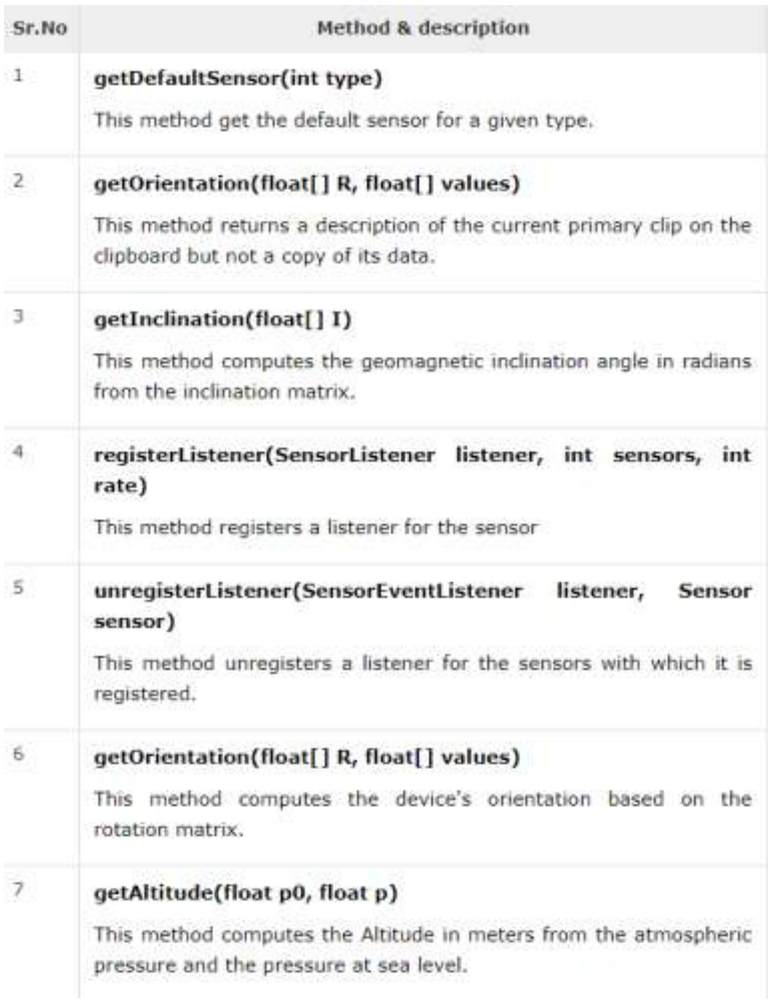
Once that sensor is declared , you need to register its listener and override two methods which are onAccuracyChanged and onSensorChanged. Its syntax is as follows –

```
sMgr.registerListener(this, light,SensorManager.SENSOR_DELAY_NORMAL);
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
public void onSensorChanged(SensorEvent event) {
}
```

Other methods of **SensorManager** class.

Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)



Sr.No	Method & description
1	**getDefaultSensor(int type)** This method get the default sensor for a given type.
2	**getOrientation(float[] R, float[] values)** This method returns a description of the current primary clip on the clipboard but not a copy of its data.
3	**getInclination(float[] I)** This method computes the geomagnetic inclination angle in radians from the inclination matrix.
4	**registerListener(SensorListener listener, int sensors, int rate)** This method registers a listener for the sensor
5	**unregisterListener(SensorEventListener listener, Sensor sensor)** This method unregisters a listener for the sensors with which it is registered.
6	**getOrientation(float[] R, float[] values)** This method computes the device's orientation based on the rotation matrix.
7	**getAltitude(float p0, float p)** This method computes the Altitude in meters from the atmospheric pressure and the pressure at sea level.

Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

OBJECTIVE 2: Understanding and using sensors

Google provides via Google play a library for using Google Maps in your application. The following description is based on the Google Maps Android API v2 which provides significant improvements to the older API version.

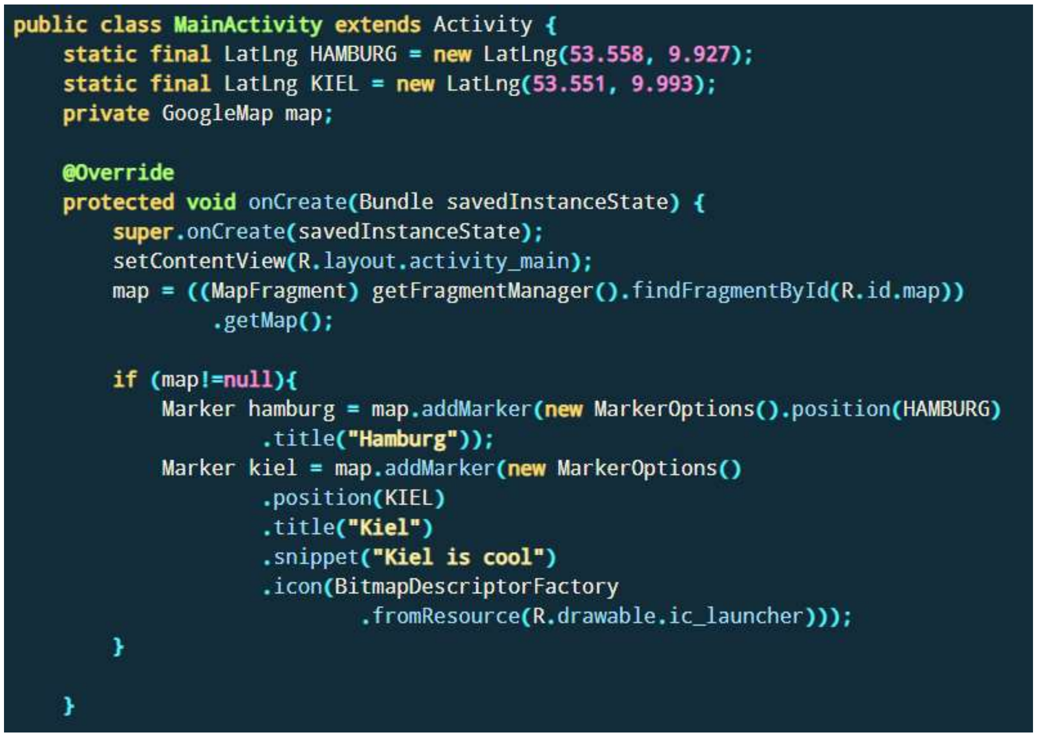
The library provides the *com.google.android.gms.maps.MapFragment* class and the MapView class for displaying the map component.

Map Fragment

The MapFragment class extends the Fragment class and provides the life cycle management and the services for displaying a GoogleMap widget. GoogleMap is the class which shows the map. The MapFragment has the getMap() method to access this class

Markers

You can create markers on the map via the Marker class. This class can be highly customized.



```java
public class MainActivity extends Activity {
    static final LatLng HAMBURG = new LatLng(53.558, 9.927);
    static final LatLng KIEL = new LatLng(53.551, 9.993);
    private GoogleMap map;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        map = ((MapFragment) getFragmentManager().findFragmentById(R.id.map))
                .getMap();

        if (map!=null){
            Marker hamburg = map.addMarker(new MarkerOptions().position(HAMBURG)
                    .title("Hamburg"));
            Marker kiel = map.addMarker(new MarkerOptions()
                    .position(KIEL)
                    .title("Kiel")
                    .snippet("Kiel is cool")
                    .icon(BitmapDescriptorFactory
                            .fromResource(R.drawable.ic_launcher)));
        }

    }
```

Google Maps API Key

To use Google Maps you need to create a valid Google Maps API key. The key is free, you can use it with any of your applications that call the Maps API. This key supports an unlimited number of users.

You get this key via the Google APIs Console. You have to provide your application signature key and the application package name.

Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

This is based on the key with which you sign your Android application during deployment. During development the Android build system, automatically creates and uses a *debug key*.

Install Google Play Services

Ensure you installed the Google Play Service.

Create New Google Maps Activity

Create a new Android project and use the *Google Maps Activity* template.

Validate XML File

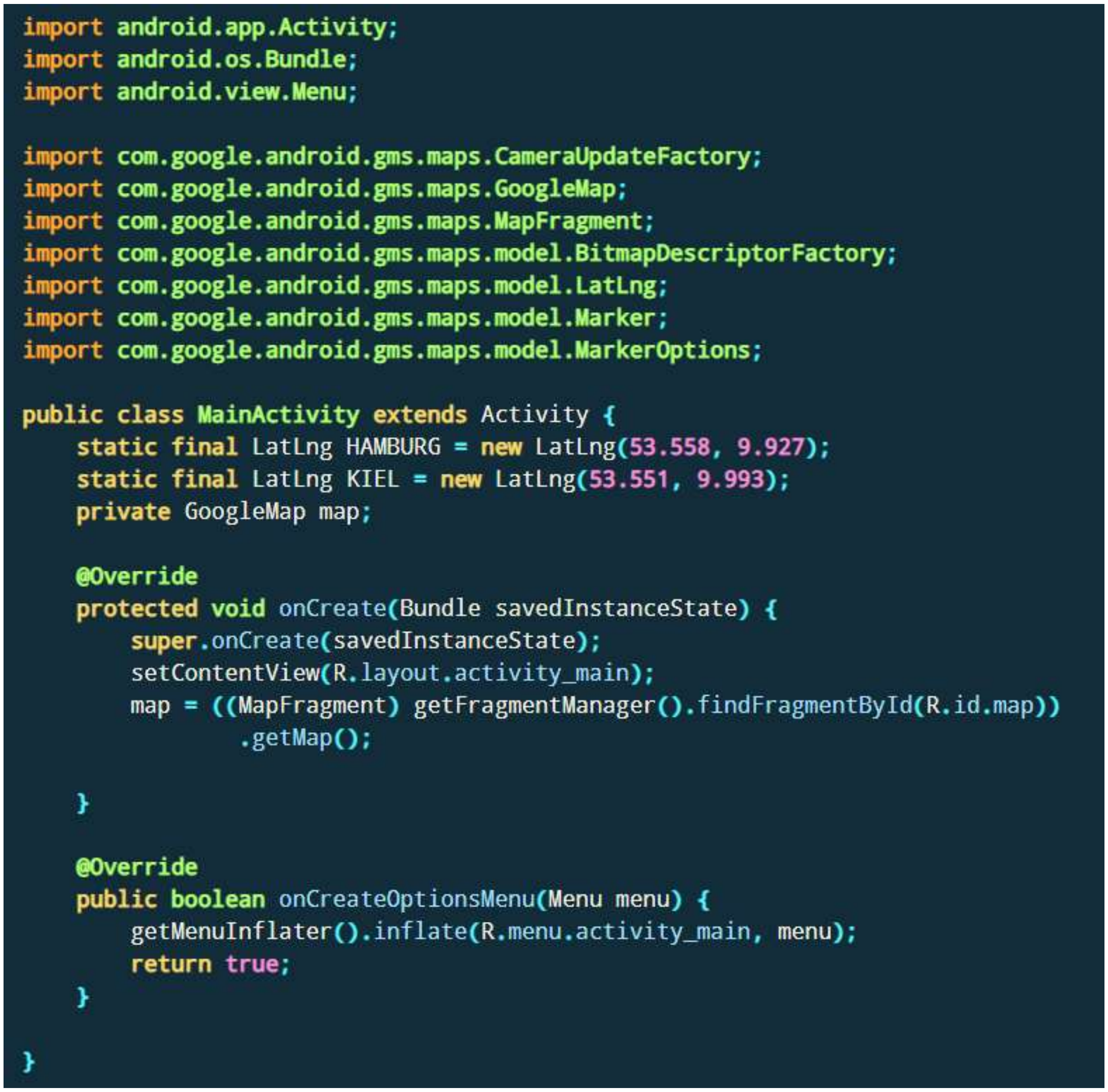
Check the manifest file for the added permissions by this template.

```
    <!--
         The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
         Google Maps Android API v2, but you must specify either coarse or fine
         location permissions for the 'MyLocation' functionality.
    -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!--
             The API key for Google Maps-based APIs is defined as a string resource.
             (See the file "res/values/google_maps_api.xml").
             Note that the API key is linked to the encryption key used to sign the APK.
             You need a different API key for each encryption key, including the release key
that is used to
             sign the APK for publishing.
             You can define the keys for the debug and release targets in src/debug/ and
src/release/.
        -->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".MapsActivity"
            android:label="@string/title_activity_maps">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

Sample Code



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

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

public class MainActivity extends Activity {
    static final LatLng HAMBURG = new LatLng(53.558, 9.927);
    static final LatLng KIEL = new LatLng(53.551, 9.993);
    private GoogleMap map;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        map = ((MapFragment) getFragmentManager().findFragmentById(R.id.map))
                .getMap();

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

Activities Section

ACTIVITY 1: Detecting a Shake/Jerk Gesture

Construct an app that toggles a lightbulb on and off when the user shakes or jerks their device.

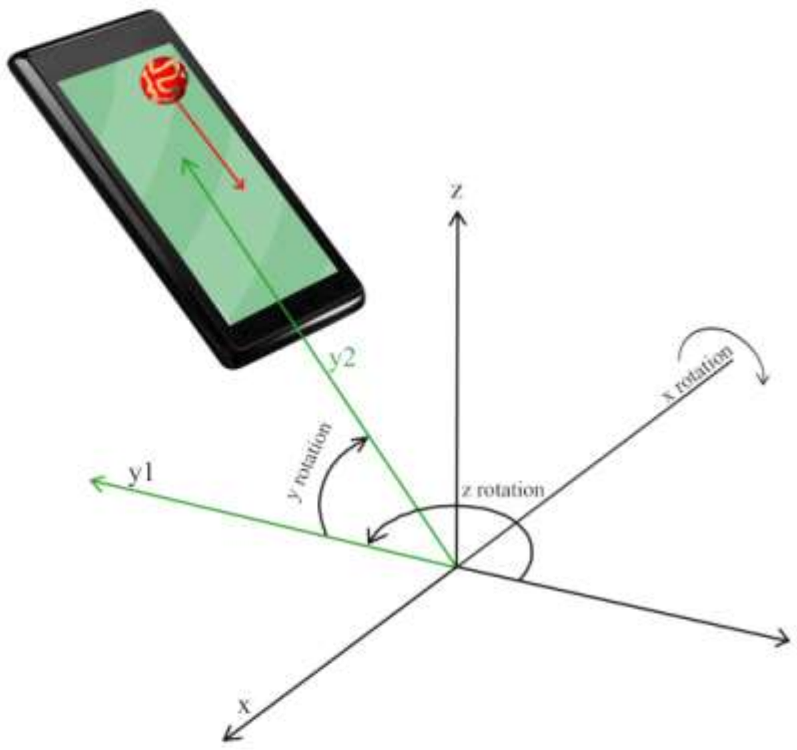
Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

Activity 2: Roaming Ball using the Accelerometer.

1. Construct an application containing a ball and three TextViews.

2. When the application first launches, the user is presented with the ball positioned at a specific x, y location on the screen.
3. The TextViews will be used to display the x, y, and z-axis readings from the accelerometer.
4. As the user tilts and rotates the device, the text fields will be updated to show the current accelerometer readings.
5. In addition, the ball will move in accordance with the tilt and roll of the device, as shown in the figure.
6. The ball will not be allowed to roll off screen, and will be deterred by the virtual boundaries of the screen.



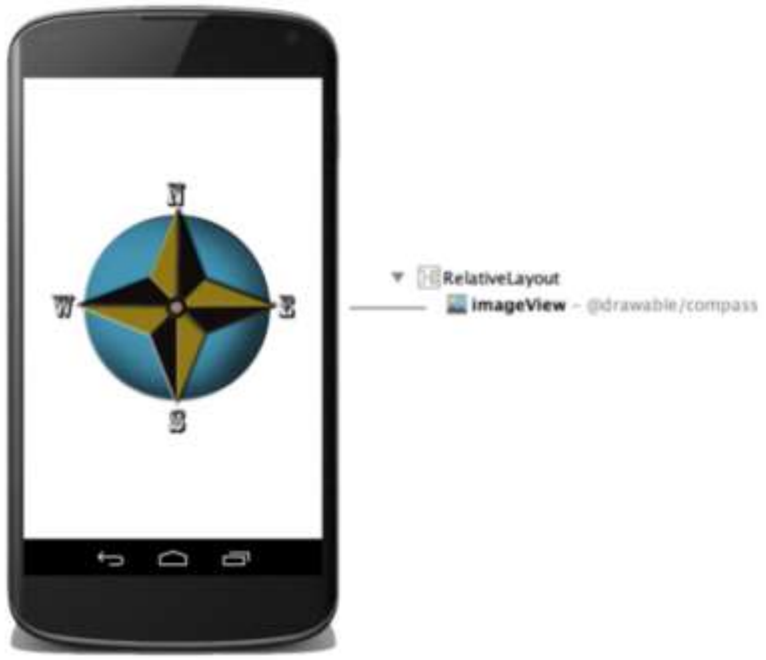
Reference:
1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

Activity 3: Geomagnetic Rotation - Compass

Implement a geomagnetic rotation vector for measuring the orientation of an Android device relative to a north, south, east, and west direction.

A geomagnetic rotation vector sensor is a composite sensor that will be defined by the accelerometer in tandem with the magnetometer. The application may need to specify each feature in a separate a *uses-feature* element within the Manifest file.

1. Indicate that motion readings will occur from an accelerometer and directional readings from a magnetometer (compass) on the device.
2. Set a portrait screen orientation for the application.
3. Use your own art creation for the compass image.



Reference:

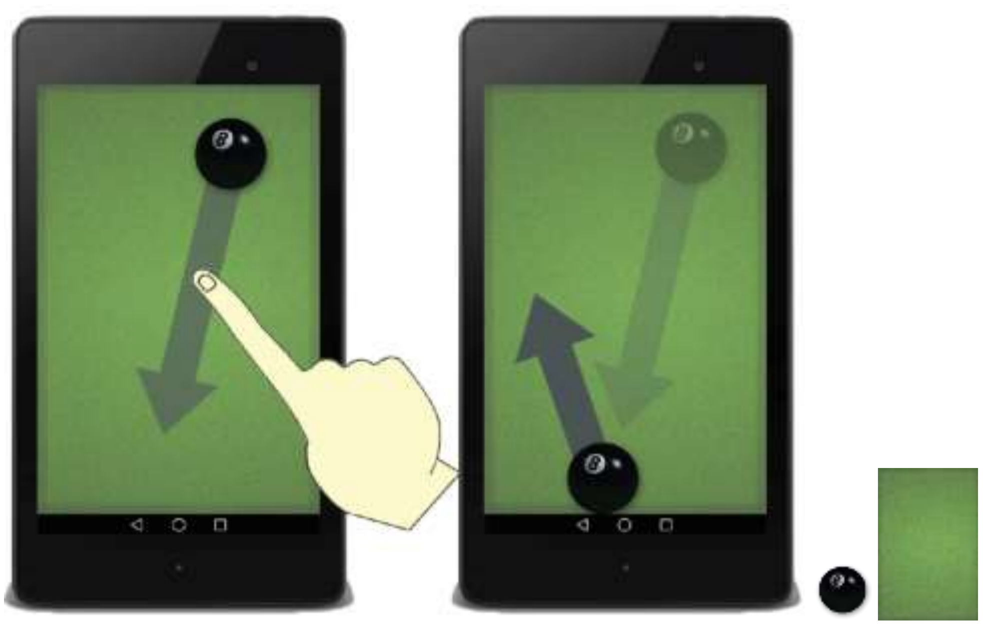
1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

Activity 4: Fling a Billiard Ball

This exercise allows users to fling a billiard ball across the screen in any direction. When the billiard ball hits the edges of the screen, it reverses direction as if it is bouncing off a wall. Friction should be applied to movement so that the ball eventually comes to a halt.

Research
GestureDetector and OnGestureListener.

Question 1: How is the velocity of a fling detected?
Question 2: Can the velocity of a fling be detected across just the x-axis?
Question 3: Identify the callback methods used to detect gestures.



NOTE: Rename *billiardball* so that it reads *ball*.

Create the following files:
ball_layout.xml
This XML file is an ImageView layout that holds the billiardball.png.

Reference:
1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

main_layout.xml
This XML file is an main activity layout. Specify the root View as a *FrameLayout*.

Set the background to *table.png*.
Set the id to *frameLayout*.

NOTE: The FrameLayout uses a coordinate system with 0,0 in the center.

Ball.java
```
    public class Ball {
      public int mX;
      public int mY;
      public int mRadius;
      public double mVelocityX;
      public double mVelocityY;
      public int left, right, top, bottom;

      public  Ball(int  mX,  int  mY,  int  mRadius,  double  mVelocityX,  double
mVelocityY){
        this.mX = mX;
        this.mY = mY;
        this.mRadius = mRadius;
        this.mVelocityX = mVelocityX;
        this.mVelocityY = mVelocityY;
      }

      public void move(){
        mX += mVelocityX;
        mY += mVelocityY;

        //COLLISIONS - REVERSE DIRECTIONS
        if (mX < left){
          mX = left;
          mVelocityX *= -1;
        }
        else if (mX > right){
          mX = right;
          mVelocityX *= -1;
        }
        if (mY < top){
          mY = top;
          mVelocityY *= -1;
        }
        else if (mY > bottom){
          mY = bottom;
          mVelocityY *= -1;
        }

        //APPLY FRICTION TO THE VELOCITY
        mVelocityX *= .93;
        mVelocityY *= .93;
```
Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

```
        if (Math.abs(mVelocityX) < 1) mVelocityX = 0;
        if (Math.abs(mVelocityY) < 1) mVelocityY = 0;

      }
      public void setCollisionBoundaries( int windowWidth, int windowHeight){
        left = -windowWidth / 2 + mRadius;
        right = windowWidth / 2 - mRadius;
        top = -windowHeight / 2 + mRadius;
        bottom = windowHeight / 2 - mRadius;
      }
    }
```

MainActivity.java

```
    import android.content.Context;
    import android.os.Bundle;
    import android.os.Handler;
    import android.support.v7.app.AppCompatActivity;
    import android.util.DisplayMetrics;
    import android.view.GestureDetector;
    import android.view.LayoutInflater;
    import android.view.MotionEvent;
    import android.view.Window;
    import android.view.WindowManager;
    import android.widget.FrameLayout;
    import android.widget.ImageView;

    public   class   MainActivity   extends   AppCompatActivity   implements
GestureDetector.OnGestureListener {

      private GestureDetector aGesture;

      private FrameLayout mainLayout;
      private Thread backgroundThread;
      private ImageView ballImageView;
      private Ball mBall;

      private int left, right, top, bottom;

      @Override
      protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //TASK 1: SET THE LAYOUT AND THE WINDOW ELEMENTS
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getSupportActionBar().hide();
        setContentView(R.layout.main_layout);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);
        mainLayout = (FrameLayout) findViewById(R.id.frameLayout);

        //TASK 2: CREATE A BILLIARD BALL
        buildBilliardBall();
```

Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

```
        //TASK 3: CREATE A GESTURE DETECTOR
        aGesture = new GestureDetector(this, this);

        //TASK 4: CREATE THE BACKGROUND THREAD
        backgroundThread = new Thread(calculateAction);
        backgroundThread.start();
      }

      private void buildBilliardBall() {
        //TASK 1: THE CREATE THE DATA MODEL FOR THE BILLIARD BALL
        mBall = new Ball(0, 0, 150, 0, 0);
        DisplayMetrics metrics = new DisplayMetrics();
        getWindowManager().getDefaultDisplay().getMetrics(metrics);
        mBall.setCollisionBoundaries(metrics.widthPixels,
metrics.heightPixels);

        //TASK 2: CREATE A LAYOUT INFLATER TO ADD THE BILLIARD BALL IMAGEVIEW
TO THE LAYOUT
        LayoutInflater layoutInflater;
        layoutInflater                    =                   (LayoutInflater)
getSystemService(Context.LAYOUT_INFLATER_SERVICE);

        //TASK 3: ADD THE BALL TO THE LAYOUT
        ballImageView                        =                       (ImageView)
layoutInflater.inflate(R.layout.ball_layout, null);
        ballImageView.setScaleX((float) .3);
        ballImageView.setScaleY((float) .3);
        ballImageView.setX((float) mBall.mX);
        ballImageView.setY((float) mBall.mY);
        mainLayout.addView(ballImageView, 0);
      }


      //**************** RUNNABLE *********************
      private Runnable calculateAction = new Runnable() {
        private static final int DELAY = 50;
        public void run() {
          try {
            while (true) {
              mBall.move();
              Thread.sleep(DELAY);
              threadHandler.sendEmptyMessage(0);
            }
          } catch (InterruptedException e) {
          }
        }
      };
      //****** HANDLER FOR UPDATING THE IMAGEVIEW CONTAINING THE BILLIARD
BALL******
      public Handler threadHandler = new Handler() {
        public void handleMessage(android.os.Message msg) {
          //UPDATE BALL LOCATION
          ballImageView.setX((float) mBall.mX);
          ballImageView.setY((float) mBall.mY);
        }
```

Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)

```
      };



      //*********************TOUCH GESTURES************************
      @Override
      public boolean onTouchEvent(MotionEvent event) {
        return aGesture.onTouchEvent(event);
      }
      @Override
      public  boolean  onFling(MotionEvent  event1,  MotionEvent  event2,  float 
velocityX, float velocityY) {
        final float ADJUST = 0.025f;
        mBall.mVelocityX = (int) velocityX * ADJUST;
        mBall.mVelocityY = (int) velocityY * ADJUST;
        return true;
      }
      @Override
      public void onLongPress(MotionEvent event) {}
      @Override
      public void onShowPress(MotionEvent event) {}
      @Override
      public boolean onDown(MotionEvent event) {return false;}
      @Override
      public boolean onScroll(MotionEvent e1, MotionEvent e2,float distanceX, 
float distanceY) {return false;}
      @Override
      public boolean onSingleTapUp(MotionEvent event) {return false;}
    }
```

Reference:

1. https://source.android.com/devices/sensors/sensor-types (all type of sensors)
2. https://developer.android.com/guide/topics/sensors/sensors_motion (proper usage)