NAME: SAKSHI VINAYAK KITTURKAR
CSU ID: 2860273
HOMEWORK 11

Build the sensor listing app using the code provided. In the homework submission, please include the screenshot for all the sensors that are supported by your phone. Please also show at least three sensors' capabilities and their values.

= I have created app for the sensor listing app where we can see the sensors capabilities and their values.

In layout: MainActivity.java this code creates an Android app with a main activity that displays a list of available sensors. When the user clicks on a sensor, it launches another activity to display the capabilities of the selected sensor.

In the onCreate method:
The layout is set using setContentView. SensorManager is obtained using the getSystemService method. mSensorsList is populated with all available sensors on the device. The ListView, adapter, and click listener are set up.

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

    mSensorManager = (SensorManager) this.getSystemService(Context.SENSOR_SERVICE);
    mSensorsList = mSensorManager.getSensorList(Sensor.TYPE_ALL);
    mSensorListView = findViewById(R.id.session_list);
    mListAdapter = new ListAdapter();
    mSensorListView.setAdapter(mListAdapter);
    mSensorListView.setOnItemClickListener(this);
}
```

This method is triggered when an item in the ListView is clicked:
It creates an Intent to start another activity (SensorCapabilityActivity). The sensor type is added as an extra to the intent. The new activity is started.

```
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    Intent intent = new Intent(getApplicationContext(), SensorCapabilityActivity.class);
    intent.putExtra(getResources().getResourceName(R.string.sensor_type),
mSensorsList.get(position).getType());
    startActivity(intent);
```

```
}
```

This is an inner class (ListAdapter) that extends BaseAdapter to handle the data for the ListView: getCount: Returns the number of items in the list. getItem: Returns the name of the sensor at a given position. getItemId: Returns the position of the item. getView: Inflates the layout for each item in the list and sets the sensor name.

```
private class ListAdapter extends BaseAdapter {
    private TextView mSensorName;

    @Override
    public int getCount() {
        return mSensorsList.size();
    }

    @Override
    public Object getItem(int position) {
        return mSensorsList.get(position).getName();
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            convertView = getLayoutInflater().inflate(R.layout.list_rows, parent, false);
        }
        mSensorName = convertView.findViewById(R.id.sensor_name);
        mSensorName.setText(mSensorsList.get(position).getName());
        return convertView;
    }
}
```

In layout: SensorCapabilityActivity this activity receives the sensor type from the previous activity, retrieves information about the corresponding sensor, and displays its capabilities in a UI. Additionally, there's a button (onClickSensorValues) that, when clicked, starts another activity to display the real-time values of the sensor (SensorValuesActivity).

In the onCreate method:
The layout is set using setContentView. The intent is used to retrieve the sensor type passed from the previous activity. The SensorManager is obtained. The default sensor of the specified type is obtained. TextViews are initialized and populated with information from the sensor.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_sensor_capability);
   Intent intent = getIntent();
   mSensorType = intent.getIntExtra(getResources().getResourceName(R.string.sensor_type), 0);
   mSensorManager = (SensorManager) this.getSystemService(Context.SENSOR_SERVICE);
   mSensor = mSensorManager.getDefaultSensor(mSensorType);
   // ... (initialize TextViews)
   // Set TextViews with sensor information
}
```

This method is triggered when a button with an onClick attribute is clicked. It creates an intent to start another activity (SensorValuesActivity) and passes the sensor type as an extra.

```
public void onClickSensorValues(View v) {
   Intent intent = new Intent(getApplicationContext(), SensorValuesActivity.class);
   intent.putExtra(getResources().getResourceName(R.string.sensor_type), mSensorType);
   startActivity(intent);
}
```

In layout: SensorValuesActivity This activity displays real-time sensor values obtained from the device's sensor specified by the sensor type passed from the previous activity. The values are updated in TextViews, and the activity registers/unregisters itself as a listener for sensor events based on its lifecycle. The SensorValuesActivity can be used to monitor and visualize the dynamic behavior of the selected sensor.

In the onCreate method:
The layout is set using setContentView. The intent is used to retrieve the sensor type passed from the previous activity. The SensorManager is obtained. The default sensor of the specified type is obtained. TextViews are initialized to display sensor values.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_sensor_values);
   Intent intent = getIntent();
   mSensorType = intent.getIntExtra(getResources().getResourceName(R.string.sensor_type), 0);
   mSensorManager = (SensorManager) this.getSystemService(Context.SENSOR_SERVICE);
   mSensor = mSensorManager.getDefaultSensor(mSensorType);
   // ... (initialize TextViews)
}
```

onResume: Registers the activity as a listener for sensor events when the activity is in the foreground.
onPause: Unregisters the activity as a listener when the activity is in the background.

```
@Override
protected void onResume() {
   super.onResume();
   mSensorManager.registerListener(this, mSensor, SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onPause() {
   super.onPause();
   mSensorManager.unregisterListener(this);
}
```

onAccuracyChanged: Callback method for changes in sensor accuracy (not used in this example).
onSensorChanged: Callback method invoked when sensor values change. Updates TextViews with real-time sensor values.

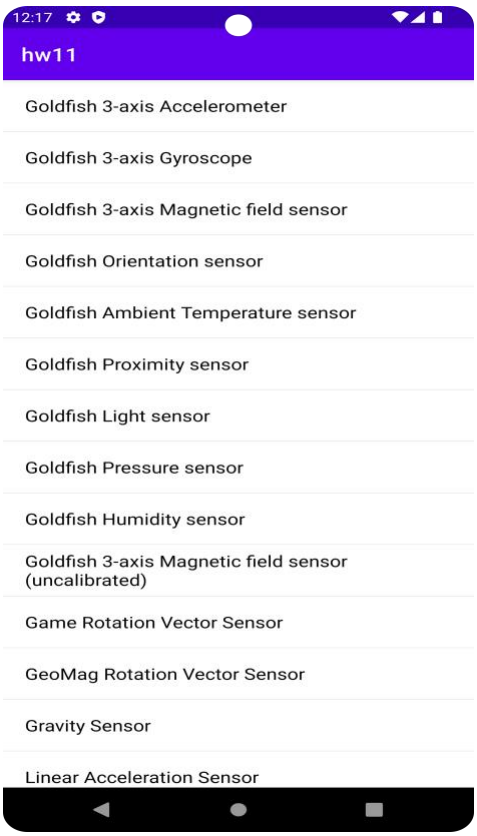
```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}

@Override
public void onSensorChanged(SensorEvent event) {
   // ... (update TextViews with sensor values)
}
```

In layout: AndroidMainfest.xml defines the structure of the Android application, including its activities, permissions, and other configurations. It indicates that MainActivity is the main launcher activity, while SensorValuesActivity and SensorCapabilityActivity are not intended for direct access by other applications. The manifest also includes some additional application settings related to backup, RTL support, and theme.

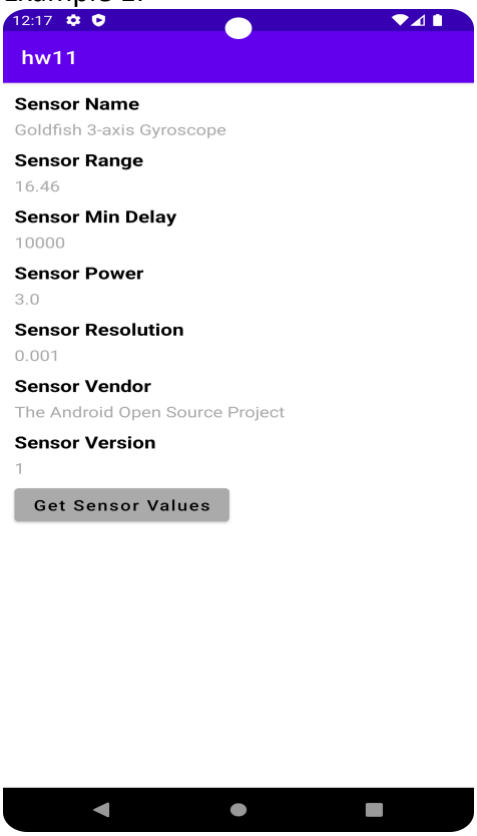
<activity> tags define the activities of the application.
android:name: Specifies the fully qualified name of the activity class.
android:exported: Indicates whether the activity is accessible by other applications. In this case, activities SensorValuesActivity and SensorCapabilityActivity are not exported (android:exported="false"), while MainActivity is exported (android:exported="true").
The <intent-filter> within the MainActivity tag specifies that this activity is the main entry point of the application (MAIN action and LAUNCHER category).

```
<activity
   android:name=".SensorValuesActivity"
   android:exported="false" />
<activity
   android:name=".SensorCapabilityActivity"
   android:exported="false" />
<activity
   android:name=".MainActivity"
   android:exported="true">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

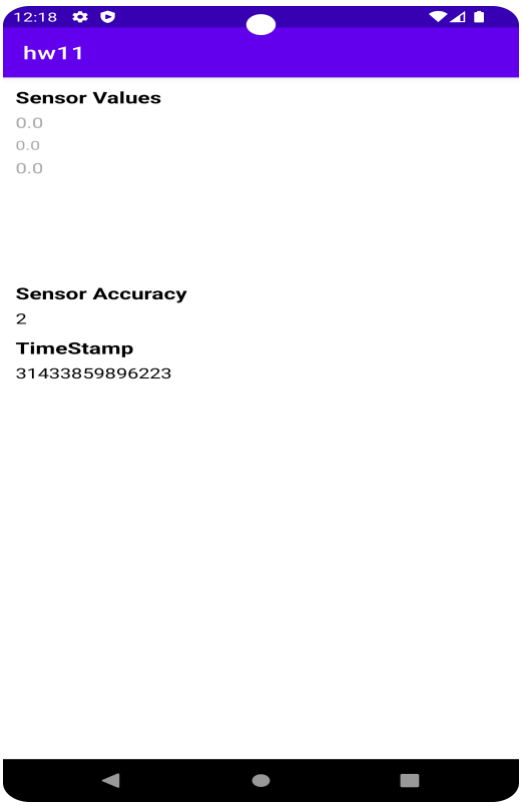
List of all the sensors:



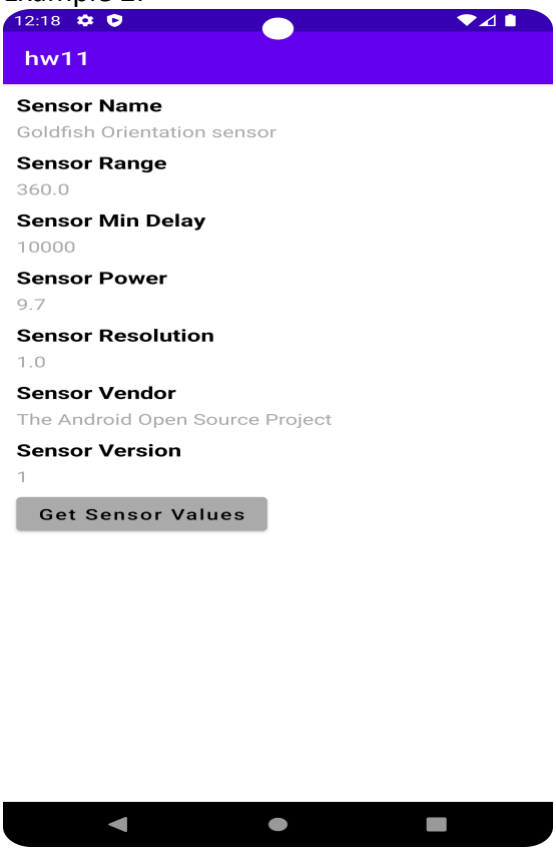
Example 1:



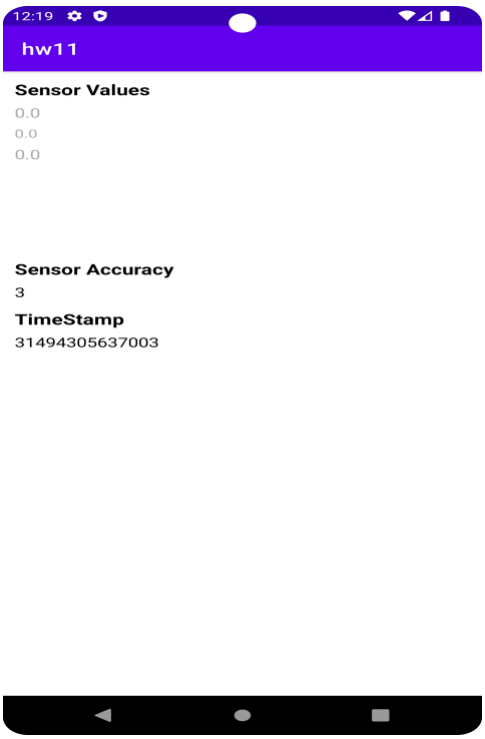
Sensor value:



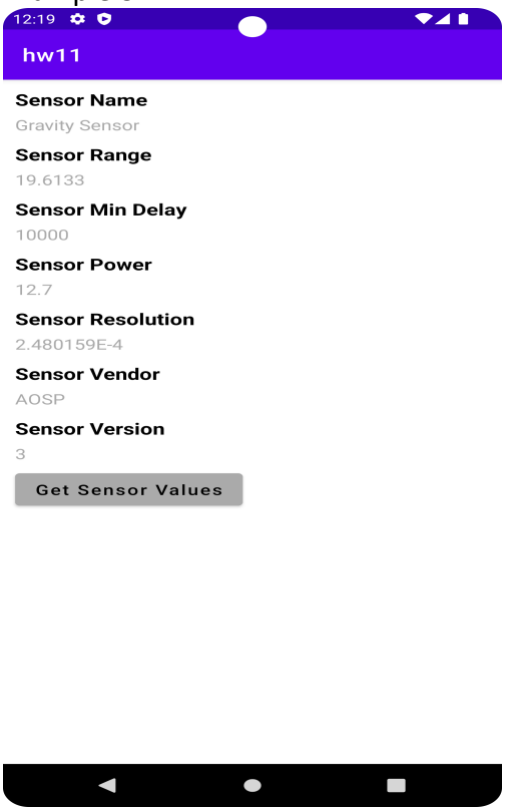
Example 2:



Sensor value:



Example 3:



Sensor value: