Practical related Questions

1. List the best practices for accessing and using sensors

- Before using the sensor coordinate system, confirm the default orientation mode of the device and check for the orientation of the x and y axes.
- Check the availability, range, minimum delay, reporting modes, and resolution of the sensor before using it.
- Before selecting the sampling period of any sensor, check for its power consumption. Also, keep your application precision and accuracy needs in mind before deciding the sampling period. It's recommended that you select one of the constants given by the operating system.
- Do not block or do heavy processing on the OnSensorChanged() method. Your app might miss callbacks or go into ANR (Application Not Responding) mode. The app might even crash in the worst cases if this callback is blocked.
- Every registration of the event listener should be paired with the un-registration of the same listener. This should be done at the right time and place. (More on this, in the next chapter).
- Avoid using deprecated sensors and any of the deprecated APIs.
- Never write any kind of application logic based on the delay between the sensor events. Always use the timestamp from the sensor event to do your time-related calculations.
- If some sensors are mandatory for your application to function, then use the uses-feature filter in the Manifest.xml file and change the required value to true.
- Check your application and its sensor behavior on more than one device, as the sensor values and range may vary with different devices.

2. Differentiate between Sensors class and sensor Manager class

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.

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.

Exercise:-

XML:-

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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/layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textview"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:gravity="center"
        android:text="Shake to change background color!"
        android:textSize="20sp"
         />

    <ScrollView

        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <!--Text View that shall display the sensor information list-->
        <TextView
            android:id="@+id/tv"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true" />

    </ScrollView>

</LinearLayout>
```

Java:-

```
package com.example.practical22;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.graphics.Color;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
```

```
import java.util.List;

public class MainActivity extends AppCompatActivity implements
SensorEventListener {
    private SensorManager sensorManager;
    TextView view;
    private boolean isColor = false;
    private long lastUpdate;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        view=(TextView) findViewById(R.id.textview);
//        view.setBackgroundColor(Color.GREEN);

        sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

        List deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);

        // Text View that shall display this list
        TextView tv =(TextView) findViewById(R.id.tv);

        // Converting List to String and displaying
        // every sensor and its information on a new line
        for (Object sensors : deviceSensors) {
            tv.append(sensors.toString() + "\n\n");
        }


    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        if(event.sensor.getType()==Sensor.TYPE_ACCELEROMETER){
            getAccelerometer(event);
//            textview.setBackgroundColor(Color.RED);
        }
    }

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

        float accelationSquareRoot = (x * x + y * y + z * z)
                / (SensorManager.GRAVITY_EARTH *
SensorManager.GRAVITY_EARTH);

        long actualTime = System.currentTimeMillis();

        if (accelationSquareRoot >= 2) //it will be executed if you shuffle
        {
```

```
            if (actualTime - lastUpdate < 200) {
                return;
            }
            lastUpdate = actualTime;//updating lastUpdate for next shuffle
            if (isColor) {
                view.setBackgroundColor(Color.GREEN);

            } else {
                view.setBackgroundColor(Color.RED);
            }
            isColor = !isColor;
        }
    }

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

    }
    protected void onResume() {
        super.onResume();
        // register this class as a listener for the orientation and
        // accelerometer sensors

sensorManager.registerListener(this,sensorManager.getDefaultSensor(Sensor.TYP
E_ACCELEROMETER),
                SensorManager.SENSOR_DELAY_NORMAL);
    }

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


}
```