Sensors

#1. What is Sensor?

Sensor → device → detect & respond → to some type of i/p from env.

Sensor ek aisa device hota hai jo environment ke input ko detect & respond krta hai.

Input type for sensor to detect & respond :- (1) Physical (2) Non physical

Physical → 1. Light 2. heat 3. motion 4. moisture 5. pressure 6. gyroscop 7. accelometer 8. barometer, & other environmental phenomena.

Non physical → 1. Camera 2. Finger print sensor 3. micro phone 4. Touch screen

- Have their own reporting mechanism
- Android sensor → provide low bandwidth data [

Output → Generally, a signal that is converted to a human readable display at sensor locatn or transmitted electronically over a network for further processing.

O/p aise signals hote hai jo na human readable hote hai at sensor locatn (device) ya fir un signals ko furth processing ke liye electronically trasmit krte hai, network ke dwara (इधर)

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device. For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing. Likewise, a weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

#2. Sensors Supported by Android platform.

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical	Determining device position.

TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.

ORIENTATION		axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the `getRotationMatrix()` method.	position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

#3. Sensor Framework

Above mentioned sensors can be accessed & acquire raw data by using Android sensor framework. Sensor framework → part → android.hardware package. Sensor framework includes following classes & interface:-

Sensor Manager:-

- It is class of sensor framework
- Create → instance of sensor services
- This class → provide → various method for accessing & listening sensor, registering & unregistering sensor event listeners, & acquiring orientation info.
- → also provide → several sensor constants → used to report sensor accuracy, set data aquisit^n (act of acquiring/takeovers) rate & calibrate sensor.

Sensor:-

- It is also class of sensor framework
- Used to create an instance of specific sensor.
- This class → provide → various method → that let you determine a sensor's capabilities.

Sensor Event

- System → use → this class → to create → sensor event object
- Sensor event object → provide info about sensor event.
- Info it provides:- raw sensor data, type of sensor that generated the event, accuracy of the data, & timestamp for event.

Sensor Event Listner:-

- This is an interface.
- It create two callback methods that recieve notificat^n (sensor event) when sensor value changes or when sensor accuracy changes.

In typical applicat^n → can use these sensor related APIs to perform

2 basic task :-

1) Identifying sensor & sensor capabilities :- It is useful if your app has features that rely on specific sensor types or capabilities.
Basically, app ke aise features jo kisi aise sensor pe work krte jo device (phone/tab) me hai hi nahi, aise features ko disable kr sakte hai.

2) Monitor sensor event :- ◦ How you acquire raw sensor data.
◦ Sensor event → occur every time → when sensor detect changes in parameter it is measuring.
◦ Provide 4 info :- 1. name of sensor that triggered the event.
2. timestamp of event
3. accuracy of event
4. raw sensor data that triggered the event.

Light Sensor

Raw data that we acquire from light, pressure & temp. req. no caliberatn, filteratn or modificatn. This make them easiest to use

Basic steps to acquire from these sensors are :-

1) Create an instance of SensorManager class → use to get an instance of physical sensor.
2) Register a sensor listener in the onResume() method.
3) Start handling incoming sensor data in onSensorChanged() callback method.

Steps to create light sensor application :-

#1. Android Manifest.xml

Since we are using sensor so we have to add "features" here; for light → add name = "light sensor" & required = "true"

```
4    <!--    since we are using sensor so we have to add features here for light add light sensor-> make is req true -->
5        <uses-feature
```

```
        android:name="android.hardware.sensor.light"
        android:required="true" />
```

#2. Make an empty activity for "Splash Screen"

- Make this activity as ☑ lancher activity { it'll be reflected in manifest file }

#3. activity_splash_screen.xml

1. Make linear layout
2. Add logo img. Remember splash screen should be visibily small, do not make it much bigger
3. give it id (we gonna need it in .kt file)

activity_first_page.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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FirstPage"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_gravity="center"
        android:layout_marginTop="250dp"
        app:srcCompat="@drawable/splashscreen_logo"
        tools:layout_editor_absoluteY="305dp" />
</LinearLayout>
```

#4. SplashScreen.kt

- firstly we'll hide action bar => supportActionBar?.hide()
- Now write Handler function in this way =>

```
Handler(Looper.getMainLooper()).postDelayed({
    val i= Intent(this, MainActivity::class.java)
    startActivity(i)
    finish()
},3000)
```

FirstPage.kt

```
package com.example.lightsensor

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.Handler
import android.os.Looper

class FirstPage : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first_page)
        supportActionBar?.hide() //to hide action bar where you get to see back arrow or left to
        //to show splash screen
        Handler(Looper.getMainLooper()).postDelayed({
            val i= Intent(this, MainActivity::class.java)
            startActivity(i)
            finish()
        },3000)
    }
}
```

We can make splash screen w/o looper

```
Handler().postDelayed({ val i= Intent(this, MainActivity::class.java)
    startActivity(i) // to start intent
    finish() // if we'll not write this then splash screen will be again shown to user
},3000)
```

↓ to stop screen for this much time

#5. activity_main.xml

- Here, 2 images have been adeded.

→ One image which is bright will be shown when light is not dim or when sensor is sensing light unit more than 30 unit
Id is names as lights_on_img

→ Other image which is dark will be

activity_main.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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
    android:background="#B88267"
    android:id="@+id/backgroundMain">

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

        <ImageView
            android:id="@+id/lighton_img"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
```

shown when light will get dim or when sensor of device is covered. & sensing value below 30 unit. Id is names as lights off_img

```
22          android:layout_marginStart="0dp"
23          android:layout_marginTop="43dp"
24          app:srcCompat="@drawable/lights_on" />
25      <ImageView
26          android:id="@+id/lightoff_img"
27          android:layout_width="wrap_content"
28          android:layout_height="wrap_content"
29          android:layout_alignParentStart="true"
30          android:layout_alignParentTop="true"
31          android:layout_marginStart="0dp"
32          android:layout_marginTop="43dp"
33          app:srcCompat="@drawable/lights_off" />
34  </RelativeLayout>
35 </LinearLayout>
```

#6. MainActivity.kt

1. Firstly make two variables inside MainActivity class
   (i) one variable will store type of sensor (what type of sensor we are using). {L 21}
   (ii) another variable to get an instance (event) of sensor services, and use that to get an instance of particular type. {L 22}
   We write ?=null to avoid null value error.

```
19  class MainActivity : AppCompatActivity(), SensorEventListener {
20
21      var sensor: Sensor?=null //1st var is of ->type variable
22      var sensorManager:SensorManager?=null //2nd var is of sensor manager type
```

2. Since we'll have to add functionality on the images we added on .xml file so making variables {L 24-L 26}

```
24      lateinit var imglightOn:ImageView
25      lateinit var imglightOff:ImageView
26      lateinit var background:LinearLayout
```

3. Now we'll initialize them under onCreate method.

```
29      override fun onCreate(savedInstanceState: Bundle?) {
30          super.onCreate(savedInstanceState)
31          setContentView(R.layout.activity_main)
32
33          imglightOn=findViewById(R.id.lighton_img)
34          imglightOff = findViewById(R.id.lightoff_img)
35          background=findViewById(R.id.backgroundMain)
```

4. We made imagelight visibility → invisible so that we can see changes happening.

```
36          imglightOn.visibility=View.INVISIBLE //hiding image at first, so that we'll know if app is working fine or not
```

5. Initializing sensorManager & sensor var inside onCreate method

```
37          //initializing sensorManager var and sensor var
38          sensorManager= getSystemService(Context.SENSOR_SERVICE) as SensorManager
39          sensor= sensorManager!!.getDefaultSensor(Sensor.TYPE_LIGHT)
40
41      }
```

6. Extend SensorEventListener in MainActivity Class & the click (Alt+Enter) to add onSensorChanged() & onAccuracyChanged()

```
18  // 1step: you have to extend sensor event listner, then implement members
19  class MainActivity : AppCompatActivity(), SensorEventListener {
```

7. onSensorChanged() extend its event with SensorEvent?, this will help record/look into changes happening.
   - From sensor we get value in array format
8. Now we'll write code in try catch to avoid app

crashing issue.

```
try {
    if (event != null)
        Log.d (TAG, "onSensorChanged: + event.value [0])
```

MainActivity.kt

```kotlin
package com.example.lightsensor

import android.content.ContentValues.TAG
import android.content.Context
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import android.media.Image
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.ImageView
import android.widget.LinearLayout
import androidx.constraintlayout.widget.ConstraintLayout
import java.io.IOException
// 1step: you have to extend sensor event listner, then implement members
class MainActivity : AppCompatActivity(), SensorEventListener {

    var sensor: Sensor?=null //1st var is of ->type variable
    var sensorManager:SensorManager?=null //2nd var is of sensor manager type
    //we have used following two in activity layout so to use them we have declared them
    lateinit var imglightOn:ImageView
    lateinit var imglightOff:ImageView
    lateinit var background:LinearLayout


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        imglightOn=findViewById(R.id.lighton_img)
        imglightOff = findViewById(R.id.lightoff_img)
        background=findViewById(R.id.backgroundMain)
        imglightOn.visibility=View.INVISIBLE //hiding image at first, so that we'll know if app is working fine or not
        //initializing sensorManager var and sensor var
        sensorManager= getSystemService(Context.SENSOR_SERVICE) as SensorManager
        sensor= sensorManager!!.getDefaultSensor(Sensor.TYPE_LIGHT)

    }

    override fun onResume() {
        // Register a listener for the sensor.
        super.onResume()
        //type "?" after sensorManager to tackle null pointer
        sensorManager?.registerListener(this, sensor, SensorManager.SENSOR_DELAY_NORMAL)
    }
    override fun onSensorChanged(event: SensorEvent?) {
        //most imp sensor
        //sensor se info array form me ati hai
        //try catch is used to avoid app crashing if some issue is faced by sensor (IMP CONCEPT)
        try{
            if (event != null) {
                Log.d(TAG,"onSensorChanged: "+ event.values[0])
            }
            if(event!!.values[0]<30){
                //light is dim
                imglightOn.visibility= View.INVISIBLE //torch image remove ho jayegi agr light dim hai to
//                background.setBackgroundColor(getColor(R.color.black))
                imglightOff.visibility=View.VISIBLE
                background.setBackgroundColor(getColor(R.color.gray))
            }else{
                //show torch if light intensity is high
                imglightOn.visibility=View.VISIBLE
                imglightOff.visibility=View.INVISIBLE
                background.setBackgroundColor((getColor(R.color.light)))
            }
        }catch (e:IOException){
            Log.d(TAG,"onSensorChanged: + ${e.message}") //$(e.message0 will show error message
        }
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
    }
    override fun onPause() {
        // Be sure to unregister the sensor when the activity pauses. so that there will be no activity if app is unused
        super.onPause()
        sensorManager?.unregisterListener(this)
    }

}
```

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
<!--    since we are using sensor so we have to add features here; for light add light sensor-> make is req true -->
    <uses-feature
        android:name="android.hardware.sensor.light"
        android:required="true" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.LightSensor"
```

```
        tools:targetApi="31">
        <activity
            android:name=".FirstPage"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

activity_first_page.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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FirstPage"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_gravity="center"
        android:layout_marginTop="250dp"
        app:srcCompat="@drawable/splashscreen_logo"
        tools:layout_editor_absoluteY="305dp" />
</LinearLayout>
```

FirstPage.kt

```
package com.example.lightsensor

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.Handler
import android.os.Looper

class FirstPage : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first_page)
        supportActionBar?.hide() //to hide action bar where you get to see back arrow or left top
        //to show splash screen
        Handler(Looper.getMainLooper()).postDelayed({
            val i= Intent(this, MainActivity::class.java)
            startActivity(i)
            finish()
        },3000)
    }
}
```

MainActivity.kt

```
package com.example.lightsensor

import android.content.ContentValues.TAG
import android.content.Context
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import android.media.Image
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.view.View
import android.widget.ImageView
import android.widget.LinearLayout
import androidx.constraintlayout.widget.ConstraintLayout
import java.io.IOException
// isme you have to attend sensor event listner, then implement members
class MainActivity : AppCompatActivity(), SensorEventListener {

    var sensor: Sensor?=null //1st var is of ->type variable
    var sensorManager:SensorManager?=null //2nd var is of sensor manager type
    //we have used following two in activity layout so to use them we have declared them
    lateinit var imglightOn:ImageView
    lateinit var imglightOff:ImageView
    lateinit var background:LinearLayout


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        imglightOn=findViewById(R.id.lighton_img)
        imglightOff = findViewById(R.id.lightoff_img)
        background=findViewById(R.id.backgroundMain)
        imglightOn.visibility=View.INVISIBLE //hiding image at first, so that we'll know if app is working fine or not
        //initializing sensorManager var and sensor var
        sensorManager= getSystemService(Context.SENSOR_SERVICE) as SensorManager
        sensor= sensorManager!!.getDefaultSensor(Sensor.TYPE_LIGHT)

    }

    override fun onResume() {
        // Register a listener for the sensor.
        super.onResume()
        //type "?" after sensorManager to tackle null pointer
        sensorManager?.registerListener(this, sensor, SensorManager.SENSOR_DELAY_NORMAL)
    }
    override fun onSensorChanged(event: SensorEvent?) {
        //most imp sensor
        //sensor se info array form me ati hai
        //try catch is used to avoid app crashing if some issue is faced by sensor (IMP CONCEPT)
        try{
            if (event != null) {
                Log.d(TAG,"onSensorChanged: "+ event.values[0])
            }
            if(event!!.values[0]<30){
                //light is dim
                imglightOn.visibility= View.INVISIBLE //torch image remove ho jayegi agr light dim hai to
//                background.setBackgroundColor(getColor(R.color.black))
                imglightOff.visibility=View.VISIBLE
                background.setBackgroundColor(getColor(R.color.gray))
            }else{
                //show torch if light intensity is high
                imglightOn.visibility=View.VISIBLE
                imglightOff.visibility=View.INVISIBLE
                background.setBackgroundColor((getColor(R.color.light)))
            }
        }catch (e:IOException){
            Log.d(TAG,"onSensorChanged: + ${e.message}") //${e.message0 will show error message
        }
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
    }
    override fun onPause() {
        // Be sure to unregister the sensor when the activity pauses. so that there will be no activity if app is unused
        super.onPause()
        sensorManager?.unregisterListener(this)
    }

}
```

activity_main.xml

```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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
    android:background="#B88267"
    android:id="@+id/backgroundMain">

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

        <ImageView
            android:id="@+id/lighton_img"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="0dp"
            android:layout_marginTop="43dp"
            app:srcCompat="@drawable/lights_on" />
        <ImageView
            android:id="@+id/lightoff_img"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="0dp"
            android:layout_marginTop="43dp"
            app:srcCompat="@drawable/lights_off" />
    </RelativeLayout>
</LinearLayout>
```