# Lab 4 – Sensors

## Sensors

The Android platform supports three broad categories of sensors:

- Motion sensors – these sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
- Environmental sensors – these sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.
- Position sensors – these sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

To use sensors on the device we need to use SensorManager:

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as
SensorManager
```

To list sensors supported by device we need to use `getSensorList` method:

```
val deviceSensors: List<Sensor> =
sensorManager.getSensorList(Sensor.TYPE_ALL)
```

Developer can also determine whether a specific type of sensor exists on a device by using the `getDefaultSensor()` method and passing in the type constant for a specific sensor. If a device has more than one sensor of a given type, one of the sensors must be designated as the default sensor:

```
sensorManager = getSystemService(Context.SENSOR_SERVICE) as
SensorManager
if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
!= null) {
    // Success!
} else {
    // Failure! No magnetometer.
}
```

To monitor raw sensor data you need to implement two callback methods that are exposed through the SensorEventListener interface: `onAccuracyChanged()` and `onSensorChanged()`.
The Android system calls these methods whenever the following occurs:

- A sensor's accuracy changes.
  In this case the system invokes the `onAccuracyChanged()` method, providing you with a reference to the Sensor object that changed and the new accuracy of

the sensor. Accuracy is represented by one of four status constants: `SENSOR_STATUS_ACCURACY_LOW`, `SENSOR_STATUS_ACCURACY_MEDIUM`, `SENSOR_STATUS_ACCURACY_HIGH`, `\` `SENSOR_STATUS_UNRELIABLE`.

- A sensor reports a new value.
  In this case the system invokes the `onSensorChanged()` method, providing you with a SensorEvent object. A SensorEvent object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.

The below code show example for light sensor:

```
class SensorActivity : Activity(), SensorEventListener {
    private lateinit var sensorManager: SensorManager
    private var mLight: Sensor? = null

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

        sensorManager =
getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mLight =
sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy:
Int) {
        //..
    }

    override fun onSensorChanged(event: SensorEvent) {
        val lux = event.values[0]
    }

    override fun onResume() {
        super.onResume()
        mLight?.also { light ->
            sensorManager.registerListener(this, light,
SensorManager.SENSOR_DELAY_NORMAL)
        }
    }

    override fun onPause() {
        super.onPause()
        sensorManager.unregisterListener(this)
```

```
    }
}
```

Important:

- use sensors only in foreground (when app is on screen),
- unregister sensor listeners in `onPause()` method and register them again in `onResume()` method.

**GPS**

To access the device's location, request either ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION. The permission you choose determines the accuracy of the location returned by the FusedLocationProviderClient API:

- If you specify ACCESS_COARSE_LOCATION, the API can use Wi-Fi, mobile cell data, or both to determine the device's location. The API returns the location with an accuracy approximately equivalent to a city block.
- If you specify ACCESS_FINE_LOCATION, the API can determine as precise a location as possible from the available location providers, including the GPS as well as Wi-Fi and mobile cell data.

We need add specific permission to manifest.xml, for example:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"

package="com.google.android.gms.location.sample.basiclocations
ample" >

  <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
</manifest>
```

To use location the LocationClient is required:

```
private lateinit var fusedLocationClient:
FusedLocationProviderClient

override fun onCreate(savedInstanceState: Bundle?) {
    // ...

    fusedLocationClient =
LocationServices.getFusedLocationProviderClient(this)
}
```

Once you have created the Location Services client you can get the last known location of a user's device. When your app is connected to these you can use the fused location provider's getLastLocation() method to retrieve the device location. The precision of the location returned by this call is determined by the permission setting you put in your app manifest:

```
fusedLocationClient.lastLocation
    .addOnSuccessListener { location : Location? ->
       // …
    }
```

The getLastLocation() method returns a Task that you can use to get a Location object with the latitude and longitude coordinates of a geographic location. The location object may be null in the following situations:

- Location is turned off in the device settings. The result could be null even if the last location was previously retrieved because disabling location also clears the cache.
- The device never recorded its location, which could be the case of a new device or a device that has been restored to factory settings.
- Google Play services on the device has restarted, and there is no active Fused Location Provider client that has requested location after the services restarted. To avoid this situation, you can create a new client and request location updates yourself.

**Exercises:**

1. (2p.) Write app which list all available sensors on the device and presents them as a list.
2. (4p.) Write a game in which user move a ball on the screen using available motion sensor on the device (emulator). User lose when ball hit the edge of the screen.
3. (4p.) Write an app which use GPS and inform user when he/she leaves some area. App need to work only in foreground.