COM4510/6510
Software Development for Mobile Devices

Lab 7: Sensing

Temitope (Temi) Adeosun
The University of Sheffield
t.adeosun@sheffield.ac.uk

© Po Yang, Temi Adeosun, University of Sheffield

Three exercises

- Exercise 1: Build an app to monitor the barometric pressure reading
- Exercise 2: Add accelerometer sensor monitoring
- Exercise 3: Extend the app to monitor acceleration and only take barometric pressure reading when a specific criteria is met. Manage rates.

© Po Yang, Temi Adeosun, University of Sheffield

Getting started

- Download the Lab7 Exercise 1 starter code
  - Explore the starter code and make sure you understand what is happening in there.
- Android powered devices contain sensors in three categories:
  - Motion, Position/Orientation, Environment sensors
    - In this lab we will implement one motion sensor, and one environment sensor
- To identify a sensors on a device, you need to
  - instantiate a SensorManager object by calling getSystemService(), passing SENSOR_SERVICE as parameter:

https://developer.android.com/guide/topics/sensors/sensors_overview

Getting started

var sensorManager = context.getSystemService(Context.*SENSOR_SERVICE*) as SensorManager

- instantiate a Sensor object using the SensorManager, and specifying the sensor type. This code snippet instantiates a proximity sensor:

  sensor = sensorManager.getDefaultSensor(Sensor.*TYPE_PROXIMITY*)

- after getting a sensor object, we monitor the sensor by registering a SensorEventListener object with the sensorManager object. SensorEventListener:
  - can be the current class, implementing this Interface
  - or an object of this type
  - This interface exposes two event listener functions **onSensorChanged**, and **onAccuracyChanged**
- Stop monitoring as follows:

  sensorManager.unregister(eventListener)

  sensorManager.unregister(this) – usage if current class

https://developer.android.com/guide/topics/sensors/sensors_overview

Getting started

```
class SensorModel : SensorEventListener {
        ...
    sensorManager?.also {
        it.registerListener(this, sensor,
            SensorManager.SENSOR_DELAY_NORMAL) }

    override fun onSensorChanged(event: SensorEvent) { … }
    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) { … }
… }
```

SensorModel implements SensorEventListener
So, it is the listener

```
barometerEventListener  = object : SensorEventListener {
    override fun onSensorChanged(event: SensorEvent) { … }
    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) { … } }

sensorManager?.also {
    it.registerListener(barometerEventListener, sensor,
        SensorManager.SENSOR_DELAY_NORMAL) }
```

SensorEventListener implemented as an object of given type

Exercise 1

- Implement barometer sensor for monitoring pressure
  - and test it with the emulator
- The project is setup with an MVVM architecture and uses LiveData for observing data changes from the View
  - See SensorView and SensorViewModel.
  - Ensure you understand what is happening

Exercise 1

- Update the Barometer model class:
  - Define variables of the type SensorManager, Sensor and SensorEventManager:

```
private var sensorManager: SensorManager?
private var barometerSensor: Sensor
private var barometerEventListener: SensorEventListener? = null
```

- In the initialization block, acquire a barometer sensor:

```
sensorManager = context.getSystemService(Context.SENSOR_SERVICE) as SensorManager
barometerSensor = sensorManager?.getDefaultSensor(Sensor.TYPE_PRESSURE)!!
```

Exercise 1

- Initialization the SensorEventListener (if you choose to use this, otherwise detele the variable from previous screen and refactor your code as needed)

```
barometerEventListener  = object : SensorEventListener {
    override fun onSensorChanged(event: SensorEvent) { }

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

}
```

You will get round to implementing onSenSorChanged() shortly.

Exercise 1

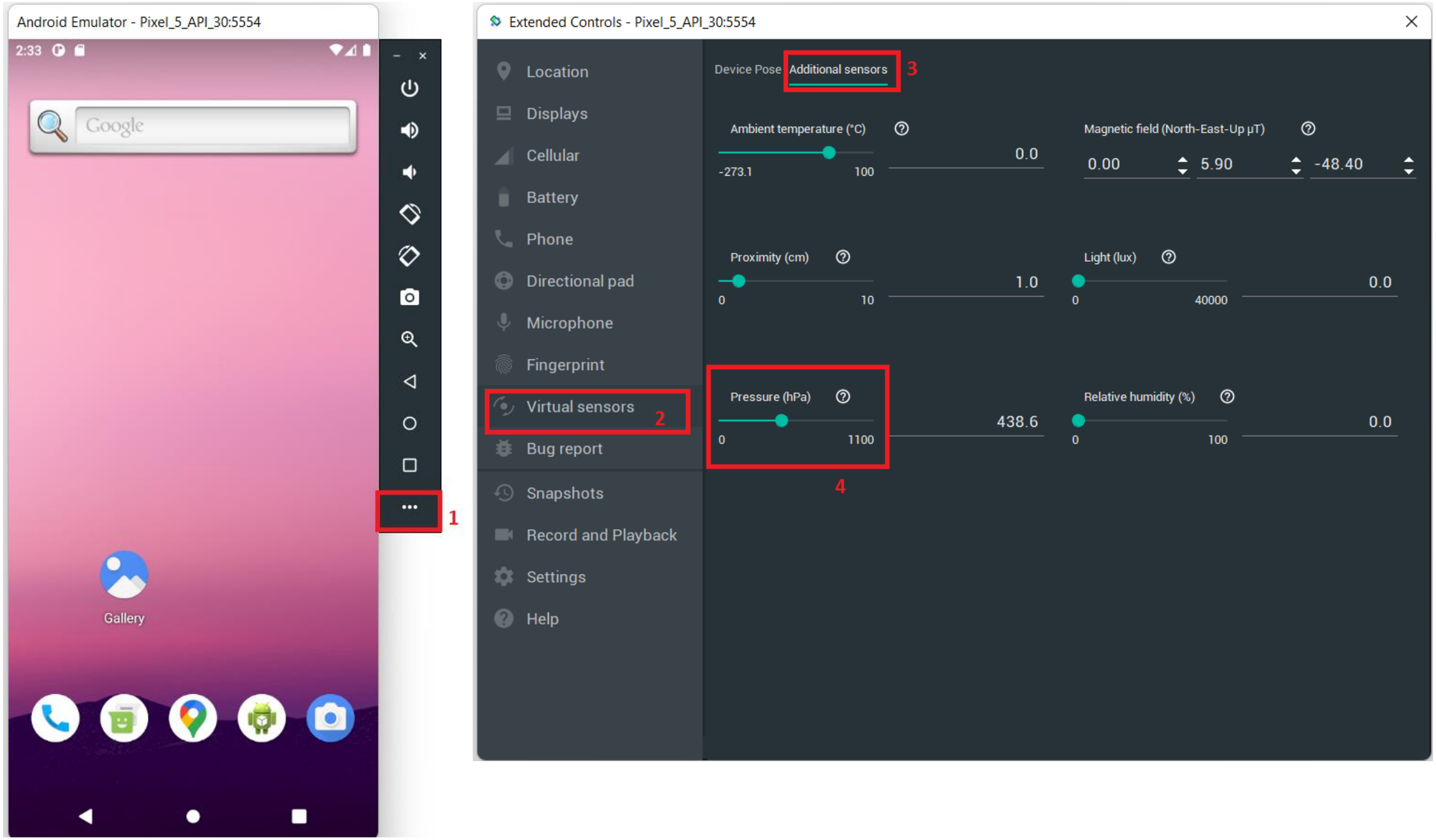
- Implement the startBarometerSensing()and stopBarometerSensing() functions:
  - Using the code snippets in the getting started section as guide, implement these two functions
- Implement the onSensorChanged() method of your sensor:
  - The sensor information is contained in the onSensorChanged event function parameter
  - Just write a log entry of the data sensor data for now.

Exercise 1

```
override fun onSensorChanged(event: SensorEvent) {
    Log.i(TAG, “Current barometric pressure: " + event.values[0])
}
```

- Run the app and observe the LogCat: View > Tools Windows > Logcat
  - Try filter logcat with value “I/Barometer”
  - click the start button to start monitoring the sensor.
- To change pressure values in the AVD:
  - Open your emulator’s extended control and go to pressure in the Virtual sensors section

Exercise 1



Observer changes in Logcat as you change pressure

Exercise 2

- Download the Lab 7 Exercise 2 starter code
  - Compare this with your solution to exercise 1. Note the following:
    - You have been provisioned with a Utility class – study it
      - The Utility function mSecToString is used in the onSensorChanged. See the usage, the output and read the comments.
        - As questions if you don't quite understand
- If you understand the solution to exercise 1, proceed with exercise 2

Exercise 2

- Now change the app to make use of an accelerometer. https://developer.android.com/guide/topics/sensors/sensors_motion

Make sure to create a new class for the accelerometer as you will need the barometer in exercise 3

- It is a different type of sensor providing a different output
  - it returns 3 values: x, y, z
  - while with the barometer we can do
    - var pressureValue = event.values[0];
  - With the accelerometer, you have too assign
    - x= event.values[0];
    - y= event.values[1];
    - z= event.values[2];

Exercise 2 - Print out

- In onSensorChanged, make the app print out

Log.i(TAG, mSecsToString(actualTimeInMseconds) + " : current position: x:” +x " y: "+ y+ " z: "+ z);

- However only do print this when there is significant movement (see next slide)

Exercise 2 - Relevant movement

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	

- The Accelerometer includes gravitational component along each axis
- So, even when stationary, it will record significant movement
  - Eliminate the gravitational component:

Exercise 2 - Relevant movement (cont.)

```
// eliminate gravity, assuming gravity value 9.8m/s^2
    x = abs(event.values[0]).let { if(it >=9.8) it-9.8 else it }.toFloat()
    y = abs(event.values[1]).let { if(it >=9.8) it-9.8 else it }.toFloat()
    z = abs(event.values[2]).let { if(it >=9.8) it-9.8 else it }.toFloat()
```

```
// get the change of the x,y,z values of the accelerometer, from this snippet, you need to
maintain a lastX…lastZ
    deltaX = abs(lastX – x);
    deltaY = abs(lastY -y);
    deltaZ = abs(lastZ - z);
```

Exercise 2 - Relevant movement (cont.)

```
// if the change is below 2, it is just plain noise

    if (deltaX < 2) deltaX = 0

    if (deltaY < 2) deltaY = 0

    if (deltaZ < 2) deltaZ = 0
```

if any delta is > 0 then there was relevant movement

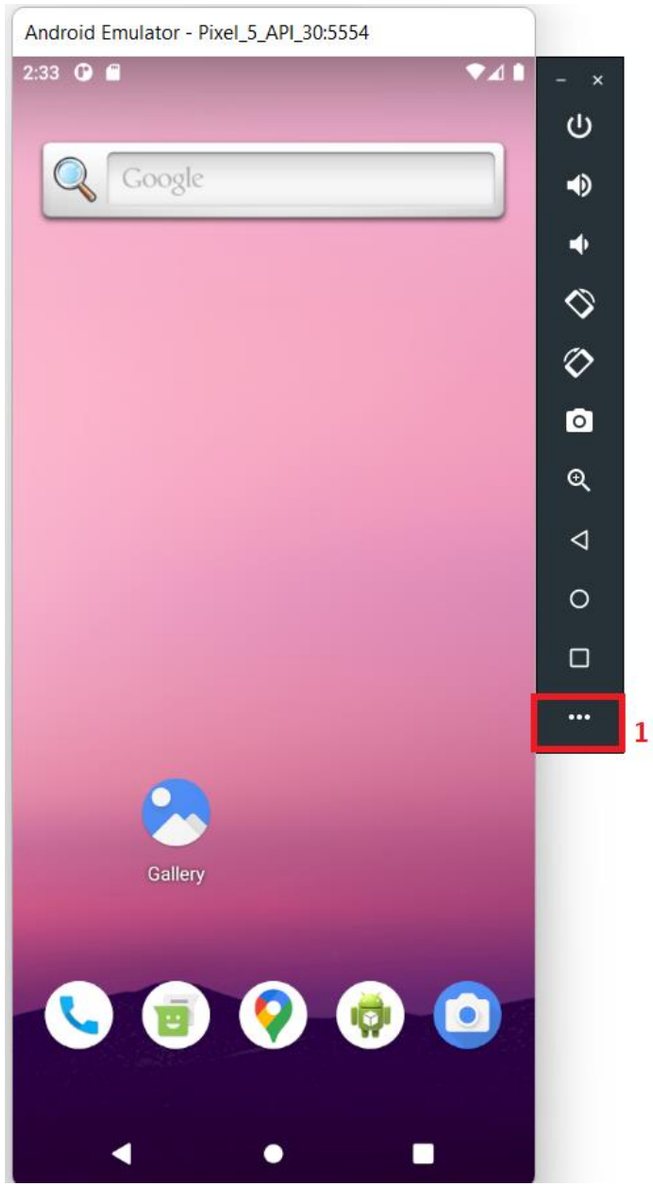
Exercise 3

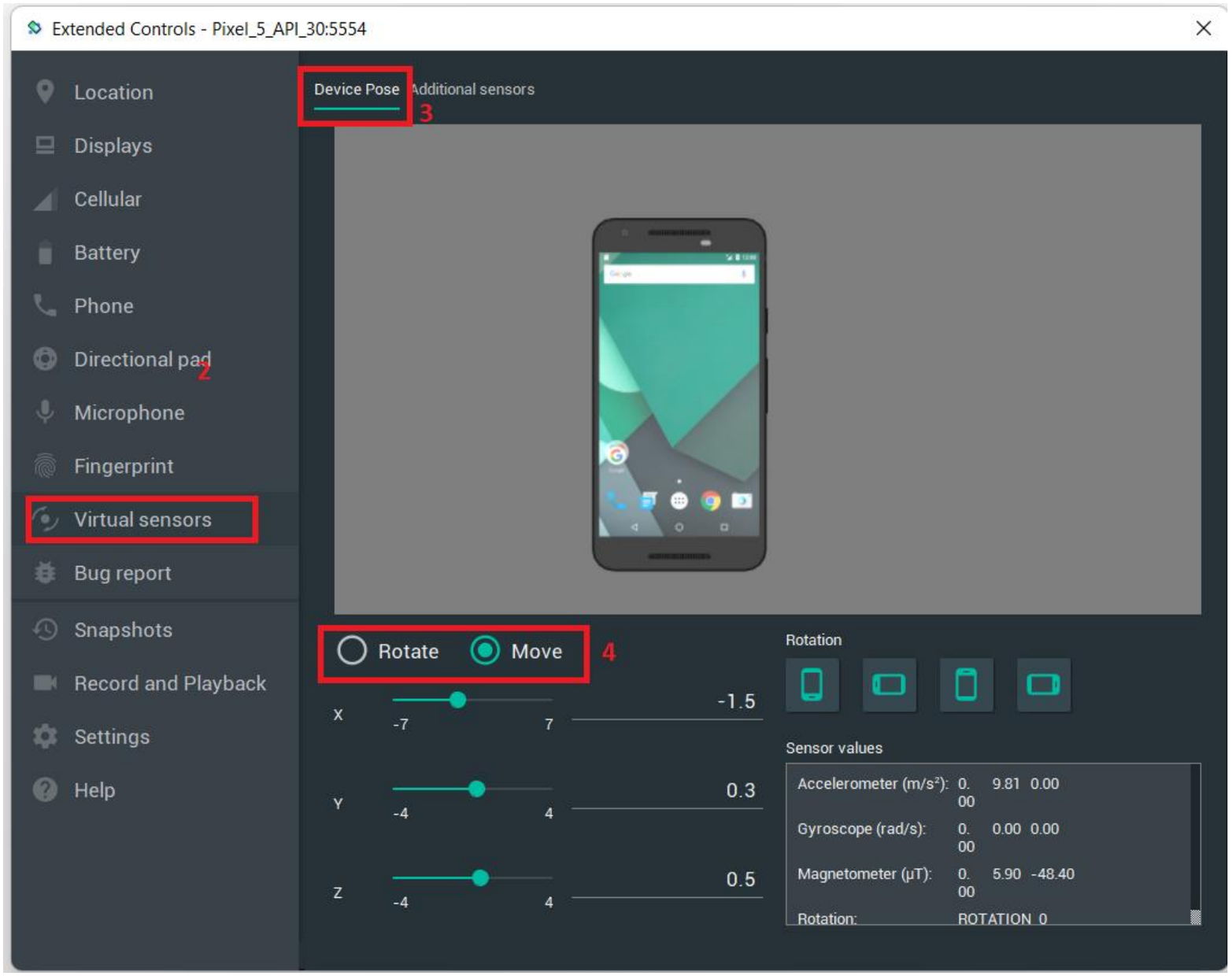
- now change the accelerometer so that
- Every time the phone had a relevant movement, the barometric pressure is taken
- How do you do that?
  - try and think of it
- Suggestion on the next slide

Exercise 3 - Solution

- Remove start/stop of the barometer from onResume, onPause, onClick - if you still have it there
- Insert the start of the barometer when significant movement is detected
- Solution:
  - so you must start the barometer when a new reading of the accelerometer is received AND the change is considerable in terms of movement
  - and stop it as soon as a reading has been provided

Exercise 3 – moving the emulator





You can drag the slides or the emulator image to simulate movemeny

Question

- What happens to your program if you continuously move the phone?

Updated: Exercise 4 – Location (optional)

- This is an optional exercise, as we will be having more on location next week. Following the guidance in the lecture slides:
  - Create an app to retrieve the location using the phone fused location mechanism
    - Try using MVVM pattern.
- Publish location into Log (using Log.i)
- To test in AVD, you do need to set AVD's location first

Set AVD's location