|  |
| --- |
| Android Tutorial – Part 5 |

|  |
| --- |
| 6-19-2018 |



Table of Contents

[Introduction 2](#_Toc517256815)

[Sensors in Android 3](#_Toc517256816)

[Sensor types supported by the Android platform. 3](#_Toc517256817)

[Sensor Categories available in Android 4](#_Toc517256818)

[Android Sensor Framework 5](#_Toc517256819)

[SensorManager (**android.hardware.SensorManager**) 5](#_Toc517256820)

[Sensor (**android.hardware.Sensor)** 5](#_Toc517256821)

[SensorEvent (**android.hardware.SensorEvent)** 5](#_Toc517256822)

[SensorEventListener (**android.hardware.SensorEventListener)** 6](#_Toc517256823)

[References 6](#_Toc517256824)

# Introduction

This is the part four of the android tutorial series. It is a continuation from last week. In order to follow this successfully, it is required to have,

* A basic understanding given about android in last session.
* The environment set up.
* The project created during last tutorial, opened in Android Studio.
* AVD or an Actual device ready for app deployment.

To catch up, in the last session (Android Tutorial Part 3),

* Different storage techniques in Android
* SQLite database
* Important classes and their methods of Android’s SQLite package
* Table structure to save a message
* Model class structure to save a message
* SQLiteOpenHelper class, and its implementation
* DB Operation helper class
* Android ListView
* Custom Adapter

<https://github.com/nadee158/android_tutorial_part_4.git>

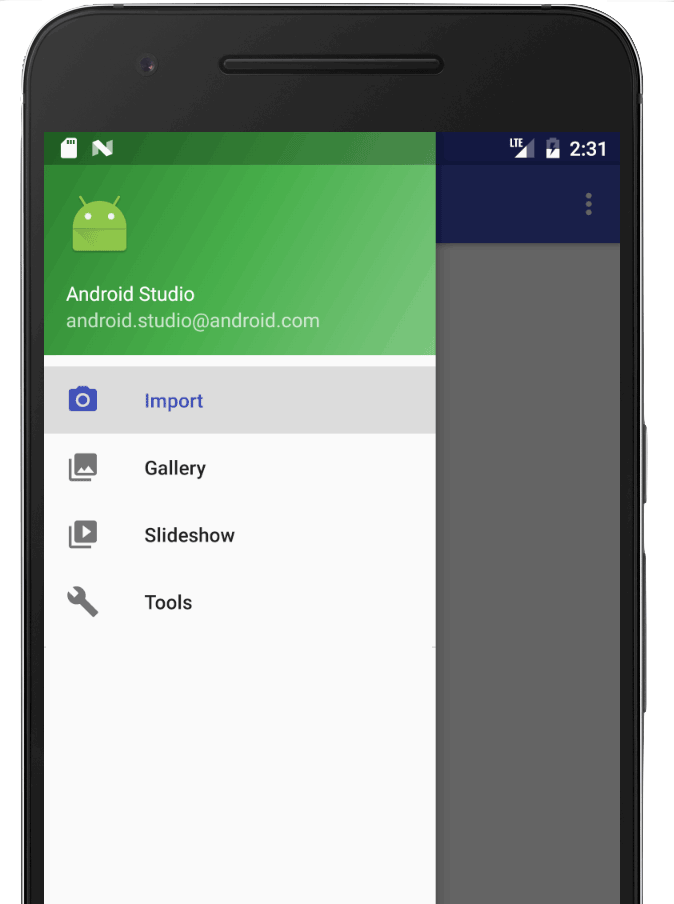
With that knowledge in hand, in this session below areas will be covered,

# Create a navigation drawer

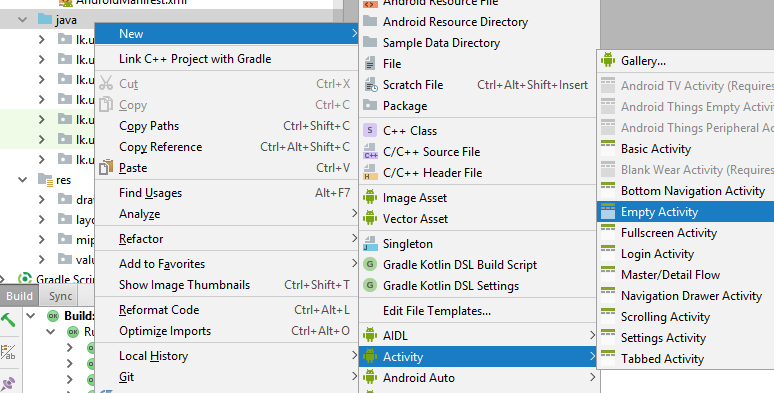
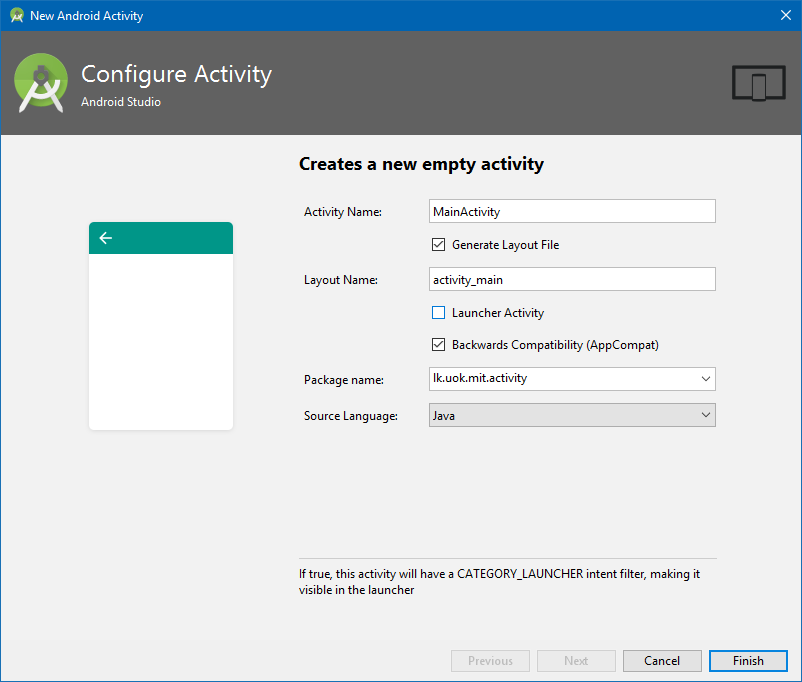
In this tutorial, our main focus is to look in to sensors available in android and how to use them for our application. Before starting that part, first we need to create a method to navigate between multiple activities, which will become useful in the next part.

## The navigation drawer

The navigation drawer is a UI panel that shows your app's main navigation menu. It is hidden when not in use, but appears when,

* the user swipes a finger from the left edge of the screen or,
* When at the top level of the app, the user touches the drawer icon in the app bar.  
  
* A navigation drawer is made up of the following components:
  + An instance of the **DrawerLayout** component.
  + An instance of the **NavigationView** component embedded as a child of the **DrawerLayout**.
  + A menu resource file containing the options to be displayed within the navigation drawer.
  + An optional layout resource file containing the content to appear in the header section of the navigation drawer.
  + A listener assigned to the **NavigationView** to detect when an item has been selected by the user.
  + An **ActionBarDrawerToggle** instance to connect and synchronize the navigation drawer to the app bar.
    - The **ActionBarDrawerToggle** also displays the drawer indicator in the app bar which presents the drawer when tapped.

Lets now see how to implement a navigation drawer using the **DrawerLayout** APIs available in **the Support Library**

1. Add an empty activity to the app – this is to be used as the landing page of the app  
   Currently, when the app starts up, it directly goes to “**SendMessageActivity**” activity, we are going to change this to a main landing page
   1. Right click on “java”, and go to “**New**”-->”**Activity**”-->”**EmptyActivity**”  
      
   2. Fill the details on the next UI as shown below;  
      
      1. Activity Name:- **MainActivity**
      2. Check the “**Generate Layout File**” option
      3. Layout Name:- **activity\_main**
      4. Package Name:- **lk.uok.mit.activity**

To begin using **DrawerLayout** and **NavigationView** in your project, you'll need to import the design support and also the Android support artifact. So add these to your module's **build.gradle** file to import them.

# Sensors in Android

Many Android-powered devices include built-in sensors that measure motion, orientation, and environmental conditions such as ambient light or temperature.

The Android sensor framework lets developers to access many types of sensors. Mainly two types of sensors;

* **hardware-based**
  + These are physical components built into a handset or tablet device.
  + They derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change.
* **software-based**.
  + These sensors are not physical devices, although they mimic hardware-based sensors.
  + Software-based sensors derive their data from one or more of the hardware-based sensors and are sometimes called **virtual sensors** or **synthetic sensors**.
  + The **linear** **acceleration sensor** and the **gravity sensor** are examples of software-based sensors.

Some Android-powered devices have every type of sensor.   
E.g.:- most handset devices and tablets have an accelerometer and a magnetometer, but fewer devices have barometers or thermometers.

Also, a device can have more than one sensor of a given type. For example, a device can have two gravity sensors, each one having a different range.

## Sensor types supported by the Android platform.

|  |  |  |  |
| --- | --- | --- | --- |
| Sensor | Type | Description | Common Uses |
| TYPE\_ACCELEROMETER | Hardware | Measures the acceleration force in m/s2 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/s2 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\_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/s2 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. |
| TYPE\_ORIENTATION | Software | Measures degrees of rotation that a device makes around all three physical 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. | Determining device 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. |
| 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. |

## Sensor Categories available in Android

The android platform supports three broad categories of sensors.

* Motion Sensors
  + Are used to infer complex user gestures and motions
  + These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
    - E.g. :- such tilt, shake, or rotation
* Environmental sensors
  + Are used measure various environmental parameters near the device
    - E.g.:- ambient air temperature and pressure, illumination, and humidity.
  + This category includes barometers, photometers, and thermometers.
* Position sensors
  + Are used to measure the physical positioning of the device
  + This category Includes orientation sensors and magnetometers.

These sensors can provide data to apps with high precision and accuracy.

The **device camera**, **fingerprint sensor**, **microphone, and GPS (location) sensor** all have **their own APIs** and **are NOT considered** part of the Android sensor framework.

## Android Sensor Framework

Android sensor framework allows the developer to access these sensors and acquire raw sensor data. The sensor framework is part of the **android.hardware** package and includes the following classes and interfaces;

### SensorManager (**android.hardware.SensorManager**)

* This class is used by **developers** to get access to the sensors available within the Android platform.
* Not every Android-equipped device will support all of the sensors in the SensorManager.
* This class can be used to create an instance of the sensor service.
* Provides various methods for
  + Accessing and listing sensors.
  + Registering and unregistering sensor event listeners.
  + Acquiring orientation information.
* This class also provides several sensor constants that are used to
  + Identify Sensor
    - Sensor type - Orientation, accelerometer, light, magnetic field, proximity, temperature, etc
  + Report sensor accuracy.
    - Accuracy - High, low, medium, unreliable
  + Set data acquisition rates.
    - Sampling rate - Fastest, game, normal, user interface
      * When an application requests a specific sampling rate, it is really only a hint, or suggestion, to the sensor subsystem.
      * There is no guarantee of a particular rate being available.
  + Calibrate sensors.
* An instance of the class can be created as below;  
  ***SensorManager sensorManager = (SensorManager)this.getSystemService(SENSOR\_SERVICE);***

### Sensor (**android.hardware.Sensor)**

* This class is used by **developers** to create an instance of a specific sensor
  + By calling the **getDefaultSensor()** method of SensorManager class.
* It provides various methods to use a sensor's capabilities.
* An instance of the class can be created as below;  
  ***Sensor lightSensor = sensorManager.getDefaultSensor(Sensor.TYPE\_LIGHT);***

### SensorEvent (**android.hardware.SensorEvent)**

* The **system** uses this class to create a sensor event object.
* It provides information about a sensor event.
* A sensor event object includes the following information:
* The raw sensor data.
* The type of sensor that generated the event.
* The accuracy of the data.
* The timestamp for the event.

### SensorEventListener (**android.hardware.SensorEventListener)**

* This interface is used by **developers** to create two callback methods that receive notifications (sensor events).
* When sensor values change. --> **onSensorChanged()**
* When sensor accuracy change --> **onAccuracyChanged()**
* Once a sensor is declared, a listener should be registered for it and override two methods which are onAccuracyChanged and onSensorChanged, as shown below;  
  ***sensorManager.registerListener(this, lightSensor, SensorManager.SENSOR\_DELAY\_NORMAL);  
  public void onAccuracyChanged(Sensor sensor, int accuracy) {  
  }  
  public void onSensorChanged(SensorEvent event) {***

***}***

Next lets see how to use sensors in our app;

### List the available sensors of Device

We can get a list of sensors supported by the device in which the app is running, by calling the **getSensorList()** method of the SensorManager, which will return a list of sensors containing their name and version number and much more information.

We can then iterate the list to get the information.

Source code for this tutorial part can be found in Git Repository given below: - <https://github.com/nadee158/android_tutorial_part_5.git>

# References

Ableson, F. (2009, June 16). *Tapping into Android's sensors*. Retrieved from www.ibm.com: https://www.ibm.com/developerworks/library/os-android-sensor/index.html

*Android AutoCompleteTextView Example*. (2018, January 1). Retrieved from JavaTPoint: https://www.javatpoint.com/android-autocompletetextview-example

*Application Fundamentals*. (2018, May 2018). Retrieved from Android Developers: https://developer.android.com

CHUGH, A. (2018, April 2). *Android AutoCompleteTextView Example Tutorial*. Retrieved from JournalDev: https://www.journaldev.com/9574/android-autocompletetextview-example-tutorial

google-developer-training. (2018, June 18). *Working with sensor data*. Retrieved from google-developer-training.gitbooks.io: https://google-developer-training.gitbooks.io/android-developer-advanced-course-practicals/unit-1-expand-the-user-experience/lesson-3-sensors/3-1-p-working-with-sensor-data/3-1-p-working-with-sensor-data.html

Green, P. (2015, April 03). *Using Android Sensors in Your App*. Retrieved from www.sitepoint.com: https://www.sitepoint.com/using-android-sensors-application/

Hathibelagal, A. (2017, January 27). *Android Sensors in Depth: Proximity and Gyroscope*. Retrieved from Envato.com: https://code.tutsplus.com/tutorials/android-sensors-in-depth-proximity-and-gyroscope--cms-28084

*Sensors Overview*. (2018, June 19). Retrieved from Android Developers: https://developer.android.com/guide/topics/sensors/sensors\_overview

Tutorialspoint. (2018, June 18). *Android - Sensors*. Retrieved from Tutorialspoint: https://www.tutorialspoint.com/android/android\_sensors.htm