|  |
| --- |
| 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>

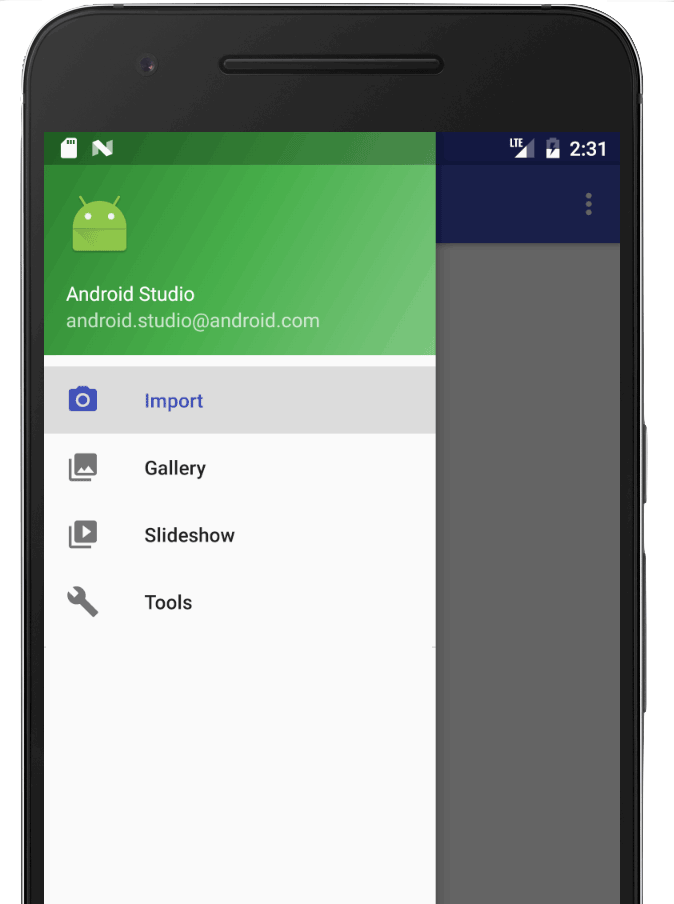
With that knowledge in hand, in this session below areas will be covered,

# 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,

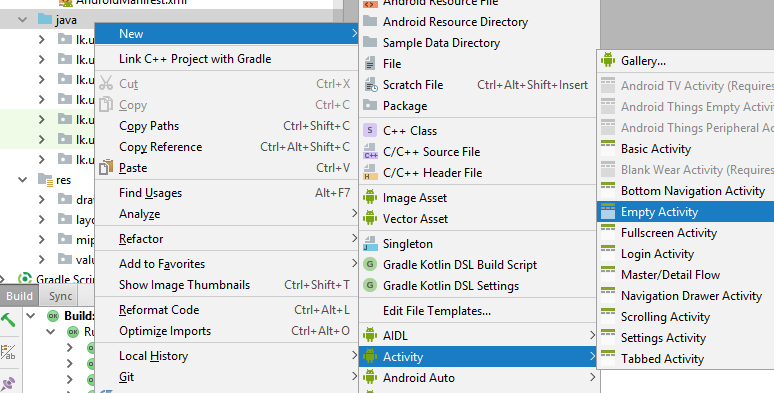
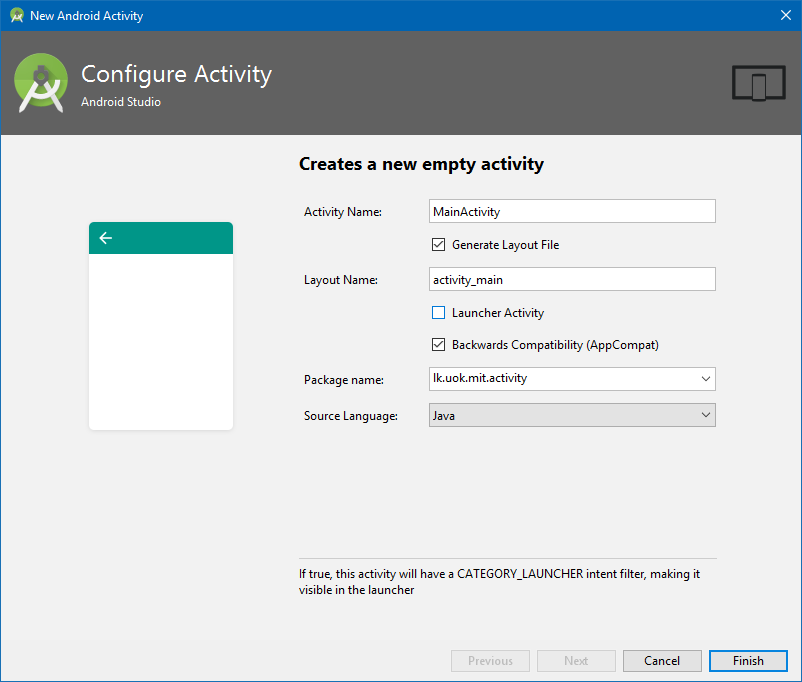
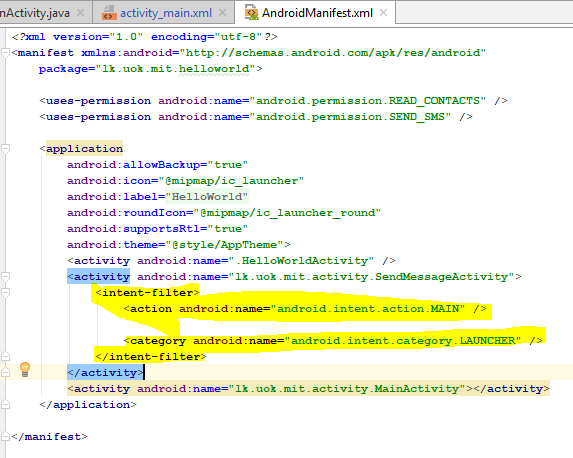
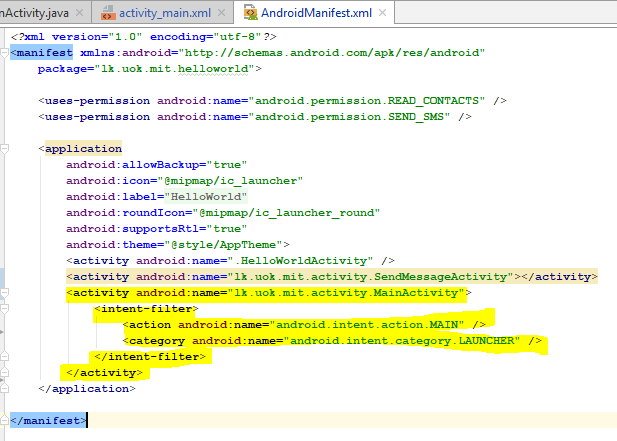
* the user swipes a finger from the left edge of the screen or,
* When at the top level of the app, the user touches the drawer icon in the app bar.  
  
* A navigation drawer is made up of the following components:
  + An instance of the **DrawerLayout** component.
  + An instance of the **NavigationView** 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.

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

## Add a new Main Activity to app

If you could recall, at the time of project creation, we added an activity called “**HelloWorldActivity**”, and it was by default set as the launcher activity (to appear at startup of app).   
After we added the “**SendMessageActivity**”, we made it as the launching activity by modifying the “**AndroidManifest.xml**”.

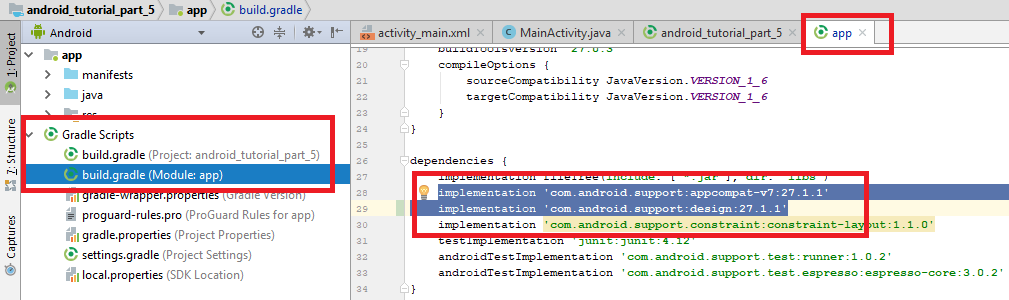
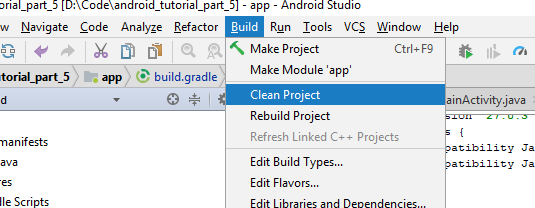
Now we need to add more activities to this app, and each time we will have to keep on making them launching activities. To avoid that, lets add a new activity named “**MainActivity**” and make it the launching activity, and from menu let’s give links to the other activities in the app.

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**
2. Make the “**MainActivity**” as the launcher activity
   1. Open the “**AndroidManifest.xml**” inside “manifests”, which looks like below,  
      
   2. Remove the content within **<intent-filter>** tags from “**SendMessageActivity**” and add it inside “**MainActivity**” as shown below;  
      

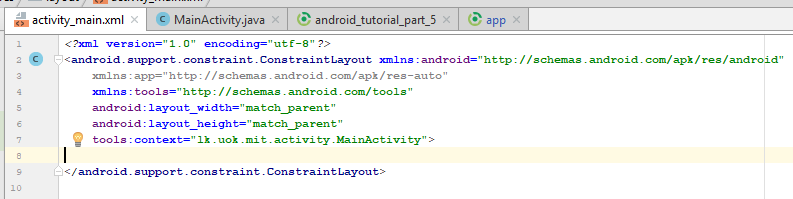
## Add dependencies to Gradle file

**DrawerLayout** acts as a **top-level container** for **window content** that allows for **interactive "drawer"** views to be **pulled out from one or both vertical edges of the window**.

In order to use the **DrawerLayout** and **NavigationView** in the project, it’s required to import the **Design support** and the **Android support** artifacts.

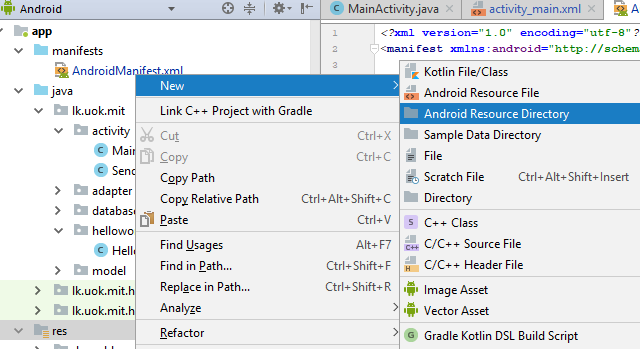
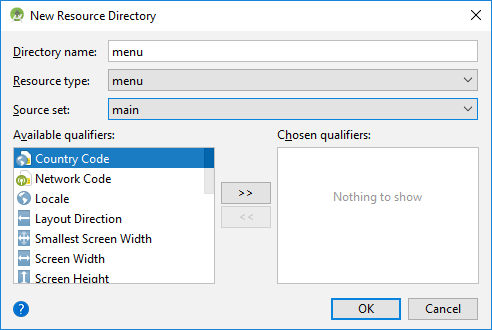
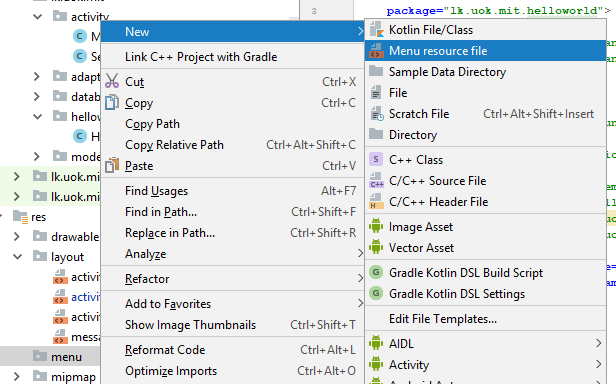
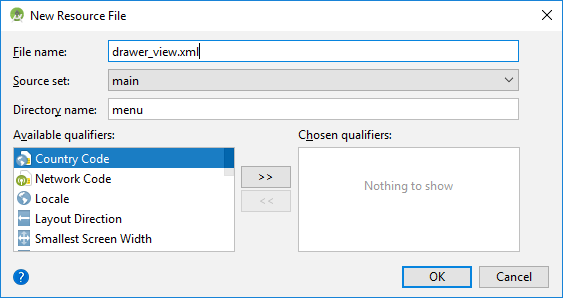
1. Below dependencies should be added to the **build.gradle** file, in order to import them.  
   ***implementation 'com.android.support:appcompat-v7:27.1.1'  
   implementation 'com.android.support:design:27.1.1'***
   1. Open the **build.gradle** file inside **app\build.gradle (**this appears under “Gradle Scripts” section of android studio**)**
   2. Add above two dependencies (if not already available) in “**dependencies{}**” section  
      
   3. After hat **Clean** and **Rebuild** the project, Build🡪Clean / Build🡪Rebuild  
      

## Add a drawer to main activity layout

1. Open “**activity\_main.xml**” created above, inside **res/layout/** and go to “Text” view, it looks like below now;  
     
   1. Above should be changed, and **DrawerLayout** widget and the **NavigationView** should be added like below; (remove all the content from **activity\_main.xml** file and add code given below;)  
      *<?***xml version="1.0" encoding="utf-8"***?>  
      <!-- Use DrawerLayout as root container for activity -->*<**android.support.v4.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"  
       xmlns:app="http://schemas.android.com/apk/res-auto"  
       android:id="@+id/drawer\_layout"  
       android:layout\_width="match\_parent"  
       android:layout\_height="match\_parent"  
       android:fitsSystemWindows="true"**>  
        
       *<!-- Layout to contain contents of main body of screen (drawer will slide over this) -->* <**FrameLayout  
       android:id="@+id/content\_frame"  
       android:layout\_width="match\_parent"  
       android:layout\_height="match\_parent"**>   
       </**FrameLayout**>  
        
       *<!-- Container for contents of drawer - use NavigationView to make configuration easier -->* <**android.support.design.widget.NavigationView  
       android:id="@+id/nav\_view"  
       android:layout\_width="wrap\_content"  
       android:layout\_height="match\_parent"  
       android:layout\_gravity="start"  
       android:fitsSystemWindows="true"** />  
      </**android.support.v4.widget.DrawerLayout**>  
      1. Here we created a **DrawerLayout** widget with the id **drawer\_layout**.
      2. The above layout uses a **DrawerLayout** with two child views:
         1. **FrameLayout** - to contain the **main content**
         2. **NavigationView -** for the **contents of the navigation drawer**.
      3. This example demonstrates some important layout characteristics:
         1. The **FrameLayout** is set to match the parent view's width and height,
            1. Because it represents the entire UI when the navigation drawer is closed.
         2. The **NavigationView** (**the drawer**) must specify its horizontal gravity with the **android:layout\_gravity** attribute.
         3. The **NavigationView** sets **android:fitsSystemWindows** to "**true**" to ensure the contents of the drawer don’t overlay the status bar and other system windows.
         4. The **DrawerLayout** also uses **fitsSystemWindows** as a sign that it needs to inset its children (such as the main content view), but still draw the status bar background in that space as per the material design specs
            1. Which defaults to app theme’s **colorPrimaryDark**.

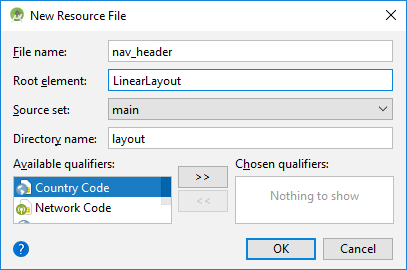
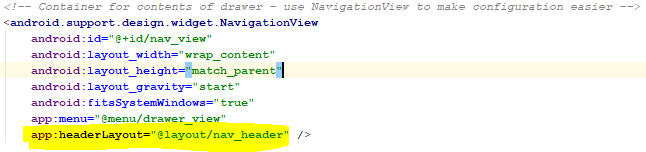
## Add menu items (links to other activities) to drawer

To add menu items, android provides a specific type of resource called “menu”. To configure the menu items listed in the drawer, a menu resource should be specified with the **app:menu** attribute. First a menu resource should be created.

1. To Create a folder named menu under “**res**” folder :-
   1. Right click on “**res**”, select “New”🡪”**Android Resource**” directory   
      
   2. Fill the details in the next screen as shown below;  
      
      1. Directory Name:- menu
      2. Resource Type:- select “menu”
      3. Source set:- main
2. Create the menu resource file with name “**drawer\_view.xml**” inside “**res/menu**” folder
   1. Right click on “**menu**”, select “**New**”🡪 “**Menu Resource File**”  
      
   2. Fill the details in the next screen as shown below;  
      
      1. File Name:- **drawer\_view.xml**
      2. Source Set:- **main**
      3. Directory:- **menu**
   3. Open the **drawer\_view.xml** inside **“res/menu**” folder, and add two menu items to it like below;
      1. **nav\_send\_message –** To access “SendMessageActivity”
      2. **nav\_hello\_world –** To access “HelloWorldActivity”
   4. The whole **drawer\_view.xml** should look like below now;*<?***xml version="1.0" encoding="utf-8"***?>*<**menu xmlns:android="http://schemas.android.com/apk/res/android"**>  
       <**group android:checkableBehavior="single"**>  
       <**item  
       android:id="@+id/nav\_send\_message"  
       android:title="Send Message"** />  
       <**item  
       android:id="@+id/nav\_hello\_world"  
       android:title="Message List"** />  
       </**group**>  
      </**menu**>
   5. Note:- above plain string values are used for demo purposes, please use string resource files each time a string value is used in android to label/give a notification/message etc.
   6. Each menu item can be given an icon if desired using “**android:icon**”
      1. Eg:- ***android:icon="@drawable/ic\_menu\_camera"***
3. Now we created a menu, and it should be added to the “**NavigationView”** of the “**activity\_main.xml**”
   1. Open **activity\_main.xml**  inside res/layout folder
   2. Find the **“<android.support.design.widget.NavigationView**” and add “**drawer\_view**” to its “**app:menu**” attribute like shown below;   
      <**android.support.design.widget.NavigationView  
       android:id="@+id/nav\_view"  
       android:layout\_width="wrap\_content"  
       android:layout\_height="match\_parent"  
       android:layout\_gravity="start"  
       android:fitsSystemWindows="true"  
       app:menu="@menu/drawer\_view"**/>

## Add a header to the nav drawer

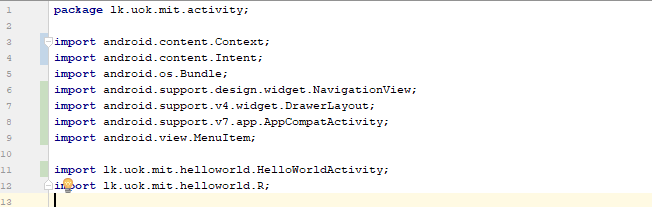
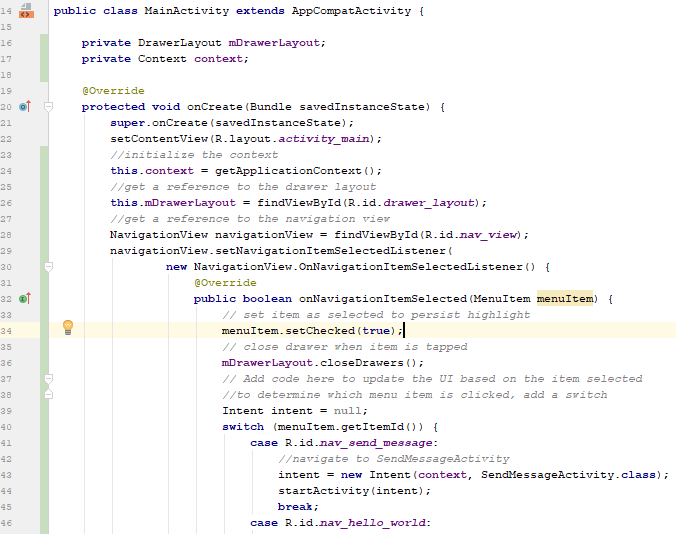
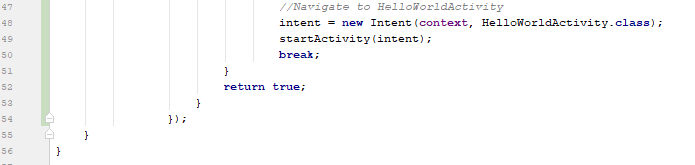
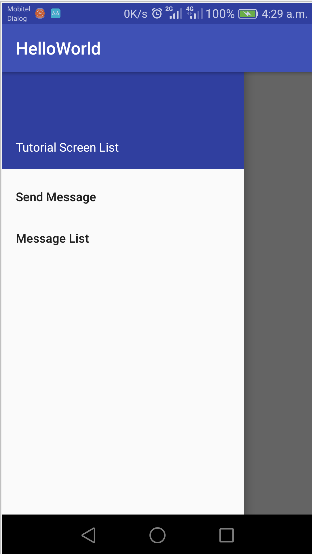
Android provides the option to add a header to the top of the “**drawer**”, that is by allowing to define a separate layout for that using “***app:headerLayout***” attribute of “**NavigationView**”.

1. Define a new layout for the header, named “**nav\_header.xml**” inside “**res/layout**” folder, containing a **LinearLayout** as root element and a **TextView**
   1. Go to “**res/layout**”, right click on it, select “**New**”🡪”**Layout Resource File**”
   2. In the next screen, fill the details as shown below;  
      
      1. File Name:- **nav\_header.xml**
      2. Root element:- **LinearLayout**
      3. Source Set**:- main**
      4. Directory name:- **layout**
2. Open the **nav\_header.xml** inside“**res/layout**” and add below attributes to the LinearLayout tag
   1. **android:layout\_width**="match\_parent"
   2. **android:layout\_height**="192dp"
   3. **android:background**="?attr/colorPrimaryDark"
   4. **android:padding**="16dp"
   5. **android:theme**="@style/ThemeOverlay.AppCompat.Dark"
   6. **android:orientation**="vertical"
   7. **android:gravity**="bottom"
3. Add a text view inside the linear layout, to add he title of the menu bar  
   <**TextView  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:text="Tutorial Screen List"  
    android:textAppearance="@style/TextAppearance.AppCompat.Body1"**/>
4. The complete code of **nav\_header.xml**  will look like below;  
   *<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="match\_parent"  
    android:layout\_height="192dp"  
    android:background="?attr/colorPrimaryDark"  
    android:padding="16dp"  
    android:theme="@style/ThemeOverlay.AppCompat.Dark"  
    android:orientation="vertical"  
    android:gravity="bottom"**>  
    <**TextView  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:text="Tutorial Screen List"  
    android:textAppearance="@style/TextAppearance.AppCompat.Body1"**/>  
   </**LinearLayout**>
5. Now add “**app:headerLayout**” to the “**NavigationView**” of the **activity\_main.xml** like shown below;  
   

## Click events of the menu items

Now the UI design part of the “**drawer**” is completed. Now let’s add the code to “**MainActivity.java**”, to handle the taps or clicks on menu items and navigate user to the desired view.

To receive callbacks when the user taps a list item in the drawer, implement the **OnNavigationItemSelectedListener** interface and attach it to the **NavigationView** by calling **setNavigationItemSelectedListener()**

1. Open the “java/MainActivity”, and add the below code inside “onCreate” method  
   *//initialize the context***this**.**context** = getApplicationContext();  
   *//get a reference to the drawer layout***this**.**mDrawerLayout** = findViewById(R.id.***drawer\_layout***);  
   *//get a reference to the navigation view*NavigationView navigationView = findViewById(R.id.***nav\_view***);  
   *//set the navigation item selected listen to get the click item*navigationView.setNavigationItemSelectedListener(  
    *//create an anonymous inner class, and override* ***onNavigationItemSelected*** method  
    **new** NavigationView.OnNavigationItemSelectedListener() {  
    @Override  
    **public boolean** onNavigationItemSelected(MenuItem menuItem) {  
    *// set item as selected to persist highlight* menuItem.setChecked(**true**);  
    *// close drawer when item is tapped* **mDrawerLayout**.closeDrawers();  
    *// Add code here to update the UI based on the item selected   
    //to determine which menu item is clicked, add a switch* Intent intent = **null**;  
    **switch** (menuItem.getItemId()) {  
    **case** R.id.***nav\_send\_message***:  
    *//navigate to SendMessageActivity* intent = **new** Intent(**context**, SendMessageActivity.**class**);  
    startActivity(intent);  
    **break**;  
    **case** R.id.***nav\_hello\_world***:  
    *//Navigate to HelloWorldActivity* intent = **new** Intent(**context**, HelloWorldActivity.**class**);  
    startActivity(intent);  
    **break**;  
    }  
    **return true**;  
    }  
    });
2. The complete code looks like below  
     
     
   
3. Now a working **drawer** is available in the app, save everything, clean and run the app and check;  
   To view drawer, swipe across the screen from left to right; it will appear like below;  
   

# 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