Creating Android Sliding Sidebar (Hamburger) Menu with Navigation Drawer Icon

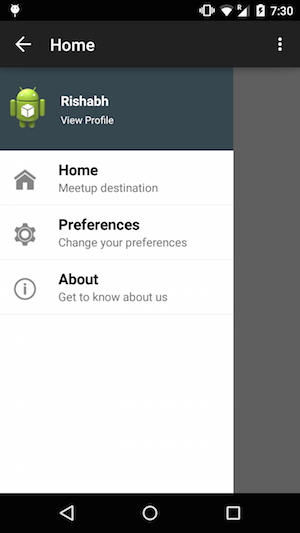
[25 Replies](http://codetheory.in/android-navigation-drawer/#comments)

By now most of us are familiar with [Navigation Drawers](http://developer.android.com/design/patterns/navigation-drawer.html) as they’ve been in use by Facebook, Gmail, Google Play Music, Google Play Music and tons of others apps. It’s that sliding panel that comes out onto the screen when you swipe a finger from the left edge of the screen (or sometimes right edge) or tap on the 3 bar button (sometimes called hamburger icon) in the Action Bar. If still confused then check out the [design guide](http://developer.android.com/design/patterns/navigation-drawer.html) and you’ll know what I’m referring to. The design guide will also tell you when to use it and when not to use it.

In this article I’ll go through how to create a navigation drawer (also referred to as sidebar menu, slide-out menu, sliding menu, hamburger menu, basements, etc.) for your own apps using the [DrawerLayout](http://developer.android.com/reference/android/support/v4/widget/DrawerLayout.html) APIs in the support library. But before getting into how to create a navigation (hamburger) drawer, you should not only go through the [design guide](http://developer.android.com/design/patterns/navigation-drawer.html) but also read a couple of articles on how you might be making a wrong decision by using a hamburger menu and you could possibly resort to other methods to avoid it:

* <https://lmjabreu.com/post/why-and-how-to-avoid-hamburger-menus/>
* <https://redbooth.com/blog/hamburger-menu-iphone-app>
* <http://blog.manbolo.com/2014/06/30/apple-on-hamburger-menus>
* <http://techcrunch.com/2014/05/24/before-the-hamburger-button-kills-you/>
* <http://ux.stackexchange.com/questions/41712/hamburger-menu-is-it-a-good-thing>
* <http://thenextweb.com/dd/2014/04/08/ux-designers-side-drawer-navigation-costing-half-user-engagement/>
* <http://exisweb.net/mobile-menu-abtest>
* <http://exisweb.net/menu-eats-hamburger>
* <https://econsultancy.com/blog/65511-hamburger-menus-for-mobile-navigation-do-they-work/>

That’s a lot to read (but won’t take much time) and too much negativity around navigation drawers. But if you think you’re fixed on using it because it does make sense in several cases, then let’s head towards creating one on Android. Here’s a screenshot of the navigation drawer we’ll create (with an Action Bar).



Simple and Clean!

**DrawerLayout ViewGroup in Your Layout**

We’ll first code our layout in res/layout/activity\_main.xml. In order to build a drawer, we’ll have to use a DrawerLayout as the root view of our layout which’ll contain two child ViewGroups where one of them will represent the main content, whereas the other will contain the contents of the navigation drawer.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76 | <android.support.v4.widget.DrawerLayout      xmlns:android="<http://schemas.android.com/apk/res/android>"      android:id="@+id/drawerLayout"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent">        <!-- The main content view -->      <RelativeLayout          android:id="@+id/mainContent"          android:layout\_width="match\_parent"          android:layout\_height="match\_parent" />        <!-- The navigation drawer -->      <RelativeLayout          android:layout\_width="280dp"          android:layout\_height="match\_parent"          android:id="@+id/drawerPane"          android:layout\_gravity="start">            <!-- Profile Box -->            <RelativeLayout              android:id="@+id/profileBox"              android:layout\_width="match\_parent"              android:layout\_height="100dp"              android:background="@color/material\_blue\_grey\_800"              android:padding="8dp" >                <ImageView                  android:id="@+id/avatar"                  android:layout\_width="50dp"                  android:layout\_height="50dp"                  android:src="@drawable/ic\_launcher"                  android:layout\_marginTop="15dp" />                <LinearLayout                  android:layout\_width="wrap\_content"                  android:layout\_height="42dp"                  android:layout\_centerVertical="true"                  android:layout\_marginLeft="15dp"                  android:layout\_toRightOf="@+id/avatar"                  android:orientation="vertical" >                    <TextView                      android:id="@+id/userName"                      android:layout\_width="wrap\_content"                      android:layout\_height="wrap\_content"                      android:text="Rishabh"                      android:textColor="#fff"                      android:textSize="16sp"                      android:textStyle="bold" />                    <TextView                      android:id="@+id/desc"                      android:layout\_width="wrap\_content"                      android:layout\_height="wrap\_content"                      android:layout\_gravity="bottom"                      android:layout\_marginTop="4dp"                      android:text="View Profile"                      android:textColor="#fff"                      android:textSize="12sp" />              </LinearLayout>          </RelativeLayout>            <!-- List of Actions (pages) -->          <ListView              android:id="@+id/navList"              android:layout\_width="280dp"              android:layout\_height="match\_parent"              android:layout\_below="@+id/profileBox"              android:choiceMode="singleChoice"              android:background="#ffffffff" />        </RelativeLayout>    </android.support.v4.widget.DrawerLayout> |

As you can see, we’ve our DrawerLayout root view and then its first child is a RelativeLayout that represents the main content UI whereas the second child represents the navigation drawer. The navigation drawer first contains a box to represent the profile details followed by a ListView that’ll display a list of different sections that the user can visit (Home, Preferences, About).

The ViewGroup representing the main content UI should be the first child inside the DrawerLayout as the ordering in XML defines the z-index and the drawer should overlay the content, not the other way round.

The XML code is enough to initialize a functional drawer. You can test your app in the emulator or device to start swiping your finger from the left to draw the hamburger menu. However, we’ve some more work to do like initializing the list and adding onclick events on the list items.

**Populating the Drawer List**

This is the first step, we’ll initialize the ListView inside the Drawer with several options. Each list option will be represented by a custom NavItem class object. So create this inner class (of theMainActivity):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class NavItem {      String mTitle;      String mSubtitle;      int mIcon;        public NavItem(String title, String subtitle, int icon) {          mTitle = title;          mSubtitle = subtitle;          mIcon = icon;      }  } |

Next we need another inner class that’ll be a [simple BaseAdapter implementation](http://codetheory.in/understanding-android-lists/) that we’ll bind to the ListView for the sake of population:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | class DrawerListAdapter extends BaseAdapter {        Context mContext;      ArrayList<NavItem> mNavItems;        public DrawerListAdapter(Context context, ArrayList<NavItem> navItems) {          mContext = context;          mNavItems = navItems;      }        @Override      public int getCount() {          return mNavItems.size();      }        @Override      public Object getItem(int position) {          return mNavItems.get(position);      }        @Override      public long getItemId(int position) {          return 0;      }        @Override      public View getView(int position, View convertView, ViewGroup parent) {          View view;            if (convertView == null) {              LayoutInflater inflater = (LayoutInflater) mContext.getSystemService(Context.LAYOUT\_INFLATER\_SERVICE);              view = inflater.inflate(R.layout.drawer\_item, null);          }          else {              view = convertView;          }            TextView titleView = (TextView) view.findViewById(R.id.title);          TextView subtitleView = (TextView) view.findViewById(R.id.subTitle);          ImageView iconView = (ImageView) view.findViewById(R.id.icon);            titleView.setText( mNavItems.get(position).mTitle );          subtitleView.setText( mNavItems.get(position).mSubtitle );          iconView.setImageResource(mNavItems.get(position).mIcon);            return view;      }  } |

We inflated R.layout.drawer\_item, so let’s create res/layout/drawer\_item.xml now:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | <?xml version="1.0" encoding="utf-8"?>  <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:paddingTop="10dp"      android:paddingBottom="10dp">        <ImageView          android:id="@+id/icon"          android:layout\_width="40dp"          android:layout\_height="40dp"          android:src="@drawable/ic\_action\_home"          android:layout\_alignParentLeft="true"          android:layout\_alignParentStart="true"          android:layout\_marginRight="20dp"          android:layout\_marginLeft="10dp"          android:layout\_marginTop="5dp"           />        <TextView          android:id="@+id/title"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:textSize="18sp"          android:gravity="center\_vertical"          android:textColor="#000"          android:text="Line 1"          android:textStyle="bold"          android:layout\_alignParentTop="true"          android:layout\_toRightOf="@+id/icon"          android:layout\_toEndOf="@+id/icon" />        <TextView android:id="@+id/subTitle"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Line 2"          android:layout\_toRightOf="@+id/icon"          android:layout\_below="@+id/title"          android:layout\_alignParentRight="true"          android:layout\_alignParentEnd="true" />    </RelativeLayout> |

Now, we’ll set the adapter for the ListView inside onCreate() of our MainActivity:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | private static String TAG = MainActivity.class.getSimpleName();    ListView mDrawerList;  RelativeLayout mDrawerPane;  private ActionBarDrawerToggle mDrawerToggle;  private DrawerLayout mDrawerLayout;    ArrayList<NavItem> mNavItems = new ArrayList<NavItem>();    @Override  protected void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      setContentView(R.layout.activity\_main);        mNavItems.add(new NavItem("Home", "Meetup destination", R.drawable.ic\_action\_home));      mNavItems.add(new NavItem("Preferences", "Change your preferences", R.drawable.ic\_action\_settings));      mNavItems.add(new NavItem("About", "Get to know about us", R.drawable.ic\_action\_about));        // DrawerLayout      mDrawerLayout = (DrawerLayout) findViewById(R.id.drawerLayout);        // Populate the Navigtion Drawer with options      mDrawerPane = (RelativeLayout) findViewById(R.id.drawerPane);      mDrawerList = (ListView) findViewById(R.id.navList);      DrawerListAdapter adapter = new DrawerListAdapter(this, mNavItems);      mDrawerList.setAdapter(adapter);        // Drawer Item click listeners      mDrawerList.setOnItemClickListener(new AdapterView.OnItemClickListener() {          @Override          public void onItemClick(AdapterView<?> parent, View view, int position, long id) {              selectItemFromDrawer(position);          }      });  } |

Our ListView will be populated now so you should see it in the navigation drawer but it’s time to write code for what will happen once a list item is clicked, which will be defined by theselectItemFromDrawer() Activity method which is what we called inside the onItemClick() method of the OnItemClickListener interface.

So when a list item is clicked, we could do several things. Since they’re different sections, we could just open different Activities [using intents](http://codetheory.in/android-intents/), insert a different Fragment for every list item into the main content view or even show dialogs. In this case we’ll keep it simple by creating a fragment called PreferencesFragment and inserting it whenever an option is selected. Although you might want to create one fragment for each option and based on the position of the clicked item, insert its respective fragement. Let’s get into some code now.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\*  \* Called when a particular item from the navigation drawer  \* is selected.  \* \*/  private void selectItemFromDrawer(int position) {      Fragment fragment = new PreferencesFragment();        FragmentManager fragmentManager = getFragmentManager();      fragmentManager.beginTransaction()              .replace(R.id.mainContent, fragment)              .commit();        mDrawerList.setItemChecked(position, true);      setTitle(mNavItems.get(position).mTitle);        // Close the drawer      mDrawerLayout.closeDrawer(mDrawerPane);  } |

This is how PreferencesFragement.java looks like:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public class PreferencesFragment extends Fragment {          public PreferencesFragment() {          // Required empty public constructor      }          @Override      public View onCreateView(LayoutInflater inflater, ViewGroup container,                               Bundle savedInstanceState) {          // Inflate the layout for this fragment          return inflater.inflate(R.layout.fragment\_preferences, container, false);      }      } |

and here’s the res/layout/fragment\_preferences.xml:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <FrameLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      xmlns:tools="<http://schemas.android.com/tools>" android:layout\_width="match\_parent"      android:layout\_height="match\_parent" tools:context="com.looper.loop.PreferencesFragment">        <!-- TODO: Update blank fragment layout -->      <TextView android:layout\_width="match\_parent" android:layout\_height="match\_parent"          android:text="@string/hello\_blank\_fragment" />    </FrameLayout> |

It’s time to test your app in the emulator or your device itself. You’ll see a nice navigation drawer when you swipe right from the left edge with the profile box and the list in which you can select various options which’ll reflect in the action bar title because of this piece of code insideselectItemFromDrawer():

|  |  |
| --- | --- |
| 1 | setTitle(mNavItems.get(position).mTitle); |

**Observe Open/Close Events of the Drawer**

You can listen for the drawer’s open and close events by implementing[DrawerLayout.DrawerListener](http://developer.android.com/reference/android/support/v4/widget/DrawerLayout.DrawerListener.html) whose [onDrawerOpened()](http://developer.android.com/reference/android/support/v4/widget/DrawerLayout.DrawerListener.html#onDrawerOpened(android.view.View)) and [onDrawerClosed()](http://developer.android.com/reference/android/support/v4/widget/DrawerLayout.DrawerListener.html#onDrawerClosed(android.view.View)) will be called. But if you’re making use of the action bar, then extending [ActionBarDrawerToggle](http://developer.android.com/reference/android/support/v7/app/ActionBarDrawerToggle.html) is a better idea. TheActionBarDrawerToggle implements DrawerLayout.DrawerListener, so you’ll have access to those callbacks but apart from that it also makes sure that the state of both the action bar icon (we’ll see how to add that in a bit) and the drawer is synced. Let’s see some code now.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout, R.string.drawer\_open, R.string.drawer\_close) {      @Override      public void onDrawerOpened(View drawerView) {          super.onDrawerOpened(drawerView);            invalidateOptionsMenu();      }        @Override      public void onDrawerClosed(View drawerView) {          super.onDrawerClosed(drawerView);          Log.d(TAG, "onDrawerClosed: " + getTitle());            invalidateOptionsMenu();      }  };    mDrawerLayout.setDrawerListener(mDrawerToggle); |

The four arguments passed to the ActionBarDrawerToggle constructor are:

* this – Activity object containing the drawer with an ActionBar.
* mDrawerLayout – The DrawerLayout to link to the Activity’s ActionBar.
* R.string.drawer\_open – String resource to describe the “open drawer” action for accessibility purpose.
* R.string.drawer\_close – String resource to describe the “close drawer” action for accessibility purpose.

There’s [another version](http://developer.android.com/reference/android/support/v7/app/ActionBarDrawerToggle.html#ActionBarDrawerToggle(android.app.Activity, android.support.v4.widget.DrawerLayout, android.support.v7.widget.Toolbar, int, int)) of this constructor that you should use when not using an ActionBar but a [ToolBar](http://developer.android.com/reference/android/widget/Toolbar.html).

You’ll notice in the callbacks we called [invalidateOptionsMenu()](http://developer.android.com/reference/android/app/Activity.html" \l "invalidateOptionsMenu()). This method declares that the option menu has changed and hence should be recreated invoking [onPrepareOptionsMenu()](http://developer.android.com/reference/android/app/Activity.html" \l "onPrepareOptionsMenu(android.view.Menu)). So basically you might want to override onPrepareOptionsMenu() to get rid of options from your overflow menu that are irrelevant to your navigation drawer.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | // Called when invalidateOptionsMenu() is invoked  public boolean onPrepareOptionsMenu(Menu menu) {      // If the nav drawer is open, hide action items related to the content view      boolean drawerOpen = mDrawerLayout.isDrawerOpen(mDrawerList);      menu.findItem(R.id.action\_search).setVisible(!drawerOpen);      return super.onPrepareOptionsMenu(menu);  } |

**Showing Up the Hamburger Menu Icon/Indicator**

According to the [design guidelines](https://developer.android.com/design/patterns/navigation-drawer.html), apart from the touch gesture, when the drawer is closed the user should be able to open it by touching the navigation drawer indicator (those 3 lines) in the action bar. So now we’ll see how to display that special icon.

First, add this piece of code inside onCreate():

|  |  |
| --- | --- |
| 1  2 | // More info: <http://codetheory.in/difference-between-setdisplayhomeasupenabled-sethomebuttonenabled-and-setdisplayshowhomeenabled/>  getSupportActionBar().setDisplayHomeAsUpEnabled(true); |

This piece of code will start showing up an arrow which is generally used for “up” (ancestral) navigation. Now when the user taps on this, onOptionsItemSelected() will be called where you can delegate the event to the ActionBarDrawerToggle.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | @Override  public boolean onOptionsItemSelected(MenuItem item) {      // Pass the event to ActionBarDrawerToggle      // If it returns true, then it has handled      // the nav drawer indicator touch event      if (mDrawerToggle.onOptionsItemSelected(item)) {        return true;      }        // Handle your other action bar items...        return super.onOptionsItemSelected(item);  } |

Now you’ll see the back arrow on which you can tap to open/close the drawer. But it won’t change to the 3 bar indicator icon when the drawer is closed. So in order to do that we’ve to override the onPostCreate() method of the Activity where we’ll call the [syncState()](http://developer.android.com/reference/android/support/v7/app/ActionBarDrawerToggle.html" \l "syncState()) method of theActionBarDrawerToggle. This method syncs the state of the indicator with the drawer. So when the drawer is closed it’ll show the correct indicator (indicating tap to open) while when the drawer is open it’ll show the back arrow indicator (indicating tap to close).

|  |  |
| --- | --- |
| 1  2  3  4  5 | @Override  protected void onPostCreate(Bundle savedInstanceState) {      super.onPostCreate(savedInstanceState);      mDrawerToggle.syncState();  } |

With that, its time to test your app that is baked with a navigation drawer now.

**Conclusion**

So we went through how to use the DrawerLayout ViewGroup to create a navigation drawer and then populate that with a list of navigation options. Then we bounded click events to the list items that would replace the main content UI with a particular fragment. We also learnt how to listen to close/open events and show a navigation drawer indicator while using an action bar. You can also definitely make use of a Toolbar in place of an action bar and bind that to ActionBarDrawerToggle.

I don’t think by default the DrawerLayout supports sliding over the action bar which is what the[material design guideline](http://www.google.com/design/spec/patterns/navigation-drawer.html) specify. So doing that might be a good exercise for you along with using custom images for the drawer icon/indicator in the action bar. If you give any of them a shot, then feel free to share your code in the comments section below!