<https://www.topcoder.com/thrive/articles/menu-in-android>

<https://www.geeksforgeeks.org/android-menus/>

<https://www.geeksforgeeks.org/how-to-implement-options-menu-in-android/>

<https://developer.android.com/develop/ui/views/components/menus>

<https://www.geeksforgeeks.org/how-to-add-and-customize-back-button-of-action-bar-in-android/>

**Experiment No 3.3**

**Design the Android application using menus and action bar.**

**3.3.1- Design the Android application using menus**

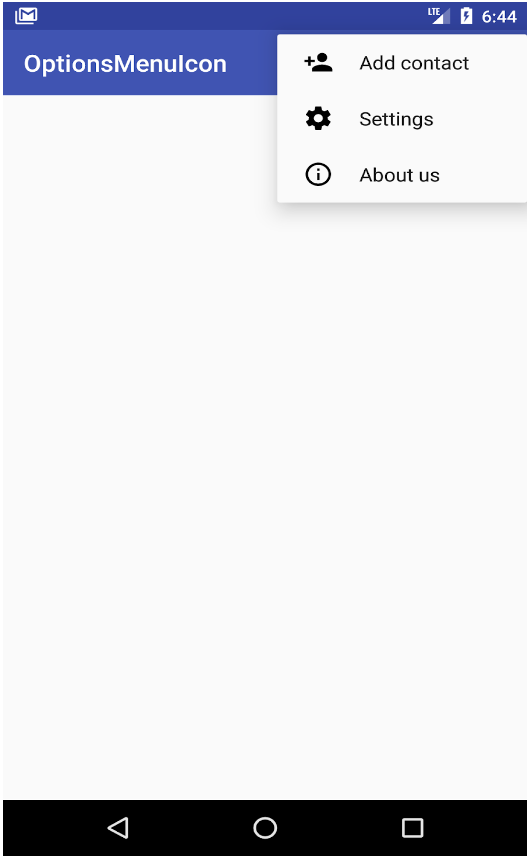
The menu is a part of the User Interface (UI) component, used to handle some common functionality around the app. To utilize the menu, you should define it in a separate XML file and use that file in your app based on your requirements. You can also use menu APIs to represent user actions and other options in your app activities.

**DIFFERENT TYPES OF MENUS**

Android provides three types of menus. They are as follows:

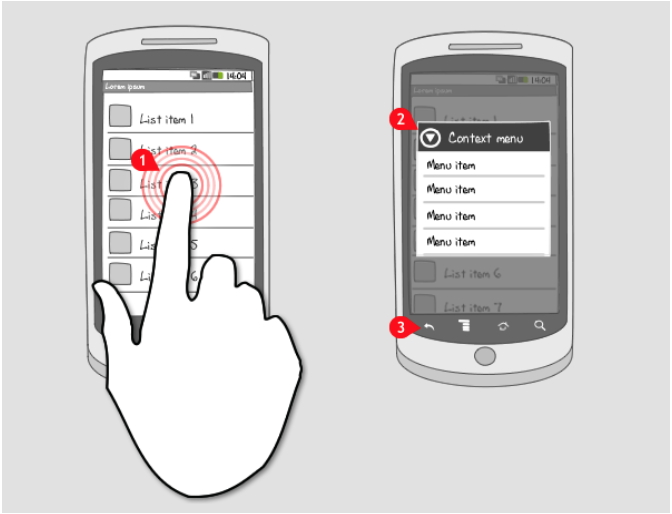
OPTION MENU

This type of menu is a primary collection of menu items in an app and is useful for actions that have a global impact on the searching app. The Option Menu can be used for settings, searching, deleting items, sharing, etc.



#### CONTEXT MENU

This type of menu is a floating menu that only appears when a user presses for a long time on an element and is useful for elements that affect the selected content or context frame.



#### POPUP MENU

Using Popup Menu we can display a list of items in a vertical list which presents the view that invokes the menu. Popup Menu is useful since it can provide an overflow of actions which are related to any specific content.



### HOW TO DEFINE A MENU IN THE XML FILE

In this step, we will write the menu’s code in an XML format to define the type of menu and its items. First, you should create a new menu folder inside of your project resource folder (res/menu) to define the menu. Add a new XML file (res/menu/file.xml) to build your menu. This XML (res/menu/file.xml) file can be given any name that you provide. There are the following important elements of a menu:

1. **<menu>**: A **<menu>** element defines a menu, which is a container for menu items that holds one or more elements. It must be the root of a file.
2. **<items>**: This element is used to create items in the menu. An **<items>** element can contain a nested **<menu>** element to create a submenu.
3. **<group>**: This element is an optional, invisible container for **<item>** elements. **<group>** allows categorizing menu items so they share properties such as active state and visibility.

#### EXAMPLE

Create a new Android project. We need to create a folder menu inside of your project resource directory and add a new XML file to build the menu.

**options\_menu.xml**

<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android" >

<item android:id="@+id/search\_item"

android:title="Search" />

<item android:id="@+id/upload\_item"

android:title="Upload" />

<item android:id="@+id/copy\_item"

android:title="Copy" />

<item android:id="@+id/print\_item"

android:title="Print" />

<item android:id="@+id/share\_item"

android:title="Share" />

<item android:id="@+id/bookmark\_item"

android:title="BookMark" />

</menu>

**MainActivity.java**

package com.example.optionsmenu;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater()

.inflate(R.menu.options\_menu, menu);

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

Toast.makeText(this, "Selected Item: " + item.getTitle(), Toast.LENGTH\_SHORT)

.show();

switch (item.getItemId()) {

case R.id.search\_item:

return true;

case R.id.upload\_item:

return true;

case R.id.copy\_item:

return true;

case R.id.print\_item:

return true;

case R.id.share\_item:

return true;

case R.id.bookmark\_item:

return true;

default:

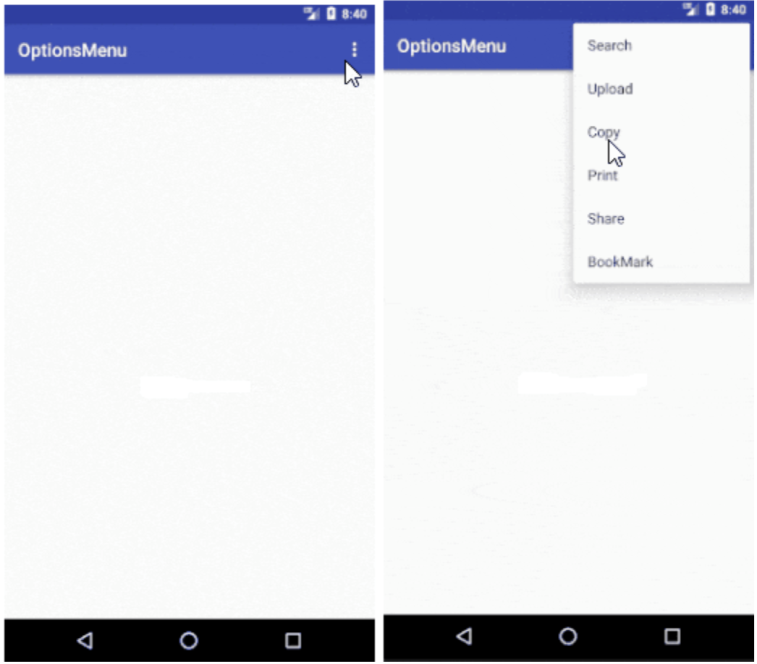
return super.onOptionsItemSelected(item);

}

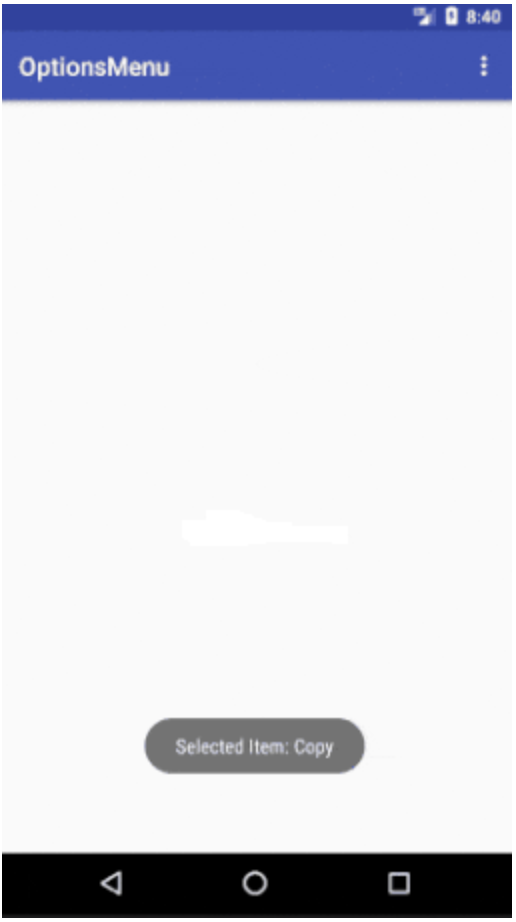
}

}

**Output**



Output after clicking on the copy menu item.



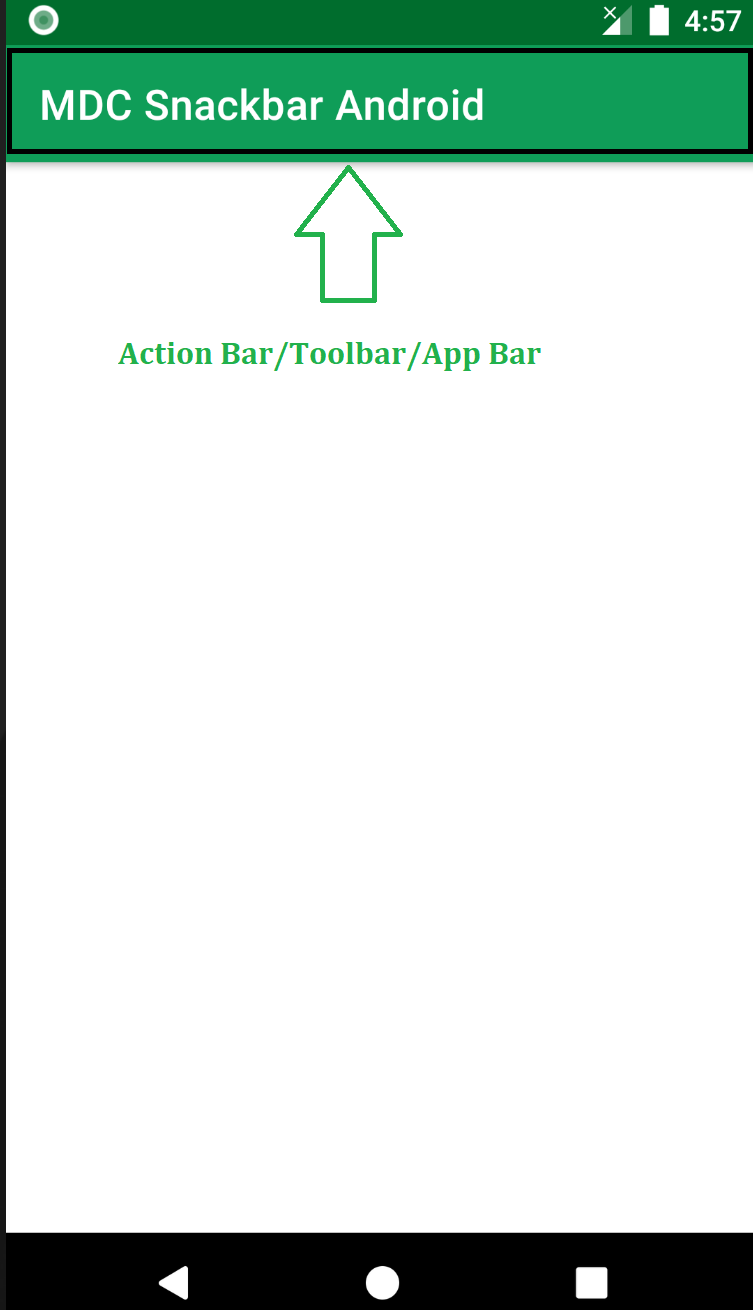
**3.3.2- Design the Android application using action bar.**

# How to Add and Customize Back Button of Action Bar in Android?

The action bar (sometimes referred to as the app bar), if it exists for an activity, will be at the top of the activity’s content area, typically directly underneath the status bar. It is a menu bar that runs across the top of the activity screen in android. Android ActionBar can contain menu items that become visible when the user clicks the “menu” button. In general, an ActionBar composed of the following four components:

* **App Icon:** App branding logo or icon will be shown here
* **View Control:** A dedicated space to display the Application title. Also provides the option to switch between views by adding spinner or tabbed navigation
* **Action Buttons:** Major actions of the app could be added here
* **Action Overflow:** All unimportant action will be displayed as a menu

Below is a sample image to show where the Action Bar/Toolbar/App Bar is present on an android device.



The action bar is a primary toolbar inside an activity that can be used to display an activity title and other interactive items. One of the most used items is a **Back Navigation Button**. The back button is used to move backward from the previously visited screen by the user. Most Android devices have a dedicated back button still a back button on the action bar enhances the user experience.

## Add Back Button in Action Bar

To create a new project in Android Studio please refer to [How to Create/Start a New Project in Android Studio](https://www.geeksforgeeks.org/android-how-to-create-start-a-new-project-in-android-studio/). There is no need to change anything in the **activity\_main.xml** file. The only file we have to work with is **Working with the MainActivity file.**

* Create action bar variable and call function **getSupportActionBar()**in the java/kotlin file.
* Show back button using **actionBar.setDisplayHomeAsUpEnabled(true)** this will enable the back button.
* Custom the back event at **onOptionsItemSelected**. This will enable the back function to the button on the press. See the below code for reference. We have provided both the **java and kotlin code for MainActivity**.

**//java code**

import android.os.Bundle;

import android.view.MenuItem;

import androidx.annotation.NonNull;

import androidx.appcompat.app.ActionBar;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// calling the action bar

ActionBar actionBar = getSupportActionBar();

// showing the back button in action bar

actionBar.setDisplayHomeAsUpEnabled(true);

}

// this event will enable the back

// function to the button on press

@Override

public boolean onOptionsItemSelected(@NonNull MenuItem item) {

switch (item.getItemId()) {

case android.R.id.home:

this.finish();

return true;

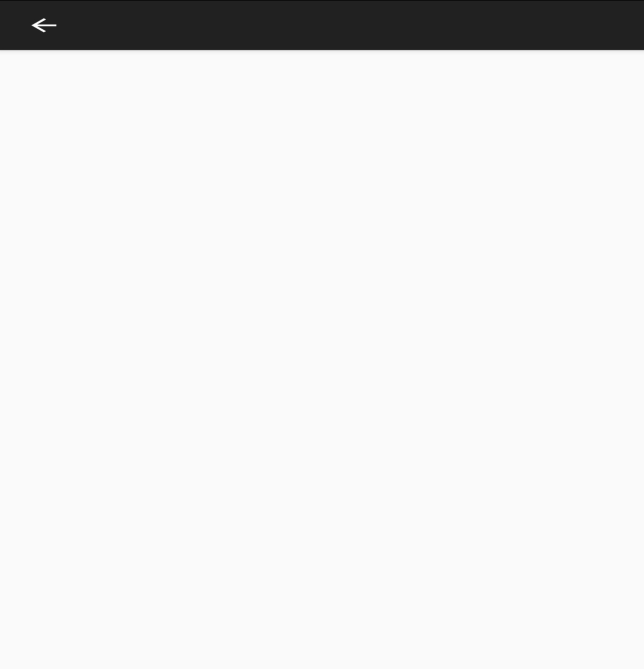
}

return super.onOptionsItemSelected(item);

}

}

### ****Output:****



## Customize Back Button in Action Bar

We can easily Customize the Back Button by using the **getSupportActionBar()** library and setting the drawable file using **setHomeAsUpIndicator** in the java/kotlin file.

*// Customize the back button*

*actionBar.setHomeAsUpIndicator(R.drawable.mybutton);*

The complete code is given below.

**//java code**

import android.os.Bundle;

import android.view.MenuItem;

import androidx.annotation.NonNull;

import androidx.appcompat.app.ActionBar;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// calling the action bar

ActionBar actionBar = getSupportActionBar();

// Customize the back button

actionBar.setHomeAsUpIndicator(R.drawable.mybutton);

// showing the back button in action bar

actionBar.setDisplayHomeAsUpEnabled(true);

}

// this event will enable the back

// function to the button on press

@Override

public boolean onOptionsItemSelected(@NonNull MenuItem item) {

switch (item.getItemId()) {

case android.R.id.home:

this.finish();

return true;

}

return super.onOptionsItemSelected(item);

}

}

### Output:

