Chapter 4

4. Android Menu

In android, **Menu** is a part of user interface (UI) component which is used to handle some common functionality around the application. By using Menus in our applications, we can provide better and consistent user experience throughout the application. Menus are useful for displaying additional options that are not directly visible on the main UI of an application.

We can use Menu APIs to represent user actions and other options in our android application activities. In android, we can define a Menu in separate XML file and use that file in our activities or fragments based on our requirements. In android, we can define a Menu in separate XML file and use that file in our activities or fragments based on our requirements.

Define Android Menu in XML File

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in our activity's code, we should define a menu and all its items in an XML menu resource and load menu resource as a Menu object in our activity or fragment.

In android, to define menu, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML file to build the menu with the following elements.

Element	Description
<menu>	It's a root element to define a Menu in XML file and it will hold one or more elements.
<item>	It is used to create a menu item and it represent a single item in menu. This element may contain a nested <menu> element in order to create a submenu.
<group>	It's an optional and invisible for <item> elements. It is used to categorize the menu items so they share properties such as active state and visibility.

Following is the example of defining a menu in XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/inbox"
        android:icon="@drawable/ic_inbox"
        android:title="@string/inbox" />
    <item android:id="@+id/compose"
        android:icon="@drawable/ic_compose"
        android:title="@string/compose"
        android:showAsAction="ifRoom" />
```

```
    <item android:id="@+id/outbox"
        android:icon="@drawable/ic_outbox"
        android:title="@string/outbox" />
</menu>
```

The **<item>** element in **menu** supports different type of attributes to define item's behaviour and appearance. Following are some of commonly used **<item>** attributes in android applications.

Attribute	Description
android:id	It is used to uniquely identify element in application.
android:icon	It is used to set the item's icon from drawable folder.
android:title	It is used to set the item's title
android:showAsAction	It is used to specify how the item should appear as an action item in the app bar.

In case if we want to add **submenu** in **menu** item, then we need to add a **<menu>** element as the child of an **<item>**. Following is the example of defining a submenu in menu item.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
        android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                android:title="@string/create_new" />
            <item android:id="@+id/open"
                android:title="@string/open" />
        </menu>
    </item>
</menu>
```

Load Android Menu from an Activity

Once we are done with creation of menu, we need to load the menu resource from our activity using **MenuInflater.inflate()** like as shown below.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
```

```
    inflater.inflate(R.menu.menu_example, menu);
}
```

If you observe above code we are calling our menu using MenuInflater.inflate() method in the form of **R.menu.menu_file_name**. Here our xml file name is **menu_example.xml** so we used file name **menu_example**.

Handle Android Menu Click Events

In android, we can handle a menu item click events using **ItemSelected()** event based on the menu type. Following is the example of handling a context menu item click event using **onContextItemSelected()**.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.mail:
            // do something
            return true;
        case R.id.share:
            // do something
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

If you observe above code, the **getItemId()** method will get the id of selected menu item based on that we can perform our actions.

Types of Menus

In android, we have three fundamental type of Menus available to define a set of options and actions in our android applications.

- Options Menu
- Context Menu
- Popup Menu

4.1. Option Menu

In android, **Options Menu** is a primary collection of menu items for an activity and it is useful to implement actions that have a global impact on the app, such as Settings, Search, etc.

By using Options Menu, we can combine multiple actions and other options that are relevant to our current activity. We can define items for the options menu from either our Activity or Fragment class.

In case, if we define items for the options menu in both activity or fragment, then those items will be combined and display in UI.

Create Android Options Menu in XML File

In android, to define **options menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**menu_example**) file to build the menu.

Following is the example of defining a menu in XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/inbox"
        android:icon="@drawable/ic_inbox"
        android:title="@string/inbox" />
    <item android:id="@+id/compose"
        android:icon="@drawable/ic_compose"
        android:title="@string/compose"
        android:showAsAction="ifRoom" />
    <item android:id="@+id/outbox"
        android:icon="@drawable/ic_outbox"
        android:title="@string/outbox" />
</menu>
```

Load Android Options Menu from an Activity

To specify the options menu for an activity, we need to override **onCreateOptionsMenu()** method and load the defined menu resource using **MenuInflater.inflate()** like as shown below.

```
@Override
public void onCreateOptionsMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_example, menu);
}
```

If you observe above code we are calling our menu using **MenuInflater.inflate()** method in the form of **R.menu.menu_file_name**. Here our xml file name is **menu_example.xml** so we used file name **menu_example**.

Handle Android Options Menu Click Events

In android, we can handle a options menu item click events using **onOptionsItemSelected()** event method.

Following is the example of handling a options menu item click event using **onOptionsItemSelected()**.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.inbox:
```

```
            // do something
            return true;
        case R.id.compose:
            // do something
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Android Options Menu Attributes

Following are some of commonly used attributes related to options menu control in android applications.

Attribute	Description
android:id	It is used to uniquely identify element in application.
android:icon	It is used to set the item's icon from drawable folder.
android:title	It is used to set the item's title
android:showAsAction	It is used to specify how the item should appear as an action item in the app bar.

Note: If you are using Android 3.0 +, the Options Menu won't support any item shortcuts and item icons in the menu.

Android Options Menu Example

Following is the example of implementing an **Options Menu** in android application.

Create a new android application using android studio and give names as **OptionsMenu**. In android, to define **options menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**options_menu.xml**) file to build the menu. Now open newly created xml (**options_menu.xml**) file and write the code like as shown below.

options_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/inbox_item"
        android:title="Inbox" />
    <item android:id="@+id/ compose_item"
        android:title="Compose" />
    <item android:id="@+id/outbox_item"
        android:title="Outbox" />
    <item android:id="@+id/spam_item"
        android:title="Spam" />
```

```
    <item android:id="@+id/logout_item"
        android:title="Logout" />
</menu>
```

Once we are done with creation of menu, we need to load this menu XML resource from our activity using **onCreateOptionsMenu()** callback method, for that open main activity file **MainActivity.java** from **\java\com.example.optionsmenu** path and write the code like as shown below.

MainActivity.java

```
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.inbox_item:
              // do your code
               return true;
            case R.id.compose_item:
               // do your code
```

```
            return true;
        case R.id.outbox_item:
            // do your code
            return true;
        case R.id.spam_item:
            // do your code
            return true;
        case R.id.logout_item:
            // do your code
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
  }
}
```

If you observe above code we are overriding **onCreateOptionsMenu()** method in activity to create options menu and loaded defined menu resource using **MenuInflater.inflate()**.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.

4.2. Context Menu

In android, **Context Menu** is a floating menu that appears when the user performs a long click on an element and it is useful to implement an action that effect the selected content or context frame.

The android Context Menu is more like the menu which displayed on right click in Windows or Linux.

In android, the Context Menu offers an actions that effect a specific item or context frame in the UI and we can provide a context menu for any view. The context menu won't support any item shortcuts and item icons.

Create Android Context Menu in Activity

The views which we used to show the context menu on long press, we need to register that views using **registerForContextMenu(View)** in our activity and we need to override **onCreateContextMenu()** in our activity or fragment. When the registered view receives a long click event, the system calls our **onCreateContextMenu()** method. By using **onCreateContextMenu()** method, we can create our menu items like as shown below.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);
    Button btn = (Button) findViewById(R.id.btnShow);
    registerForContextMenu(btn);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("Context Menu");
    menu.add(0, v.getId(), 0, "Inbox");
    menu.add(0, v.getId(), 0, "Outbox");
}
```

If you observe above code, we registered our Button control using **registerForContextMenu()** to show the context menu on button long-click and binding the Context Menu items using **onCreateContextMenu()** method.

Handle Android Context Menu Click Events

In android, we can handle a context menu item click events using **onContextItemSelected()** method.

Following is the example of handling a context menu item click event using **onContextItemSelected()** method.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    if (item.getTitle() == "Inbox") {
        // do your coding
    }
    else {
        return  false;
    }
    return true;
}
```

Note: If you are using Android 3.0 +, the Context Menu won't support any item shortcuts and item icons in the menu.

Android Context Menu Example

Following is the example of implementing a **Context Menu** in android application.

Create a new android application using android studio and give names as **ContextMenuExample**. Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

```
<?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="match_parent"
    android:orientation="vertical" >
```

```
    <Button
        android:id="@+id/btnShow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Long press me"
        android:layout_marginTop="200dp" android:layout_marginLeft="100dp"/>
</LinearLayout>
```

If you observe above code we created a one Button control in XML Layout file to show the context menu when we do long press on Button. Once we are done with creation of layout with required control, we need to load the XML layout resource from our activity **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.example.contextmenuexample** path and write the code like as shown below.

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button) findViewById(R.id.btnShow);
        registerForContextMenu(btn);
    }
    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        menu.setHeaderTitle("Context Menu");
        menu.add(0, v.getId(), 0, "Inbox");
        menu.add(0, v.getId(), 0, "Compose");
        menu.add(0, v.getId(), 0, "Outbox");
        menu.add(0, v.getId(), 0, "Logout");
    }
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
        return true;
    }
}
```

If you observe above code we are overriding **onCreateContextMenu()** method in activity to create context menu and registered view for context menu using **registerForContextMenu()**.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.

4.3. Popup Menu

In android, **Popup Menu** displays a list of items in a vertical list that's anchored to the view that invoked the menu and it's useful for providing an overflow of actions that related to specific content.

In android, the Popup Menu provides an overflow of actions that are related to specific content and the actions in popup menu won't affect the corresponding content. The popup menu won't support any item shortcuts and item icons.

In android, Popup menu is available with API level 11 (Android 3.0) and higher versions. If you are using Android 3.0 +, the Popup Menu won't support any item shortcuts and item icons in the menu.

Create Android Popup Menu in XML File

In android, to define **popup menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (menu_example) file to build the menu.

Following is the example of defining a menu in XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/inbox"
      android:icon="@drawable/ic_inbox"
      android:title="@string/inbox" />
   <item android:id="@+id/compose"
      android:icon="@drawable/ic_xompose"
      android:title="@string/compose"
      android:showAsAction="ifRoom" />
   <item android:id="@+id/outbox"
      android:icon="@drawable/ic_outbox"
      android:title="@string/outbox" />
</menu>
```

Once we are done with the creation of menu, we need to create a view element which anchored the menu.

```
<Button
   android:id="@+id/btnShow"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Show Popup Menu"
   android:onClick="showPopup" />
```

Now in our activity we need to implement showPopup method to show the popup menu.

Load Android Popup Menu from an Activity

To show the popup menu for the view, we need to instantiate **Popup** constructor and use **MenuInflater** to load the defined menu resource using **MenuInflater.inflate()** like as shown below.

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.menu_example, popup.getMenu());
    popup.show();
}
```

Handle Android Popup Menu Click Events

To perform an action when the user selects a menu item, we need to implement the **PopupMenu.OnMenuItemClickListener** interface and register it with our **PopupMenu** by calling **setOnMenuItemclickListener()**. When the user selects an item, the system calls the **onMenuItemClick()** callback in your interface.

Following is the example of handling a popup menu item click event using **onMenuItemClick()**.

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.inbox:
            inbox(item);
            return true;
        case R.id.compose:
            compose(item);
            return true;
        default:
            return false;
    }
}
```

Note: If you are using Android 3.0 +, the Popup Menu won't support any item shortcuts and item icons in the menu.

Android Popup Menu Example

Following is the example of implementing a **Popup Menu** in android application.

Create a new android application using android studio and give names as **PopupMenuExample**.

Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

```
<?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="match_parent"
   android:orientation="vertical" >

   <Button
      android:id="@+id/btnShow"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Show Popup Menu"
      android:layout_marginTop="200dp" android:layout_marginLeft="100dp"/>
</LinearLayout>
```

If you observe above code we created a one Button control in XML Layout file to show the popup menu when we click on Button. In android, to define **popup menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**popup_menu.xml**) file to build the menu. Now open newly created xml (**popup_menu.xml**) file and write the code like as shown below.

popup_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
   <item android:id="@+id/inbox_item"
      android:title="Inbox" />
   <item android:id="@+id/compose_item"
      android:title="Compose" />
   <item android:id="@+id/outbox_item"
      android:title="Outbox" />
   <item android:id="@+id/spam_item"
      android:title="Spam" />
   <item android:id="@+id/logout_item"
      android:title="Logout" />
  </menu>
```

Once we are done with creation of **menu**, we need to load this menu XML resource from our activity by instantiating a **Popup** constructor, for that open

main activity file **MainActivity.java** from **\java\com.example.popupmenuexample**path and write the code like as shown below.

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.PopupMenu;
import android.widget.Toast;
```

```
public
class MainActivity extends AppCompatActivity implements PopupMenu.OnMenuItemClickListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button) findViewById(R.id.btnShow);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                PopupMenu popup = new PopupMenu(MainActivity.this, v);
                popup.setOnMenuItemClickListener(MainActivity.this);
                popup.inflate(R.menu.popup_menu);
                popup.show();
            }
        });
    }
    @Override
    public boolean onMenuItemClick(MenuItem item) {
        Toast.makeText(this, "Selected Item: " +item.getTitle(), Toast.LENGTH_SHORT).show();
        switch (item.getItemId()) {
            case R.id.inbox_item:
                // do your code
                return true;
            case R.id.compose_item:
                // do your code
                return true;
            case R.id.outbox_item:
                // do your code
                return true;
            case R.id.spam_item:
                // do your code
                return true;
            case R.id.logout_item:
                // do your code
                return true;
            default:
                return false;
        }
    }
}
```

If you observe above code we are trying to show popup menu on Button click, loaded defined menu resource using **Popup.inflate()** and implement popup menu items click event.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.