Mobile Programming – BCA 6th Semester

Unit-5: UI Fragments, Menus and Dialogs

Unit -5
UI Fragments, Menus and Dialogs [6 HRS]
The need for UI flexibility, Introduction to fragments, Lifecycle of fragment, Creating a UI fragment, Creating a fragment class, Wiring widgets in fragment, Introduction to fragment manager, Difference between Activity and Fragments. Menus (Introduction, Types, Implementing menu in an application) Dialogs (Introduction, Creating a dialog fragment, Setting a dialog's content)

Introduction to Fragments:

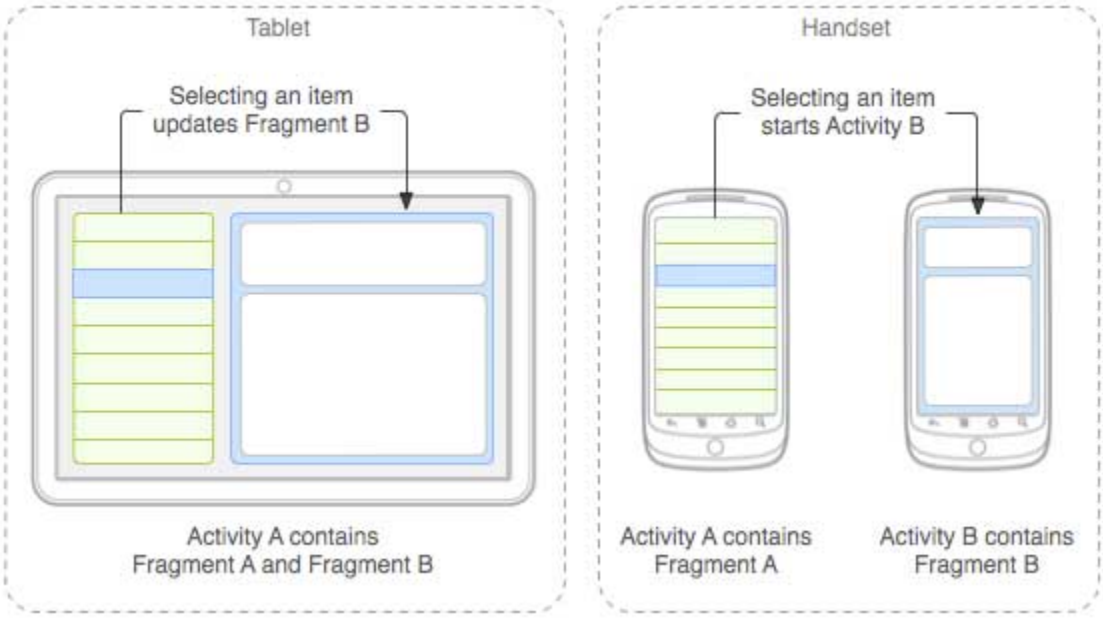
In Android, Fragment is a part of an activity which enables more modular activity design. It will not be wrong if we say a fragment is a kind of sub-activity. It represents a behavior or a portion of user interface in an Activity. We always need to embed Fragment in an activity and the fragment lifecycle is directly affected by the host activity's lifecycle.

Following are important points about fragment

- A fragment has its own layout and its own behavior with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused all the fragments available in the activity will also be stopped.
- A fragment can implement a behavior that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

You create fragments by extending **Fragment** class and you can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.

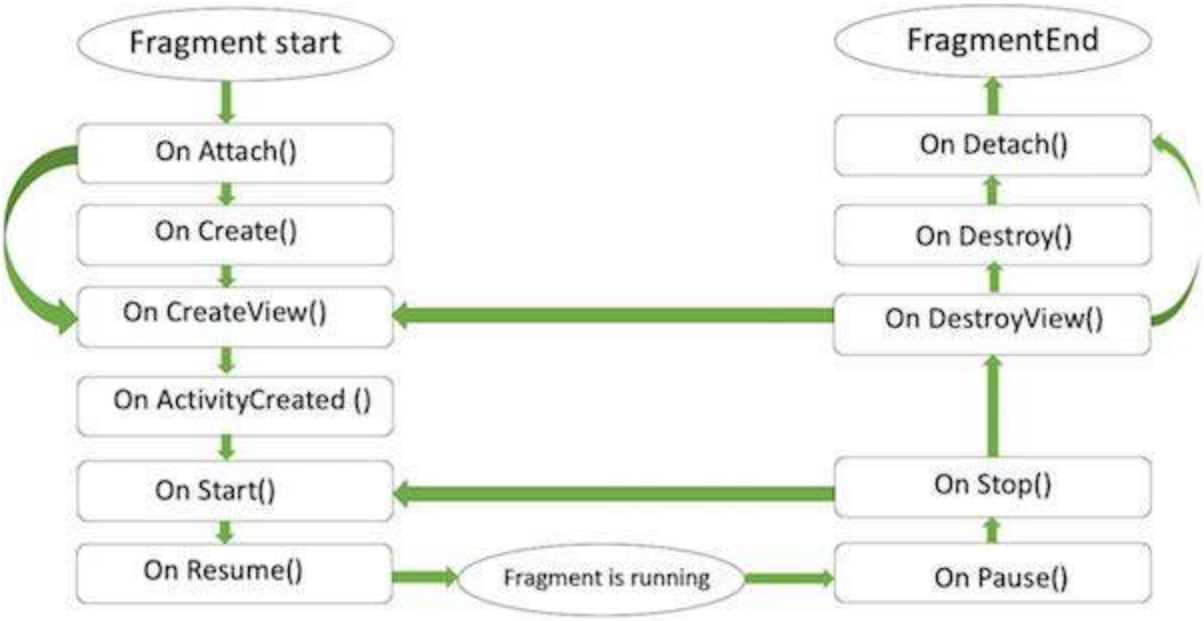
Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

Lifecycle of Fragment:

Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.



Lifecycle of fragment

Here is the list of methods which you can to override in your fragment class −

- **onAttach():** The fragment instance is associated with an activity instance. The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work.
- **onCreate():** The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- **onCreateView():** The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a **View** component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- **onActivityCreated():** The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method.

- **onStart():** The onStart() method is called once the fragment gets visible.
- **onResume():** Fragment becomes active.
- **onPause()**: The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.
- **onStop():** Fragment going to be stopped by calling onStop()
- **onDestroyView():** Fragment view will destroy after call this method
- **onDestroy():** onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.
- **onDetach():** Called when the fragment is no longer attached to its activity.

How to use Fragments?

This involves number of simple steps to create Fragments.

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Creating a UI fragment:

Fragment can create by following steps

1. Go to java folder
2. Right click and click on fragment
3. Click on blank fragment
4. Place fragment name

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <fragment
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:name="com.example.myapplication.AFragment"
        android:text="fragment1"
        android:layout_weight="1"
        android:id="@+id/frag1"
        />

    <fragment
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:name="com.example.myapplication.BFragment"
        android:text="fragment2"
        android:layout_weight="1"
        android:id="@+id/frag2"
        />

</LinearLayout>
```

MainActivity.java

```
package com.example.myapplication;
import android.os.Bundle;
import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;


public class MainActivity extends AppCompatActivity {
```

```
    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
}
```

fragment_a.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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=".AFragment">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:id="@+id/tv1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/tvone" />

</LinearLayout>
```

Afragment.java

```
package com.example.myapplication;
import android.os.Bundle;
import androidx.fragment.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;


public class AFragment extends Fragment {

    public AFragment() {
```

```
    // Required empty public constructor
  }

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    View v=inflater.inflate(R.layout.fragment_a, container, false);
    TextView tv=v.findViewById(R.id.frag1);
    return v;
  }
}
```

fragment_b.xml

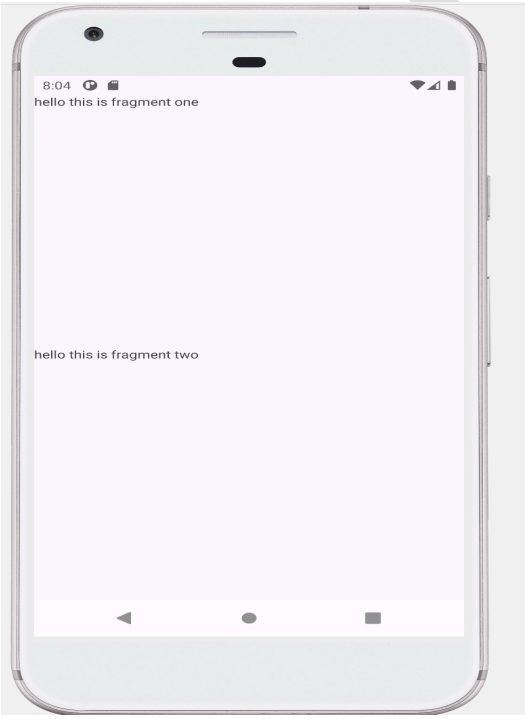
```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".BFragment">

  <!-- TODO: Update blank fragment layout -->
  <TextView
    android:id="@+id/tv2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text="@string/tvtwo" />

</FrameLayout>
```

Bfragment.java

```
package com.example.myapplication;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
```

```
public class BFragment extends Fragment {


    public BFragment() {
        // Required empty public constructor
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View v=inflater.inflate(R.layout.fragment_b, container, false);
        TextView tv=v.findViewById(R.id.frag2);
        return  v;
    }
}
```

Output:



Fragment Class

To create fragment extend the Fragment class, the override key lifecycle method to insert your app logic.

In the fragment we have to create the onCreateView() callback to define the layout.

fragment.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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"
    android:background="#ff0000"
    tools:context=".Red_fragment">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to Red Fragment"
        android:textColor="#ffffff"
        android:textSize="20dp"
        android:layout_gravity="center"/>
</FrameLayout>
```

Fragment.java

```
package com.example.androidpractice;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

```
public class Green_fragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_green_fragment, container, false);
    }
}
```

Wiring the widgets in Fragment:

We can place different widgets in the fragment as required. Following example shows the implementation of TextView widget in fragment.

First we have to create fragments and then call fragments in the activity.

red_fragment.xml

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

    <TextView
        android:id="@+id/redfrag"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to Red Fragment"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:layout_gravity="center"/>

</FrameLayout>
```

green_fragment.xml

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

    <TextView
        android:id="@+id/greenfrag"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to Green Fragment"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:layout_gravity="center"/>

</FrameLayout>
```

Redfragment.java

```
package com.example.myapplication;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class RedFragment extends Fragment {

    public RedFragment() {
        // 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_r, container, false);
    }
}
```

Green_fragment.java

```
package com.example.myapplication;

import android.os.Bundle;
```

```
import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class GreenFragment extends Fragment {

    public GreenFragment() {
        // 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_green, container, false);
    }
}
```

ActivityMain.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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Dynamic Layout"
        android:textStyle="italic"
        android:textSize="30dp"
        android:layout_gravity="center" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_marginTop="10dp"
        android:layout_gravity="center">
```

```
        <Button
            android:id="@+id/idred"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Red Fragment"
            android:layout_gravity="center" />

        <Button
            android:id="@+id/idgreen"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Green Fragment"
            android:layout_gravity="center" />
    </LinearLayout>

    <FrameLayout
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">

        <!-- Placeholder TextView -->
        <TextView
            android:id="@+id/placeholder_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Welcome to Dynamic Layout"
            android:textStyle="italic"
            android:textSize="30dp"
            android:layout_gravity="center" />
    </FrameLayout>
</LinearLayout>
```

MainActivity.java

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button btnRed = findViewById(R.id.idred);
        Button btnGreen = findViewById(R.id.idgreen);
        TextView placeholderText = findViewById(R.id.placeholder_text);

        btnRed.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                placeholderText.setVisibility(View.GONE); // Hide the placeholder text
                loadFragment(new RedFragment());
            }
        });

        btnGreen.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                placeholderText.setVisibility(View.GONE); // Hide the placeholder text
                loadFragment(new GreenFragment());
            }
        });
    }

    private void loadFragment(Fragment fragment) {
        // Create a FragmentManager
        FragmentManager fm = getSupportFragmentManager();

        // Create a FragmentTransaction to begin the transaction and replace the Fragment
        FragmentTransaction ft = fm.beginTransaction();

        // Replace the FrameLayout with the new Fragment
        ft.replace(R.id.fragment_container, fragment);
        ft.addToBackStack(null);

        // Commit the transaction
        ft.commit();
    }
}
```

output:


6:05
Dynamic Layout
Red Fragment
Green Fragment
Welcome to Dynamic Layout


6:05
Dynamic Layout
Red Fragment
Green Fragment
Welcome to Green Fragment

Introduction to Fragment Manager:

FragmentManager is the class responsible for performing actions on your app's fragments, such as adding, removing, or replacing them, and adding them to the back stack.

You might never interact with FragmentManager directly if you're using the Jetpack Navigation library, as it works with the FragmentManager on your behalf.

We can create object of FragmentManager as…

```
FragmentManager frg = getFragmentManager()
```

Accessing in an activity:

Every FragmentActivity and subclasses thereof, such as AppCompatActivity, have access to the FragmentManager through
the getSupportFragmentManager() method.

Accessing in a Fragment:

Fragments are also capable of hosting one or more child fragments. Inside a fragment, you can get a reference to the FragmentManager that manages the fragment's children through getChildFragmentManager(). If you need to access its host FragmentManager, you can use getParentFragmentManager().

Difference between Activity and Fragment

Activity	Fragment
Activity is an application component that gives a user interface where the user can interact.	The fragment is only part of an activity, it basically contributes its UI to that activity.
Activity is not dependent on fragment	Fragment is dependent on activity. It can't exist independently.
we need to mention all activity it in the manifest.xml file	Fragment is not required to mention in the manifest file
We can't create multi-screen UI without using fragment in an activity,	After using multiple fragments in a single activity, we can create a multi-screen UI.
Activity can exist without a Fragment	Fragment cannot be used without an Activity.
Creating a project using only Activity then it's difficult to manage	While Using fragments in the project, the project structure will be good and we can handle it easily.
Lifecycle methods are hosted by OS. The activity has its own life cycle.	Lifecycle methods in fragments are hosted by hosting the activity.

Activity	Fragment
Activity is not lite weight.	The fragment is the lite weight.

Menus:

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu to present user actions and other options in your activities.

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

Using a menu resource is a good practice for a few reasons:

- It's easier to visualize the menu structure in XML.
- It separates the content for the menu from your application's behavioral code.
- It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

Creating menu:

- To create the menu directory just right-click on res folder and navigate to **res->New->Android Resource Directory**. Give resource directory name as menu and resource type also menu. one directory will be created under res folder.
- To create xml resource file simply right-click on menu folder and navigate to New->Menu Resource File. Give name of file as menu_example. One menu_example.xml file will be created under menu folder.

menu_example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto">
   <item android:id="@+id/item1"
      android:icon="@drawable/ic_menu"
      android:title="Item 1"
      app:showAsAction="ifRoom">
   </item>
   <item android:id="@+id/item3"
      android:title="Item 1"
      app:showAsAction="never">
      <menu>
         <item android:id="@+id/subitem1"
            android:title="sub item 1">
         </item>
         <item android:id="@+id/subitem2"
            android:title="sub item 2">
         </item>
      </menu>
   </item>
</menu>
```

Menu elements:

<menu>

- Defines a Menu, which is a container for menu items. A <menu> element must be the root node for the file and can hold one or more <item> and <group> elements.

<item>

- Creates a MenuItem, which represents a single item in a menu. This element may contain a nested <menu> element in order to create a submenu.

<group>

- An optional, invisible container for <item> elements. It allows you to categorize menu items so they share properties such as active state and visibility.

Attributes

The <item> element supports several attributes you can use to define an item's appearance and behavior. The items in the above menu include the following attributes:

- **android:id**
  A resource ID that's unique to the item, which allows the application to recognize the item when the user selects it.
- **android:icon**
  A reference to a drawable to use as the item's icon.
- **android:title**
  A reference to a string to use as the item's title.
- **android:showAsAction**
  Specifies when and how this item should appear as an action item in the app bar.

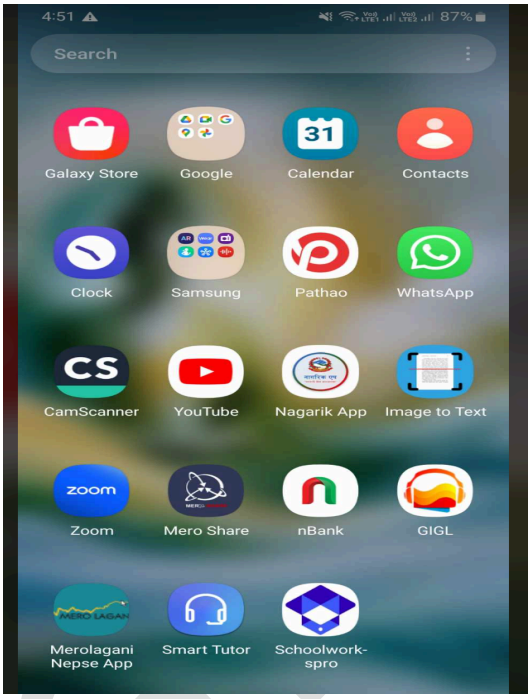
Types of Menus:

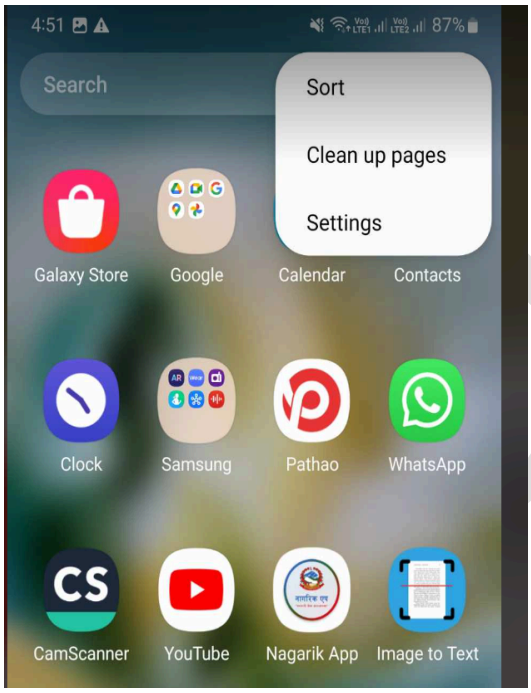
There are 3 types of menus in Android:

1. Option Menu
2. Context Menu
3. Pop-up Menu

Option Menu:

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a overall impact on the app, such as Search, Compose Email and Settings.

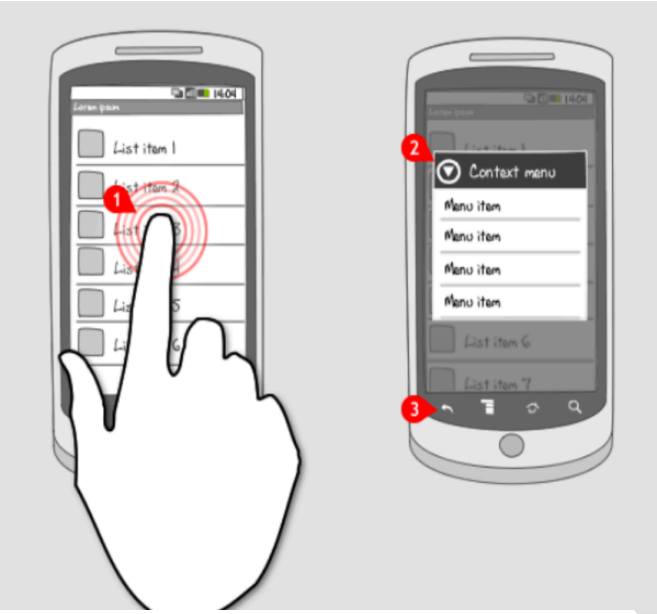




Context Menu:

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.

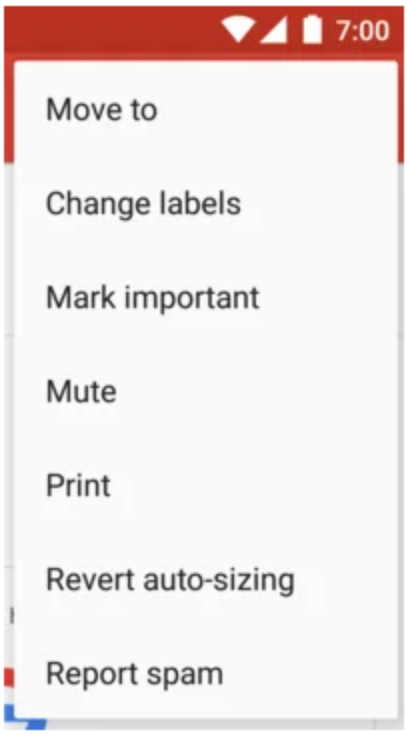
Teksan Gharti



Popup menu

A popup menu displays a list of items in a vertical list that is anchored(sticked) to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command.

Note: Actions in a popup menu should not directly affect the corresponding content that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.



Implementing Menu in Application:

Following codes shows the implementation of menus in application

mymenu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">


<item android:id="@+id/item1"
        android:icon="@drawable/ic_menu"
        android:title="Item 1"
        app:showAsAction="ifRoom">
    </item>


    <item android:id="@+id/item2"
        android:title="Item 1"
```

```
        app:showAsAction="never">
    </item>

    <item android:id="@+id/itm3"
        android:title="Item 3">
    </item>

    <item android:id="@+id/item3"
        android:title="Item 1"
        app:showAsAction="never">
        <menu>
            <item android:id="@+id/subitem1"
                android:title="sub item 1">
            </item>
            <item android:id="@+id/subitem2"
                android:title="sub item 2">
            </item>
        </menu>
    </item>
</menu>
```

Menu.java

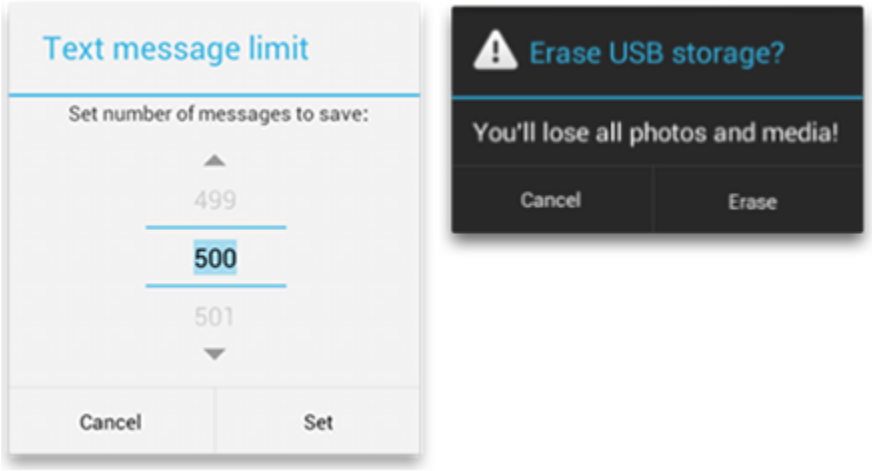
```
package com.example.androidpractice;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

import androidx.annotation.NonNull;
```

```
public class menu extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
   }
   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.mymenu,menu);
      return true;
   }
   @Override
   public boolean onOptionsItemSelected(@NonNull MenuItem item) {
      switch (item.getItemId())
      {
         case R.id.item1:
            Toast.makeText(this, "Item 1 selectid", Toast.LENGTH_SHORT).show();
            return true;
         case R.id.item2:
            Toast.makeText(this, "Item 2 selectid", Toast.LENGTH_SHORT).show();
            return true;
         case R.id.subitem1:
            Toast.makeText(this, "Sub Item 1 selectid",
Toast.LENGTH_SHORT).show();
            return true;
         case R.id.subitem2:
            Toast.makeText(this, "Sub Item 2 selectid",
Toast.LENGTH_SHORT).show();
            return true;
         default:
            return super.onOptionsItemSelected(item);
      }
```

```
    }
}
```

Dialogs:

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

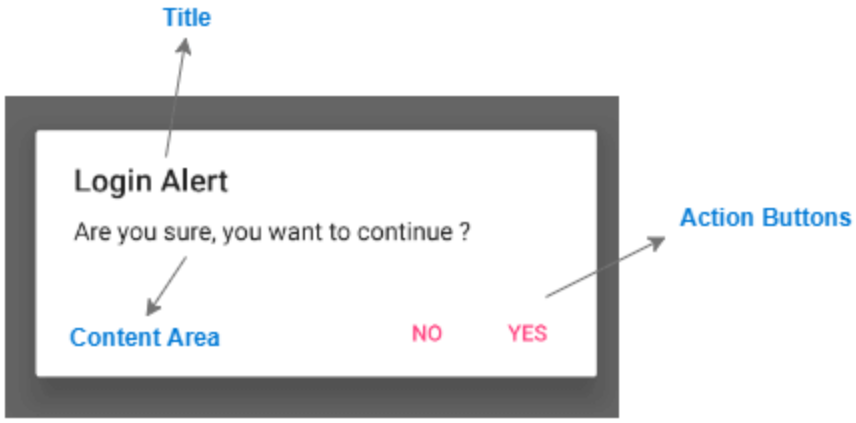


The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly.

There are different types of dialog in android, some of them are:

Alert Dialog

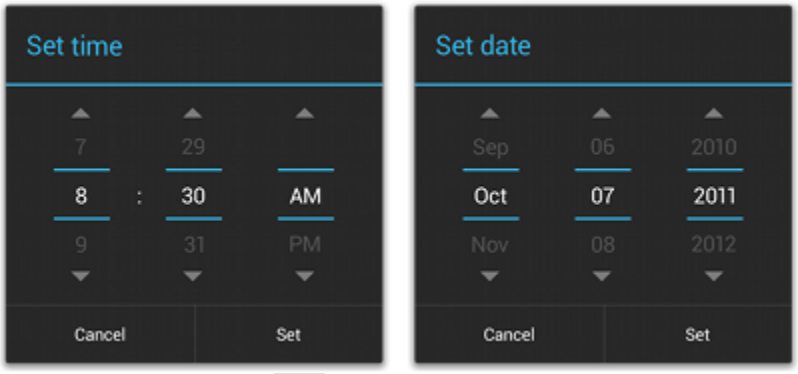
A dialog that can show a title, up to three buttons a list of selectable items, or a custom layout. The AlertDialog in an android application will contain three regions like as shown below.



DatePickerDialog or TimePickerDialog

A dialog with a pre-defined UI that allows the user to select a date or time.

These classes define the style and structure for your dialog, but you should use a DialogFragment as a container for your dialog. The DialogFragment class provides all the controls you need to create your dialog and manage its appearance, instead of calling methods on the Dialog object.



Using DialogFragment to manage the dialog ensures that it correctly handles lifecycle events such as when the user presses the Back button or rotates the screen. The DialogFragment class also allows you to reuse the dialog's UI as an embeddable component in a larger UI, just like a traditional Fragment (such as when you want the dialog UI to appear differently on large and small screens).

Alert Dialog

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Alertt dialog box demo"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="click here"
        android:id="@+id/btn1"
        />

</LinearLayout>
```

MainActivity.java

```
package com.example.myapplication;

import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
```

```
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class MainActivity extends AppCompatActivity {

   Button mybutton;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      mybutton=findViewById(R.id.btn1);
      mybutton.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            AlertDialog obj=builderDialog();
            obj.show();

         }
      });

   }

   AlertDialog builderDialog()
   {
      AlertDialog.Builder builder=new AlertDialog.Builder(this);
      builder.setMessage("Doyou want to delete?");

      builder.setPositiveButton("yes", new DialogInterface.OnClickListener() {
         @Override
         public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(MainActivity.this, "yes",
Toast.LENGTH_SHORT).show();
         }
      });

      builder.setNegativeButton("no", new DialogInterface.OnClickListener() {
```

```
        @Override
        public void onClick(DialogInterface dialog, int which) {
            Toast.makeText(MainActivity.this, "no",
Toast.LENGTH_SHORT).show();
        }
    });
    return builder.create();
  }

}
```

Output: