Unit -5
UI Fragments, Menus and Dialogs [6 Hrs]

Introduction to Fragments

Android Fragment is the part of activity, which is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity.

A Fragment represents a behaviour or a portion of user interface in a FragmentActivity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

A fragment must always be hosted in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle. For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments. However, while an activity is running (it is in the resumed lifecycle state), you can manipulate each fragment independently, such as add or remove them. When you perform such a fragment transaction, you can also add it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred. The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the Back button.

The Need for UI Flexibility

Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens, such as tablets. Because a tablet's screen is much larger than that of a handset, there's more room to combine and interchange UI components. Fragments allow such designs without the need for you to manage complex changes to the view hierarchy. By dividing the layout of an activity into fragments, you become able to modify the activity's appearance at runtime and preserve those changes in a back stack that's managed by the activity.

You should design each fragment as a modular and reusable activity component. That is, because each fragment defines its own layout and its own behavior with its own lifecycle callbacks, you can include one fragment in multiple activities, so you should design for reuse and avoid directly manipulating one fragment from another fragment. This is especially important because a modular fragment allows you to change your fragment combinations for different screen sizes. When designing your application to support both tablets and handsets, you can reuse your fragments in different layout configurations to optimize the user experience based on the available screen space. For example, on a handset, it might be necessary to separate fragments to provide a single-pane UI when more than one cannot fit within the same activity.

For example, a news application can use one fragment to show a list of articles on the left and another fragment to display an article on the right—both fragments appear in one activity, side by side, and each fragment has its own set of lifecycle callback methods and handle their own user input events. Thus, instead of using one activity to select an article and another activity to read the article, the user can select an article and read it all within the same activity, as illustrated in the tablet layout in figure 5-1.

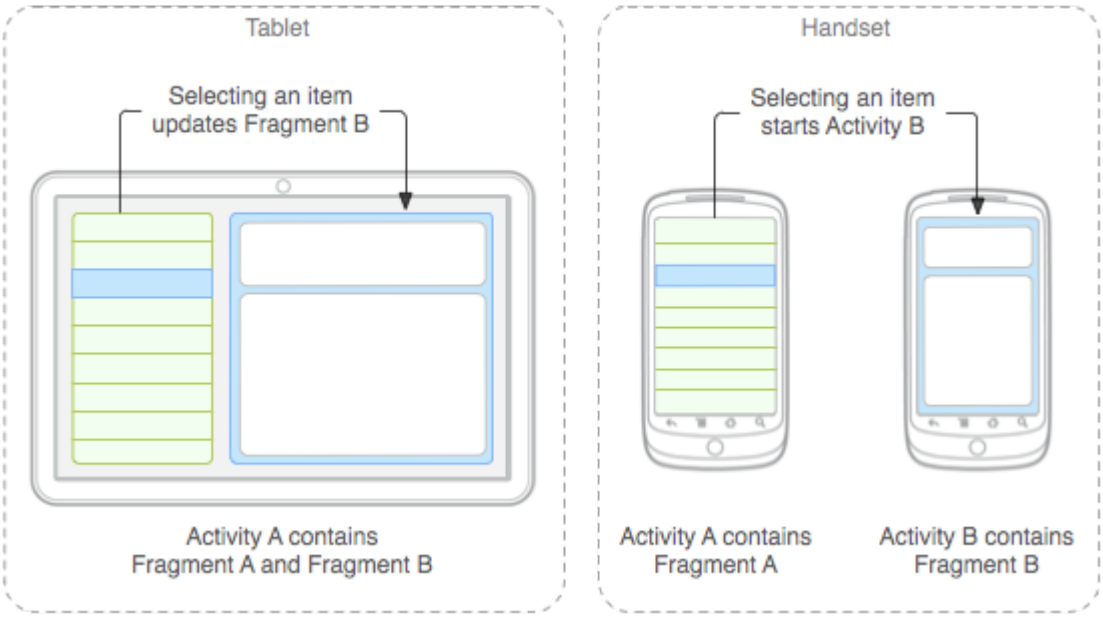


Figure 5-1. An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

For example—to continue with the news application example—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. Thus, the application supports both tablets and handsets by reusing fragments in different combinations, as illustrated in above figure.

Lifecycle of Fragment

The lifecycle of android fragment is like the activity lifecycle. There are 12 lifecycle methods for fragment. It is shown below:

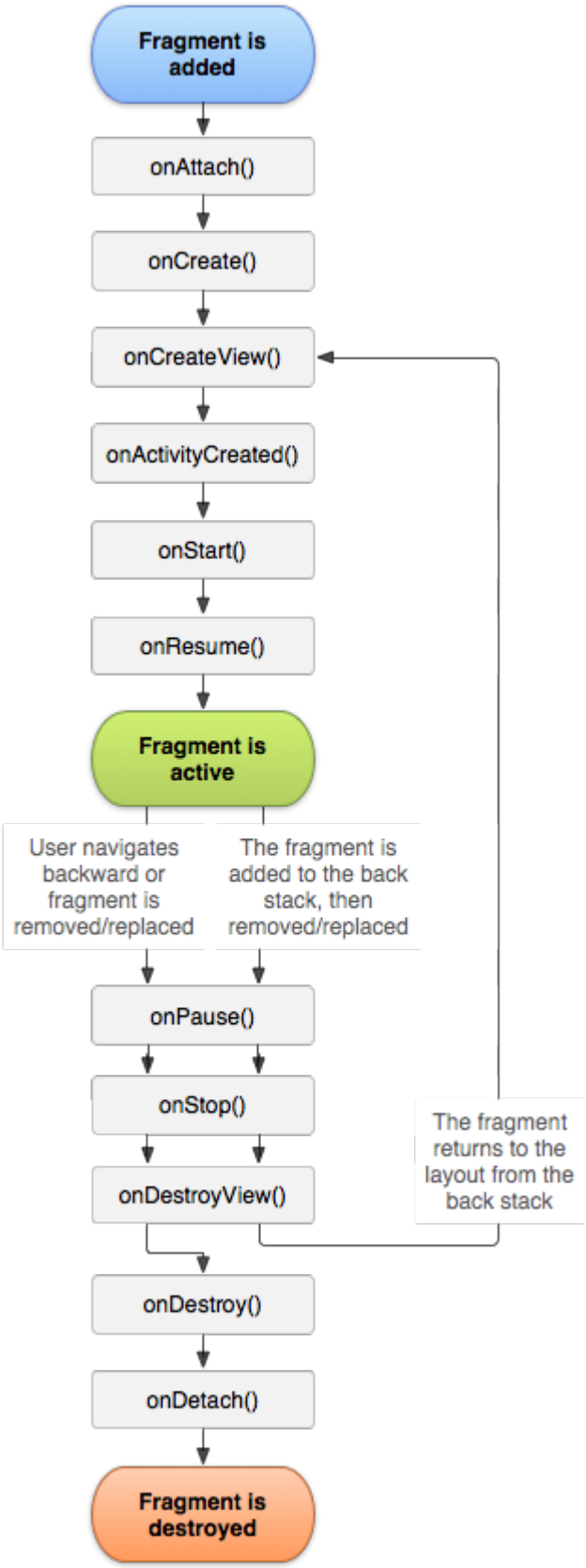


Figure 5-2. The lifecycle of a fragment (while its activity is running).

S.N.	Method	Description
1)	**onAttach(Activity)**	it is called only once when it is attached with activity.
2)	**onCreate(Bundle)**	It is used to initialize the fragment.
3)	**onCreateView(LayoutInflater, ViewGroup, Bundle)**	creates and returns view hierarchy.
4)	**onActivityCreated(Bundle)**	It is invoked after the completion of onCreate() method.
5)	**onViewStateRestored(Bundle)**	It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6)	**onStart()**	makes the fragment visible.
7)	**onResume()**	makes the fragment interactive.
8)	**onPause()**	is called when fragment is no longer interactive.
9)	**onStop()**	is called when fragment is no longer visible.
10)	**onDestroyView()**	allows the fragment to clean up resources.
11)	**onDestroy()**	allows the fragment to do final clean-up of fragment state.
12)	**onDetach()**	It is called immediately prior to the fragment no longer being associated with its activity.

Creating a UI Fragment

Fragment can be created in UI resource file as follows:

```
<fragment android:name="com.example.raazu.myapplication.Fragment1"
     android:id="@+id/headlines_fragment"
     android:layout_weight="1"
     android:layout_width="0dp"
     android:layout_height="match_parent" />
```

Creating a Fragment Class

To create a fragment, extend the Fragment class, then override key lifecycle methods to insert your app logic, similar to the way you would with an Activity class.

One difference when creating a Fragment is that you must use the onCreateView() callback to define the layout. In fact, this is the only callback you need in order to get a fragment running. For example, here's a simple fragment that specifies its own layout:

```
public class Fragment1 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
            container,Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container,
            false);
    }
}
```

Example of Fragment

Following example creates two fragments named as **Fragment1** and **Fragment2**. After creating fragments, we will add these two fragments in activity named as **FirstActivity** and display them.

Firstly, we start by creating two fragments,

fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#AEB6BF"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am inside Fragment 1"
        android:layout_gravity="center"
        android:textSize="20sp"
        android:layout_marginTop="20sp"
        android:textStyle="bold" />
</LinearLayout>
```

fragment2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#D0ECE7"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am inside Fragment 2"
        android:layout_gravity="center"
        android:textSize="20sp"
        android:layout_marginTop="20sp"
        android:textStyle="bold" />
</LinearLayout>
```

Fragment1.java

```
public class Fragment1 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
            container, Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view= inflater.inflate(R.layout.fragment1, container,
                false);
        return view;
    }
}
```

Fragment2.java

```
public class Fragment2 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
            container, Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view= inflater.inflate(R.layout.fragment2, container,
                false);
        return view;
    }
}
```

Now we are going to add these fragments to Activity.

first_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:weightSum="2"
    android:layout_height="match_parent">

    <fragment android:name="com.example.raazu.myapplication.Fragment1"
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <fragment android:name="com.example.raazu.myapplication.Fragment2"
        android:id="@+id/fragment2"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

FirstActivity.java

```
public class FirstActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_activity);
    }
}
```

Above code will produce following output:

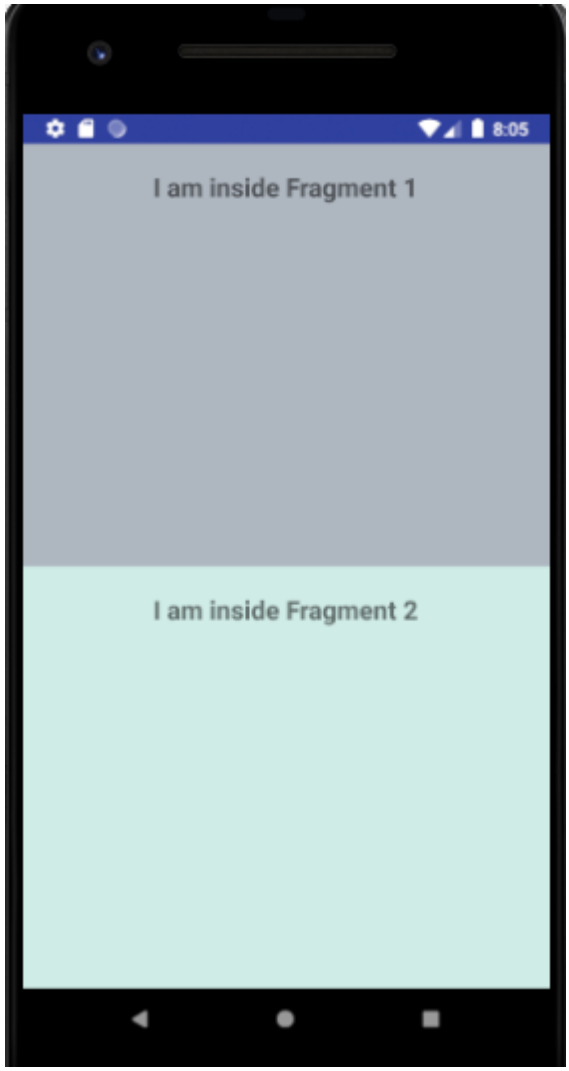


Figure 5-3. Output Demonstrating Fragments.

Wiring Widgets in Fragment

To demonstrate this topic, now I am going to create a fragment named as Fragment1 and add this fragment to Activity named as FirstActivity. Additionally, to show wiring of widgets inside fragment I will create UI for adding two numbers using two EditText, one Button and one TextView for displaying result.
So, Let's begin by creating fragment.

fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter first number"
        android:id="@+id/edtFirst" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter second number"
        android:id="@+id/edtSecond" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Caluclate"
        android:id="@+id/btnCalculate"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Result:"
        android:textSize="20sp"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:id="@+id/txtResult" />

</LinearLayout>
```

Fragment1.java

```
public class Fragment1 extends Fragment {
    EditText edtFirst,edtSecond;
    Button btnCalculate;
    TextView txtResult;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
            container, Bundle savedInstanceState) {
```

```
        // Inflate the layout for this fragment
        View view= inflater.inflate(R.layout.fragment1, container,
                false);

        //wiring up widgets
        edtFirst=view.findViewById(R.id.edtFirst);
        edtSecond=view.findViewById(R.id.edtSecond);
        btnCalculate=view.findViewById(R.id.btnCalculate);
        txtResult=view.findViewById(R.id.txtResult);

        btnCalculate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                int first,second, result;
                first=Integer.parseInt(edtFirst.getText().toString());
                second=Integer.parseInt(edtSecond.getText().toString());
                result=first+second;
                txtResult.setText("Result="+result);
            }
        });

        return view;
    }
}
```

Now add above fragment to Activity.

first_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent">

    <fragment android:name="com.example.raazu.myapplication.Fragment1"
        android:id="@+id/fragment1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

FirstActivity.java

```
public class FirstActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_activity);
    }
}
```

Above code will produce following output:

Figure 5-4. Output demonstrating wiring of widgets in Fragment.

Introduction to Fragment Manager

When the Fragment class was introduced in Honeycomb, the Activity class was changed to include a piece called the FragmentManager. The FragmentManager is responsible for managing your fragments and adding their views to the activity's view hierarchy.

We can use FragmentManager to manage fragments as follows:

FragmentManager fm = getFragmentManager();

The FragmentManager handles two things: a list of fragments and a back stack of fragment transactions. It is shown below:

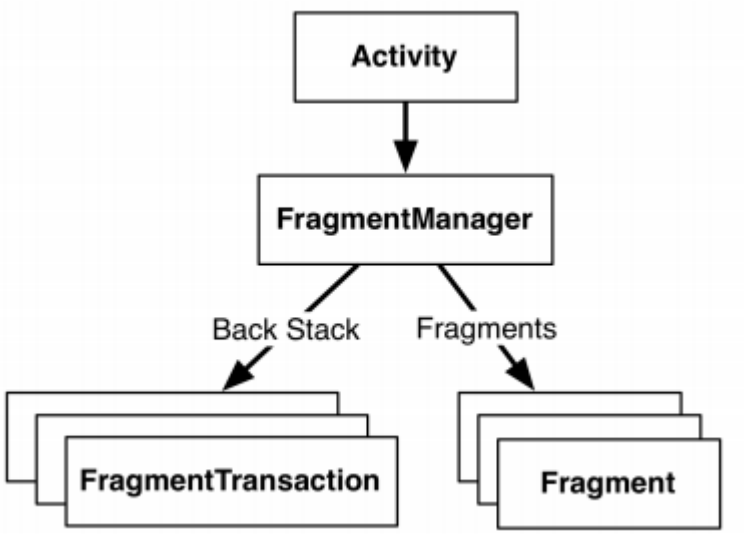


Figure 5-5. The FragmentManager

Fragment transactions are used to add, remove, attach, detach, or replace fragments in the fragment list. They are the heart of how you use fragments to compose and recompose screens at runtime. The FragmentManager maintains a back stack of fragment transactions that you can navigate.
The FragmentManager.beginTransaction() method creates and returns an instance of FragmentTransaction. The FragmentTransaction class uses a fluent interface - methods that configure FragmentTransaction return a FragmentTransaction instead of void, which allows you to chain them together. So the highlighted code in Listing 7.12 says, "**Create a new fragment transaction, include one add operation in it, and then commit it**."

The process is shown below:

```
Fragment fragment=new Fragment1();
FragmentManager manager = getFragmentManager();
FragmentTransaction transaction = manager.beginTransaction();
transaction.add(R.id.myfragment, fragment);
transaction.commit();
```

Following example will demonstrate the use of FragmentManager. Here, we are creating two Fragments and one Activity. Activity contains two buttons for switching fragments i.e, if first Button is clicked Fragment1 is displayed and if second Button is clicked Fragment2 is displayed on screen.
So, let's begin by creating two fragments.

fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#AEB6BF"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am inside Fragment 1"
        android:layout_gravity="center"
        android:textSize="20sp"
        android:layout_marginTop="20sp"
        android:textStyle="bold" />
</LinearLayout>
```

fragment2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#D0ECE7"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am inside Fragment 2"
        android:layout_gravity="center"
        android:textSize="20sp"
        android:layout_marginTop="20sp"
        android:textStyle="bold" />
</LinearLayout>
```

Fragment1.java

```
public class Fragment1 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
            container, Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view= inflater.inflate(R.layout.fragment1, container,
                false);
        return view;
    }
}
```

Fragment2.java

```
public class Fragment2 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
            container, Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        View view= inflater.inflate(R.layout.fragment2, container,
                false);
        return view;
    }
}
```

Now it's time to create Activity with two Buttons as follows.

first_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnFirst"
        android:text="Fragment 1" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnSecond"
        android:text="Fragment 2" />

    <fragment android:id="@+id/myfragment"
        android:name="com.example.raazu.myapplication.Fragment1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

FirstActivity.java

```
public class FirstActivity extends Activity {
    Button btnFirst,btnSecond;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_activity);

        btnFirst=findViewById(R.id.btnFirst);
        btnSecond=findViewById(R.id.btnSecond);

        btnFirst.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Fragment fragment=new Fragment1();
                FragmentManager manager = getFragmentManager();
                FragmentTransaction transaction =
                                        manager.beginTransaction();
                transaction.replace(R.id.myfragment, fragment);
                transaction.commit();
            }
        });

        btnSecond.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
```

```
                Fragment fragment=new Fragment2();
                FragmentManager manager = getFragmentManager();
                FragmentTransaction transaction =
                                    manager.beginTransaction();
                transaction.replace(R.id.myfragment, fragment);
                transaction.commit();
            }
        });
    }
}
```

Above code will produce following output:

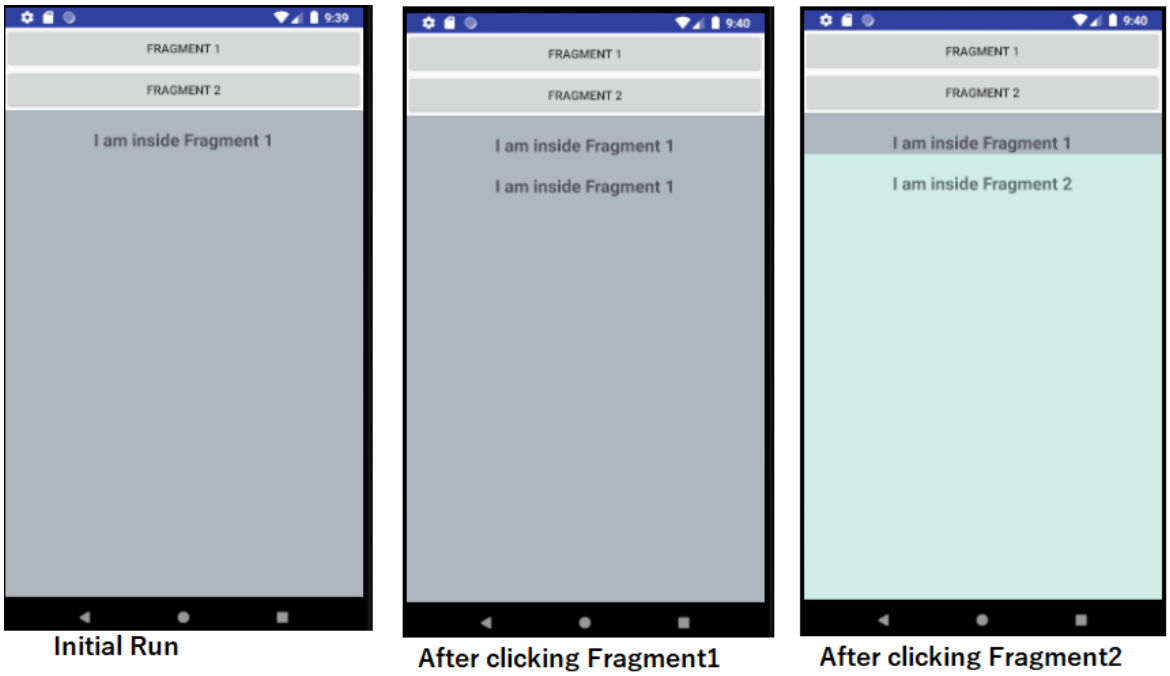
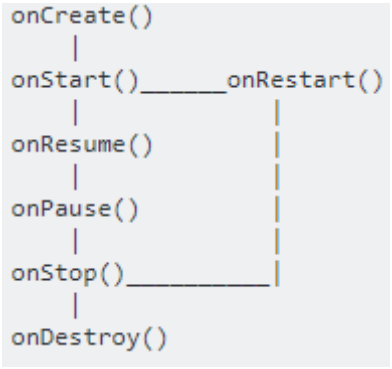
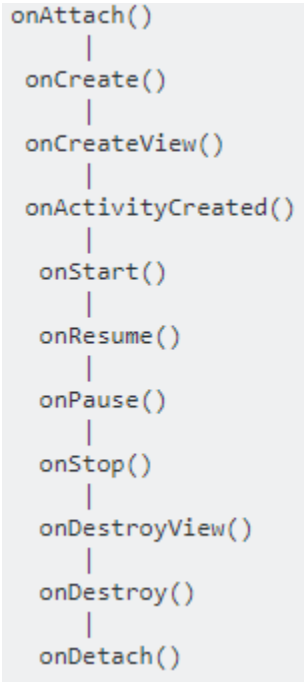


Figure 5-6. Output demonstrating FragmentManager

Difference between Activity and Fragment

Following are the major differences between Activity and Fragment:

S.N.	Activity	Fragment
1)	Activity is an application component that gives a user interface where the user can interact.	Fragment is a part of an activity, which contributes its own UI to that activity.
2)	Activity is not dependent on Fragment.	Fragment is dependent on Activity, it can't exist independently.
3)	Without using fragment in Activity we can't create multi-pane UI.	Using multiple fragments in a single activity we can create multi-pane UI.
4)	For Activity, we just need to mention in Manifest.	For fragment it's not required.
5)	Activity is heavy weight.	Fragment is light weight.

6)	Activity use a lot of memory.	Fragment doesn't use any memory because it resides on Activity.
7)	Activity is the UI of an application through which user can interact.	Fragment is the part of the Activity, it is an sub-Activity inside activity which has its own Life Cycle which runs parallel to the Activities Life Cycle.
8)	Lifecycle of Activity: 	Lifecycle of Fragment: 
9)	Lifecycle methods of Activity are hosted by OS.	Lifecycle methods of Fragments are hosted by are hosted by hosting activity.
10)	When an activity is placed to the back stack of activities the user can navigate back to the previous activity by just pressing the back button.	When an fragment is placed to the activity we have to request the instance to be saved by calling addToBackstack() during the fragment transaction.

Menus

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

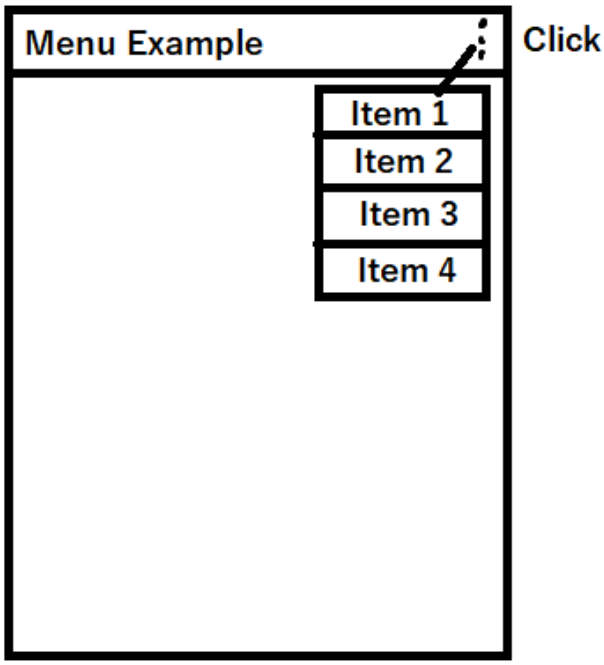


Figure 5-7. Example of Menu

Types of Menu

In Android there are three types of fundamental menus. They are options menu, contextual menu and popup menu.

Options Menu

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings." Menu shown in above diagram is an example of options menu.

Context Menu

Android context menu appears when user press long clicks on the element. It is also known as floating menu. It affects the selected content while doing action on it. It doesn't support item shortcuts and icons.

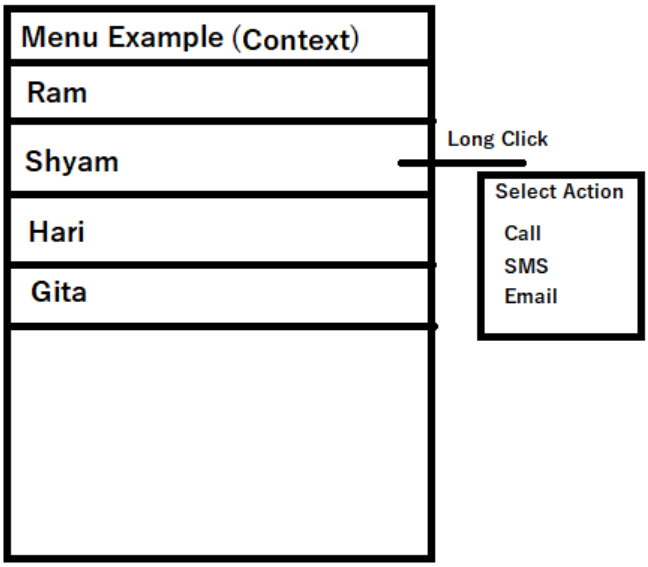


Figure 5-8. Example of Context Menu

Popup Menu

A popup menu displays a list of items in a vertical list that's anchored 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. 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.

Android Popup Menu displays the menu below the anchor text if space is available otherwise above the anchor text. It disappears if you click outside the popup menu.

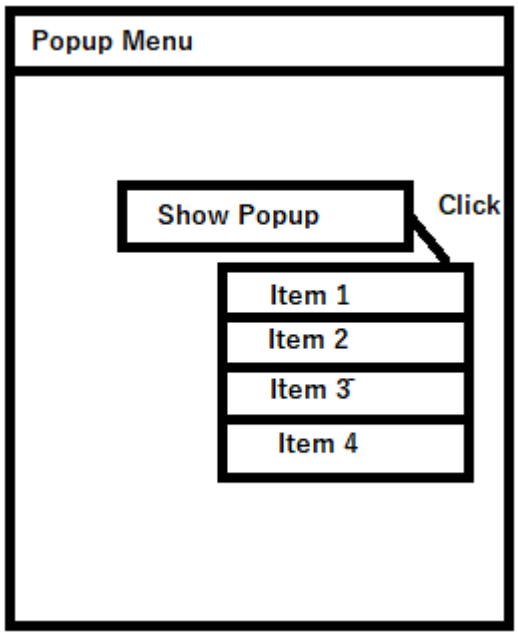


Figure 5-9. Example of Popup Menu

Implementing menu in an Application

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 behavioural code.
- It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

To define the menu, create an XML file inside your project's **res/menu/** directory and build the menu with the following 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 Menu Item, 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.

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:title="Item 1"/>

    <item  android:id="@+id/item2"
        android:title="Item 2"/>

    <item  android:id="@+id/item3"
        android:title="Item 3"
        app:showAsAction="withText"/>

</menu>
```

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.

Creating Options Menu

We can create options menu by overriding **onCreateOptionsMenu()** method. For handling clicks we must override **onOptionsItemSelected()** method. It is shown by the following example.

FirstActivity.java

```
public class FirstActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_activity);
    }

    //adding options menu
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.mymenu, menu);
        return true;
    }

    //handling clicks
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection
        switch (item.getItemId()) {
            case R.id.item1:
                //your stuffs
                return true;
            case R.id.item2:
                //your stuffs
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

first_activity.xml

```
<?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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World !"
        android:textSize="20sp"
        android:layout_centerInParent="true" />

</RelativeLayout>
```

Above code produces following output:

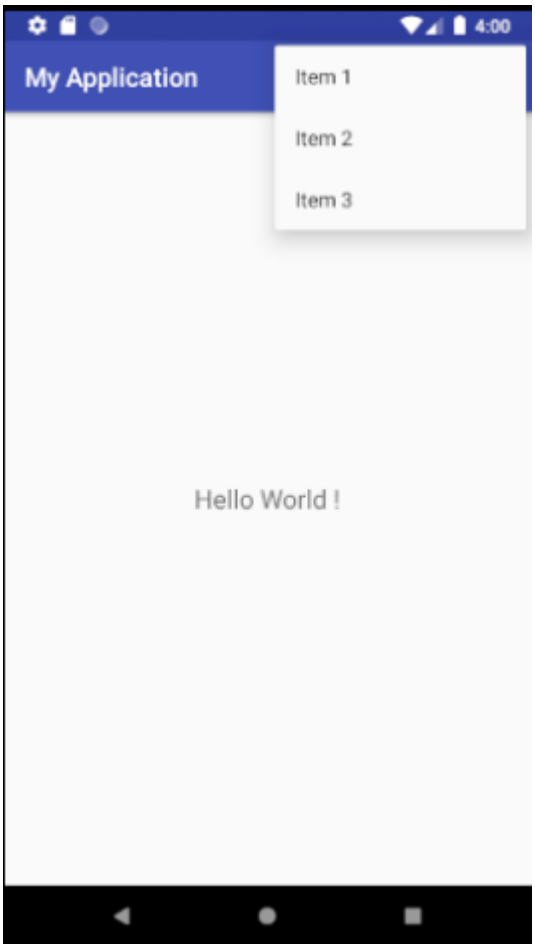


Figure 5-10. Output Demonstrating Options Menu

Creating Context Menu

To provide a floating context menu:

1. Register the View to which the context menu should be associated by calling **registerForContextMenu**() and pass it the View.
2. Implement the **onCreateContextMenu**() method in your Activity or Fragment.
3. For event handling on click implement **onContextItemSelected**().

FirstActivity.java

```
public class FirstActivity extends AppCompatActivity {
    Button btnClick;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_activity);

        btnClick=findViewById(R.id.btnClick);

        //registering view for context menu
        registerForContextMenu(btnClick);

    }

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

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.item1:
                //your stuffs
                return true;
            case R.id.item2:
                //your stuffs
                return true;
            default:
                return super.onContextItemSelected(item);
        }
    }
}
```

first_activity.xml

```
<?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">
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show context menu"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20sp"
        android:id="@+id/btnClick"
        android:textSize="20sp" />

</RelativeLayout>
```

Above code produces following output:

Figure 5-11. Output Demonstrating Context Menu

Creating Popup Menu

If you define your menu in XML, here's how you can show the popup menu:

- Instantiate a PopupMenu with its constructor, which takes the current application Context and the View to which the menu should be anchored.
- Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu().
- Call PopupMenu.show().

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

FirstActivity.java

```
public class FirstActivity extends Activity implements
PopupMenu.OnMenuItemClickListener{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_activity);
    }
    //showing popup menu
    public void showMenu(View v) {
        PopupMenu popup = new PopupMenu(this, v);
        // This activity implements OnMenuItemClickListener
        popup.setOnMenuItemClickListener(this);
        popup.inflate(R.menu.mymenu);
        popup.show();
    }
    //handling clicks
    @Override
    public boolean onMenuItemClick(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.item1:
                //your stuffs
                return true;
            case R.id.item2:
                //your stuffs
                return true;
            default:
                return false;
        }
    }
}
```

first_activity.xml

```
<?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">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show popup menu"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20sp"
        android:onClick="showMenu"
        android:textSize="20sp" />

</RelativeLayout>
```

Above code produces following output.

Figure 5-12. Output Demonstrating Popup Menu

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.

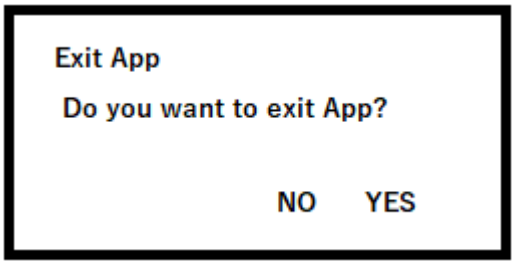


Figure 5-13. Example of a Dialog

Following are the different types of dialogs:

AlertDialog

- A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.

DatePickerDialog or TimePickerDialog

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

Custom Dialog

- A custom dialog built by programmer as per the requirement.

Creating a Dialog Fragment

We can create a wide variety of dialog designs—including custom layouts, by extending DialogFragment and creating a AlertDialog in the onCreateDialog() callback method. For example, here's a basic AlertDialog that's managed within a DialogFragment:

```
public class MyActivity extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the Builder class for convenient dialog construction
        AlertDialog.Builder builder = new AlertDialog.Builder
                                        (getActivity());
        builder.setMessage("Do you want to Exit?")
               .setPositiveButton("YES", new
                        DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        // your stuffs
                    }
               })
               .setNegativeButton("NO", new
                        DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        // User cancelled the dialog
                    }
               });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

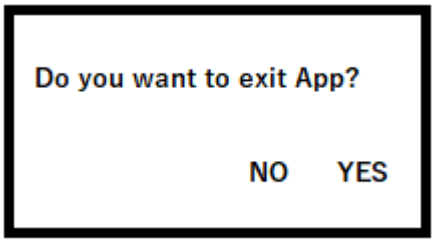


Figure 5-14. Output Demonstrating DialogFragment

Building an Alert Dialog

There are three regions of an alert dialog:

Title

This is optional and should be used only when the content area is occupied by a detailed message, a list, or custom layout. If you need to state a simple message or question (such as the dialog in figure 1), you don't need a title.

Content area

This can display a message, a list, or other custom layout.

Action buttons

There should be no more than three action buttons in a dialog.

FirstActivity.java

```
public class FirstActivity extends Activity{
    Button btnClick;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_activity);
        btnClick=findViewById(R.id.btnClick);

        btnClick.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                showDialog();
            }
        });

    }

    public void showDialog(){
        AlertDialog.Builder builder = new AlertDialog.Builder
                                (FirstActivity.this);
        builder.setTitle("Exit App");
        builder.setMessage("Do you want to exit App?");
        builder.setCancelable(true);

        builder.setPositiveButton(
                "Yes",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        //your stuffs
                    }
                });
        builder.setNegativeButton(
                "No",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        dialog.cancel();
                    }
                });
        AlertDialog alert = builder.create();
        alert.show();
    }
}
```

first_activity.xml

```
<?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">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show Dialog"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20sp"
        android:id="@+id/btnClick"
        android:textSize="20sp" />

</RelativeLayout>
```

Above code produces following output:

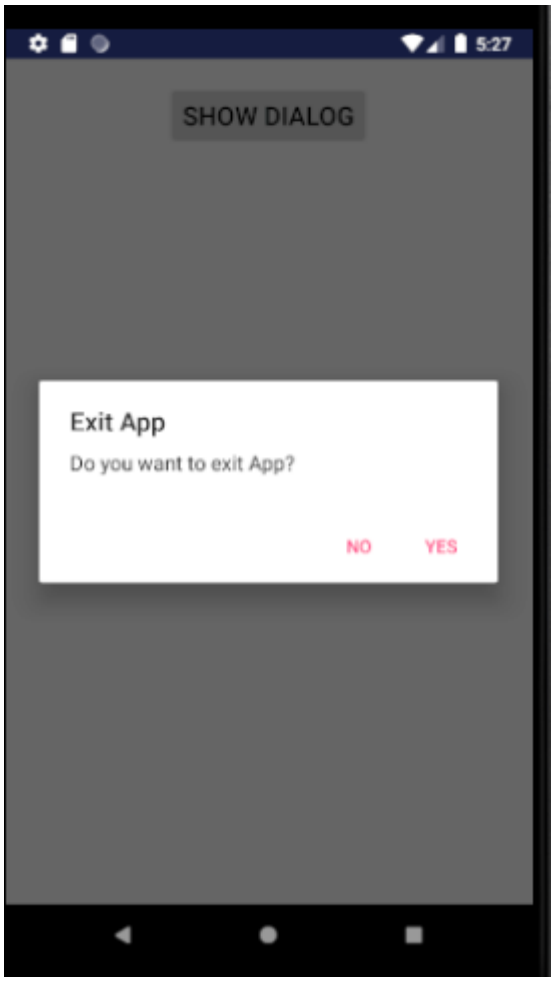


Figure 5-15. Output Demonstrating Alert Dialog

Building Custom Dialog and Setting Dialog Content

To demonstrate this topic, I am going to create a custom layout file for a dialog. This file contains two EditText for inputting two numbers and a Button. After clicking Button sum of these numbers will be displayed in a TextView.
So, let's begin by creating custom layout for dialog.

custom_dialog.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter first number"
        android:inputType="number"
        android:id="@+id/edtFirst" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter second number"
        android:inputType="number"
        android:id="@+id/edtSecond" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Caluclate"
        android:id="@+id/btnCalculate"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Result:"
        android:textSize="20sp"
        android:layout_gravity="center"
        android:layout_marginTop="20dp"
        android:id="@+id/txtResult" />

</LinearLayout>
```

first_activity.xml

```
<?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">

    <Button
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Show Dialog"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20sp"
        android:id="@+id/btnClick"
        android:textSize="20sp" />

</RelativeLayout>
```

FirstActivity.java

```
public class FirstActivity extends Activity{
    EditText edtFirst,edtSecond;
    Button btnClick,btnCalculate;
    TextView txtResult;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.first_activity);
        btnClick=findViewById(R.id.btnClick);

        btnClick.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                showDialog();
            }
        });

    }

    public void showDialog(){
        AlertDialog.Builder builder = new AlertDialog.Builder
                                        (FirstActivity.this);
        builder.setTitle("Calculate Sum");
        builder.setCancelable(true);

        // Inflate and set the layout for the dialog
        // Pass null as the parent view because it's going in the dialog
              layout
        LayoutInflater inflater = getLayoutInflater();
        View view=inflater.inflate(R.layout.custom_dialog, null);
        builder.setView(view);

        //wiring up widgets
        edtFirst=view.findViewById(R.id.edtFirst);
        edtSecond=view.findViewById(R.id.edtSecond);
        btnCalculate=view.findViewById(R.id.btnCalculate);
        txtResult=view.findViewById(R.id.txtResult);

        btnCalculate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                int first,second, result;
                first=Integer.parseInt(edtFirst.getText().toString());
                second=Integer.parseInt(edtSecond.getText().toString());
                result=first+second;
                txtResult.setText("Result="+result);

            }
        });
```

```
        AlertDialog alert = builder.create();
        alert.show();
    }
}
```

Above code produces following output:

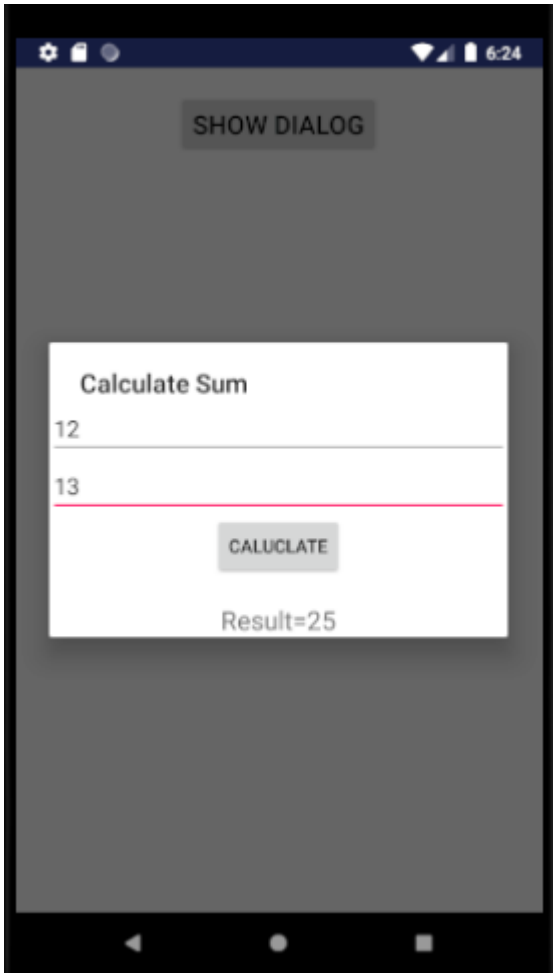


Figure 5-16. Output Demonstrating Custom Dialog

Exercise

1. What do you mean by fragment? Explain lifecycle of fragment in detail.
2. What do you mean by UI flexibility? Why UI needs to be flexible? Explain how fragments help to make UI flexible.
3. How can you create UI fragment? Explain with example.
4. How can you create fragment class? Explain with example.
5. Develop an android application to demonstrate fragments.
6. What do you mean by fragment manager? Explain with example.
7. Develop an android application to display multiple fragments in activity using fragment manager.
8. Develop an android application to calculate simple interest using fragment.

9. Develop an android application to calculate area and perimeter of rectangle. Your application must calculate and display area in one fragment and perimeter in another fragment.
10. Differentiate activity and fragment with example.
11. What do you mean by menu? Explain its types.
12. Develop an android application to demonstrate options menu.
13. Develop an android application to demonstrate context menu.
14. Develop an android application to demonstrate popup menu.
15. What do you mean by dialog box? Explain its types.
16. How can you create a dialog fragment? Explain with example.
17. Develop an android application to demonstrate alert dialog.
18. How can you open custom dialog on button click? Explain with example.
19. Develop an android application to calculate simple interest in a dialog.
20. Develop an android application to calculate area and perimeter of a rectangle in a dialog.