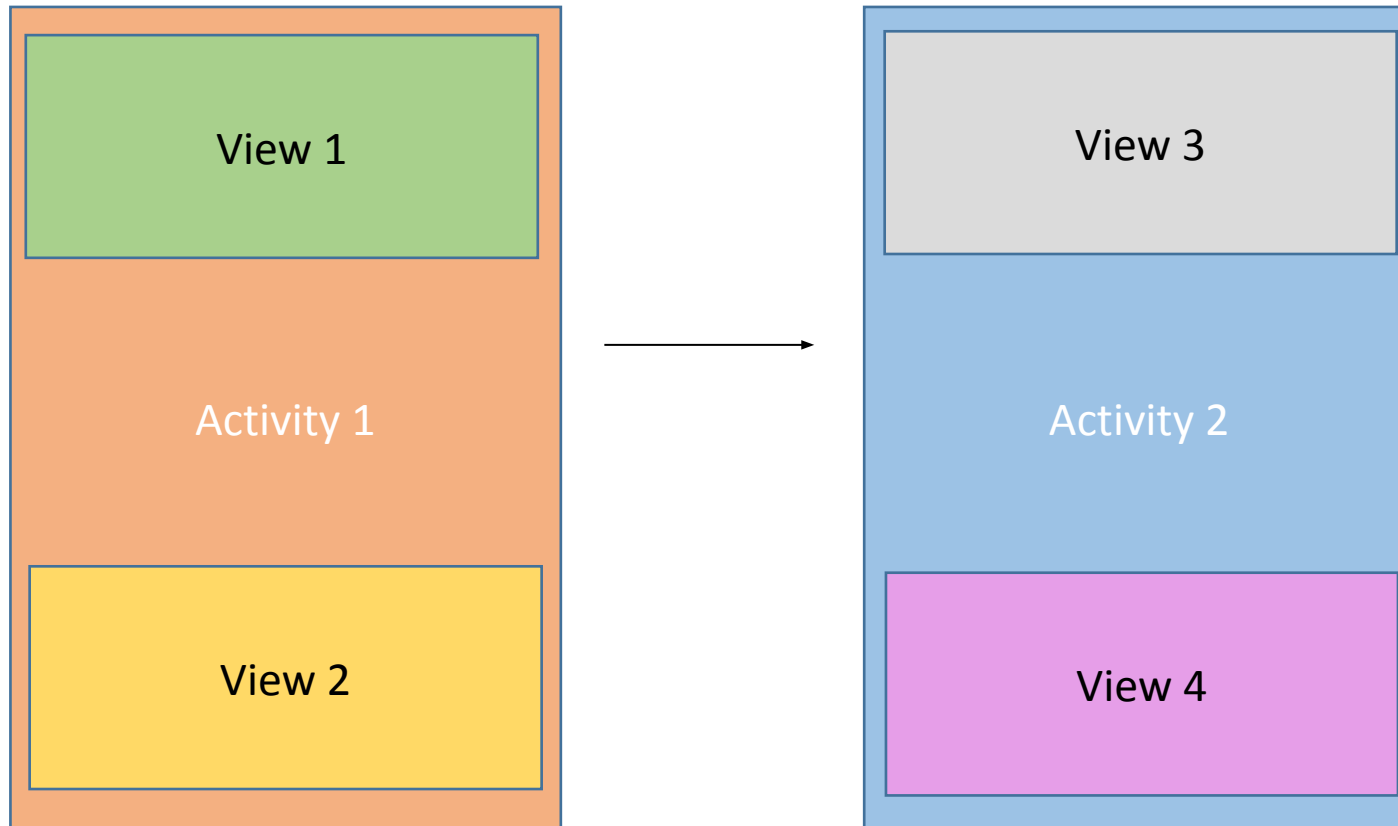


UI Programming - Fragment

What is a Fragment?



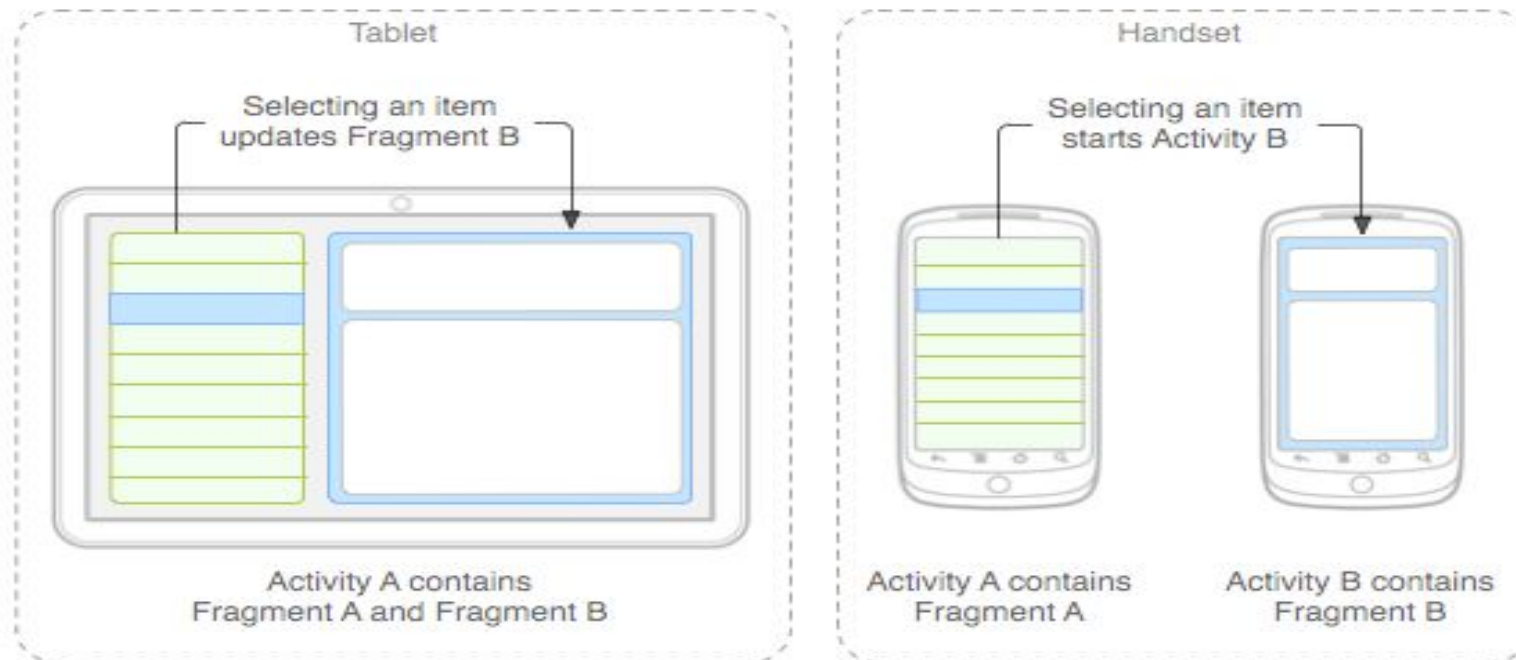
Before android 3.0 ,
If you have to show
View 3 and View 4
you have to
create new activity

Fragments

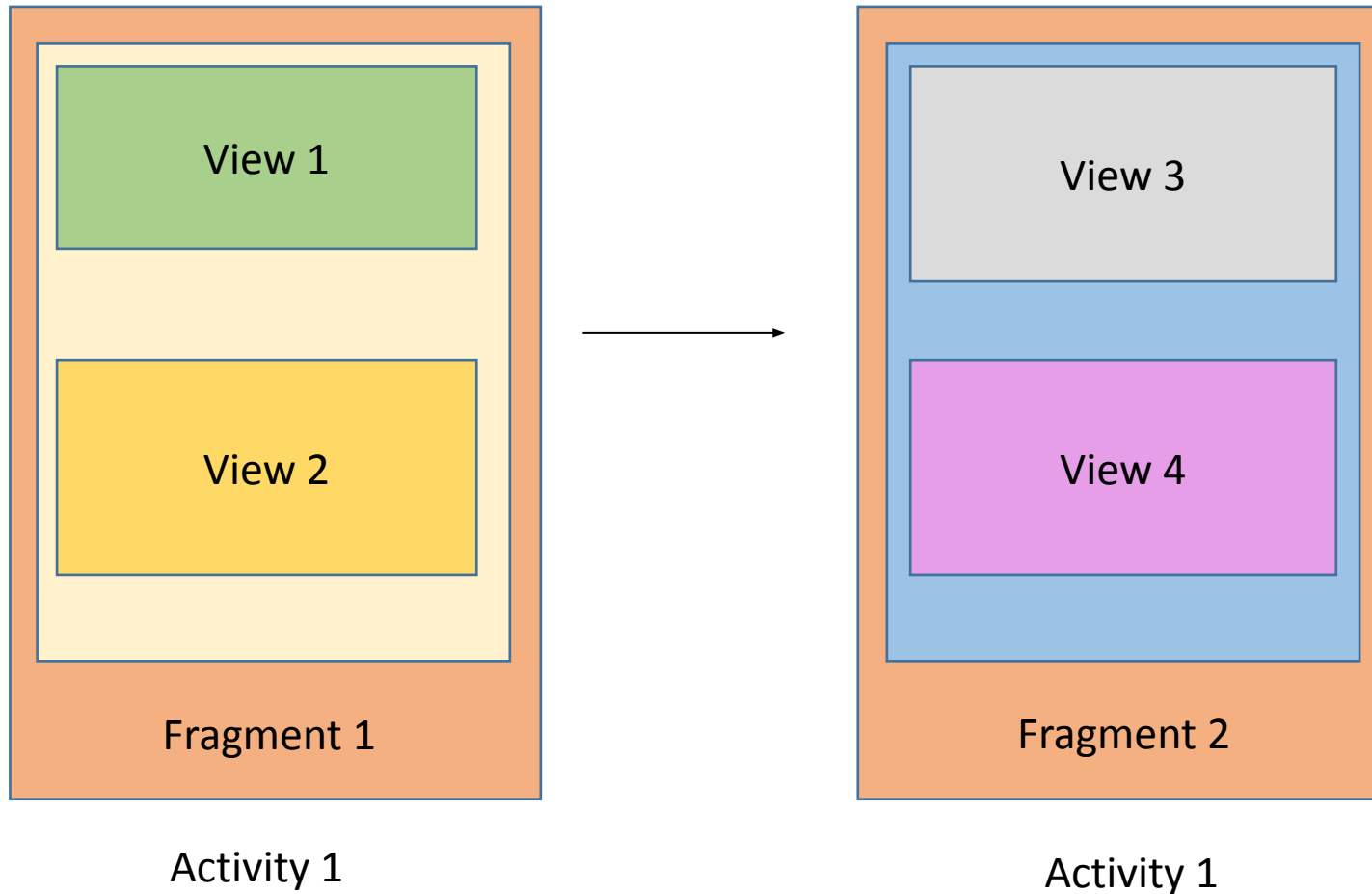
- A Fragment represents a behaviour or a portion of user interface in a `FragmentActivity`. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- You can think of a fragment as a modular section of an activity, which has its own 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.

Fragments

- Mini-activities, each with its own set of views
- One or more fragments can be embedded in an Activity
- You can do this dynamically as a function of the device type (tablet or not) or orientation



What is a Fragment?



Now, just insert fragment or delete it and insert another fragment.

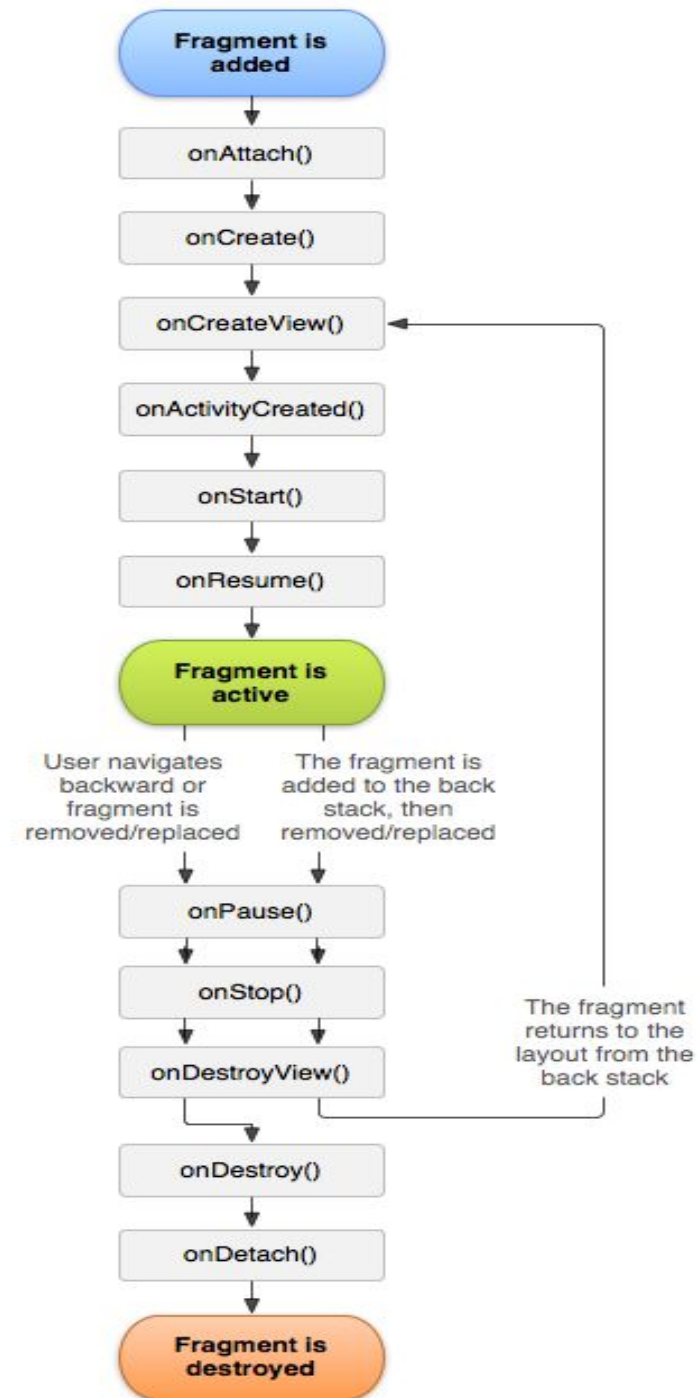
Fragments are so much customizable.

Fragments

- 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.

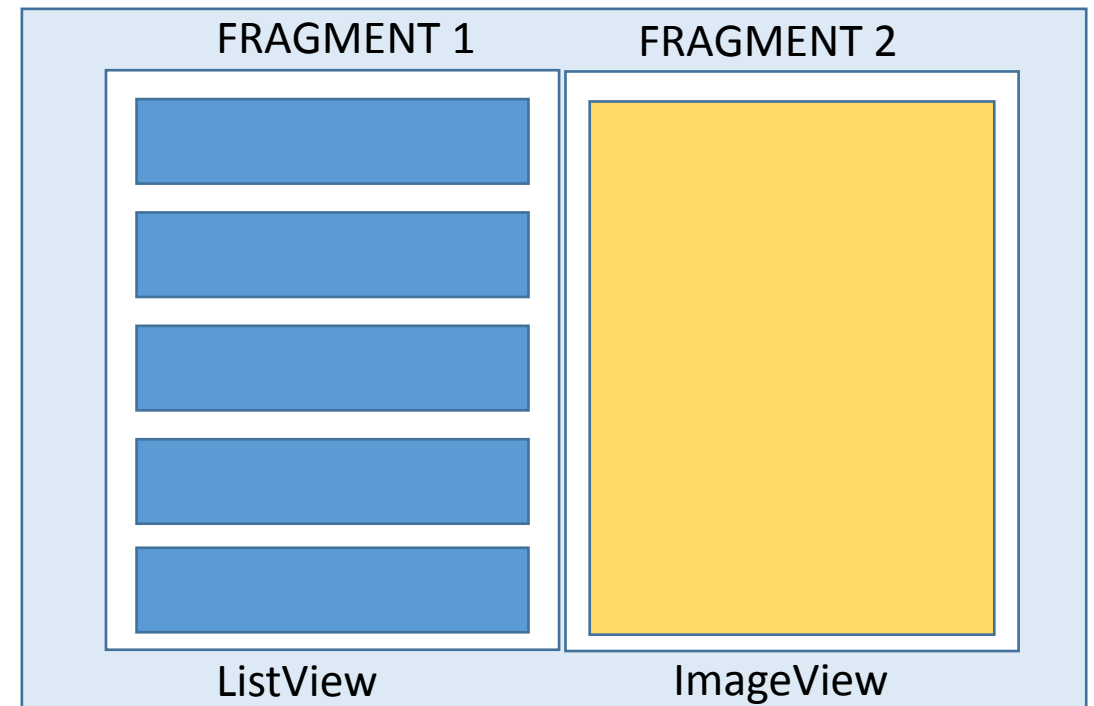
Fragment Lifecycle

- Fragment in an Activity---Activity Lifecycle influences
 - Activity paused ☐ all its fragments paused
 - Activity destroyed ☐ all its fragments paused
 - Activity running ☐ manipulate each fragment independently.
- Fragment transaction ☐ add, remove, etc.
 - adds 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**.



- Fragment is a chunk of UI.
- It has its own Lifecycle.
- It can process its own events
- It can be added or removed while the Activity runs.
- It was introduced in Honeycomb API 11.
- You can use Fragments on older devices using a Support Library from 1.6 to 2.3

ACTIVITY



Why do you need Fragments ?

- Combine Several Fragments in One Activity
- Reuse the same Fragment across several Activity
- Make better use of larger screen space on tablets

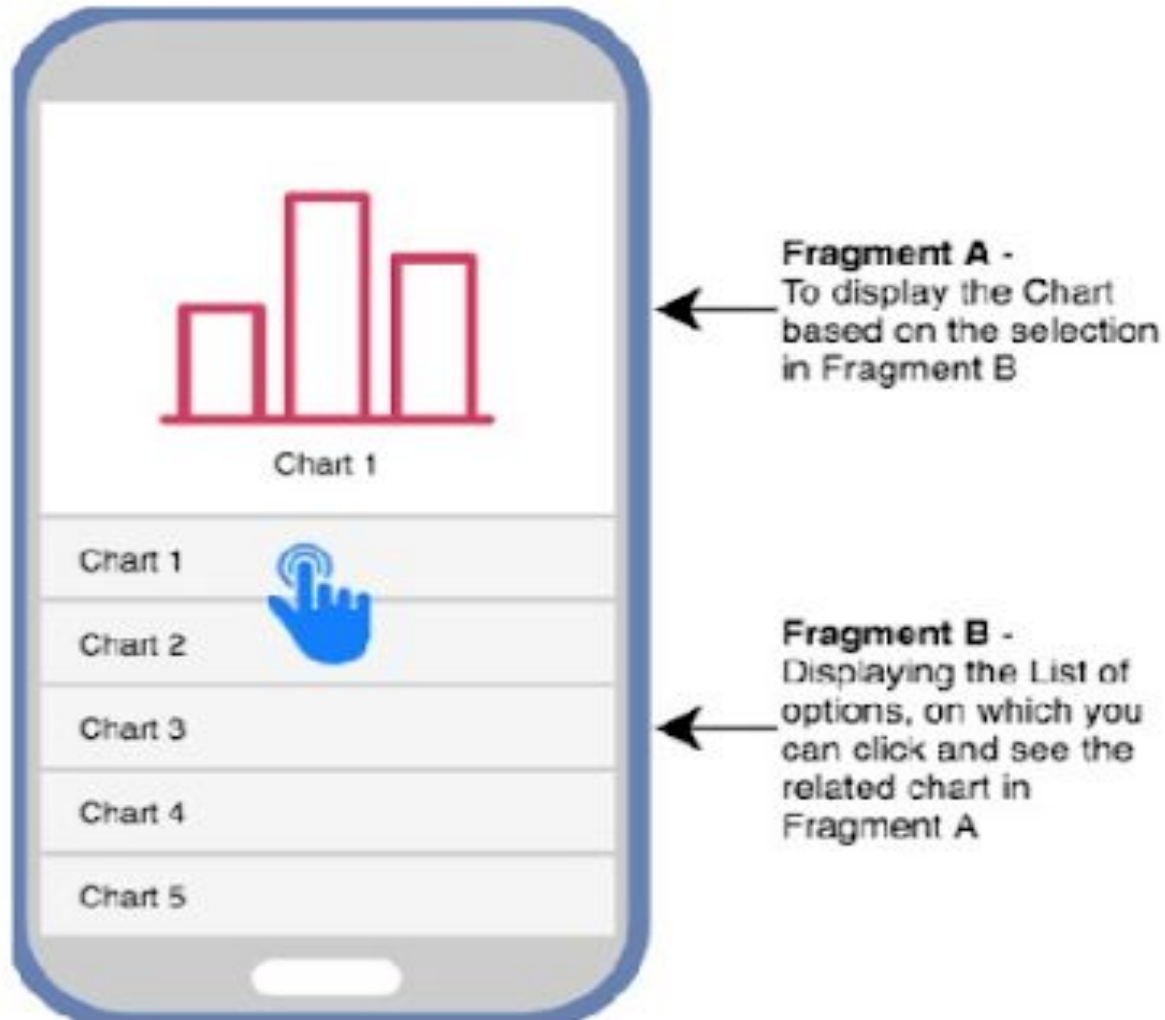
Uses of Fragments?

- Flexible user interfaces across different screen sizes
- Fixed/Scrolling/Swipe tab displays
- Dialog boxes

Advantages

- **Modularity:** divide it into independent fragments, hence making the code more organized and easier to maintain.
- **Reusability:** If we define any particular feature in a fragment, then that feature becomes a reusable component which can be easily integrated into any activity.
- **Adaptability:** If we break UI components of an app screen into fragments, then it becomes easier to change their orientation and placement, based on screen size etc.

Example



How to make a Fragment ?

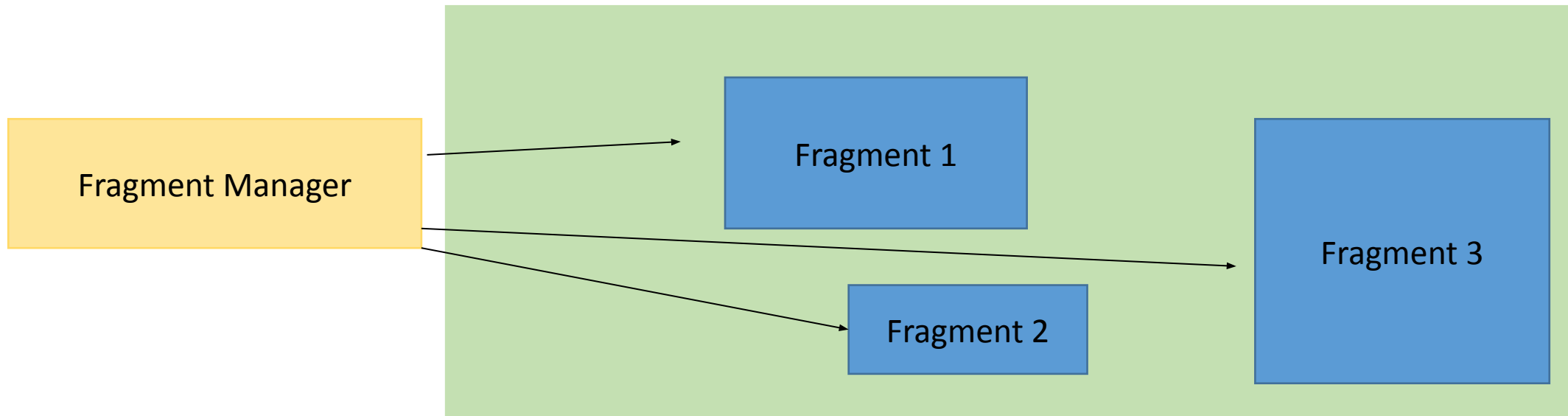
- Extend Fragment class
- Provide apperarance in XML/Java
- Override **onCreateView** to link the appearance.
- Use the Fragment in XML/Java(Dynamically).

The Fragment Manager

Every Activity has its own Fragment Manager
Accessible through **getFragmentManager()**

It maintains references to all fragments inside the Activity

Use **findFragmentById()** or **findFragmentByTag()** to get reference to a particular Fragment.



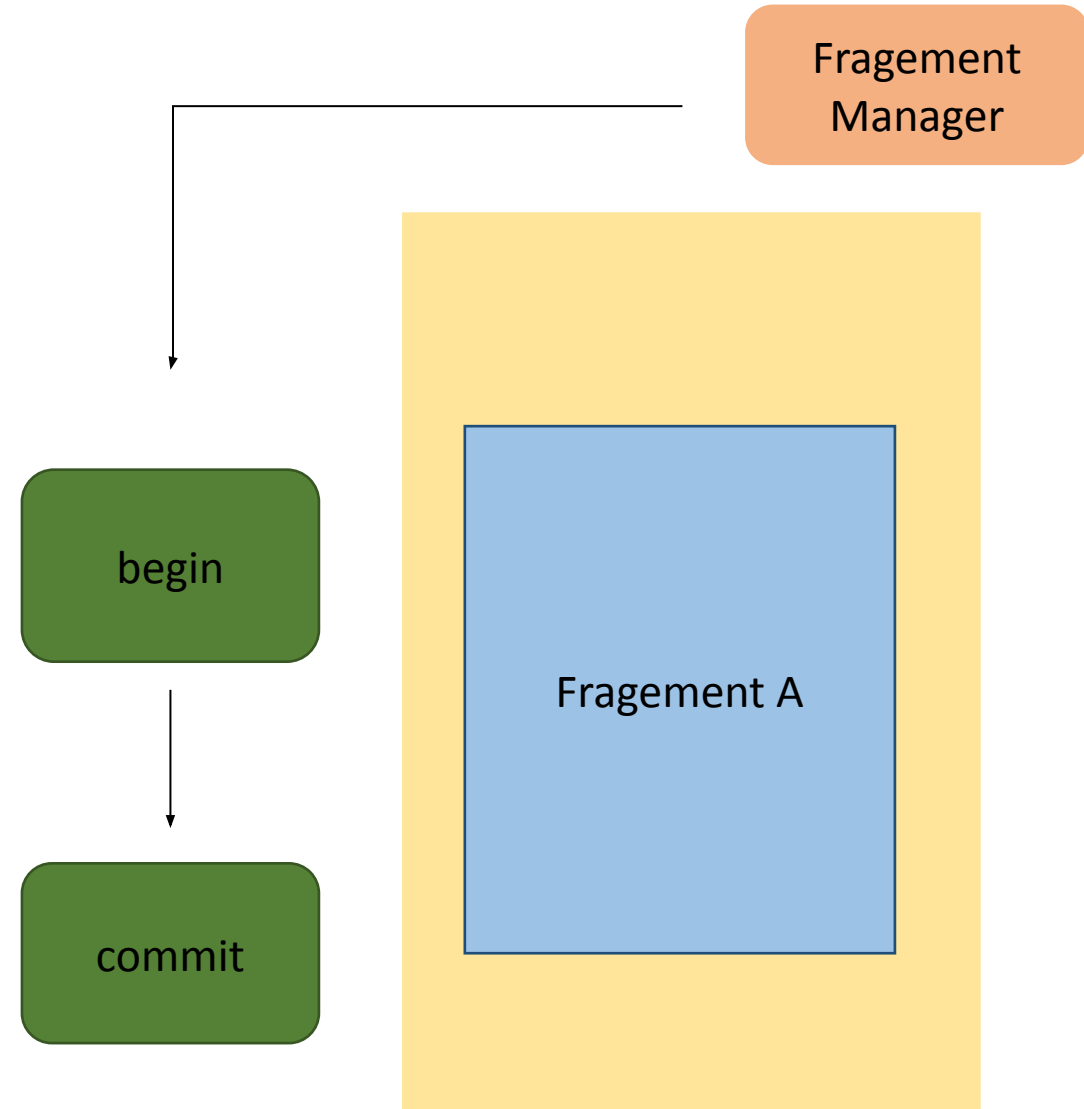
Fragment Transactions

Changes to the UI in terms of adding, Removing and replacing Fragments are Conducted as **FragmentTransactions**

Begin a transaction

Add, remove, replace whatever fragments you want

Commit the transaction



Fragments and their UI

- Most fragments will have a UI
- Will have its own layout
- you must implement the [onCreateView\(\)](#) callback method, which the Android system calls when it's time for the fragment to draw its layout. Your implementation of this method must return a [View](#) that is the root of your fragment's layout.

Fragments and their UI – onCreateView() using XML

- Can implement onCreateView using XML

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

The diagram illustrates the parameters of the `onCreateView` method. An orange box labeled "Activity parent's ViewGroup" has a blue arrow pointing down to the `ViewGroup container` parameter in the method signature. A blue box labeled "Bundle that provides data about the previous instance of the fragment, if the fragment is being resumed" has a blue arrow pointing up to the `Bundle savedInstanceState` parameter. Another orange box at the bottom, labeled "Have *example_fragment.xml* file that contains the layout This will be contained in resource layout folder.", has a blue arrow pointing up to the `R.layout.example_fragment` argument in the `inflate` method call.

OPTION1 –adding to an Activity via Activity layout XML.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

2 fragment classes



Need unique ids for each so system can restore the fragment if the activity is restarted

OPTION2 –creating and adding to an Activity via CODE.

*/*Inside Activity Code where you want to add Fragment (dynamically anywhere or in onCreate() callback)*

**/*

//get FragmentTransaction associated with this Activity

```
FragmentManager fragmentManager = getFragmentManager\(\);  
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction\(\);
```

//Create instance of your Fragment

```
ExampleFragment fragment = new ExampleFragment();
```

//Add Fragment instance to your Activity

```
fragmentTransaction.add(R.id.fragment_container, fragment);  
fragmentTransaction.commit();
```

This points to the Activity ViewGroup in which the fragment should be placed, specified by resource ID



Managing Fragments

FragmentManager methods:

- Get fragments that exist in Activity =
 - [findFragmentById\(\)](#) (for fragments that provide a UI in the activity layout)
 - [findFragmentByTag\(\)](#) (for fragments that do or don't provide a UI).
- Pop fragments off the back stack,
 - [popBackStack\(\)](#) (simulating a *Back* command by the user).
- Register a listener for changes to the back stack,
 - [addOnBackStackChangeListener\(\)](#).

Fragment Transactions – adding, removing and replacing dynamically

// Create new fragment and transaction

```
Fragment newFragment = new ExampleFragment();
```

```
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
```

// Replace whatever is in the fragment_container view with this fragment


// and add the transaction to the back stack

```
transaction.replace(R.id.fragment_container, newFragment);
```

```
transaction.addToBackStack(null);
```

// Commit the transaction

```
transaction.commit();
```



newFragment replaces whatever fragment (if any) is currently in the layout container identified by the **R.id.fragment_container**

If you do not call [addToBackStack\(\)](#) when you perform a transaction that removes a fragment, then that fragment is destroyed when the transaction is committed and the user cannot navigate back to it. Whereas, if you do call [addToBackStack\(\)](#) when removing a fragment, then the fragment is *stopped* and will be resumed if the user navigates back.

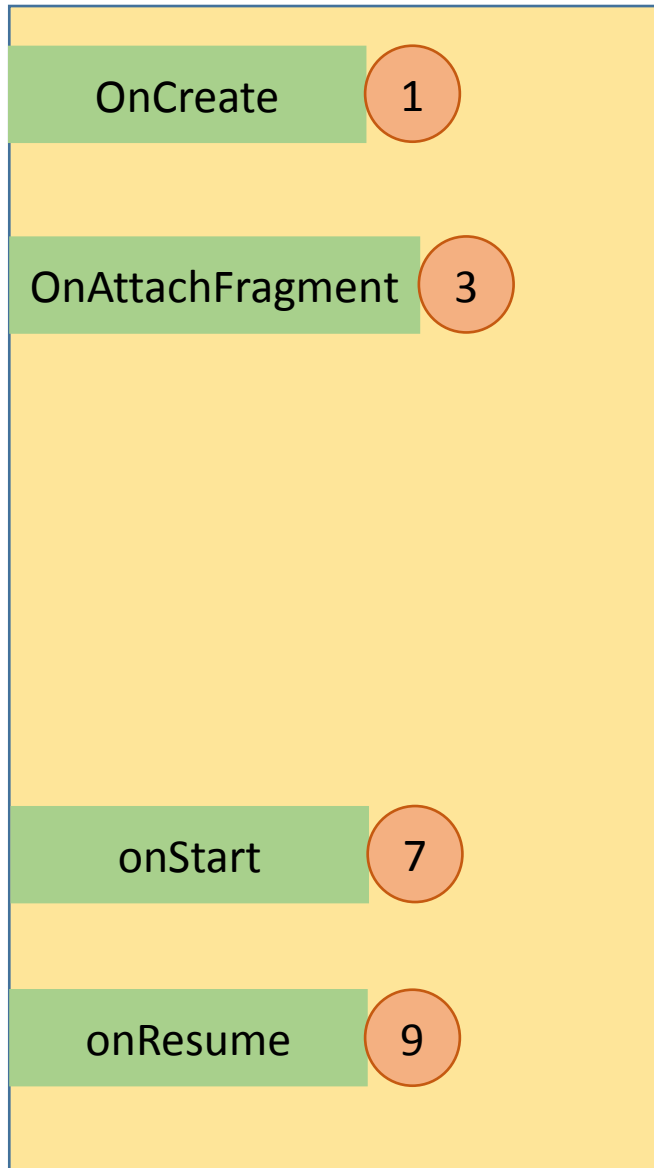
OPTION 3- Adding Fragment that has NO UI using Code

- use a fragment to provide a background behavior for the activity without presenting additional UI.
- use [`add\(Fragment, String\)`](#) (supplying a unique string "tag" for the fragment, rather than a view ID).
 - it's not associated with a view in the activity layout, it does not receive a call to [`onCreateView\(\)`](#). So you don't need to implement that method.
- If you want to get the fragment from the activity later, you need to use [`findFragmentByTag\(\)`](#).

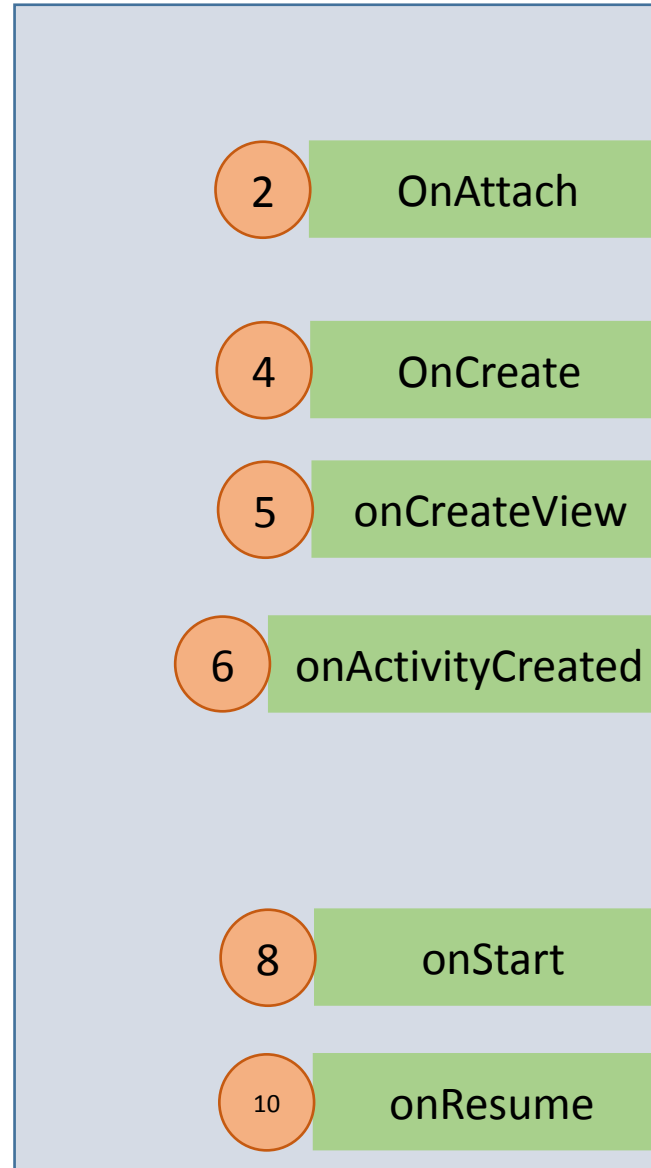
Create your own Fragment class or use known sub-classes

- [DialogFragment](#) Displays a floating dialog. Using this class to create a dialog is a good alternative to using the dialog helper methods in the [Activity](#) class, because you can incorporate a fragment dialog into the back stack of fragments managed by the activity, allowing the user to return to a dismissed fragment.
- [ListFragment](#) Displays a list of items that are managed by an adapter (such as a [SimpleCursorAdapter](#)), similar to [ListActivity](#). It provides several methods for managing a list view, such as the [onListItemClick\(\)](#) callback to handle click events.
- [PreferenceFragment](#) Displays a hierarchy of [Preference](#) objects as a list, similar to [PreferenceActivity](#). This is useful when creating a "settings" activity for your application.

Activity



Fragment



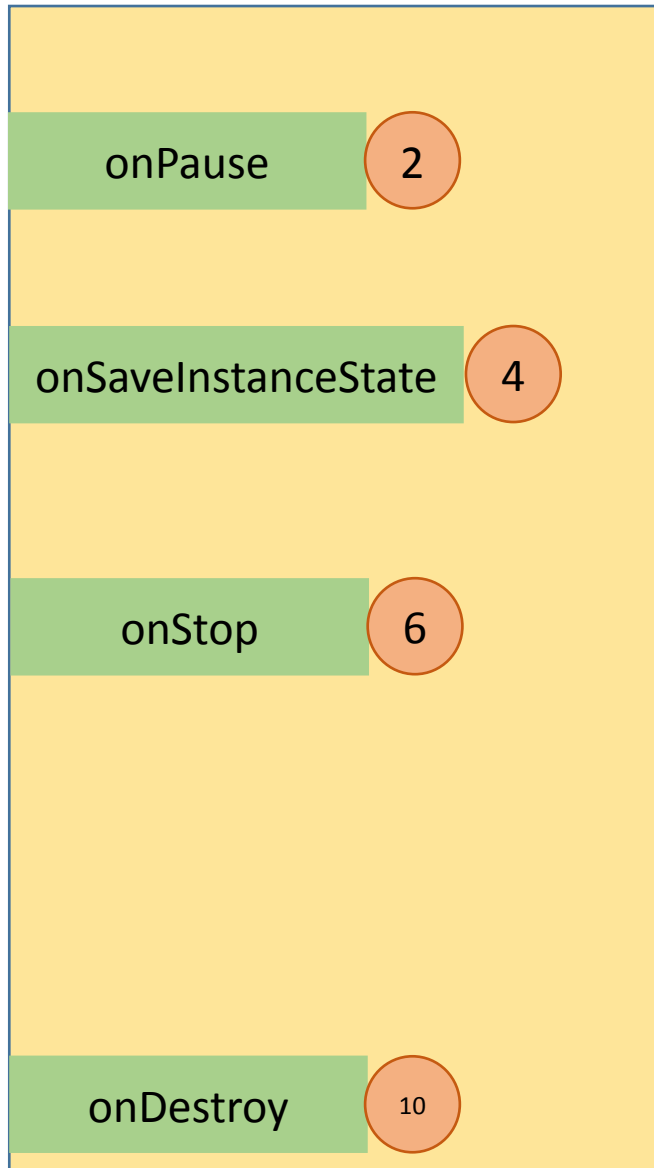
onAttach is called after Fragment is associated with its Activity. Gets a reference to the Activity object which can be used as Context.

onCreate. The system calls this 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. You are expected to return a View Hierarchy for your fragment.

onActivityCreated Called after Activity onCreate has completed execution. Use this method to access/modify UI elements.

Activity



Fragment



onSaveInstanceState Use this to save information inside a Bundle object.

onDestroyView Called after the Fragment View Hierarchy is no longer accessible

onDestroy Called after fragment is not used. It still exists as a java object attached to the Activity

onDetach Fragment is not tied to the Activity and does not have a View hierarchy

Android Studio interface showing the development of an Android application titled "Testing Fragments".

Project Structure (Left Panel):

- app
 - manifests
 - java
 - com.uol.tanzila.mynewtestapp
 - frag1
 - frag2
 - MainActivity
 - res
 - drawable
 - layout
 - activity_main.xml
 - frag1.xml
 - frag2.xml
 - frag3.xml
 - mipmap
 - values
 - Gradle Scripts

Palette (Top Left):

- Layouts
- Widgets
 - Plain TextView
 - Large Text
 - Medium Text
 - Small Text
 - Button
 - Small Button
 - RadioButton
 - CheckBox
 - Switch
 - ToggleButton
 - ImageButton
 - ImageView
 - ProgressBar (Large)
 - ProgressBar (Normal)
 - ProgressBar (Small)
 - ProgressBar (Horizontal)
 - SeekBar
 - RatingBar
 - Spinner
 - WebView
- Text Fields
- Containers

Device Screen (Center):

MyNewTestApp

Testing Fragments

SHOW FRAGMENT 1

SHOW FRAGMENT 2

Fragment placeholder area.

Component Tree (Top Right):

- Device Screen
 - LinearLayout (vertical)
 - textView - @string/testing_fragments
 - btn1 (Button) - "Show Fragment 1"
 - btn2 (Button) - "Show Fragment 2"
 - fragment - com.uol.tanzila.mynewt...frag1

Properties (Bottom Right):

activity_main.xml | frag1.java | frag2.xml | frag2.java | MainActivity.java | frag1.xml

LinearLayout | fragment

```
30
31
32 android:layout_width="wrap_content"
33 android:layout_height="wrap_content"
34 android:text="New Button"
35 android:id="@+id/btn2"
36 android:layout_gravity="center_horizontal" />
37
38 <fragment
39     android:layout_width="match_parent"
40     android:layout_height="match_parent"
41     android:name="com.uol.tanzila.mynewtestapp.frag1"
42     android:id="@+id/fragment"
43     android:layout_below="@+id/textView"
44     android:layout_alignParentLeft="true"
45     android:layout_alignParentStart="true" />
46
47 </LinearLayout>
```

Context Menu (Bottom Left):

- New
 - Java Class
 - Android resource file
 - Android resource directory
 - File
 - Package
 - C++ Class
 - C/C++ Source File
 - C/C++ Header File
 - Image Asset
 - Vector Asset
 - Singleton
 - Edit File Templates...
 - AIDL
 - Activity
 - Android Auto
 - Folder
 - Fragment
 - Fragment (Blank)
 - Fragment (List)
 - Fragment (with a +1 button)
 - Google
 - Other
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Copy Path (Ctrl+Shift+C)
- Copy as Plain Text
- Copy Reference (Ctrl+Alt+Shift+C)
- Paste (Ctrl+V)
- Find Usages (Alt+F7)
- Find in Path... (Ctrl+Shift+F)
- Replace in Path... (Ctrl+Shift+R)
- Analyze
- Refactor
- Add to Favorites
- Show Image Thumbnails (Ctrl+Shift+T)
- Reformat Code (Ctrl+Alt+L)
- Optimize Imports (Ctrl+Alt+O)
- Delete... (Delete)
- Run 'Tests in 'com.uol.tanzila.myne...' (Ctrl+Shift+F10)
- Debug 'Tests in 'com.uol.tanzila.myne...' (Ctrl+Shift+F10)


```

activity_main.xml x frag1.java x frag2.xml x frag2.java x MainActivity.java x
1 package com.uol.tanzila.mynewtestapp;
2 import ...
7 public class frag2 extends Fragment {
8     @Override
9     public View onCreateView(LayoutInflater inflater, ViewGroup container,
10                             Bundle savedInstanceState) {
11         // Inflate the layout for this fragment
12         return inflater.inflate(R.layout.frag2, container, false);
13     }
14 }

```

```

activity_main.xml x frag1.java x frag2.xml x frag2.java x MainActivity.java x frag1.xml x
1 package com.uol.tanzila.mynewtestapp;
2
3 import ...
10
11
12 public class frag1 extends Fragment {
13
14     @Override
15     public View onCreateView(LayoutInflater inflater, ViewGroup container,
16                             Bundle savedInstanceState) {
17         // Inflate the layout for this fragment
18         return inflater.inflate(R.layout.frag1, container, false);
19     }
20 }

```

```

activity_main.xml x frag1.java x frag2.xml x frag2.java x MainActivity.java x
1 package com.uol.tanzila.mynewtestapp;
2 import android.support.v4.app.Fragment;
3 import android.support.v4.app.FragmentTransaction;
4 import android.support.v7.app.AppCompatActivity;
5 import android.os.Bundle;
6 import android.view.View;
7 import android.widget.Button;
8
9 public class MainActivity extends AppCompatActivity {
10     Fragment frag;
11     Button btn1, btn2;
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16         btn1=(Button) findViewById(R.id.btn1);
17         btn2=(Button) findViewById(R.id.btn2);
18         onBtnClick();
19     }
20     void onBtnClick() {
21         btn1.setOnClickListener((v) -> {
22
23             frag=new frag1();
24             FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
25             ft.replace(R.id.fragment, frag);
26             ft.addToBackStack(null);
27             ft.commit();
28
29         });
30         btn2.setOnClickListener(new View.OnClickListener() {
31             @Override
32             public void onClick(View v) {
33                 frag=new frag2();
34                 FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
35                 ft.replace(R.id.fragment, frag);
36                 ft.addToBackStack(null);
37                 ft.commit();
38
39             }
40         });
41     }
42 }

```