Fragment

What is the Fragment Back Stack?

When you replace, add, or remove fragments, you can choose to add those fragment transactions to the back stack. The back stack in fragments in Android refers to a mechanism that allows the app to manage a series of fragment transactions in a way that users can navigate backward through those fragments. Essentially, it provides the ability to retain fragments in memory and allow backward navigation when a user presses the Back button.

LayoutInflater: LayoutInflater is a system service in Android that is responsible for converting XML layout files (like activity_main.xml) into View objects that can be displayed on the screen.
In other Words: LayoutInflater = XML → Java/Kotlin UI objects

Fragment: Fragments in Android represent a modular and reusable portion of an app's user interface within an Activity. A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments can't live on their own. They must be *hosted* by an activity or another fragment. The fragment's view hierarchy becomes part of, or *attaches to*, the host's view hierarchy.

Fragment is like **sub-activity** — it has:

- Its own lifecycle,
- Its own layout (XML),
- And its own code (a Fragment subclass).

What is the FragmentManager?

The **FragmentManager** is the Android system class responsible for **adding**, **removing**, **replacing**, and **managing** fragments inside an activity.

It is provided by the **AppCompatActivity** class and is accessed via:

```
getSupportFragmentManager()
```

(If you're using the older API, you'd use getFragmentManager(), but getSupportFragmentManager() is the modern AndroidX version.)

Key Responsibilities of FragmentManager

Task	Description
Add / Replace / Remove Fragments	Manages fragment transactions (i.e., showing/hiding fragments in containers).
Back Stack Management	Keeps a stack of fragment states so you can navigate back (like activities).
State Restoration	Automatically restores fragment states after configuration changes (like rotation).
Lifecycle Management	Keeps fragments in sync with their parent activity lifecycle.

Static Fragment vs. Dynamic Fragment

Static Fragment:

A Static Fragment is defined directly in the Activity's XML layout and is part of the initial UI when the Activity is created. It is useful when the Fragment layout is static and doesn't need to change at runtime.

Key Characteristics:

- Declared in XML layout file.
- Created at launch time when the Activity starts.
- Does not change dynamically unless the layout is modified or the Fragment is removed/replaced via code.

Example of Static Fragment (in Java)

1. Create the Fragment Class (ExampleFragment.java):

```
public class ExampleFragment extends Fragment {


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

2. **Declare the Fragment in the Activity's XML Layout** (activity_main.xml):

```
<androidx.fragment.app.FragmentContainerView
  android:id="@+id/fragment_container"
  android:name="com.example.app.ExampleFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />
```

Dynamic Fragment:

A Dynamic Fragment is added, removed, or replaced at runtime using the FragmentManager. This allows you to modify the UI based on user interactions, like navigating to different screens within the same Activity.

Key Characteristics:

- Added at runtime via FragmentTransaction.
- Can be replaced or removed dynamically.
- Ideal for changing the UI based on user actions, like navigating between screens.

Example of Dynamic Fragment (in Java)

1. **Create the Fragment Class** (ExampleFragment xml / java):

```
<!-- fragment_example.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/text_example"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to Example Fragment"
        android:textSize="18sp"/>

    <Button
        android:id="@+id/button_click_me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:layout_marginTop="16dp"/>
</LinearLayout>


// -------→ JAVA
public class ExampleFragment extends Fragment {

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

2. **Activity Code to Add/Replace the Fragment Dynamically** (MainActivity.java):

You can use the **FragmentManager** to dynamically add or replace a Fragment in the Activity's layout.

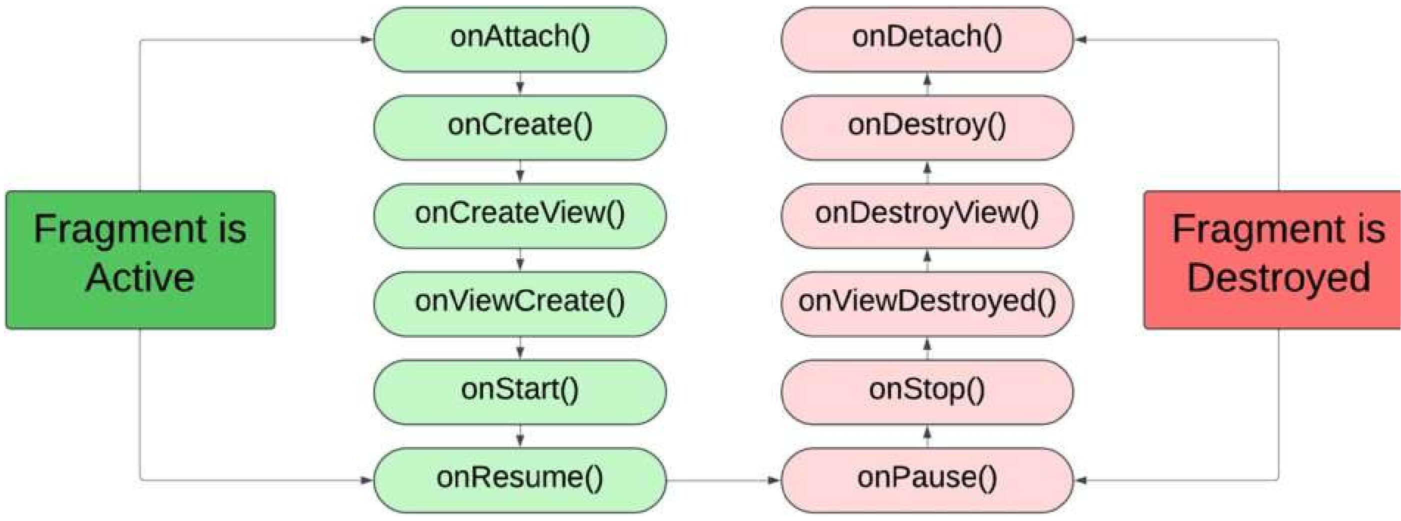
1. **Main Activity Layout** (activity_main.xml):

```
<<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="20dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">



    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="20dp"
        android:id="@+id/frame"
        />

</LinearLayout>

//   >JAVA

public class MainActivity extends AppCompatActivity {
  ---------------
    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);


        ExampleFragment fragment = new ExampleFragment();

        // Begin a FragmentTransaction to add the fragment to the container

        FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();

        transaction.replace(R.id.fragment_container, fragment); //
R.id.fragment_container is the container for the fragment

        transaction.addToBackStack(null);  // Add to back stack to handle back
navigation

        transaction.commit(); // Commit the transaction

    }

}
```

- The **FragmentTransaction** replaces the existing fragment in the container (R.id.fragment_container) with a new instance of the ExampleFragment.

- The **addToBackStack()** method is used to enable **back navigation**. If the user presses the back button, the previous fragment will be displayed.

Fragment LifeCycle: Android fragments have their own lifecycle very similar to an android activity these are components of android lifecycle.



Each Lifecycle Method Explained:-

Method	When It's Called	What You Should Do Here
onAttach(Context context)	When the fragment is attached to its host activity.	Initialize things that need the Activity context. Example: get references, communicate with Activity.
onCreate(Bundle savedInstanceState)	When the fragment is first created (but UI not ready yet).	Initialize variables, adapters, or data you want to keep even if the view is recreated. Don't access UI elements here.
onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)	Called to create the UI layout of the fragment.	Inflate your XML layout and return the root view. Example: return inflater.inflate(R.layout.fragment_home, container, false);
onViewCreated(View view, Bundle savedInstanceState)	Immediately after the view is created.	Initialize views (findViewById), set click listeners, bind data to UI.
onStart()	When the fragment becomes visible to the user.	Start animations, refresh UI data.

Method	When It's Called	What You Should Do Here
onResume()	When the fragment is fully visible and interactive.	Start listening for user actions, resume updates (like a camera preview or live data).
onPause()	When the user leaves the fragment (but it's still partially visible or transitioning).	Pause animations, save temporary data.
onStop()	When the fragment is no longer visible.	Stop background tasks, unregister receivers, or listeners.
onDestroyView()	When the fragment's UI view is destroyed, but the fragment itself still exists in memory.	Clean up references to binding objects or views to prevent memory leaks.
onDestroy()	When the fragment itself is being destroyed.	Release final resources like adapters or connections.
onDetach()	When the fragment is completely disconnected from its activity.	Clean up any remaining references to the activity to avoid memory leaks.

How It Works (Step by Step)

1. **Define a container** in your layout:

```
<LinearLayout
    android:id="@+id/frag_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

2. **Use the FragmentManager** to start a transaction:

```
FragmentManager fm = getSupportFragmentManager();
FragmentTransaction ft = fm.beginTransaction();
```

3. **Perform actions** on the transaction:

```
ft.add(R.id.frag_container, new Fragment1());
// or
ft.replace(R.id.frag_container, new Fragment2());
// or
ft.remove(fragment);
```

4. **Commit the transaction**:

```
ft.commit();
```

Send Data In Fragment

Sending data to an Android Fragment can be achieved using several methods, depending on the source of the data and the relationship between the sending and receiving components.

1. From an Activity to a Fragment (using Bundle):

This is a common method for passing initial data to a fragment when it's created or added to an activity. In the Activity.

```
Bundle bundle = new Bundle();
bundle.putString("key_name", "your_data_string"); // Or other data types like int, boolean, etc.
MyFragment fragment = new MyFragment();
fragment.setArguments(bundle);
getSupportFragmentManager().beginTransaction().add(R.id.fragment_container, fragment).commit();
```

In the Fragment (to retrieve data).

```
Java
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    if (getArguments() != null) {
        String receivedData = getArguments().getString("key_name");
        // Use the receivedData
    }
    return inflater.inflate(R.layout.my_fragment_layout, container, false);
}
```