**✅ Section 22: Fragments in Android – Complete & Combined Notes**

**🔷 I. What Are Fragments?**

A **Fragment** is a modular, reusable portion of an Android UI that:

* Has its **own layout, logic, and lifecycle**
* Must be hosted within an **Activity or another Fragment**
* Enables flexible and adaptive UI on different screen sizes (e.g., tablets vs phones)

**🧱 Real-World Analogy**

Think of fragments as **Lego blocks** — reusable UI pieces that can be stacked or replaced to build complex layouts.

**✅ Benefits of Using Fragments**

| **Benefit** | **Description** |
| --- | --- |
| **Modular UI** | Reusable and maintainable components |
| **Multi-pane Layouts** | Ideal for tablets (e.g., master-detail UI) |
| **Single Activity Architecture** | Navigate between fragments in one activity |
| **Lifecycle Control** | Manage logic and cleanup independently from the host activity |
| **Device Adaptability** | UI can dynamically adapt to screen sizes and orientations |

**🔷 II. Fragment Lifecycle**

Understanding the lifecycle is essential for managing UI logic, memory, and performance.

**🔁 Full Lifecycle Callbacks**

| **Method** | **When Called** | **Use Case** |
| --- | --- | --- |
| onAttach() | Fragment is associated with activity | Access context, initialize communication |
| onCreate() | Initial creation | One-time setup, retrieve arguments |
| onCreateView() | Inflate UI | Bind layout using XML or programmatically |
| onViewCreated() | After onCreateView() | Safe to use findViewById() |
| onActivityCreated() | Host activity is created | Use activity’s views if needed |
| onStart() | Fragment becomes visible | Prepare for user interaction |
| onResume() | Fragment is visible & interactive | Start tasks, update UI |
| onPause() | Fragment going invisible | Pause animations, save state |
| onStop() | No longer visible | Stop operations, release resources |
| onDestroyView() | Destroy UI-related objects | Nullify view bindings, adapters |
| onDestroy() | Final cleanup | De-initialize non-UI resources |
| onDetach() | Detached from activity | Null references to context |

**🔷 III. Tools, APIs & Project Setup**

**🛠️ Key APIs and Classes**

| **Class/API** | **Purpose** |
| --- | --- |
| Fragment | Base class for UI component |
| FragmentManager | Manages fragments and back stack |
| FragmentTransaction | Used to add, replace, remove fragments |
| FrameLayout | Container to hold fragments (legacy) |
| FragmentContainerView | Jetpack alternative to FrameLayout |
| getSupportFragmentManager() | Get FragmentManager using support libraries |
| commit() / commitNow() | Finalize the transaction |

**🔷 IV. Creating and Hosting Fragments – Step-by-Step**

**✅ Step 1: Create a Fragment (Java)**

public class FirstFragment extends Fragment {

@Nullable

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

// Inflate layout

View view = inflater.inflate(R.layout.fragment\_first, container, false);

// Access and setup UI

Button btn = view.findViewById(R.id.button);

btn.setOnClickListener(v ->

Toast.makeText(getActivity(), "Welcome to First Fragment!", Toast.LENGTH\_SHORT).show());

return view;

}

@Override

public void onAttach(Context context) {

super.onAttach(context);

Toast.makeText(context, "onAttach Called", Toast.LENGTH\_SHORT).show();

}

@Override

public void onResume() {

super.onResume();

Toast.makeText(getContext(), "Fragment Ready for Interaction", Toast.LENGTH\_SHORT).show();

} }

**✅ Step 2: Design Layout – fragment\_first.xml**

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="#FF9800">

<TextView

android:id="@+id/textView"

android:text="First Fragment"

android:textSize="48sp"

android:layout\_gravity="center"/>

<Button

android:id="@+id/button"

android:text="Say Hello"

android:layout\_gravity="center"

android:layout\_marginTop="100dp"/>

</FrameLayout>

**✅ Step 3: Host Fragment in Main Activity Layout – activity\_main.xml**

<LinearLayout ... orientation="vertical">

<TextView

android:id="@+id/titleText"

android:text="Fragments App"

android:textSize="32sp" />

<Button

android:id="@+id/btnFragment1"

android:text="Show Fragment 1" />

<Button

android:id="@+id/btnFragment2"

android:text="Show Fragment 2" />

<!-- Fragment container -->

<FrameLayout

android:id="@+id/frameLayout"

android:layout\_width="match\_parent"

android:layout\_height="200dp"

android:layout\_marginTop="40dp"/>

</LinearLayout>

**✅ Step 4: Handle Fragment Transactions – MainActivity.java**

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

findViewById(R.id.btnFragment1).setOnClickListener(v -> loadFragment(new FirstFragment()));

findViewById(R.id.btnFragment2).setOnClickListener(v -> loadFragment(new SecondFragment()));

}

private void loadFragment(Fragment fragment) {

FragmentManager fm = getSupportFragmentManager();

FragmentTransaction ft = fm.beginTransaction();

ft.replace(R.id.frameLayout, fragment);

ft.commit(); // Use ft.addToBackStack(null) if you want back navigation

}

}

**🔷 V. Best Practices & Professional Tips**

| **Area** | **Best Practice** |
| --- | --- |
| **Fragment Container** | Use FragmentContainerView (modern) over FrameLayout |
| **UI Initialization** | Use onViewCreated() instead of onCreateView() for findViewById() |
| **Back Navigation** | Use addToBackStack(null) to allow navigation reversal |
| **Reordering** | Use setReorderingAllowed(true) for better performance |
| **Memory Management** | Release resources in onDestroyView() |
| **Communication** | Use interfaces or FragmentResultListener/Shared ViewModel |
| **Navigation** | Prefer Jetpack **Navigation Component** with XML-based nav graph |

**🔷 VI. Real-World Fragment Use Cases**

| **Scenario** | **Why Fragments?** |
| --- | --- |
| **Bottom Navigation** | Load different screens using fragments |
| **Tablet Master-Detail** | Side-by-side fragments for list & detail |
| **Multi-step Forms** | Each step as a fragment |
| **Login/Register Switch** | Toggle fragments for login/register screen |
| **Dynamic Content** | Load fragments based on user roles |

**🔷 VII. Advanced Topics – Part B (Not Covered in Demo But Important)**

| **Topic** | **Importance** |
| --- | --- |
| FragmentContainerView | Preferred container (Jetpack) |
| Navigation Component | Simplifies fragment navigation |
| Safe Args | Type-safe argument passing |
| Fragment Arguments | Pass data via setArguments(Bundle) |
| Shared ViewModel | Share state between fragments |
| Fragment Communication | Use interface or FragmentResultListener |
| DialogFragment | UI popups (e.g., DatePicker) |
| Transitions & Animations | ft.setCustomAnimations() for smooth UI |
| FragmentScenario | Test fragments in isolation |
| ViewBinding / DataBinding | Type-safe and cleaner UI logic |
| FragmentFactory | Used for injecting dependencies in fragments |

**✅ Final Checklist for Projects Using Fragments**

✔ Use modular fragments for maintainable UI  
✔ Use onCreateView() + onViewCreated() for UI setup  
✔ Replace FrameLayout with FragmentContainerView when using Navigation Component  
✔ Add transactions to back stack when needed  
✔ Use lifecycle callbacks for cleanups and state management  
✔ Avoid memory leaks: null adapters, views, references  
✔ Communicate cleanly: via shared ViewModel or interface  
✔ Use ViewModel, LiveData, or StateFlow for reactive data  
✔ Test with FragmentScenario or UI tests (Espresso)

✅ **Instructor’s Emphasis**  
“Fragments allow scalable, dynamic user interfaces. Master lifecycle management and use modern navigation tools to build robust Android applications.”