

UNIT 5

UI Fragments, Menus and Dialogs

(6 Hours) — Mobile Programming (CAC351)

5.1 The Need for UI Flexibility

Early Android apps used one activity per screen. That works on phones but is rigid on tablets and foldables, where a list and its details can be shown side by side. Hard-wiring the UI inside an activity makes it hard to reuse or rearrange. **Fragments** solve this by splitting the UI into modular, reusable pieces that can be combined in different ways depending on screen size.

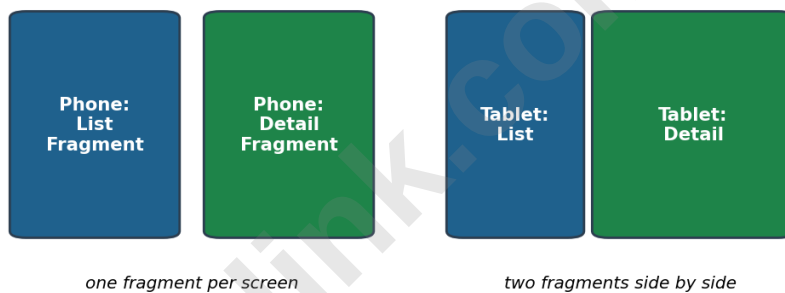


Figure 5.1: The same two fragments arranged differently on phone and tablet

5.2 Introduction to Fragments

A **Fragment** is a reusable portion of an app's UI with its own layout, its own life cycle and its own event handling. A fragment always lives inside a host activity (or another fragment), and one activity can host many fragments. Fragments can be added, replaced or removed at run time, and can be placed on a back stack.

Life Cycle of a Fragment

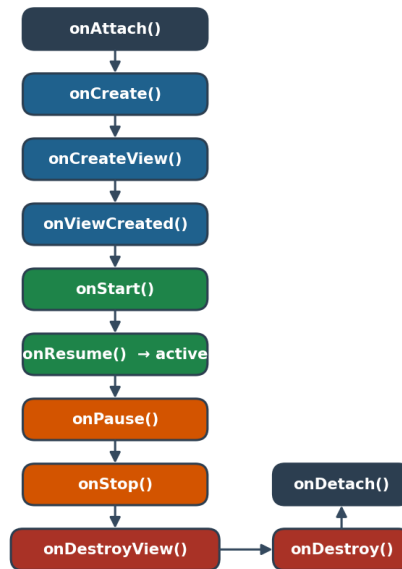


Figure 5.2: Main fragment life-cycle callbacks (top to bottom)

Callback	Purpose
onAttach()	Fragment attached to its host activity (context available)
onCreate()	Initial creation; read arguments, non-UI setup
onCreateView()	Inflate and return the fragment's layout
onViewCreated()	View ready: find views, set listeners
onStart() / onResume()	Fragment visible / interactive
onPause() / onStop()	Losing focus / no longer visible
onDestroyView()	View is destroyed (release view references)
onDestroy() / onDetach()	Fragment destroyed / detached from activity

Note: Unlike an activity, a fragment has separate callbacks for its view (onCreateView/onDestroyView) because the view can be destroyed while the fragment object survives on the back stack.

5.3 Creating a UI Fragment

Creating a fragment involves three steps: (1) define its layout XML, (2) write the Fragment class, (3) add the fragment to an activity. Use **File → New → Fragment → Fragment (Blank)**, or create manually.

Step 1 – Fragment layout (res/layout/fragment_greeting.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">
    <EditText android:id="@+id/etName" android:hint="Your name"
        android:layout_width="match_parent" android:layout_height="wrap_content" />
    <Button android:id="@+id/btnHello" android:text="Say Hello"
        android:layout_width="wrap_content" android:layout_height="wrap_content" />
    <TextView android:id="@+id/tvOut"
        android:layout_width="wrap_content" android:layout_height="wrap_content" />
</LinearLayout>
```

Listing 5.1: fragment_greeting.xml

Step 2 – Fragment class and wiring widgets

```
public class GreetingFragment extends Fragment {
    private EditText etName; private TextView tvOut;

    public GreetingFragment() { super(R.layout.fragment_greeting); } // inflates layout

    @Override
    public void onCreateView(View view, Bundle savedInstanceState) {
        super.onCreateView(view, savedInstanceState);
        etName = view.findViewById(R.id.etName);
        tvOut = view.findViewById(R.id.tvOut);
        view.findViewById(R.id.btnHello).setOnClickListener(v ->
            tvOut.setText("Hello, " + etName.getText()));
    }
}
```

Listing 5.2: GreetingFragment.java – widgets are wired in onCreateView()

Notice that inside a fragment we call `view.findViewById()` on the fragment's root view, not on the activity. Use `requireContext()` or `requireActivity()` where a Context is needed.

Step 3 – Hosting the fragment in an activity

```
<!-- activity_main.xml -->
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/fragment_container"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:name="com.example.app.GreetingFragment" />
```

Listing 5.3a: Static (XML) hosting

```
// Dynamic hosting in MainActivity.onCreate()
if (savedInstanceState == null) {
    getSupportFragmentManager().beginTransaction()
        .replace(R.id.fragment_container, new GreetingFragment())
        .addToBackStack(null) // lets Back return to previous fragment
        .commit();
}
```

Listing 5.3b: Dynamic hosting with FragmentManager

5.4 Introduction to Fragment Manager

The **FragmentManager** adds, removes, replaces and finds fragments in an activity. Changes are grouped into a **FragmentTransaction** that is applied on *commit()*. Get it with *getSupportFragmentManager()* in an activity or *getChildFragmentManager()* inside a fragment.

Method	Effect
<code>add(containerId, fragment)</code>	Adds a fragment to the container
<code>replace(containerId, fragment)</code>	Removes the current fragment and adds a new one
<code>remove(fragment)</code>	Removes a fragment
<code>addToBackStack(name)</code>	Saves the transaction so Back reverses it
<code>findFragmentById(id)</code>	Looks up an existing fragment
<code>commit()</code>	Schedules the transaction to run

Communication: fragments should not talk to each other directly. Use a shared *ViewModel*, the Fragment Result API (*setFragmentResult()*), or an interface implemented by the host activity.

Difference between Activity and Fragment

Activity	Fragment
Independent, full screen component	Modular part of a screen; cannot exist without a host
Declared in <code>AndroidManifest.xml</code>	Not declared in manifest
Started with Intents	Managed by <code>FragmentManager</code> transactions
Has its own window	Uses the host activity's window
Callbacks: <code>onCreate</code> ... <code>onDestroy</code>	Extra callbacks: <code>onAttach</code> , <code>onCreateView</code> , <code>onDestroyView</code> , <code>onDetach</code>
Not reusable across screens	Reusable in many activities
Multiple activities cost more memory	Lightweight, several can show at once

5.5 Menus

A **menu** presents a set of actions to the user. Menus are defined in XML under *res/menu/*.

Menu type	Description
Options menu	Primary menu in the app bar (toolbar); includes items and overflow (⋮) menu

Menu type	Description
Context menu	Floating menu shown on long-press of a view; contextual action mode for multi-select
Popup menu	Anchored list of items shown for a specific view (PopupMenu)
Bottom navigation / Navigation drawer	Top-level navigation destinations

```

<!-- res/menu/main_menu.xml -->
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_settings" android:title="Settings"
          app:showAsAction="never" />
    <item android:id="@+id/action_search" android:title="Search"
          android:icon="@drawable/ic_search" app:showAsAction="ifRoom" />
</menu>

```

Listing 5.4: Menu XML

```

@Override public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main_menu, menu); // inflate menu
    return true;
}
@Override public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.action_settings) {
        startActivity(new Intent(this, SettingsActivity.class));
        return true;
    }
    return super.onOptionsItemSelected(item);
}

// Context menu: registerForContextMenu(view), onCreateContextMenu(),
// onContextItemSelected()
// Popup menu:
PopupMenu pm = new PopupMenu(this, anchorView);
pm.getMenuInflater().inflate(R.menu.main_menu, pm.getMenu());
pm.setOnMenuItemClickListener(item -> { /* handle */ return true; });
pm.show();

```

Listing 5.5: Implementing an options menu and a popup menu

Note: In newer AndroidX code, the `MenuProvider` API is recommended in fragments instead of overriding `onCreateOptionsMenu()`.

5.6 Dialogs

A **dialog** is a small window that prompts the user to decide or enter information; it appears above the current activity and blocks interaction until dismissed. Common types: *AlertDialog* (title, message, up to three buttons, list), *DatePickerDialog*, *TimePickerDialog*, and custom dialogs with their own layout.

Creating a DialogFragment and setting its content

The recommended way is to wrap the dialog in a **DialogFragment** so that it survives rotation and is managed by the **FragmentManager**. Override *onCreateDialog()* and return an *AlertDialog* built with *AlertDialog.Builder*.

```
public class ConfirmDialogFragment extends DialogFragment {
    @NonNull @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(requireContext())
            .setTitle("Delete item")
            .setMessage("Do you really want to delete this item?")
            .setPositiveButton("Yes", (d, which) ->
                Toast.makeText(getContext(), "Deleted", Toast.LENGTH_SHORT).show())
            .setNegativeButton("No", null)
            .create();
    }
}

// show it from an activity:
new ConfirmDialogFragment().show(getSupportFragmentManager(), "confirm");
```

Listing 5.6: A confirmation DialogFragment

```
// Custom content: inflate your own layout
View v = requireActivity().getLayoutInflater().inflate(R.layout.dialog_login, null);
return new AlertDialog.Builder(requireContext())
    .setView(v).setTitle("Login")
    .setPositiveButton("OK", (d, w) -> { /* read fields from v */ })
    .create();
```

Listing 5.7: Setting a dialog's content with a custom layout

Example 1 — Master/detail

Design a note app that shows a list on phones and list + detail on tablets.

Solution

Create *NoteListFragment* and *NoteDetailFragment*. Provide *layout/activity_main.xml* with one *FragmentContainerView* (phone) and *layout-sw600dp/activity_main.xml* with two containers (tablet). On phones, replace the list with the detail fragment and add to the back stack; on tablets show both.

Key Takeaways

- ✓ Fragments are modular UI pieces with their own life cycle, hosted inside activities and managed by `FragmentManager`.
- ✓ Inflate the layout in `onCreateView` (or the constructor) and wire widgets in `onViewCreated`.
- ✓ Activities are declared in the manifest and started by intents; fragments are transacted (add/replace/remove/back stack).
- ✓ Menus: options, context and popup; defined in `res/menu` and inflated in code.
- ✓ Use `DialogFragment` with `AlertDialog.Builder` for dialogs; `setView()` supplies custom content.

— *End of Unit 5* —

bcalink.com