Mobile Programming

Course 3
Service and UI components

Qiong LIU

qiong.liu@cyu.fr
CY Tech, CY Cergy Paris University

2024 - 2025

What You'll Learn Today

Service

- ok Unbounded service
- ok Bounded service

Some essential and interesting UI components

- ok Floating Action Button
  - Menu
  - Fragments
  - Navigation
  - ...

Service.

Services

- A **Service** is an Android component that runs in the background to perform long-running operations.
- Services do not provide a user interface (unlike activities) and are primarily used for background tasks.
- Common use cases include:
  - Playing music in the background
  - Downloading files or fetching data from the network
  - Handling background tasks even if the user switches to another app
- Services run on the **main thread** by default, so heavy operations should be run in a separate thread (e.g., using ‘AsyncTask‘, ‘Thread‘, or ‘Handler‘).

Types of Services

There are two main types of Android services:

- Started Service (unbounded services):
  - A service that is started when a component (like an Activity) calls `onCreate()`.
  - Runs in the background indefinitely, even if the starting activity ends.
  - The service stops only when it calls `stopSelf()` or another component calls `stopService()`.
  - Example: Playing music in the background. The music keeps playing even if the user navigates to other activities.
- Bound Service:
  - Bound Service is tied to the lifecycle of the activity or component that binds to them using `bindService()`.
  - They stop automatically when the bound activity is destroyed or the connection is unbound.
  - Example: A service that provides real-time sensor data to an activity. The service stops when the activity is destroyed.

Example SoundService: Unbounded Service

Object: show a background music when you open your app.

Res → New → Android Resource File:

- File name: mysong (**NO** Capital)
- Resource type: raw
- Start directly at Oncreat()

Upload your music file to folder "raw".

```
import android.media.MediaPlayer;
public class MainActivity extends AppCompatActivity {
    // declear your member variable
    MediaPlayer music;
    @Override
        protected void onCreate(Bundle savedInstanceState) {

        // MediaPlayer
        music = MediaPlayer.create(MainActivity.this, R.raw.
            junebarcarolle);
        music.start();
        }
    }
```

Service

Declare the service in the AndroidManifest.xml:

```
<service
    android:name=".MyService"
    android:enabled="true"
    android:exported="true">
</service>
```

Bounded Service Example

An bounded example:

- In the `onStart()` method, the service is started when the activity becomes visible.
- The `onDestroy()` method ensures the service is stopped when the activity is destroyed, such as when the user closes the app or navigates away from this activity.

```
@Override
protected void onStart() {
    super.onStart();
    Intent serviceIntent = new Intent(this,
   BackgroundSoundService.class);
    this.startService(serviceIntent);
}
@Override
protected void onDestroy() {
    super.onDestroy();
    Intent serviceIntent = new Intent(this,
   .BackgroundSoundService.class);
    this.stopService(serviceIntent);
```

Bound Service: Life Cycle

We show how to implement a bound service that can pause and resume audio playback based on user interaction.

```
// onStartCommand to handle PAUSE and RESUME actions
@Override
public int onStartCommand(Intent intent, int flags, int
    startId) {
    // Get the action from the Intent
    String action = intent.getAction();

    if (action != null && action.equals("PAUSE")) {
        if (player.isPlaying()) {
            player.pause();  // Pause the audio
        }
    } else if (action != null && action.equals("RESUME")) {
        player.start();  // Resume audio playback
    } else {
        player.start();  // Default action: start playback
    }
    return START_STICKY;
}
```

Bound Service: Life Cycle

Explanation:

- The `onStartCommand()` method receives an `Intent` with an action (`PAUSE` or `RESUME`).
- If the action is `PAUSE`, the audio is paused using `player.pause()`.
- If the action is `RESUME`, the audio playback is resumed with `player.start()`.
- If no action is provided, the audio starts playing by default.

Bound Service: Life Cycle

When jumping to MainActivity2, you can send a pause command via Intent to pause the audio playback without stopping the service.

```
// Stop the BackgroundSoundService when MainActivity2 is
    created
Intent serviceIntent = new Intent(this,
    BackgroundSoundService.class);
serviceIntent.setAction("PAUSE");
startService(serviceIntent);
```

Exercise 1

A 20-minute quiz.

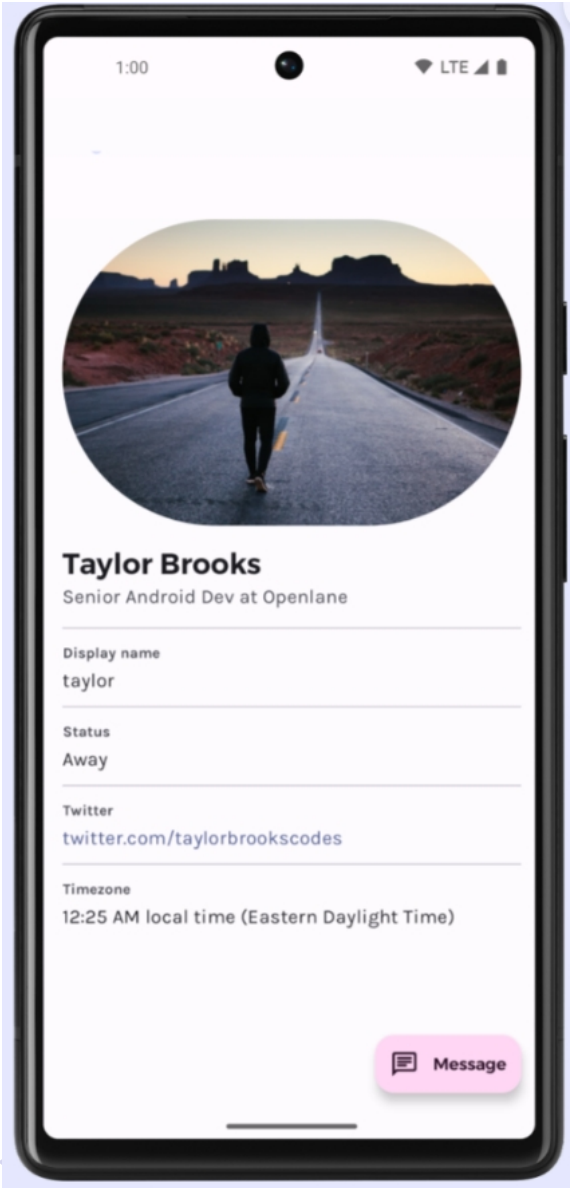
- **Background Music Service** :
  - Implement a `Service` to play background music in MainActivity.
  - Pause the music when navigating to SecondActivity, and resume when returning to MainActivity.

Android UI components: Floating Action Button

Floating Action Button

Floating Action Button:

- A floating action button is a special kind of button used for a primary action.
- It appears in front of all screen content, typically as a circular shape with an icon in its center.
- Only one floating action button is recommended per screen.



Floating Action Button (FAB) vs. Regular Button

- **Design and Visibility:**
  - FAB is "floats" above other elements.
  - Regular buttons are usually blend in with other content.
- **Behavior:**
  - FAB stays visible even when scrolling.
  - Regular buttons scroll along with the content.
- **Positioning:**
  - FAB is typically positioned at the bottom-right or center, each activity only one FAB at most usually.
  - Regular buttons are placed in-line with other UI elements.

Floating Action Button

Use case examples:

- Send Emails
- Refresh data on a screen
- In a food ordering app, allowing customers to view their current order summary at any time

```
<com.google.android.material.floatingactionbutton.
    FloatingActionButton
android:id="@+id/fab"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="end|bottom"
android:layout_margin="16dp"
android:src="@drawable/ic_add" />
```

Exercise 2

A 15-minute quiz:

- Create an app with two activities, MainActivity and SecondActivity.
- In the MainActivity, add a Floating Action Button (FAB) that navigates to the SecondActivity.
- In the SecondActivity, introduce a novel, and use FAB that always go back to up.

p.s., you can also change the button type for your TP1.

Android UI components:Option Menu

Android UI Components: Options Menu

What is an Options Menu?

- A set of menu items that appear in the action bar or overflow menu.
- Provides additional options or settings to the user.
- Commonly used for actions like Settings, Search, or Help.

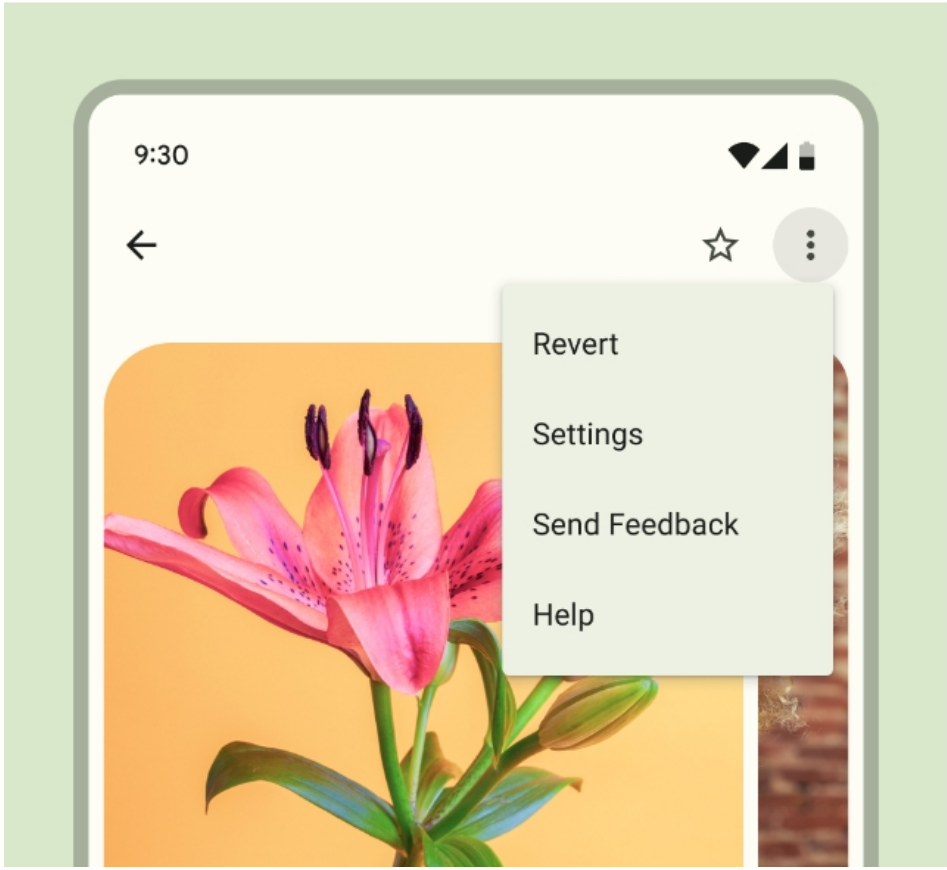


Figure: Options Menu Example

Step 1: Create the Menu XML

- In **res**: New **Android Resoruce Directory**, choose **menu**, then New **Menu Resource File**, e.g., `menu_main.xml`.
- Define each menu item with a title and optional icon.

Example:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/action_settings"
        android:title="Settings"
        android:icon="@drawable/ic_settings" />
    <item
        android:id="@+id/action_search"
        android:title="Search"
        android:icon="@drawable/ic_search" />
    <item
        android:id="@+id/action_help"
        android:title="Help"
      />
</menu>
```

Step 2: Inflate the Menu in Your Activity

- Override `onCreateOptionsMenu` in your activity to inflate(load) the menu.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

Step 3: Handle Menu Item Clicks

- Override `onOptionsItemSelected` to define actions for each menu item.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();

    if (id == R.id.action_settings) {
        Toast.makeText(this, "Settings selected", Toast.
            LENGTH_SHORT).show();
        return true;
    } else if (id == R.id.action_search) {
        Toast.makeText(this, "Search selected", Toast.
            LENGTH_SHORT).show();
        return true;
    } else if (id == R.id.action_help) {
        Toast.makeText(this, "Help selected", Toast.
            LENGTH_SHORT).show();
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

Concept of onOptionsItemSelected and Return Values

- onOptionsItemSelected(MenuItem item) handles clicks on menu items.
- return true indicates the event has been handled, stopping further processing.
- return false means the event has not been fully handled, allowing further handling by the system or other components.
- If ‘false‘ is returned, the system may look for other ‘onOptionsItemSelected‘ handlers (e.g., in parent activities) to handle the event.

Why Isn't the Options Menu Displayed?

Sometimes, the Options Menu may not appear if the Activity theme lacks an Action Bar.

Explanation:

- The Options Menu is typically shown in the Action Bar.
- If an Activity uses a NoActionBar theme, the Action Bar will not display, so the Options Menu will be invisible.

Solution: Set a Theme with Action Bar

- In `AndroidManifest.xml`, set the theme to `Theme.AppCompat.Light.DarkActionBar` or another theme that includes an Action Bar.

Example:

```
    <activity
    android:name=".MainActivity4"
    android:theme="@style/Theme.AppCompat.Light.
        DarkActionBar"
    android:exported="true">
</activity>
```

Exercise 3

A 15-minute quiz:

- Create an new activity.
- In this activity, add an Option Menu that has three menu items: "Settings", "Search", "Help" and "Transfer". Each menu item should display a different message when clicked.
- When click "Transfer", jump to a new page.

Have fun: Android provides several AppCompat themes, for example:

- Theme.AppCompat.Light.DarkActionBar
- Theme.AppCompat.Light
- Theme.AppCompat
- ...

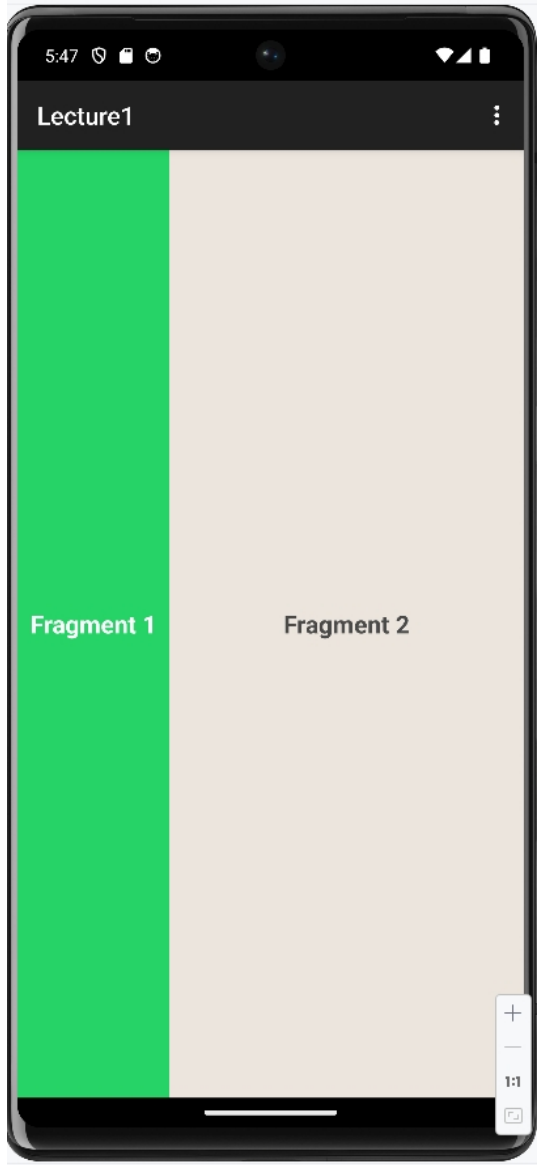
Android UI components: Fragments

`https://developer.android.com/guide/fragments`

Fragments

Fragments:

- A fragment is a portion of user interface in an Activity.
- You can combine multiple fragments in a single activity to build a multi-panel UI
- You can reuse a fragment in multiple activities (like a "sub-activity").
- You can dynamically add or remove fragments during runtime.
- Dividing your UI into fragments makes it easier to modify your activity's appearance at runtime.



Fragments

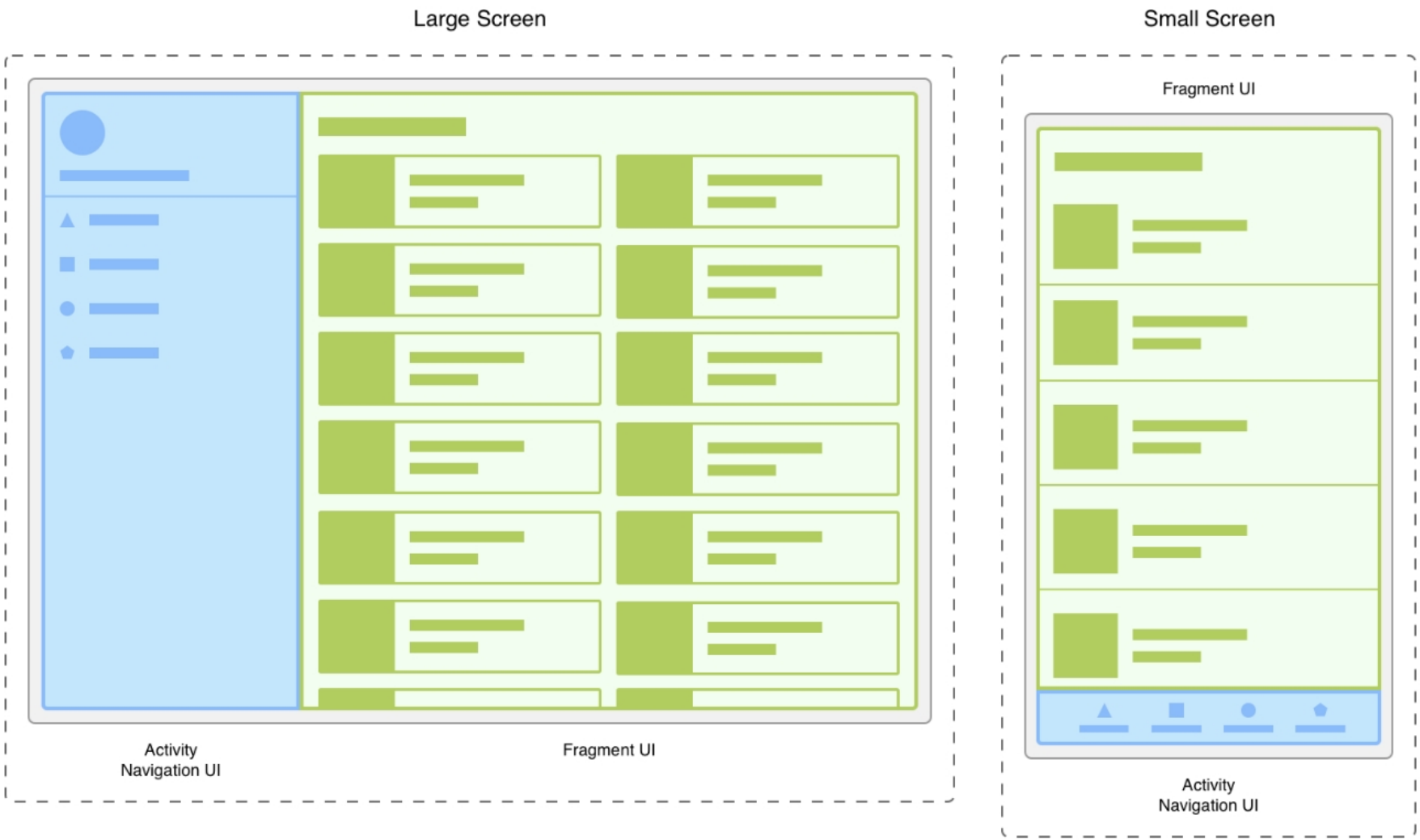


Figure 1. *Two versions of the same screen on different screen sizes. On the left, a large screen contains a navigation drawer that is controlled by the activity and a grid list that is controlled by the fragment. On the right, a small screen contains a bottom navigation bar that is controlled by the activity and a linear list that is controlled by the fragment.*

Fragments creating

Creating and Using Fragments in Android:

1. Creating a New Fragment
2. Writing Fragment Java Code/xml code
3. Setting Up the Fragment Containers
4. Adding Fragments in MainActivity
5. Running and Testing

Step 1 – Creating a New Fragment

1. Right-click on the **package** in your project (e.g., `com.example.app`).
2. Select **New** $\rightarrow$ **Fragment** $\rightarrow$ **Fragment (Blank)**.
3. Name the fragment (e.g., `Fragment1`) and click **Finish**.

This step creates a new Java file and XML layout for the fragment.

Step 2 – Editing the Fragment Layout (XML)

Open `fragment1.xml` and set up a `LinearLayout` with centered text and a green background.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#25D366"
    android:gravity="center"
    android:orientation="vertical">

<TextView
    android:id="@+id/text_fragment1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Fragment 1"
    android:textColor="#FFFFFF"
    android:textSize="20sp"
    android:textStyle="bold" />
</LinearLayout>
```

Step 3 – Writing Fragment Java Code

Define onCreateView to set up the fragment's layout in Fragment1.java.

```
package com.example.lecture1;
import ...

public class Fragment1 extends Fragment {

    public Fragment1() {
        // Required empty public constructor
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container,Bundle savedInstanceState) {
        // fragment 1 layout
        return inflater.inflate(R.layout.fragment1,
            container, false);
    }
}
```

This code inflates the fragment layout when the fragment is created.

Step 4 – Setting Up Fragment Containers in Main Layout

Open activity_main.xml and add FrameLayout containers for the fragments.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="
    http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity5"
    android:id="@+id/main5">

    <FrameLayout
        android:id="@+id/fragment_container_left"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toStartOf="@id/
            fragment_container_right"
        app:layout_constraintWidth_percent="0.3" />
</LinearLayout>
```

Step 5 – Adding Fragments in MainActivity

Use `FragmentTransaction` to attach `ExampleFragment` instances to each container in `MainActivity.java`.

```
// Load the left container
FragmentManager fragmentManager = getSupportFragmentManager
    ();
FragmentTransaction fragmentTransaction = fragmentManager.
    beginTransaction();

fragmentTransaction.replace(R.id.fragment_container_left,
    new Fragment1());

// Load the right container
fragmentTransaction.replace(R.id.fragment_container_right,
    new Fragment2());
fragmentTransaction.commit();
```

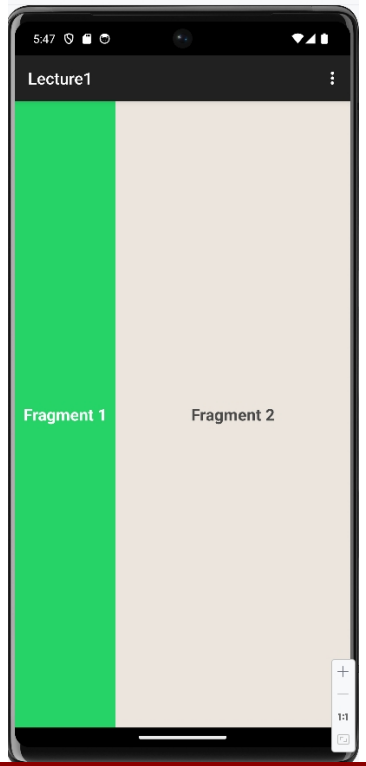
Final Code and Running the App

- Save and build the app.
- Run on an emulator or device.
- Expect two fragments side by side, each showing “Hello from Fragment” in WhatsApp green.

Summary of Fragments

- Fragments are modular UI components that can be embedded within activities.
- `FrameLayout` containers act as placeholders for dynamically added fragments.
- `FragmentTransaction` provides programmatic control over fragment lifecycle events.

Q&A: How can we dynamically change layouts using fragments?



Fragment: How to change layout?

We can do it by only changing `activity_main.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
<!-- up-->
<FrameLayout
    android:id="@+id/fragment_container_left"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintEnd_toStartOf="@id/
        fragment_container_right"
    app:layout_constraintHeight_percent="0.4" />
<!-- down -->
<FrameLayout
    android:id="@+id/fragment_container_right"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toBottomOf="@id/
        fragment_container_left"
    app:layout_constraintHeight_percent="0.6" />
```

Fragment Lifecycle

Usually, you should implement at least the following lifecycle methods:

- **onCreate()**
  - The system calls this when creating the fragment. You should initialize essential components of the fragment.
- **onCreateView()**
  - The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. (null if the fragment does not provide a UI.)
- **onPause()**
  - The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any persistent changes.

Exercise 3

A 20-minute quiz:

- Create an new activity.
- In this activity, add two fragments in Vertical order;
- try to define background colors or some interesting functions for your fragments.

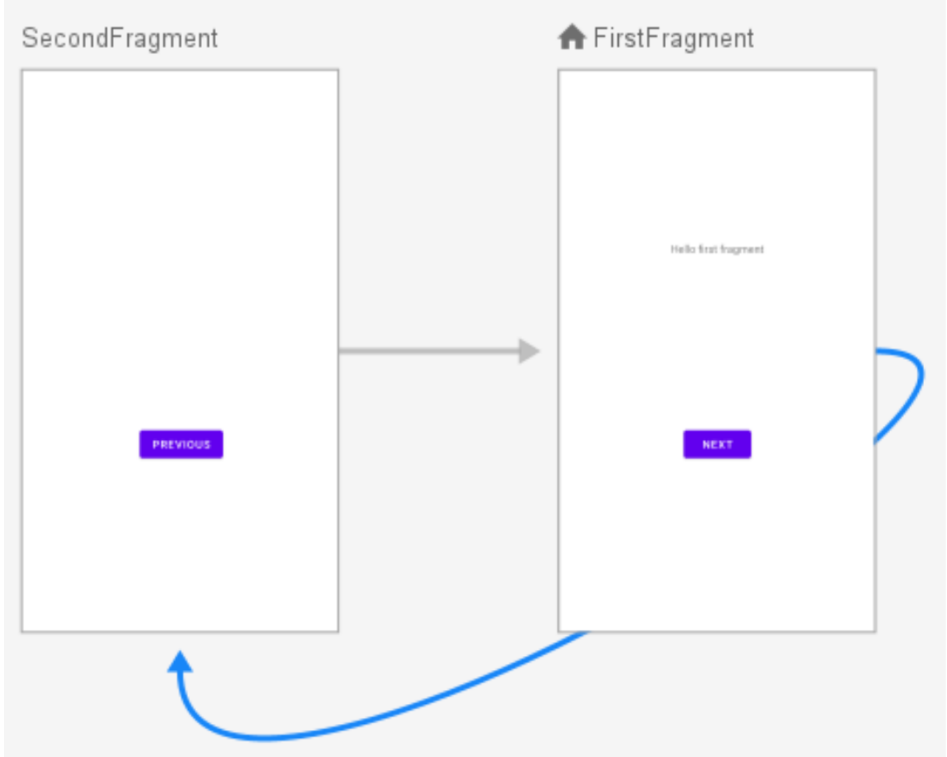
Have fun experimenting! Try different styles and functions within your fragments to make them stand out. Here are some ideas:

- Use different background colors for each fragment.
- Try adding interactive elements like buttons or text inputs.
- Use styles to make each fragment unique!

Android UI components:Navigation

Navigation

- Navigation refers to the interactions that let users navigate across, into, and back out from the different pieces of content within your app.
- The fragments are called "destinations" and connected via actions.
- A navigation graph is a resource file that contains all fragment destinations, all actions and the associated navigation paths.
- The navigation host is an empty container where these fragments destinations are swapped in and out as the user navigates through the app.



Navigation creation

1. Setting up navigation component
2. create a Navigation Graph to define screens and transitions
3. Add NavHostFragment to Activity
4. Create Fragments for Navigation
5. Use NavController to navigate between destinations.

Step 1: Setting Up Navigation Component

- Create a Navigation Graph (‘res/navigation/nav_graph.xml‘) to define screens and transitions.

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/
    res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/nav_graph"
    app:startDestination="@id/fragment1">

    <!-- Fragment 1-->
    <fragment
        android:id="@+id/fragment1"
        android:name="com.example.lecture1.Fragment1"
        android:label="Home">
        <action
            android:id="@+id/action_fragment1_to_fragment2"
            app:destination="@id/fragment2" />
    </fragment>

    <!-- Fragment 2-->
```

Step 2: Adding NavHostFragment to Activity

- Add ‘NavHostFragment‘ to your activity layout file:

```
<fragment
    android:id="@+id/nav_host_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    app:navGraph="@navigation/nav_graph"
    app:defaultNavHost="true"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- ‘navGraph‘ attribute links to your navigation graph.
- ‘defaultNavHost‘ ensures it handles system back navigation.

Step 3: Creating Fragments for Navigation

- Define Fragment classes (e.g., ‘Fragment1‘ and ‘Fragment2‘) and their layouts.
- Add fragments to the navigation graph as destinations:

```
<fragment
    android:id="@+id/fragment1"
    android:name="com.example.lecture1.Fragment1"
    android:label="Fragment 1">
    <action
        android:id="@+id/action_fragment1_to_fragment2"
        app:destination="@id/fragment2" />
</fragment>
```

- Define actions for transitions between fragments.

Step 4: Navigating Between Fragments

- Use ‘NavController‘ to navigate between destinations.
- Example in ‘Fragment1.java‘:

```
Button button = view.findViewById(R.id.buttonNavigate);
button.setOnClickListener(v ->
    Navigation.findNavController(v).navigate(R.id.
        action_fragment1_to_fragment2)
);
```

- ‘NavController‘ is used to initiate actions defined in ‘nav_graph.xml‘.

Example

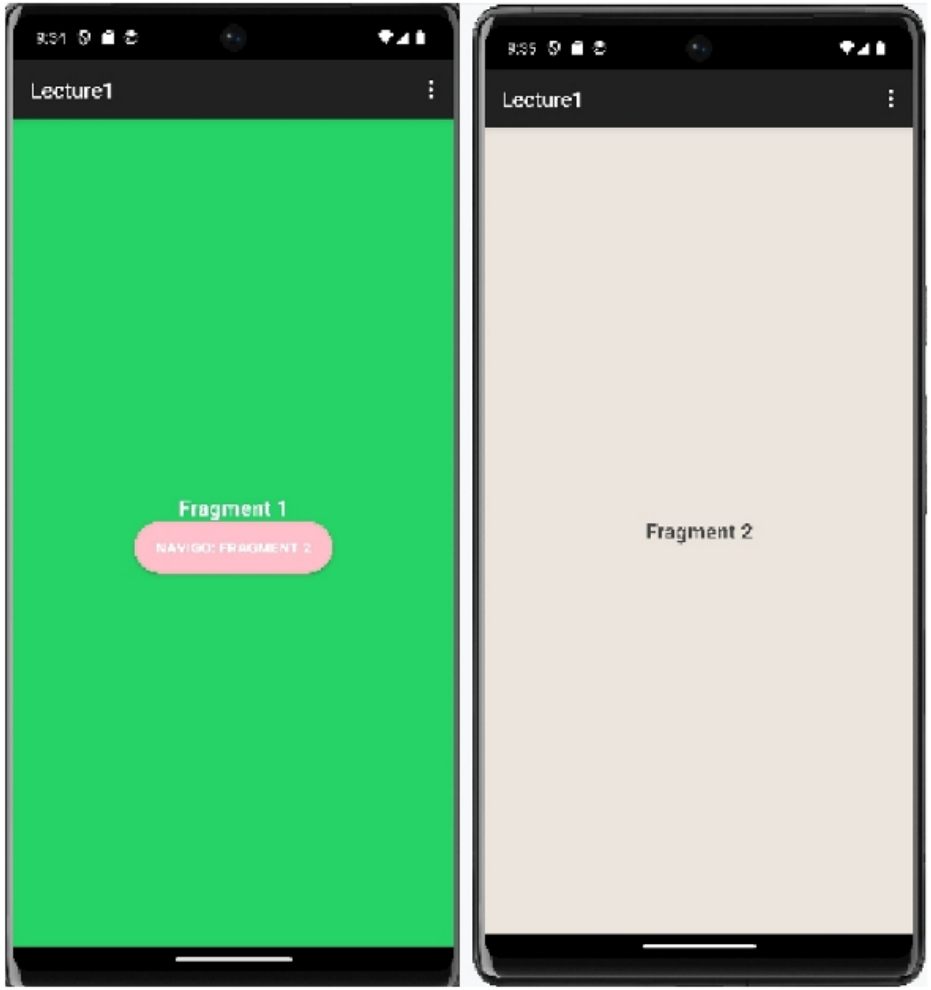


Figure: Navigation between two fragments