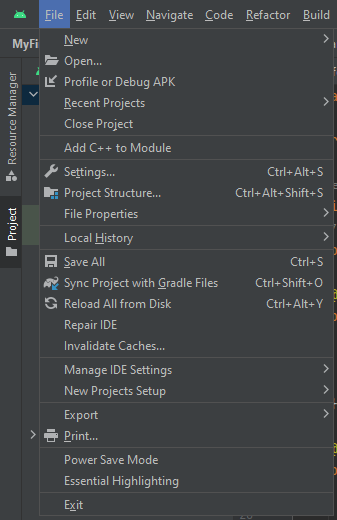
**Class Activity: Fragments**

In this activity, you will be working with fragments in Android Java. This includes creation of fragments, transaction between fragments and creating list fragments.

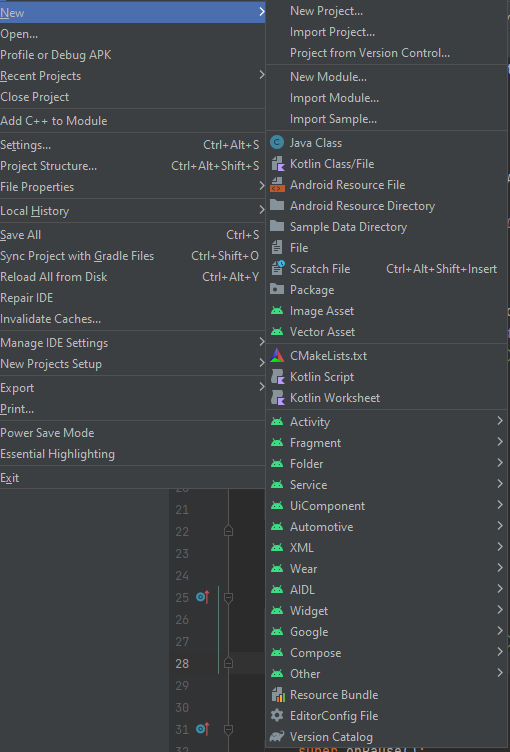
* **Creating Fragments**

In this section, you will acquire the skill on how to create a fragment in Android Studio. The steps are mentioned briefly in the slides. This section includes the detailed steps on the procedure.

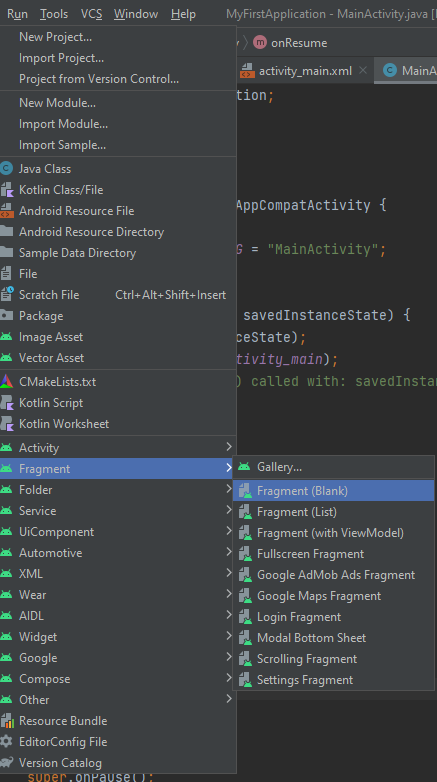
1. In Android Studio, select the File option location on the top left hand corner of the screen.



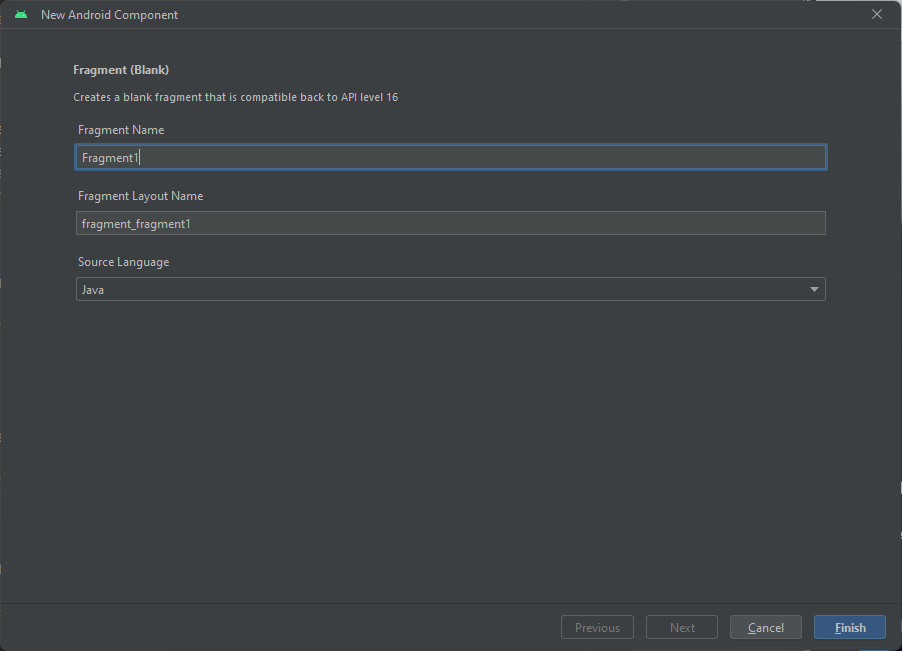
1. Next, select the ‘New’ option in the option menu.



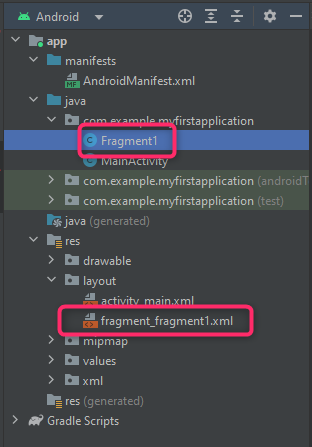
1. Select the ‘Fragment’ option, then select the ‘Fragment(Blank)’ option.



1. A dialog box will appear. Rename the Fragment name to ‘Fragment 1’ and rename the Fragment Layout Name to ‘fragment\_fragment1’. Once you are done, select ‘Finish’

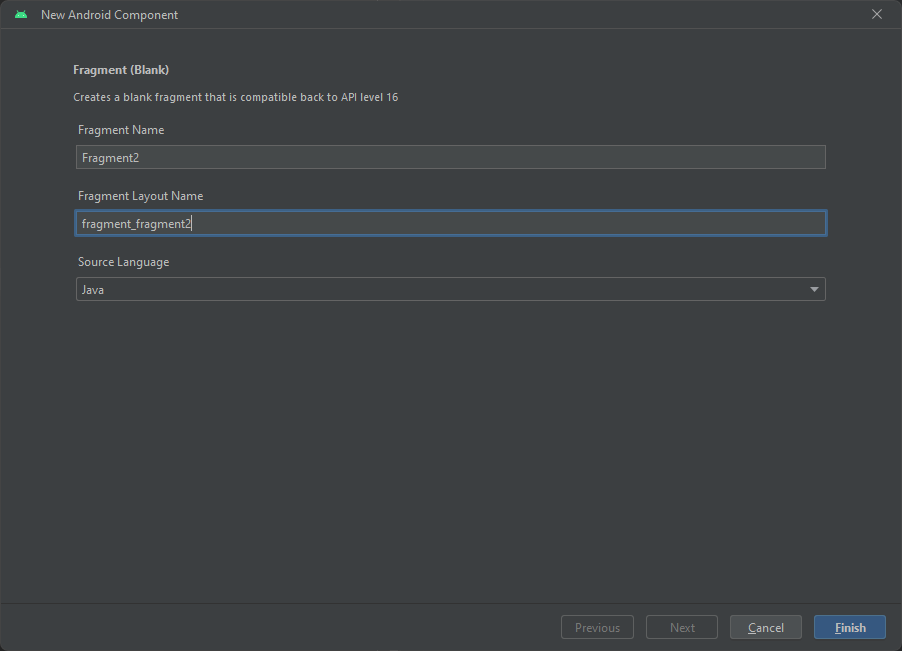


1. Upon successful creation, a Fragment1.java file and fragment\_fragment1.xml file will be created.

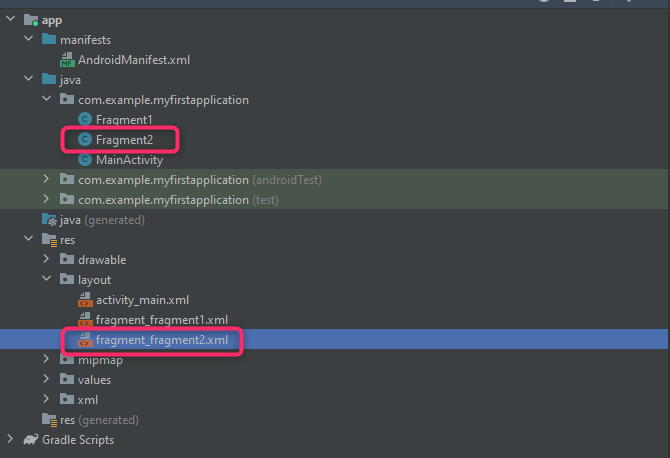


* **Implementing Fragment in a Activity**

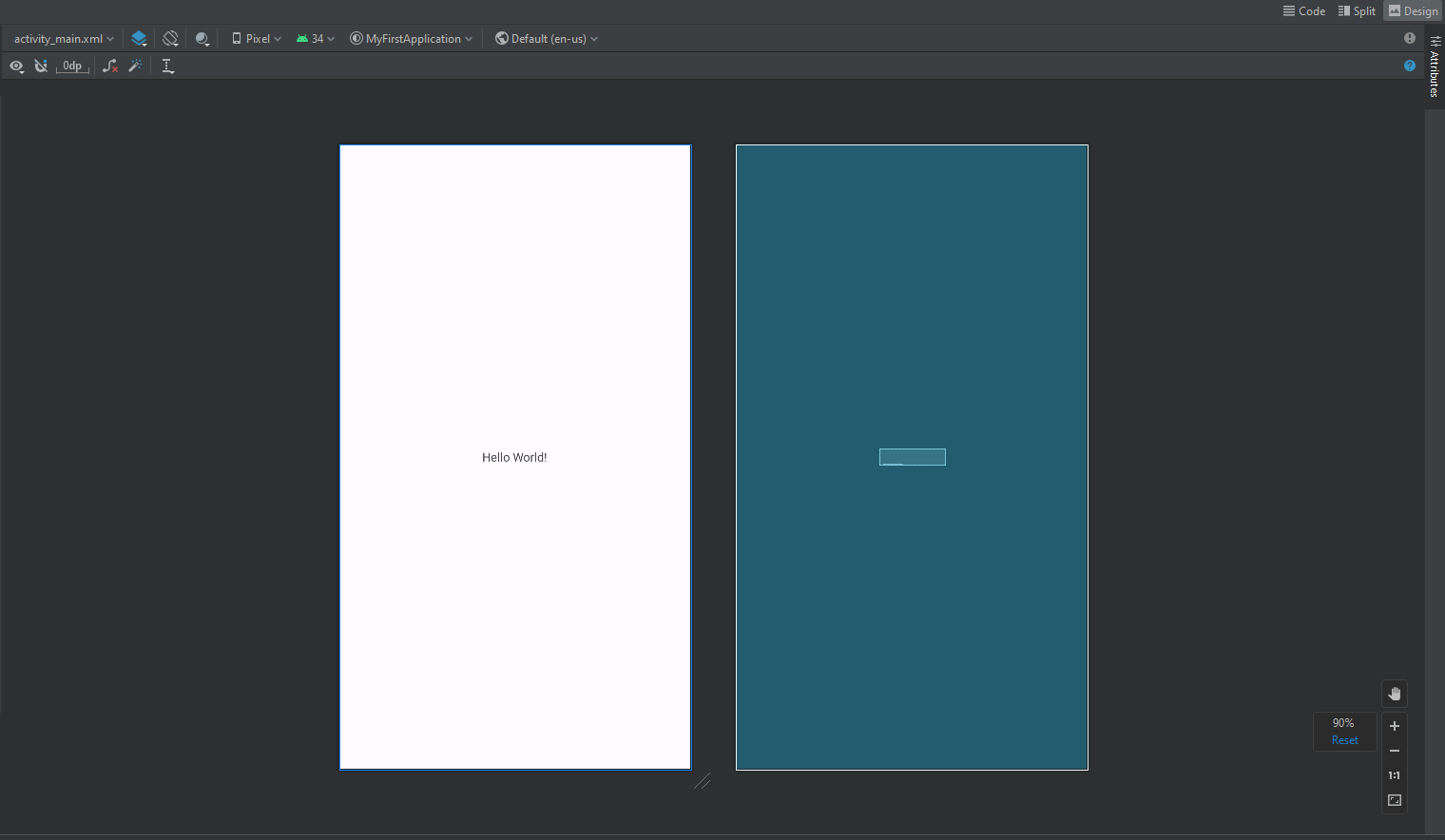
1. Following up, create another fragment and rename the Fragment name to ‘Fragment 2’ and the Fragment Layout Name to ‘fragment\_fragment2’.



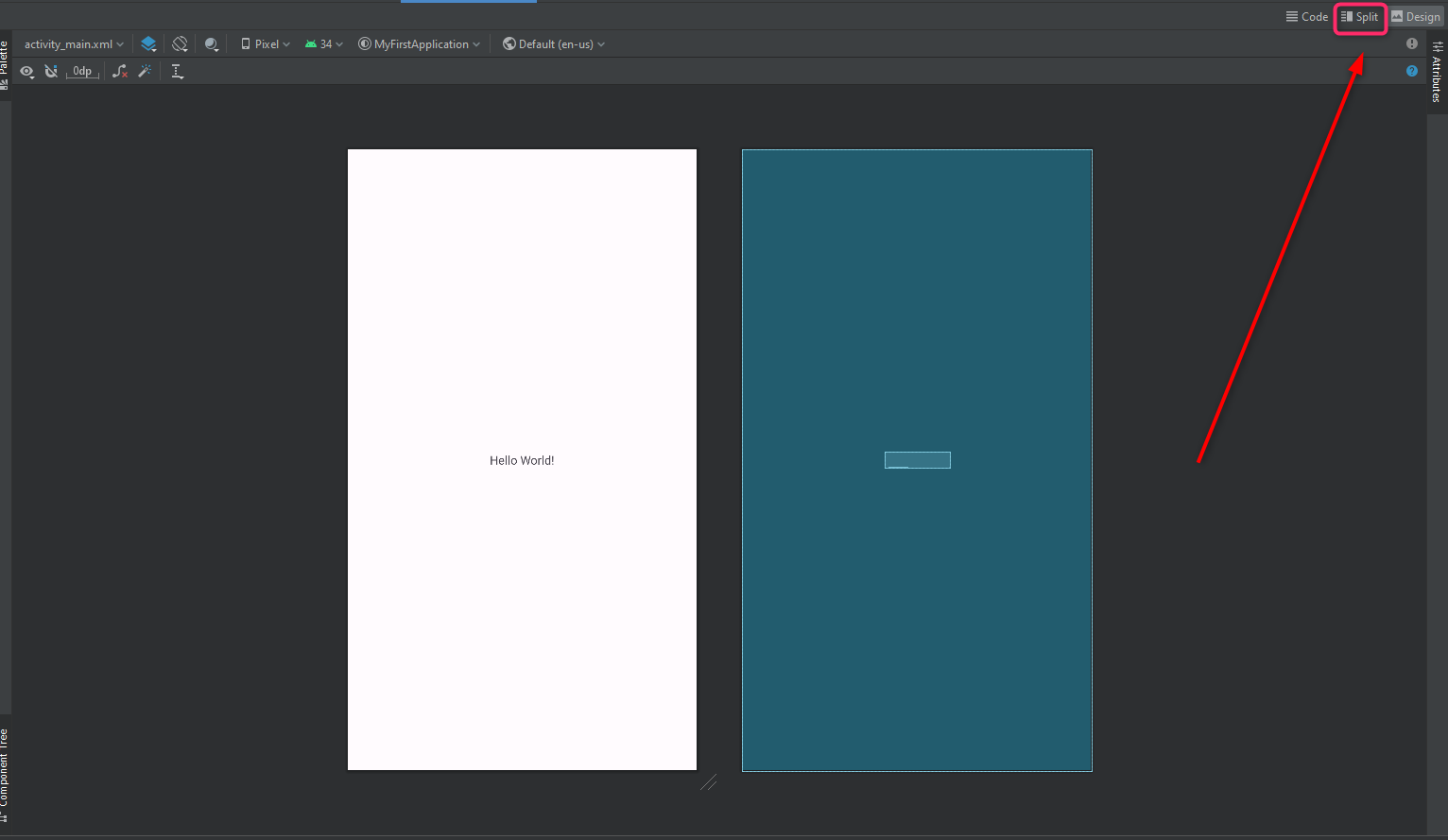
1. Fragment2.java and fragment\_fragment2.xml will show up on the file tab



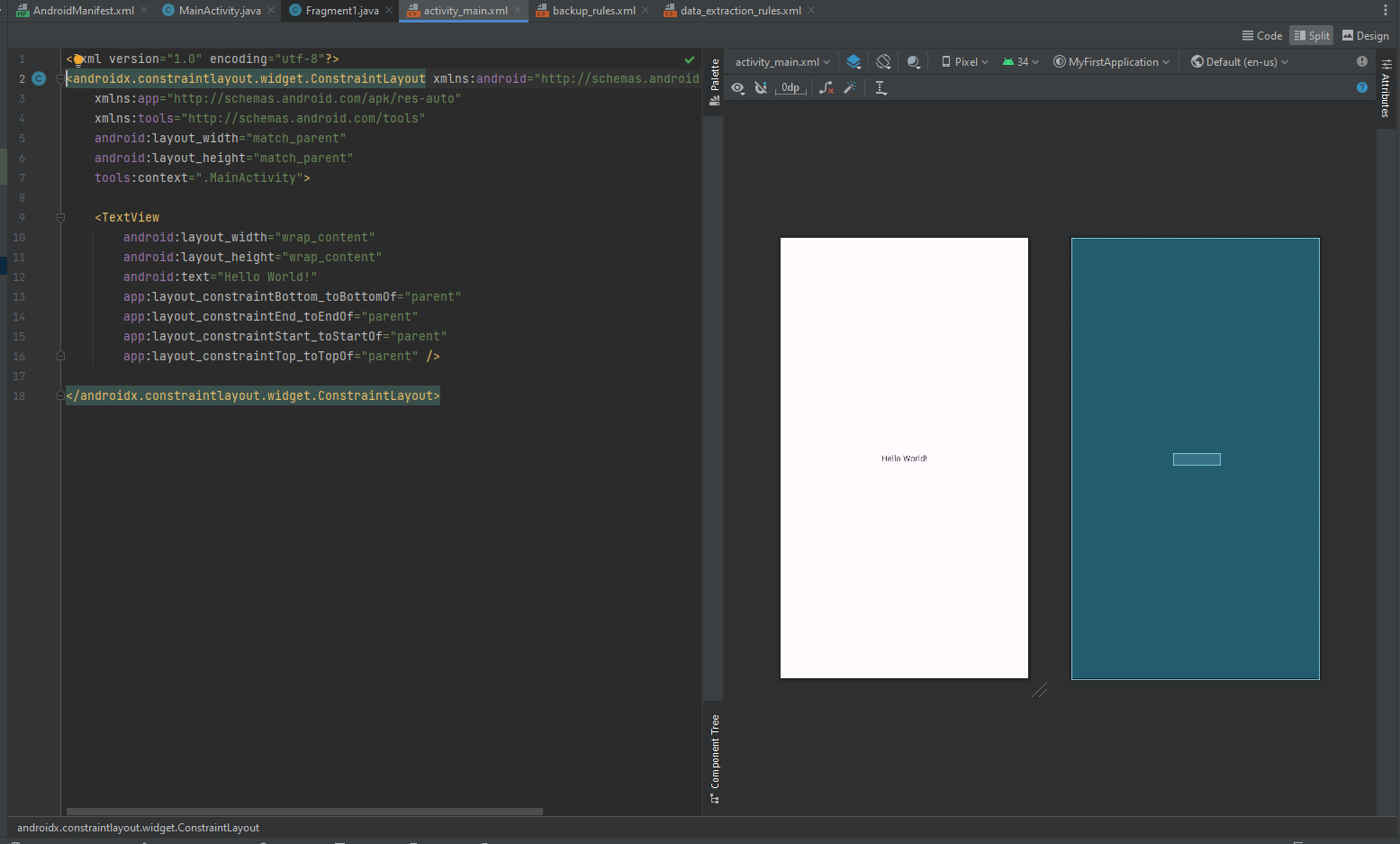
1. Double click on activity\_main.xml file, you will be able to see two screens. This shows that the activity\_main.xml file is in the design tab.



1. To access the code of activity\_main.xml and also see the design tab simultaneously, select the ‘Split’ option on the screen.



1. You will be able to see both the coding screen as well as the design screen.



1. In activity\_main.xml, copy this set of code into the xml file.

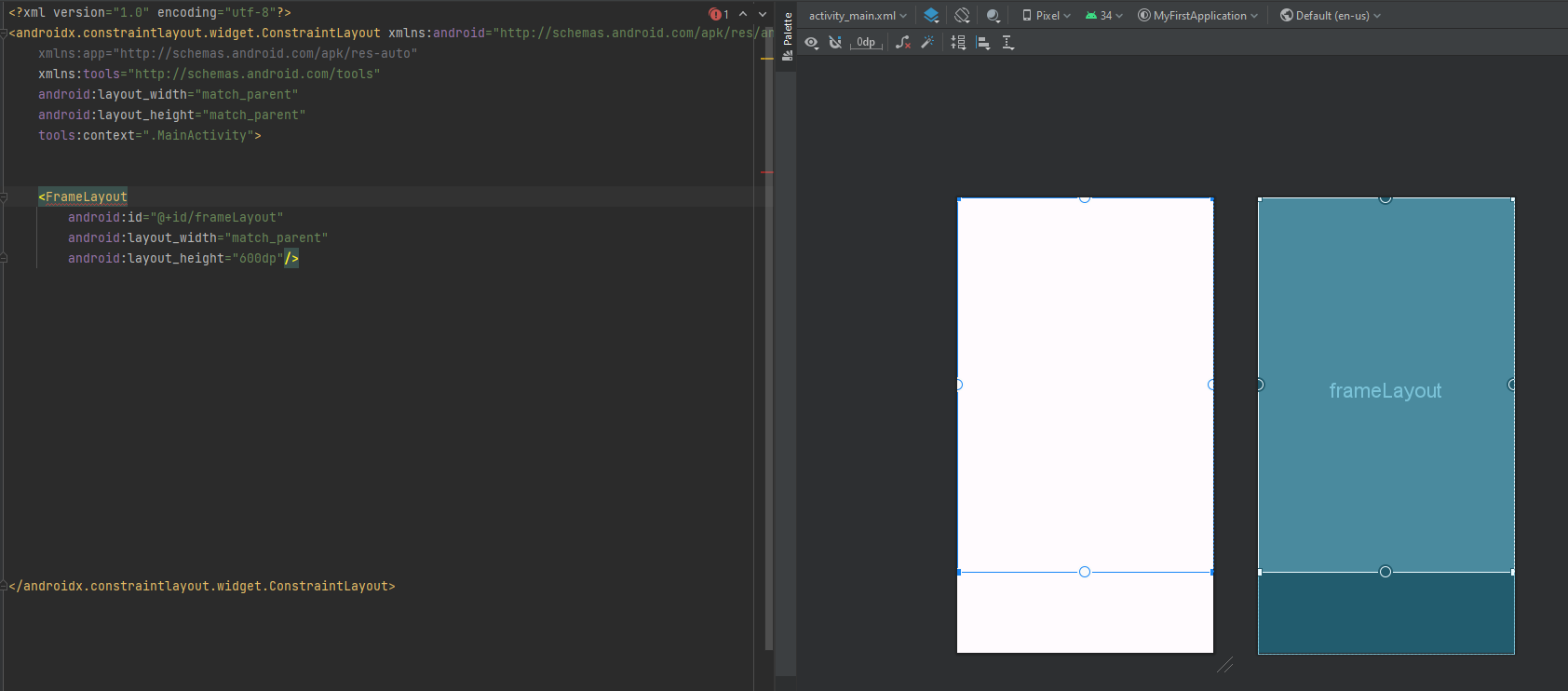
<FrameLayout

android:id="@+id/frameLayout"

android:layout\_width="match\_parent"

android:layout\_height="600dp"

| **Code Explanation:**  A frame layout is created with a ‘layout\_ width’ that matches the parent width. This means that the width of the frame has the same dimension as the activity. The layout\_height of the frame is set to 600dp, this width length to the frame width can be seen in the image shown below. The area shaded in light blue colour represents a width length of 600dp. |
| --- |



1. Next, add two buttons at the bottom of the screen using the following code.

<Button

android:id="@+id/fragment1btn"

android:layout\_width="150dp"

android:layout\_height="60dp"

android:layout\_alignParentBottom="true"

android:layout\_marginStart="50dp"

android:layout\_marginBottom="50dp"

android:backgroundTint="#ffde6a"

android:textColor="#000000"

android:text="Fragment 1" />

<Button

android:id="@+id/fragment2btn"

android:layout\_width="150dp"

android:layout\_height="60dp"

android:layout\_alignParentRight="true"

android:layout\_alignParentBottom="true"

android:layout\_marginEnd="50dp"

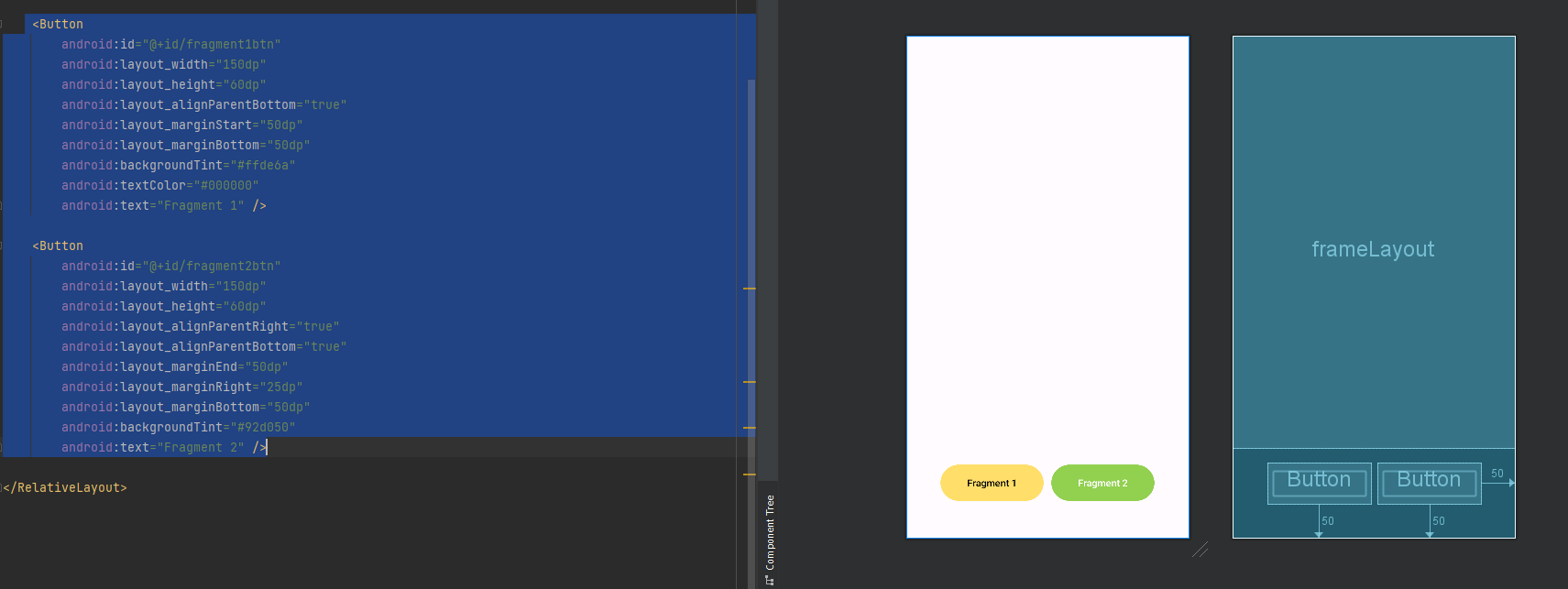
android:layout\_marginRight="25dp"

android:layout\_marginBottom="50dp"

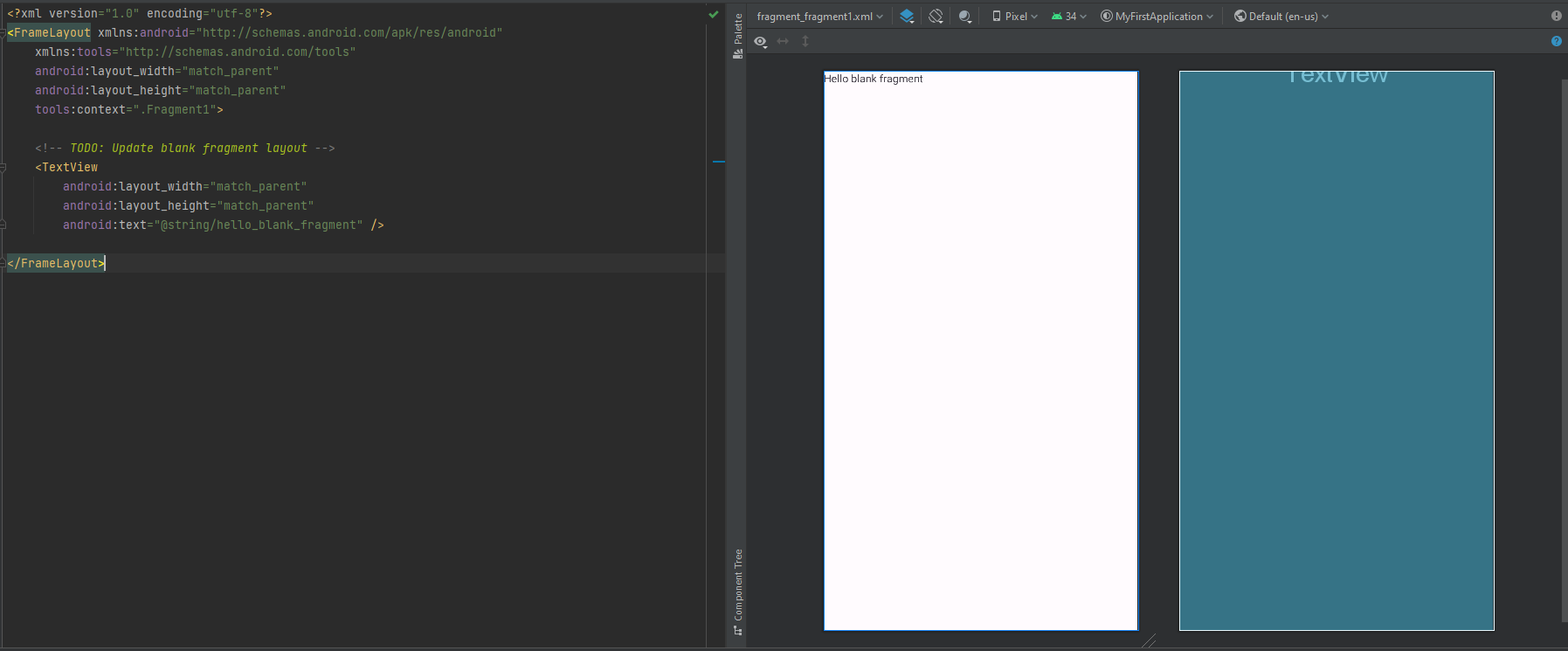
android:backgroundTint="#92d050"

android:text="Fragment 2" />

| **Code Explanation:**  Two buttons are added to the button of the screen with a width of 150dp and a height of 60dp. This determines the size of the button. GIven the properties alignParentRIght, alignParentBottom, layout\_marginStart, layout\_marginBotoom, layout\_marginEnd, these properties work together to position the button in the screen. The backgroundTint and textColor enables you to define the background colour of the button and the text colour. In this case, the button's colour is set to the following hexadecimal stated in the code. Lastly, to set the text for the button, android:text = “text desired” is utilised. |
| --- |



1. Navigate to the ‘fragment\_fragment1.xml’ file and access both the Design tab and the Code tab simultaneously in split mode.



1. Change the text in the fragment from “hello\_blank\_fragment” to “Fragment 1” with a text size of 50 dp and text colour to black colour for better visibility using the following code. The expected outcome is shown in the displayed image below.

<TextView

android:layout\_width="match\_parent"

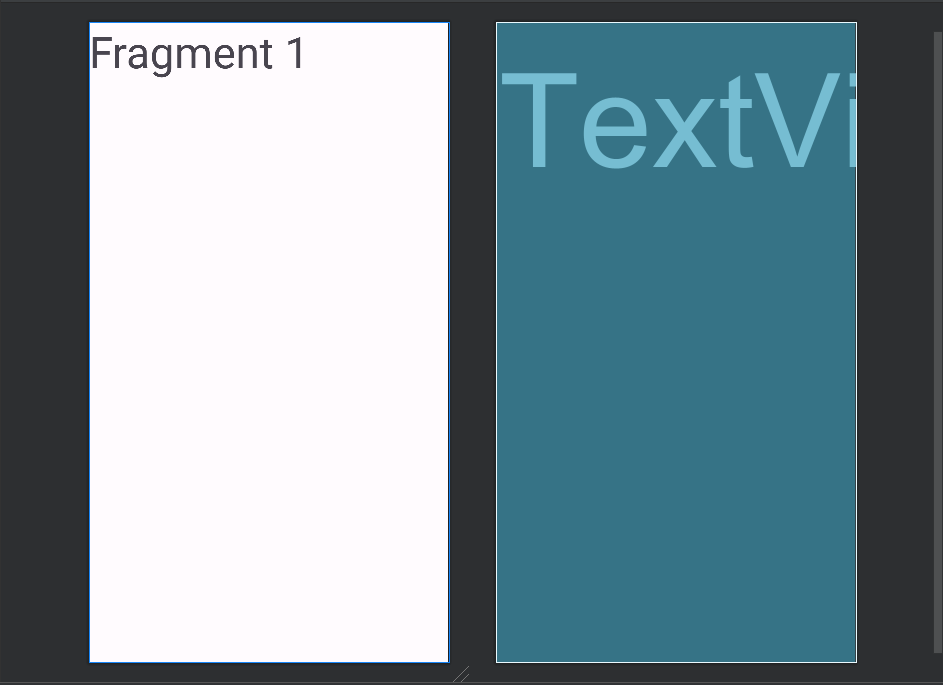
android:layout\_height="match\_parent"

android:text="Fragment 1"

android:textSize="50sp"

android:textColor="@color/black"

/>



1. To position the text at the center of the screen. , switch from using FrameLayout to Relative Layout, and then implement the provided code.

*<?*xml version="1.0" encoding="utf-8"*?>*

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

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".Fragment1">

*<!-- TODO: Update blank fragment layout -->*

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Fragment 1"

android:textSize="50sp"

android:textStyle="bold"

android:layout\_centerHorizontal="true"

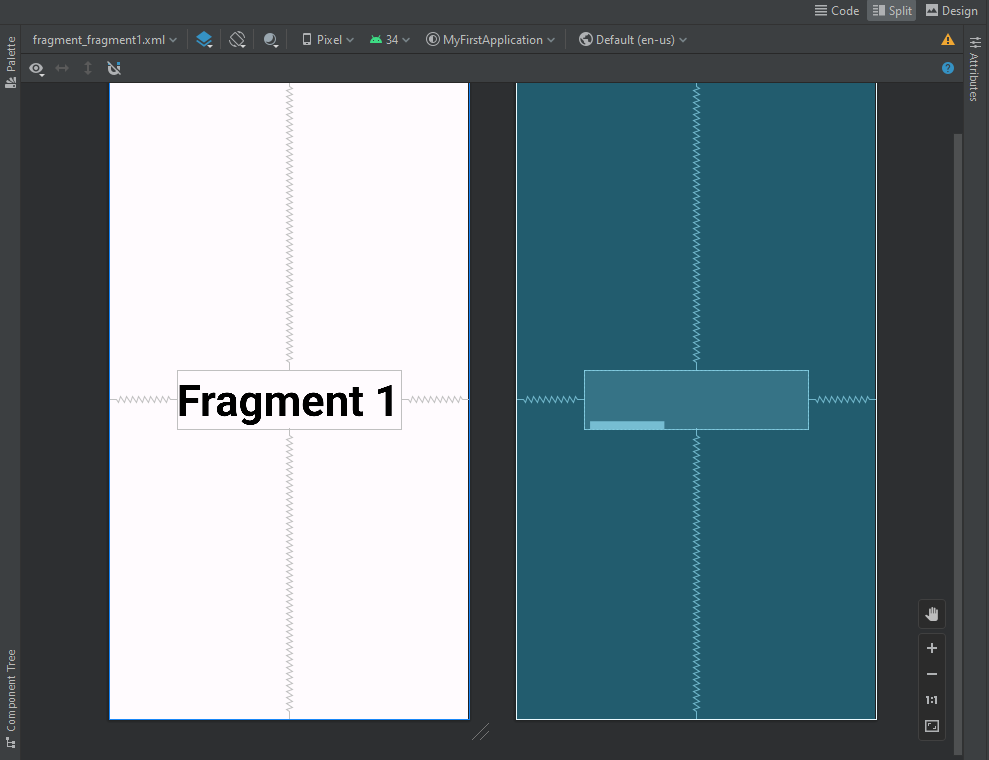
android:layout\_centerVertical="true"

android:textColor="@color/black"

/>

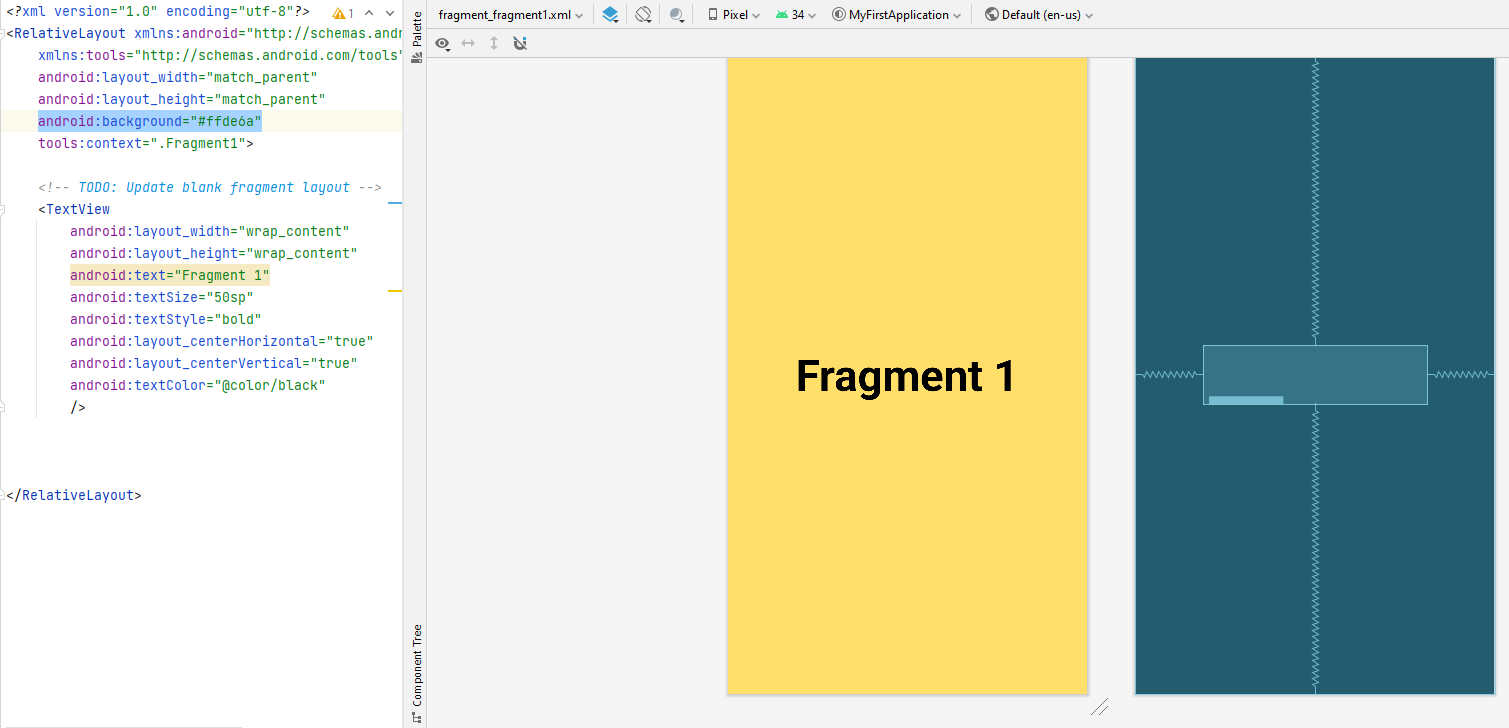
</RelativeLayout>

| **Code Explanation:**  In this case, a Relative Layout is preferred for centering the text in the middle of the screen in Android for its more flexible and versatile positioning capabilities. Attributes such as layout\_centerHorizontal="true" and android:layout\_centerVertical="true" centers the text, “Fragment 1” both horizontally and vertically on the page. The attribute android:layout\_width="wrap\_content" and android:layout\_height="wrap\_content" specifies that the element's width and height should be adjusted to fit its content. This ensures that the text element only takes up as much space as needed. |
| --- |



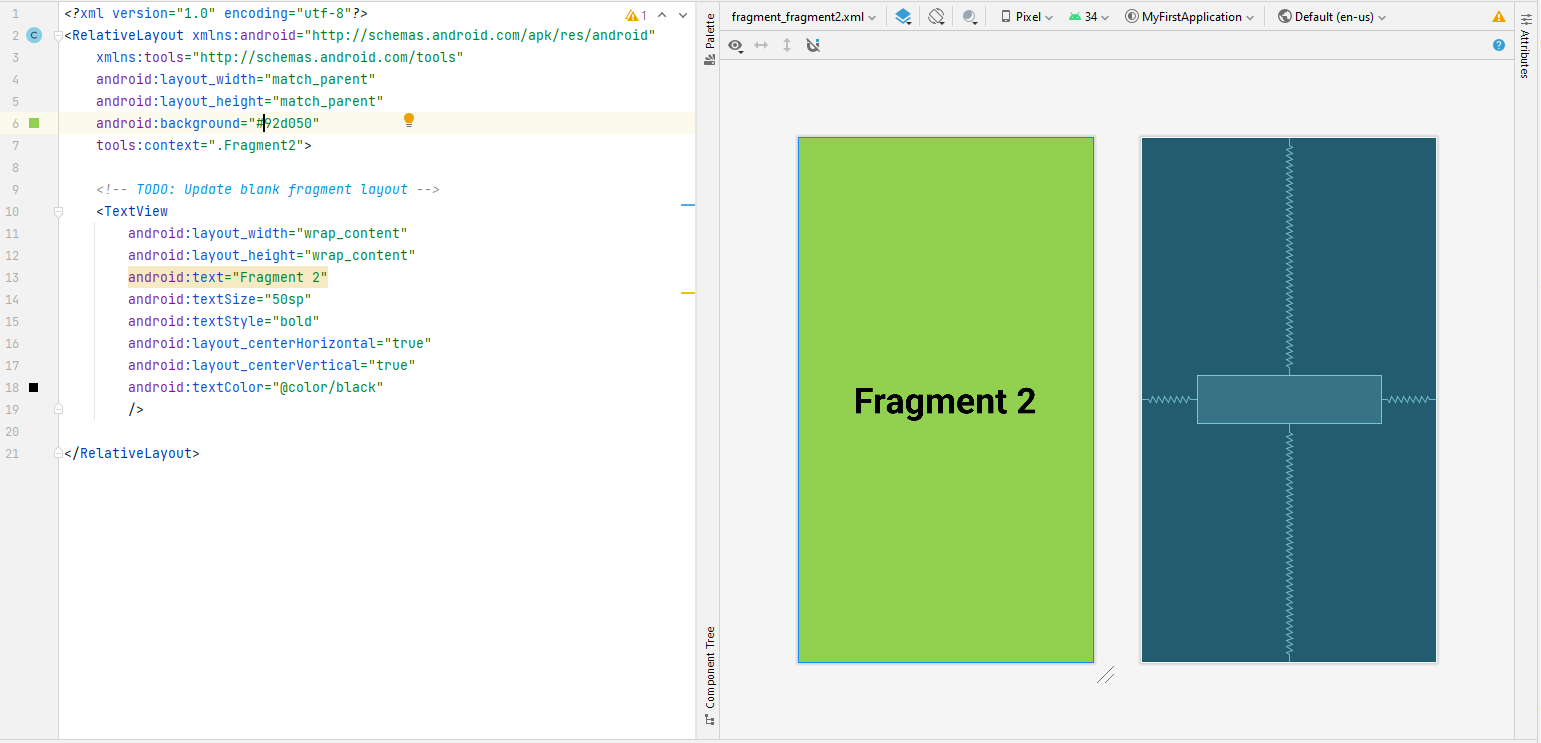
1. Modify the background colour of this fragment to match the colour of the “Fragment 1 ” button for better recognition during the latter stage of Fragment transaction.

android:background="#ffde6a"



**Practice**

Please apply the following procedure to the fragment\_fragment2.xml file. The expected outcome of the layout screen is shown in the image below.



1. Remove the unnecessary code generated by Android Studio upon creation

package com.example.myfirstapplication;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

public class Fragment1 extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

*// Inflate the layout for this fragment*

return inflater.inflate(R.layout.*fragment\_fragment1*, container, false);

}

}

1. Next, modify fragment1.java with the following code

package com.example.myfirstapplication;

import android.os.Bundle;

import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

public class Fragment1 extends Fragment {

View view;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

*// Inflate the layout for this fragment*

view = inflater.inflate(R.layout.*fragment\_fragment1*, container, false);

return view;

}

}

1. Perform the same steps for fragment2.java

| **Code Explanation:**  In the OnCreateView method, the codes does the following:   * Inflates a layout resources file name “**fragment\_fragment1**’ using the provided ‘**inflater**’. This layout defines the user interface for the fragment. In the context of Android development, “**inflating**” refers to the process of converting an XML layout into a view hierarchy or usable view that can be used to display the user interface of an Android application and on the device’s screen. * The ‘**container**’ parameter is the parent view that the fragment’s UI will be attracted to. * The **‘false’** argument indicates to not immediately attach the fragment’s UI to the **‘container’.**  Deferring the attachment allows dynamic UI manipulation as it allows the developer to manipulate the fragment’s UI programmatically before displaying it and this is extremely useful in circumstances where the developer needs to set initial values, configure UI components, or perform other actions on the fragment’s view before it becomes visible.   In the case that there is a fragment that should only be displayed after a user actions, such as clicking a button. By setting it to ‘**false’,**  it waits till the user clicks the button to attach and display the fragment’s UI.   * The inflated view is stored in the created **‘view’** variable object. * The method returns **‘view’**, which will be displayed as the UI of this fragment. |
| --- |

1. In Main Actvity.java, import the necessary imports using this set of code.

import androidx.appcompat.app.AppCompatActivity;

import androidx.fragment.app.Fragment;

import androidx.fragment.app.FragmentManager;

import androidx.fragment.app.FragmentTransaction;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

1. Create two button object variable using this set of code.

Button firstFragmentBtn, secondFragmentBtn;

1. Then, in the OnCreate() method, initialise firstFragmentBtn, secondFragmentBtn button object variable by finding the corresponding buttons by their IDs

firstFragmentBtn = findViewById((R.id.*fragment1btn*));

secondFragmentBtn = findViewById((R.id.*fragment2btn*));

1. Define two Button Click Listeners for **‘firstFragmentBtn’**  and ‘**secondFragmentBtn’.**

**firstFragmentBtn.setOnClickListener(new View.OnClickListener() {**

**@Override**

**public void onClick(View v) {**

**replaceFragment(new Fragment1());**

**}**

**});**

**secondFragmentBtn.setOnClickListener(new View.OnClickListener() {**

**@Override**

**public void onClick(View v) {**

**replaceFragment(new Fragment2());**

**}**

**});**

* **Fragment Transaction**

To handle fragment transactions where transition from one fragment to another when the corresponding button is clicked. The following code is utlised.

private void replaceFragment(Fragment fragment) {

FragmentManager fragmentManager = getSupportFragmentManager();

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

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

fragmentTransaction.commit();

}

| **Code Explanation:**  This method is responsible for the replacement of existing fragments in the application UI with new fragments. Firstly, the code retrieves the **‘FragmentManager’**  which is used to manage fragments in the app. Secondly, a new transaction is started using, **‘FragmentTranscation’**  to define the fragment transaction. Thirdly, the **‘replace’** method is utilised to specify which fragment to replace in which container. This method takes in two arguments, the container Id (R.id.*frameLayout*) and the new fragment (fragment). Lastly, the transaction is committed by calling **‘commit’**  on the fragmentTransaction in order to apply the transaction and effectively updating the UI to display the new fragment by replacing the current fragment with the new one. |
| --- |

Full Code

package com.example.myfirstapplication;

import androidx.appcompat.app.AppCompatActivity;

import androidx.fragment.app.Fragment;

import androidx.fragment.app.FragmentManager;

import androidx.fragment.app.FragmentTransaction;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

private static final String *TAG* = "MainActivity";

Button firstFragmentBtn, secondFragmentBtn;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

firstFragmentBtn = findViewById((R.id.*fragment1btn*));

secondFragmentBtn = findViewById((R.id.*fragment2btn*));

firstFragmentBtn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

replaceFragment(new Fragment1());

}

});

secondFragmentBtn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

replaceFragment(new Fragment2());

}

});

}

private void replaceFragment(Fragment fragment) {

FragmentManager fragmentManager = getSupportFragmentManager();

FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

fragmentTransaction.replace(R.id.*frameLayout*,fragment);

fragmentTransaction.commit();

}

}

**Expected Outcome**

