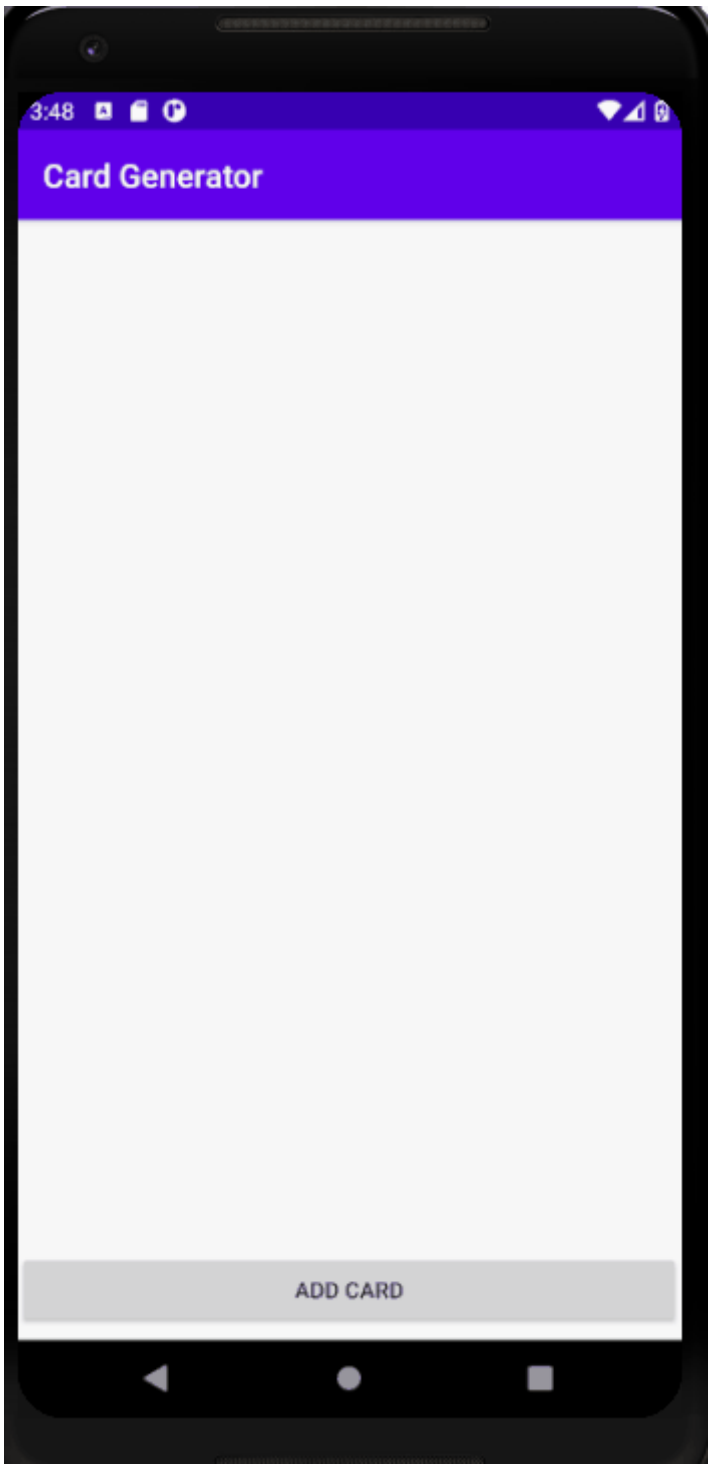
What isÂ RecyclerView?
A RecyclerView is a flexible viewgroup for providing a limited window into a large data set. The purpose of the RecyclerView is to allow information to be presented to the user in the form of a scrollable list. The RecyclerView is significantly more efficient in the way it manages the views that make up a list, essentially reusing existing views that make up list items as they scroll off the screen instead if creating new ones (hence the name â€œrecyclerâ€ •).

Random Card Generator Application

In the following sections, we are going to develop an application that uses RecyclerView to display Cards that are generated randomly. Each card is

implemented as a



CardView.

CardView

The CardView class is a user interface view that allows information to be presented in groups using a card metaphor. Cards are usually presented in lists using a RecyclerView instance and may be configured to appear with

shadow effects and rounded corners. The user interface layout to be presented with a CardView instance is defined within an XML layout resource file and loaded into the CardView at runtime. The CardView layout can contain a layout of any complexity using the standard layout managers such as RelativeLayout and LinearLayout. card_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    card_view:cardBackgroundColor="@android:color/holo_blue_dark"
    card_view:cardCornerRadius="12dp">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp">


        <TextView
            android:id="@+id/suit_id"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_marginStart="58dp"
            android:layout_marginTop="8dp"
            android:layout_toEndOf="@+id/card_id"
            android:textSize="30sp" />

        <TextView
            android:id="@+id/card_id"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentStart="true"
            android:layout_alignParentTop="true"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:textSize="30sp" />
    </RelativeLayout>

</androidx.cardview.widget.CardView>
```

As you can see, the cardview works as a top-level container that has a relative layout as a child element. Inside the relative layout, we have two text views that will be used to display our data. You can replace the relative layout with any layout you do prefer.

RecyclerView

It is a container for rendering a larger data set of views that can be recycled and scrolled very efficiently.Â Â It uses a subclass of RecyclerView.Adapter for providing views that represent items in a data set.
activity_main.xml

```
<?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=".MainActivity">

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.91" />

    <Button
        android:id="@+id/add_item"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Add card"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/guideline" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/my_recycler_view"
        android:scrollbars="vertical"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="8dp"
```

```
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toTopOf="@+id/guideline"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

The code provided above represents the layout file of the main activity. It uses constraint layout as the main container (@line2). The horizontal guideline is used to separate the button from the recycler view. Item.java

```
package com.fit2081.recyclercard;

public class Item {

    private String suit;
    private String card;

    public Item(String suit, String card) {
        this.suit = suit;
        this.card = card;
    }

    public String getSuit() {
        return suit;
    }

    public String getCard() {
        return card;
    }
}
```

Each card consists of a suit and a card number. Item.java is a class that works as a record for one card. Later in the code, we will create an array list of cards using this class. The constructor is used to initialize the two instance variables while the two getters are used to retrieve their values.

RecyclerView.Adapter

The adapter is created as a subclass of the RecyclerView.Adapter class and must, at a minimum, implement the following methods, which will be called at various points by the RecyclerView object to which the adapter is assigned:

- **getItemCount()** â€“ This method must return a count of the number of items that are to be displayed in the list.
- **onCreateViewHolder()** â€“ This method creates and returns a ViewHolder object initialized with the view that is to be used to display the data. This view is typically created by inflating the XML layout file.
- **onBindViewHolder()** â€“ This method is passed the ViewHolder object created by the onCreateViewHolder() method together with an integer value indicating the list item that is about to be displayed. Contained within the ViewHolder object is the layout assigned by the onCreateViewHolder() method. It is the responsibility of the onBindViewHolder() method to populate the views in the layout with the text and graphics corresponding to the specified item and to return the object to the RecyclerView where it will be presented to the user.

MyRecyclerAdapter.java

```
package com.fit2081.recyclercard;

import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import java.util.ArrayList;

public class MyRecyclerAdapter extends
RecyclerView.Adapter<MyRecyclerAdapter.ViewHolder> {

    ArrayList<Item> data = new ArrayList<Item>();

    public void setData(ArrayList<Item> data) {
        this.data = data;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
int viewType) {
```

```
        View v =
LayoutInflater.from(parent.getContext()).inflate(R.layout.card_layout
, parent, false); //CardView inflated as RecyclerView list item
        ViewHolder viewHolder = new ViewHolder(v);
        Log.d("week6App","onCreateViewHolder");
        return viewHolder;
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int
position) {
        holder.cardTv.setText(data.get(position).getCard());
        holder.suitTv.setText(data.get(position).getSuit());
        Log.d("week6App","onBindViewHolder");

    }

    @Override
    public int getItemCount() {
        return data.size();
    }

    public class ViewHolder extends RecyclerView.ViewHolder {
        public TextView suitTv;
        public TextView cardTv;

        public ViewHolder(@NonNull View itemView) {
            super(itemView);
            suitTv = itemView.findViewById(R.id.suit_id);
            cardTv = itemView.findViewById(R.id.card_id);
        }
    }
}
```

The code provided is the implementation of the RecyclerAdapter. It implements the three methods mentioned earlier and the ViewHolder subclass (@line44).Â The data is passed to the adaptor through a method called **setData** (@line18) which is called from the MainActivity.java (@line44). The methodÂ **getItemCount** (@line40) returns the size of the array list which is the number of items to be displayed in the list. The methodÂ onCreateViewHolder inflates the card's layout that we have implemented earlier each time a new card is required (@line25). It the passes v, which is a reference to the card layout to the contractor of ViewHolder local class. At line 28, the method returns the view holder object that will be an input to the next method **onBindViewHolder**. In the

latter method, we receive two parameters, a viewer holder and a position. Our job is to retrieve the data at that position and place it in that view holder. MainActivity.java

```
package com.fit2081.recyclercard;

import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import java.util.ArrayList;
import java.util.Random;

public class MainActivity extends AppCompatActivity {

    ArrayList<Item> data = new ArrayList<>();
    String suits[] = {"Hearts", "Diamonds", "Clubs", "Spade"};
    String cards[] = {"2", "3", "4", "5", "6", "7", "8", "9", "10",
"Jack", "Queen", "King", "Ace"};

    Button btn;
    RecyclerView recyclerView;
    RecyclerView.LayoutManager layoutManager;
    MyRecyclerAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn = findViewById(R.id.add_item);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                addItem();
            }
        });

        recyclerView = findViewById(R.id.my_recycler_view);
```

```
        layoutManager = new LinearLayoutManager(this);  //A
RecyclerView.LayoutManager implementation which provides similar
functionality to ListView.
        recyclerView.setLayoutManager(layoutManager);   // Also
StaggeredGridLayoutManager and GridLayoutManager or a custom Layout
manager


        adapter = new MyRecyclerAdapter();
        adapter.setData(data);
        recyclerView.setAdapter(adapter);

    }

    public void addItem() {
        Random random = new Random();
        int randCard = random.nextInt(cards.length);
        int randSuit = random.nextInt(suits.length);
        Item item = new Item(suits[randSuit], cards[randCard]);
        data.add(item);
        adapter.notifyDataSetChanged();

    }

}
```

In the code above, we have created three entities, a recycler view (@line37), dataset (@line16), and an adapter (@line43). The adapter works as a bridge between the recyclerview and the dataset (@line43,44).

Each time the data gets changed, you should notify the adapter by callingÂ adapter.notifyDataSetChanged() in order to update the recycler view (i.e. the UI)

References:

- Smyth, Neil. Android Studio 3.5 Development Essentials - Java Edition: Developing Android 10 (Q) Apps Using Android Studio 3.5, Java and Android Jetpack . Payload Media, Inc.. Kindle Edition.
- https://developer.android.com/

Question: What is JSON? Answer: JSON is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is discovered by Douglas Crockford.

Example: represent the following data items in JSON: Name is Tim, Age is 23, Address is Melbourne, and units taken are: FIT1051, FIT2095, and FIT2081.

```
{
   "name":"Tim",
   "age":23,
   "address":"Melbourne",
   "units":[
      "FIT1051",
      "FIT2095",
      "FIT2081"
   ]
}
```

Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

Nested Data

Arrays and objects can be nested in JSON format. For example:

```
{
   "firstname":"Tim",
   "lastname":"John",
   "websites":[
      {
         "description":"Company",
         "url":"http://company.com",
         "live":false
      },
      {
         "description":"School",
         "url":"http://school.com",
```

```
        "live":true
      }
   ]
}
```

References

- https://www.digitalocean.com/community/tutorials/an-introduction-to-json
- https://www.w3schools.com/js/js_json_syntax.asp
- https://www.whoishostingthis.com/resources/json-resource/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

Q) Shared Preferences saves primitive data types only. How can I save an array of objects in shared preferences?

A) Convert the array of objects into a string using Gson.

Gson

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of. Gson provides simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa. Add dependency

```
dependencies {
  implementation 'com.google.code.gson:gson:2.8.6'
}
```

Create a new instance of Gson

```
Gson gson = new Gson();
```

Now, let's assume you have an array of objects

```
ArrayList<Item> db = new ArrayList<>();
db.add(item1);
db.add(item2);
db.add(item3);
```

Convert the ArrayList object into as string (i.e. json)

```
String dbStr = gson.toJson(db);
```

Now, you can push the string into your database (or shared preferences)

```
SharedPreferences.Editor edit = sP.edit();
edit.putString(DB_KEY, dbStr);
edit.apply();
```

Now, let's convert the string back to the ArrayList data type

```
Type type = new TypeToken<ArrayList<Item>>() {}.getType();
db = gson.fromJson(dbStr,type);
```

The method **fromJson** converts the data type of the first parameter into the type that is specified in the second parameter.

Question: I want to have an activity with two tasks such that they have different layouts (XML), different logic (Java), and can be reused in other activities. Answer: You have to use Fragments.

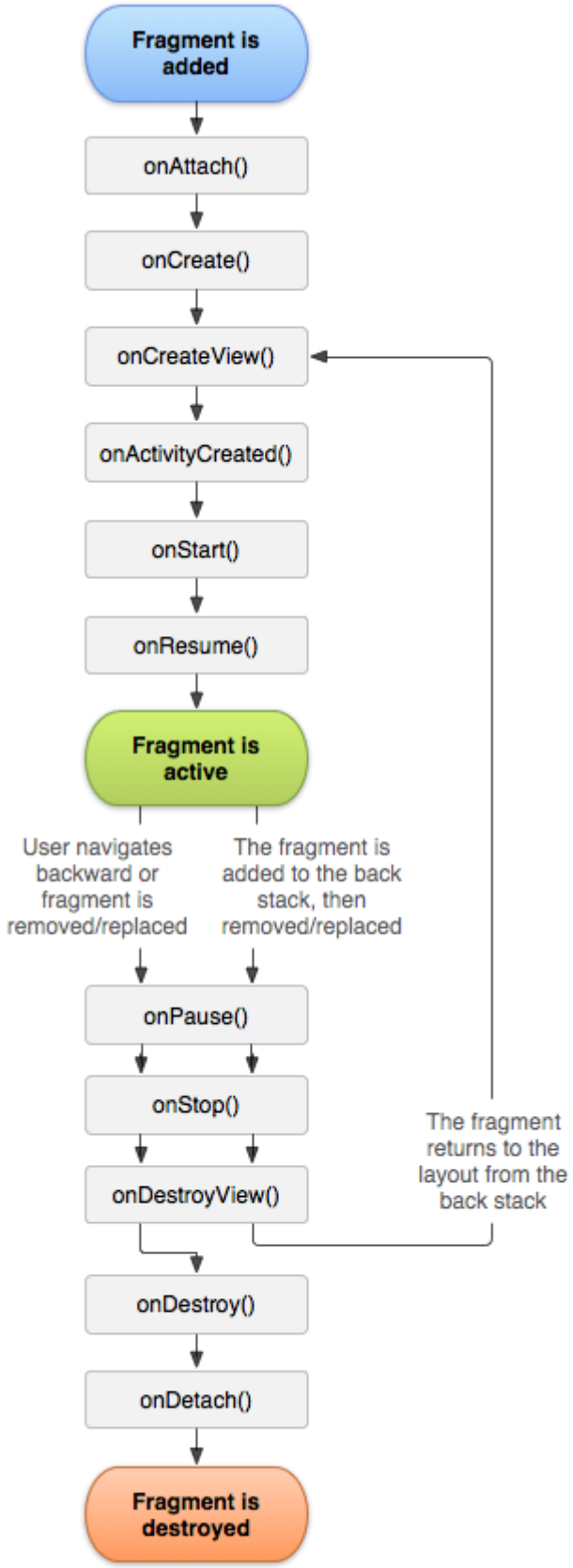
What are Fragments?

A Fragment represents a behaviour or a portion of user interface in a FragmentActivity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "subactivity" that you can reuse in different activities). [1]

What are the differences between FragmentsÂ and Activities?

- Activity is an applicationÂ component that represents a full screen. A fragment is a portion of userÂ interface in an activity.
- An activity may contain 0 or multiple fragments.
- Fragments can be reused in multiple activities.
- A fragment can't exist independently. It should be always part of an activity.
- A fragment can be added or removed while the activity (the host) is running.
- A fragment has its own lifecycle events.

Fragment Life Cycle

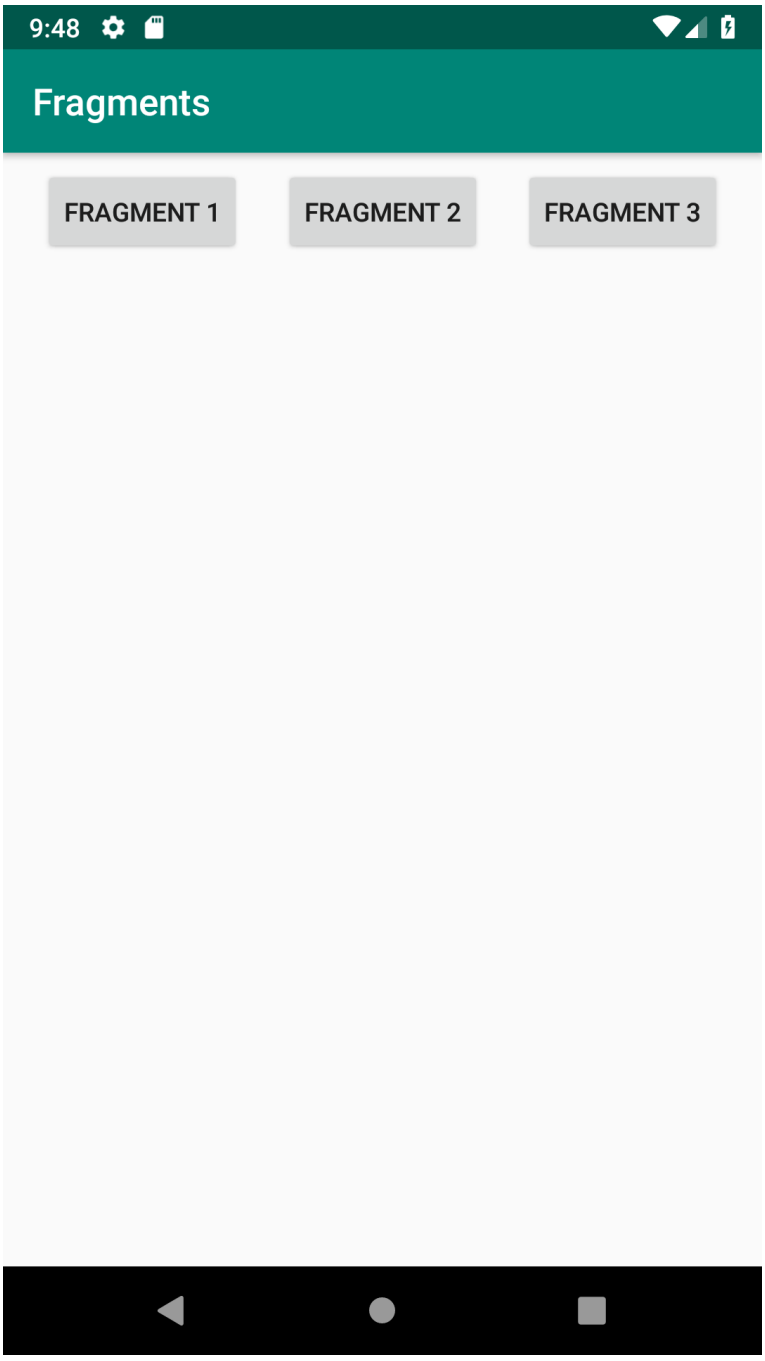


onAttach() Called when the fragment has been associated with the activity (the Activity is passed in here) onCreate() The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the

fragment that you want to retain when the fragment is paused or stopped, then resumed. 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. You can return null if the fragment does not provide a UI. onActivityCreated() Called when the activity's onCreate() method has returned. onStart() This method is called once the fragment gets visible. onPause() The system calls this method as the first indication that the user is leaving the fragment (though it doesn't always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back). onDestroyView() Called when the view hierarchy associated with the fragment is being removed. onDetach() Called when the fragment is being disassociated from the activity.

Example: Build an application that consists of one activity and three fragments. The activity should have three buttons which are used to switch the fragments. Each fragment has its own layout and logic.

App source code: CLICK HERE The layout of the



activity:

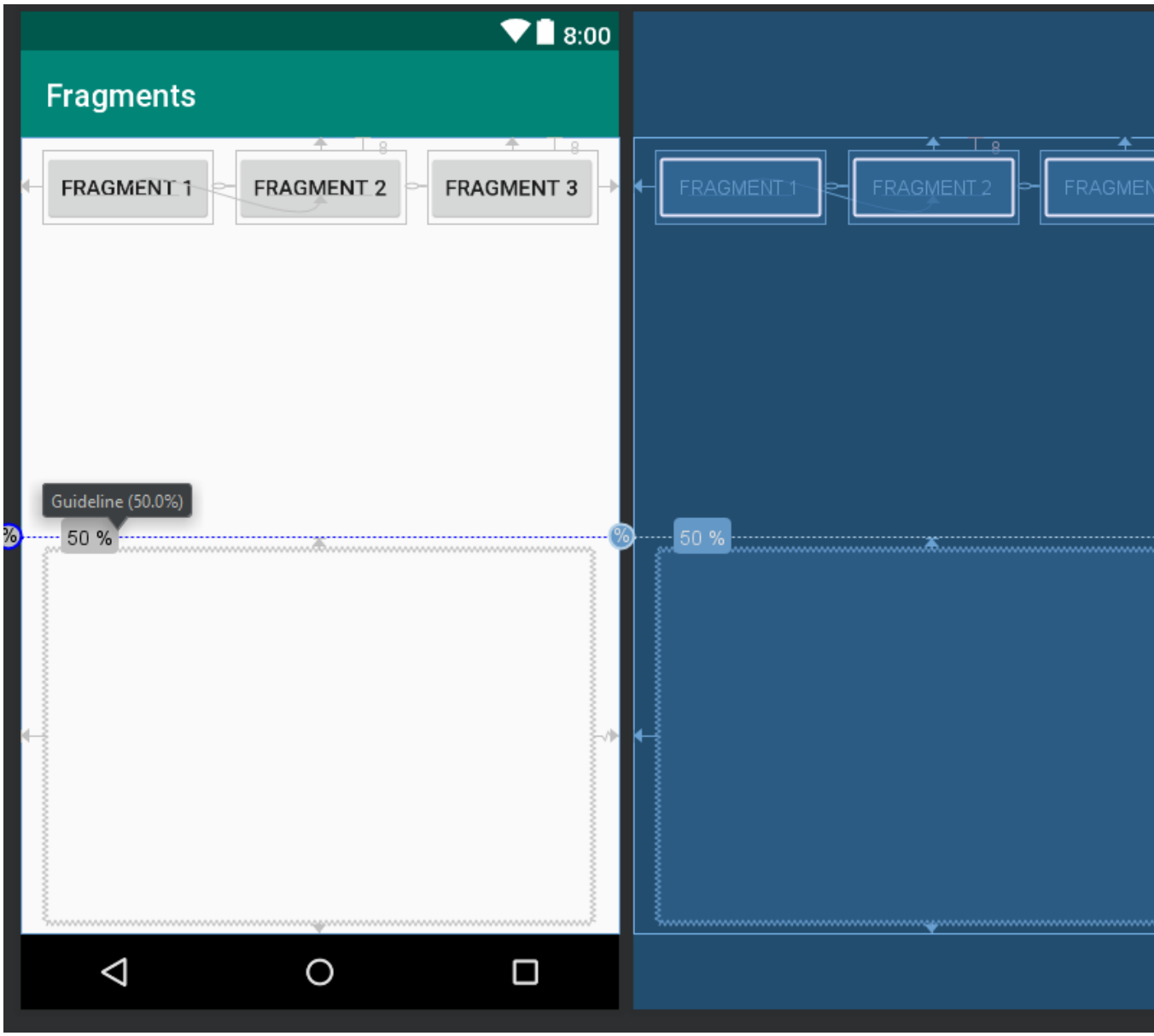
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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=".MainActivity">

    <Button
```

```
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="handleBtn1"
    android:text="Fragment 1"
    app:layout_constraintBaseline_toBaselineOf="@+id/button2"
    app:layout_constraintEnd_toStartOf="@+id/button2"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintHorizontal_chainStyle="spread"
    app:layout_constraintStart_toStartOf="parent"/>

<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:onClick="handleBtn2"
    android:text="Fragment 2"
    app:layout_constraintEnd_toStartOf="@+id/button3"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toEndOf="@+id/button"
    app:layout_constraintTop_toTopOf="parent"/>

<Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:onClick="handleBtn3"
    android:text="Fragment 3"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toEndOf="@+id/button2"
    app:layout_constraintTop_toTopOf="parent"/>

<FrameLayout
    android:id="@+id/frag1"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toTopOf="@+id/guideline"></FrameLayout>


    <android.support.constraint.Guideline
        android:id="@+id/guideline"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent="0.5"/>

</android.support.constraint.ConstraintLayout>
```

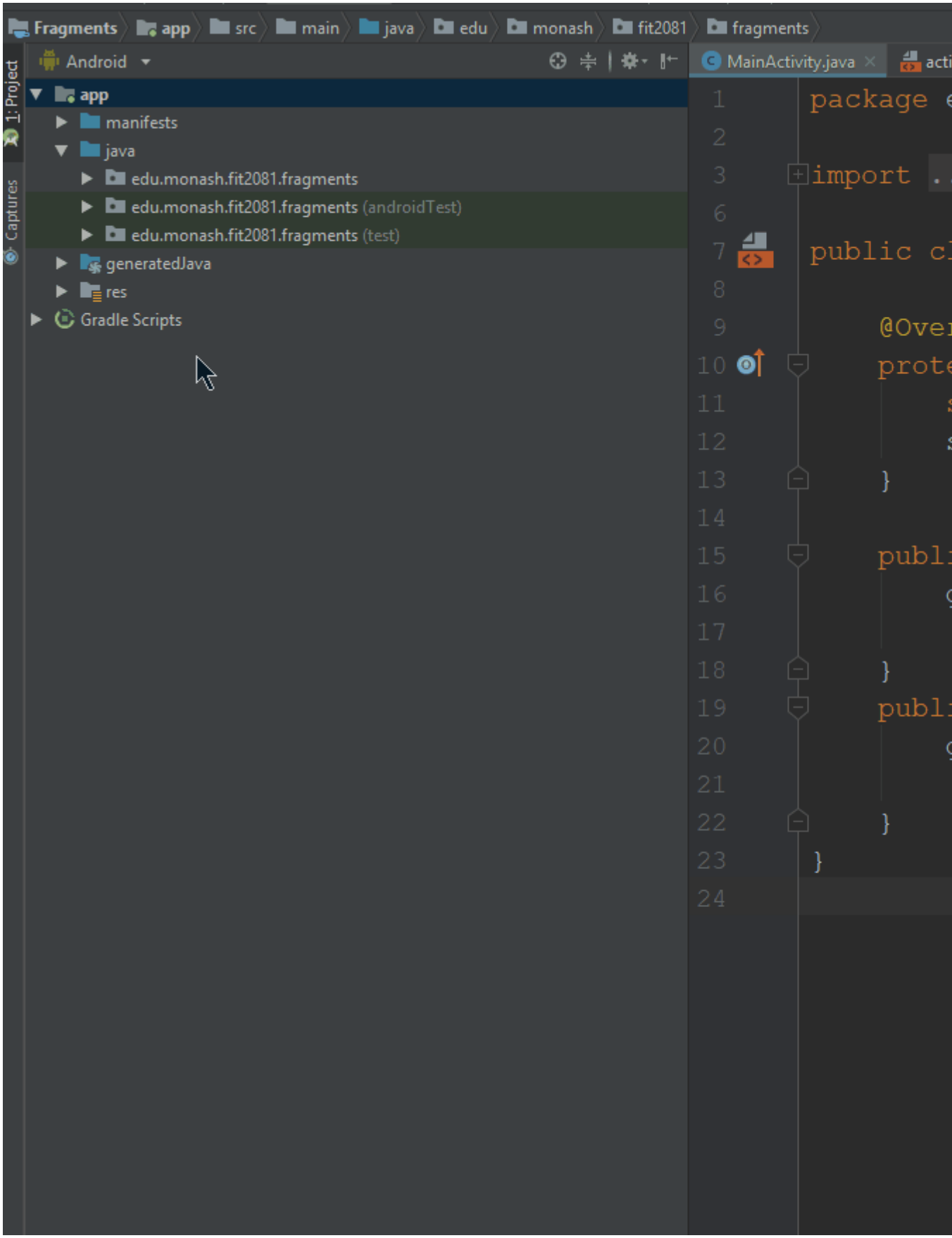
IMPORTANT: The framelayout in lines (46-58) with its **id R.id.frag1Â** works as a container for the fragments.
and the design mode:



Note the following:

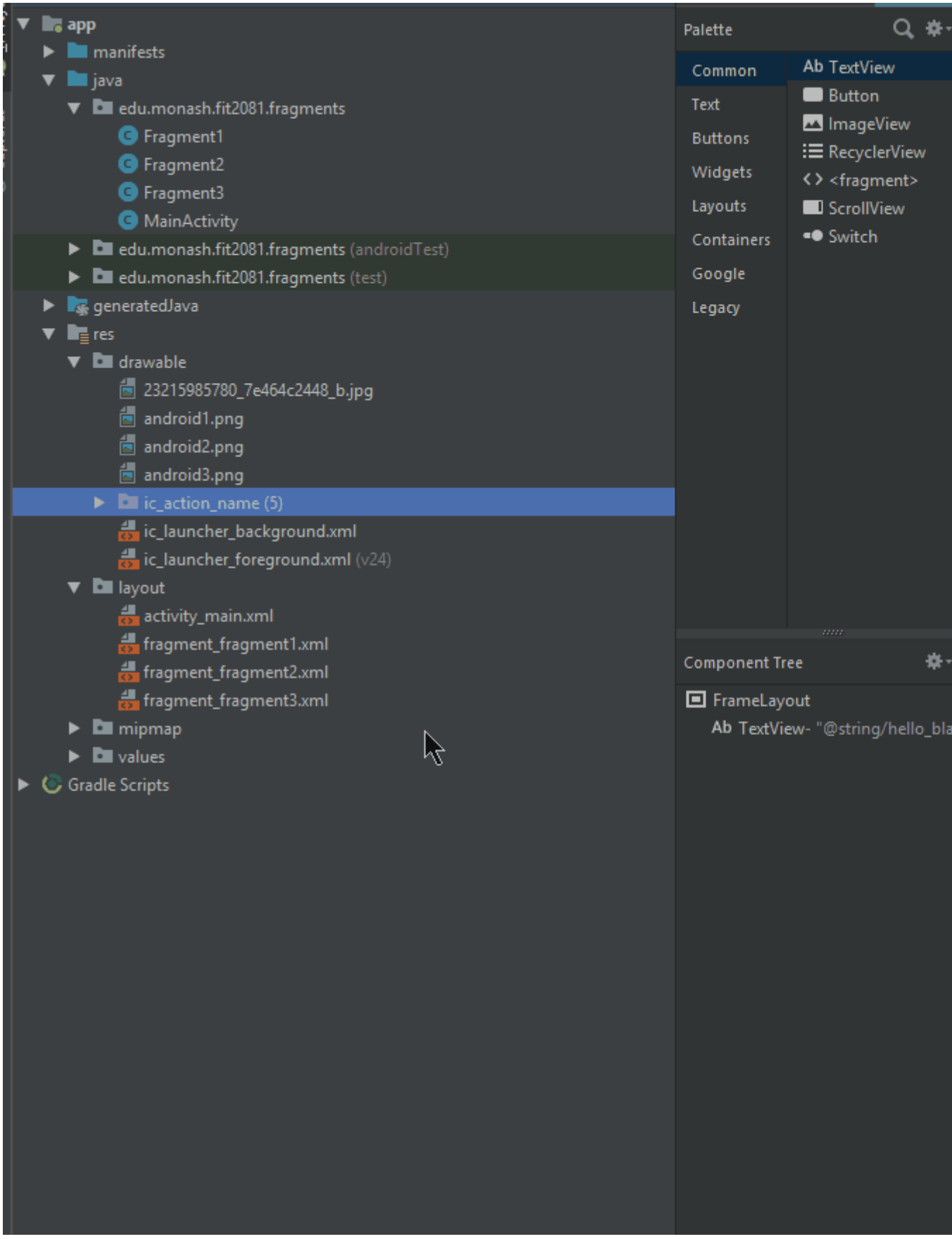
- The three buttons have been chained to expand equally to fill the available space.
- There is a Horizontal Guideline that is positioned at the center (50%) of the hight.
- The box at the lower half is a frame layoutr that works as a container for the fragments.

Now, lets add a new fragment:

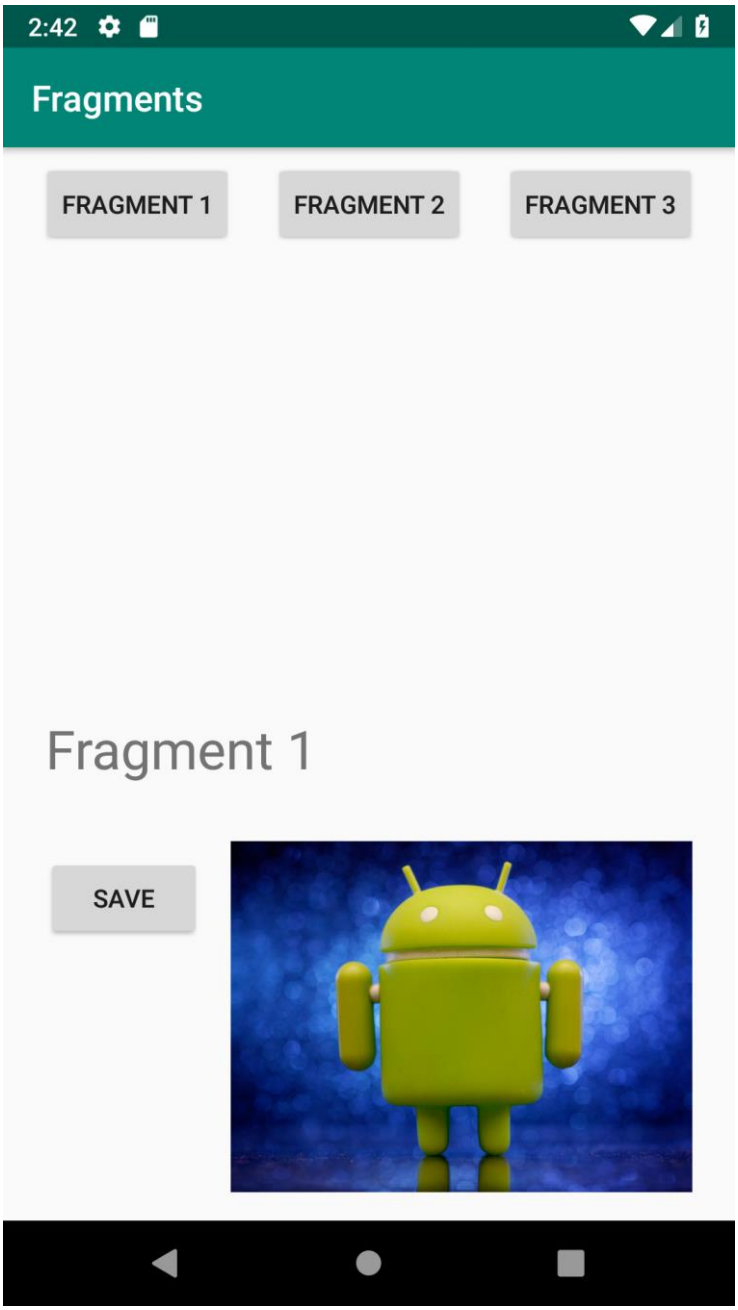


by default, the layout of new fragments is framelayout. To change it to

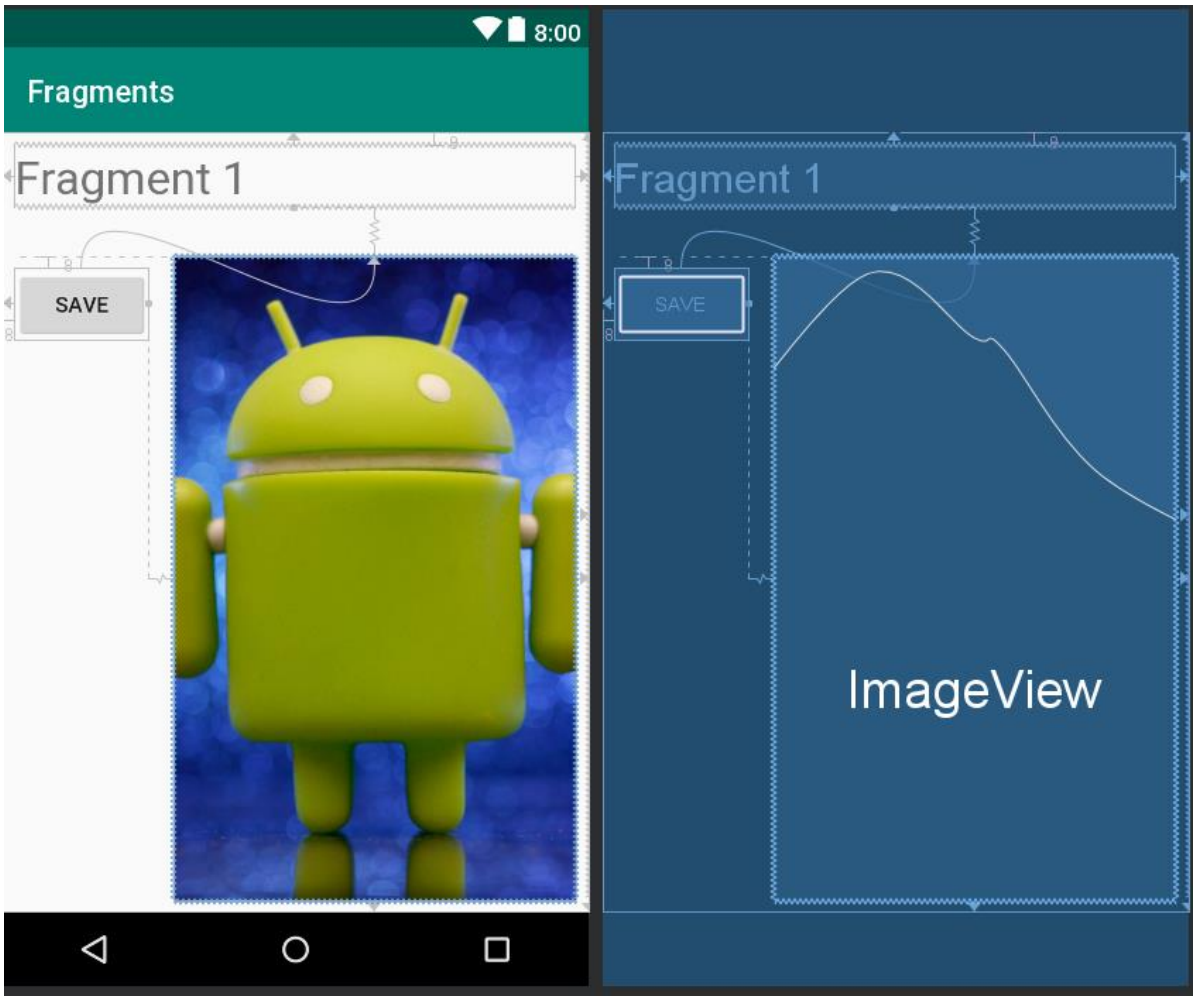
constraint
layout:



Now, build the layouts of your fragments: Fragment 1: If a click(tap) occurs on the first button (FRAGMENT 1), the expected output is:



and here is the design:



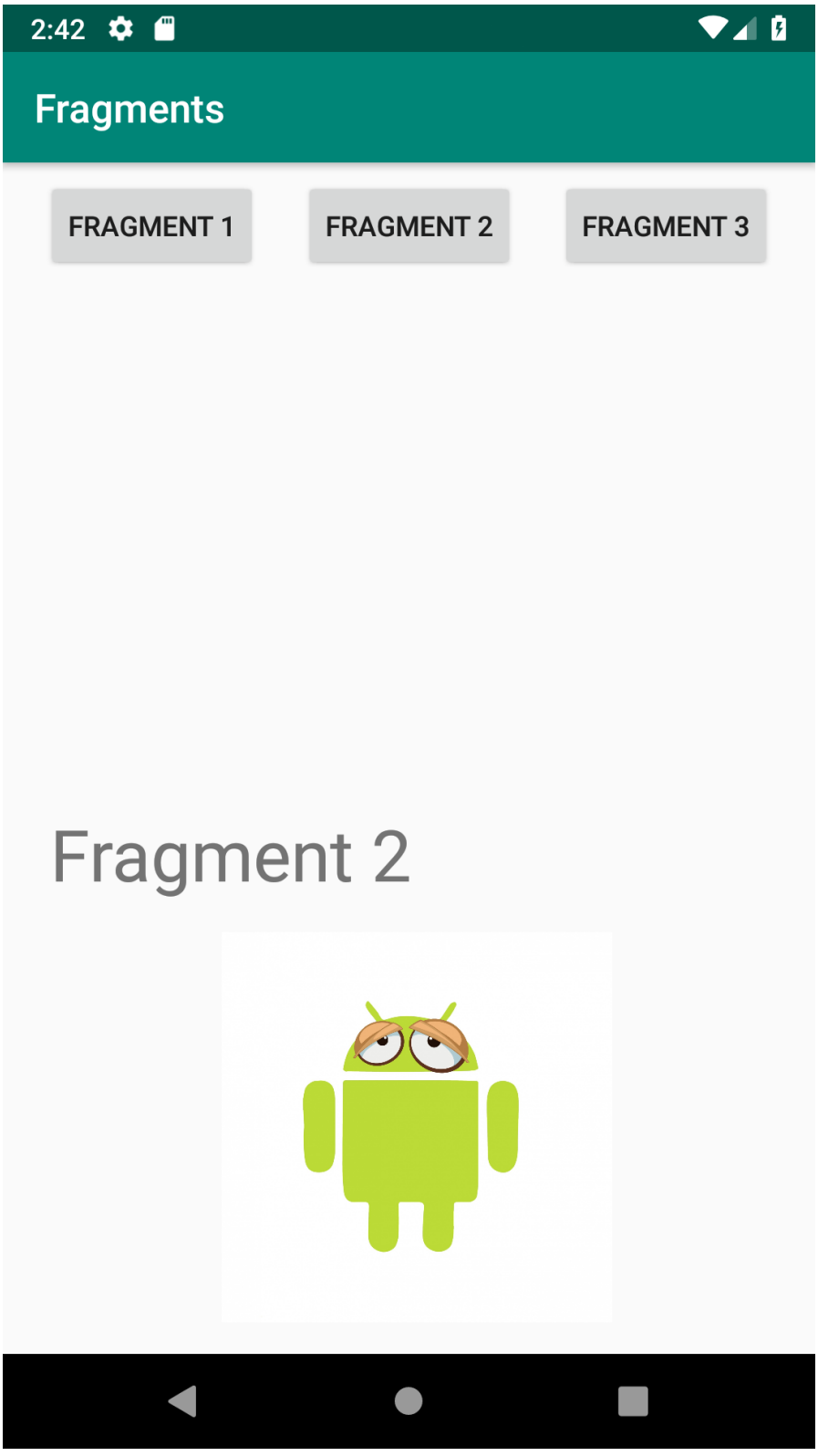
fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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:id="@+id/frameLayout"

android:layout_width="match_parent"

android:layout_height="match_parent"

tools:context=".Fragment1">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:id="@+id/textView"
```

```
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="Save"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/imageView"/>

    <TextView
        android:id="@+id/textView2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:text="Fragment 1"
        android:textSize="30sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="16dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:scaleType="centerCrop"
        android:src="@drawable/android1"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/button4"
        app:layout_constraintTop_toBottomOf="@+id/textView2"/>
```

</android.support.constraint.ConstraintLayout>

Fragment

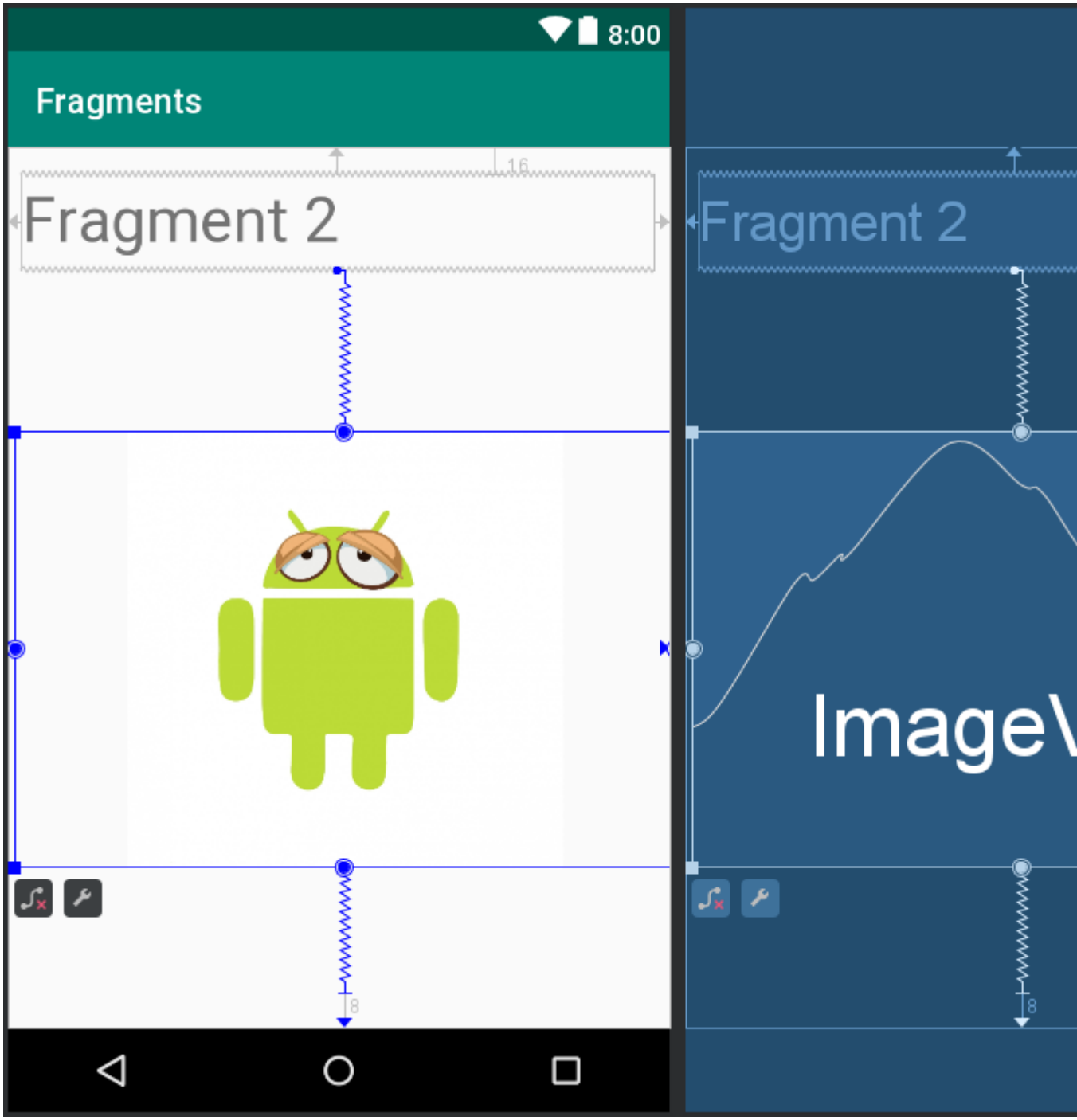


2: fragme

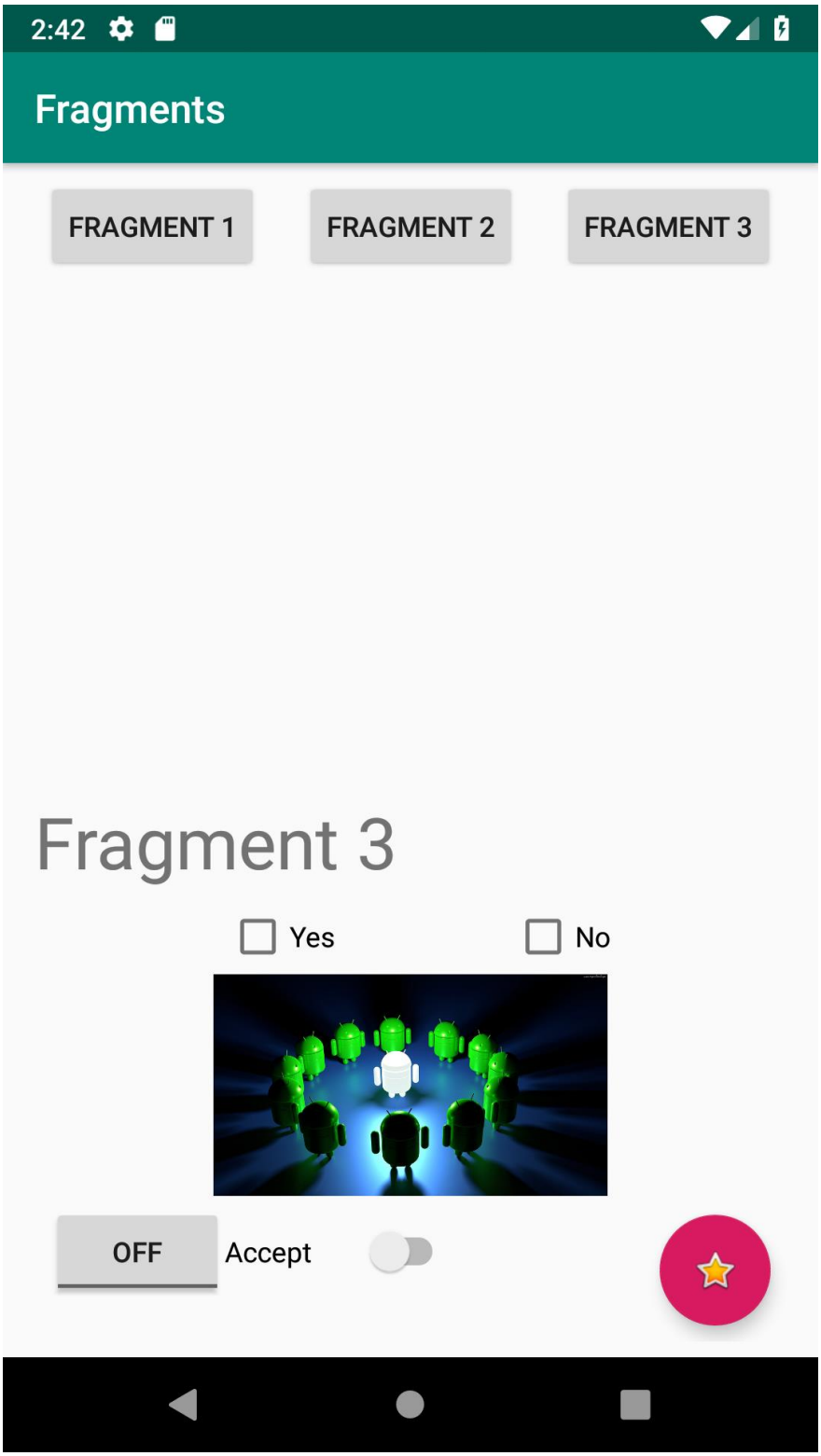
nt2.xml

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

```
<android.support.constraint.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:id="@+id/frameLayout2"

android:layout_width="match_parent"

android:layout_height="match_parent"

tools:context=".Fragment2">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:id="@+id/textView3"
        android:layout_width="0dp"
        android:layout_height="56dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="8dp"
        android:text="Fragment 2"
        android:textSize="36sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        android:src="@drawable/android2"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView3"/>

</android.support.constraint.ConstraintLayout>
```



Fragment



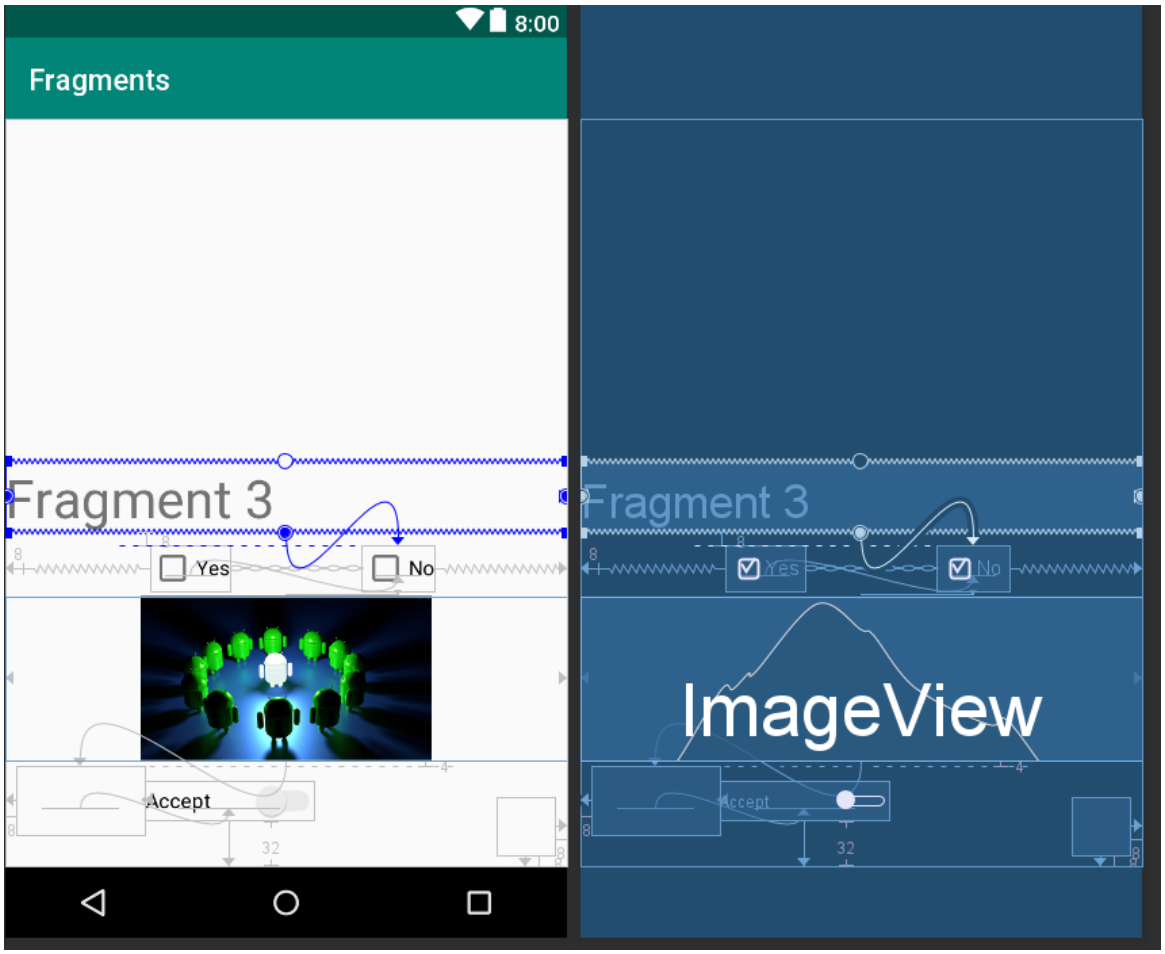
3: fragment3.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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:id="@+id/fragmentLayout3"

android:layout_width="match_parent"

android:layout_height="match_parent"

tools:context=".Fragment3">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:id="@+id/textView4"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:text="Fragment 3"
        android:textSize="36sp"
        app:layout_constraintBottom_toTopOf="@+id/checkBox2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"/>

    <ImageView
        android:id="@+id/imageView3"
        android:layout_width="wrap_content"
        android:layout_height="112dp"
        android:layout_marginBottom="4dp"
        android:src="@drawable/android3"
        app:layout_constraintBottom_toTopOf="@+id/toggleButton2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        />

    <CheckBox
        android:id="@+id/checkBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:text="Yes"
        app:layout_constraintBaseline_toBaselineOf="@+id/checkBox2"
        app:layout_constraintEnd_toStartOf="@+id/checkBox2"
```

```
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"/>

<CheckBox
    android:id="@+id/checkBox2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="3dp"
    android:text="No"
    app:layout_constraintBottom_toTopOf="@+id/imageView3"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toEndOf="@+id/checkBox"
    />

<ToggleButton
    android:id="@+id/toggleButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:text="ToggleButton"
    app:layout_constraintBaseline_toBaselineOf="@+id/switch1"
    app:layout_constraintStart_toStartOf="parent"/>

<Switch
    android:id="@+id/switch1"
    android:layout_width="116dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="32dp"
    android:text="Accept"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toEndOf="@+id/toggleButton2"
    />

<android.support.design.widget.FloatingActionButton
    android:id="@+id/floatingActionButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:clickable="true"
    android:src="@android:drawable/btn_star_big_on"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"/>
```

</android.support.constraint.ConstraintLayout>



MainActivity.java

```
package edu.monash.fit2081.fragments;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
    }

    public void handleBtn1(View view){

getSupportFragmentManager().beginTransaction().replace(R.id.frag1,new
Fragment1()).addToBackStack("f1").commit();

    }
    public void handleBtn2(View view){

getSupportFragmentManager().beginTransaction().replace(R.id.frag1,new
Fragment2()).addToBackStack("f2").commit();

    }
    public void handleBtn3(View view){

getSupportFragmentManager().beginTransaction().replace(R.id.frag1,new
Fragment3()).addToBackStack("f3").commit();

    }
}
```

The above code represents the business logic of the main activity. In addition to onCreate callback, it has three functions (handleBtn1, handleBtn2, and handleBtn3) that work as handlers for the three buttons Button 1, Buton 2, and Button 3 respectively. In each function, there is a statement that is responsible for replacing the current fragment with a new one.

```
getSupportFragmentManager().beginTransaction().replace(R.id.frag1,new
Fragment1()).addToBackStack("f1").commit();
```

Note the following:

- getSupportFragmentManager: returns the FragmentManager which is used to create transactions for adding, removing or replacing fragments.
- beginTransaction: starts a series of edit operations on the Fragments
- .replace: replaces the current fragment with a new fragment of type Fragment1 on the layout with id R.id.frag1
- addToBackStack: Adds this transaction to the back stack. This means that the transaction will be remembered after it is committed, and will reverse its operation when later popped off the stack.

References

[1]Â https://developer.android.com/guide/components/fragments [2]Â https://www.tutorialspoint.com/android/android_fragments.htm