FRAGMENTS: (REFER APP: Memey)

Up until now we have seen how we can add different widgets to our app. These widgets might include buttons, TextViews, EditTexts, progress bars, etc. However in Android Studio, it is also possible to make a complex widget out of these simple widgets. This complex widget is called a ‘Fragment’. A fragment essentially is a collection of pre-defined widgets which we can use as a whole in different activities. It is similar to how in Java or C++ a class or a structure is a combination of several different primitive datatypes and can ITSELF be treated as a datatype and thus can be instantiated. In a similar way a ‘Fragment’ is also a form of a ‘DERIVED WIDGET’ which we get by combining different primitive widgets in a specific manner.

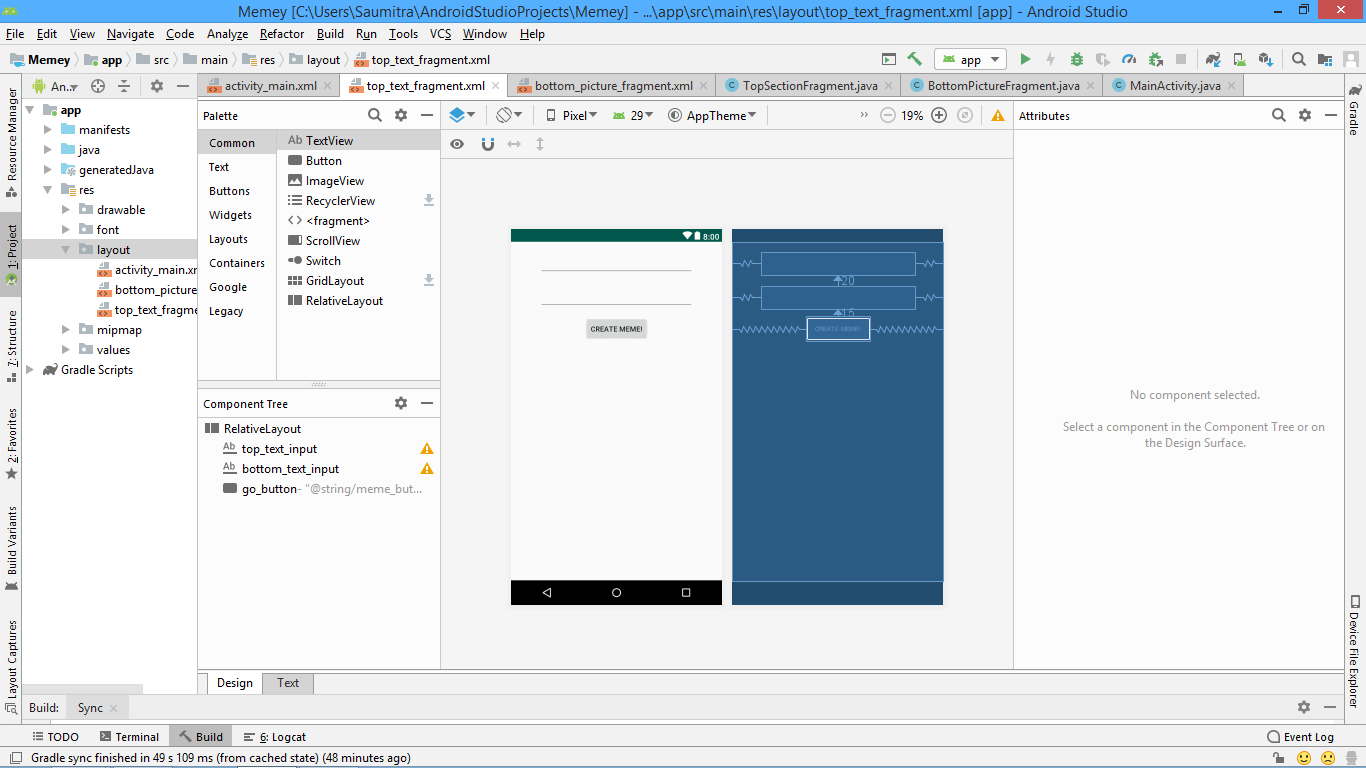
One of the chief advantages of a fragment is that it can be reused however times we may like. Thus if we have certain UI elements in our app that we want to maintain constant in each and every activity of our app, then rather than manually creating the UI for every activity we can simply create a fragment and then attach it to every activity of our app.

We here are creating an app which will have 2 fragments-one in which the user enters 2 texts (the top text and the bottom text) and in the second fragment the text entered by the user is displayed on a meme. In order to design a fragment we need to follow these 3 steps.

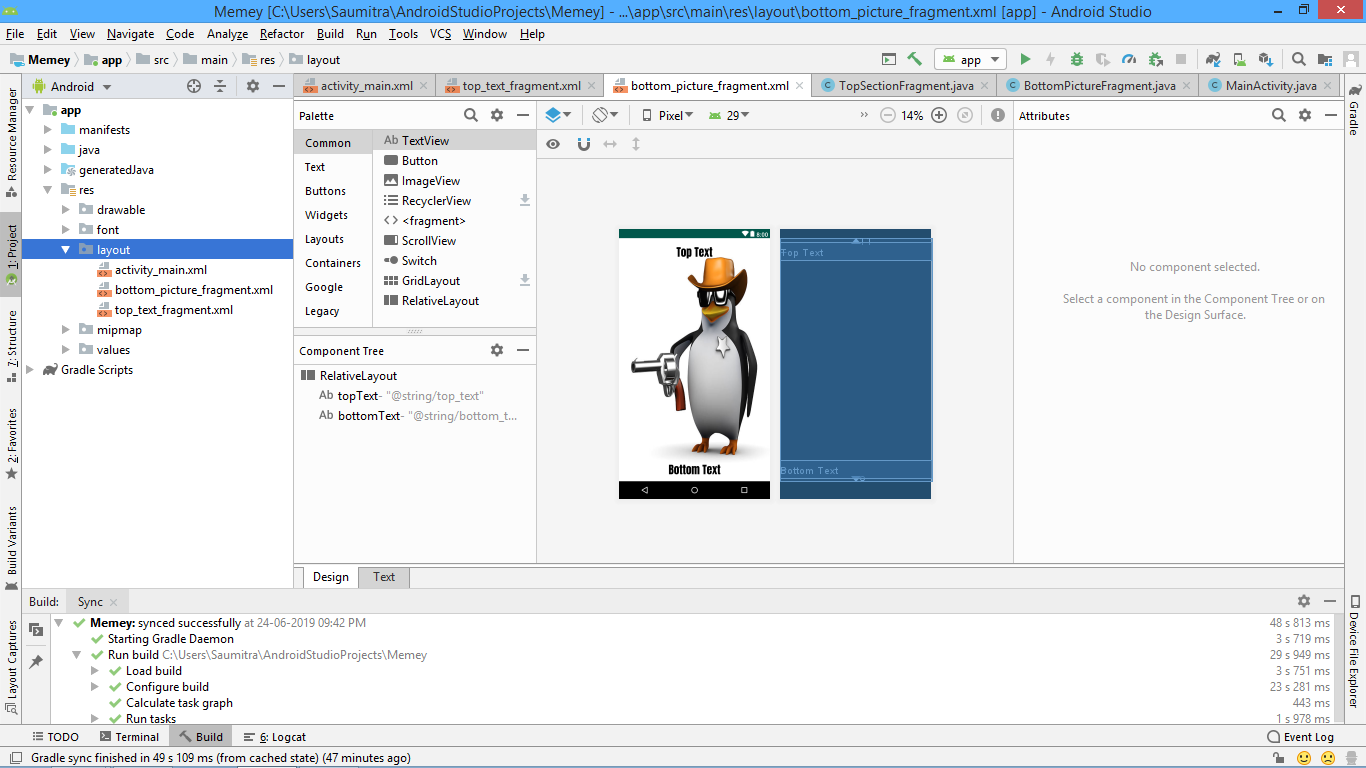
1. Create an xml file of the fragment and decide the UI of that particular fragment. We can create an xml file by just right clicking the layout folder (i.e. Memey/app/src/main/res/layout) and selecting new->layout resource file.
2. Create a java file that particular fragment in order to specify the functionality of the fragment (essentially deciding the brains of the fragment)
3. Attaching (or gluing together) the fragment with the MainActivity and then specifying the layout and the functionality of the main\_activity using the java and the xml file.

Thus we first create the xml files of the 2 fragments that we want to add to our app. The first fragment will represent that part of the page where the user enters the ‘top-text’ and the ‘bottom-text’ for the meme. On the other hand, the second fragment will represent the actual meme where the text entered by the user is displayed at the respective positions. The design of the fragments are as shown:

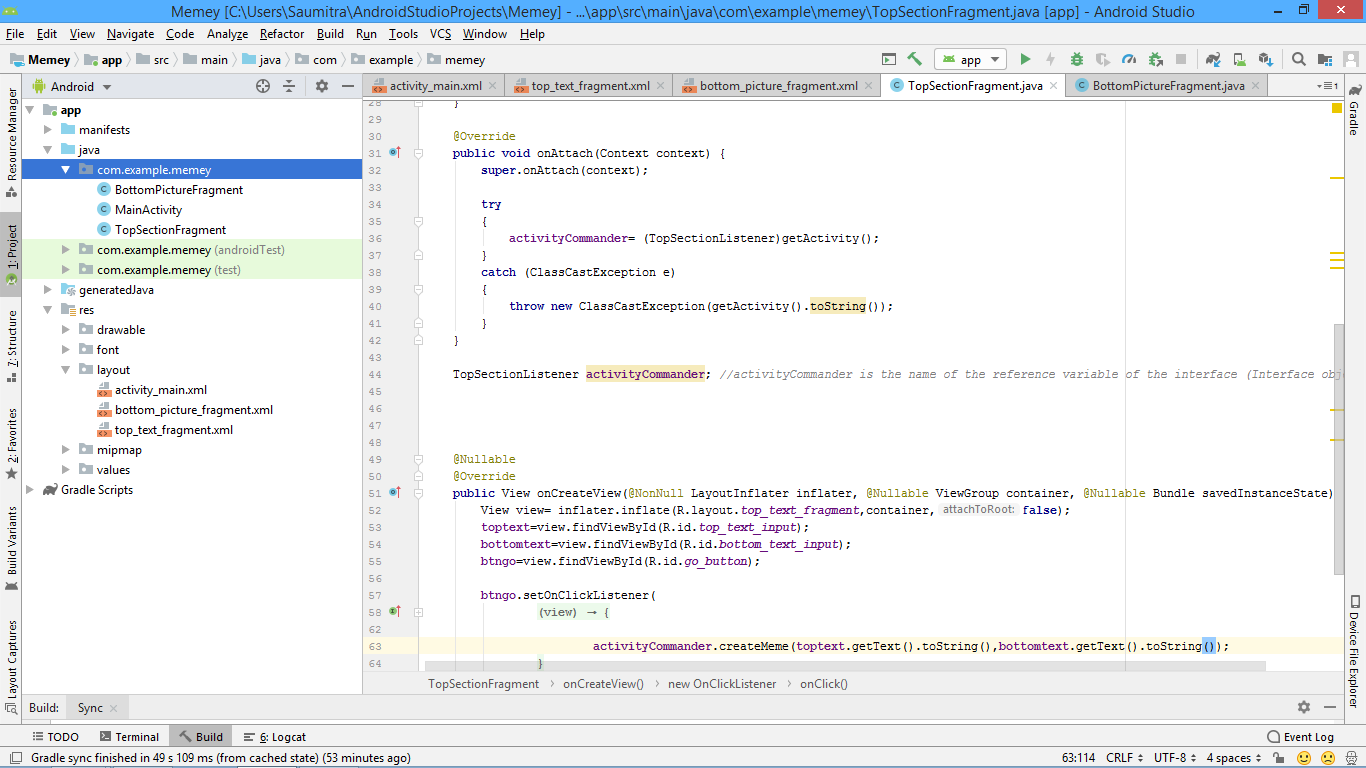
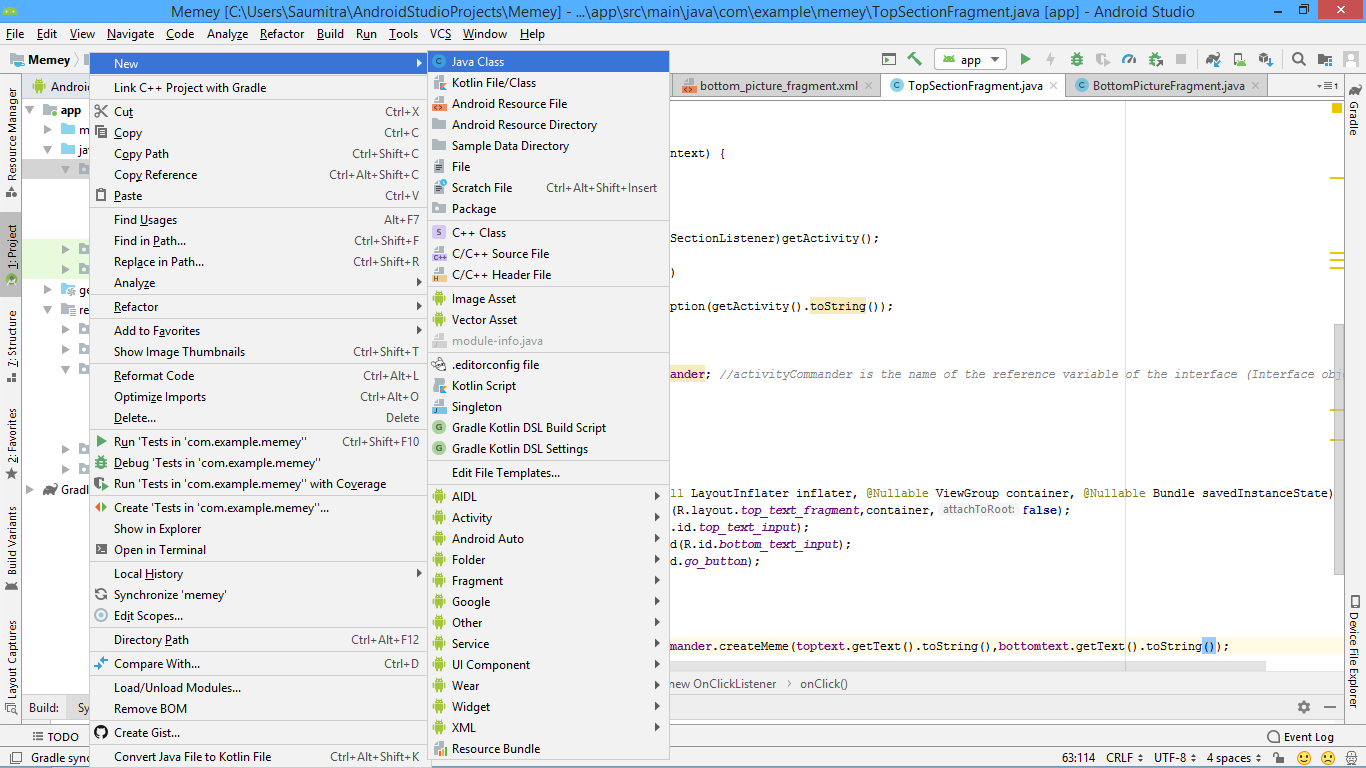
TOP FRAGMENT:



BOTTOM FRAGMENT:



Then, we create the two java files which act as the ‘brains’ of the two fragments. These java files can be created by right clicking the com.example.memey directory inside the java directory. And selecting new->java class.

TOP SECTION FRAGMENT JAVA FILE:

<https://www.youtube.com/watch?v=yOBQHf5nM2I&list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrnxl&index=25>

<https://www.youtube.com/watch?v=MHHXxWbSaho&list=PL6gx4Cwl9DGBsvRxJJOzG4r4k_zLKrnxl&index=28>

This file contains all the functionality of the top fragment where the user enters the text. In order to denote that UI element as a fragment, we need to import the following packages:

**import** android.os.Bundle;//Standard import for all java files  
**import** android.view.LayoutInflater;// To identify which xml file we are going to use as a fragment  
**import** android.view.View;   
**import** android.view.ViewGroup;

These are essentially used to identify UI elements of the XML file and describe their functionality

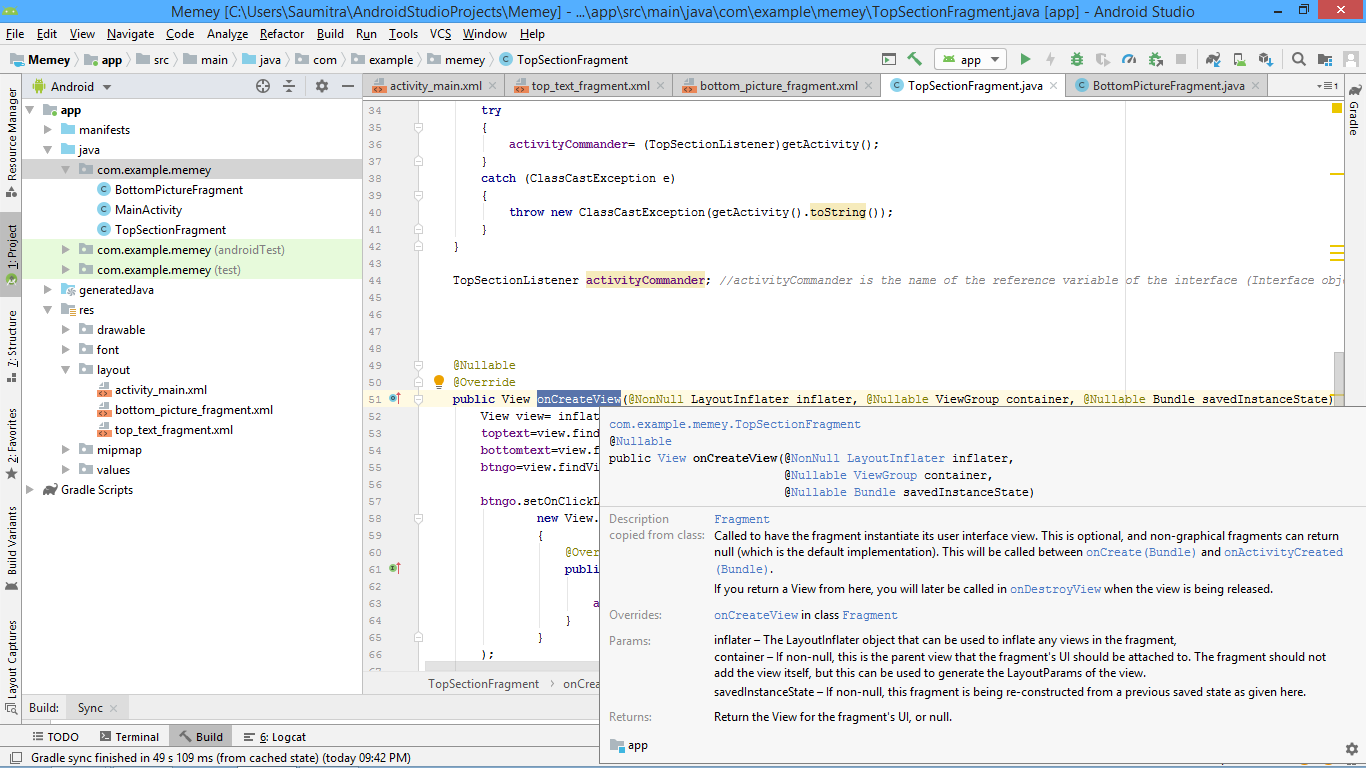
**import** android.app.Activity;//Used to attach the fragment to MainActivity  
**import** android.widget.Button;//Used for the button in our fragment  
**import** android.widget.EditText;//Used for the EditText in our fragment  
  
**import** androidx.annotation.NonNull;   
**import** androidx.annotation.Nullable;   
**import** androidx.fragment.app.Fragment;

These get auto imported as we type

Now for the TopSectionFragment class, we need the class to extend the Fragment class. Thus, it can override (and or directly use) all the methods inside the fragment class. We then create reference variables for our EditTexts as well as out Button.

**public class** TopSectionFragment **extends** Fragment {  
  
 EditText **toptext**;  
 EditText **bottomtext**;  
 Button **btngo**;

Now we override the onCreateView method of the Fragment class. This can be done by pressing alt+insert and selecting Override Methods.



This method essentially creates a new object of the View class which specifies which xml file do we want to link to this fragment. This process of linking the xml file is done with the help of the ‘inflate’ method of the LayoutInflater class. It essentially converts our xml file into java view objects.

View view= inflater.inflate(R.layout.***top\_text\_fragment***,container,**false**);  
**toptext**=view.findViewById(R.id.***top\_text\_input***);  
**bottomtext**=view.findViewById(R.id.***bottom\_text\_input***);  
**btngo**=view.findViewById(R.id.***go\_button***);

Find more here:

<https://stackoverflow.com/questions/19862456/can-someone-explain-inflate-method-deeper-understanding-of-android-views>

<https://www.youtube.com/watch?v=fxVeFwtIpVc>

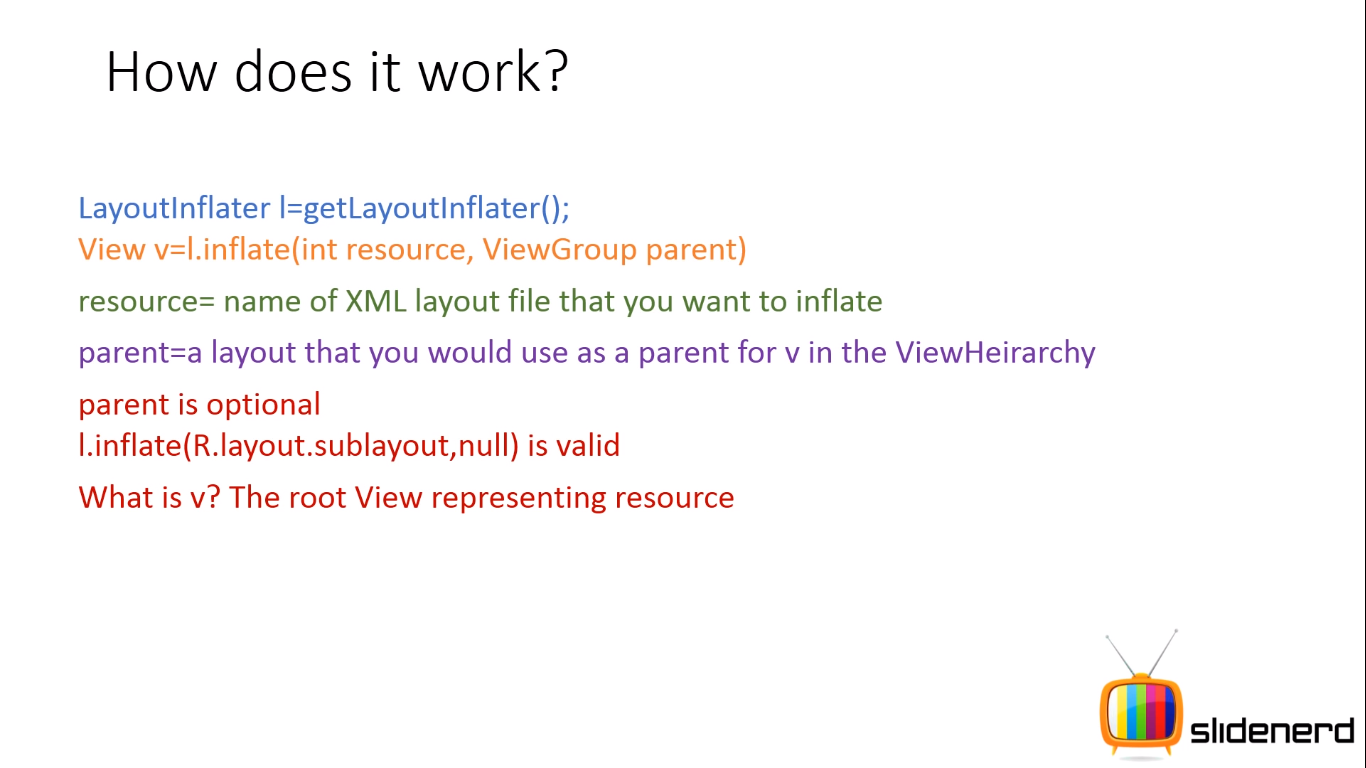
<https://www.youtube.com/watch?v=1Y0LlmTCOkM>

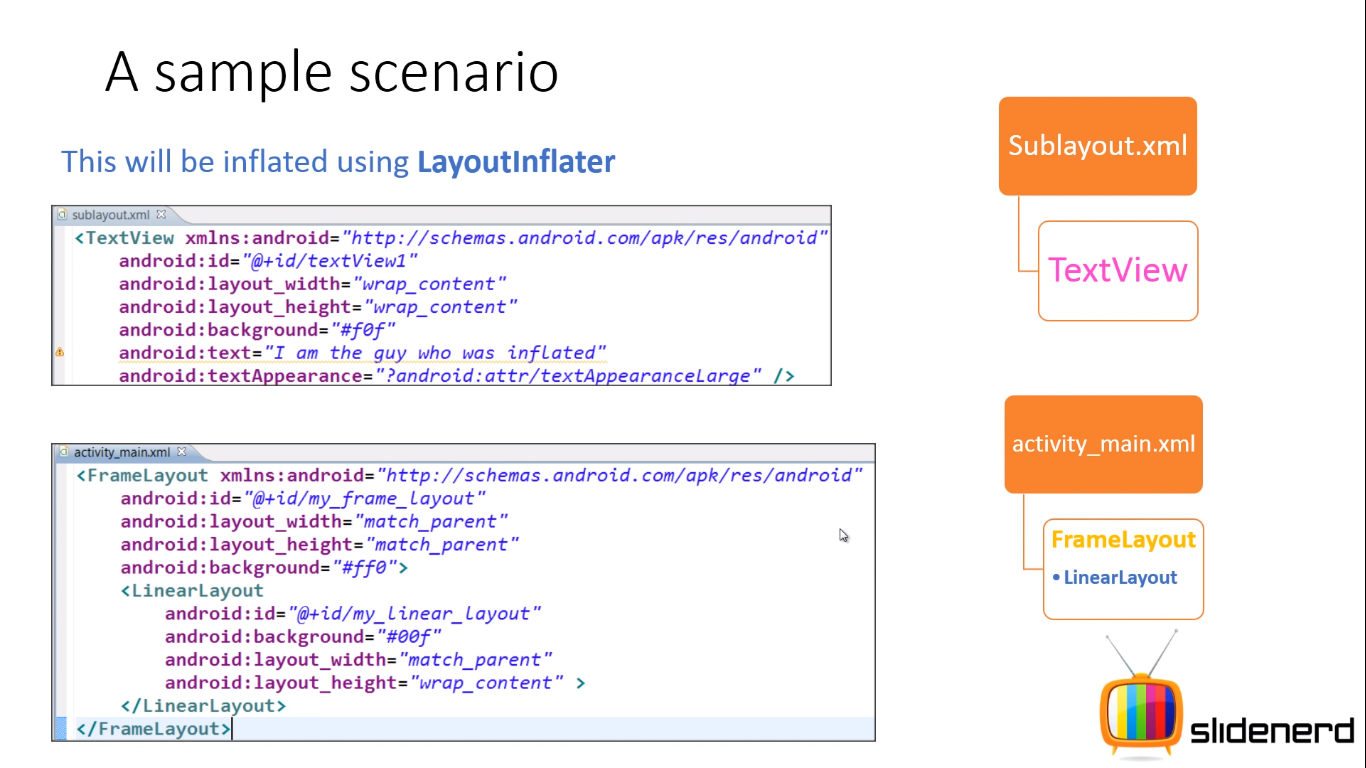
As you might already know by now that every visible component in Android is a View. That includes Button, Spinner, TextView, EditText and the likes. You are also right about the way we access the Views that are defined in an xml file in our program, by inflating it and then finding the view by using its id. The usual way to do this is by using setContentView() method.

But there is this pretty useful class called LayoutInflater that can be used to "inflate" a layout. Now consider the scenario where you have a ListView in your xml file that you inflated using setContentView () method. In this ListView you want the items to contain an ImageView and a TextView. The default list-item just have a TextView. So you decided to write up a custom adapter in which you'll be using a new listitems.xml in which you have a TextView and ImageView defined. Now you can't use setContentView() in this case as it will inflate this layout on the whole activity which obviously you don't want. So you use LayoutInflater in this case to help you out. You temporarily inflate a layout using inflate() method. The first argument takes the layout file which is needed to be inflated. The second argument is the root of this newly inflated layout. In our case it can be set to the ListView in which the layout is going to be actually inflated.

So when you use this inflate() method, a View is returned containing the views that are inside the inflated xml. Now with this returned view instance you can call findViewById() to get the contained views to set text to the TextView and image source to the ImageView.

More often than not, you'll end up using LayoutInflater as its use cases are wider than the scope of this discussion.





We want the textview in the sublayout somewhere here

We also set an onClickListener for our button using this:

**btngo**.setOnClickListener(  
 **new** View.OnClickListener()  
 {  
 @Override  
 **public void** onClick(View view) {  
  
 **activityCommander**.createMeme(**toptext**.getText().toString(),**bottomtext**.getText().toString());  
 }  
 }  
);

We’ll discuss the activityCommander thing in a bit… Just hold On.

We then return the view which was created by the LayoutInflater.

**return** view;

We now create a new interface which will help us in ensuring that whatever class implements this interface will have to coompulsorily have to override the method in it. This interface will contain a method called createMeme. This interface is implemented by the MainActivity class and then this class will have to override the createMeme method.

**public interface** TopSectionListener *//This is done to ensure that the method inside this interface is compulsorily implemented by whatever class implements it*{  
 **void** createMeme(String top, String bottom); *//This is the method which will be compulsorily overridden by the class that implements it*}

In the onclicklistener of the button, we are simply calling the createMeme method by passing the text which was entered by the user. This is done using the getText method.

We now create an instance (or a reference variable) for the interface TopSectionListener. Let us call this variable as: activityCommander.

TopSectionListener **activityCommander**; *//activityCommander is the name of the reference variable of the interface (Interface object)*

We now override a method called onAttach().Basically, onAttach() passes a reference of the main activity to the fragment. And since main activity implements the TopSectionListener interface, we're able to type cast it as a TopSectionListener and use it to call createMeme. This is done by the following try-catch block:

**public void** onAttach(Context context) {  
 **super**.onAttach(context);  
  
 **try** {  
 **activityCommander**= (TopSectionListener)getActivity();  
 }  
 **catch** (ClassCastException e)  
 {  
 **throw new** ClassCastException(getActivity().toString());  
 }  
}

Actually there is no need for try block at all, you could check 'if(context instanceof TopSectionListener)' and only after that cast context to TopSectionListener

@Override

public void onAttach(Context context) {

super.onAttach(context);

if(context instanceof TopSectionListener) {

activityCommander = (TopSectionListener)context;

}

}

BOTTOM PICTURE FRAGMENT JAVA FILE:

We do very similar things with the BottomPictureFragment.java file. The source code is as follows:

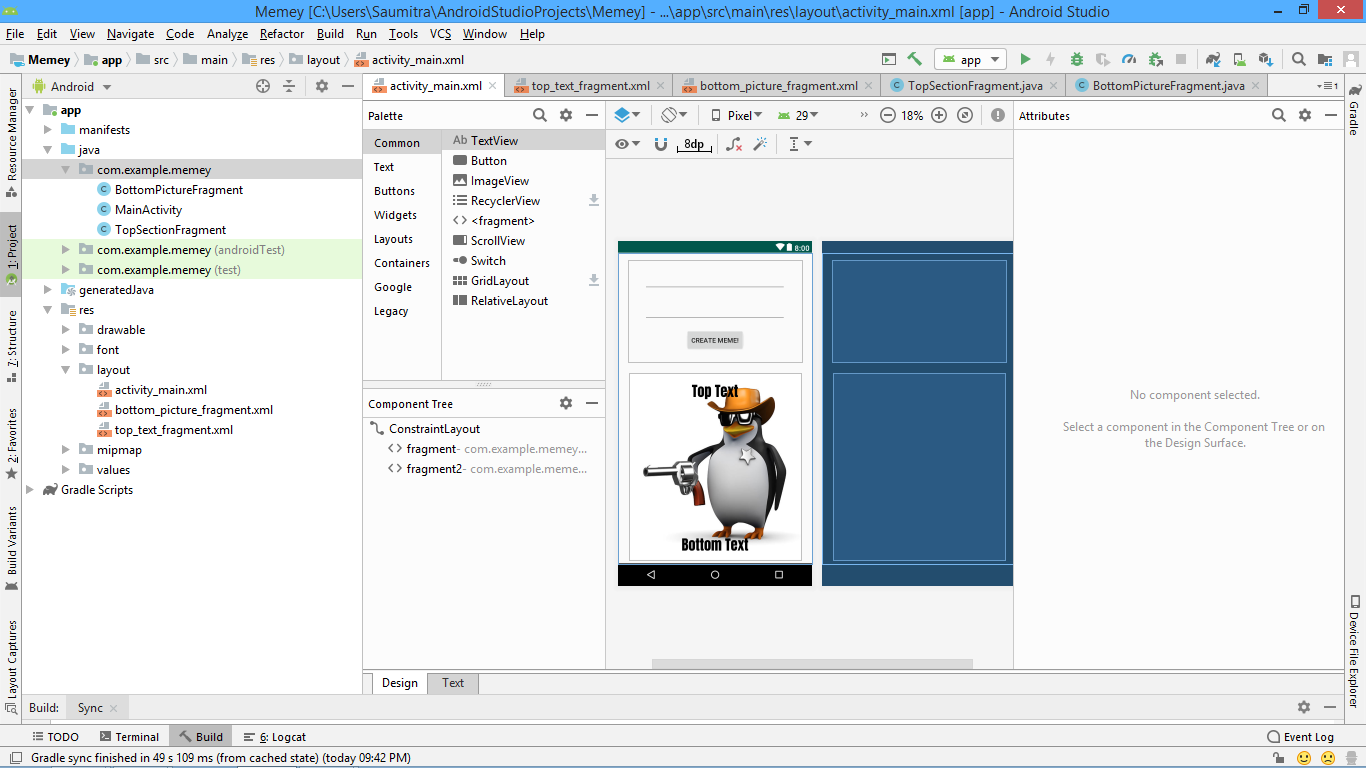
**package** com.example.memey;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.view.LayoutInflater;  
  
**import** androidx.annotation.NonNull;  
**import** androidx.annotation.Nullable;  
**import** androidx.fragment.app.Fragment;  
  
**import** android.widget.TextView;  
  
**public class** BottomPictureFragment **extends** Fragment {  
  
 **private static** TextView *txttp*;  
 **private static** TextView *txtbtm*;  
 @Nullable  
 @Override  
 **public** View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {  
 View view=inflater.inflate(R.layout.***bottom\_picture\_fragment***,container,**false**);  
  
 *txttp*=view.findViewById(R.id.***topText***);  
 *txtbtm*=view.findViewById(R.id.***bottomText***);  
  
 **return** view;  
 }

We now create a method called setMemetext which accepts 2 string values. This method will be called inside the MainActivity class after creating an object of the BottomPictureFragment class.

**void** setMemeText(String top,String bottom)  
 {  
 *txttp*.setText(top);  
 *txtbtm*.setText(bottom);  
  
 }  
}

CREATING THE MAINACTIVITY.XML FILE:

We now arrange the two fragments together in our final view and see the final layout:



CREATING THE MAINACTIVITY.JAVA FILE:

Here, our mainActivity class will implement the TopSectionListener interface and thus has to compulsorily override the createMeme method. This is where we provide a definition for our method. Inside this definition, we create an instance of the bottom picture fragment class and then call the method setText using this instance.

@Override  
**public void** createMeme(String top, String bottom) {  
  
BottomPictureFragment bottomFragment = (BottomPictureFragment) getSupportFragmentManager().findFragmentById(R.id.***fragment2***);  
 bottomFragment.setMemeText(top,bottom);  
}

Thus:

1. The INTERFACE IS CREATED IN THE TOPSECTIONFRAGMENT CLASS. THE METHOD createMeme IS DECLARED INSIDE IT.
2. THIS METHOD IS CALLED USING AN OBJECT CALLED activityCommander INSIDE THE TOPSECTIONFRAGMENT CLASS
3. THE ACTUAL DEFINITION OF THIS METHOD IS EXPLAINED IN THE MAINACTIVITY CLASS WHICH ALSO IMPLEMENTS THIS INTERFACE. THIS IS BECAUSE THE DEFINITION ALSO INVOLVES A METHOD FROM THE BOTTOMPICTUREFRAGMENT CLASS.

FINAL OUTPUT:

