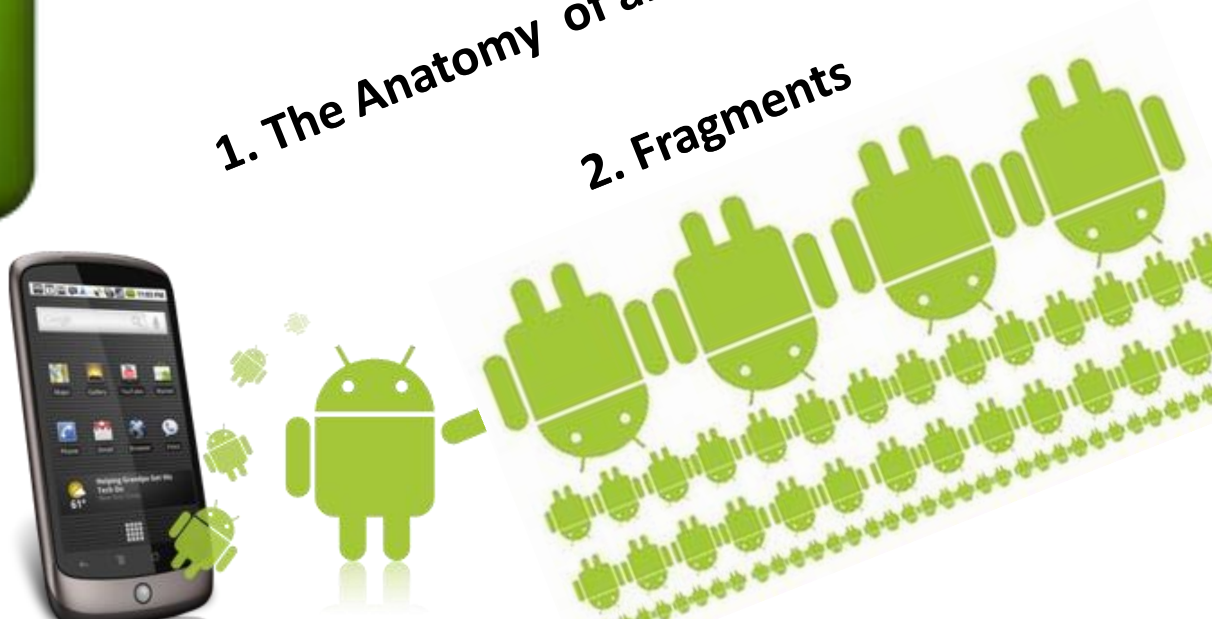
West University of Timisoara, Computer Science Department
PMD, **IE II, IR II, spring 2021**
Course no. 3 *phd. lecturer Liviu Octavian Mafteiu-Scai*

1. The Anatomy of an Android Application

2. Fragments



The Anatomy of an Android Application

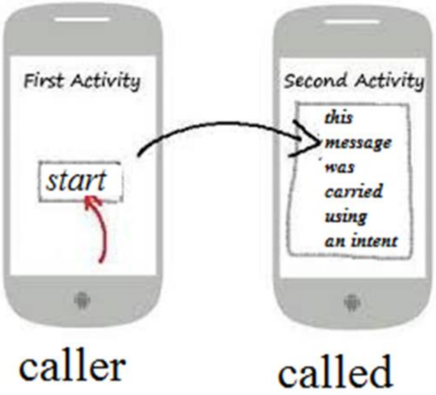
Content:

- Android Activities
- Android Intents
- Broadcast Intents
- Broadcast Receivers
- Android Services
- The Application Manifest
- Application Resources
- Application Context

The Anatomy of an Android Application : ***Android Activities***

Apps = one ar more *Activities* linked together, to do one or more tasks

Remember: Activity is not a task



Definition: An activity is a single, standalone module of application functionality that usually has a single user interface screen (a view) and its corresponding functionality (layout).

Example: In a game, we have an activity screen (main activity) that displays game's scene, score, user's account, themes, and so on. A second activity could be a screen where the user types their personal data.

Each activity is implemented as a single class that extends the Android Activity base class.

The Anatomy of an Android Application : *Android Activities*

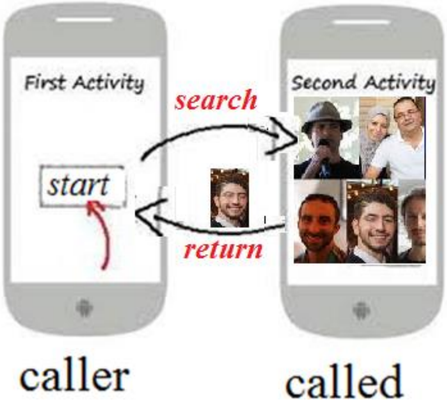
In fact, most mobile apps consist of multiple screens.
! For each screen we have an activity !

Example: a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages. Each of these screens would be implemented as an activity.

Moving to another screen means starting a new activity.

In some cases an Activity may return a value/object to the previous activity.

Example: an activity that lets the user pick a photo in called activity would return the chosen photo to the caller.



The Anatomy of an Android Application : ***Android Activities***

When a new screen opens, the current screen (at this moment will be previous) is paused and put onto a history stack.

->

The user can navigate backward through previously opened screens in the history.

->

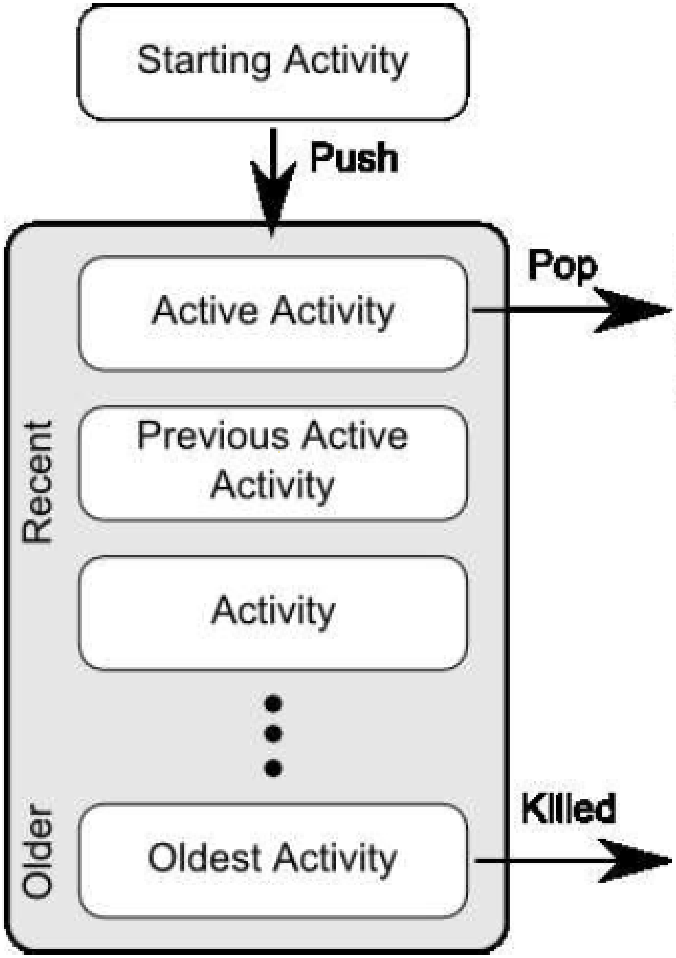
Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain.

->

Android retains history stacks for each application launched from the home screen.

The Anatomy of an Android Application : ***Android Activities***

Activity stack



Current
(will be previous when a new activity comes)

Terminated
(to free memory)

An activity cannot directly call methods or access data of another activity.

This is achieved using *Intents* and *Content Providers*.

The Anatomy of an Android Application : *Android Intents*

Intents are the mechanism by which one activity (caller) is able to launch another activity (called)

An intent implement the flow through the activities.

Intents consist of a description of the operation to be performed (action and data)

Android uses a special class called *Intent* to move from screen to screen (caller activity -> called activity)

The Anatomy of an Android Application : ***Android Intents***

Intents describe what an application wants done.

The two most important parts of the intent data structure are the action and the data transmitted.

Typical values for action are MAIN, VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI).

What is URI?
URI is a string of characters that identifies a particular resource.
URI does nothing!

Example: to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI:

```
new Intent(android.content.Intent.VIEW_ACTION, ContentURI.create("http://uvt.ro"));
```

The Anatomy of an Android Application : ***Android Intents***

Explicit Intents: they request the launch of a specific activity by specifying the activity by class name.

Implicit Intents: by starting the type of action to be performed. Android runtime will select the activity to launch that most closely matches the criteria specified by the Intent using a process referred to as *Intent Resolution*.

Examples:

Explicit intent: pass the information from one activity to another (see our multiple activities example where string “123” were passed, Course no. 2)

Implicit intent: send an intent requesting that the content of a particular web page be loaded and displayed to the user (see second example from same course)

The Anatomy of an Android Application : ***Android Intents***

Navigating from screen to screen is accomplished by resolving intents.

To navigate forward, an activity calls *startActivity(myIntent)*.

The system then looks at the intent filters for all installed applications and picks the activity whose intent filters best matches *myIntent*.

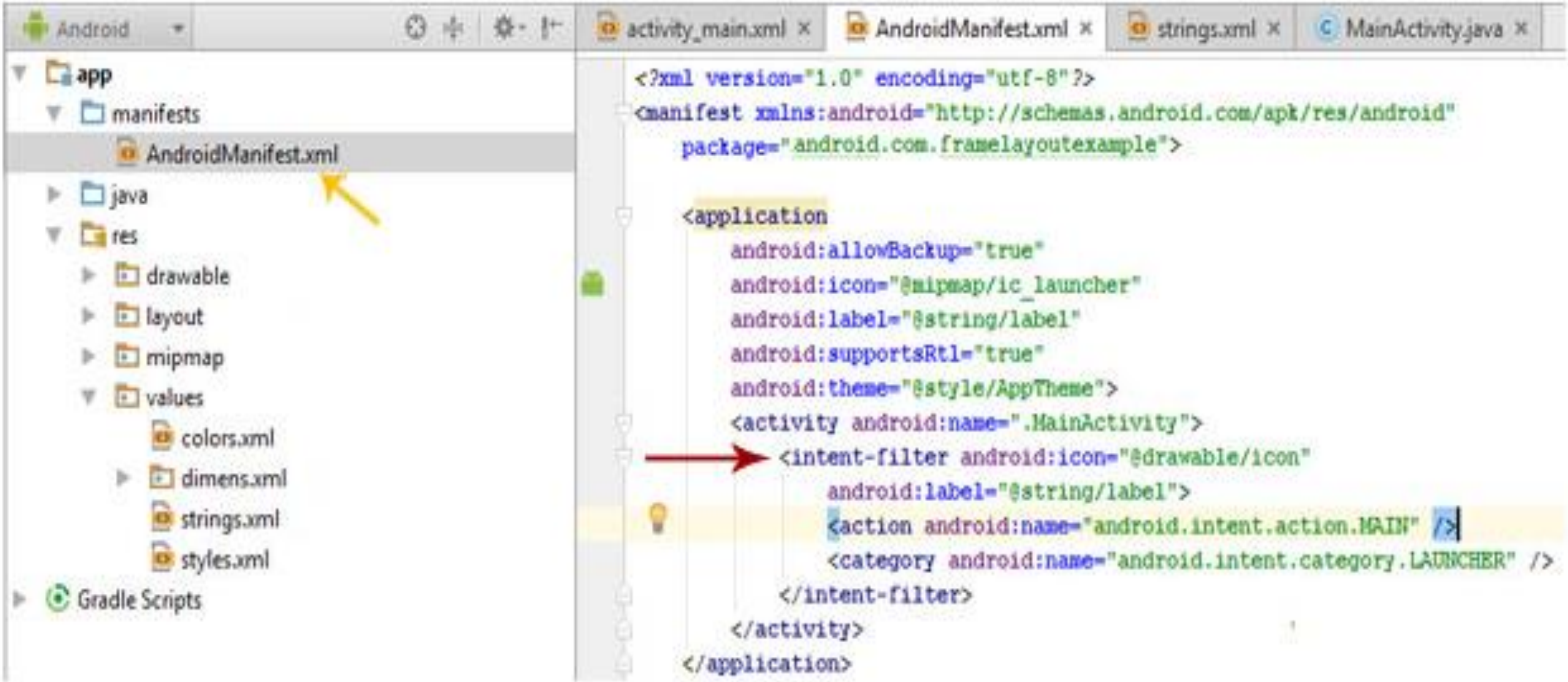
The new activity is informed of the intent, which causes it to be launched.

The Anatomy of an Android Application : ***Android Intents***

The process of resolving intents happens at runtime when *startActivity* is called.

An **intent filter** is a description of what intents an activity is capable of handling.

Activities publish their IntentFilters in *AndroidManifest.xml*



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="android.com.framelayoutexample">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/label"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter android:icon="@drawable/icon"
                android:label="@string/label">
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

The Anatomy of an Android Application : ***Android Intents***

Intent Receiver

You can use an Intent Receiver when you want to code an app that has a reaction to an external event (*when the phone rings, or when the data network is available, or when it's midnight)*.

To create an alert when your phone rings:

-import android.telephony.TelephonyManager;

-**create** : extend class **public class PhoneReceiver extends** BroadcastReceiver

-**declare a method**: **public void** onReceive(Context context, Intent intent)

-use intent to take the caller phone number: givenstring= intent.getExtras()

-     Put the given number into a string to be displayed)or other action):

String phoneNumber = givenstring.getString(TelephonyManager.EXTRA_INCOMING_NUMBER);

In most cases, Intent receivers do not display a UI. Sometimes they may display *Notifications* to alert the user if something interesting has happened.

Intent receivers are also registered in *AndroidManifest.xml*

The Anatomy of an Android Application : ***Android Intents***

Broadcast Intents

Broadcast Intent: is sent out to all applications that have registered an “interested” *Broadcast Receiver*.

Example: Broadcast Intents can be used to indicate changes in device status such as the completion of system start up, connection of an external power source to the device or the screen being turned off.

A Broadcast Intent can be:

- *normal* (asynchronous): it is sent to all interested Broadcast Receivers at the same time
- *ordered:* it is sent to one Broadcast receiver at a time where it can be processed.

The Anatomy of an Android Application : ***Android Intents***

Broadcast Intents - Example:

```
Intent broadcastintent = new Intent();
broadcastintent.setAction(“bci.example");
broadcastintent.putExtra(“SentData", 123);
sendBroadcast(broadcastintent);
```

The code creates and sends a broadcast intent including a *unique action string(SentData)* and *data(123)*.
The action string(*SentData*), which identifies the broadcast event, must be unique

In Manifest.xml file must be the <action> tag inside <activity> tag to set an action:
<action android:name=“bci.example" > </action>

The Anatomy of an Android Application : ***Android Intents***

Broadcast Receivers

Broadcast Receivers are the mechanism by which applications are able to respond to Broadcast Intents.
An application listens for specific broadcast intents by registering a broadcast receiver.

A **Broadcast Receiver must be registered by an application** and configured with an *Intent Filter* to indicate the types of broadcast in which it is interested (in Manifest file)

Note that **a broadcast receiver does not need to be running all the time**. Only when an event that a matching intent is detected, the Android runtime system will automatically start up the broadcast receiver before calling the *onReceive()* method.

Broadcast Receivers operate in the background and do not have a user interface.

The Anatomy of an Android Application : *Android Intents*

Broadcast Receivers – Example

The Broadcast Receiver subclass:

```
import android.content.BroadcastReceiver;
….
public class BCReceiver extends BroadcastReceiver {
        public BCReceiver() {
       }
              public void onReceive(Context context, Intent intent) {

              // here must be the code to be performed when  the broadcast is detected
       }
}
```

In Manifest file, a <receiver> entry must be added for the receiver:

```
<receiver android:name=“BCReceiver" >
```

The Anatomy of an Android Application : *Android Intents*

Android Services

Android Services are processes that run in the background and do not have a user interface.

They can be started and managed from Activities, Broadcast Receivers or other Services.

Android Services are ideal for situations where an application needs to continue performing tasks but does not necessarily need a user interface to be visible to the user.

Although Services lack a user interface, they can still notify the user of events using notifications and *toasts* (small notification messages that appear on the screen without interrupting the currently visible activity).

The Anatomy of an Android Application : *Android Intents*

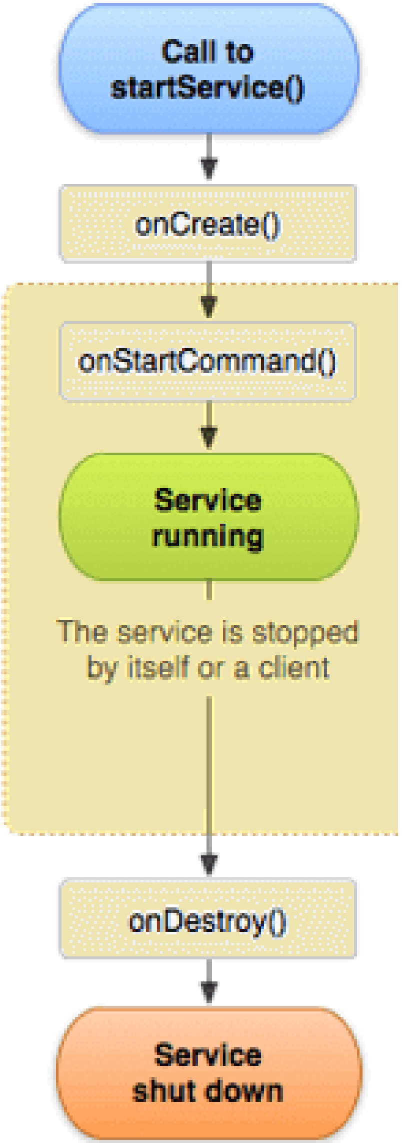
Android Services

<u>Example</u>:
A media player playing songs from a play list. In this media player application, there would probably be one or more activities that allow the user to choose songs and start playing them (UI exists). However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen (another app). The system will then keep the music playback service running until it has finished.

When connected to a service, you can communicate with it through an interface exposed by the service. *In case of music service example, this might allow you to pause, rewind, choose songs, set volume.. etc.*

StartService() and *stopService()* techniques are needed to start and stop the service.
public class SampleService extends Service {…..

!!! The service can be stoped by itself or a user! !!



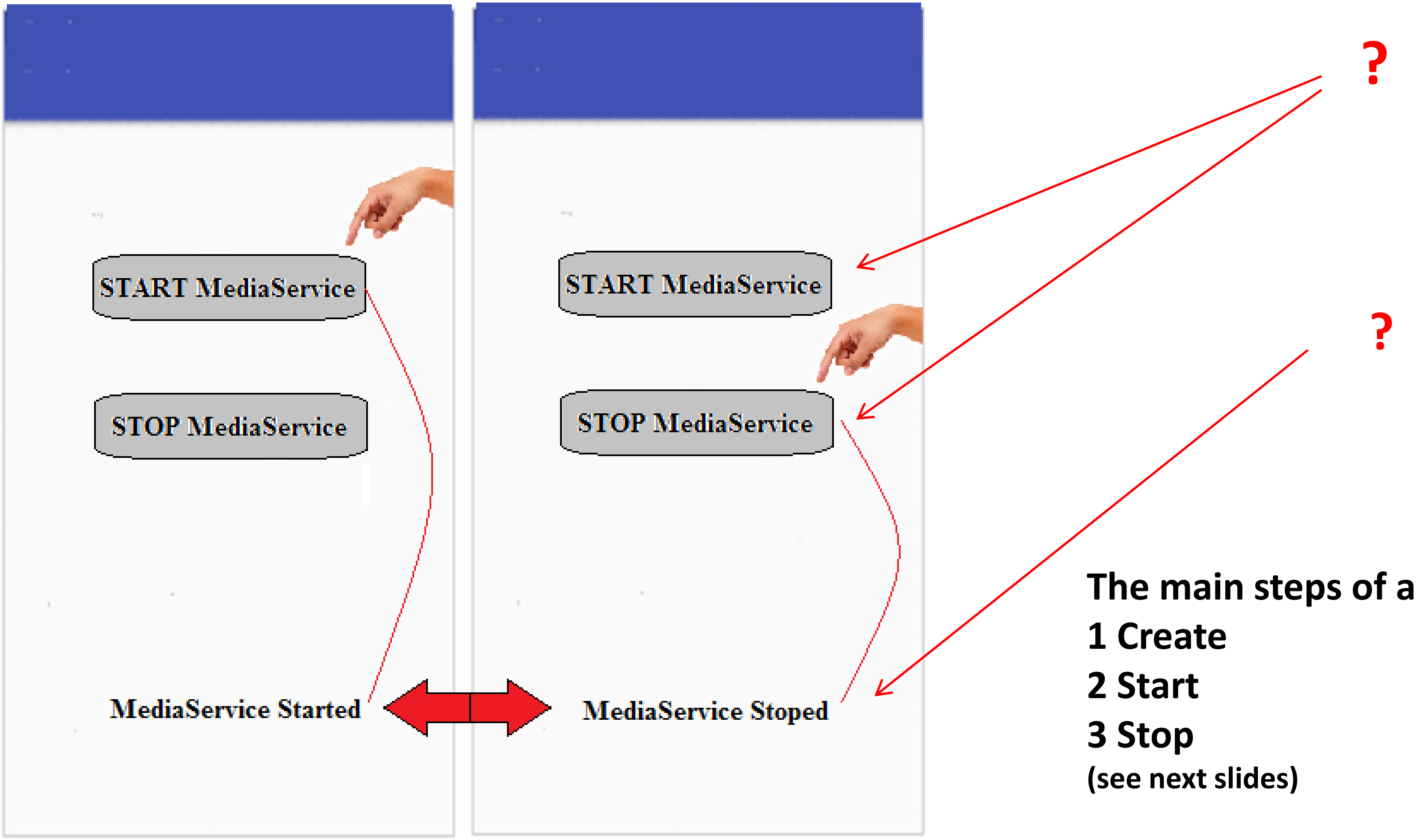
The Anatomy of an Android Application : *Android Intents*

EXAMPLE Android Services – a media player service implementation

Start MediaService => the default ringtone will start playing

Stop MediaService => will stop the service

It will continue playing between two ClickOn() (START ->STOP) (until we stop the service)



The main steps of a service:
1 Create
2 Start
3 Stop
(see next slides)

The Anatomy of an Android Application : **Android Intents**

Android Services –a media player service implementation

MainActivity.java

```
package com. mafteiuscai.liviu;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }

   public void startService(View view) {
      startService(new Intent(this, MediaService.class));
   }

   public void stopService(View view) {
      stopService(new Intent(this, MediaService.class));
   }
}
```

1 create

Set the activity content from a layout resource

2 start **? What type activity is ?**

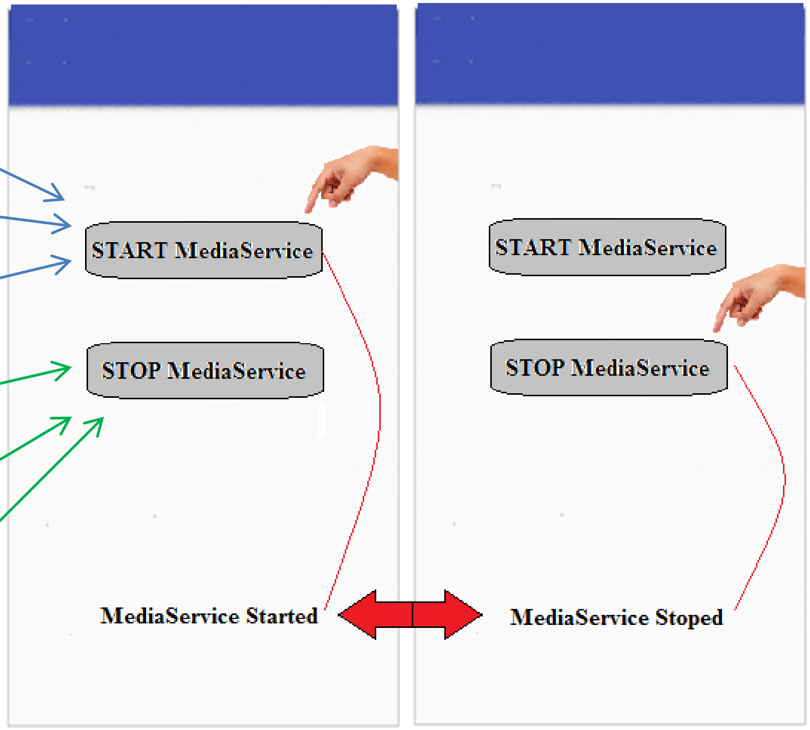
The Intent constructor takes two arguments for an **explicit** intent: an application Context and the specific component that will receive that intent. **? Who is *this*?**

3 stop

The Anatomy of an Android Application : *Android Intents*

Android Services –a media player service implementation-Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="match_parent"
  android:layout_height="match_parent">
  <Button
    android:id="@+id/btnStart"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="startService"
    android:layout_marginLeft="130dp"
    android:layout_marginTop="150dp"
    android:text="START MediaService"/>
  <Button
    android:id="@+id/btnstop"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="stopService"
    android:layout_marginLeft="130dp"
    android:layout_marginTop="20dp"

    android:text="STOP MediaService"/>
</LinearLayout>
```



MediaService Started

MediaService Stoped

? LinearLayout ?

The Anatomy of an Android Application : ***Android Intents***

Android Services –a media player service implementation

LinearLayout: a view group that aligns all children in a single direction, vertically or horizontally. To specify the layout direction use *android:orientation attribute.*

AbsoluteLayout is less flexible and harder to maintain than linear layout, relative layout, table layout, etc. To specify views inside absolute layout, you have to use *android:layout_x* for x-coordinate and *android:layout_y* for y-coordinate

! It is a little deprecated!

RelativeLayout : a view group that displays child views in relative positions.

- in positions relative to the parent (aligned to the bottom, left or center);
- relative to sibling elements (such as to the left-of or below another view)

Relative layouts are one of the more common types of layouts in android

TableLayout: arranges its children/controls into rows and columns

The Anatomy of an Android Application : ***Android Intents***

Android Services –a media player service implementation MediaService.java

```
package com.mafteiuscai.liviu;
import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.provider.Settings;
import android.widget.Toast;

public class MediaService extends Service {
    private MediaPlayer player;
    public IBinder onBind(Intent intent) {
        return  null;
    }
    public void onCreate() {
        Toast.makeText(this, “MediaService was created", Toast.LENGTH_LONG).show();
    }
```

Binds MainActivity and Service (like in a client-server application)

? when and how many times the text “MediaService was created” ***is displayed?***

The Anatomy of an Android Application : ***Android Intents***

Android Services –a media player service implementation

MediaService.java *- part2*

```
  public int onStartCommand(Intent intent, int flags, int startId) {
    player = MediaPlayer.create(this, Settings.System.DEFAULT_RINGTONE_URI);

    player.setLooping(true);

    player.start();
    Toast.makeText(this, “MediaService Started", Toast.LENGTH_LONG).show();
    return START_STICKY;
  }
  public void onDestroy() {
    super.onDestroy();

    player.stop();
    Toast.makeText(this, “MediaService Stopped", Toast.LENGTH_LONG).show();
  }
}
```

play the ringtone until the service is stoped

start the mediaplayer

START_STICKY- tells the system to create a fresh copy of the service, when sufficient memory is available, after it recovers from low memory.
The computed results -before- will be lost.

stop mediaplayer and destroy the service

Toast*: a view containing a little message for the user*

? when and how many times are displayed these messages?

Android Services –a media player service implementation

AndroidManifest.xml

A service must be registered in Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mafteiuscai.liviu">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true“
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name="MediaService" />
    </application>
</manifest>
```

support right-to-left (RTL) layouts.
default value is false (min API = 17)

The Anatomy of an Android Application : ***Android Intents***

Content Providers

Content Providers implement a mechanism for the sharing of data between applications.

Access to the data is provided via a Universal Resource Identifier (URI) defined by the Content Provider.

Data can be shared in the form of a file or an entire SQLite database.

The native Android applications include a number of standard Content Providers allowing applications to access data such as contacts and media files.

The Anatomy of an Android Application : ***Android Intents***

Content Providers

Applications can store their data in text files or SQLite databases or even over a network.

A content provider is useful when an app wants to share its data with other apps.

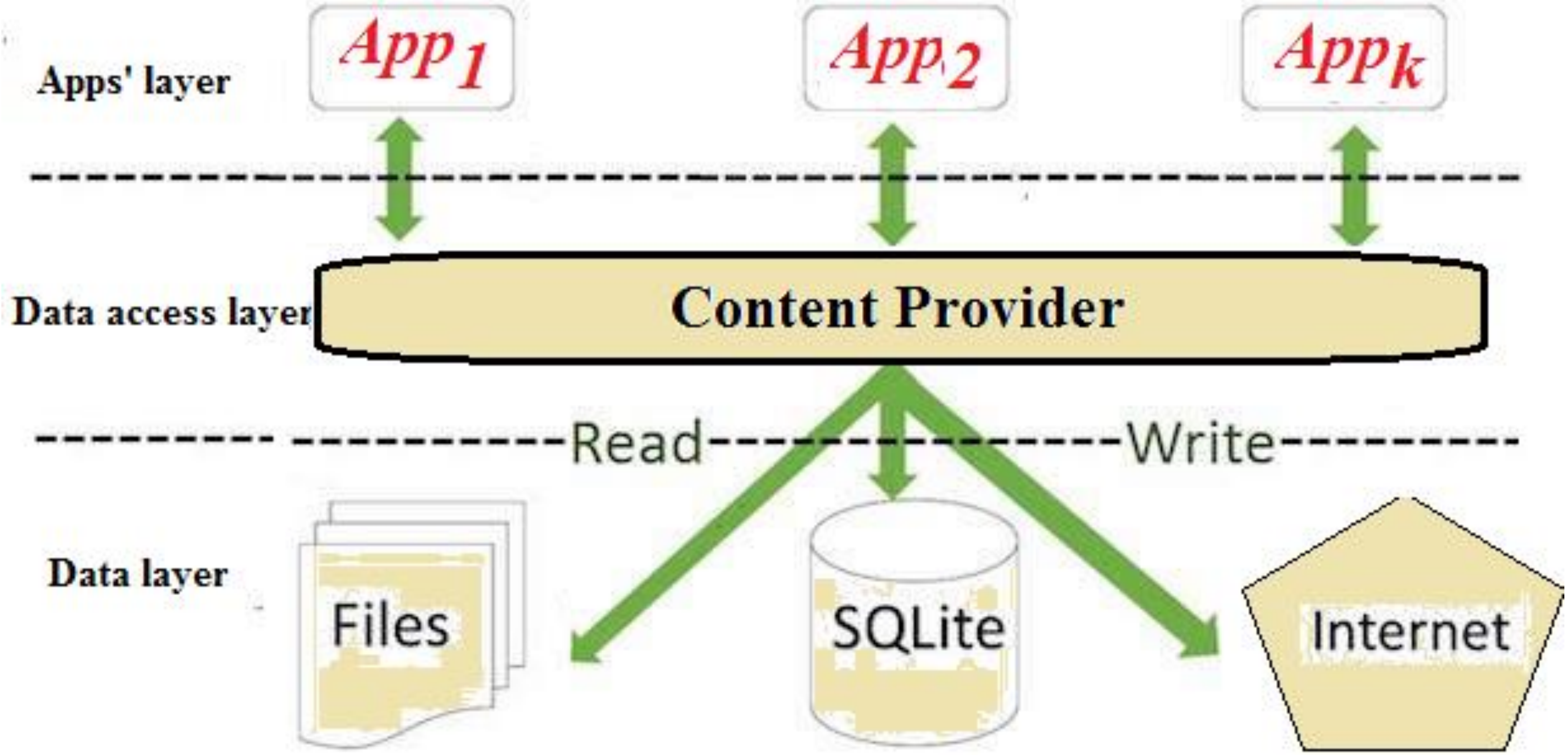
A content provider component supplies data from one application to others on request.

A content provider behaves very much like a database: you can query it, edit its content, add or delete content using *insert(), update(), delete()*, and *query()* methods

A content provider is implemented as a subclass **ContentProvider** class:
public class MyApp extends ContentProvider {….}

The Anatomy of an Android Application : *Android Intents*

Content Providers



The Anatomy of an Android Application : *Android Intents*

Content Providers

How to code a content provider?

```
import android.content.ContentProvider;
….
public class ExampleContentProvider extends ContentProvider {
        public ExampleContentProvider() { }
……
        public boolean onCreate() {
                …….
                return false; }
…….
}
```

The Anatomy of an Android Application : *Android Intents*

The Application Manifest

The glue that pulls together the various elements that comprise an application is the Application Manifest file: activities, services, broadcast receivers, data providers and and so on.

Application Resources

In addition to the Manifest file and the DEX files that contain the byte code, an Android application package will also typically contain a collection of *resource files*.

These resources files contain strings, images, fonts and colors that appear in the user interface together with the XML representation of the user interface layouts.

By default, these files are stored in the ***…/res*** sub-directory of the application project's hierarchy.

The Anatomy of an Android Application : ***Android Intents***

Application Context

When an application is compiled, **a class named *R*** is created that contains references to the application resources.

ApplicationManifest file + R class/file = *Application Context*

Application Context (represented by the Android *Context* class) be used in the application code to gain access to the application resources at runtime.

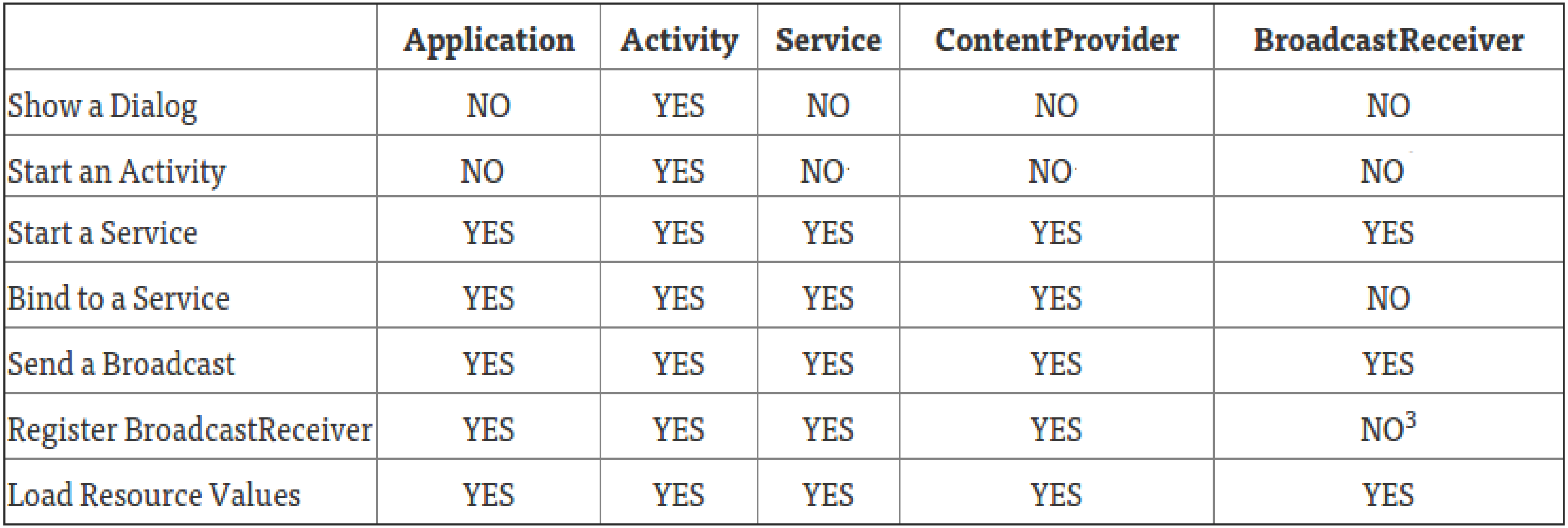
Difference between *Activity Context* and *Application Context*

Even they are both instances of *Context*,
-Activity Context refers the lifecycle of an activity
-Application Context refers the lifecycle of the application

The Anatomy of an Android Application : *Android Intents*

Application Context

Recommendations in using different types of Contexts



	Application	Activity	Service	ContentProvider	BroadcastReceiver
Show a Dialog	NO	YES	NO	NO	NO
Start an Activity	NO	YES	NO	NO	NO
Start a Service	YES	YES	YES	YES	YES
Bind to a Service	YES	YES	YES	YES	NO
Send a Broadcast	YES	YES	YES	YES	YES
Register BroadcastReceiver	YES	YES	YES	YES	NO[3]
Load Resource Values	YES	YES	YES	YES	YES

Fragments

Fragments

Fragments are is reusable component that encapsulates functionality.

Fragment has its own life cycle but it depends on its Activity.

It cannot be used apart from the activity. If the activity is stopped then the fragment cannot be started and if the Activity is destroyed all fragments inside that activity destroyed automatically.

Fragment has its own layout or user interface and it is also possible to create Fragment without user interface.

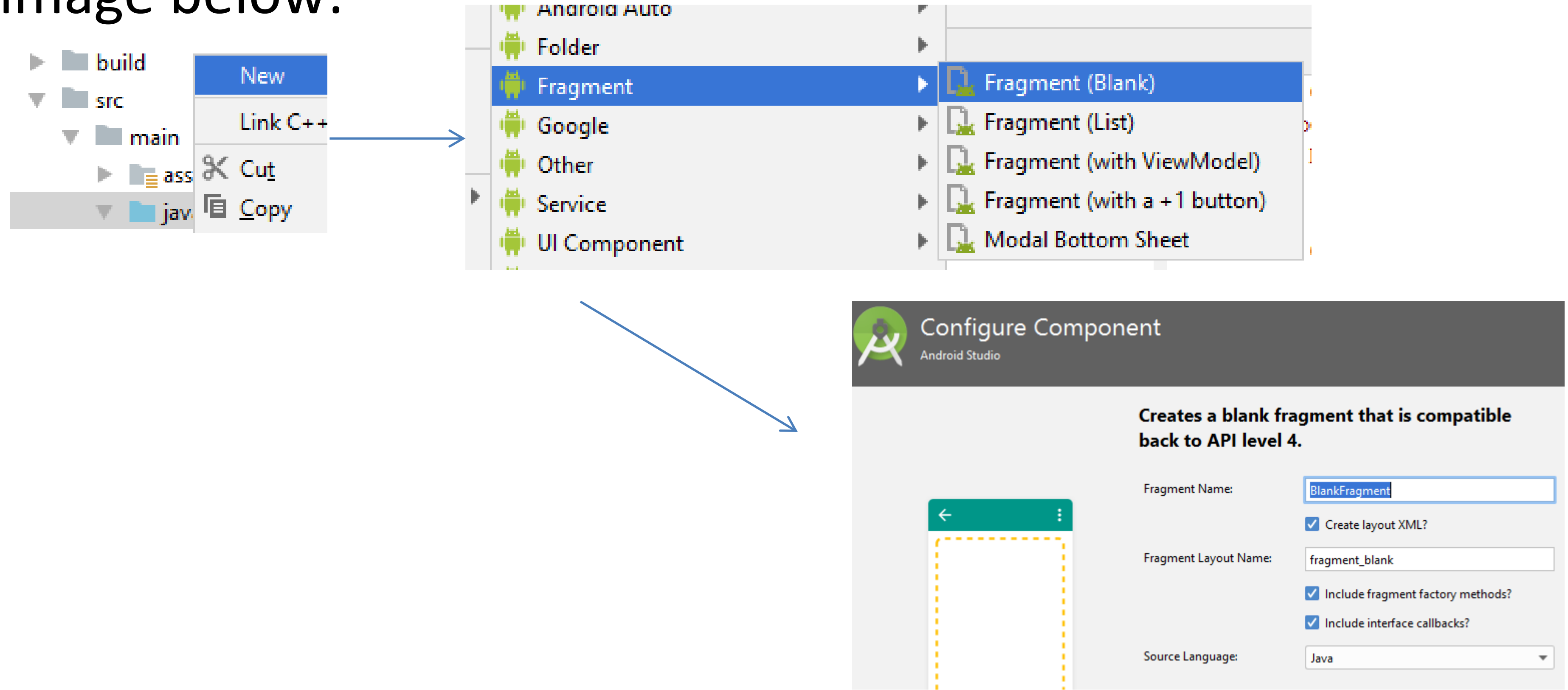
Fragment can be added dynamically or statically in the activity

Fragments

How to create Fragment?

Fragment creation is almost similar to the Activity.

Use Android studio to create new Fragment: create new fragment by right clicking on the java folder as you can see in image below:



Creates a blank fragment that is compatible back to API level 4.

Fragment Name: BlankFragment

☑ Create layout XML?

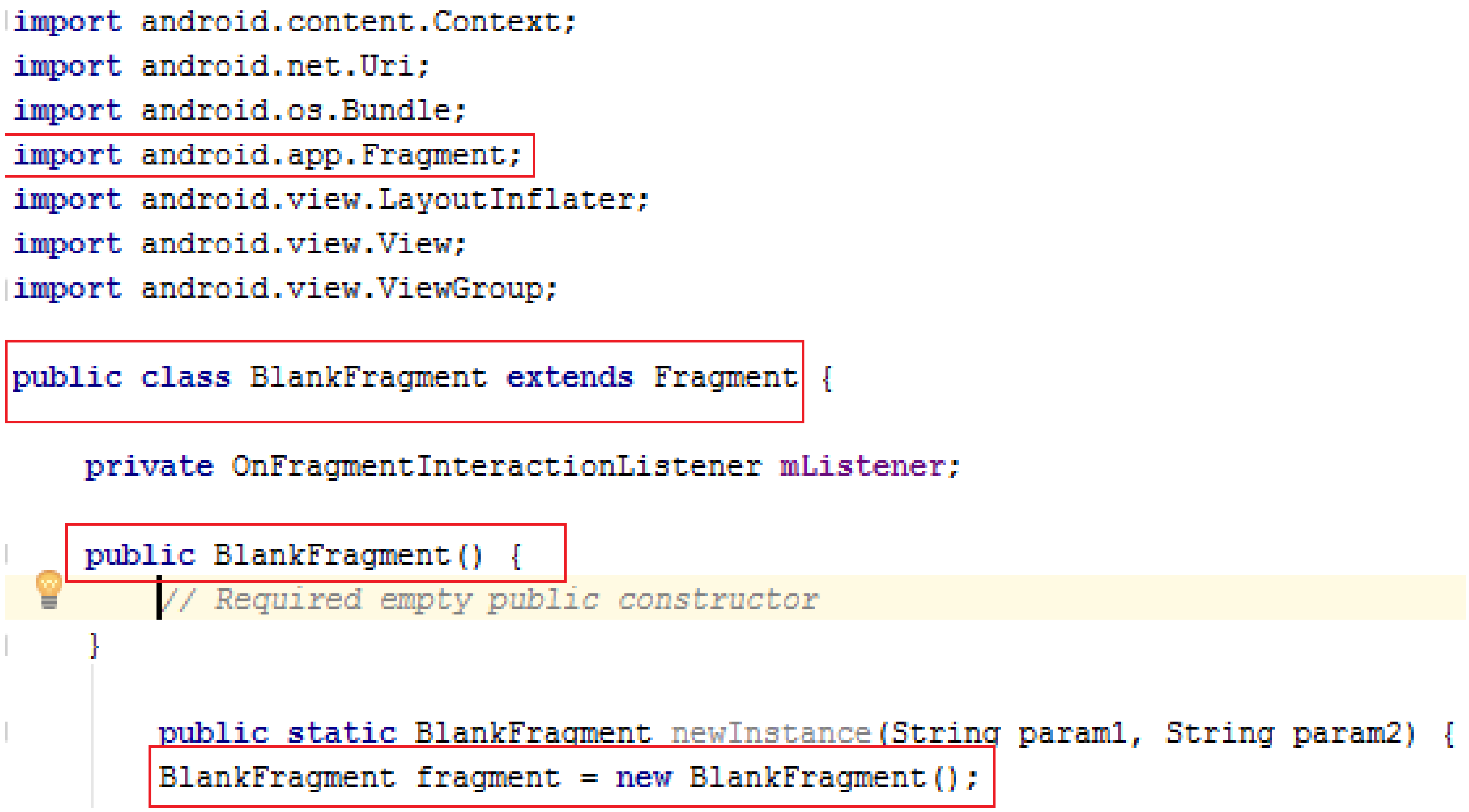
Fragment Layout Name: fragment_blank

☑ Include fragment factory methods?

☑ Include interface callbacks?

Source Language: Java

Fragments



```
import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class BlankFragment extends Fragment {

    private OnFragmentInteractionListener mListener;

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

        public static BlankFragment newInstance(String param1, String param2) {
        BlankFragment fragment = new BlankFragment();
```

Fragments

A Fragment is a little similar to the Activity and it has its own life cycle.

Fragment contains callback methods similar to activity such as *onStart*, *onPause* and *onStop*.

The Fragment always render inside the Activity like controls in other programming languages. You can add <fragment> inside the layout file of the activity and specifies the properties:

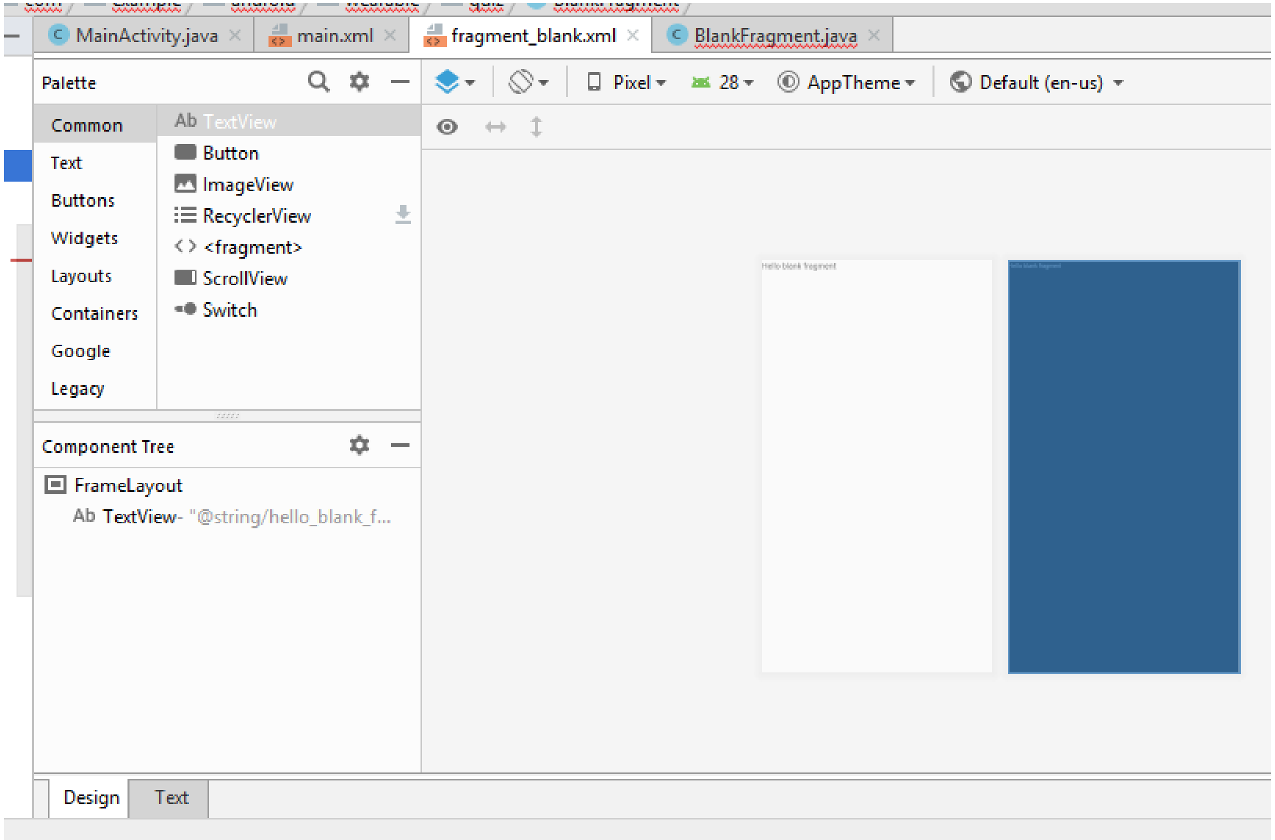
```
<fragment android:name="com.example.fragment.
InformationFragment "
android:layout_height="match_parent"
android:layout_width="match_parent">
</fragment>
```

Fragments

Set Fragment Layout

The layout designing for the fragment is same as Activity.

The layout contains the controls definition in the XML and you can also design using the Design mode



Fragments

An Activity hosting a Fragment can send data to and receive data from the Fragment.

Recommendation: A Fragment can't communicate directly with another Fragment, even within the same Activity. The host Activity must be used as an intermediary.

In order for an activity to communicate with a fragment, the activity must identify the fragment object via the ID assigned to it using the *findViewById()* method.

Once this reference has been obtained, the activity can simply call the public methods of the fragment object.

Fragments

Summary

-Fragments provide a powerful mechanism for creating re-usable modules of user interface layout and application behavior, which, once created, can be embedded in activities.

-A fragment consists of a user interface layout file and a class.

-All communication between fragments should be performed via the activity within which the activities are embedded.

-To design an interface, you can design fragments and put it together.

-A Fragment represents a portion of a user interface or an operation that runs within an Activity.

-A single activity can contain multiple fragments and many fragments can be reused in many and different activities.

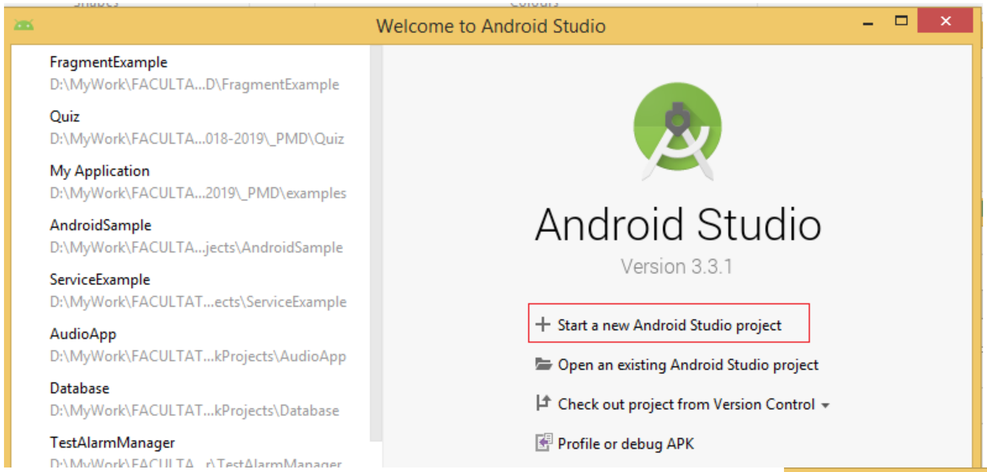
-It is not wrong if we say that a fragment is a type of sub-activity that can be utilized again in multiple activities.

-Even if each fragment has each own lifecycle, because it is connected with the Activity, it's lifecycle is directly influenced by the activity's lifecycle.

-The main advantage of using fragments is due to the convenience of reusing the components in different layouts.

Fragments -> *Example*

Start a New Project –> Empty activity –



Welcome to Android Studio

FragmentExample
D:\MyWork\FACULTA...D\FragmentExample

Quiz
D:\MyWork\FACULTA...018-2019_PMD\Quiz

My Application
D:\MyWork\FACULTA...2019_PMD\examples

AndroidSample
D:\MyWork\FACULTA...jects\AndroidSample

ServiceExample
D:\MyWork\FACULTAT...ects\ServiceExample

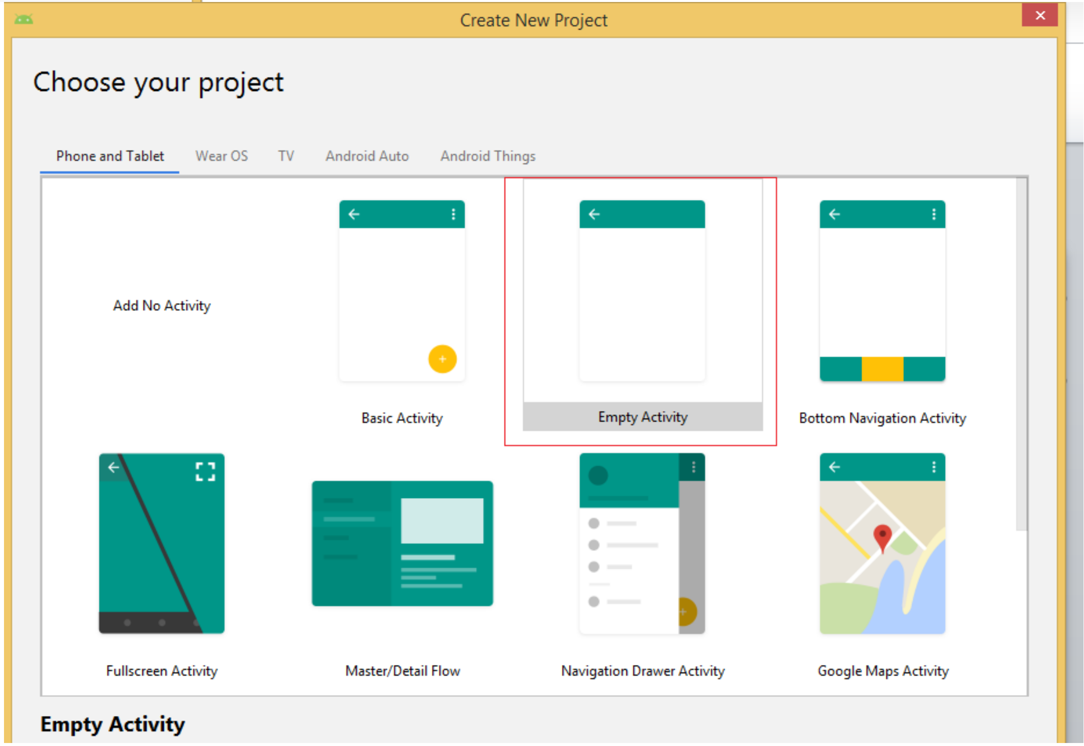
AudioApp
D:\MyWork\FACULTAT...kProjects\AudioApp

Database
D:\MyWork\FACULTAT...kProjects\Database

TestAlarmManager
D:\MyWork\FACULTA...r\TestAlarmManager

Android Studio
Version 3.3.1

+ Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK

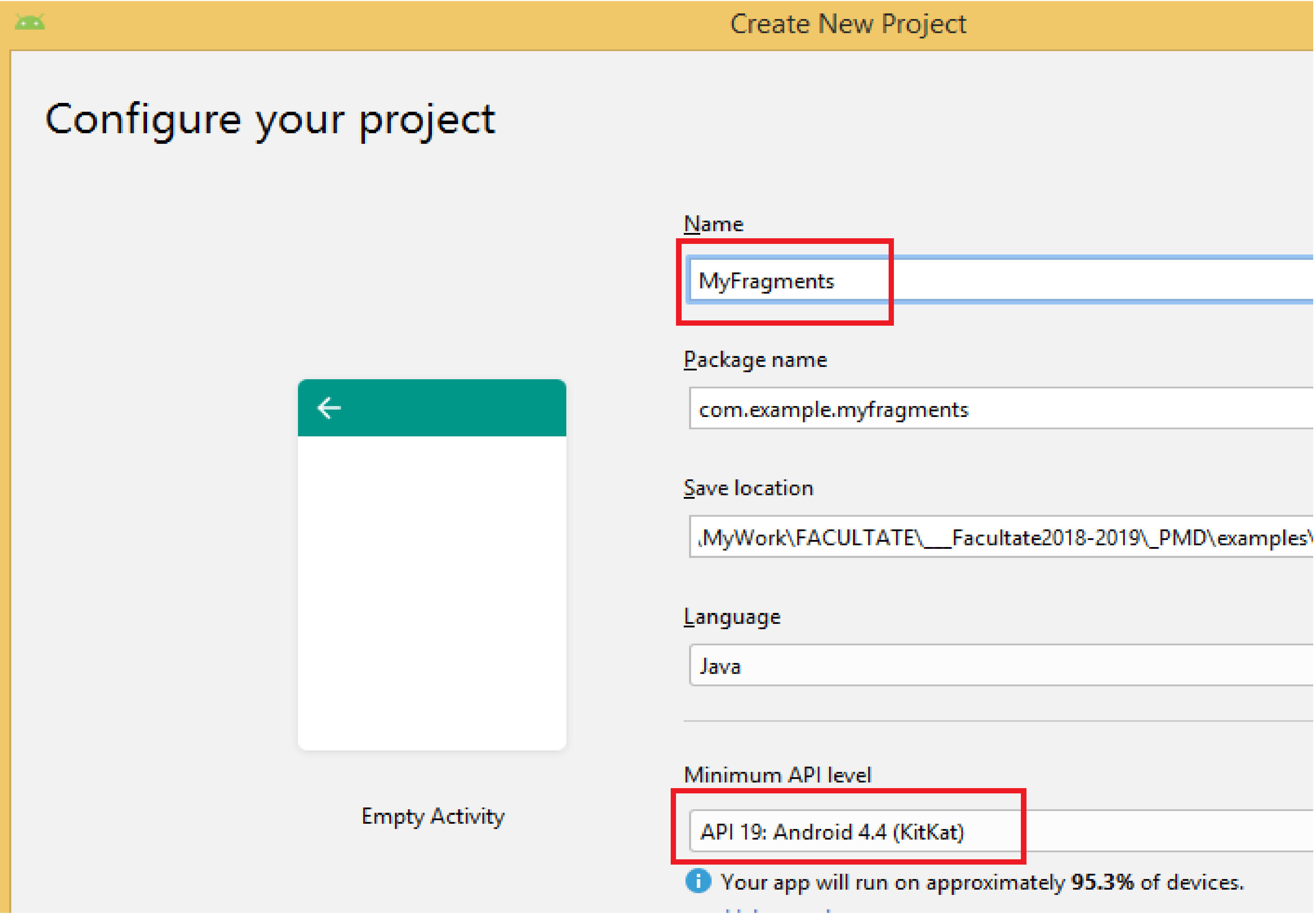


Create New Project

Choose your project

Empty Activity

Fragments -> *Example*



Create New Project

Configure your project

Name

MyFragments

Package name

com.example.myfragments

Save location

.MyWork\FACULTATE\___Facultate2018-2019_PMD\examples\

Language

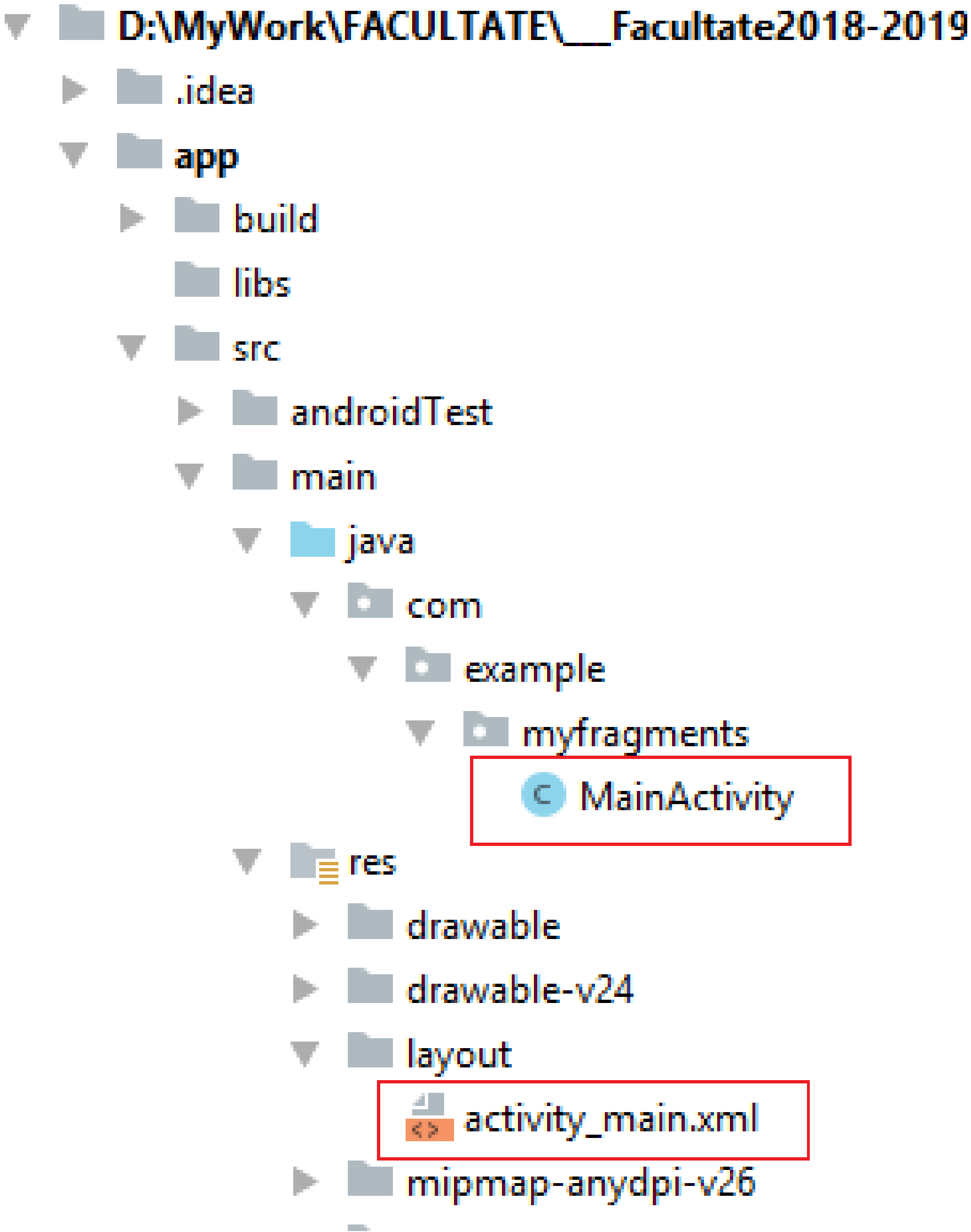
Java

Minimum API level

API 19: Android 4.4 (KitKat)

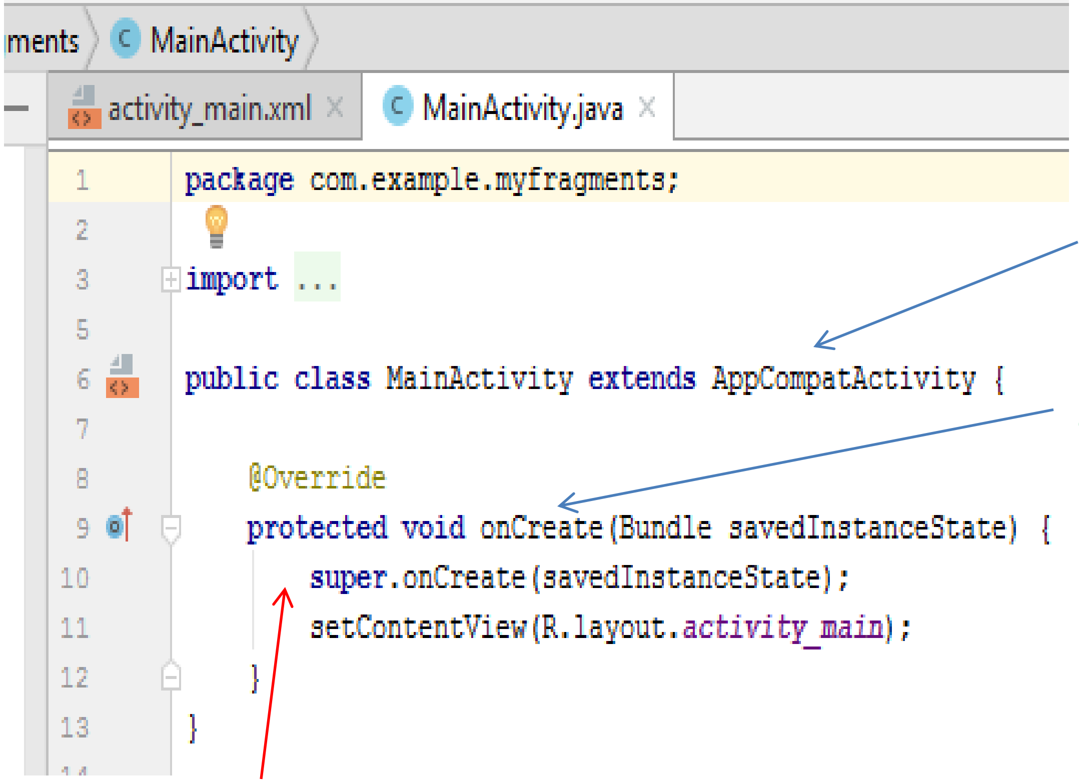
Your app will run on approximately **95.3%** of devices.

Fragments -> *Example*



```
D:\MyWork\FACULTATE\__Facultate2018-2019
  .idea
  app
    build
    libs
    src
      androidTest
      main
        java
          com
            example
              myfragments
                MainActivity
        res
          drawable
          drawable-v24
          layout
            activity_main.xml
          mipmap-anydpi-v26
```

Fragments -> *Example*



```
package com.example.myfragments;

import ...

public class MainActivity extends AppCompatActivity {

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

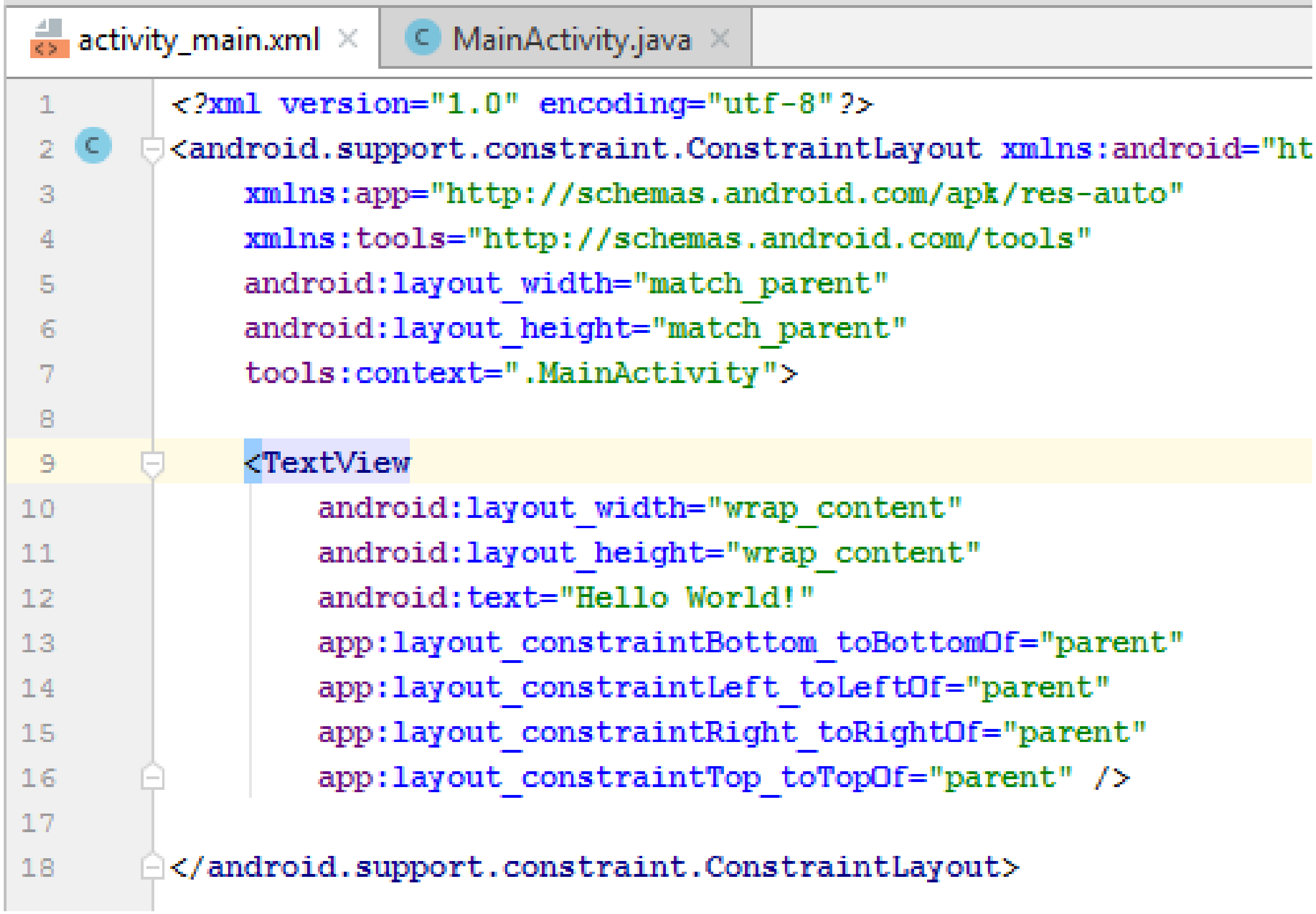
Base class for activities that use the support library action bar features.

When an Activity first call or launched then onCreate(Bundle savedInstanceState) method is responsible to create the activity.

WHY SUPER?

Because the super class potentially also has to execute code to work properly during creation. You are overriding that method in your class and unless you don't call super.onCreate the method in the super class will never be called, potentially leading to unwanted behavior.

Fragments -> *Example*



```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="ht
    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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

A ConstraintLayout is a ViewGroup which allows you to position and size widgets in a flexible way.

- It allows us to lay out child views using 'constraints' to define position based relationships between different views found in our layout.

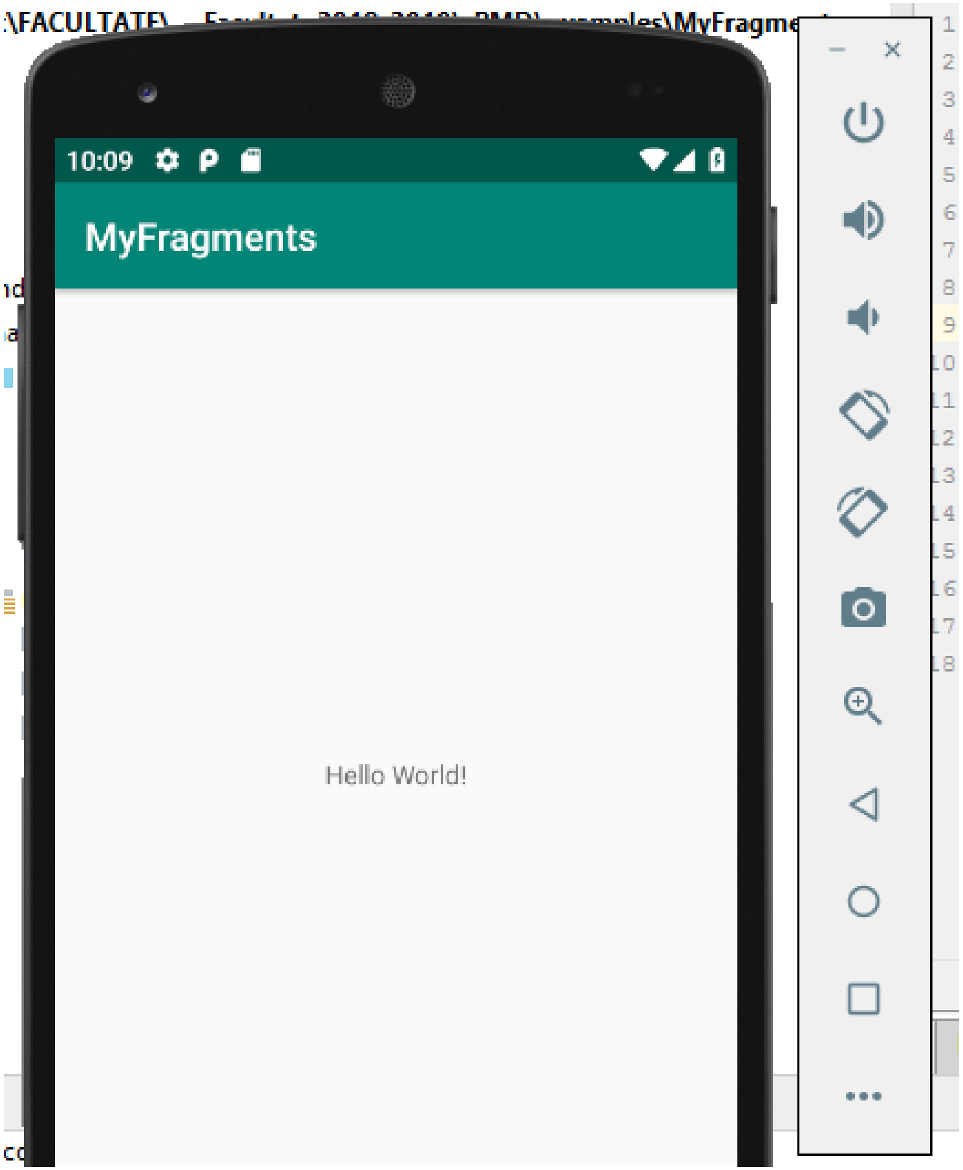
- The aim of the ConstraintLayout is to help reduce the number of nested views, which will improve the performance of our layout files.

Fragments -> *Example* <- ConstraintLayout attributes:

layout_constraintTop_toTopOf - Align the **top** of the desired view to the **top** of another.
layout_constraintTop_toBottomOf - Align the **top** of the desired view to the **bottom** of another.
layout_constraintBottom_toTopOf -Align the **bottom** of the desired view to the **top** of another.
layout_constraintBottom_toBottomOf -Align the **bottom** of the desired view to the **bottom** of another.
layout_constraintLeft_toTopOf - Align the **left** of the desired view to the **top** of another.
layout_constraintLeft_toBottomOf -Align the **left** of the desired view to the **bottom** of another.
layout_constraintLeft_toLeftOf - Align the **left** of the desired view to the **left** of another.
layout_constraintLeft_toRightOf - Align the **left** of the desired view to the **right** of another.
layout_constraintRight_toTopOf - Align the **right** of the desired view to the **top** of another.
layout_constraintRight_toBottomOf- Align the **right** of the desired view to the **bottom** of another.
layout_constraintRight_toLeftOf -Align the **right** of the desired view to the **left** of another.
constraintRight_toRightOf -Align the **right** of the desired view to the **right** of another.

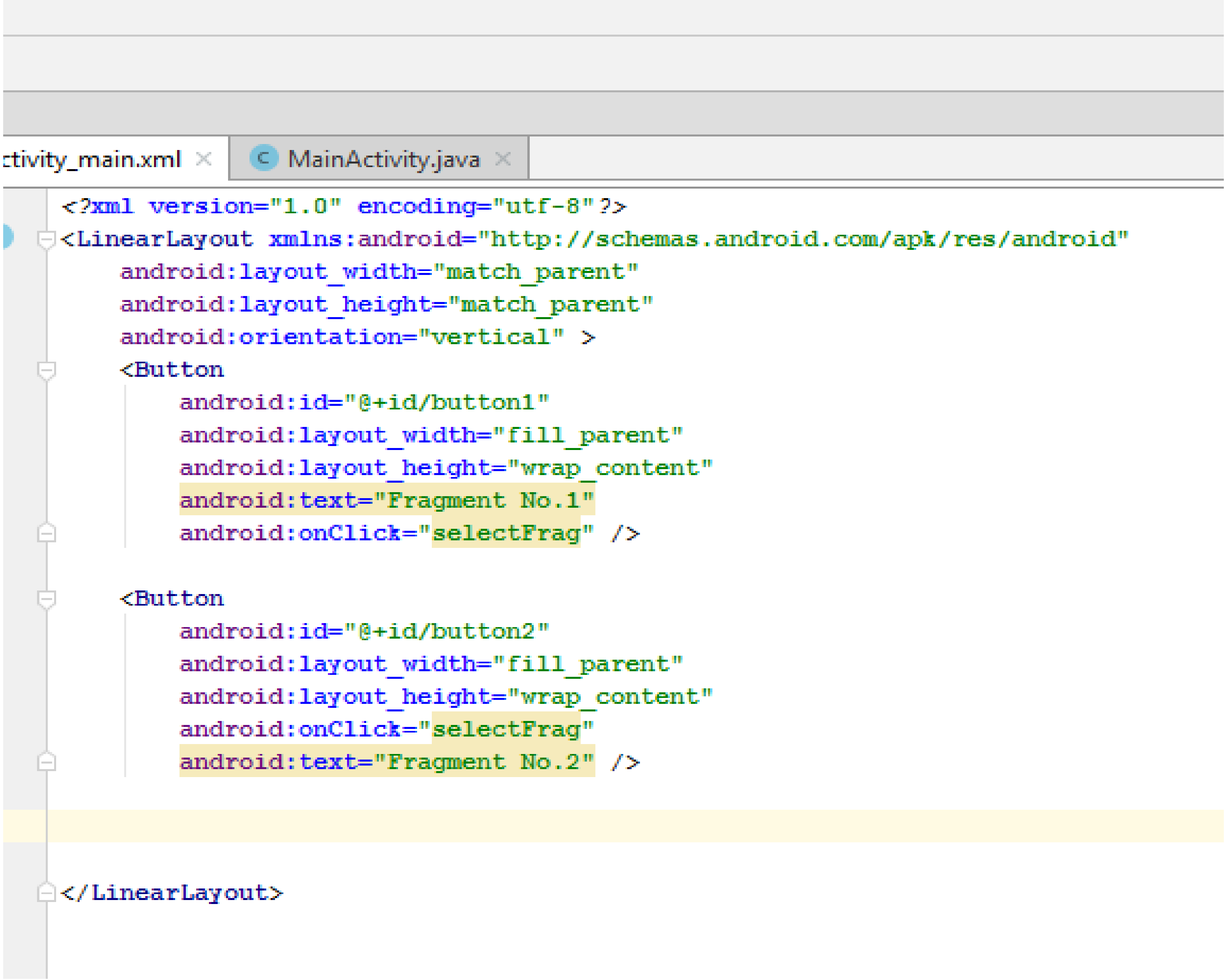
These all give us a great amount of control over the positioning of our views within the *ConstraintLayout*, much more so than that of the *RelativeLayout*.

Fragments -> *Example*



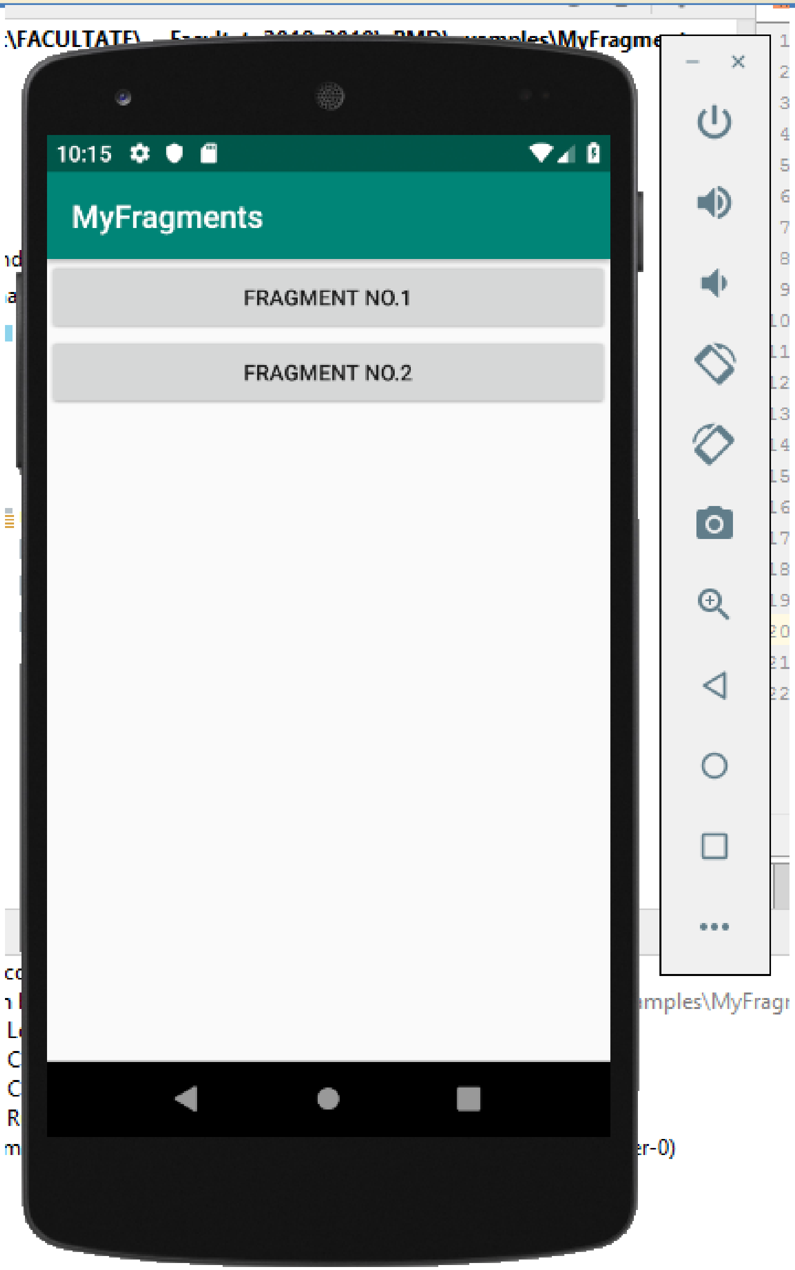
Fragments -> *Example*

ConstraintLayout => LinearLayout



```
<?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:orientation="vertical" >
    <Button
        android:id="@+id/button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Fragment No.1"
        android:onClick="selectFrag" />

    <Button
        android:id="@+id/button2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="selectFrag"
        android:text="Fragment No.2" />



</LinearLayout>
```

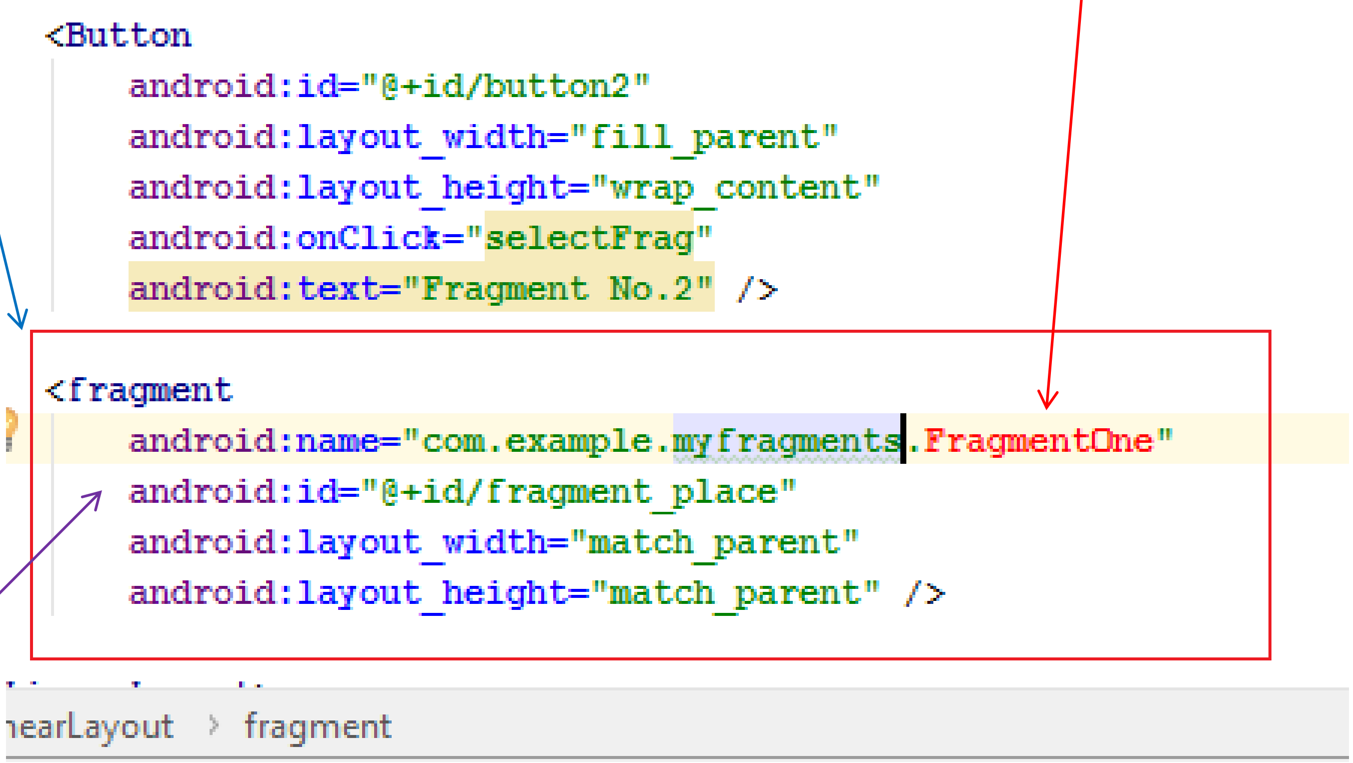
Fragments -> *Example*



Fragments -> *Example*

Add code for first fragment in activity_main.xml

The android:name defines an object of a Fragment Class



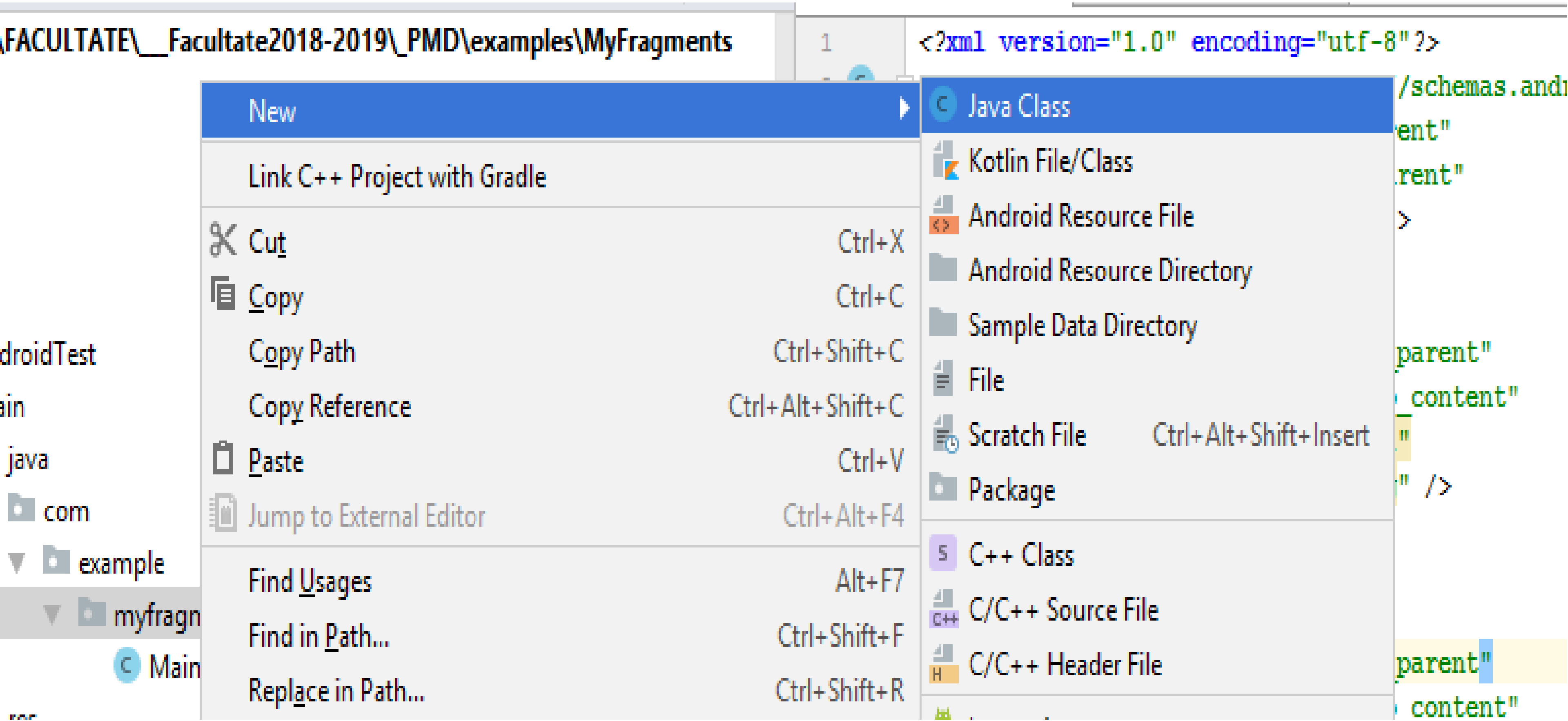
```
<Button
    android:id="@+id/button2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="selectFrag"
    android:text="Fragment No.2" />

<fragment
    android:name="com.example.myfragments.FragmentOne"
    android:id="@+id/fragment_place"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

android:id specifies the unique id of that fragment

Fragments -> *Example*

Add a class called *FragmentOne* to define the fragments, that extends the Fragment Class.

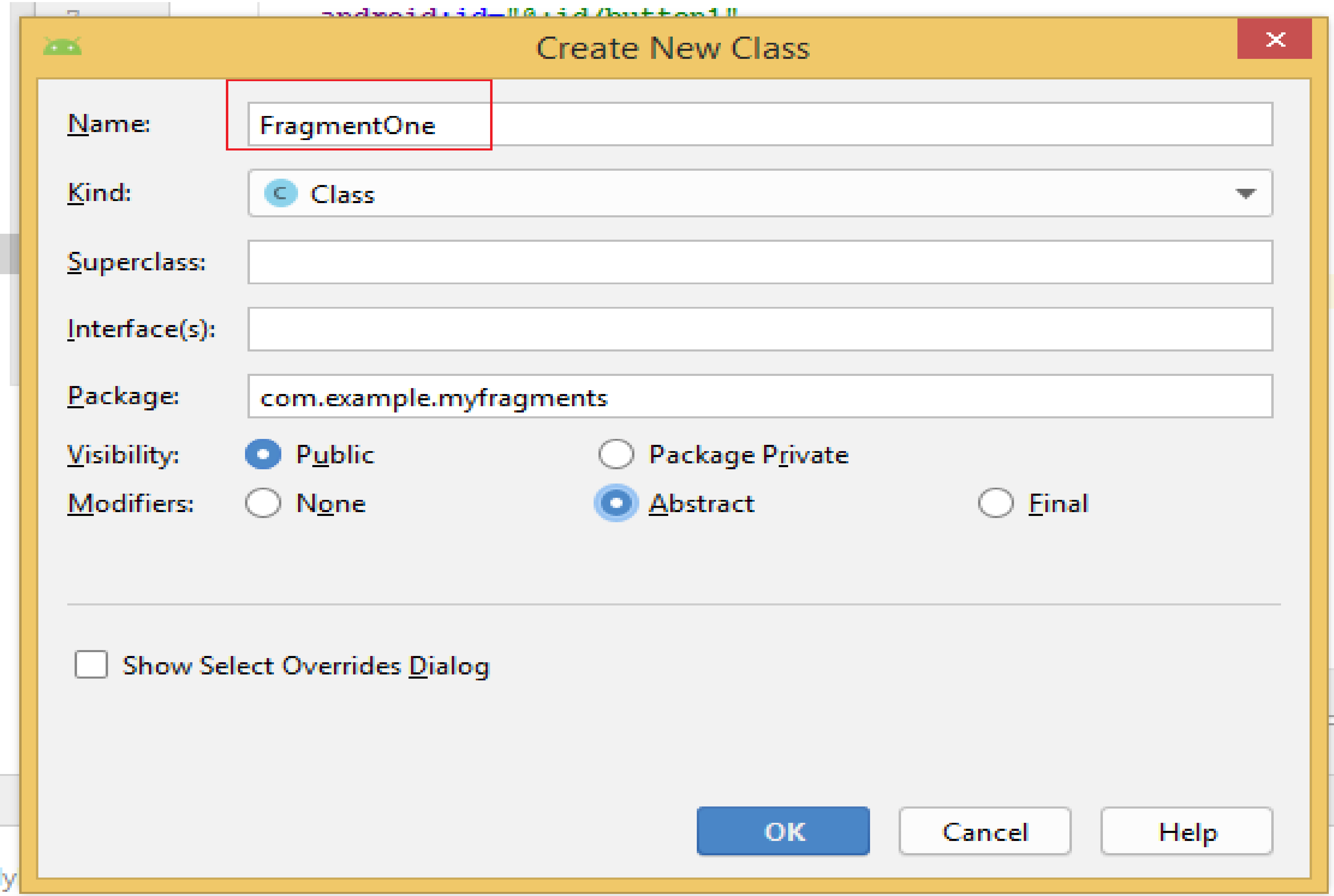
Put it in the same package as the MainActivity.java file.
Right click on package → New → Class.



\FACULTATE__Facultate2018-2019_PMD\examples\MyFragments

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

Fragments -> *Example*



Create New Class

Name: FragmentOne

Kind: Class

Superclass:

Interface(s):

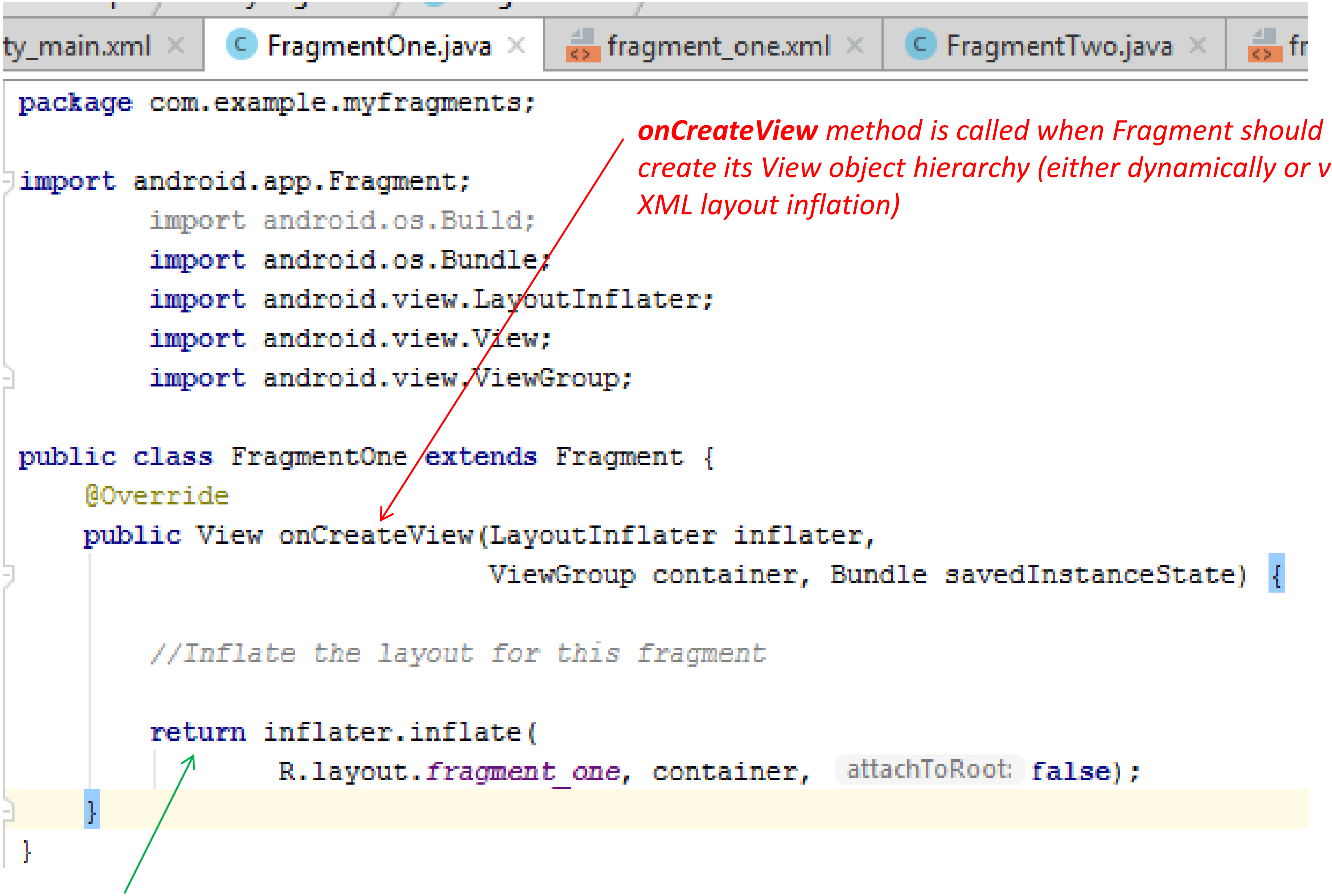
Package: com.example.myfragments

Visibility: ● Public ○ Package Private

Modifiers: ○ None ● Abstract ○ Final

□ Show Select Overrides Dialog

Fragments -> *Example*



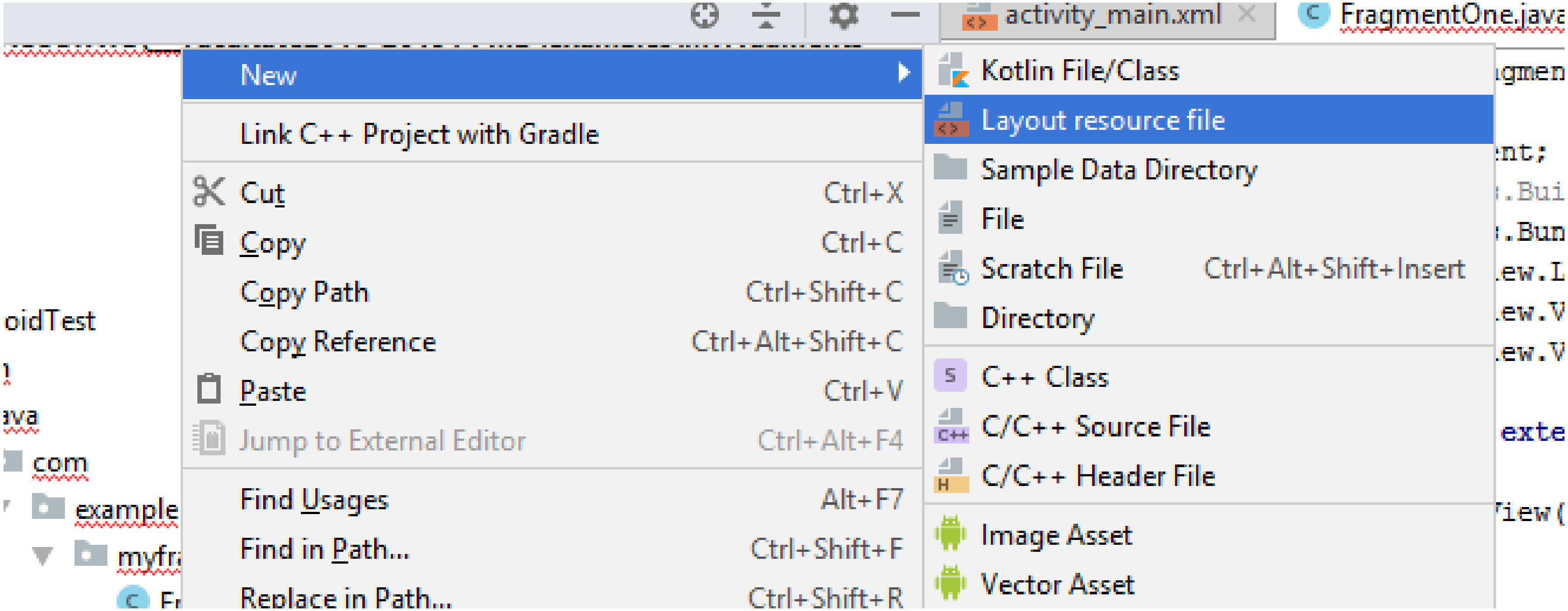
```java
package com.example.myfragments;

import android.app.Fragment;
        import android.os.Build;
        import android.os.Bundle;
        import android.view.LayoutInflater;
        import android.view.View;
        import android.view.ViewGroup;

public class FragmentOne extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {

        //Inflate the layout for this fragment

        return inflater.inflate(
                R.layout.fragment_one, container, attachToRoot: false);
    }
}
```

***onCreateView** method is called when Fragment should create its View object hierarchy (either dynamically or via XML layout inflation)*

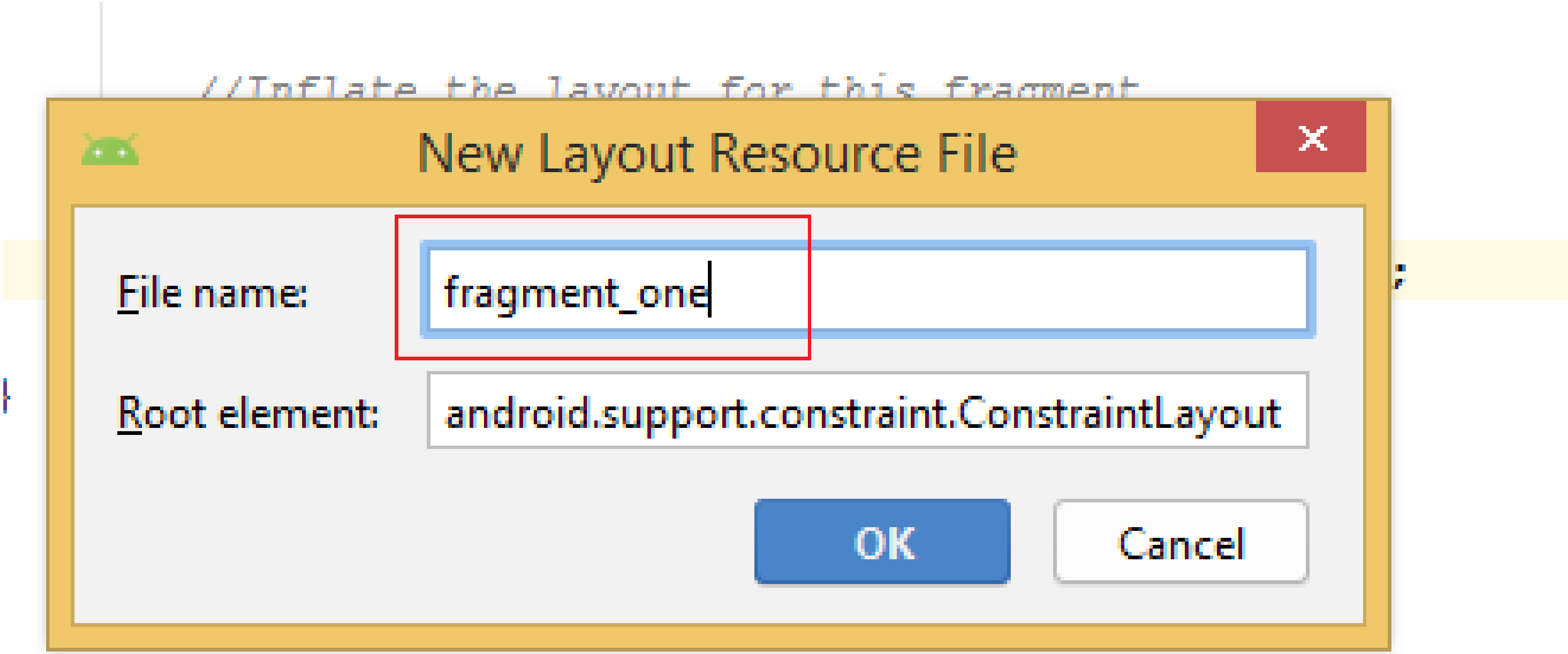
Defines the xml file for the fragment

Fragments -> *Example*

Create the layout of the Fragment.
Create a simple UI for fragment layout: fragment_one.xml
Right click on res/layout folder → New → Android XML File and name the xml file. Choose the LinearLayout as root element.



Fragments -> *Example*



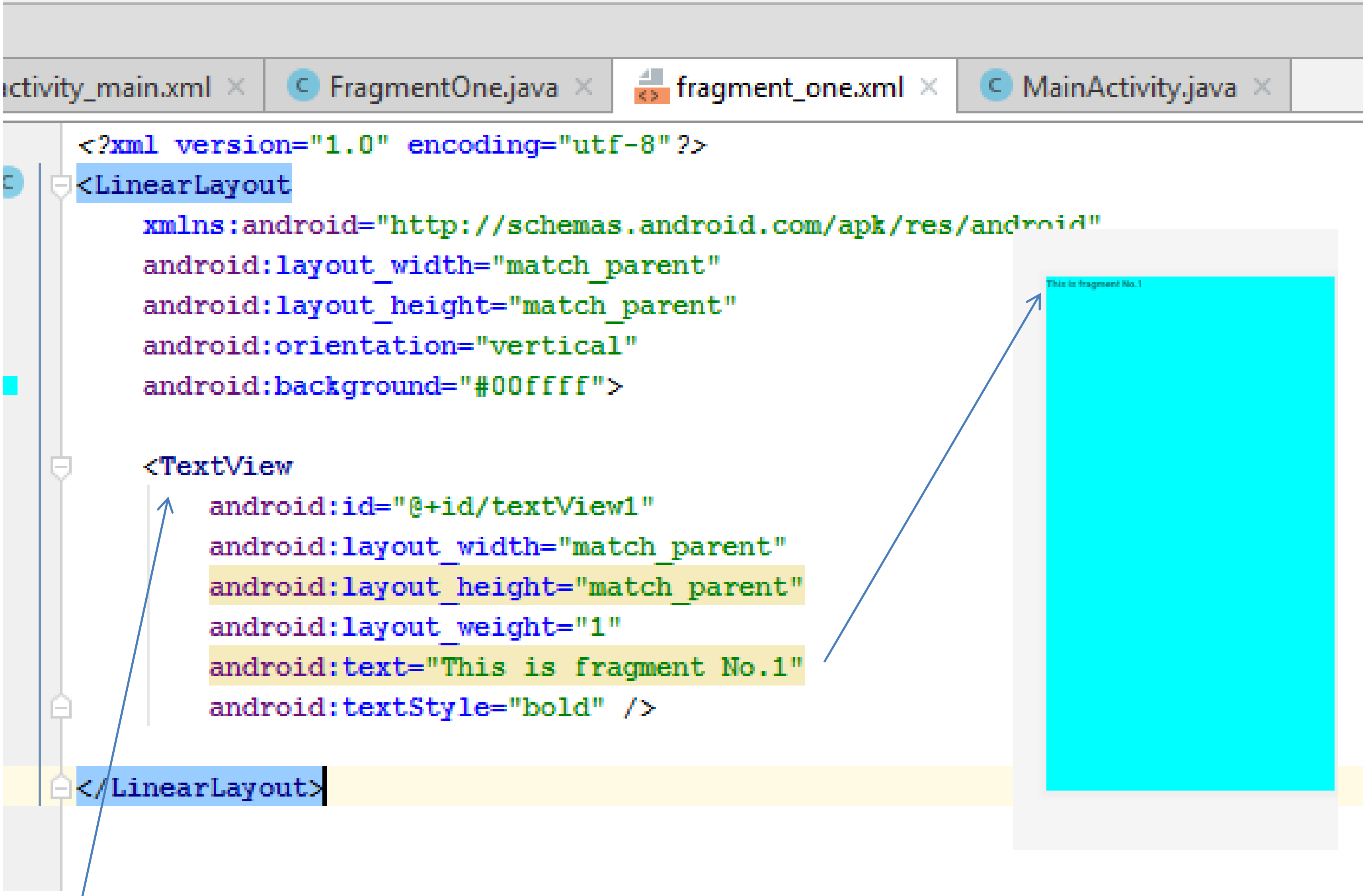
```
//Inflate the layout for this fragment
```

New Layout Resource File

File name: fragment_one

Root element: android.support.constraint.ConstraintLayout

Fragments -> *Example*

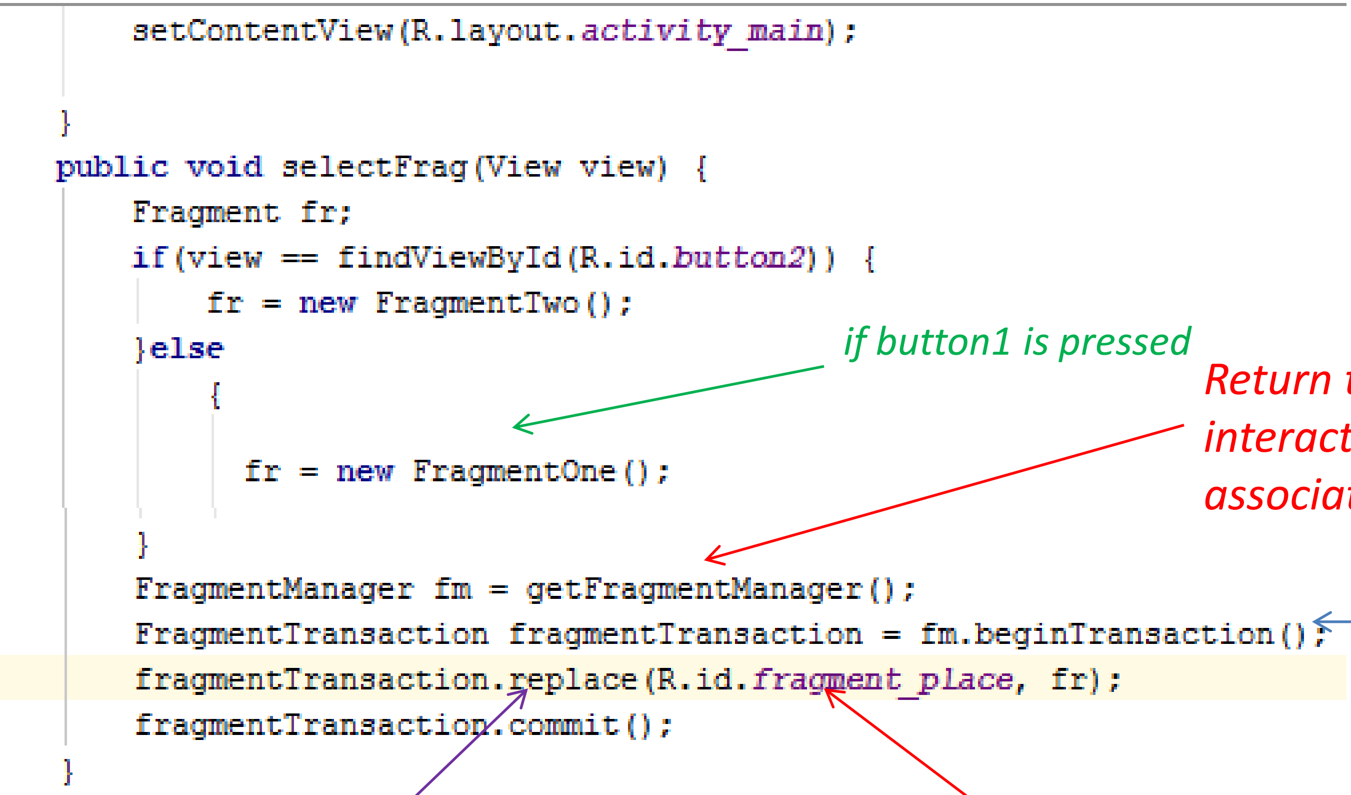


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#00ffff">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="This is fragment No.1"
        android:textStyle="bold" />

</LinearLayout>
```

Add a TextView :

Fragments -> *Example*

Add to MainActivity a method to select fragments:



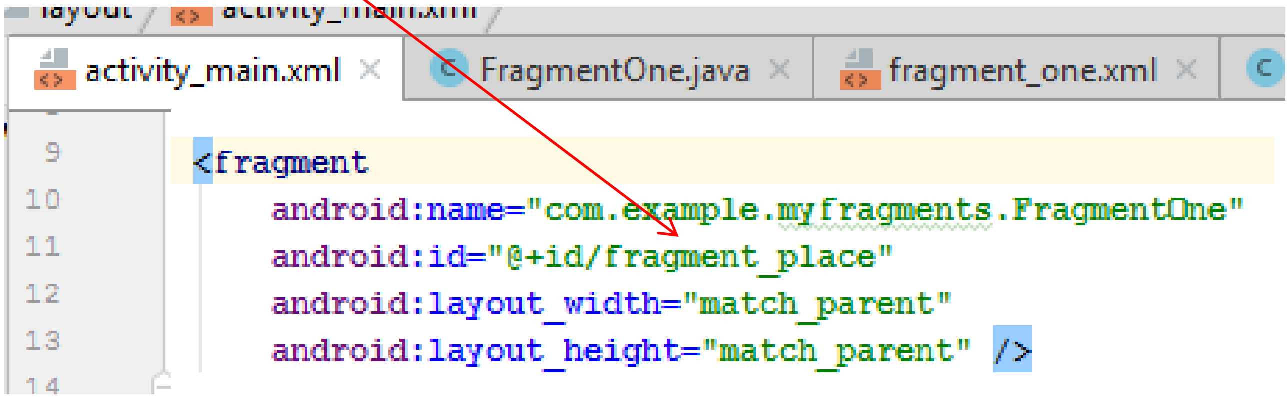
```
    setContentView(R.layout.activity_main);

}
public void selectFrag(View view) {
    Fragment fr;
    if(view == findViewById(R.id.button2)) {
        fr = new FragmentTwo();
    }else
        {

            fr = new FragmentOne();

    }
    FragmentManager fm = getFragmentManager();
    FragmentTransaction fragmentTransaction = fm.beginTransaction();
    fragmentTransaction.replace(R.id.fragment_place, fr);
    fragmentTransaction.commit();
}
```

if button1 is pressed

Return the FragmentManager for interacting with fragments associated with this MainActivity.

the fragment is placed in a FragmentTransaction

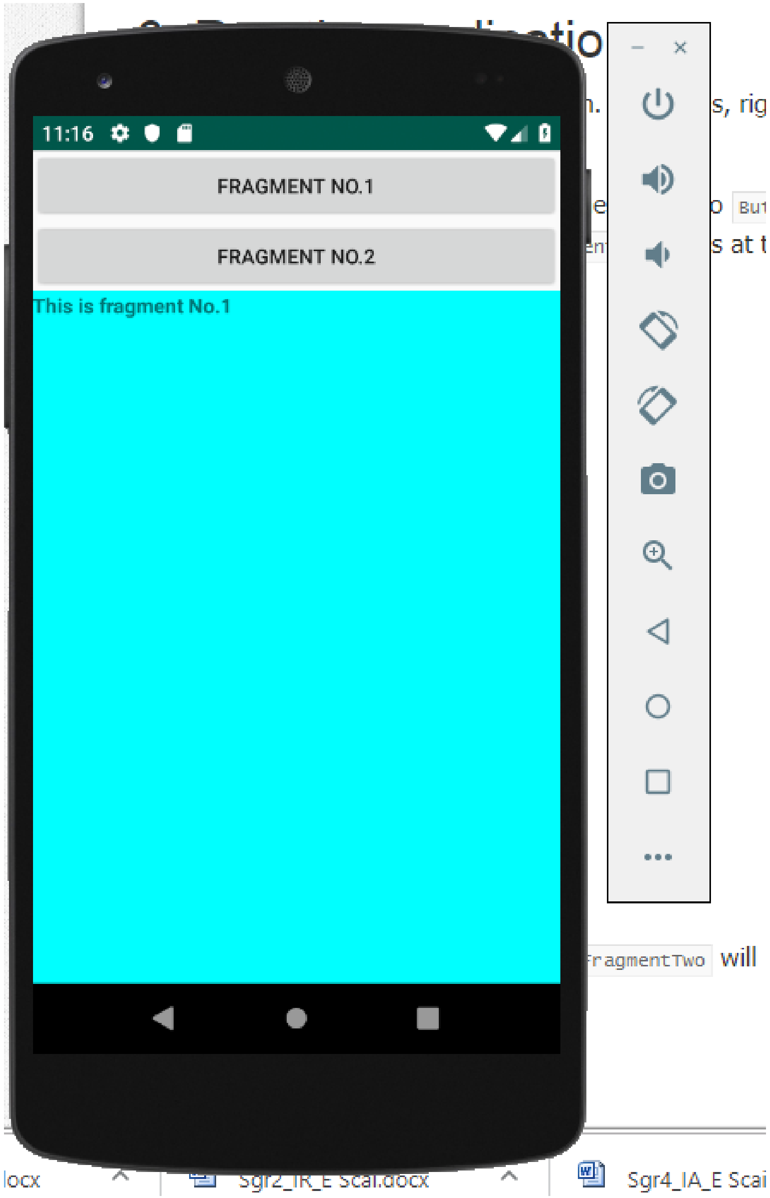
For the given container view id, we can replace existing fragment by new given fragment.



```
<fragment
    android:name="com.example.myfragments.FragmentOne"
    android:id="@+id/fragment_place"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Fragments -> *Example*

•



This is fragment No.1

Fragments - *Example*

Now:

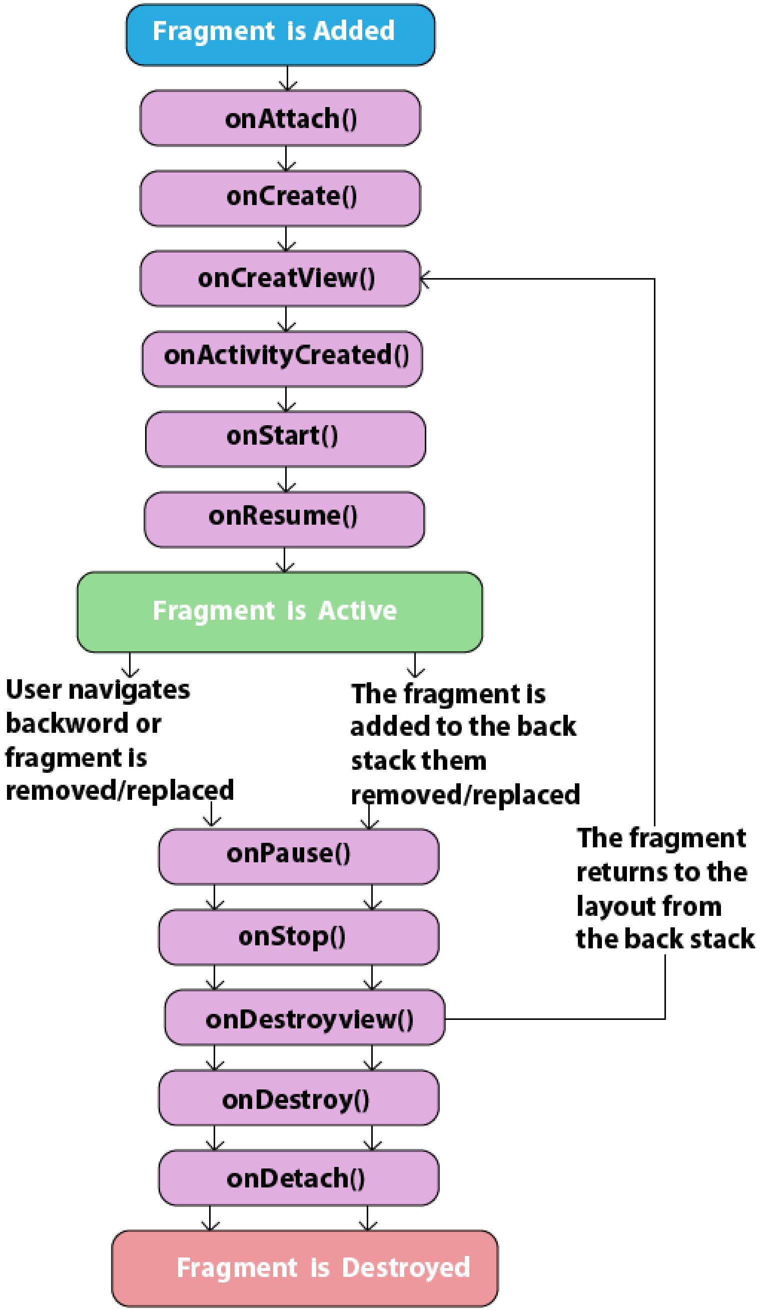
1 Create *FragmentTwo* <u>class</u> and its <u>layout</u>,

2 Modify *MainActivity*,

3 Rebuild,

4 Run

Fragments - *Android Fragment Lifecycle*



The lifecycle of android fragment is like the activity lifecycle. There are 12 lifecycle methods for fragment.

onAttach: When the fragment attaches to its host activity.
onCreate: When a new fragment instance initializes, which always happens after it attaches to the host
onCreateView: When a fragment creates its portion of the view hierarchy, which is added to its activity's view hierarchy.
onActivityCreated: When the fragment's activity has finished its own onCreate event.
onStart: When the fragment is visible; a fragment starts only after its activity starts and often starts immediately after its activity does.
onResume: When the fragment is visible and interactable; a fragment resumes only after its activity resumes and often resumes immediately after the activity does.
Fragment is active
onPause: When the fragment is no longer interactable; this occurs when either the fragment is about to be removed or replaced or when the fragment's activity pauses.
onStop: When the fragment is no longer visible; this occurs either after the fragment is about to be removed or replaced or when the fragment's activity stops.
onDestroyView: When the view and related resources created in onCreateView are removed from the activity's view hierarchy and destroyed.
onDestroy: When the fragment does its final clean up.
onDetach: When the fragment is detached from its activity.

Sending Data to Fragment from Activity

In Activity

One way to get data from activity is by calling a method on the activity that returns data as shown above. Data can also be sent to fragment when it is created by adding data to bundle:

```
Bundle bundle = new Bundle();
bundle.putString("user", "user name");
SampleFragment fragment = new SampleFragment();
fragment.setArguments(bundle);
```

In Fragment

Read the data in onCreateView method of the fragment by calling getArguments() method to get the bundle and calling appropriate methods on it to read values from it.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
String user = getArguments().getString("user");
return inflater.inflate(R.layout.fragment, container, false);
 }
```

Fragments

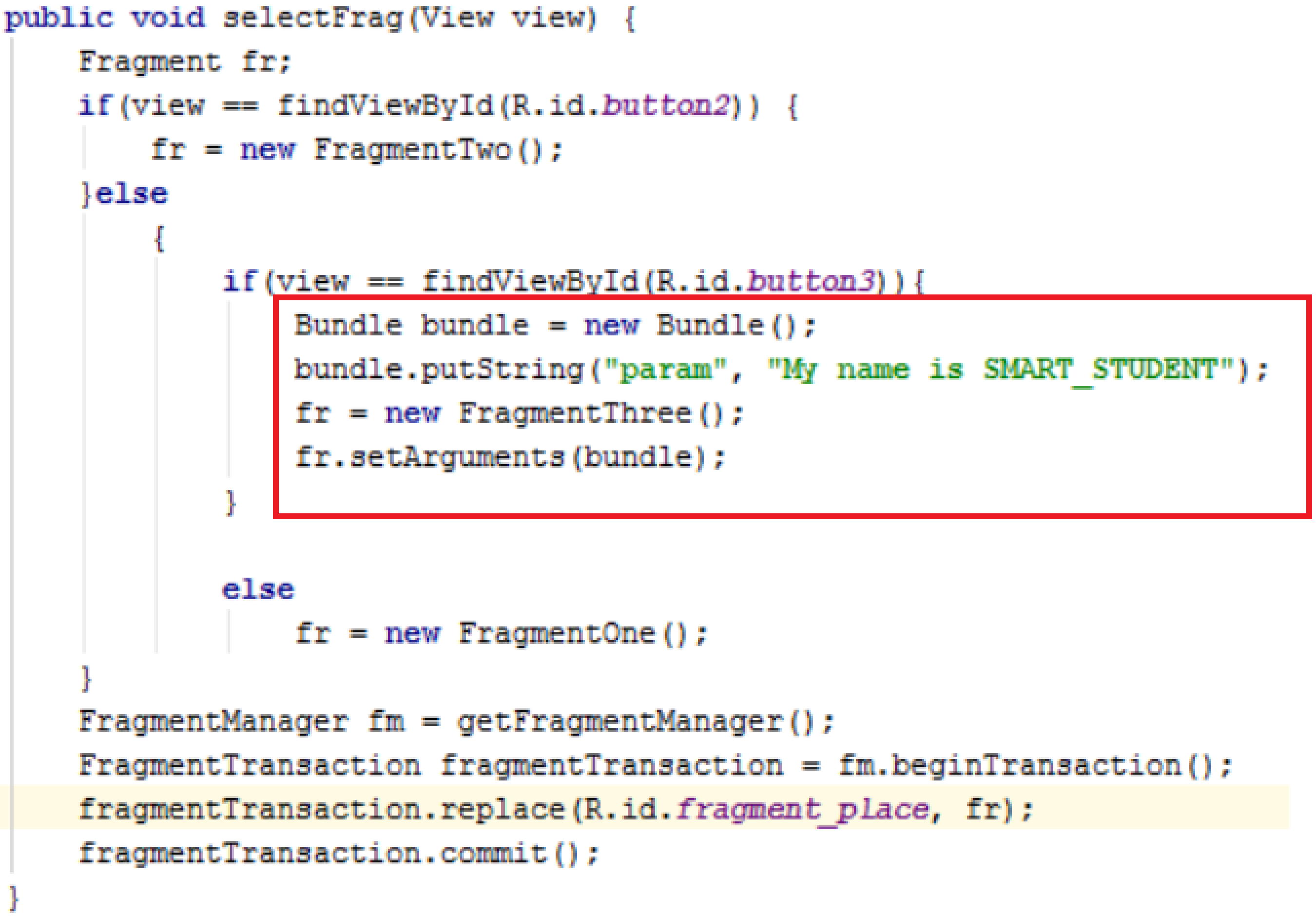
Sending Data from Fragment to Activity

Fragment can send data to activity by calling a setter method in the activity.

```
((FragmentActivity)getActivity()).setData("333");
```

Fragments - Example-continued with FragmentThree – **send data**

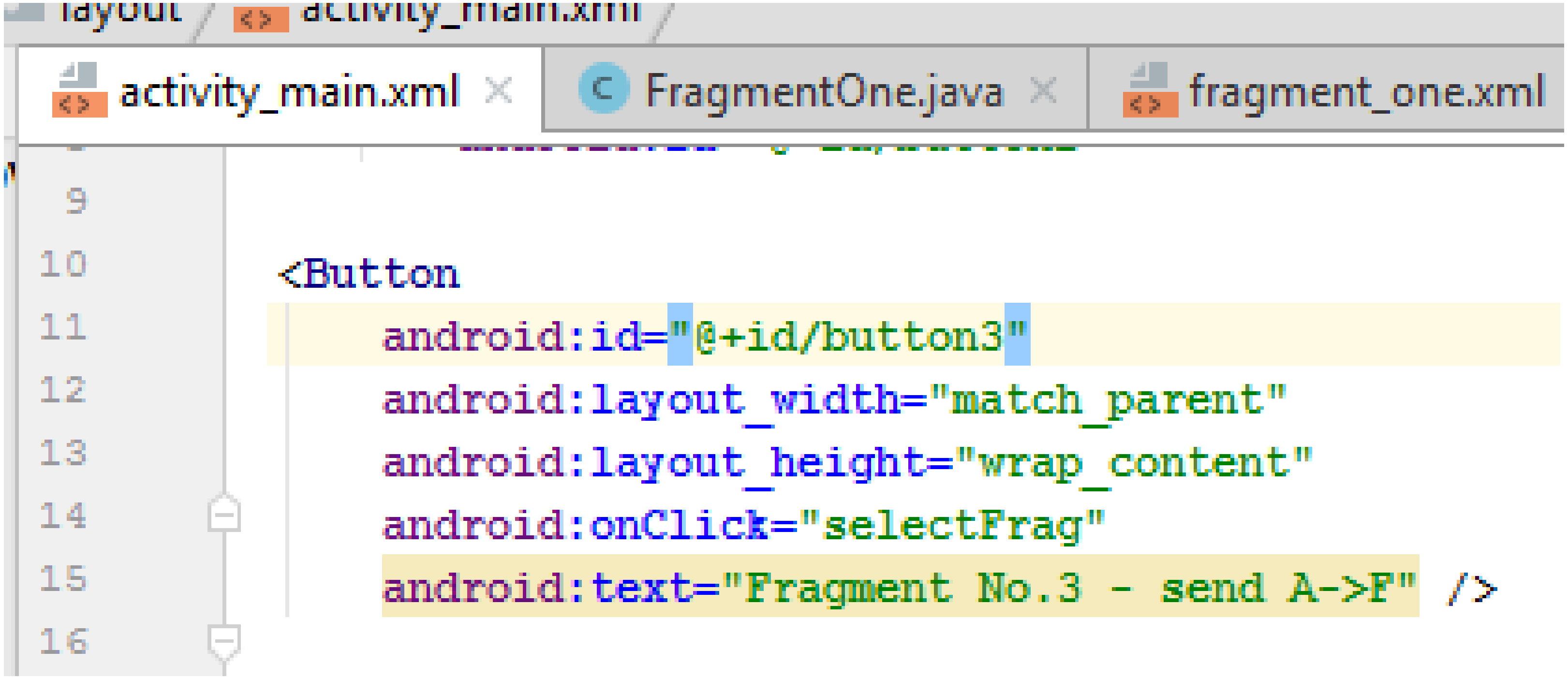
How to pass data (string) from MainActivity to fragment:



```
public void selectFrag(View view) {
    Fragment fr;
    if(view == findViewById(R.id.button2)) {
        fr = new FragmentTwo();
    }else
        {
            if(view == findViewById(R.id.button3)){
                Bundle bundle = new Bundle();
                bundle.putString("param", "My name is SMART_STUDENT");
                fr = new FragmentThree();
                fr.setArguments(bundle);
            }

            else
                fr = new FragmentOne();
    }
    FragmentManager fm = getFragmentManager();
    FragmentTransaction fragmentTransaction = fm.beginTransaction();
    fragmentTransaction.replace(R.id.fragment_place, fr);
    fragmentTransaction.commit();
}
```

Fragments

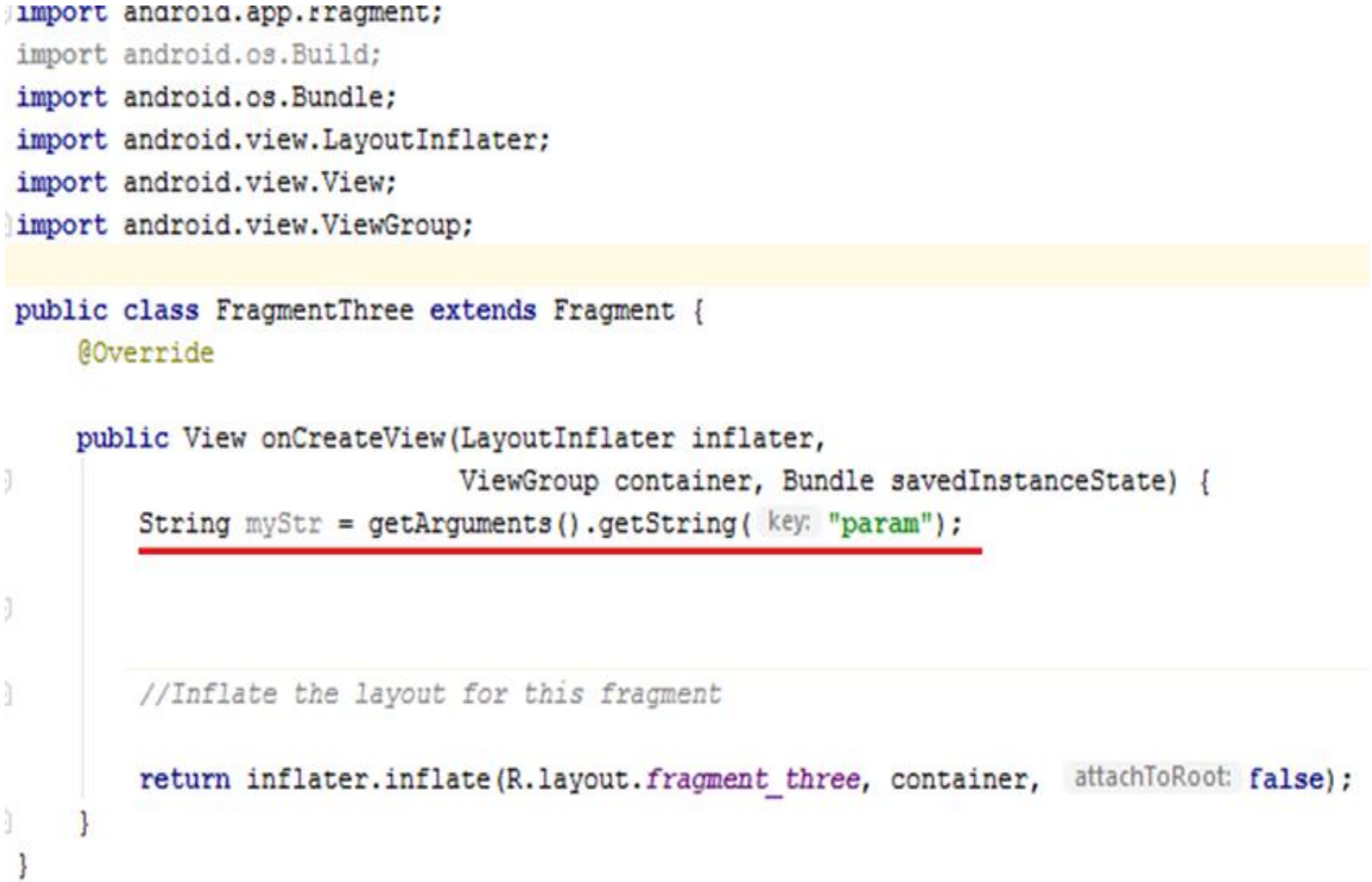
Add a new button on activity_main.xml



```
<Button
    android:id="@+id/button3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="selectFrag"
    android:text="Fragment No.3 - send A->F" />
```

Fragments

Create java class for FragmentThree.:



```
import android.app.Fragment;
import android.os.Build;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FragmentThree extends Fragment {
    @Override

    public View onCreateView(LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState) {
        String myStr = getArguments().getString( key: "param");



        //Inflate the layout for this fragment

        return inflater.inflate(R.layout.fragment_three, container,  attachToRoot: false);
    }
}
```

Fragments

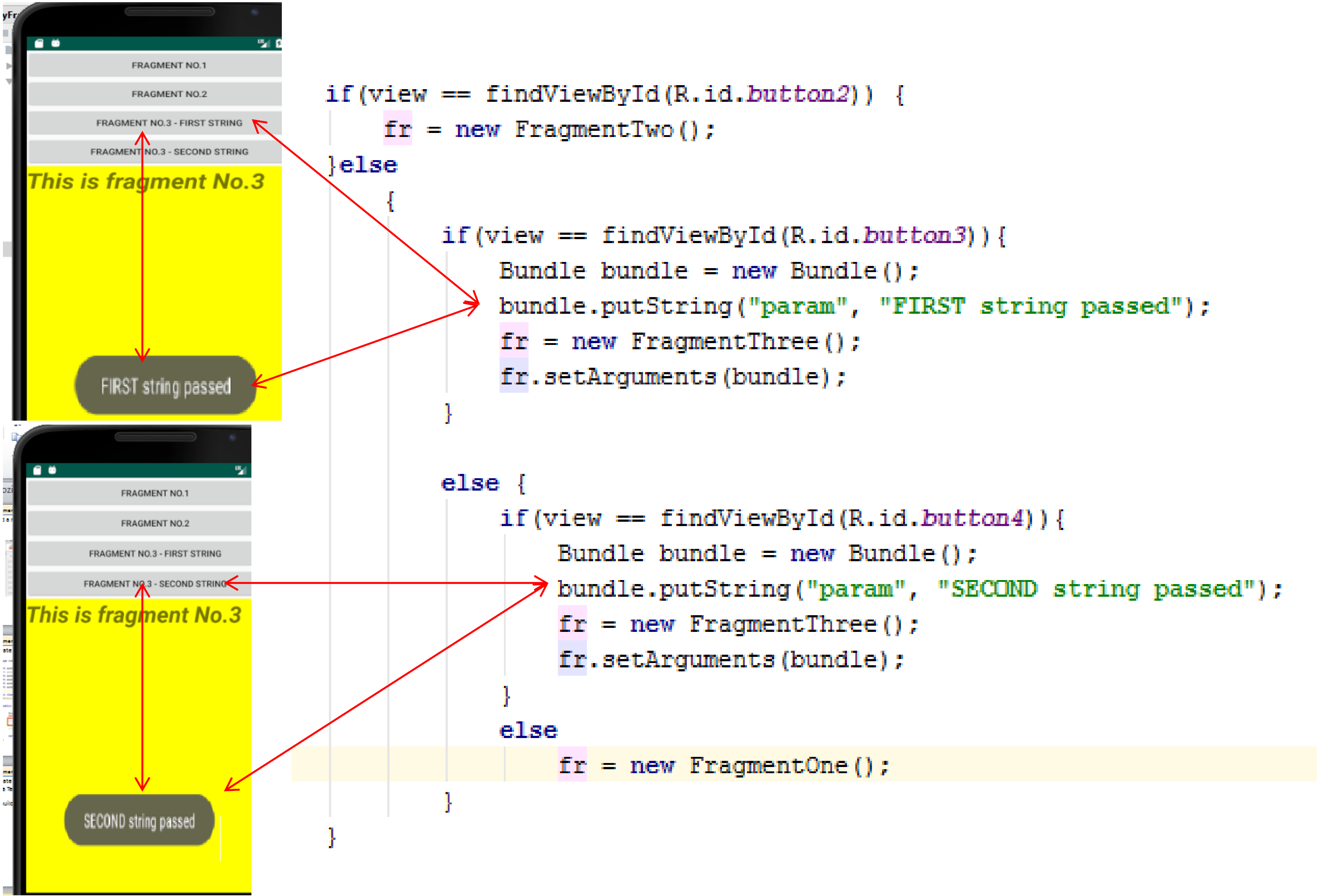
Create a layout for FragmentThree

- a TextView control must be added

Rebuild and run the app

Fragments

If you want to use a fragment for two different action (strings):



```
if(view == findViewById(R.id.button2)) {
    fr = new FragmentTwo();
}else
    {
        if(view == findViewById(R.id.button3)){
            Bundle bundle = new Bundle();
            bundle.putString("param", "FIRST string passed");
            fr = new FragmentThree();
            fr.setArguments(bundle);
        }

        else {
            if(view == findViewById(R.id.button4)){
                Bundle bundle = new Bundle();
                bundle.putString("param", "SECOND string passed");
                fr = new FragmentThree();
                fr.setArguments(bundle);
            }
            else
                fr = new FragmentOne();
        }
}
```

Fragments

You can try to call a fragment inside of another fragment (like a call of a function inside of another function)

-see the final state of the discussed project-

Source code for MainActivity, fragments and their layouts are in *MyFragments* arhive on PMD site.

Note: there is a semantic bug in *FragmentFour.java*. It is not recommended to call a fragment inside another fragment, as you saw at project presentation (see FragmentFour.java lines 24-32). I inssisted about this in my presentation. A fragment can call another fragment only if an activity is uses as an intermediate.