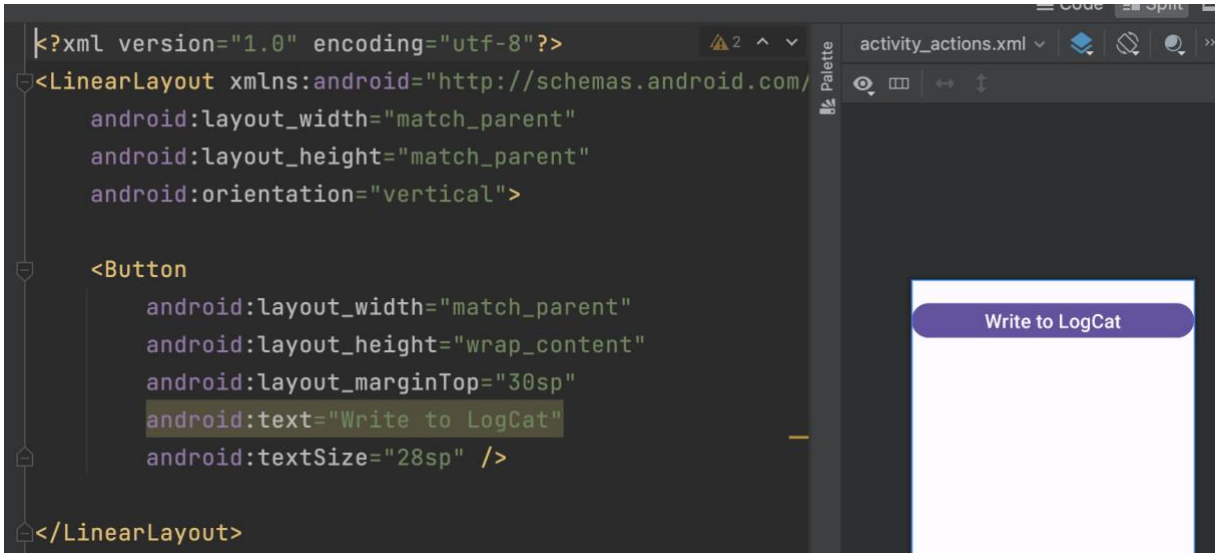
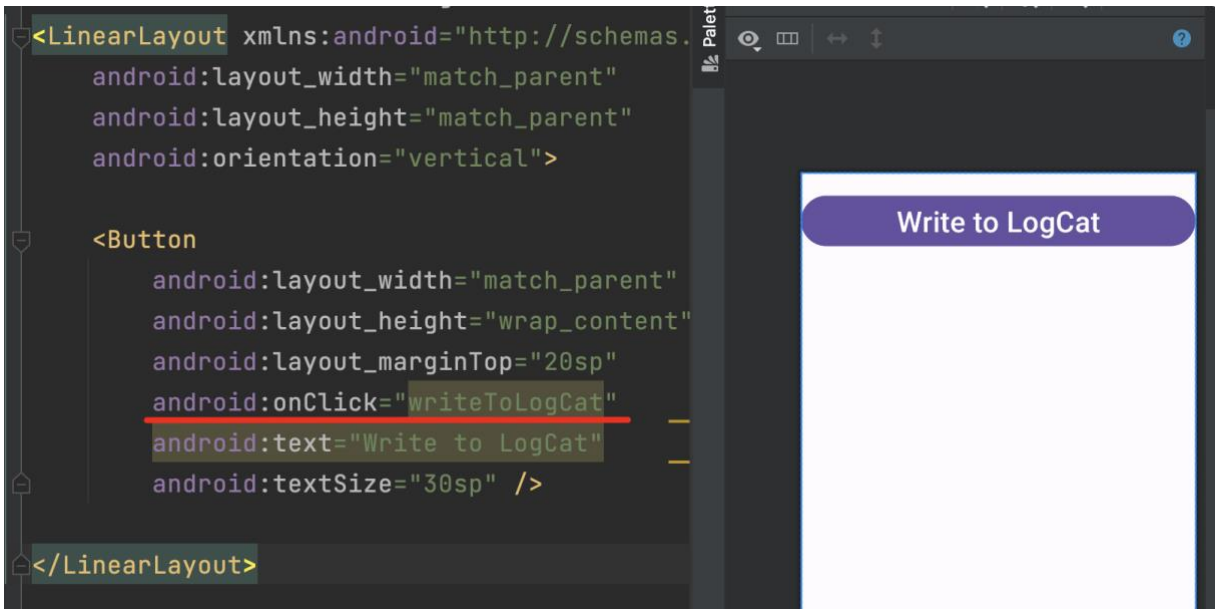
Android Apps

- The exercises given in this instruction can be completed as the continuation of the tasks done on the previous class
- Let's start working on doing actions. To do this – let's define a new layout file. Let's call it activity_actions and let's set LinearLayout as the root_element up.
- In the setContentView method of the MainActivity class, let's switch the layout to the one you have just added.
- In the XML layout definition file, let's add a new button, let's set the text informing about the action which will be performed when the button will be touched (clicked). In our first exercise we will write the message to the system log (LogCat) so let set the text to something like „Write to LogCat". So, we should get something like this



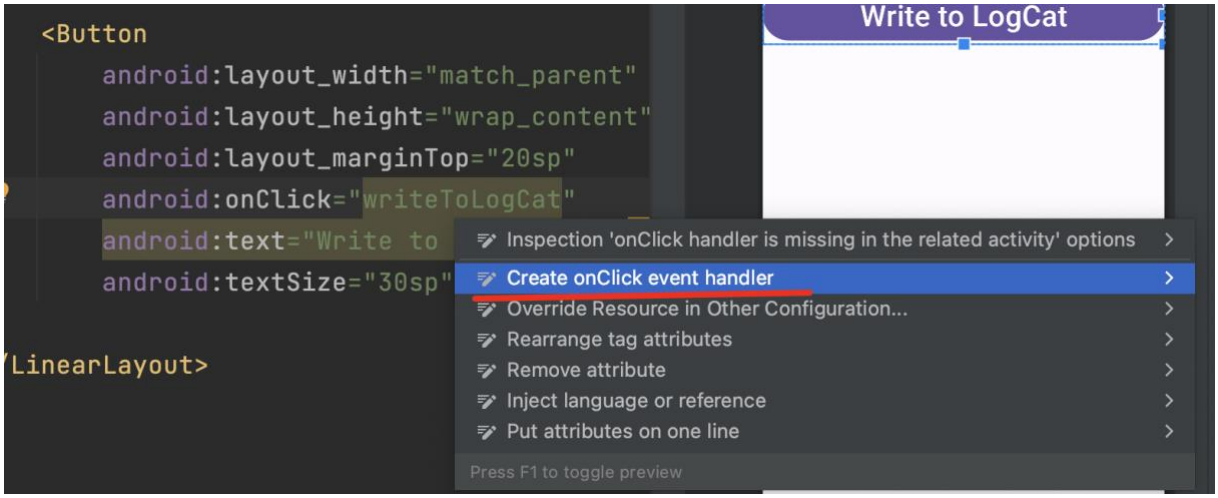

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30sp"
        android:text="Write to LogCat"
        android:textSize="28sp" />

</LinearLayout>
```

- Next, let's work on actions. First, we will do it in an old-fashioned way i.e., by setting the onClick property for the button in the layout definition, as it is done below.




```
<LinearLayout xmlns:android="http://schemas.
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20sp"
        android:onClick="writeToLogCat"
        android:text="Write to LogCat"
        android:textSize="30sp" />

</LinearLayout>
```

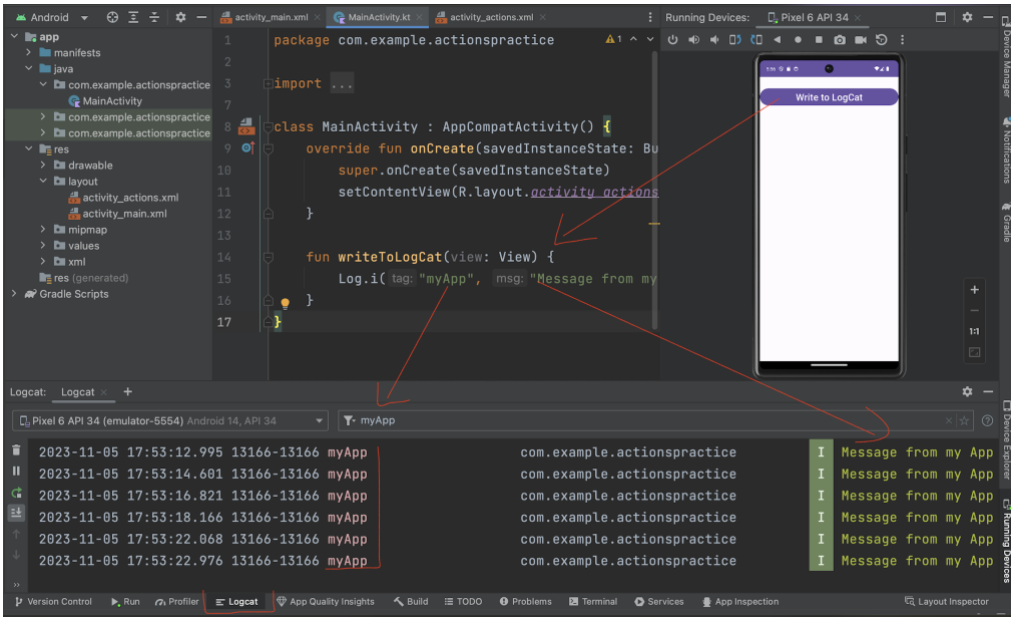
- Next, we need to generate writeToLogCat function in MainActivity class. You may do it by your own (then please remember that the function needs to take a View as a argument) or you may generate it using quick actions menu (alt + enter (or cmd + enter on macOS) being with the cursor on writeToLogCat) and we are selecting Create writeToLogCat ... in "MainActivity" from the context menu.



- Now, let's go to the MainActivity class to implement the function. In our case it is as simple as logging a message to LogCat (a tool for logging events in the system) as shown below.

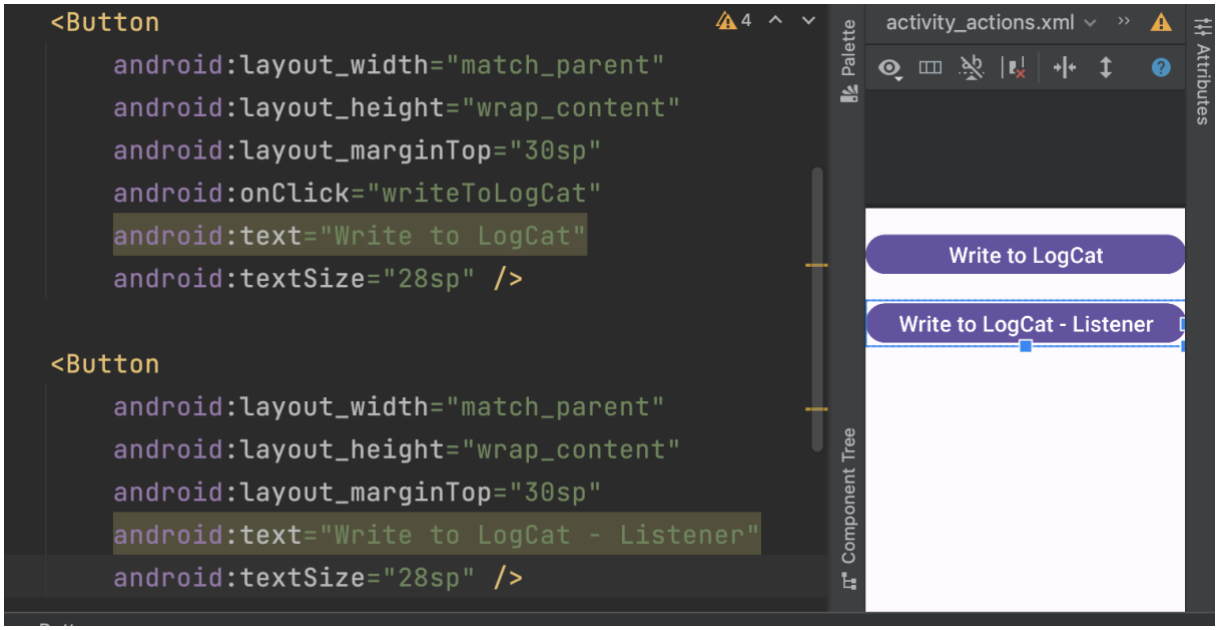
- And that's it. Let's run the app, let's open LogCat console (one of the tabs at the bottom of the Android Studio) and we expect that every time when we tap the button the log message is written to the LogCat.

- To make it easier to watch in the LogCat only the messages that we are interested in, we may filter the messages e.g., by the Tag (as below).



- Next, let's do the same but by setting onClickListener for the button directly in our class (which is a recommended approach).
- Let's add to our activity_actions layout the second button as it is done below.

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30sp"
    android:onClick="writeToLogCat"
    android:text="Write to LogCat"
    android:textSize="28sp" />

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30sp"
    android:text="Write to LogCat - Listener"
    android:textSize="28sp" />
```

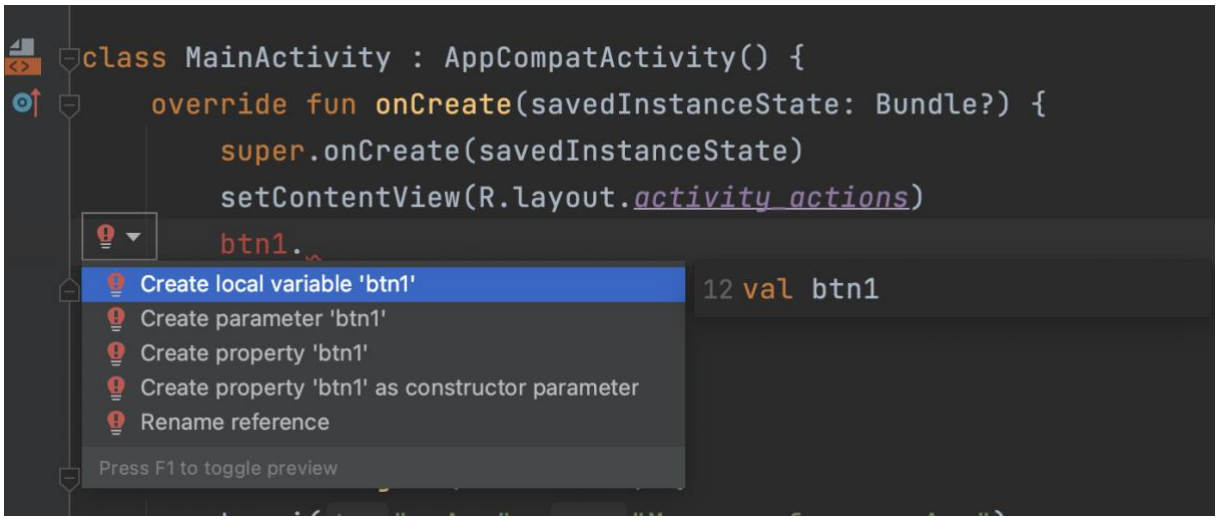


- To set the listener we need to be able to get access the button directly from the code To make it possible we need to set the id property for our button as it is shown below (your identifier will be btn1 – of course you may set it to any other name you like).

```
<Button
    android:id="@+id/btn1"
    android:layout_width="match_parent"
```

- The problem is that when you try to refer to this identifier in your code e.g. in the onCreate method of MainActivity class , it is not recognized for now.

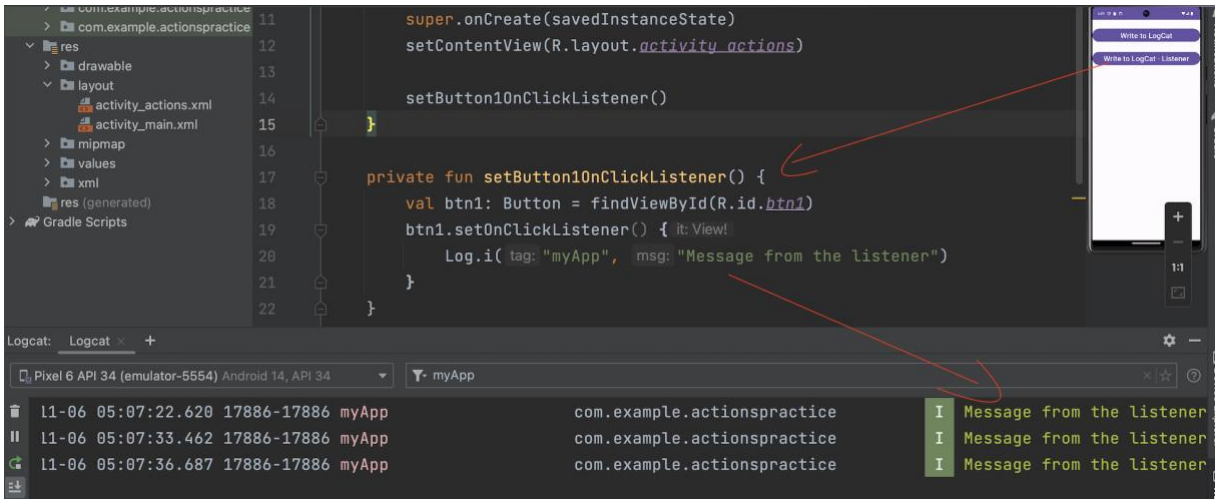
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_actions)
        btn1.
```



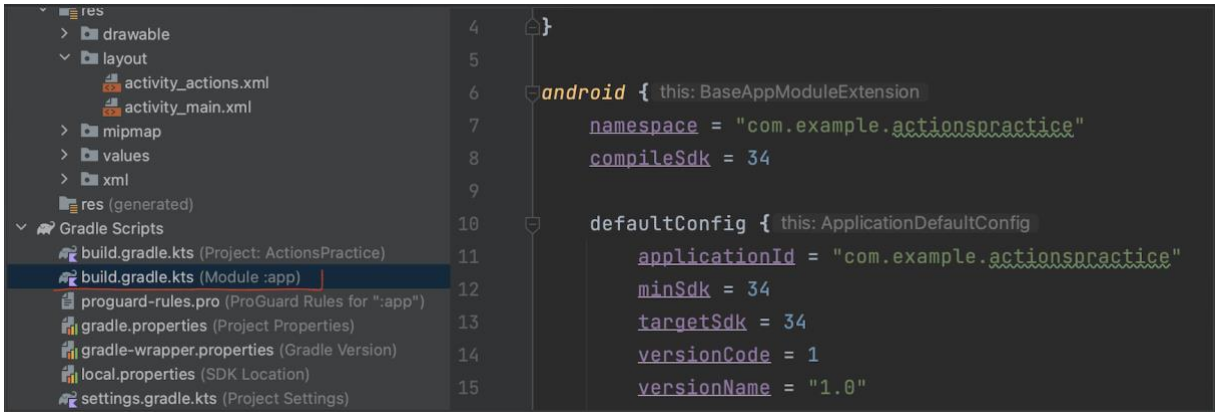
- The first solution is by using an old-fashioned (but generic) findViewById method like it is done below

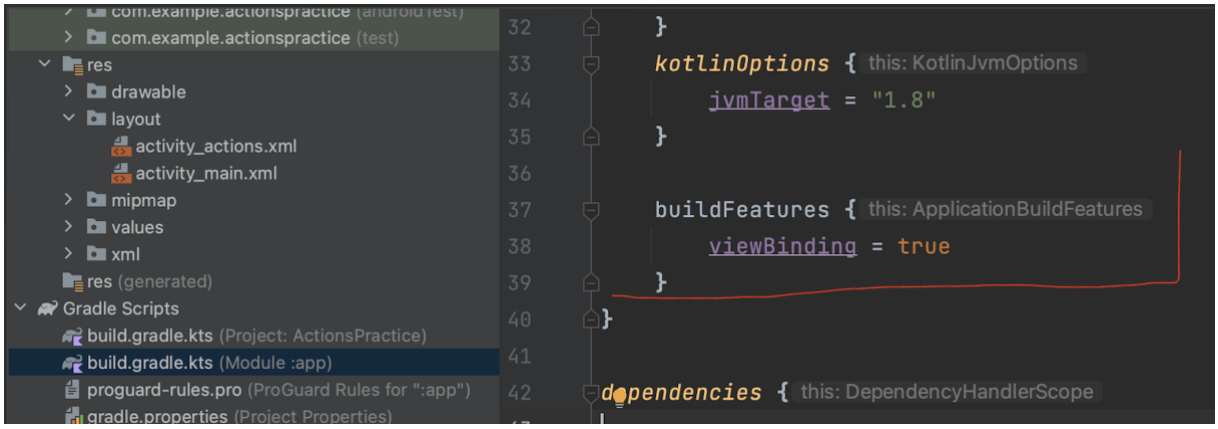
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_actions)

        setButton1OnClickListener()
    }

    private fun setButton1OnClickListener() {
        val btn1: Button = findViewById(R.id.btn1)
        btn1.setOnClickListener() { it: View!
            Log.i( tag: "myApp",  msg: "Message from the listener")
        }
    }
```

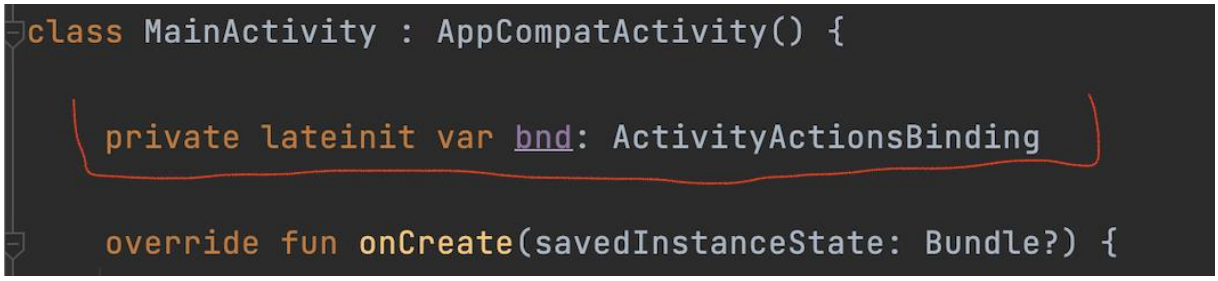
- And while tested you should see the expected result, i.e.:



- The better solution is to use view binding mechanism that allows you the get access your UI elements directly from the code (i.e. you don't have to call findViewById method to "find" them)
- To enable view binding let's open our Gradle script i.e.

- 

- And in android section we need to turn on viewBinding mechanism for our app as it is shown below:
- 

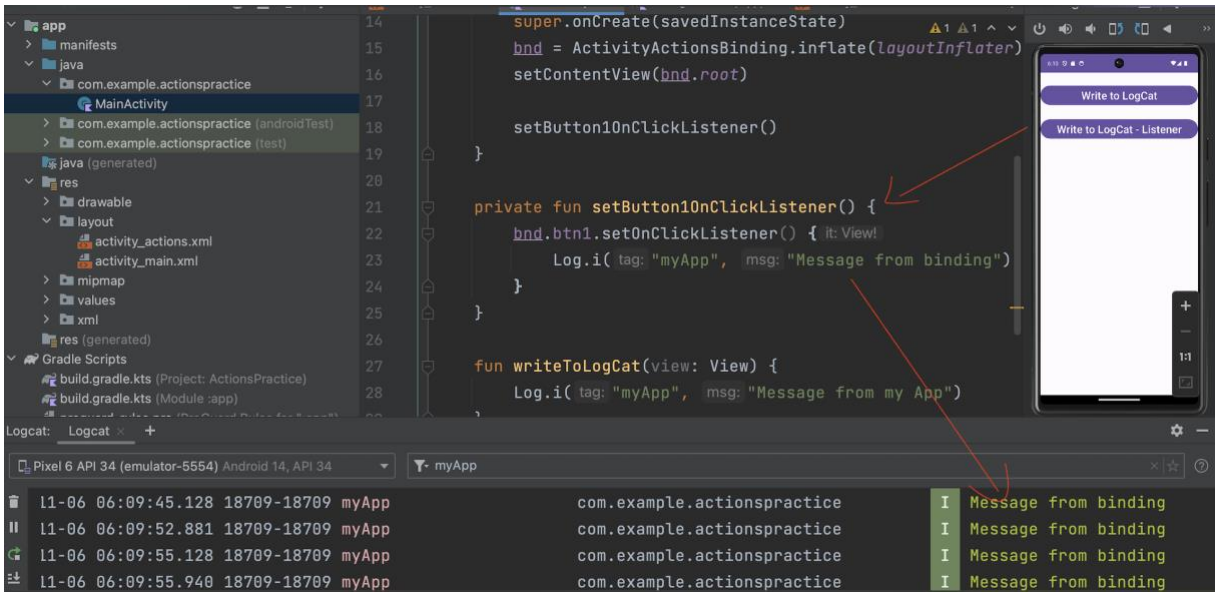
- After that (and syncing the Gradle), we need a binding field in your class:
- 

- And now we may refactor a bit our onCreate method as below:
- 

- And from now on we may refer our UI elements using a binding variable. So we may change a bit our onClickListener this way:

```
private fun setButton1OnClickListener() {
    bnd.btn1.setOnClickListener() { it: View!
        Log.i( tag: "myApp",  msg: "Message from binding")
    }
}
```

- And, obviously, while testing we should see the result as expected, i.e.:



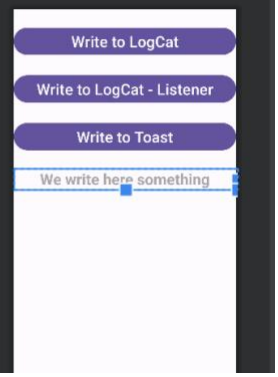
- Next, let's try to write something in the application itself. We start with showing the message on the toast.
- So, let's add the next button (btn2) to activity_action and let's define onClickListener as we just did it for btn1.
- Next, in our listener, let's create and show the Toast as it is shown below.

```
private fun setButtonsOnClickListeners() {
    bnd.apply { this: ActivityActionsBinding
        btn1.setOnClickListener() { it: View!
            Log.i( tag: "myApp",  msg: "Message from binding")
        }
        btn2.setOnClickListener { it: View!
            Toast.makeText( context: this@MainActivity,  text: "Message from my App",
                Toast.LENGTH_LONG)
                .show()
        }
    }
}
```

- Next, let's try to write anything directly to UI elements of our app.

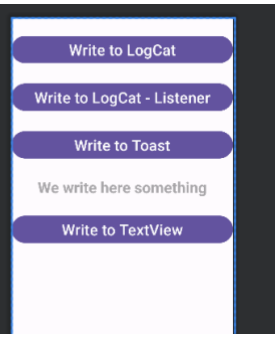
- So, let's add the TextView element to our actions_layout layout, let's give it the @+id/tvSampleMessage id, and of course you may "embellish" it a bit. For me, the definition of this TV is as follows:

```
<TextView
    android:id="@+id/tvSampleMessage"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30sp"
    android:textSize="28sp"
    android:hint="We write here something"
    android:gravity="center"
    android:textStyle="bold"
    />
```



- Then, let's add the new button to our layout (similarly to the previous buttons)

```
<Button
    android:id="@+id/btn3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30sp"
    android:text="Write to TextView"
    android:textSize="28sp" />
```

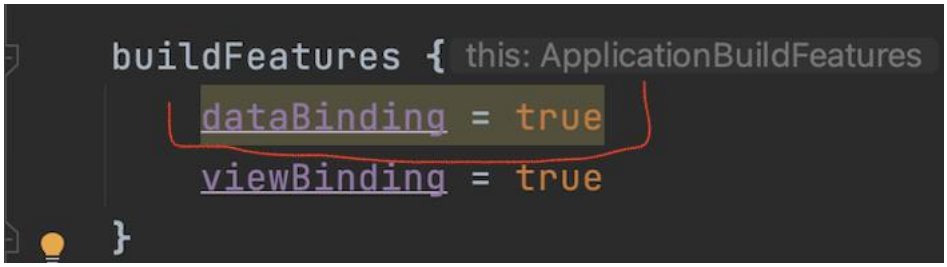


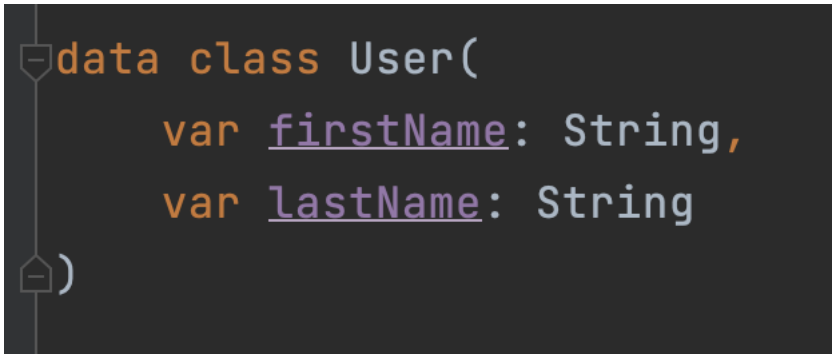
- And, obviously let's create onClickListener as we did it before. But this time, let's set the text property of our tvSampleMessage TextView e.g. as it is shown below:

```
btn3.setOnClickListener{ it: View!
    tvSampleMessage.text = "I love this game"
}
```

- Let's run the app and, as expected, while the last button is clicked the content of our TextView should change.

- 

- In the last exercise we set the textview content explicitly (like tvSampleMessage.text). The better option is to use databinding to let the developer to focus on working with the "busines logic" and the UI elements will be updated "automatically". Let's try.
- First we need to turn on data binding mechanism in our Gradle configuration:
- 

- Next, let's imagine that in our logic we are working with the user class including his/her first and last name.
- So, let's add to our Kotlin file a simple User data class as it is shown below (It may be done in existing kt file, or you may create the new one).
- 

- Now let's add the user field in MainActivity Class
- 

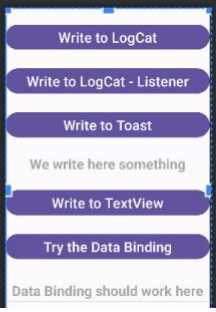
- To bind user object with our layout we need to define a data model variable in our layout. To do that we need to use <data> tag in our layout file which is available only for a generic (not for Linear, Constraint etc.) layout.
- So, we need to wrap up our LinearLayout that we are working with, with a generic layout tag as it is done below:

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

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        tools:context=".MainActivity">

        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

- Next Inside the (generic) layout we may define the data model binding as it is done below:

```
<layout 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">
    <data>
        <variable name="user" type="com.example.actionspractice.User"/>
    </data>
```

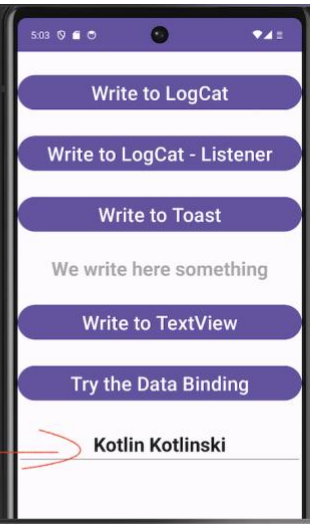
- And now, we may “configure” the EditText text property to use user model data, e.g. as it is shown below:

-
```
    <EditText
        android:id="@+id/etDataBinding"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30sp"
        android:gravity="center"
        android:hint="Data Binding should work here"
        android:textSize="28sp"
        android:text='@{user.firstName + " " + user.lastName}'
        android:textStyle="bold" />

</LinearLayout>
```


- Finally, let's initialize user field e.g. in onCreate method and let's set the binding.user model data to just initialized user object, as it is shown below:

-
```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        bnd = ActivityActionsBinding.inflate(layoutInflater)
        setContentView(bnd.root)
        user = User("Kotlin", "Kotlinski")
        bnd.user=user
        setButtonsOnClickListeners()
```

- And, generally we may now test the app and we should see the following

-
```
data class User(
    var firstName: String, var lastName: String
)

class MainActivity : AppCompatActivity() {

    private lateinit var bnd: ActivityActionsBinding
    private lateinit var user: User

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        bnd = ActivityActionsBinding.inflate(layoutInflater)
        setContentView(bnd.root)
        user = User("Kotlin", "Kotlinski")
        bnd.user=user
        setButtonsOnClickListeners()
```


-

- Next let's add the next button to our layout file where we may update our user object and reassign the updated user object to user model data as it is done below:

```
btn4.setOnClickListener() {
    user?.firstName = "Radek"
    user?.lastName = "Radlinski"
    bnd.user = user
}
```

- And the solution should now work as expected i.e.

- Now let's try to turn our one-way data binding into the tow-ways one.
- First, we need to slightly change our User class which needs to inherit from ViewModel and the data inside must be of MutableLiveData<> type. So after that changes my User class looks as follows:

```
class User() : ViewModel() {
    var firstName = MutableLiveData<String>()
    var lastName = MutableLiveData<String>()
}
```

- And now we need to make some changes in our onCreate method as shown below:

```
class ActionsActivity : AppCompatActivity() {

    private lateinit var bnd: ActivityActionsBinding
    private lateinit var user: User

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        bnd = ActivityActionsBinding.inflate(layoutInflater)
        setContentView(bnd.root)
        user = ViewModelProvider( owner: this)[User::class.java]
        user.firstName = MutableLiveData( value: "Mietek")
        user.lastName = MutableLiveData( value: "Mietkowski")
        bnd.user = user
        bnd.lifecycleOwner = this
        setButtonsOnClickListeners()
    }
```

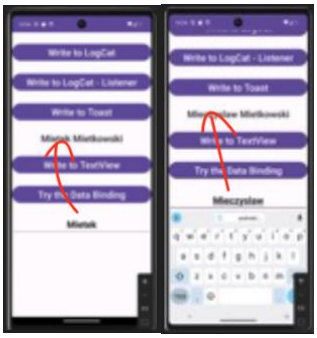
- Finally, some small changes in our layout file, i.e. let's "sync" the content of our previously defined TextView content with the values displayed/provided by the user to EditText. So first, let's set up the text property of our TectView element, as shown below:

```
<TextView
    android:id="@+id/tvSampleMessage"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30sp"
    android:gravity="center"
    android:hint="We write here something"
    android:textSize="28sp"
    android:text='@{user.firstName + " " + user.lastName}'
    android:textStyle="bold" />
```

- And now, let's imagine that our EditText element's role is to edit the user first name, so we need to change our EditText text property as shown below:

```
<EditText
    android:id="@+id/etDataBinding"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30sp"
    android:gravity="center"
    android:hint="Data Binding should work here"
    android:textSize="28sp"
    android:text="@={user.firstName}"
    android:textStyle="bold" />
```

- And that's it. Let's run and test our app:



- The next actions that we would like to perform is navigating between different activities and fragments in our app.
- Let's make some "cleaning" in our project. I.e. let's rename MainActivity file and class to ActionsActivity
- Let's add to the project a new class MainActivity and let's implement it in a basic form as below:

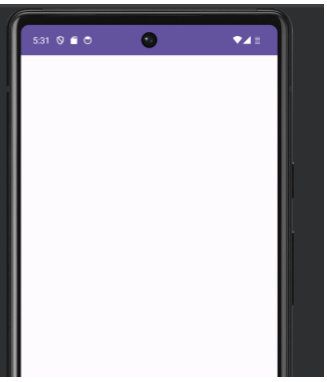
```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

- Next let's check/update the manifest file to use MainActivity as the entry point/main activity for our app:

```
tools:targetApi="31">
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

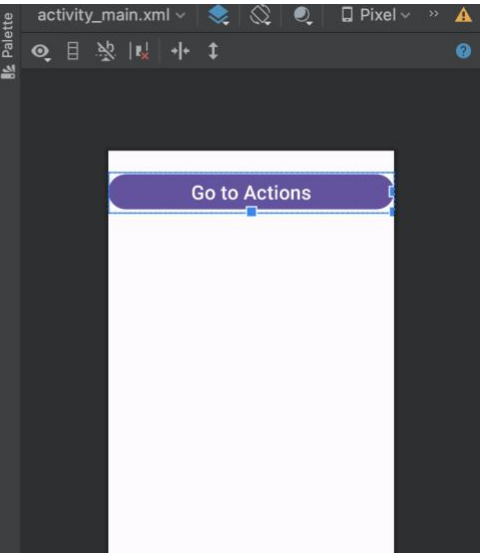
- And let's run the app. The result should be as follows:

```
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity


class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```



- Now, let's replace the main layout in our main_activity.xml to the LinearActivity and let's add there a simple Button as it is shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.co
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btnNav1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30sp"
        android:text="Go to Actions"
        android:textSize="28sp" />

</LinearLayout>
```



- Now, let's go back to the MainActivity class, and let's define the view binding there as we did it before, so as:

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding:ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
```

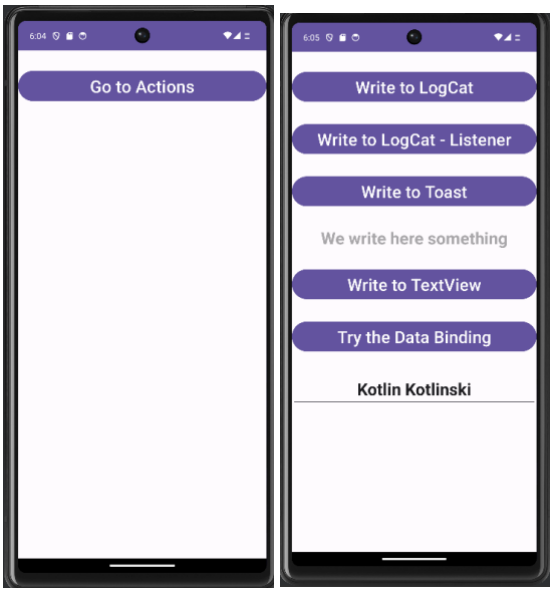
- Let’s run and test the app. The result should be as below:

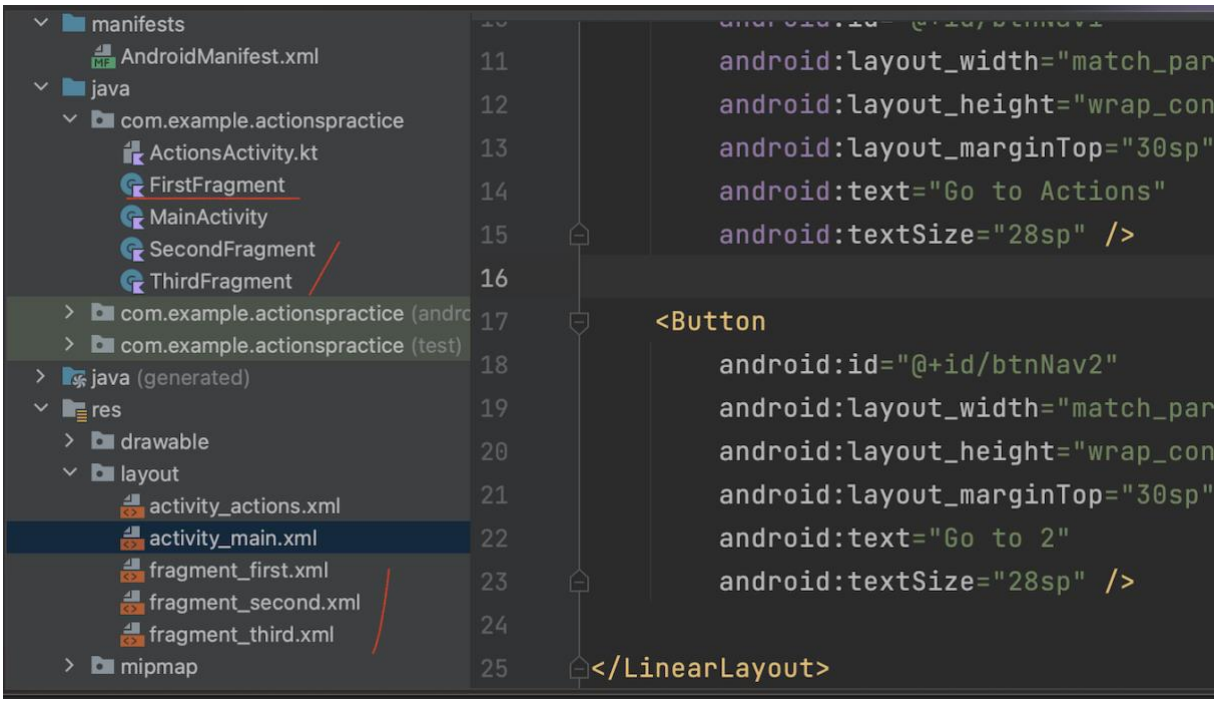
- Now let’s createthe btnNav1 onClickListener and let’s try to navigate from MainActivity to ActionsActivity. In the first step we will do it in an “old-fashioned” way i.e. using Intents and startActivity method. So let’s make the implementation of our btnNav1 onClickListener this way:

```
    private fun setButtonsOnClickListeners() {
        binding.btnNav1.setOnClickListener { it: View!
            var intent = Intent( packageContext: this, ActionsActivity::class.java)
            startActivity(intent)
        }
    }
```

- To make the activity to be “findable” by the intent we need to declare it in the manifest file as below:

```
        </activity>
        <activity
            android:name=".ActionsActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
```

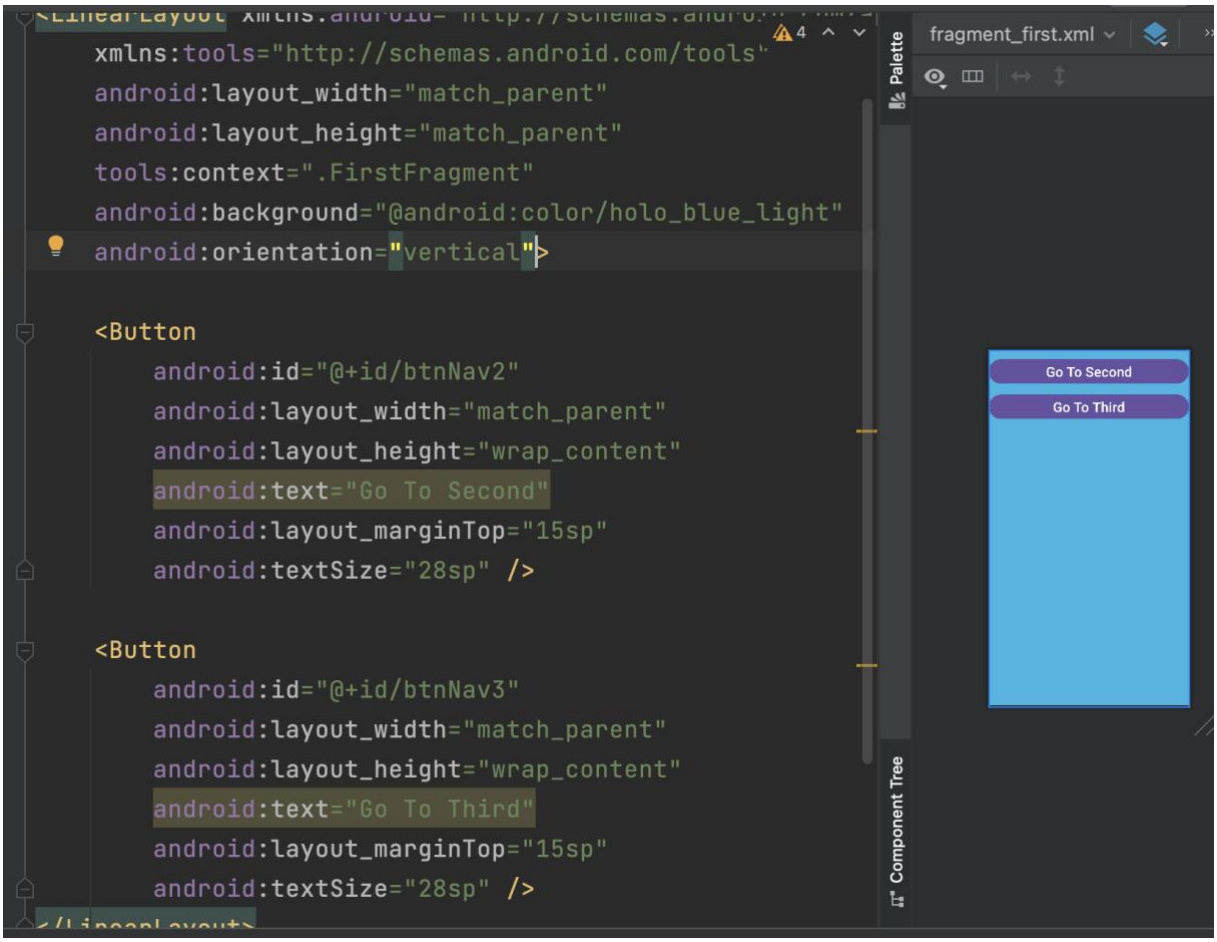
- And now, we may run and test the app and it should work as expected, i.e. by tapping the “Go to actions” Button we should be “redirected” to our ActionsActivity:
-] 

- Now let’s define the navigation in a more up-to-date way i.e. using Fragments and the Navigation Component.
- So, let’s add three blank fragments to our projects and let’s call them like FirstFragment, SecondFragment and ThirdFragment
- 

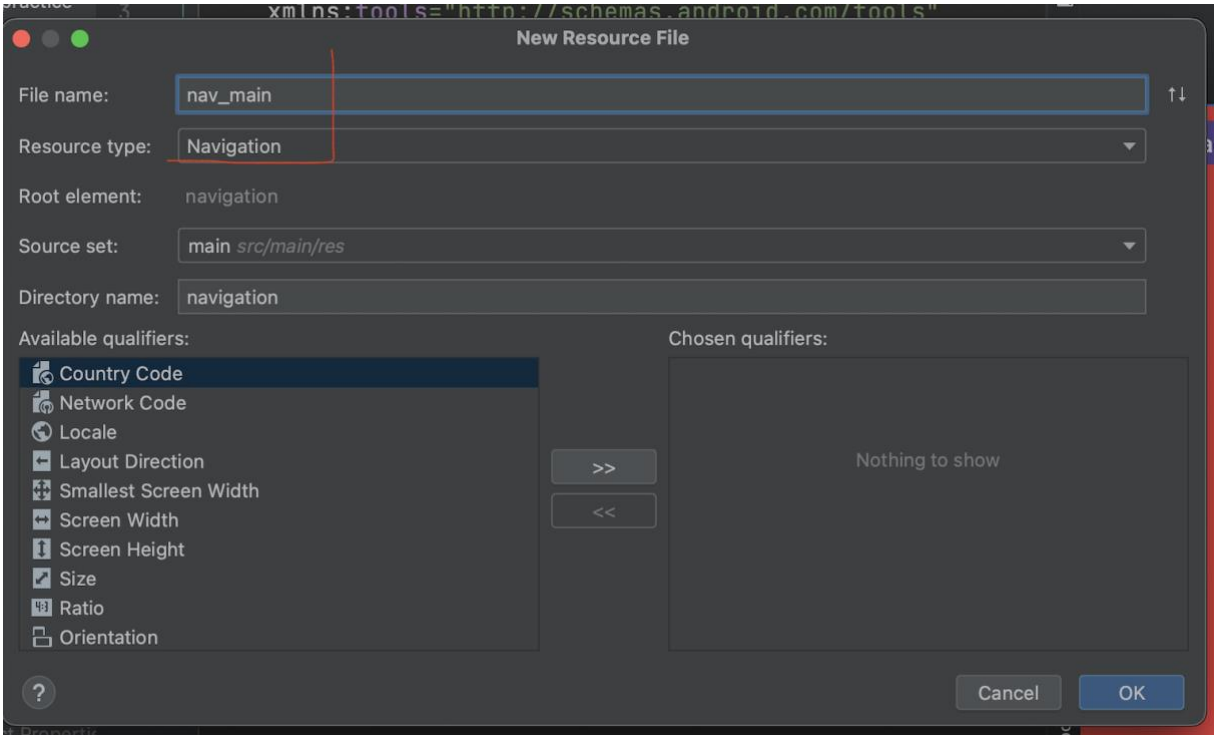
- Now, in all three fragment classes let’s remove all but onCreateView methods:

```
class FirstFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_first, container, attachToRoot: false)
    }
}
```

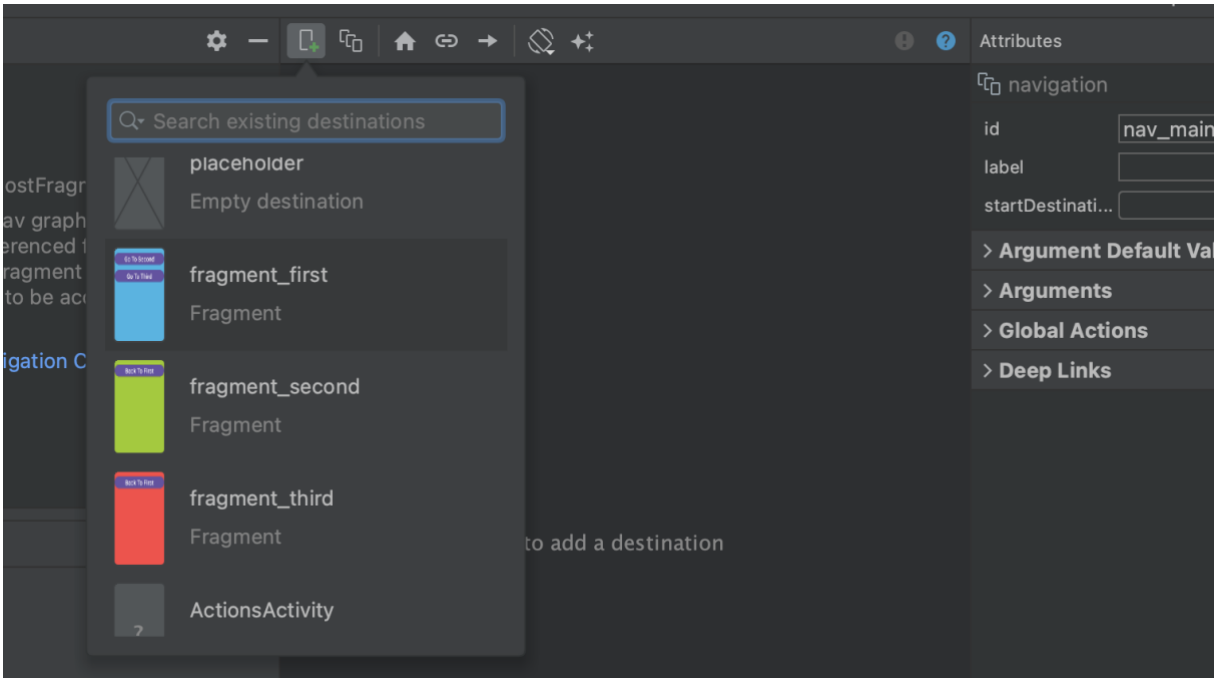
- Now, let's define fragment_first.xml layout as below:



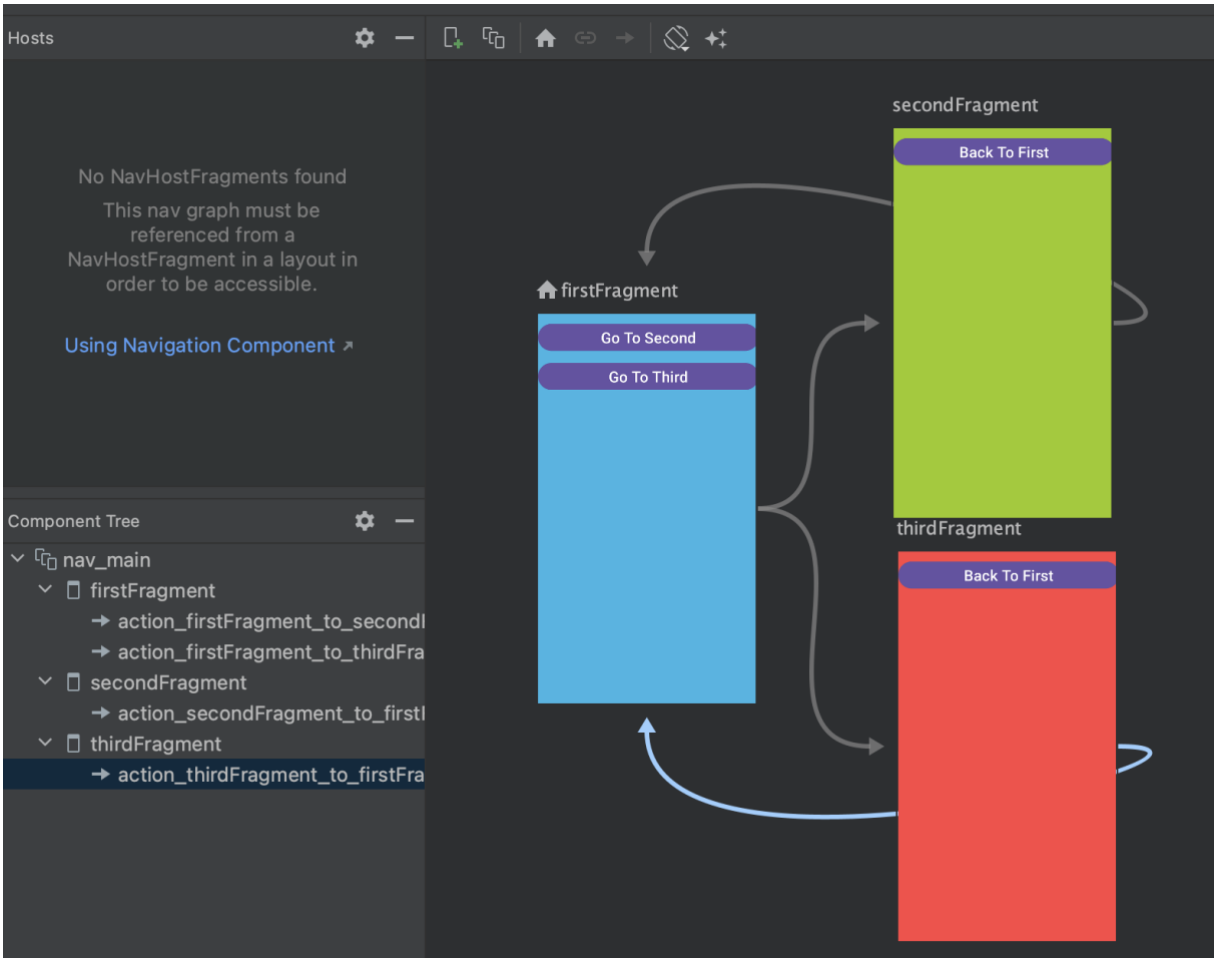
- Now, let's add the navigation resource to our project. We may do it by right clicking on the res-> new -> Android Resource File and in the dialog box let's choose the navigation as type and nav_main as the name:



- Now let’s add our all three fragments to our navigation:



- And let’s define transitions (actions) between fragments as below:



- Now, in the FirstFragment class let's define the view binding by analogy as we did it before:

```
class FirstFragment : Fragment() {

    private lateinit var binding: FragmentFirstBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentFirstBinding.inflate(inflater,container, attachToParent: false)
        return binding.root
    }
}
```

- And finally, let's add onViewCreated overrided method, where we may set the buttons' onClickListeners as it is shown below:

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    binding = FragmentFirstBinding.inflate(inflater,container, attachToParent: false)
    return binding.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding.btnNav2.setOnClickListener{ it: View!

    }
}
```

```
private lateinit var binding: FragmentFirstBinding

override fun onCreateView(
    inflater: LayoutInflater, cont
    savedInstanceState: Bundle?
): View? {
    binding = FragmentFirstBinding
    return binding.root
}

override fun onViewCreated(view: V
    super.onViewCreated(view, save
    binding.btnNav2.setOnClickList
        findNavController().navigate(R.id.)
    }
}
```

```
btnNav2
firstFragment
action_firstFragment_to_thirdFragment
action_firstFragment_to_secondFragment
action_thirdFragment_to_firstFragment
action_secondFragment_to_firstFragment
btn1
btn2
btn3
btn4
btnNav3
Press ⏎ to insert, ⇥ to replace Next Tip
```

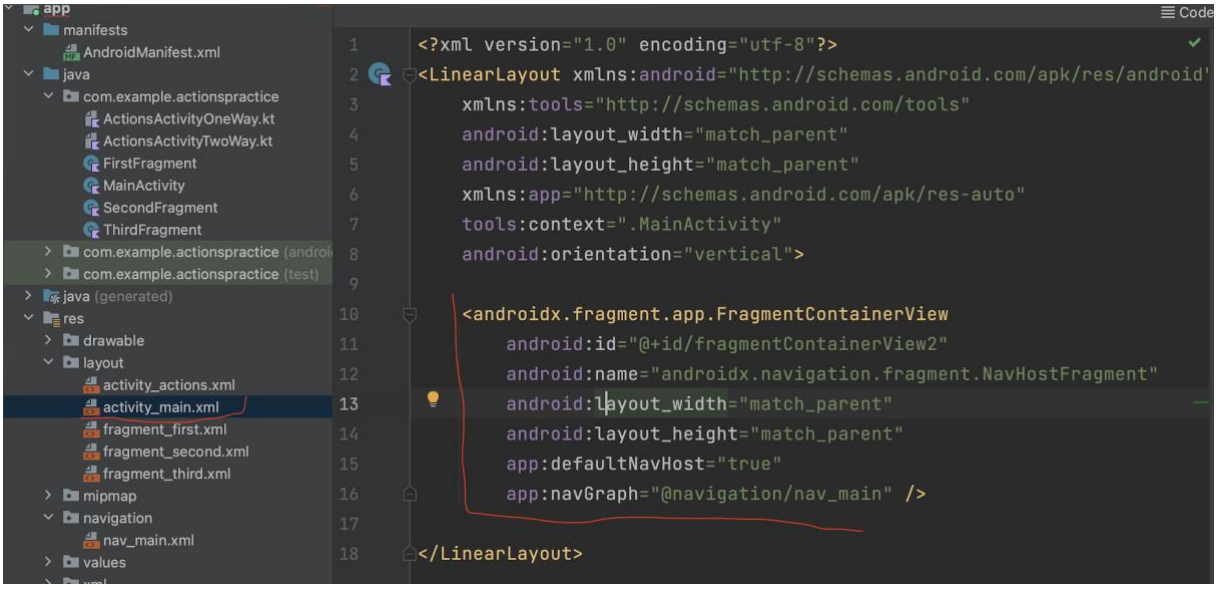
- So after defining the two listeners in FirstFragment class it should look as follows:

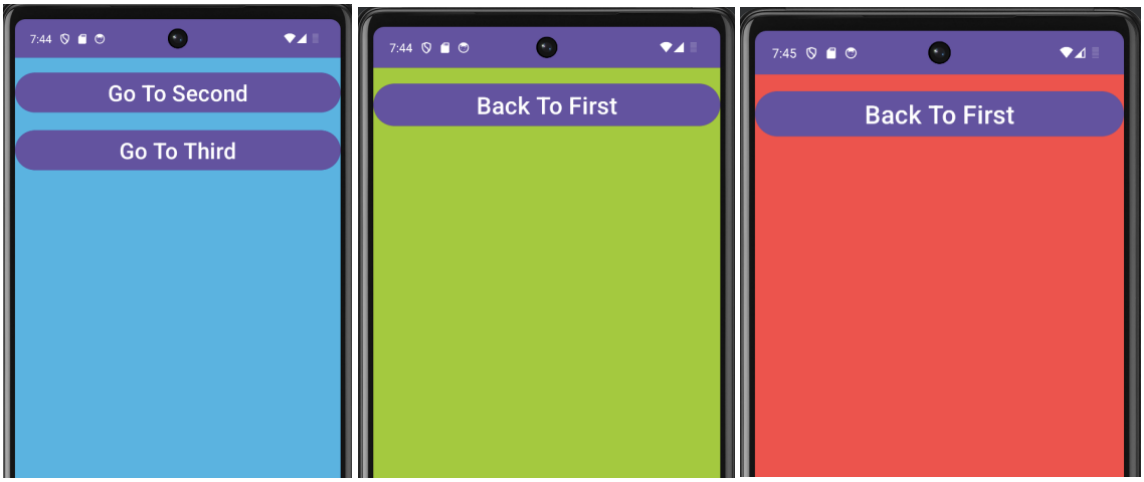
```
class FirstFragment : Fragment() {

    private lateinit var binding: FragmentFirstBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentFirstBinding.inflate(inflater,container, attachToParent: false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.btnNav2.setOnClickListener{ it: View!
            findNavController().navigate(R.id.action_firstFragment_to_secondFragment)
        }

        binding.btnNav3.setOnClickListener{ it: View!
            findNavController().navigate(R.id.action_firstFragment_to_thirdFragment)
        }
    }
}
```

- And the two other fragment classes should look as follows:

```
class SecondFragment : Fragment() {

    private lateinit var binding: FragmentSecondBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentSecondBinding.inflate(inflater,container, attachToParent: false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.btnNav4.setOnClickListener{ it: View!
            findNavController().navigate(R.id.action_secondFragment_to_firstFragment)
        }
    }
}
```

- And finally let’s change the main_activity layout to include our fragments as it is shown below:

- 

- And, that's all. So, let's run the app and the navigation should work as expected:
- 

- As for practicing add the navigation from the first fragment to ActionsActivity
- When done please upload the screenshots of your code and app as the solutions of Actions Intro Task on UPEL platform (Done During The Class and Full Solution)
- As for your homework/independent work please create the "Calculators" app where the numerical and BMI calculators should be provided as the available functionality. You may use the layout that you defined on the previous class. It is up to you how your app will be implemented but please use the mechanisms that we learned and trained during the class.
- Hints for BMI calculator
- $$BMI = \frac{\text{masa w kg}}{(\text{wzrost w m})^2}$$

- 

- Sample Graphics may be found here: https://shorturl.at/GJNQ3