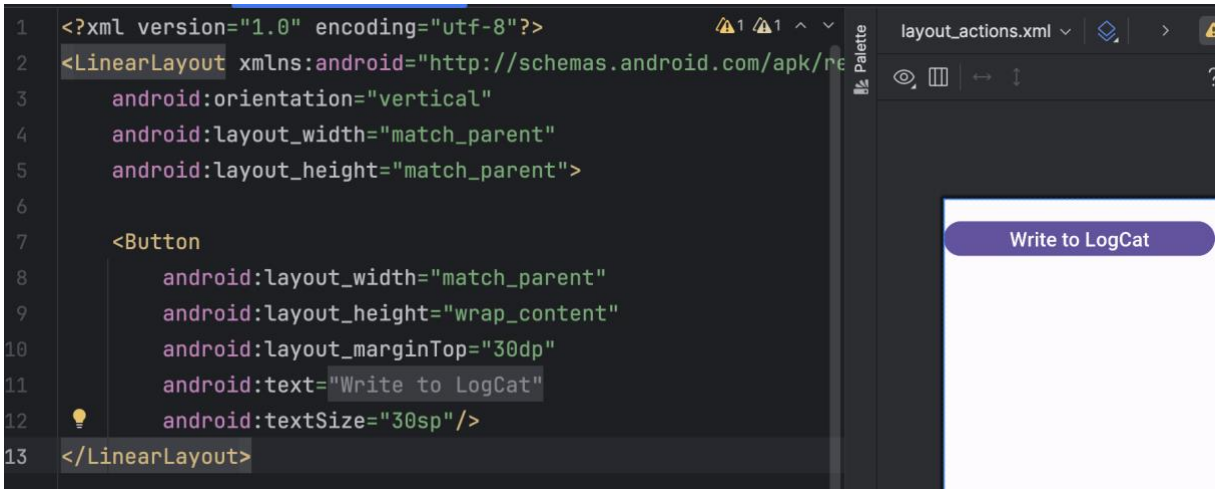
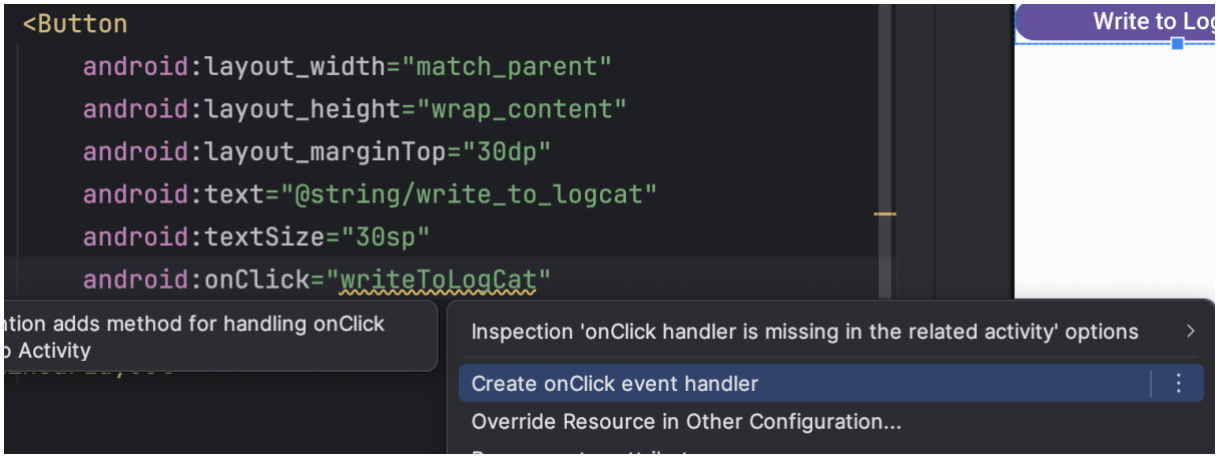
Android Apps – Actions, View- and DataBinding with static layouts

- 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
- 

- 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.

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

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:text="@string/write_to_logcat"
        android:textSize="30sp"
        android:onClick="writeToLogCat"
        />
</LinearLayout>
```

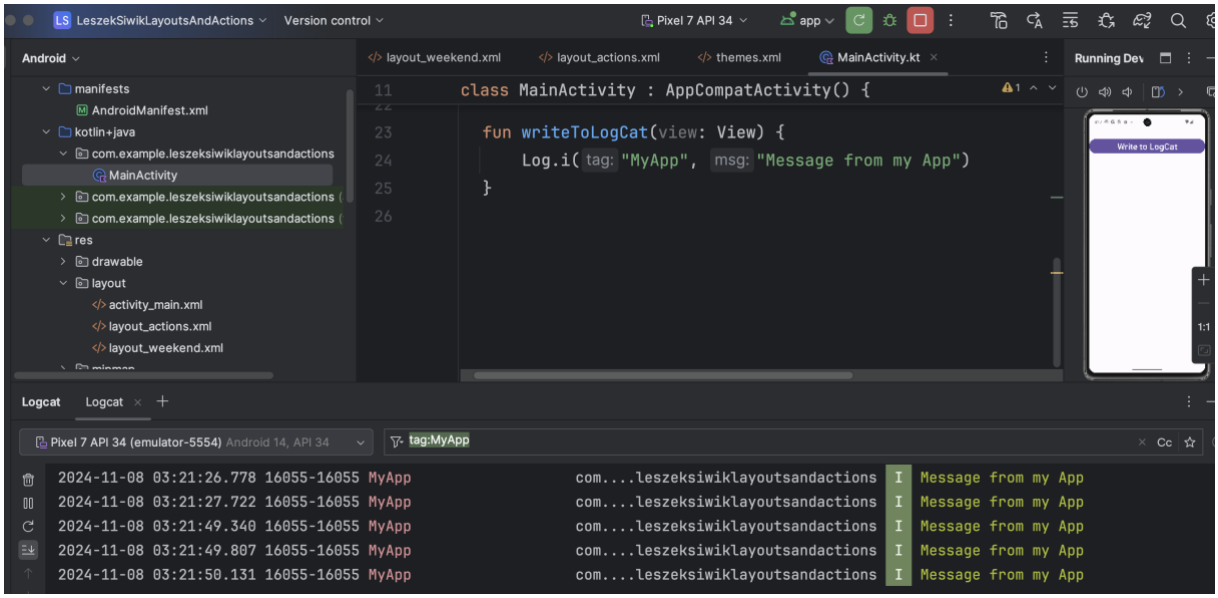
- Next, we need to generate writeToLogCat function in MainActivity class. The easiest way is pressing just (alt + enter being with the cursor on writeToLogCat then Create onClick even handler from the context menu.

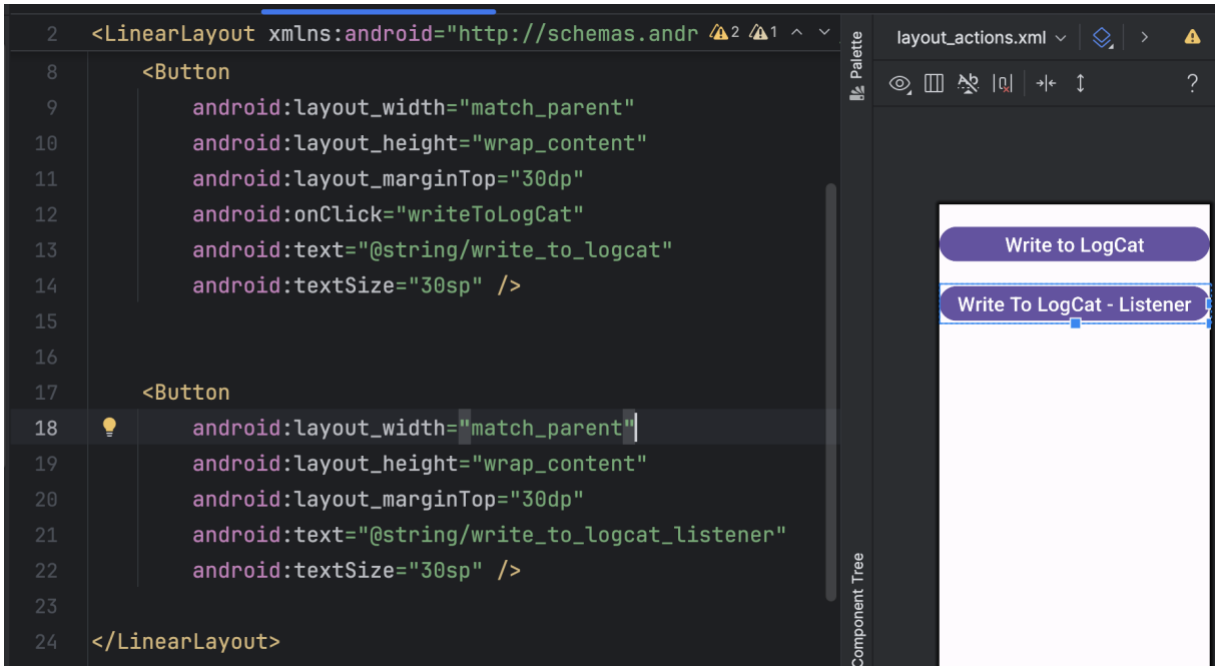


- Now, let's go to the MainActivity class to implement this method. In our case it is as simple as logging a message to LogCat as shown below.

```
fun writeToLogCat(view: View) {
    Log.i( tag: "MyApp",  msg: "Message from my App")
}
```

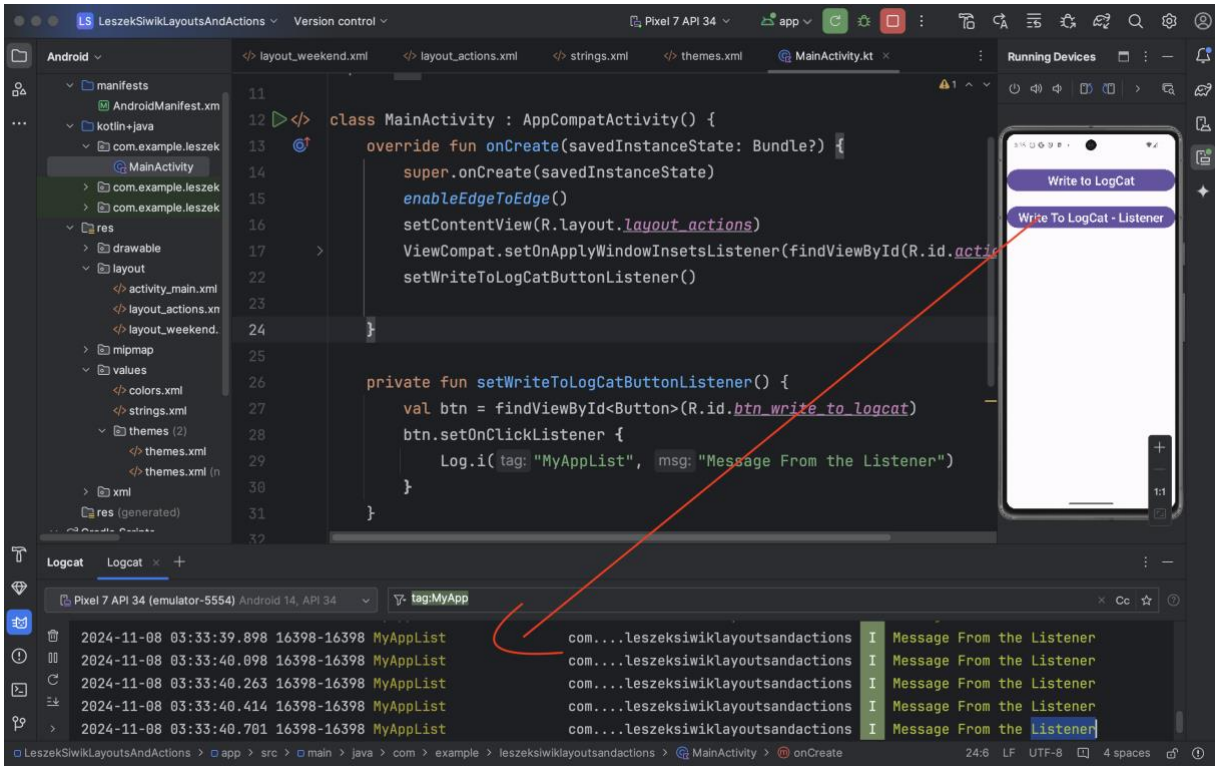
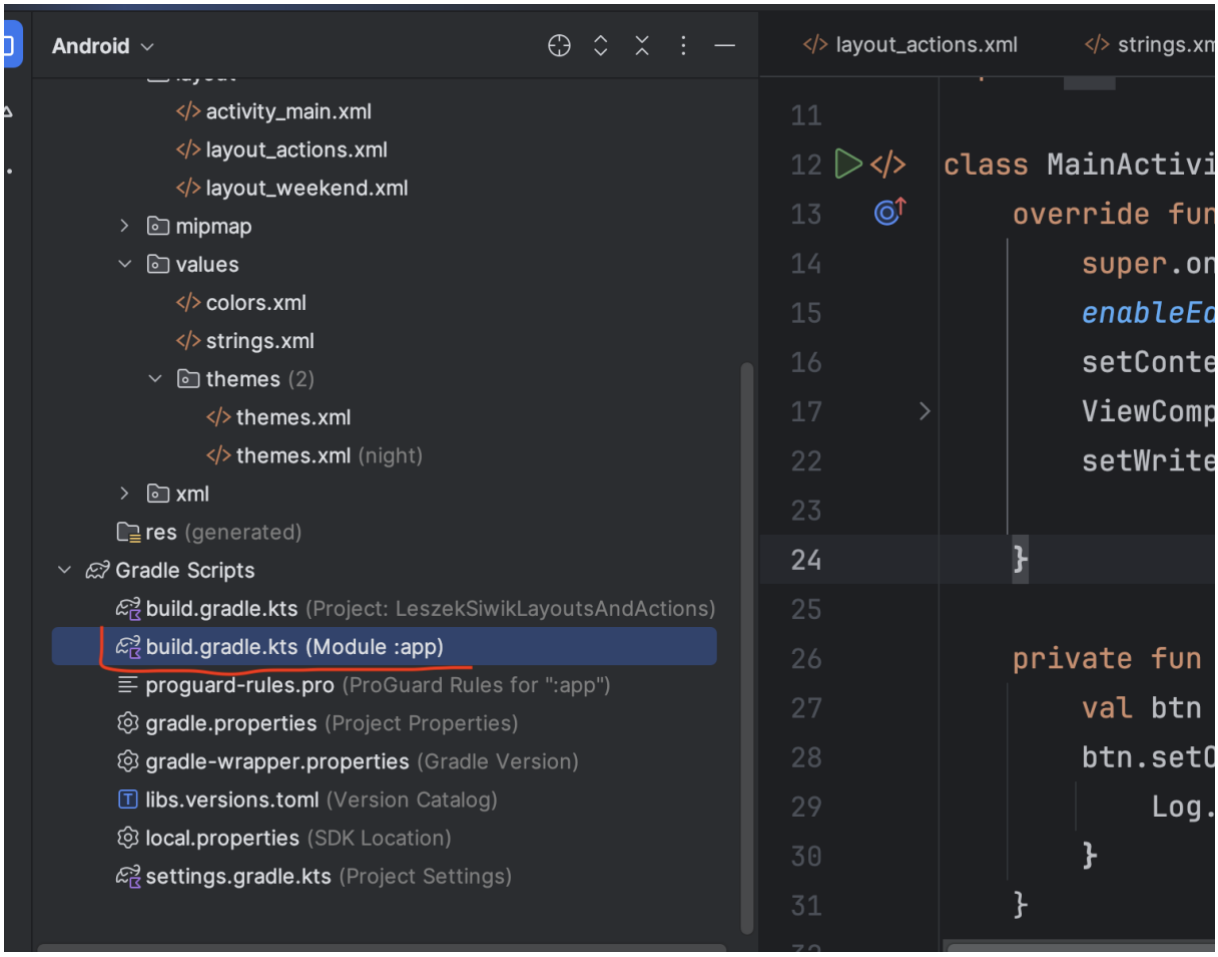
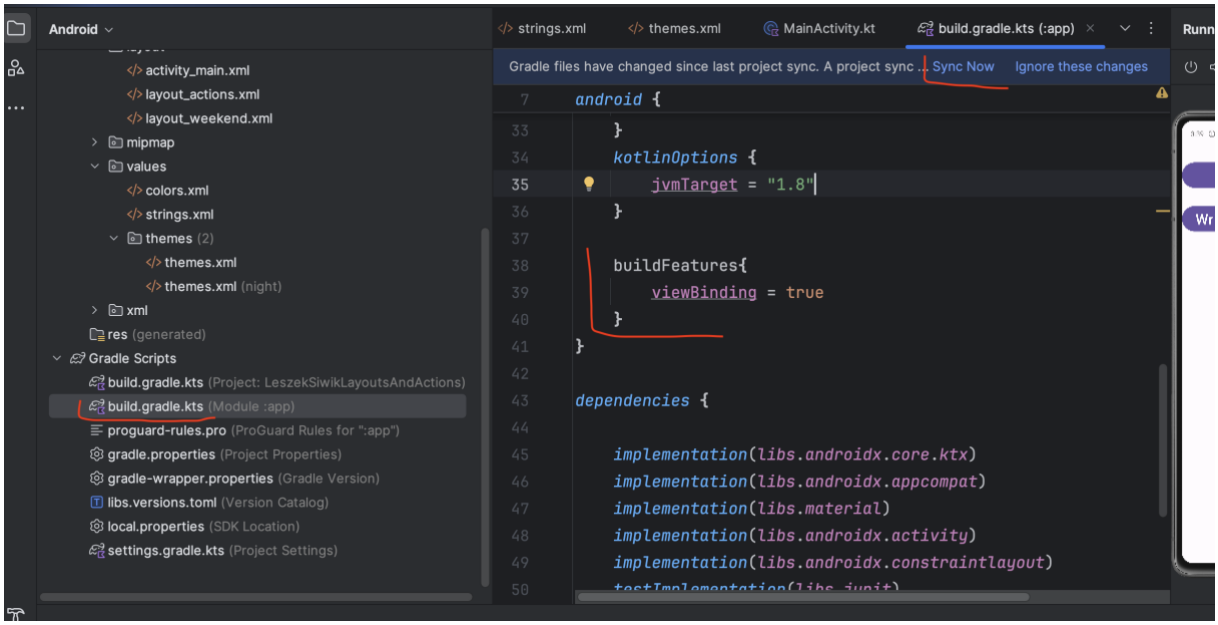
- And that's it. Let's run the app, let's open LogCat console and we should see the log message while tapping on the button.

- 

- 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.
- 

- 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

- ```
  <Button
      android:id="@+id/btn_write_to_logcat"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="30dp"
      android:text="@string/write_to_logcat_listener"
      android:textSize="30sp" />
  ```
  .
- Now we would like to set onClickListener for this button in MainActivity class. 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.
- ```
  .ass MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.layout_actions)
        btn
  ```
- 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)
          enableEdgeToEdge()
          setContentView(R.layout.layout_actions)
          ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.actic
          setWriteToLogCatButtonListener()

      }

      private fun setWriteToLogCatButtonListener() {
          val btn = findViewById<Button>(R.id.btn_write_to_logcat)
          btn.setOnClickListener {
              Log.i( tag: "MyAppList",  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 (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:

```
class MainActivity : AppCompatActivity() {

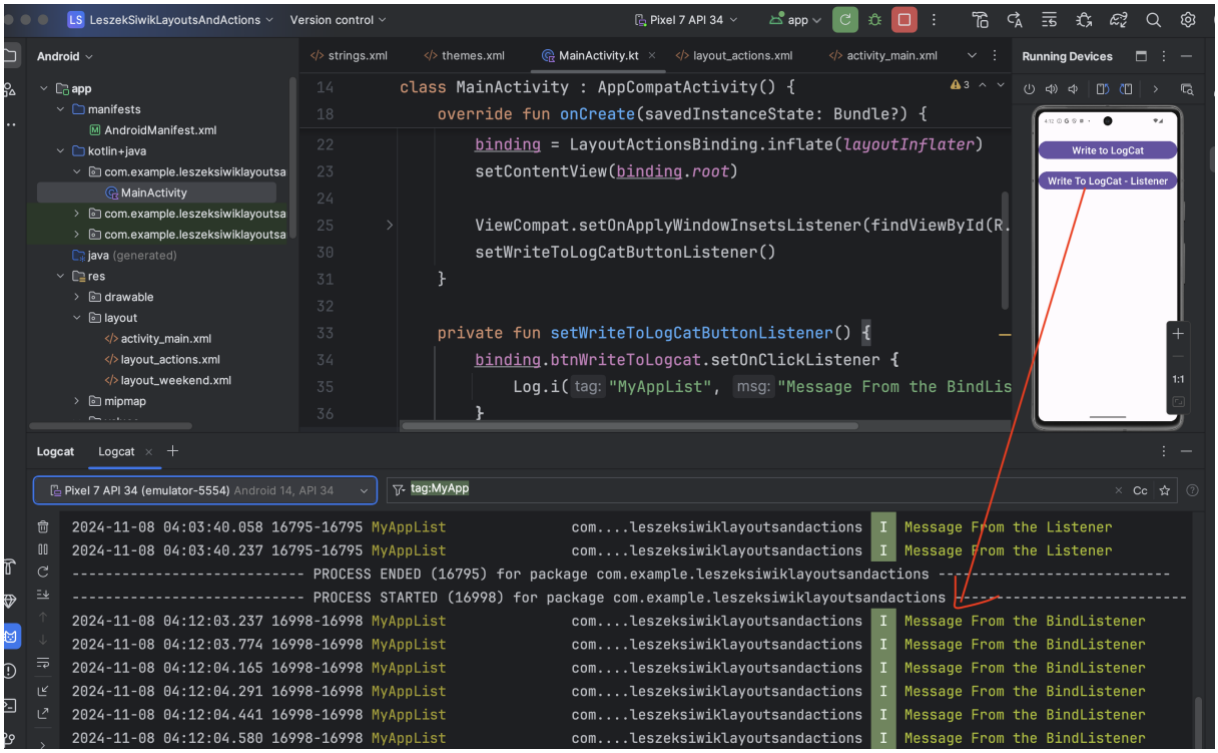
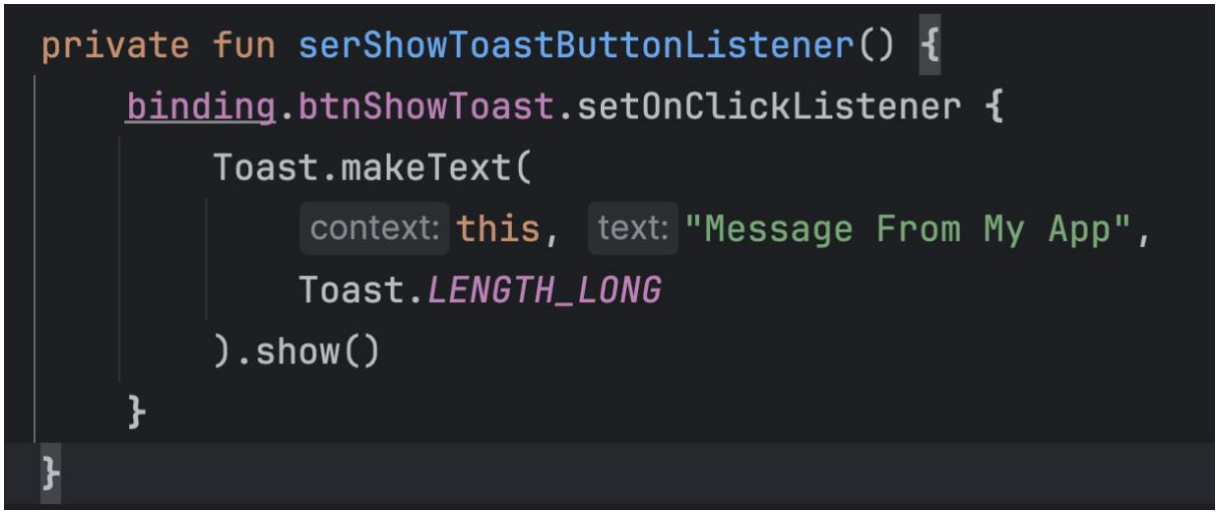
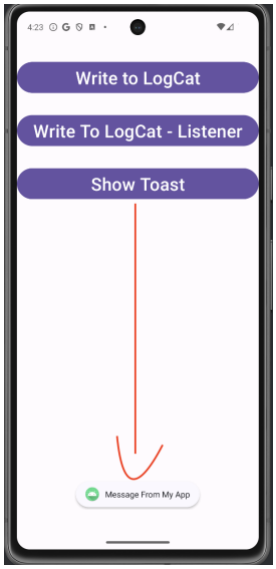
    private lateinit var binding: LayoutActionsBinding

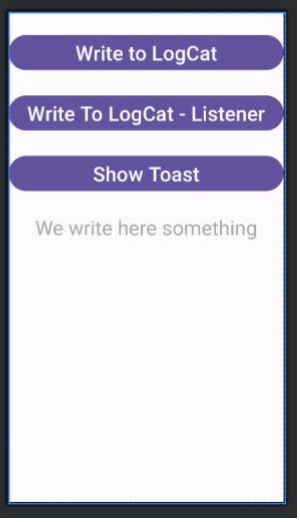
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
```

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

```
class MainActivity : AppCompatActivity() {

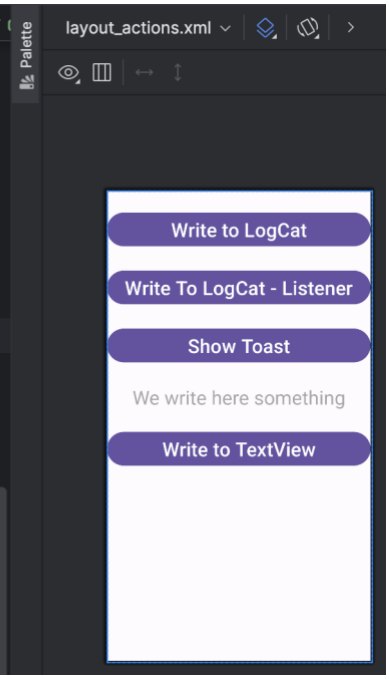
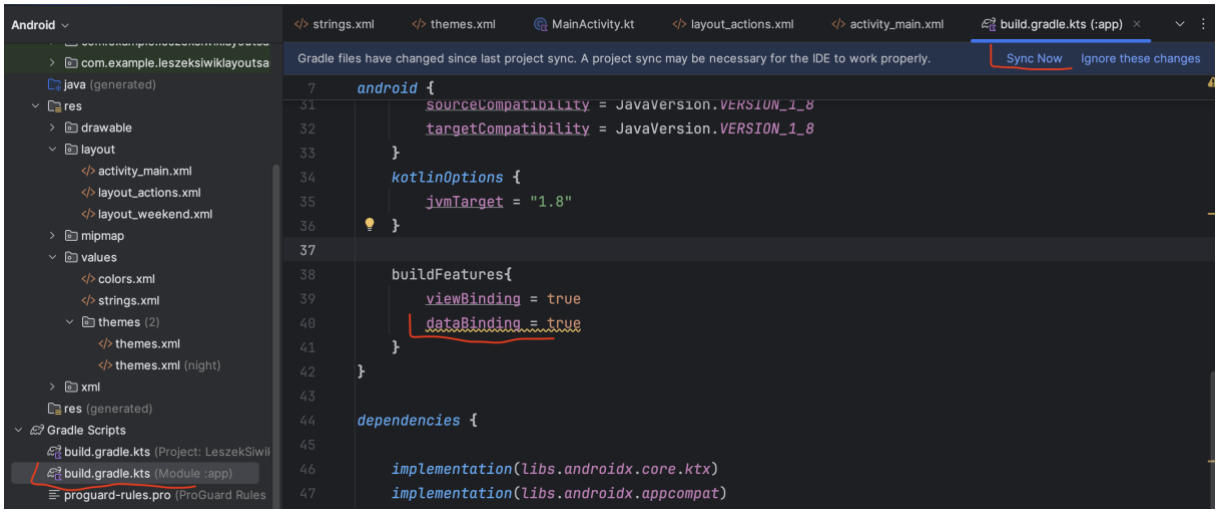
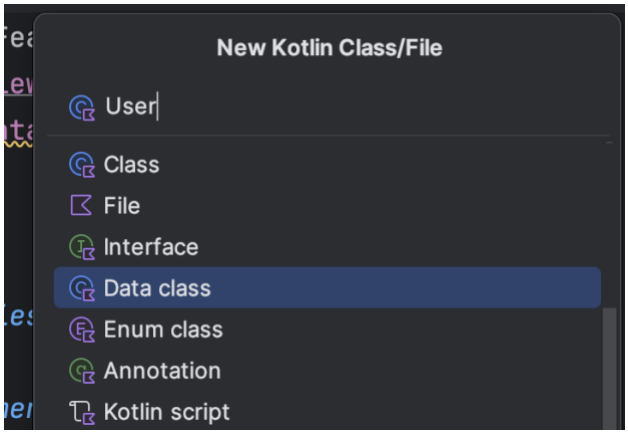
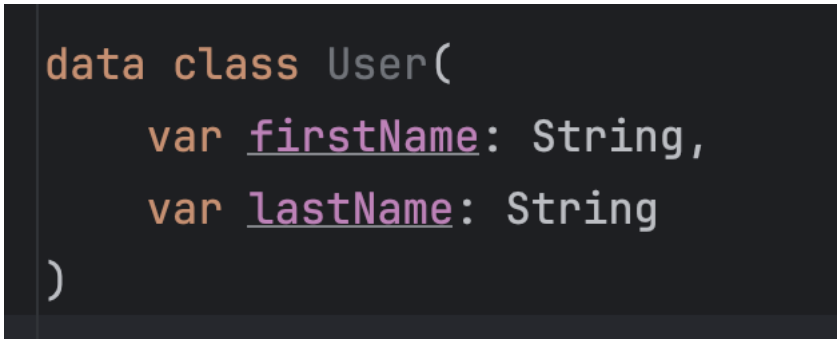
    private lateinit var binding: LayoutActionsBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()

        binding = LayoutActionsBinding.inflate(layoutInflater)
        setContentView(binding.root)

        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.actions)) {...}
        setWriteToLogCatButtonListener()
    }
```

- 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 setWriteToLogCatButtonListener() {
    binding.btnWriteToLogcat.setOnClickListener {
        Log.i( tag: "MyAppList",  msg: "Message From the BindListener")
    }
}
```

- 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 to layout_actions and let’s define onClickListener as we just did it before.
- Next, in our listener, let’s create and show the Toast as it is shown below.
- 

- And while testing we should see the following:

- 

-

- Next, let’s try to write anything directly to UI elements of our app.
- So, let’s add the TextView element to our layout_actions, let’s give it the id like tv_write_something, and of course you may "embellish" it a bit. For me, the definition of this TextVView is as follows:

```
<Button
    android:id="@+id/btn_show_toast"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:text="@string/show_toast"
    android:textSize="30sp" />

<TextView
    android:id="@+id/tv_write_something"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:textSize="30sp"
    android:textAlignment="center"
    android:hint="@string/we_write_here_something"/>
```



-
- Then, let’s add the new button similarly to as we did it before

-
```
<LinearLayout xmlns:android="http://schemas.android.com
    <TextView
        android:id="@+id/tv_write_something"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:hint="@string/we_write_here_something"
        android:textAlignment="center"
        android:textSize="30sp" />

    <Button
        android:id="@+id/btn_write_to_textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:text="@string/write_to_textview"
        android:textSize="30sp" />
```



- 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:

```
private fun setWriteToTextViewButtonListener() {
    binding.btnWriteToTextview.setOnClickListener {
        binding.tvWriteSomething.text = "I love this game"
        //or with creating the string resource
        binding.tvWriteSomething.text = getString(R.string.i_love_this_game)
    }
}
```

- And while testing, it should work as expected i.e.
-
  - 

- 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 the data class to our project ((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

- ```
  class MainActivity : AppCompatActivity() {

      private lateinit var binding: LayoutActionsBinding
      private lateinit var user: User
  ```
- 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 that is why, we need to wrap up our LinearLayout that we are working with, with a generic layout tag as it is done below:
- ```
  themes.xml   MainActivity.kt   layout_actions.xml ×   activity_main.xml   build.gradle.kts (
  <?xml version="1.0" encoding="utf-8"?>
  <layout>

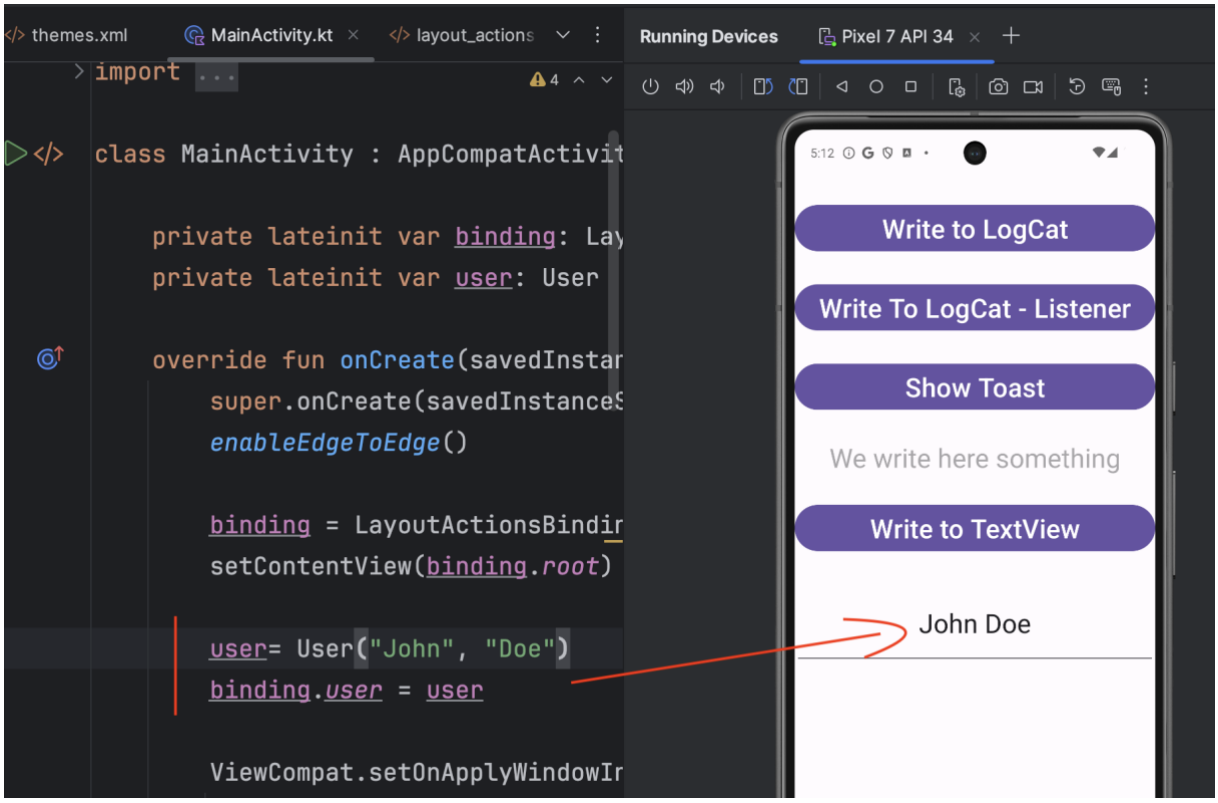
      <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/actions"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:orientation="vertical">

          <Button
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_marginTop="30dp"
              android:onClick="writeToLogCat"
              android:text="@string/write_to_logcat"
              android:textSize="30sp" />
  ```
- Next Inside the (generic) layout we may define the data model binding as it is done below:

- ```
  themes.xml | MainActivity.kt | layout_actions.xml | activity_main.xml | build.gradle.kts (:app)
  <?xml version="1.0" encoding="utf-8"?>
  <layout>
      <data>
          <variable
              name="user"
              type="com.example.leszeksiwiklayoutsandactions.User" />
      </data>

      <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/actions"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:orientation="vertical">

          <Button
  ```
- Now, let’s add a new EditText and let’s set it up to use user model data, e.g. as it is shown below:
- ```
  <EditText
      android:id="@+id/et_data_binding"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="30sp"
      android:layout_marginTop="30dp"
      android:textAlignment="center"
      android:hint="@string/data_binding_should_work_here"
      android:text='@{user.firstName + " " + user.lastName }'/>
  ```
- 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

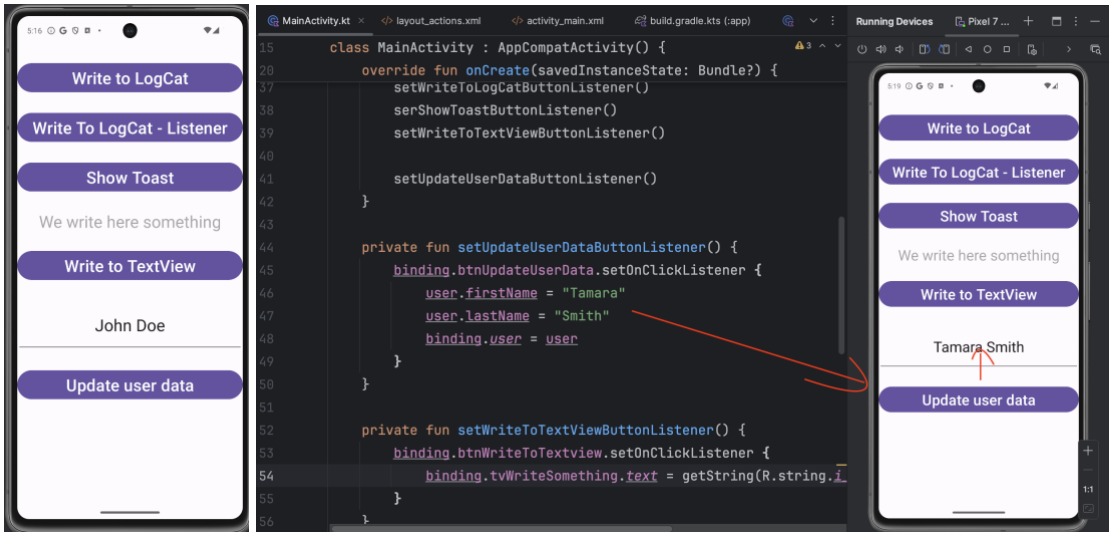


-

- 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 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:

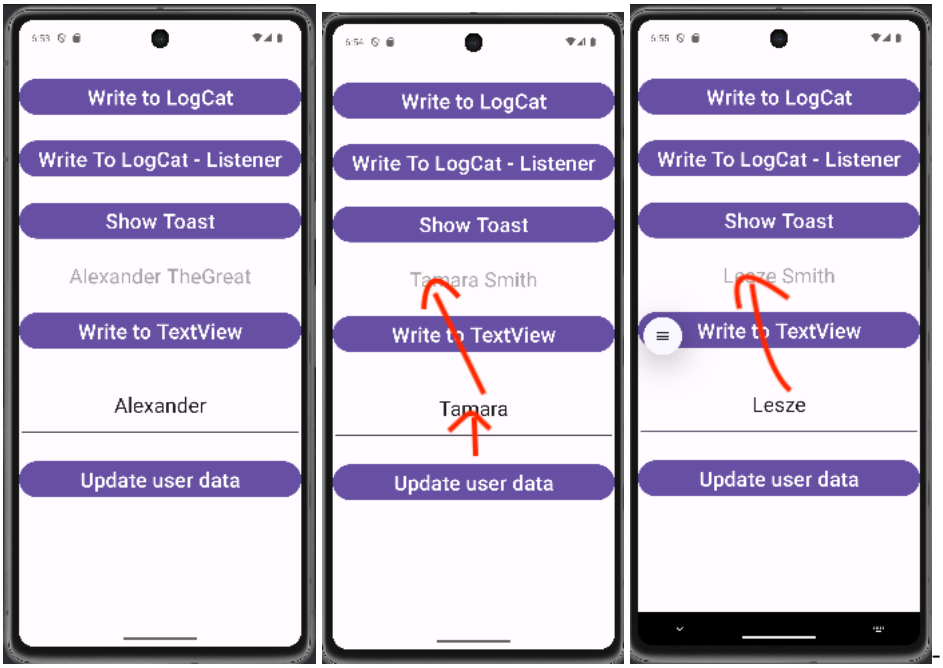
- ```
  override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      enableEdgeToEdge()

      binding = LayoutActionsBinding.inflate(layoutInflater)
      setContentView(binding.root)

      user = ViewModelProvider( owner: this)[User::class.java]
      user.firstName = MutableLiveData( value: "Alexander")
      user.lastName = MutableLiveData( value: "TheGreat")

      binding.user = user
      binding.lifecycleOwner = this
  ```
- 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/tv_write_something"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:hint='@{user.firstName + " " + user.lastName}'
    android:textAlignment="center"
    android:textSize="30sp" />
```

- 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/et_data_binding"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="30sp"
    android:layout_marginTop="30dp"
    android:textAlignment="center"
    android:hint="Data Binding Should work here"
    android:text="@={user.firstName}"/>
```

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



- As or practicing pls add analogous field for updating user lastname
- When done please upload the screenshots of your code and app on UPEL platform