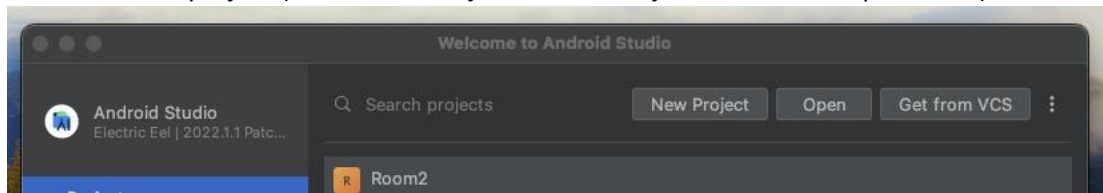
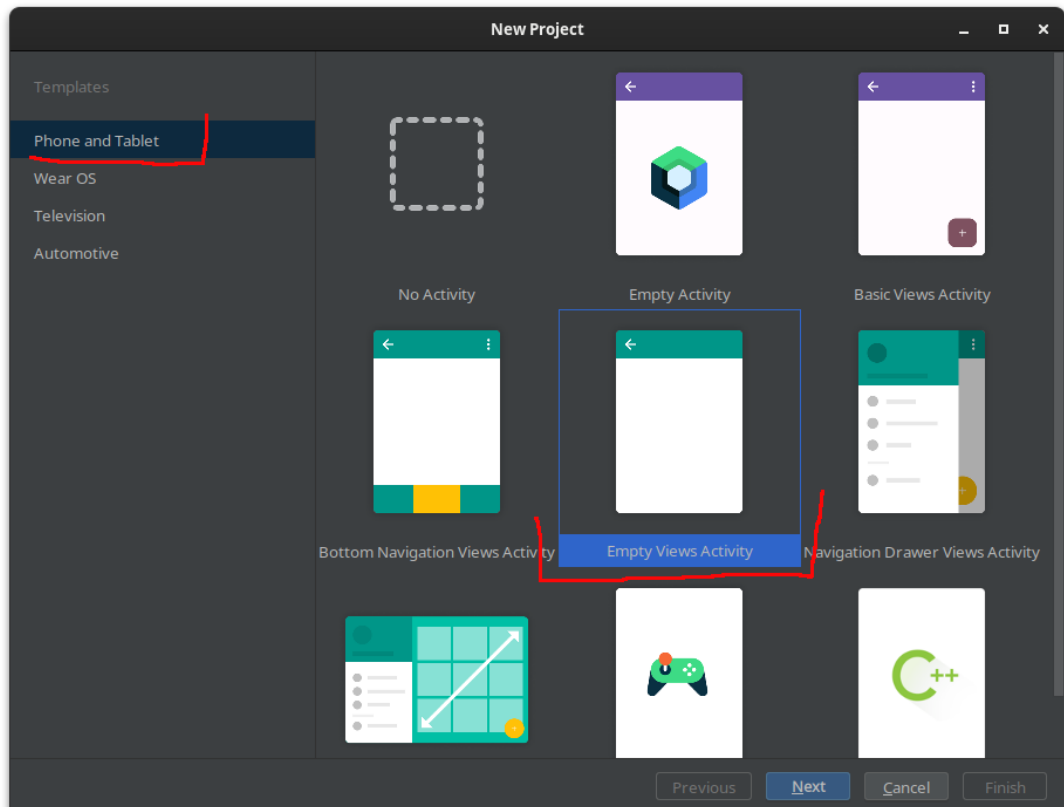


Android Apps - 1

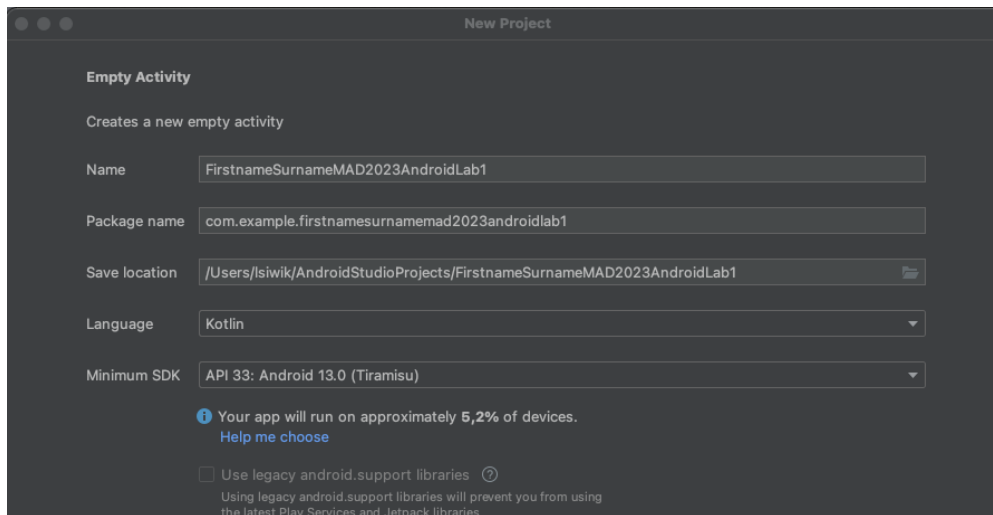
- Launch Android Studio
- Create a new project (File -> New Project or New Project from a Startup window)



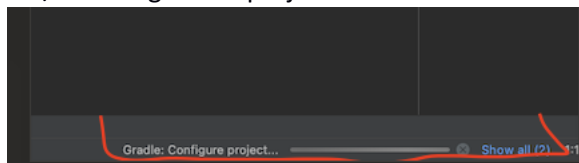
- Select Phone and Tablet and Empty Viewer Activity as a project template



- On the next screen set:
 - FirstnameSurnameAndroidLab1 as the project name
 - Default value as the package name and project location
 - Kotlin as project language
 - API 33 Android 13.0 (Tiramisu) as the Minimum SDK
 - And we are not going to use any legacy code so keep it unchecked
 - And press Finish



- Wait until all the configuration tasks by Gradle are done (what may take a time when AS / Creating a new project is launched for the first time)



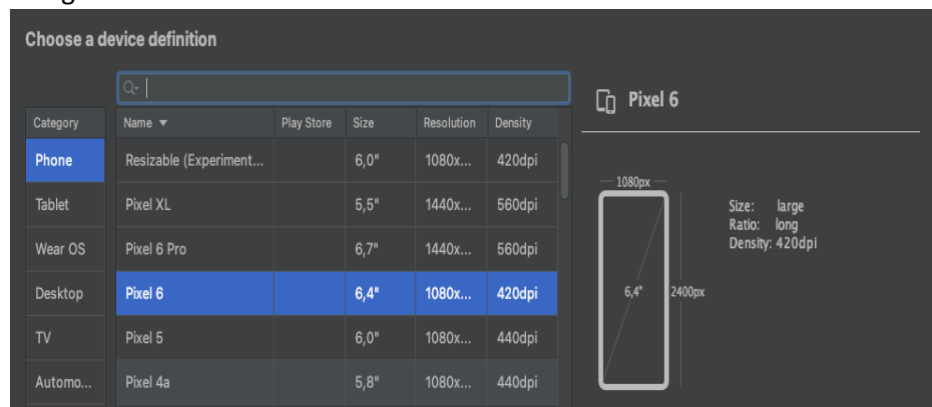
- To run you projects you need to connect a physical device or use the virtual device, what is the easiest way. During the classes I'm going to use Pixel 6 device running Android Tiramisu (API level 33). If you want to have the same, and it is not available in Android Studio yet, you need to create it. To do so you need to:

- go to the Device manager



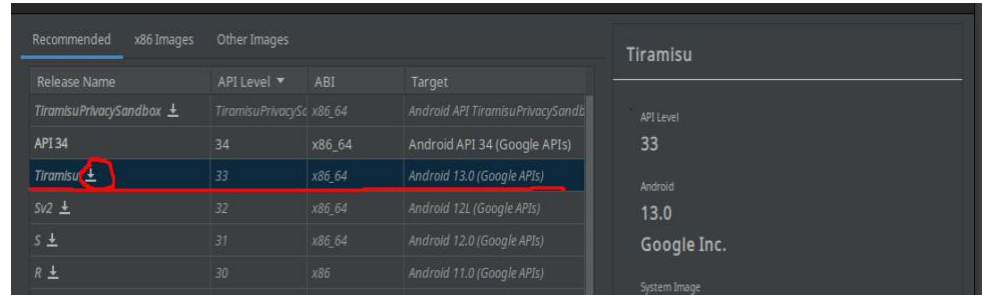
- (or Menu Tools -> Device manager)

- select Create device -> Pixel 6 (or other preferable) as the device template and go Next

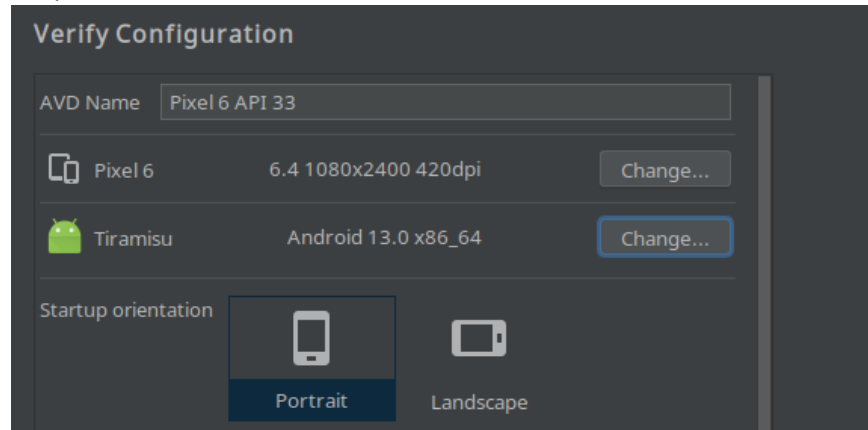


- select API 33 System Image (You must download it if not done before)

○



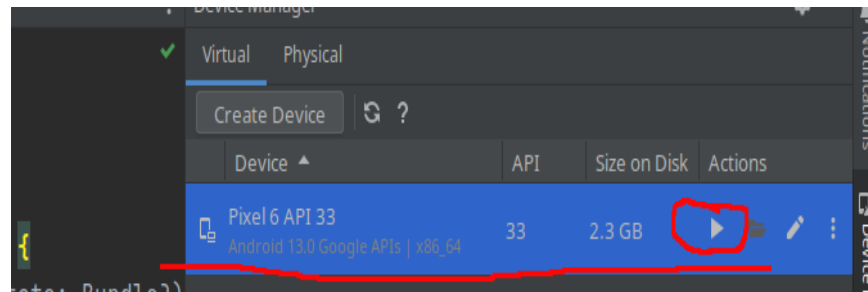
○ keep the defaults on the next screen



○

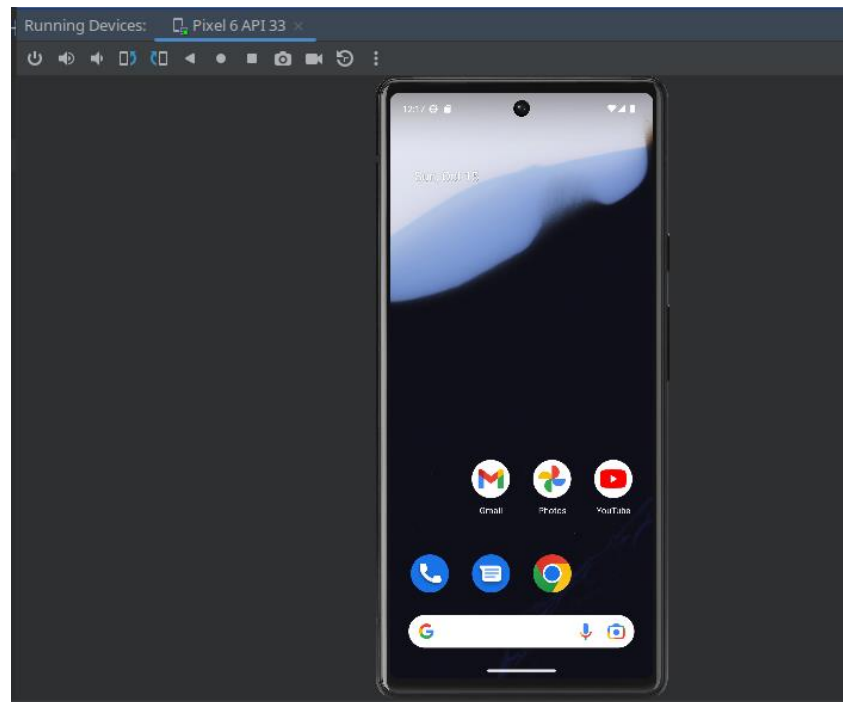
○ and press Finish

○ when everything goes fine, the new device should appear on the list of available virtual devices/emulator. So please, select and launch the newly added device

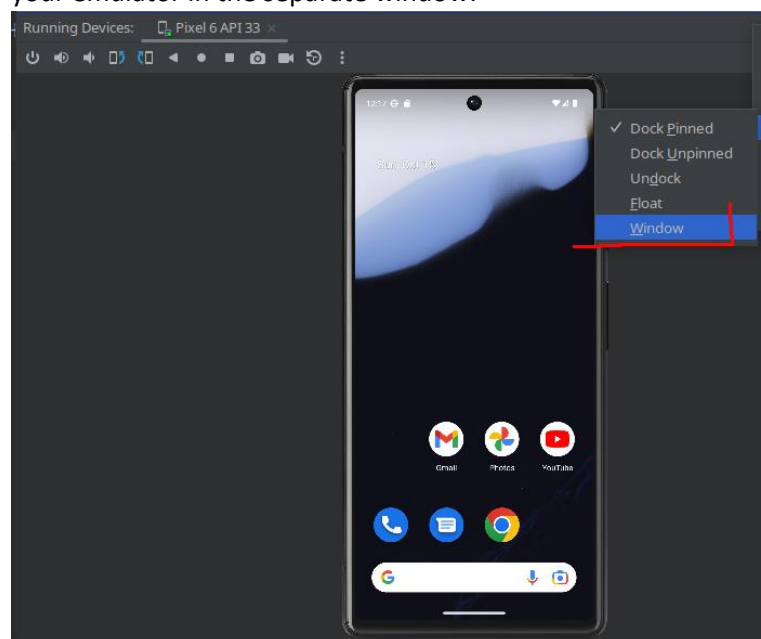


○

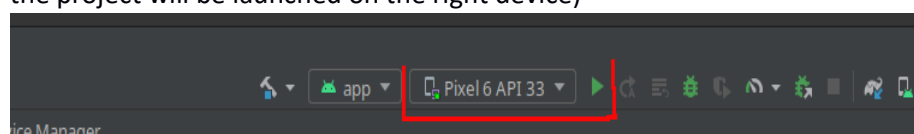
○ and again, after a while, you should see your virtual device up and running



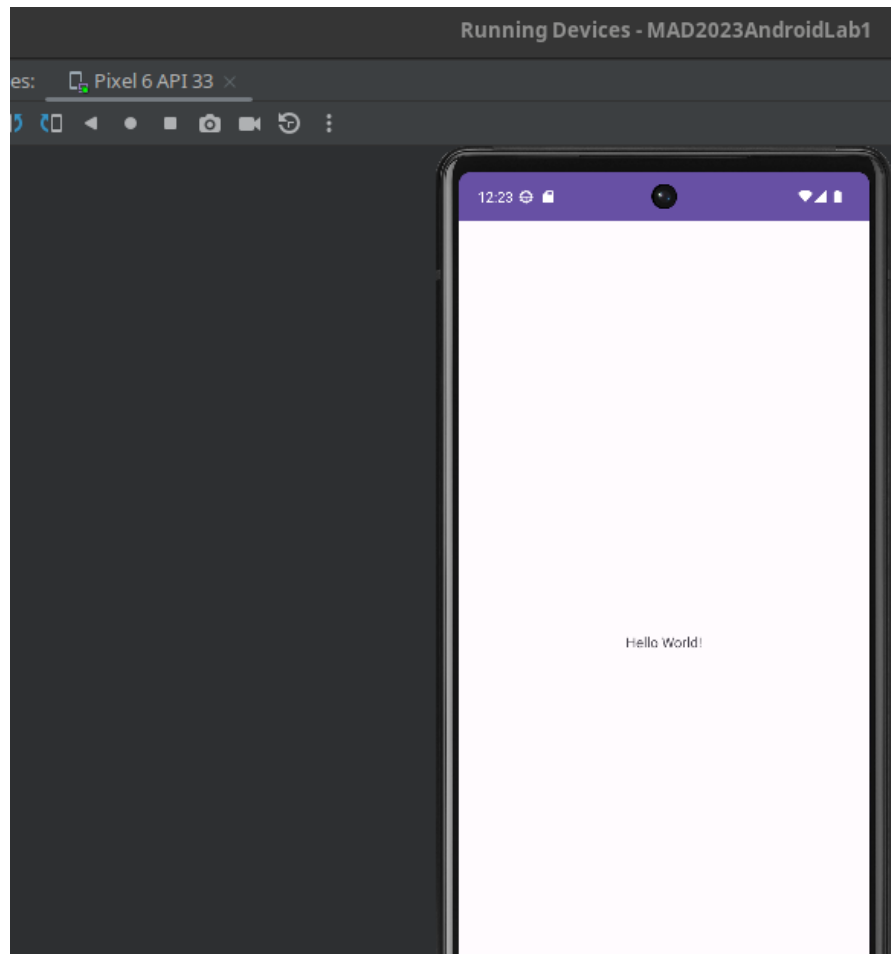
-
- Potentially, it may be more convenient to change the view mode and to run your emulator in the separate window:



- Now we are ready to launch our project, so press the green arrow on the tool bar (or press the shortcuts (Shift + F10 on windows, Ctrl + R on mac) (also, make sure that the project will be launched on the right device)

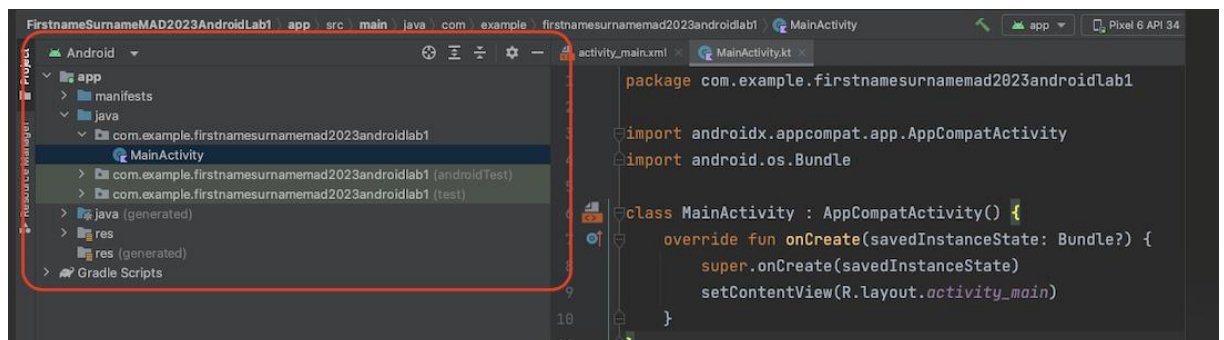


-
- and (again, after the while ☐ you should see your application/projects running on your virtual device

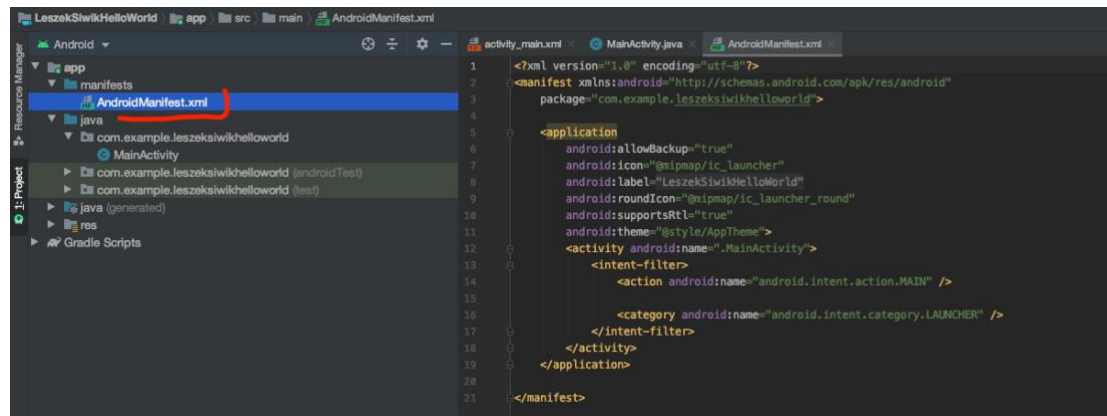


- **Project structure**

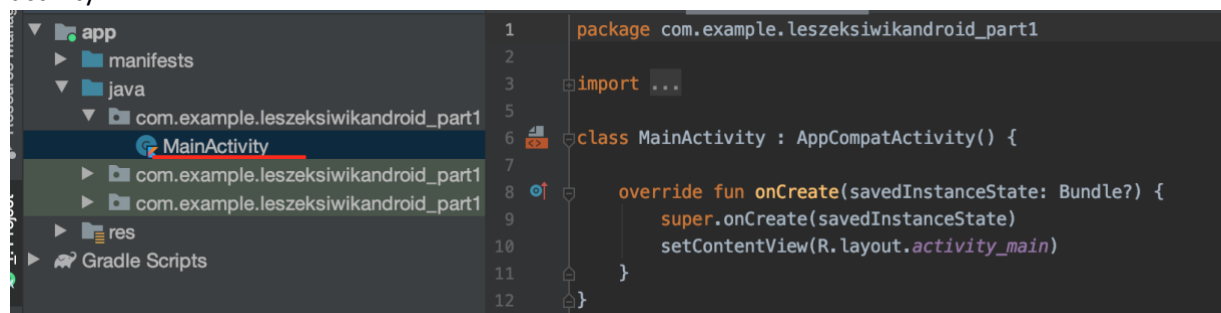
- Now, let's get familiar with the project structure.
- In Android Studio, on the left side you will find the project files.:



- In app / manifests / AndroidManifest.xml you will find the application manifest:



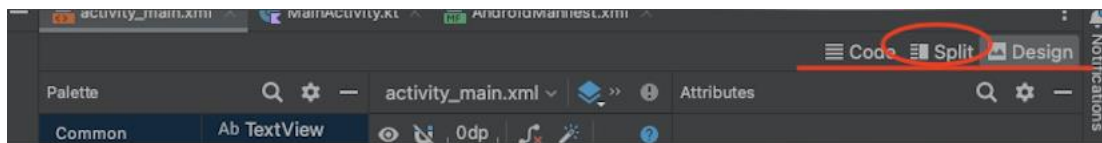
- A manifest is a kind of a high-level app "description". It is used, among others, when the app is installed and launched on the device (startup activity, application icon, package name, etc.)
- More information about the app manifest can be found here:
<https://developer.android.com/guide/topics/manifest/manifest-intro>
- In the java directory you will find application sources. In our case, at the beginning you will find there a MainActivity file containing the implementation of the first (created by the wizard) activity.



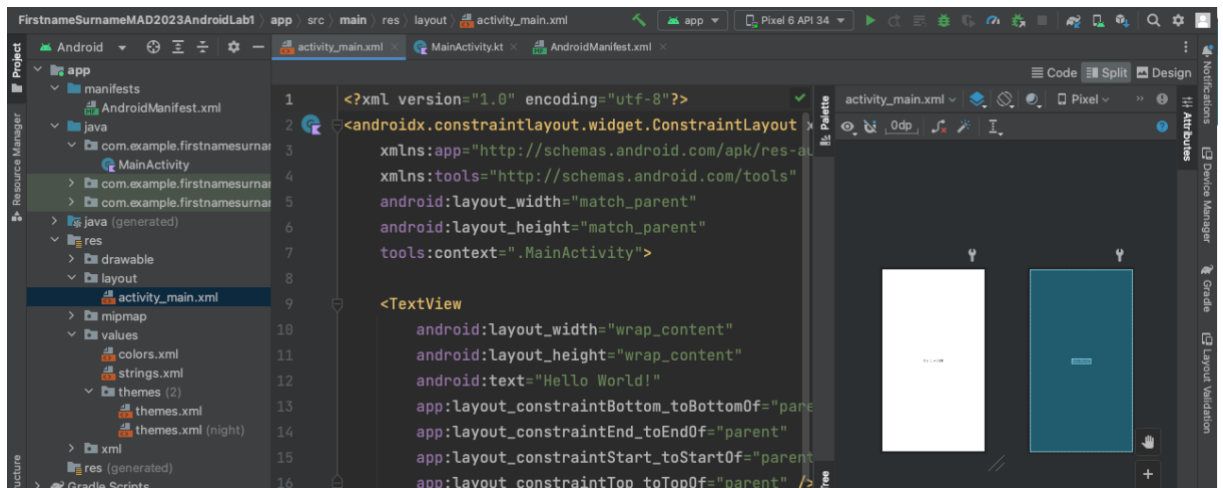
- Activity is the definition of the single "view"/"screen". More information can be found here:
<https://developer.android.com/guide/components/activities/intro-activities>
- The res folder is a place for the application's "resources":
 - drawable - a place for "drawable" graphics (sometimes also photos / images) used by applications
 - layout - a place for layouts definitions (i.e xml files containing definition of the view / screen / user interface)
 - values / colors.xml space for color definitions
 - Values / strings.xml place for string definitions
 - Etc.

• Layouts definitions

- Definition of how our first activity (first screen) looks like is contained in the res -> layout -> activity_main.xml file
- To define the user interface definition you may work either with the designer or directly with the xml file containing the UI definition.
- **Because we are practicing, please complete the ongoing exercises by editing the xml file(s) directly!**
- Possibly the most convenient view mode for us will be a split view showing both: the UI source code and the designer where you may follow the effects of your changes made in the source code. So please change the view mode into the SplitView



- And you should get something like:

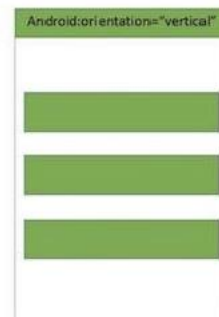
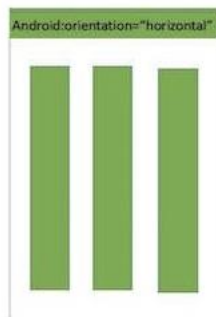


• Layouts

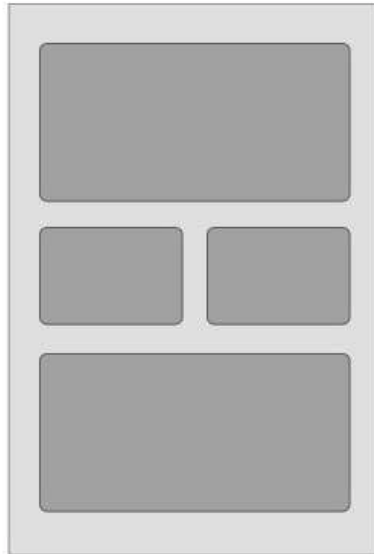
- When we are defining the user interface, we use "layouts" - a type of containers where we may drop off user interface elements like buttons, lists, text fields, other nested layouts, etc.)
- Android SDK provides some predefined layouts like:

Sr.No	Layout & Description
1	Linear Layout LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout TableLayout is a view that groups views into rows and columns.
4	Absolute Layout AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout The FrameLayout is a placeholder on screen that you can use to display a single view.
6	List View ListView is a view group that displays a list of scrollable items.
7	Grid View GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

- (https://www.tutorialspoint.com/android/android_user_interface_layouts.htm)
- LinearLayout – aligns all the elements dropped off inside linearly (vertically or horizontally depending on the “orientation”) one by one:

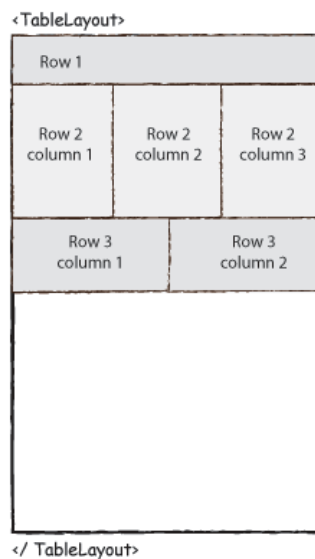


- More information about LinearLayouts can be found e.g. here: https://www.tutorialspoint.com/android/android_linear_layout.htm
- RelativeLayout - where elements are placed "relatively" to each other - e.g. we place something to the right of something, something else under something, etc.

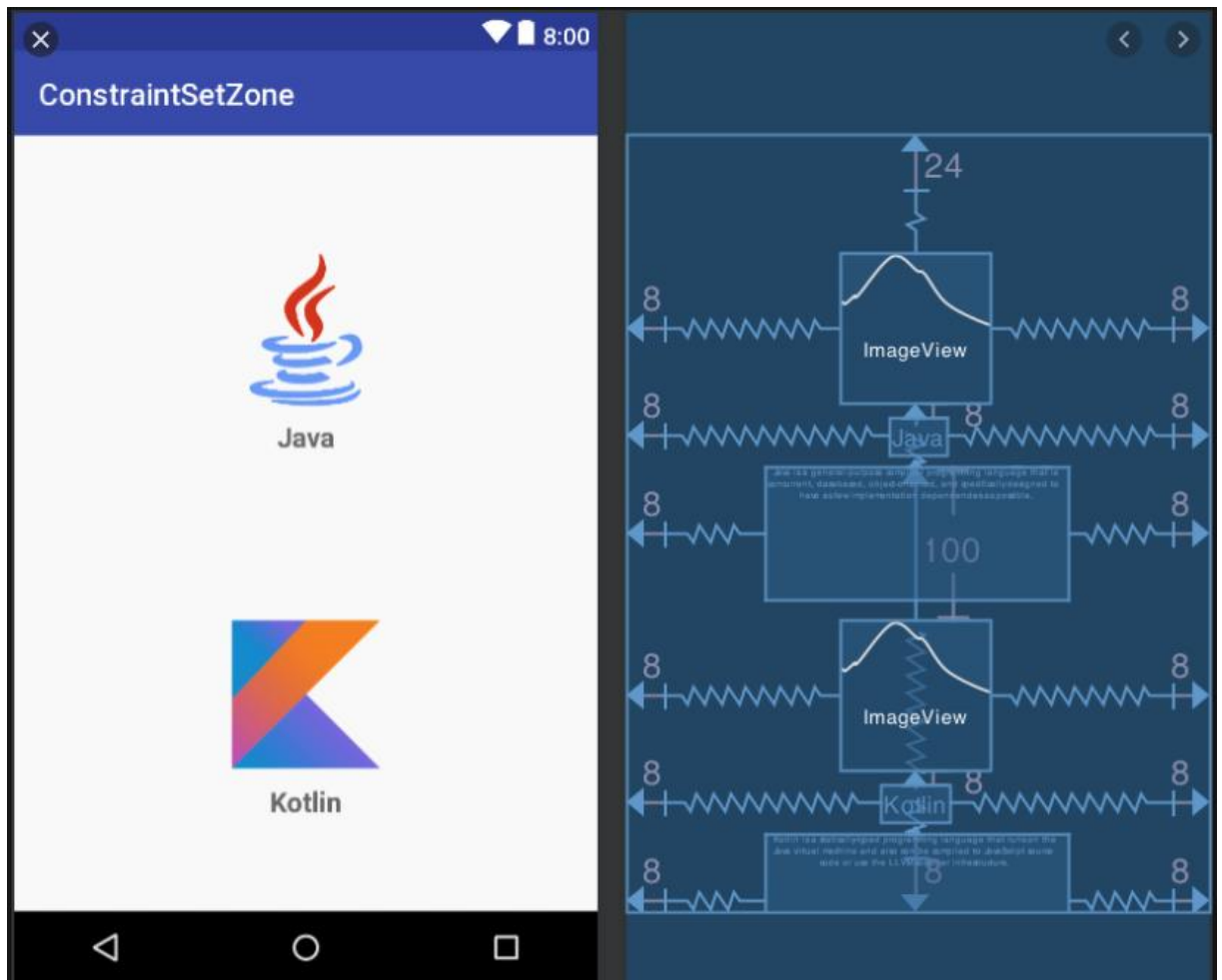


Relative Layout

- More information about RelativeLayouts can be found e.g. here: https://www.tutorialspoint.com/android/android_relative_layout.htm
- TableLayout – where elements are located in rows and columns

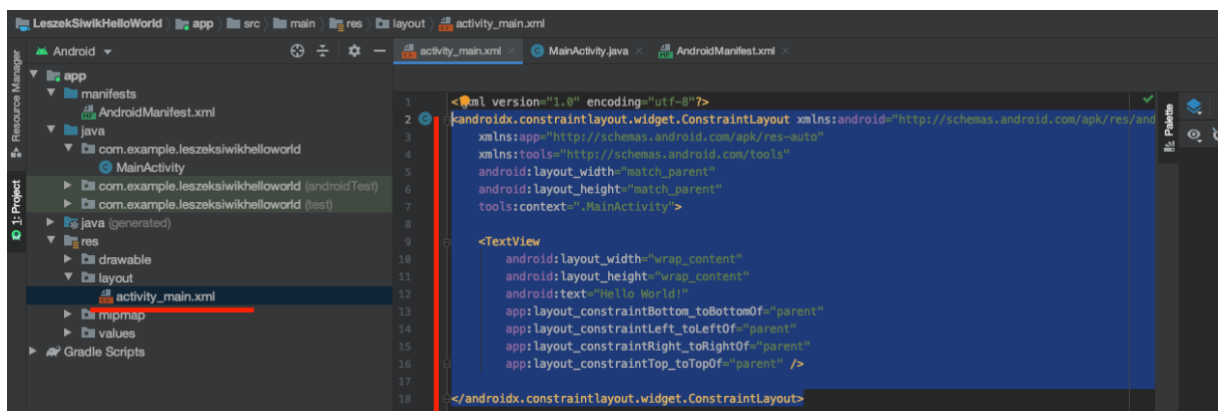


- More information about TableLayouts can be found e.g. here: https://www.tutorialspoint.com/android/android_table_layout.htm
- ConstraintLayout – where elements are "pinned together" with constraints/strings

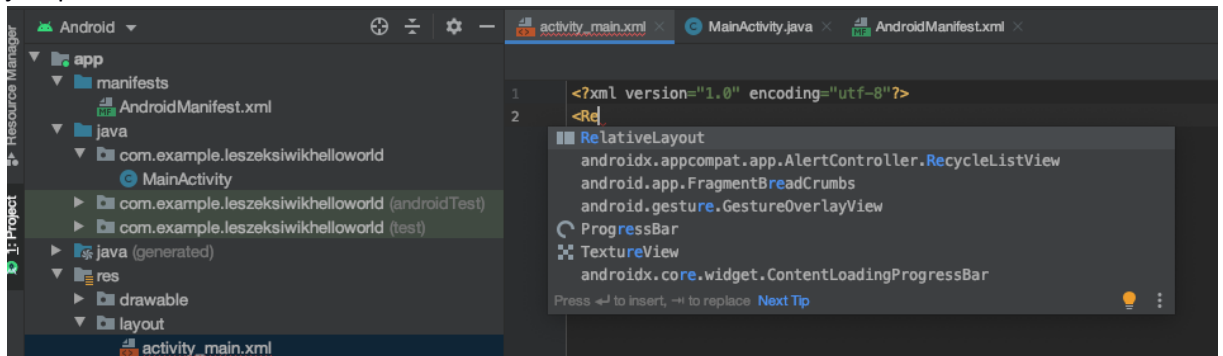


• RelativeLayout

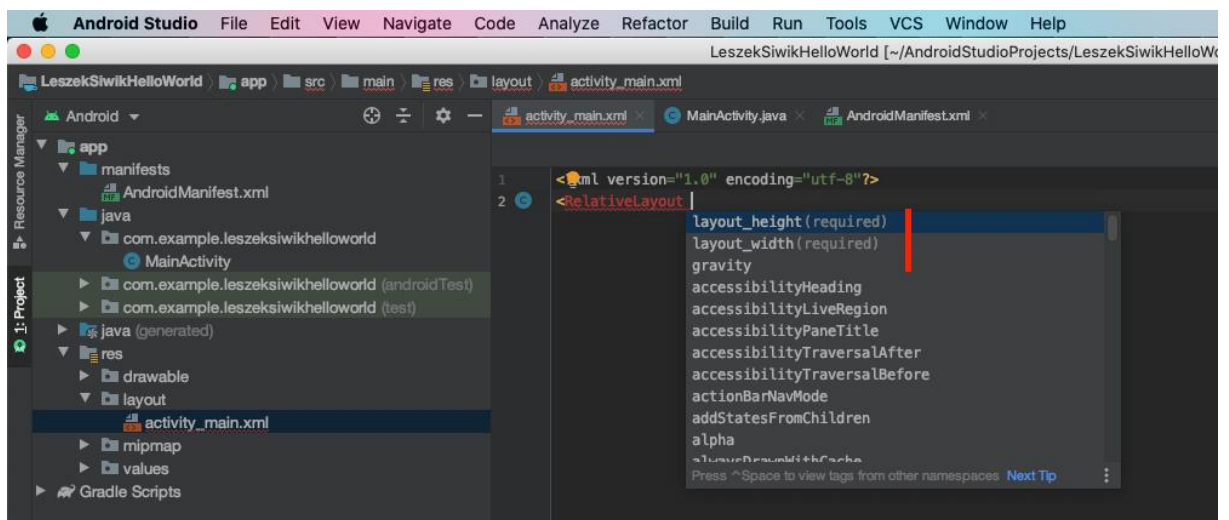
- For the beginning, we start working with the RelativeLayout
- As it was said, the RelativeLayout is a "container" where elements are placed "relatively" to each other – i.e, we place something to the right of something, something else under something etc. We will work with it for a while now.
- Please, open the main_activity.xml file. Delete the ConstraintLayout definition added there by the wizard. (So remove lines 2 to 18)



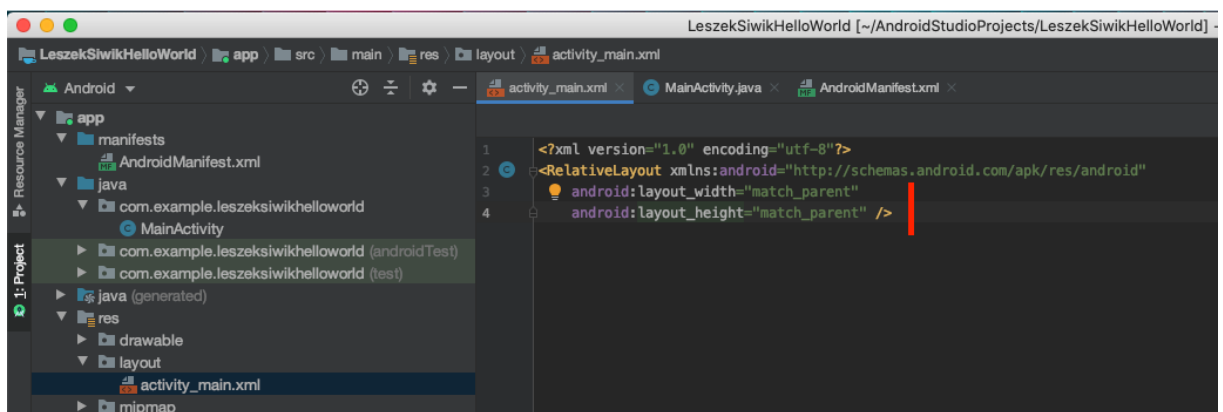
- Instead, please start typing "<Rel" and when the intellisense shows up the RelativeLayout just press the enter:



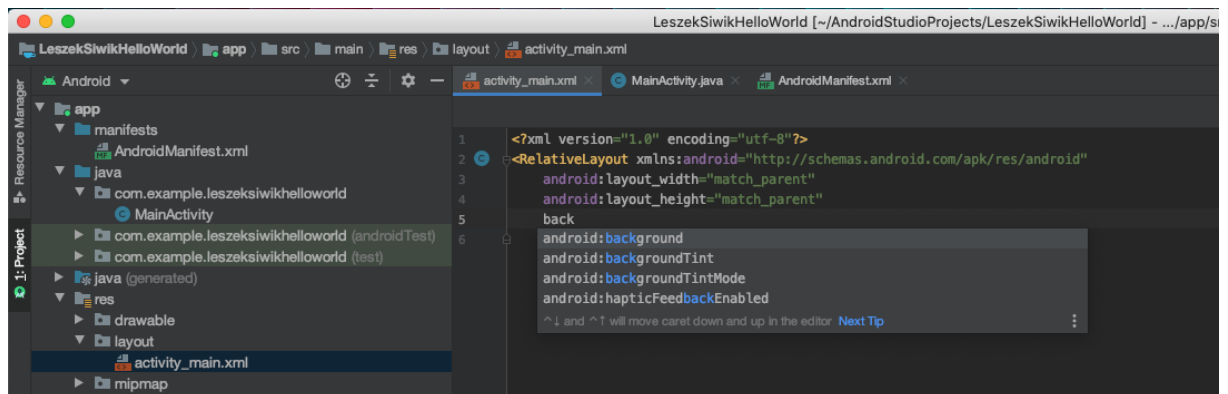
- Next, after RelativeLayout we put the space and press ctrl + space and intellisense will suggest us possible attributes



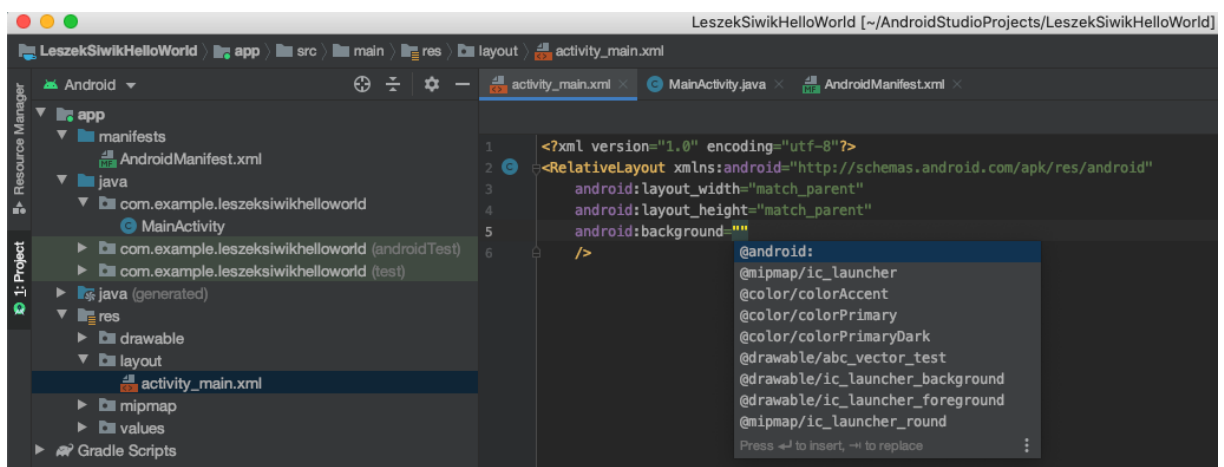
- As we can see, the RelativeLayout has two mandatory parameters - layout_width and layout_height. We set both to match_parent - which means that we want to define the layout as width and as height as its "parent" is, and because the created layout is the "top most" layout – (it is not nested in any other layout) - its "parent" is just the screen of the phone on which the application will be launched - so by setting layout_width and layout_height to "match_parent" we say that we want the created layout to be as wide and as high as the screen of the device on which the application will be launched .



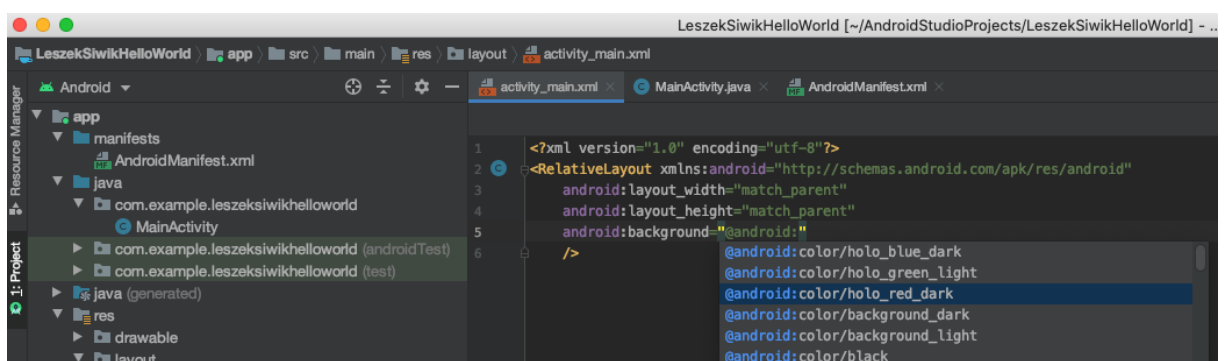
- Additionally, let's set the background color of the layout, to red (or any other), what makes it easy to “track” of how the layout is defined. So, being inside the layout definition, we start typing “back” and Intellisense should support us:



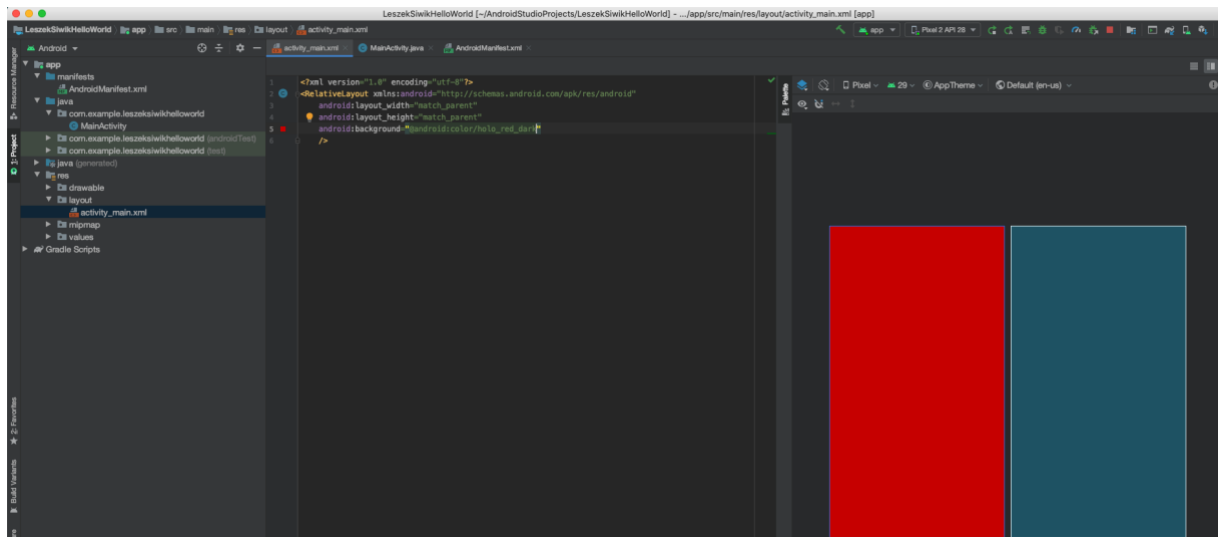
- We choose android: background option
- Then being inside "quotation marks" we select the option @android (Please remember, if intellisense doesn't appear you can always launch it by pressing ctrl + space)



- And then we can choose the color as we like (e.g. color/holo_red_dark)

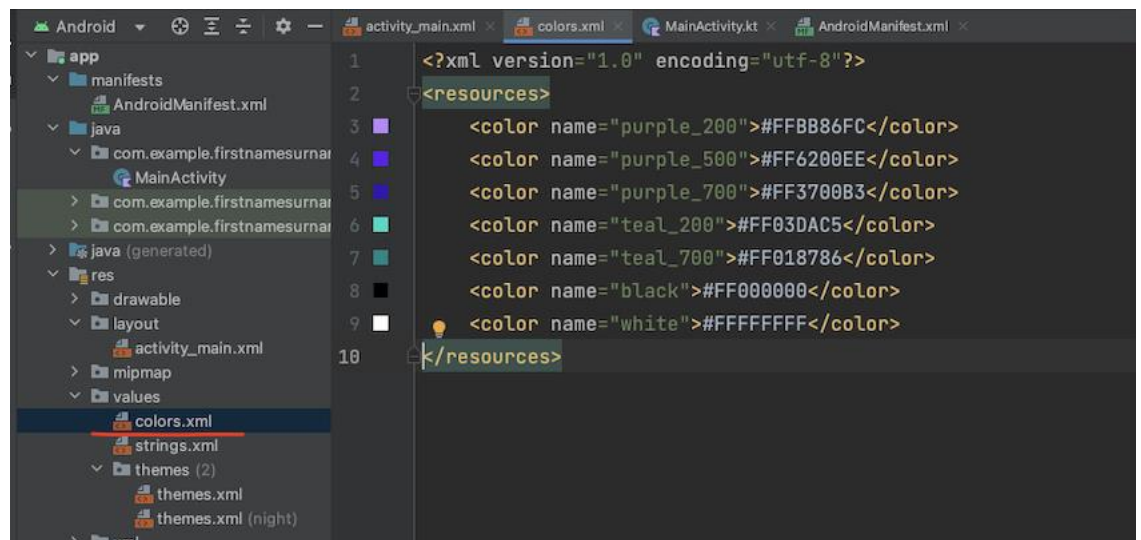


- The background color change should be visible on the designer / preview as it is shown below:

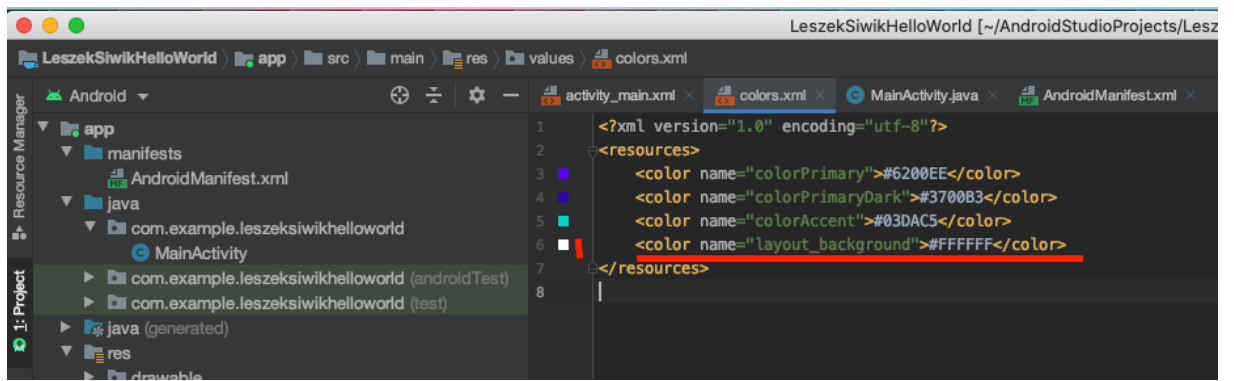


• Working with custom colors

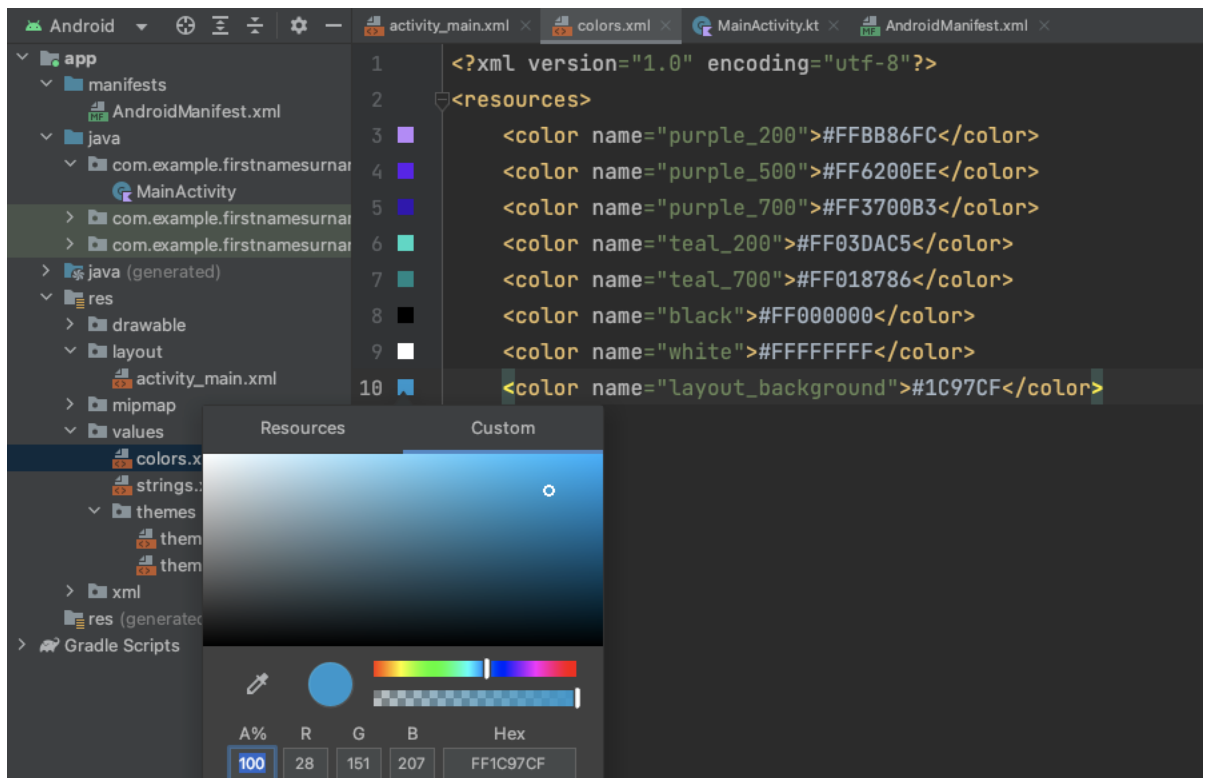
- As we touched the color topic - let's define our own color and set it as the layout background color. For this purpose:
 - Open res/values/colors.xml file



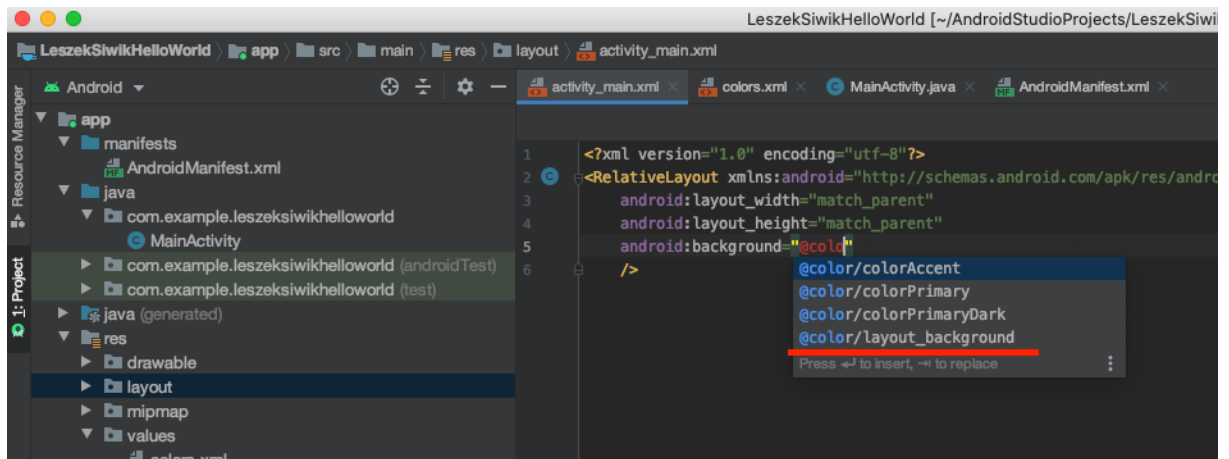
- Similarly to those already defined there - let's add a new "color item"
- Set the name of the color to be added (e.g. "layout_background") and its value to #FFFFFF



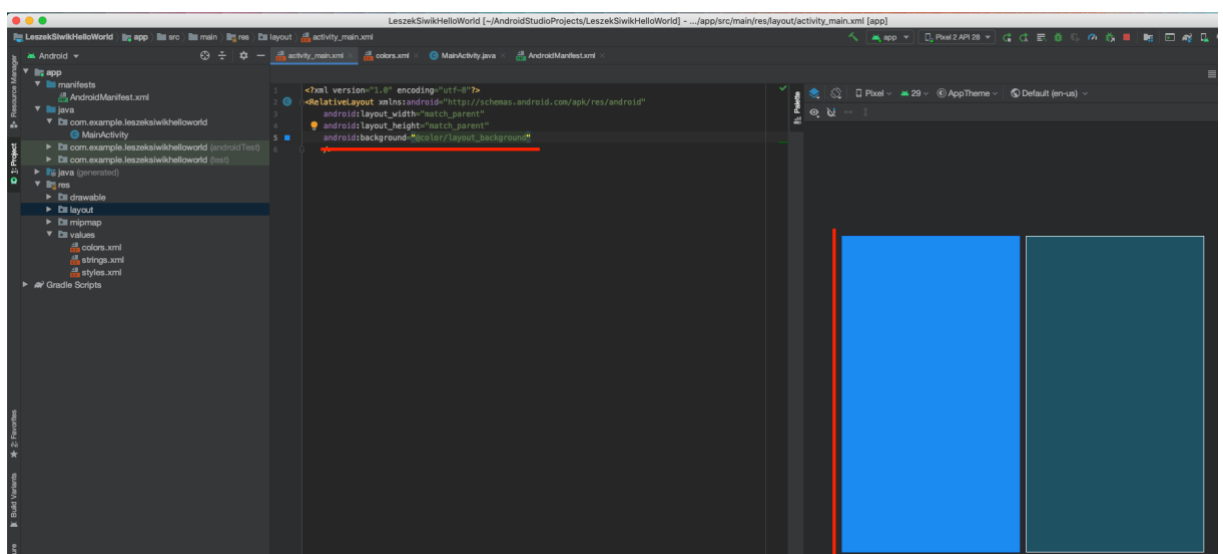
- Click on the white square to the right of the line number (for me to the right of line 6) and choose any color you want to be used



- Now, let's go back to activity_main.xml file and set the value of the layout_background property to the color we have just defined.
- So, being in the line defining the background color property, we remove the value in the quotation marks, let's start typing @color (since we would like to use the color defined in the @color "namespace" we set the value to our layout_background color.

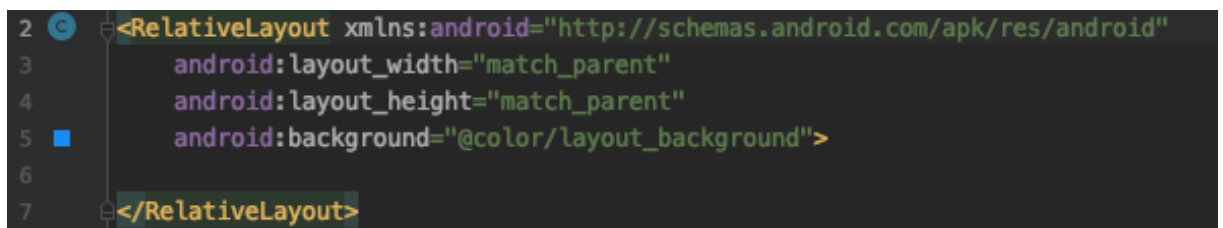


- The color change should be visible on the designer / preview



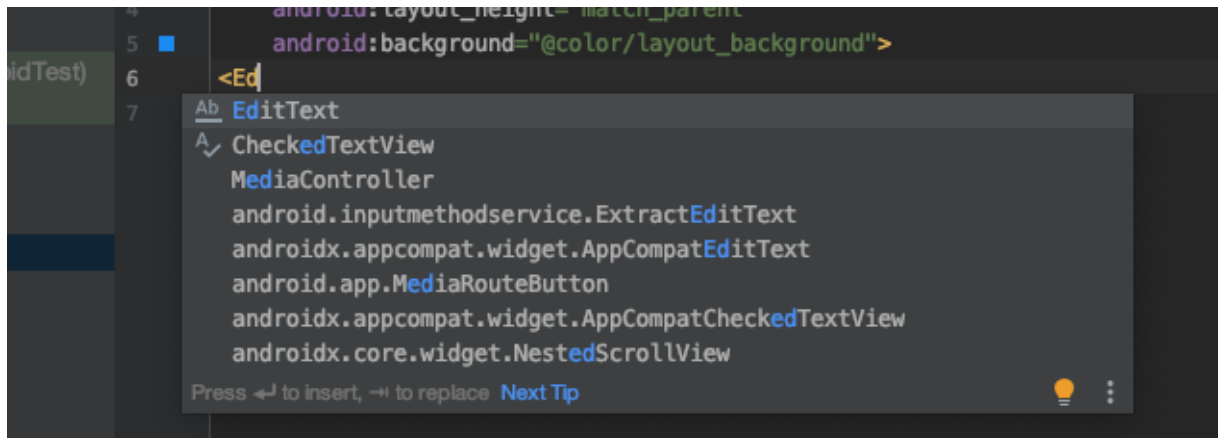
• Working with label

- Let's add some "label" (static inscription) to the layout, namely:
- Being in the activity_main.xml file - we split RelativeLayout definition into two separate tags - opening and closing ones.
- So, we change its definitions of our layout as follows:

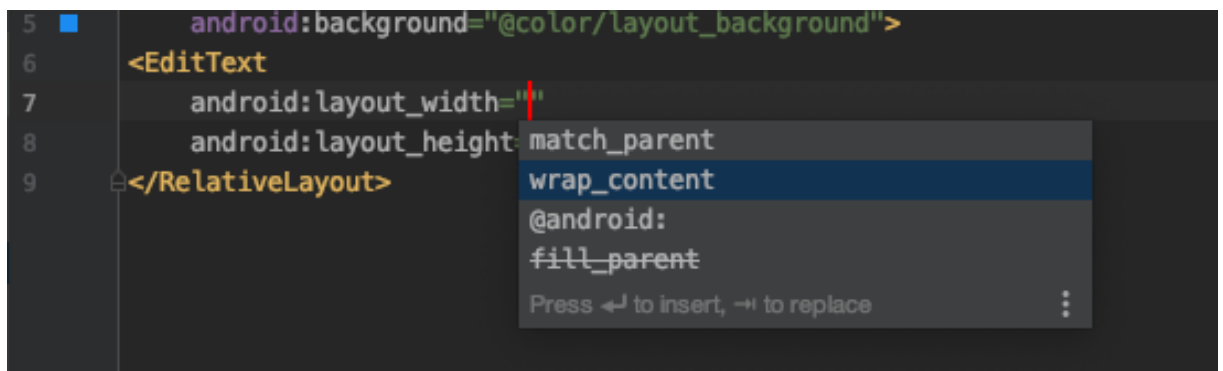


- As the consequence of the fact that we separated the "beginning" of the layout from its "end", we can add some elements in the middle (i.e. in line 6)

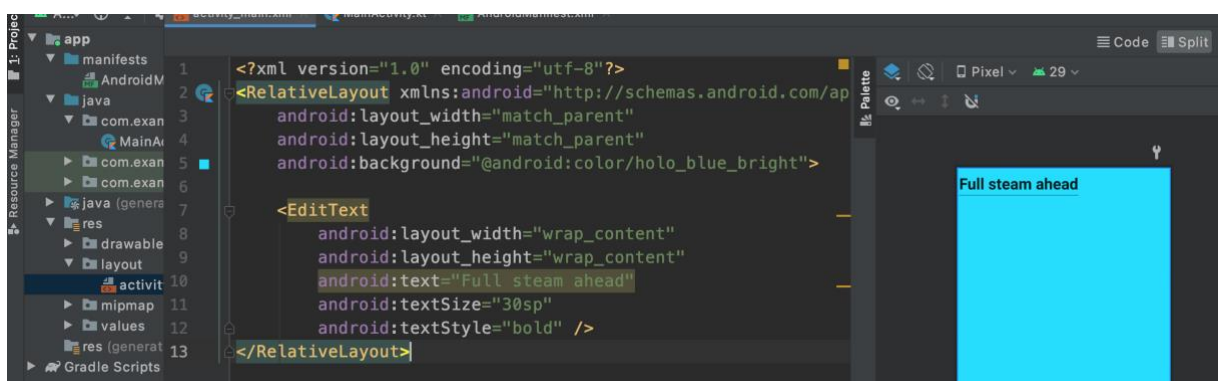
- So, being with the cursor on my line 6, I start typing <Edi and Intellisense should suggest possible options:



- From among the available options let's choose the EditText (a static inscription)
- This time, let's set the width and height of this element to "wrap_content" - that is, we want the EditText to be as wide and as high as the inscription we will place inside.



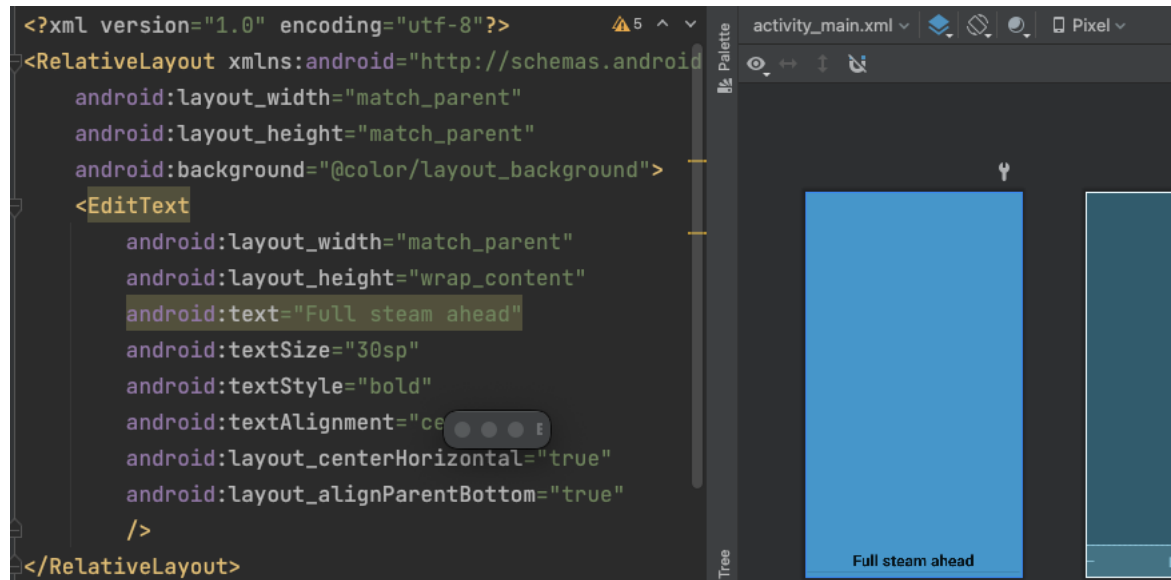
- Next, let's set the text property of our EditText to "Full steam ahead",
- Additionally, let's set textSize to "30sp" and textStyle to Bold. The definition and effect should be as below:



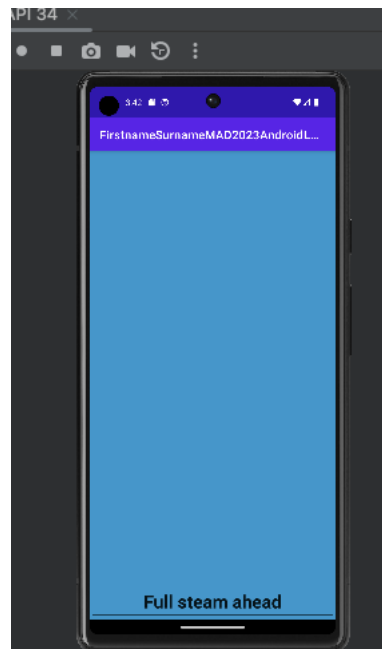
- The "sp" unit that we specified when defining the size of the text it is so-called scale-independent-pixels, which means the abstract pixel size taking into account the physical "density" of the screen of the device on which the application is launched and user preferences (fonts size) as well. More information can be found, e.g. here:

<https://stackoverflow.com/questions/2025282/what-is-the-difference-between-px-dip-dp-and-sp>

- Let's make our label to be centered horizontally (property: `android:layout_centerHorizontal = "true"`) and let's place it at the bottom of the screen (property: `android:layout_alignParentBottom = "true"`). So the definition and the effect should be as follows::



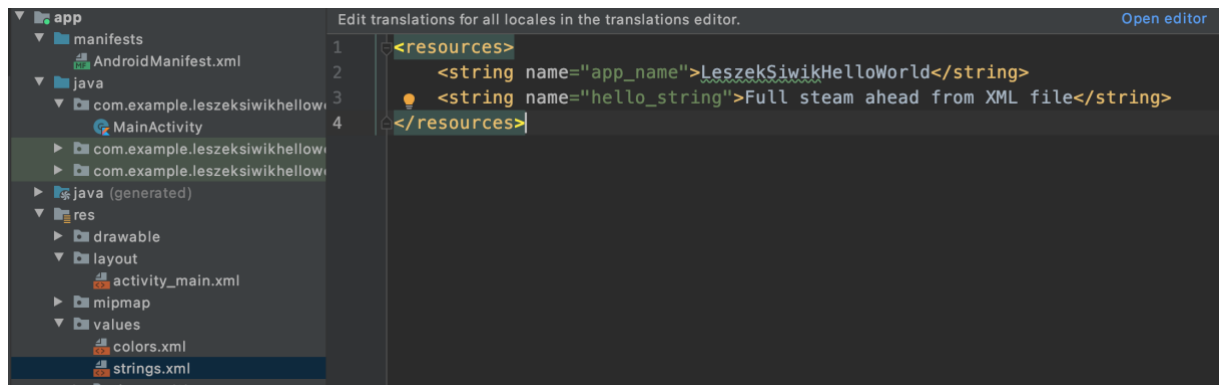
- We can now run our application to see effects on the emulator and we should see something like this:



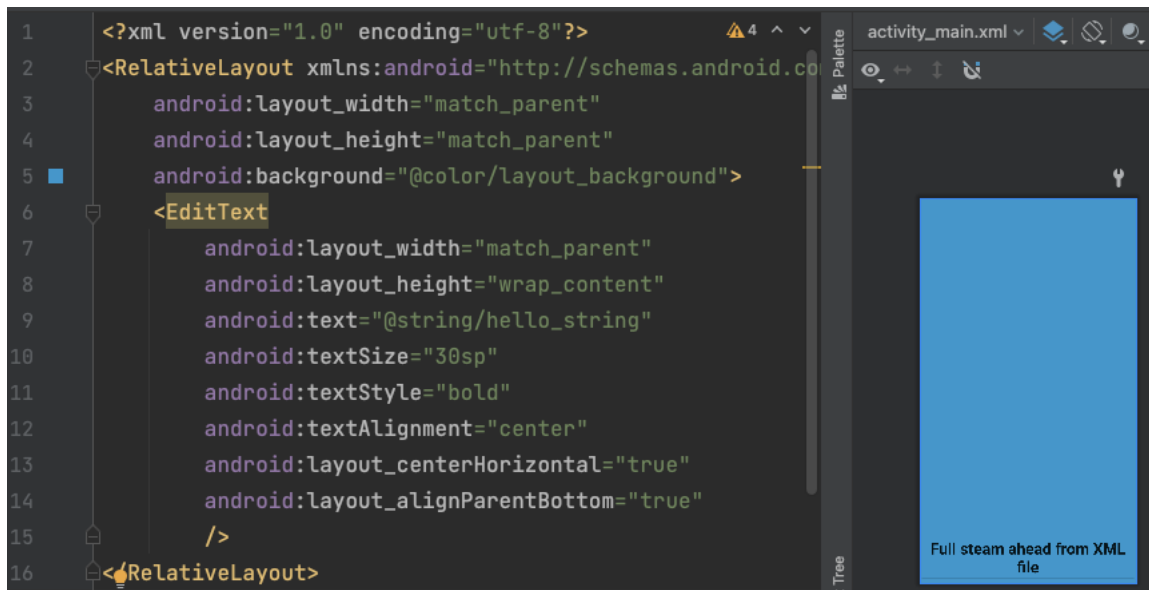
- The text appearing in our EditText has been “hardcoded” in the definition of this label. Generally, when creating an Android application the convention is that strings are defined rather as application resources in the res -> values -> strings.xml file:



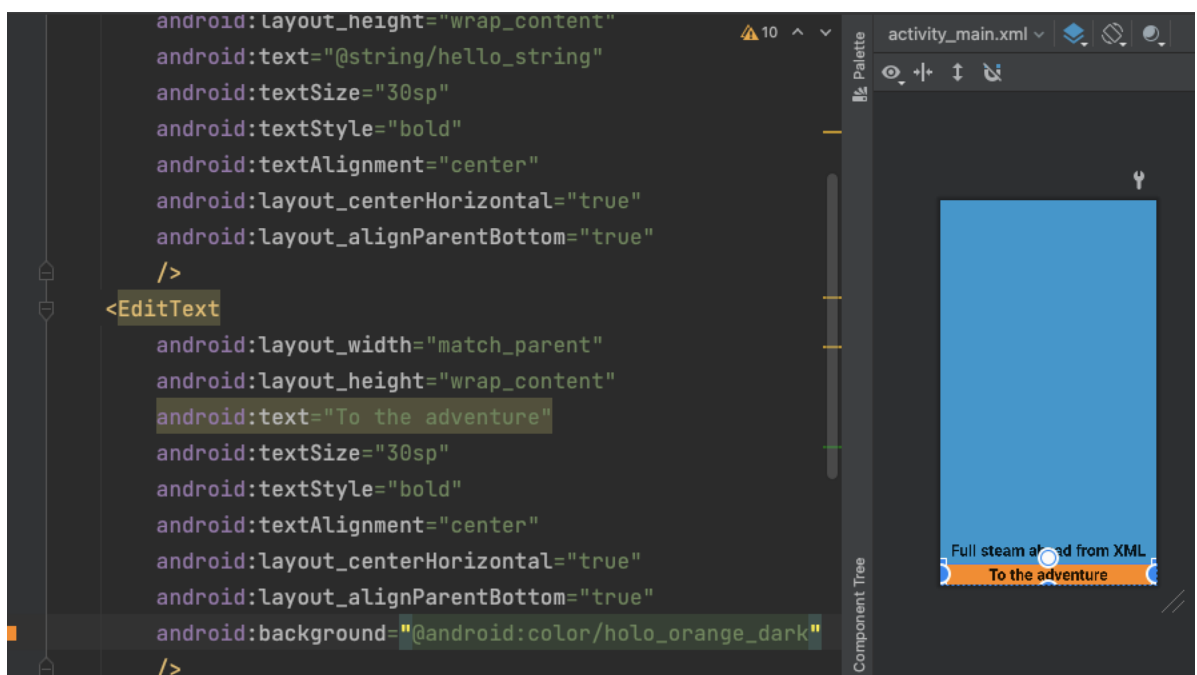
- This is to make the application maintenance easier (if we want to change some strings, we are looking for their definitions in one single file instead of looking for all occurrences throughout many source files. It also facilitates translating the application into various languages.
- So, to define our inscription as the app resource let's add a new "string item in stings.xml file.
- Let it be called hello_string and let's set its value to "Full steam ahead from XML file"



- Now, let's go back to main_activity.xml and let's change the value for the text property from the hardcoded value to @string/hello_string. So the definition and effect should be as below:



- Let's us now add a second EditText to the layout. Let's set the text inside the new label to "to a new adventure". Additionally, let's set the background color to holo_orange_dark. We set the rest of the properties as in the case of the previous EditText
- So now, without any further modifications, the effect should be like this



- As one may see, the labels overlap it is because both editTexts are aligned to the bottom of the "parent"
- Let's fix it now and make the inscription "towards the adventure" to aligned to the bottom of the screen with an additional margin equal to say 20sp (property layout_marginBottom), and the inscription "All Forward" will be placed over the latter
- So, first, we have to add "id" property to the second edit text

```

<EditText
    android:id="@+id/second_edit_text"
    android:layout_width="wrap_content"

```

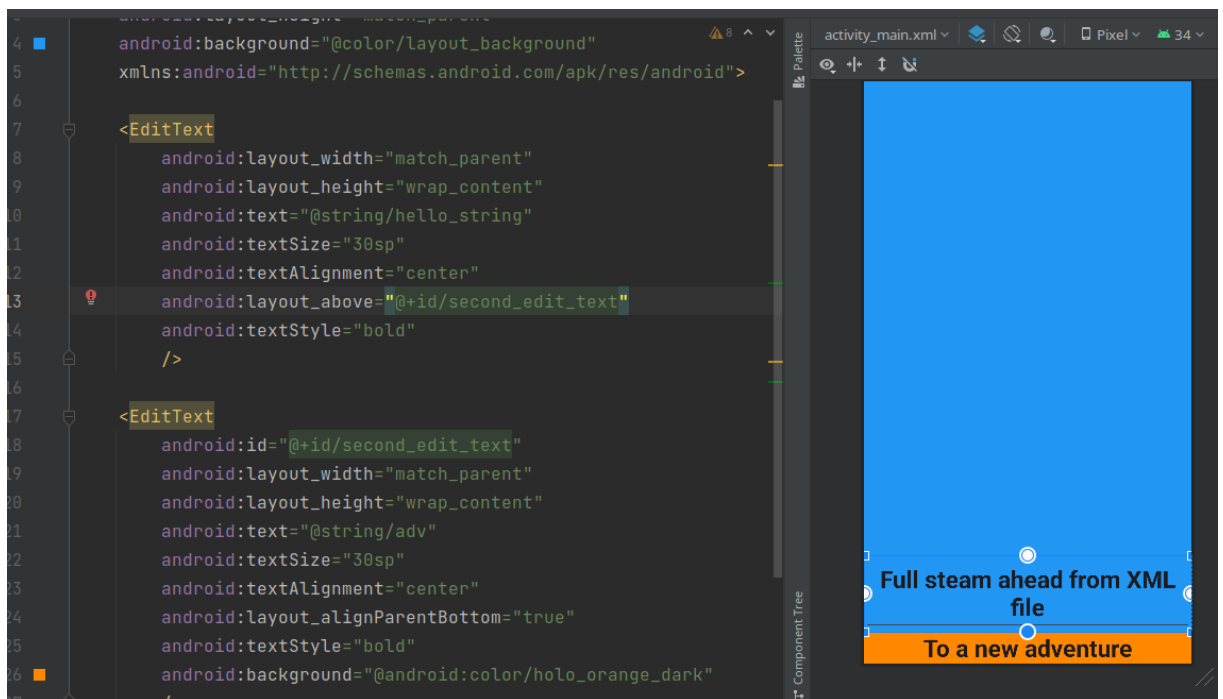
-
- And, now, in the definition of the first edit text, instead of making it aligned to the “parentbottom” we have to set the “layout_above” property as below:

```

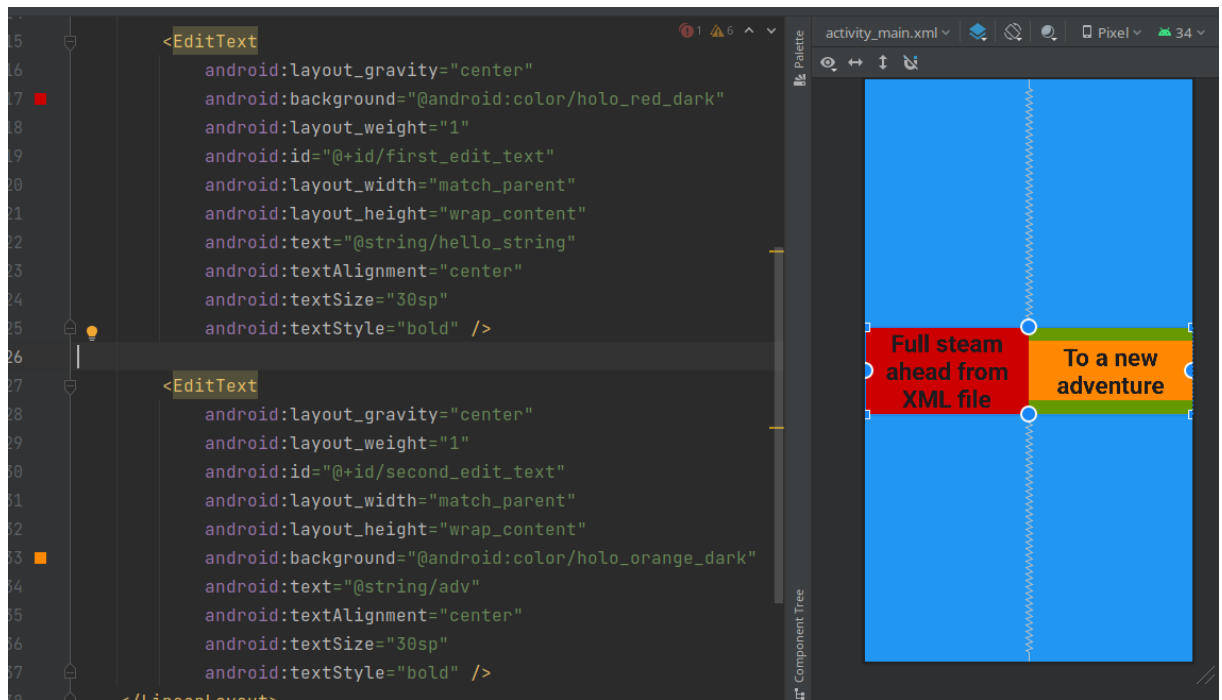
    android:layout_centerHorizontal="true"
    android:layout_above="@id/second_edit_text"

```

- So, now, the editText definitions and the effect should be like this



- For practicing, please place both EditTexts in the layout as below:

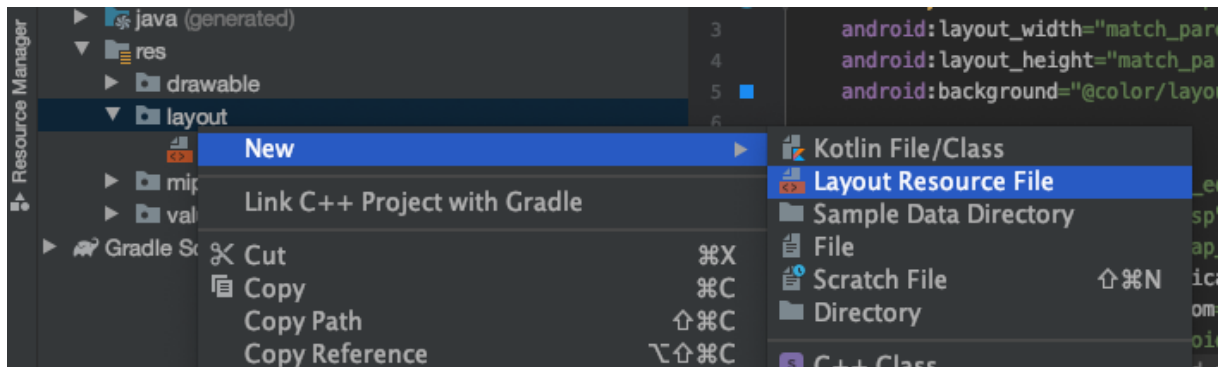


- **Working with new layout and (image) assets/resources**

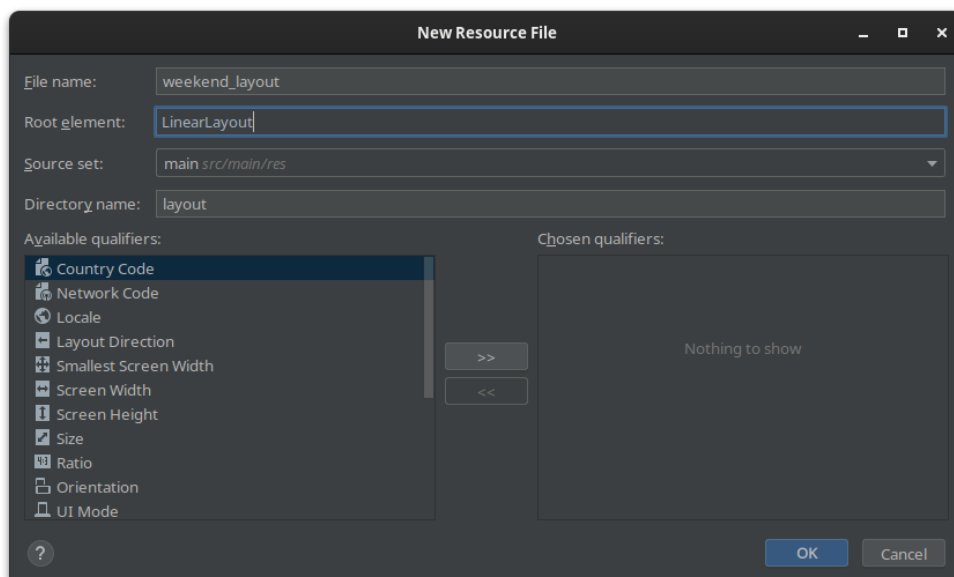
- Now, let's play with new resources, and let's prepare the new layout like the one below.



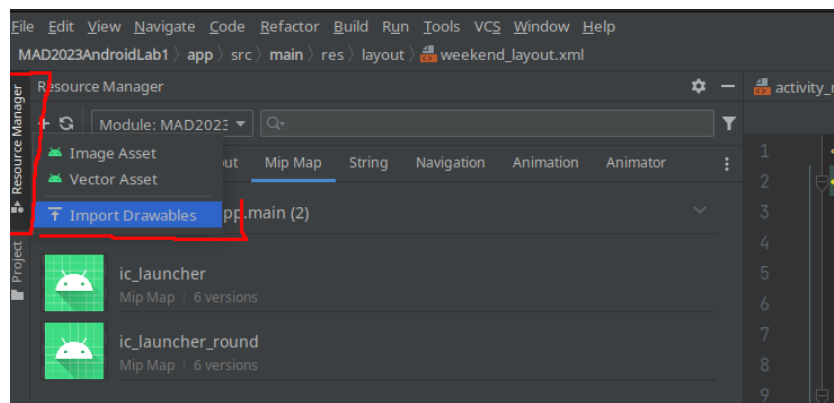
- But we will do it in a new/separate layout file. To do that – pls make a aright click on the layout folder and then choose New -> Layout Resource File



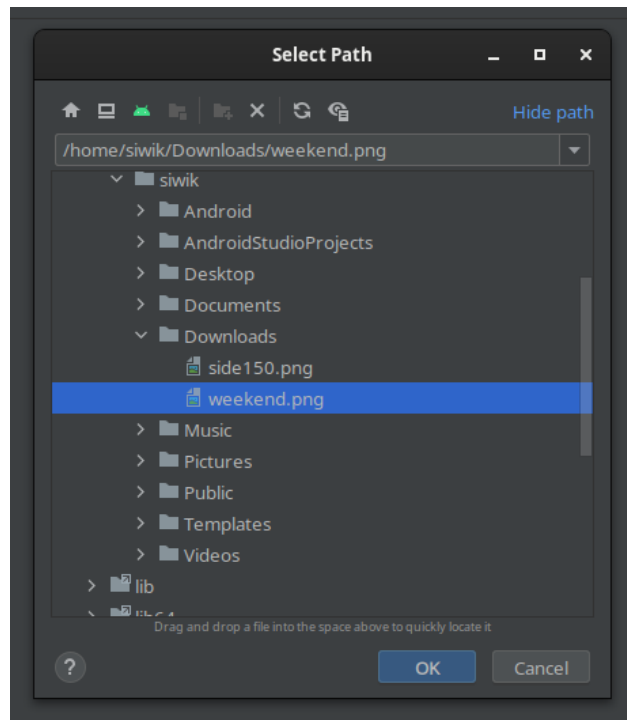
- In the wizard, set the file name to weekend_layout and the root element to LinearLayout



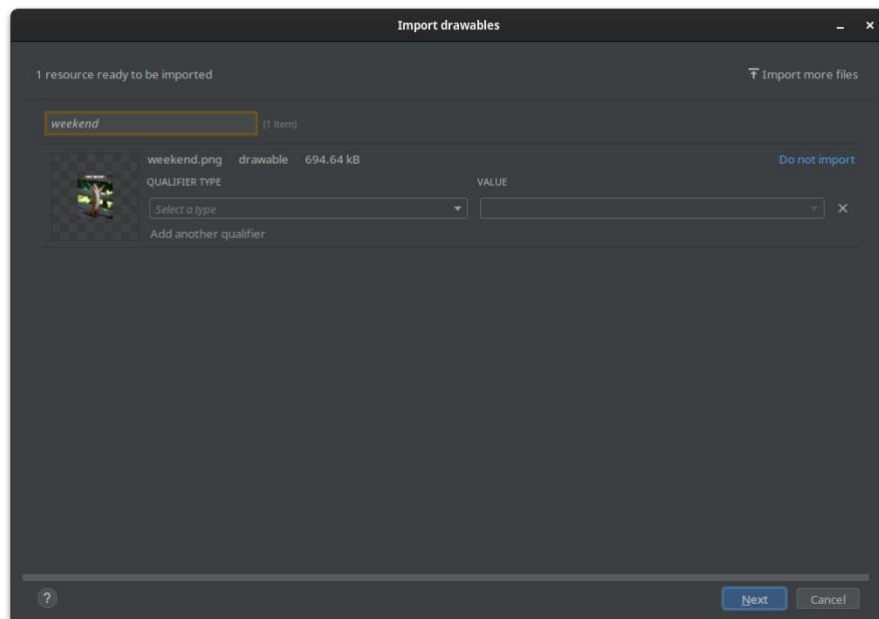
- Now, let's compose the layout out of two elements - EditText and ImageView.
- To set the ImageView content:
 - If you want to use the same picture as I did – you can use the following file: <https://drive.google.com/file/d/1n92485iaf30Tc3ogNEu9Y3LnVd0KG2tu/view?usp=sharing>
 - We need to add a picture to the project - so let's go to Resource Manager, Click on "+" and select "Import Drawables" option



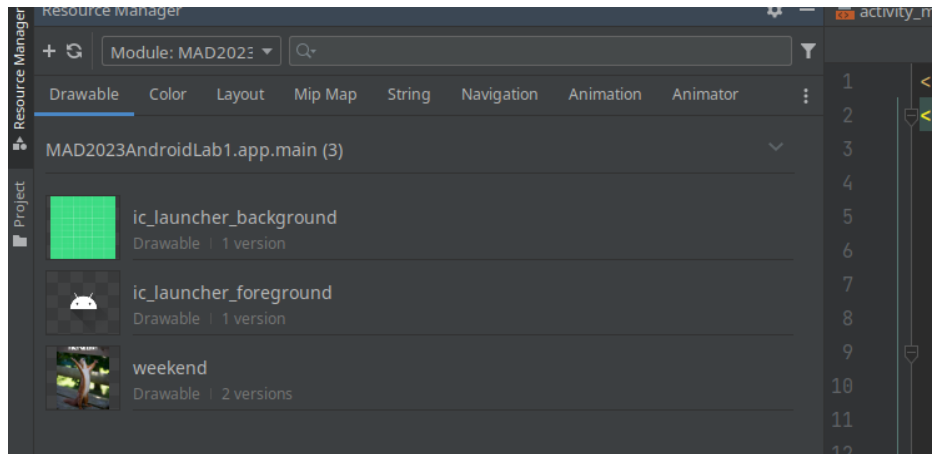
- Lets choose the file we want to import



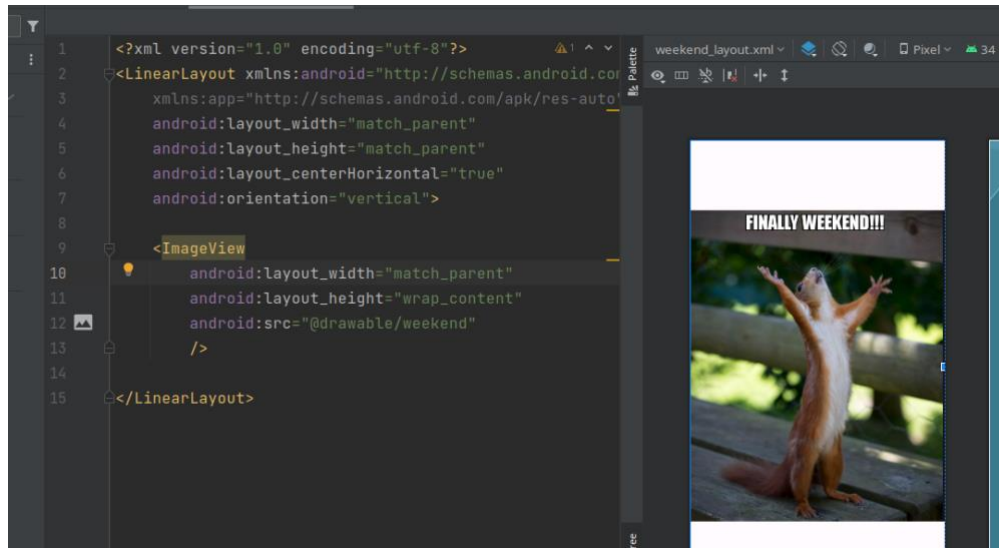
-
- Click ok, then Next



-
- And finally Import, and now the file should be listed as the Drawable in the Resource Manager



- Now, let's add to our new layout the ImageView element, let's set width to match_parent, height to wrap_content, src to @drawable/weekend and we should see our image added to the layout

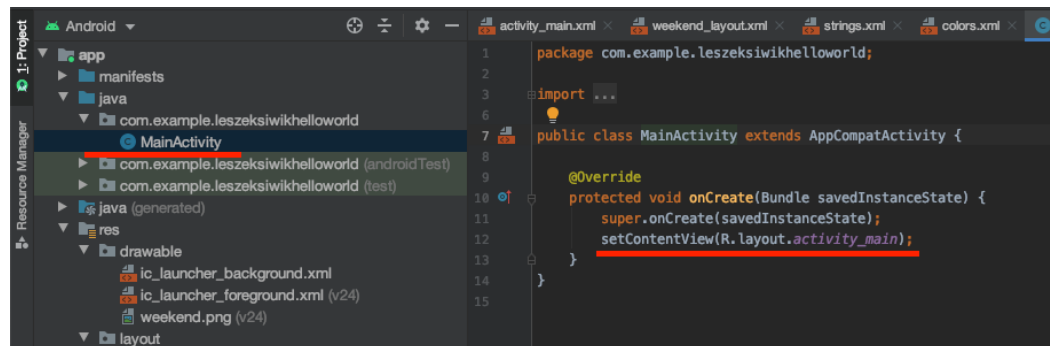


- The problem is that when we run the app we see:



- which is our main_activity layout.

- This is because we didnt say the application to use the new layout. We have to do it in the MainActivity file in the Java directory:

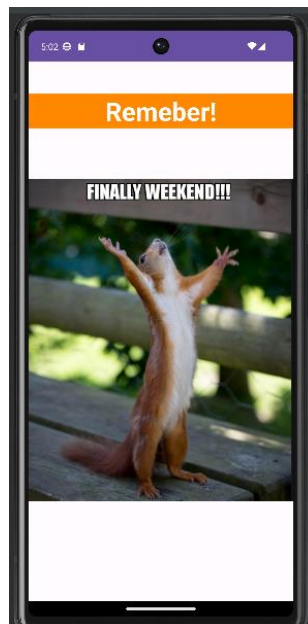


```

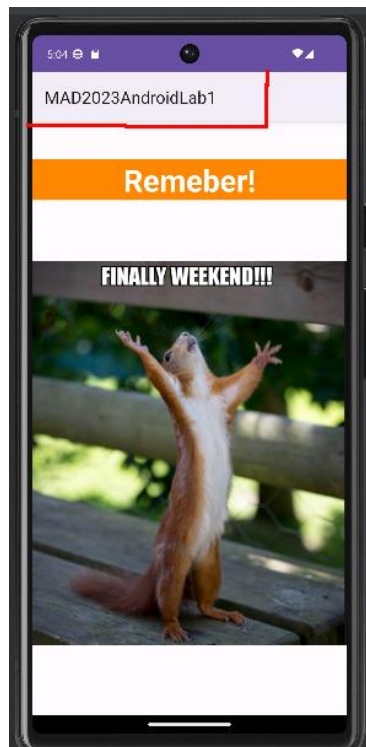
1 package com.example.leszeksiwikhelloworld;
2
3 import ...
4
5
6
7 public class MainActivity extends AppCompatActivity {
8
9
10
11 @Override
12 protected void onCreate(Bundle savedInstanceState) {
13     super.onCreate(savedInstanceState);
14     setContentView(R.layout.activity_main);
15 }

```

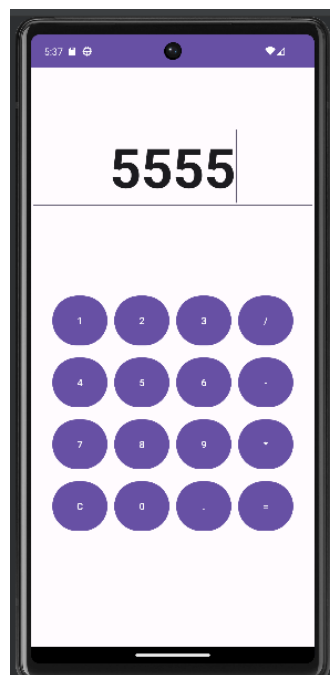
- Here in setContentView method (i.e. for me in line 12) we define which file should be used as the layout definition – if we want to use weekend_layout we have to change activity_main to weekend_layout as the setContentView argument. After that, we should see the following when the app is launched:



- For practicing – pls add the element marked in red below (so-called ActionBar) with the title of the application. To achieve that, we go to: res -> values -> themes.xml file and, in the Style section, we have to add two new items: i.e.: windowActionBar which should be set to true and windowNoTitle set to false.



- Now, please prepare by your own the following UI/layout definition



- When completed pls prepare the short pdf report containing code and app screenshots and upload it as of the Layouts task solution on UPEL platform