Content

Java GUI 2

Different ways to program GUIs in Java 2

AWT (Abstract Window Toolkit) 2

Swing 2

SWT (Standard Widget Toolkit) 2

JavaFX 2

Comparison of their characteristics 2

Scene Builder 3

JFoenix 3

MVC in JavaFx 3

Model 4

View 4

Controller 4

Material we will use: 4

Download Intellij IDEA 4

Descargar JDK 17.0.1 7

Descargar J FoeniX for Java 9 8

Create new project in Intellij IDEA 8

Scene builder contains: 12

Explicación archivo module-info.java 13

Exercise 1: Hello World 14

Explanation of the controller 19

@FXML: 19

Exercise 2: Login 27

Apply css 38

Advantages of using CSS vs. styling elements from the Scene Builder's own options 40

Advantages of using CSS: 40

Disadvantages of using Scene Builder for layouts: 40

Notes 41

Java GUI

A GUI in Java is a visual interface that allows users to interact with an application graphically, rather than using the command line. Instead of typing text commands, users can use buttons, menus, text fields, and other visual elements to interact with the application.

Different ways to program GUIs in Java

AWT (Abstract Window Toolkit)

AWT is an older library that is also part of the Java standard library. It provides basic components for building graphical interfaces, such as buttons and menus, but its functionality is limited compared to Swing and JavaFX.

Swing

Swing is a graphics library that has been part of the Java standard library since JDK 1.2. It provides a number of graphical components, such as buttons, panels, tables and dialog boxes, that you can use to build graphical interfaces.

SWT (Standard Widget Toolkit)

SWT is a toolkit for building graphical interfaces in Java using native operating system components, providing a more consistent look and feel with system applications.

JavaFX

JavaFX is a platform for creating graphical user interface (GUI) applications in Java, introduced in Java 8 to replace Swing as the main GUI library. It provides powerful tools for developing modern, stylized visual interfaces, from desktop to mobile and web applications, using CSS for styling and FXML (an XML language) for interface design.

Comparison of their characteristics:

Aspect	AWT	SWT	Swing	JavaFX
Launch	Early years of Java	Developed by Eclipse Foundation	Introduced in Java 1.2 (1998)	Successor to Swing, launched in 2011
Components	Basic, native to the operating system	Native operating system widgets	More advanced, non-native	Advanced and customizable components
Styles	Limited to OS options	More options, but still limited	Customization with Look & Feel	Full CSS support
Advanced Graphics	No	No	No	Yes (2D and 3D)
Performance	Medium	High	Medium	High
Portability	High	Medium	High	High
Ease of Use	Basic and simple, but less flexible	More advanced, but requires OS knowledge	Más flexible que AWT, pero más complejo que JavaFX	Modern and easy to use with tools such as Scene Builder

We will work with JavaFx

Scene Builder

It is a graphical tool that simplifies the design of interfaces in JavaFX.

It allows you to design user interfaces by dragging and dropping visual components, without the need to write code.

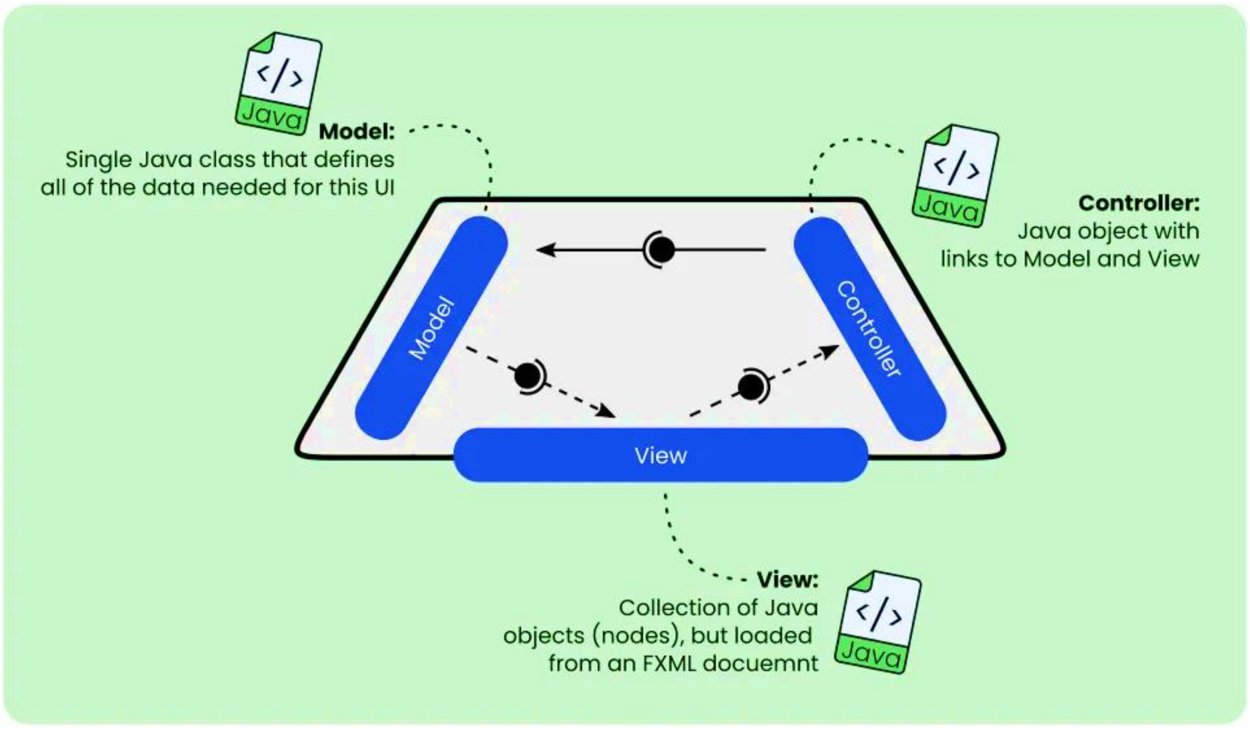
Features:

- **Visual Interface:** Provides an intuitive design environment.
- **Real Time Preview:** Allows you to see how the interface will look like as you design it.
- **Automatic Code Generation:** Generates the FXML code that describes the graphical interface.

JFoenix

It is a component library for JavaFX that provides UI controls with a modern style inspired by Google's Material Design. It is designed to help developers create desktop applications with a more attractive and responsive look and feel. JFoenix extends the standard JavaFX controls, adding new styles and functionality, such as buttons, text boxes, progress bars, lists and dialogs, that follow the visual principles of Material Design.

MVC in JavaFx



Model
The model represents the application data and business logic. It should not have any direct dependencies on the view or controller, thus allowing the business logic to be kept separate from the user interface.

Features:

- **Properties:** Usually JavaFX properties such as StringProperty, IntegerProperty, etc., are used to enable observation and data binding.
- **Validation:** You can include logic to validate data within the model.
- **Persistence:** The model can include methods to persist data in a database or file.

View
The view is responsible for displaying data to the user and providing user interface controls. In JavaFX, you can build the view using JavaFX components in code or in FXML files.

Features:

- **Data Binding:** You can bind model properties directly to view controls.
- **Styling:** You can apply CSS to style the view.

Controller
The controller handles the user interface logic and acts as an intermediary between the model and the view. It responds to view events and updates the model as needed.

Features:

- **Event Handling:** Configures event handlers for view controls.
- **View Update:** Can update the view when the model changes, and vice versa.
- **Model Interaction:** The controller can call methods on the model to obtain or modify data.

Material we will use:
JDK 17.0.1

JavaFX SceneBuilder 22.0.0

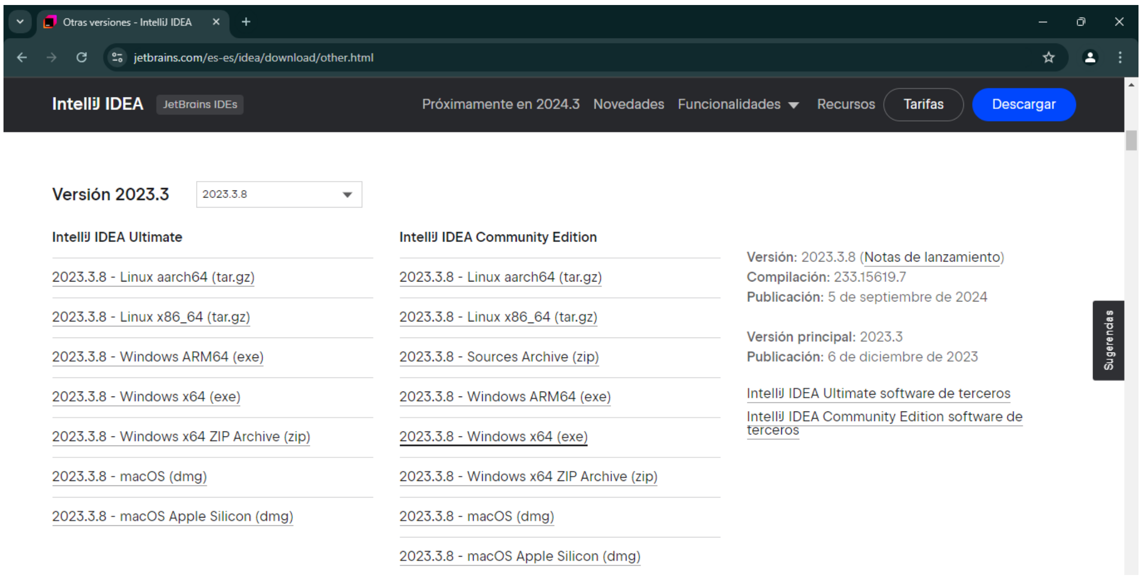
IntelliJ IDEA 2023.3.8

JFoenix 9.0.0

Download Intellij IDEA
Search for Intellij IDEA download or enter the following link

https://www.jetbrains.com/es-es/idea/download/other.html

Search IntelliJ IDEA Community Edition to download the free version, download .exe (Windows)


Otras versiones - IntelliJ IDEA

jetbrains.com/es-es/idea/download/other.html

IntelliJ IDEA JetBrains IDEs Próximamente en 2024.3 Novedades Funcionalidades Recursos Tarifas Descargar

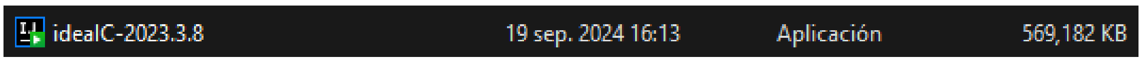
Versión 2023.3 2023.3.8

IntelliJ IDEA Ultimate	IntelliJ IDEA Community Edition
2023.3.8 - Linux aarch64 (tar.gz)	2023.3.8 - Linux aarch64 (tar.gz)
2023.3.8 - Linux x86_64 (tar.gz)	2023.3.8 - Linux x86_64 (tar.gz)
2023.3.8 - Windows ARM64 (exe)	2023.3.8 - Sources Archive (zip)
2023.3.8 - Windows x64 (exe)	2023.3.8 - Windows ARM64 (exe)
2023.3.8 - Windows x64 ZIP Archive (zip)	2023.3.8 - Windows x64 (exe)
2023.3.8 - macOS (dmg)	2023.3.8 - Windows x64 ZIP Archive (zip)
2023.3.8 - macOS Apple Silicon (dmg)	2023.3.8 - macOS (dmg)
	2023.3.8 - macOS Apple Silicon (dmg)

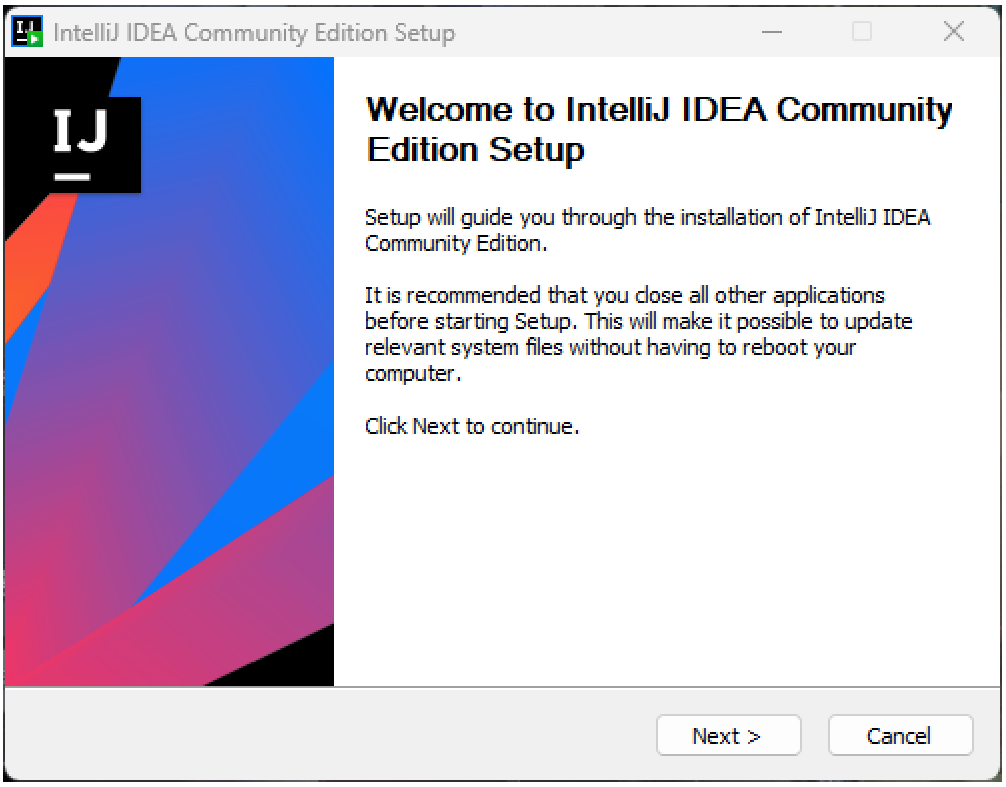
Versión: 2023.3.8 (Notas de lanzamiento)
Compilación: 233.15619.7
Publicación: 5 de septiembre de 2024

Versión principal: 2023.3
Publicación: 6 de diciembre de 2023

IntelliJ IDEA Ultimate software de terceros
IntelliJ IDEA Community Edition software de terceros


idealC-2023.3.8 19 sep. 2024 16:13 Aplicación 569,182 KB

The wirzar opens

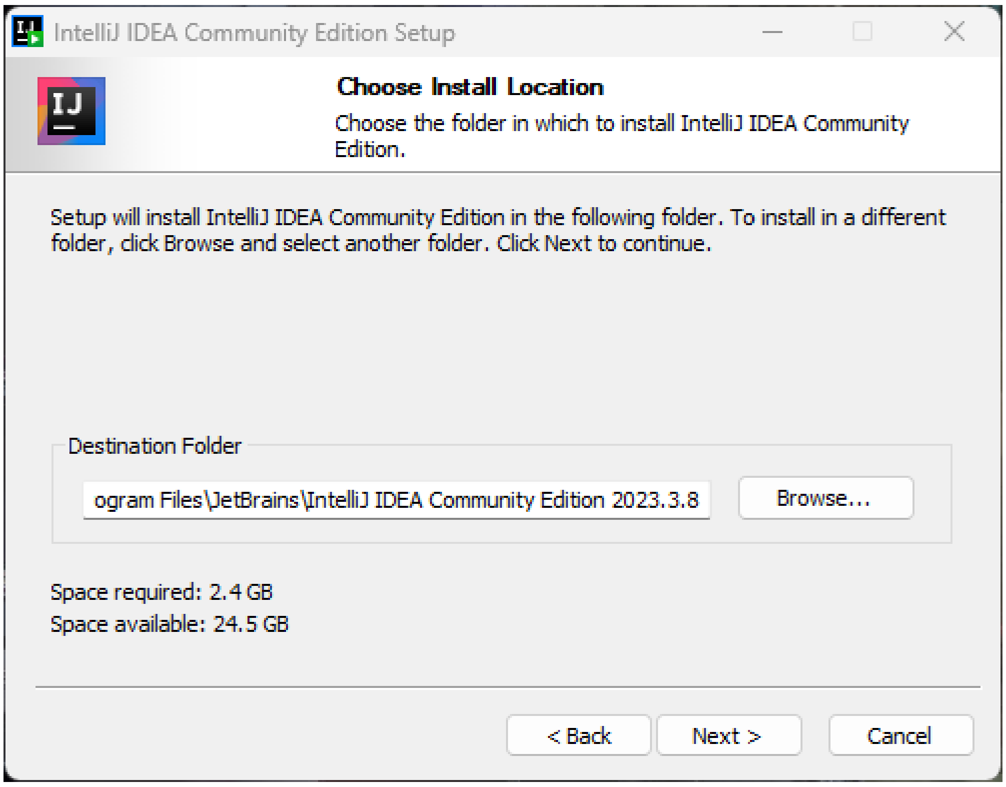

IntelliJ IDEA Community Edition Setup

Welcome to IntelliJ IDEA Community Edition Setup

Setup will guide you through the installation of IntelliJ IDEA Community Edition.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.


IntelliJ IDEA Community Edition Setup

Choose Install Location

Choose the folder in which to install IntelliJ IDEA Community Edition.

Setup will install IntelliJ IDEA Community Edition in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

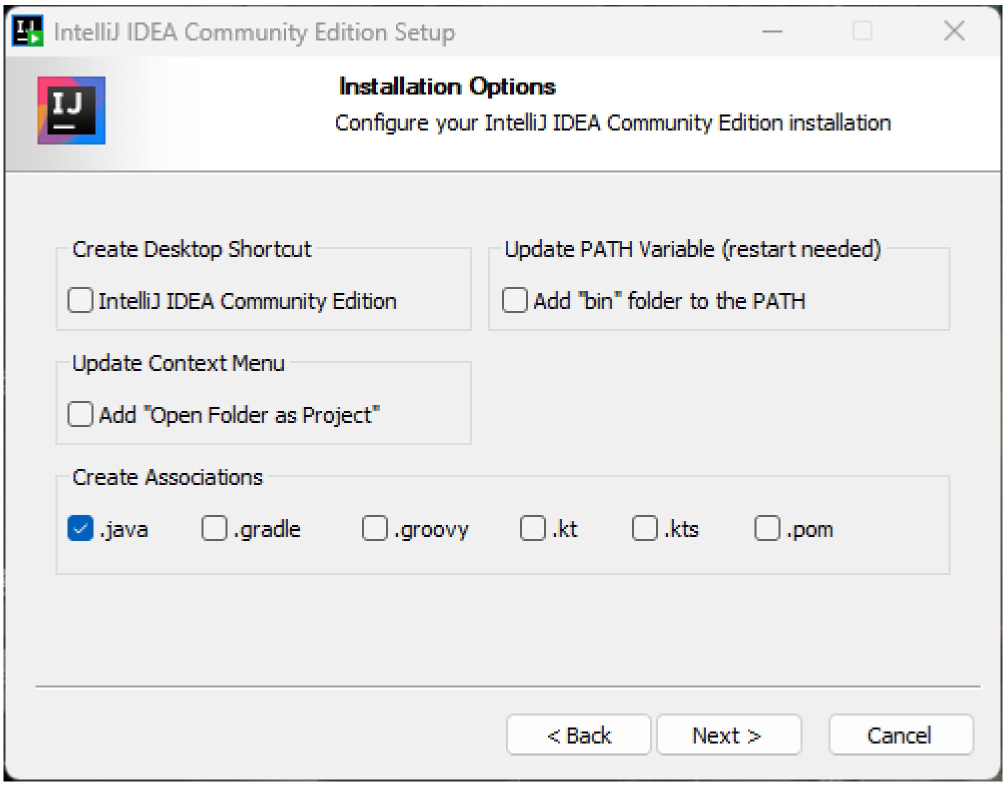
Destination Folder

ogram Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.8

Browse...

Space required: 2.4 GB
Space available: 24.5 GB

< Back | Next > | Cancel


IntelliJ IDEA Community Edition Setup

Installation Options

Configure your IntelliJ IDEA Community Edition installation

Create Desktop Shortcut

☐ IntelliJ IDEA Community Edition

Update PATH Variable (restart needed)

☐ Add "bin" folder to the PATH

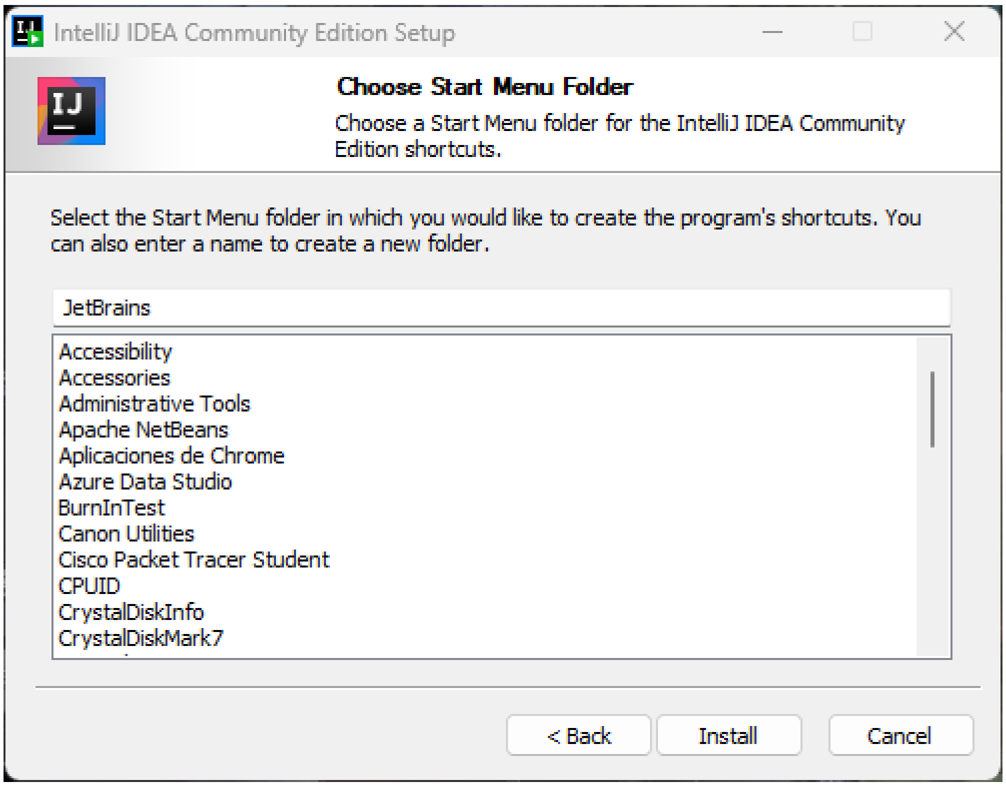
Update Context Menu

☐ Add "Open Folder as Project"

Create Associations

☑ .java ☐ .gradle ☐ .groovy ☐ .kt ☐ .kts ☐ .pom

< Back | Next > | Cancel



IntelliJ IDEA Community Edition Setup

Choose Start Menu Folder

Choose a Start Menu folder for the IntelliJ IDEA Community Edition shortcuts.

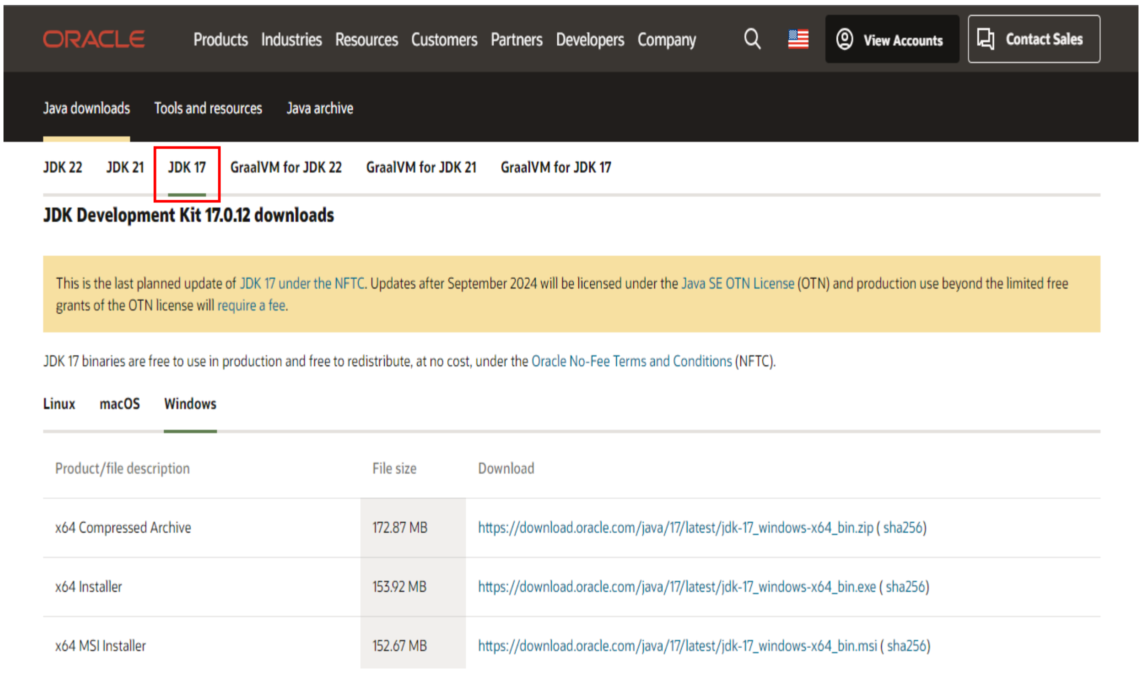
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

JetBrains

Accessibility
Accessories
Administrative Tools
Apache NetBeans
Aplicaciones de Chrome
Azure Data Studio
BurnInTest
Canon Utilities
Cisco Packet Tracer Student
CPUID
CrystalDiskInfo
CrystalDiskMark7

Descargar JDK 17.0.1

https://www.oracle.com/java/technologies/downloads/?er=221886


ORACLE
Products Industries Resources Customers Partners Developers Company
View Accounts
Contact Sales
Java downloads Tools and resources Java archive
JDK 22 JDK 21 JDK 17 GraalVM for JDK 22 GraalVM for JDK 21 GraalVM for JDK 17

JDK Development Kit 17.0.12 downloads

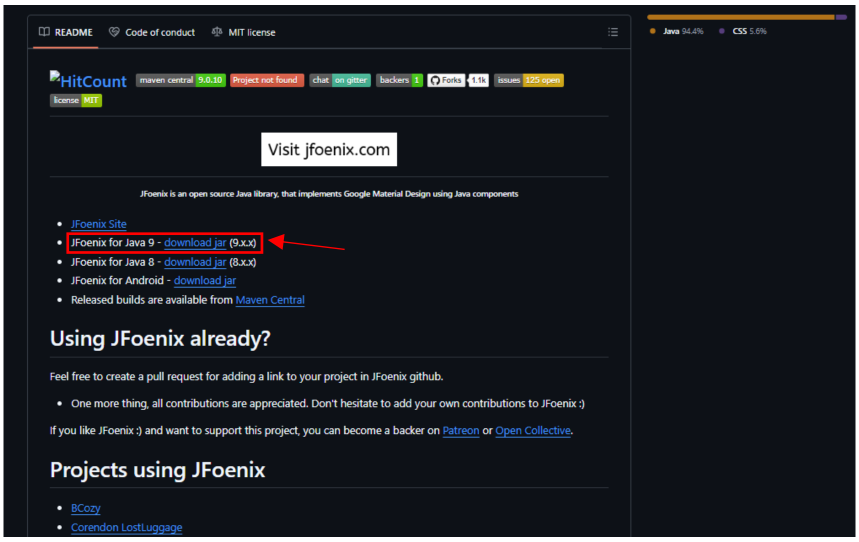
This is the last planned update of JDK 17 under the NFTC. Updates after September 2024 will be licensed under the Java SE OTN License (OTN) and production use beyond the limited free grants of the OTN license will require a fee.

JDK 17 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).

Product/file description	File size	Download
x64 Compressed Archive	172.87 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256)
x64 Installer	153.92 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256)
x64 MSI Installer	152.67 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256)

Descargar J FoeniX for Java 9

https://github.com/sshahine/JFoenix?tab=readme-ov-file



JFoenix is an open source Java library, that implements Google Material Design using Java components

- JFoenix Site
- JFoenix for Java 9 - download jar (9.x.x)
- JFoenix for Java 8 - download jar (8.x.x)
- JFoenix for Android - download jar
- Released builds are available from Maven Central

Using JFoenix already?

Feel free to create a pull request for adding a link to your project in JFoenix github.

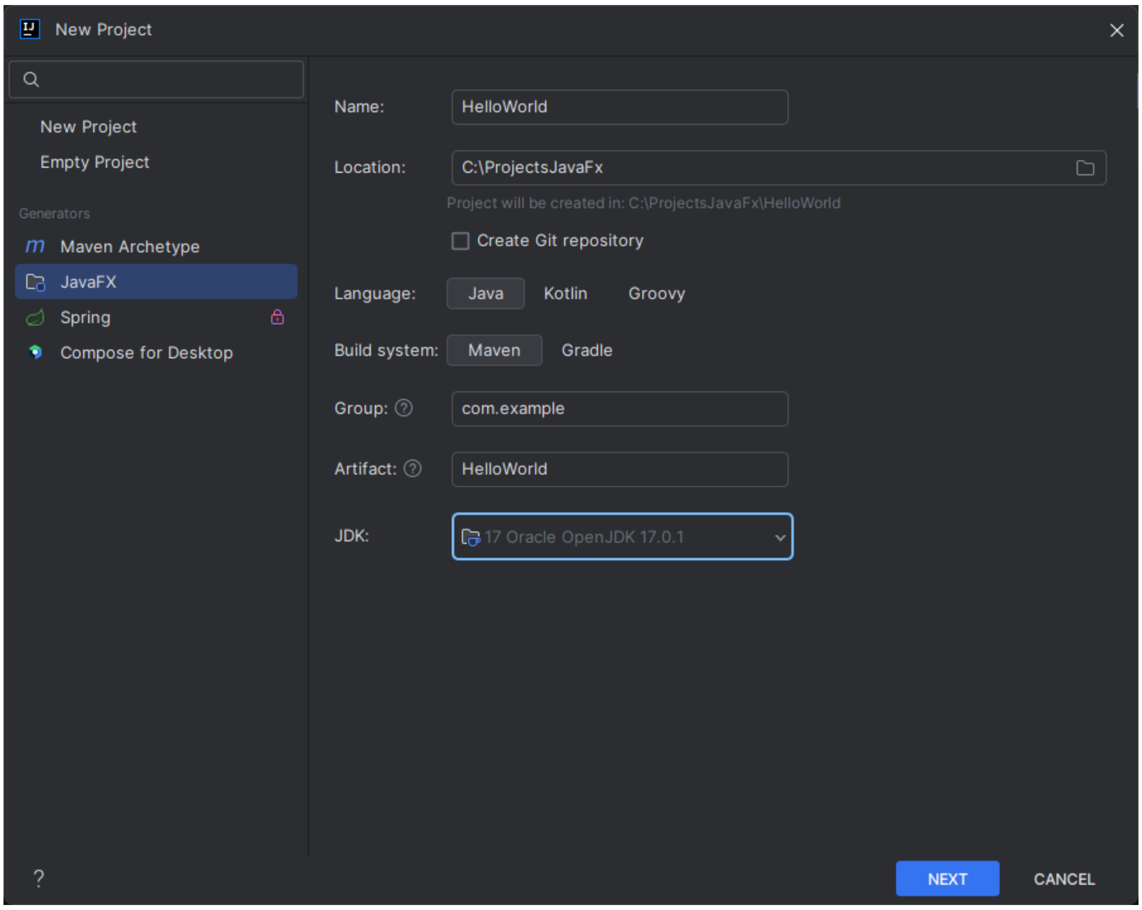
- One more thing, all contributions are appreciated. Don't hesitate to add your own contributions to JFoenix :)

If you like JFoenix :) and want to support this project, you can become a backer on Patreon or Open Collective.

Projects using JFoenix

- BCozy
- Corendon LostLuggage

Create new project in Intellij IDEA



Name: HelloWorld

Location: C:\ProjectsJavaFx

Project will be created in: C:\ProjectsJavaFx\HelloWorld

☐ Create Git repository

Language: [✓ Java] Kotlin Groovy

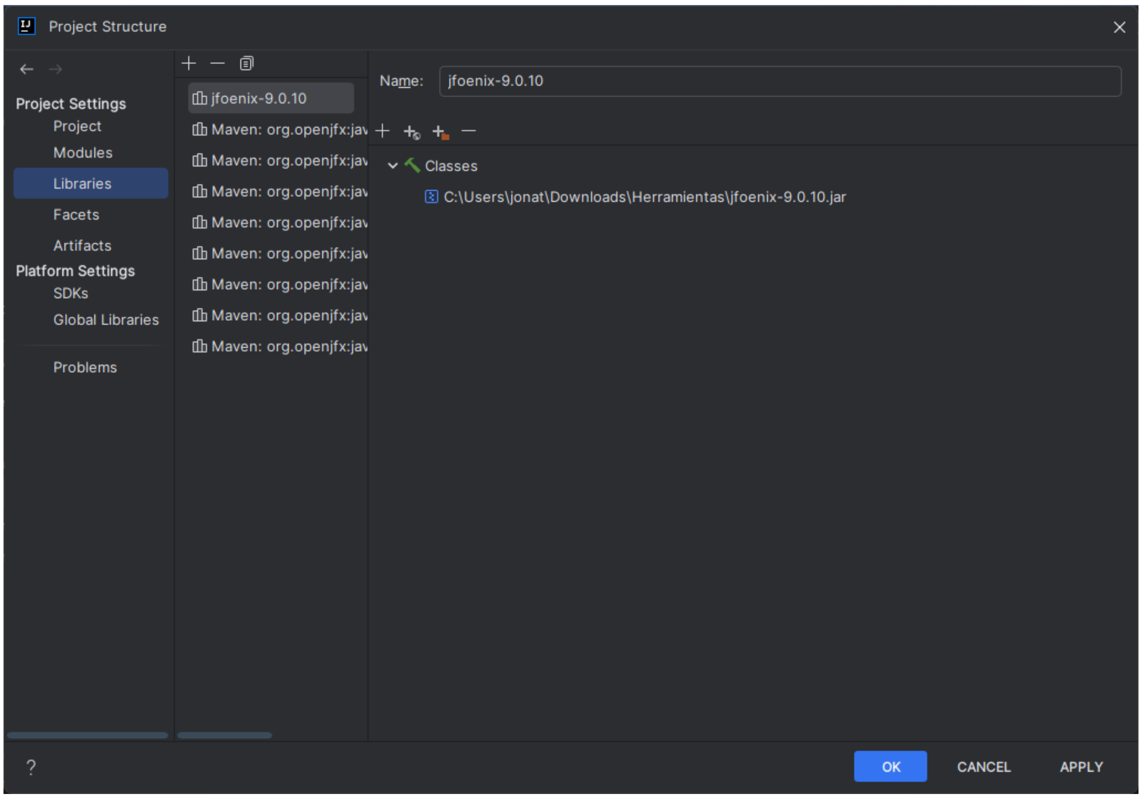
Build system: [✓ Maven] Gradle

Group: com.example

Artifact: HelloWorld

JDK: 17 Oracle OpenJDK 17.0.1

Configure environment to work with scene builderGo to File > Project Structure > Libraries > + > Java > Find the JFoenix .jar file and add it.

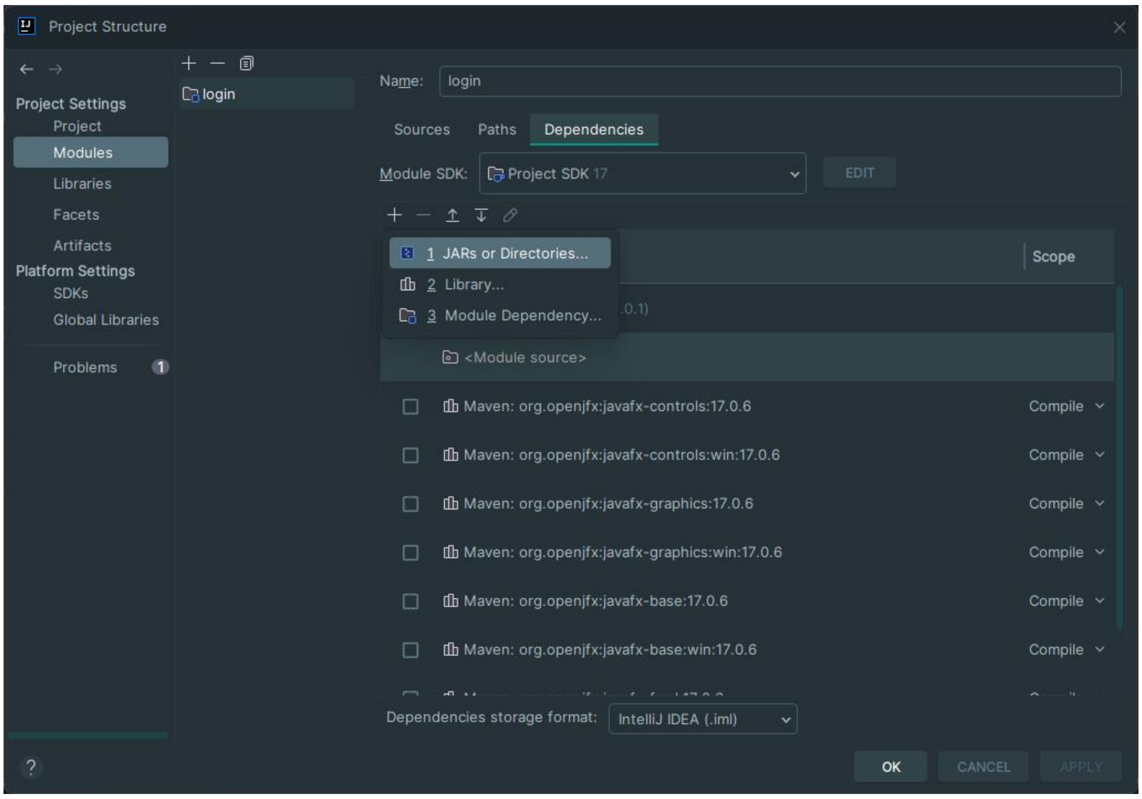

Project Structure

Name: jfoenix-9.0.10

Classes

C:\Users\jonat\Downloads\Herramientas\jfoenix-9.0.10.jar

Go to File > Project Structure > Modules > + > Directory JARs > Browse for the JFoenix .jar file and add it.


Project Structure

Name: login

Sources Paths Dependencies

Module SDK: Project SDK 17

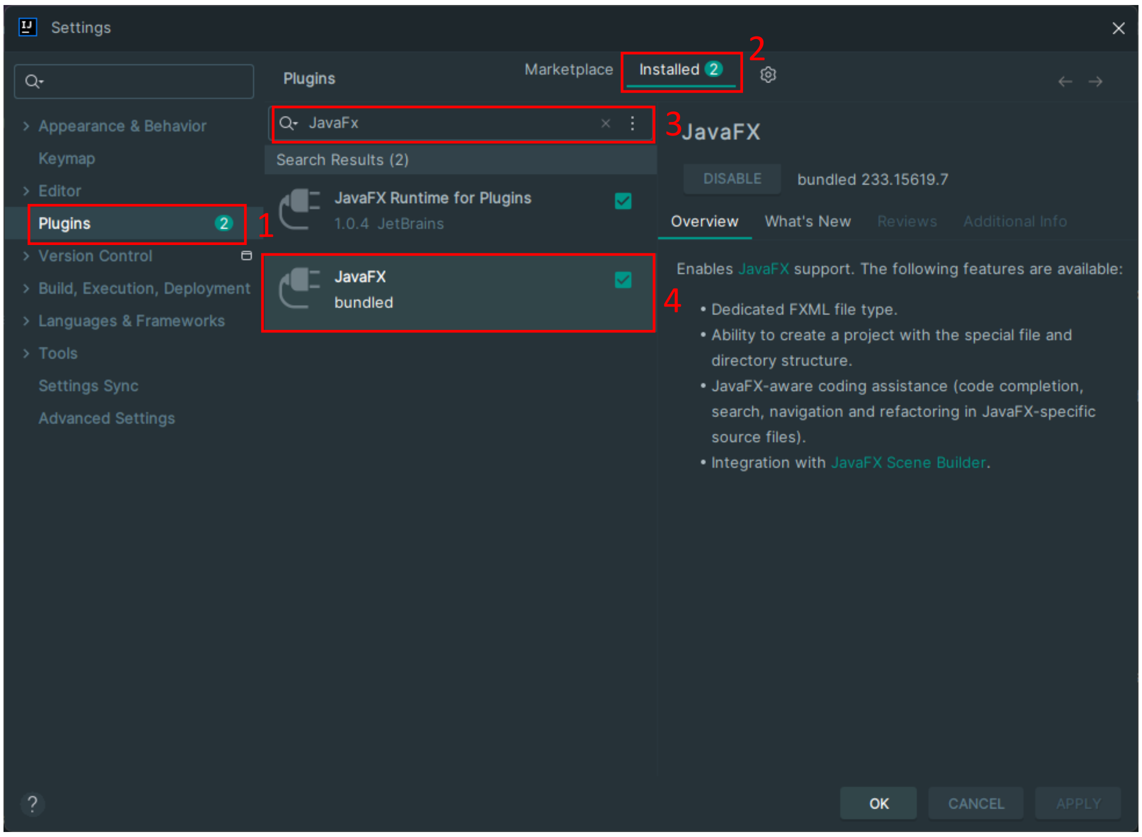
1 JARs or Directories...
2 Library...
3 Module Dependency...

<Module source>

	Scope
Maven: org.openjfx:javafx-controls:17.0.6	Compile
Maven: org.openjfx:javafx-controls:win:17.0.6	Compile
Maven: org.openjfx:javafx-graphics:17.0.6	Compile
Maven: org.openjfx:javafx-graphics:win:17.0.6	Compile
Maven: org.openjfx:javafx-base:17.0.6	Compile
Maven: org.openjfx:javafx-base:win:17.0.6	Compile

Dependencies storage format: IntelliJ IDEA (.iml)

Go to File > Configuration > Plugin > look under installed plugins to see if the Java Fx plugin is installed and enabled



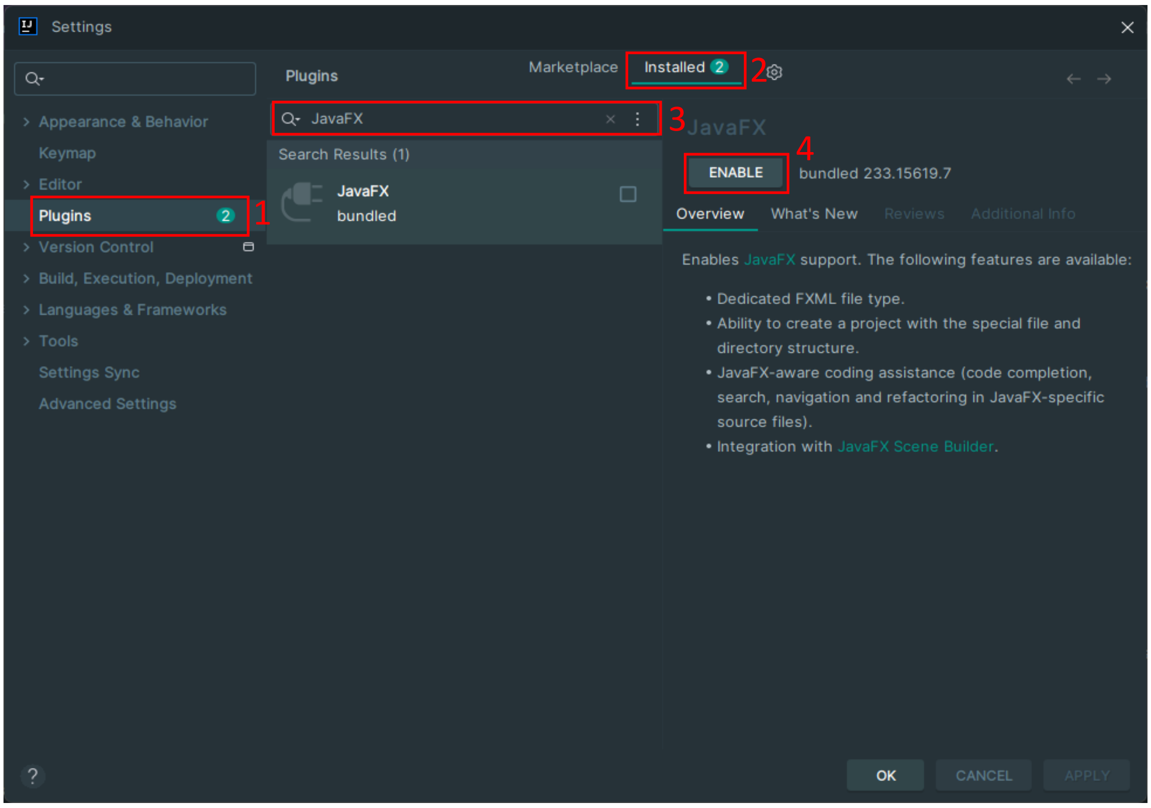
JavaFX

bundled 233.15619.7

Enables JavaFX support. The following features are available:

- Dedicated FXML file type.
- Ability to create a project with the special file and directory structure.
- JavaFX-aware coding assistance (code completion, search, navigation and refactoring in JavaFX-specific source files).
- Integration with JavaFX Scene Builder.

If not enable: Go to File > Configuration > Plugin > installed > Search for JavaFx and enable it



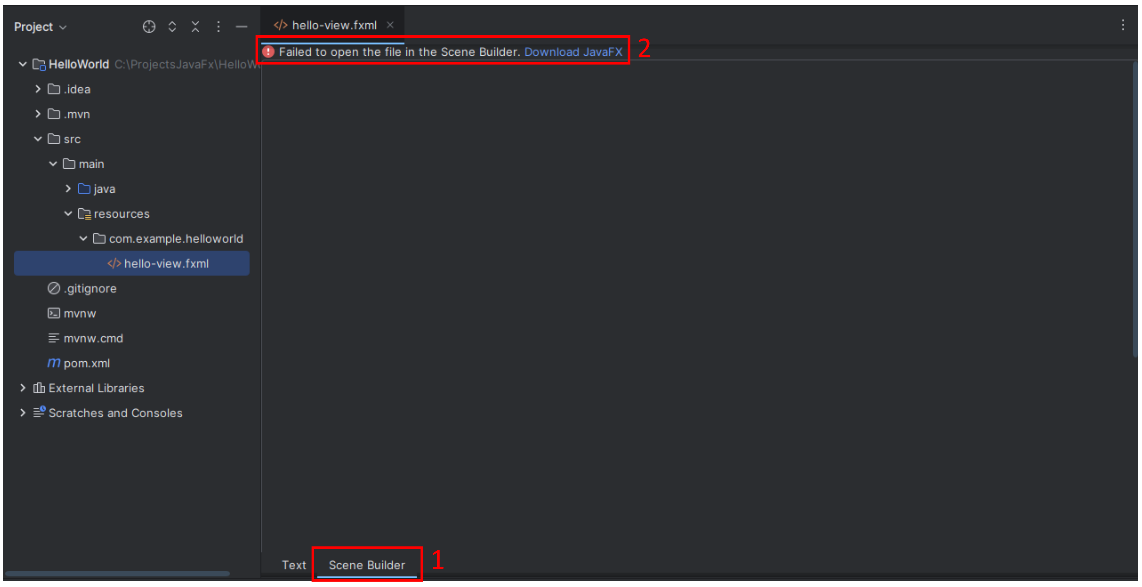
JavaFX

bundled 233.15619.7

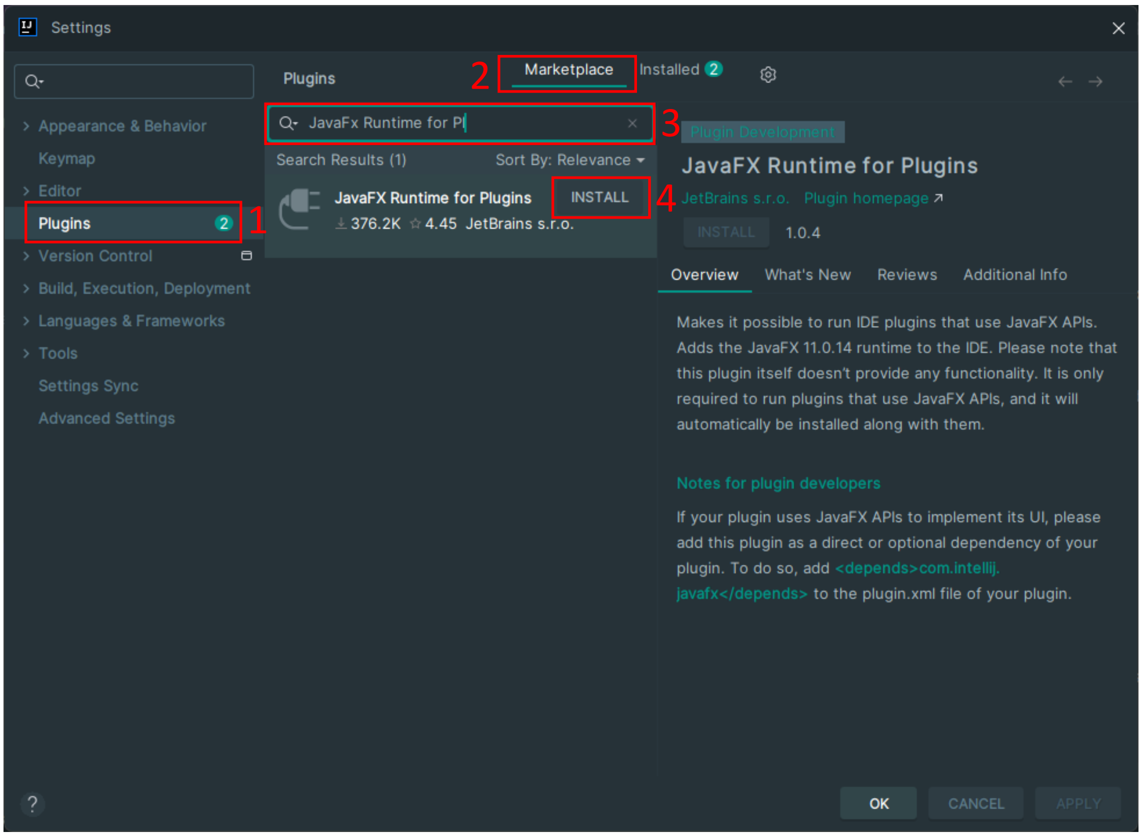
Enables JavaFX support. The following features are available:

- Dedicated FXML file type.
- Ability to create a project with the special file and directory structure.
- JavaFX-aware coding assistance (code completion, search, navigation and refactoring in JavaFX-specific source files).
- Integration with JavaFX Scene Builder.

Open .fxml file go to src/main/resources/hello-view.fxml, at the bottom you will see scene builder when you change it you will get this error



Go to File > Settings > Plugin > download JavaFX Runtime for Plugins -> restart IDE

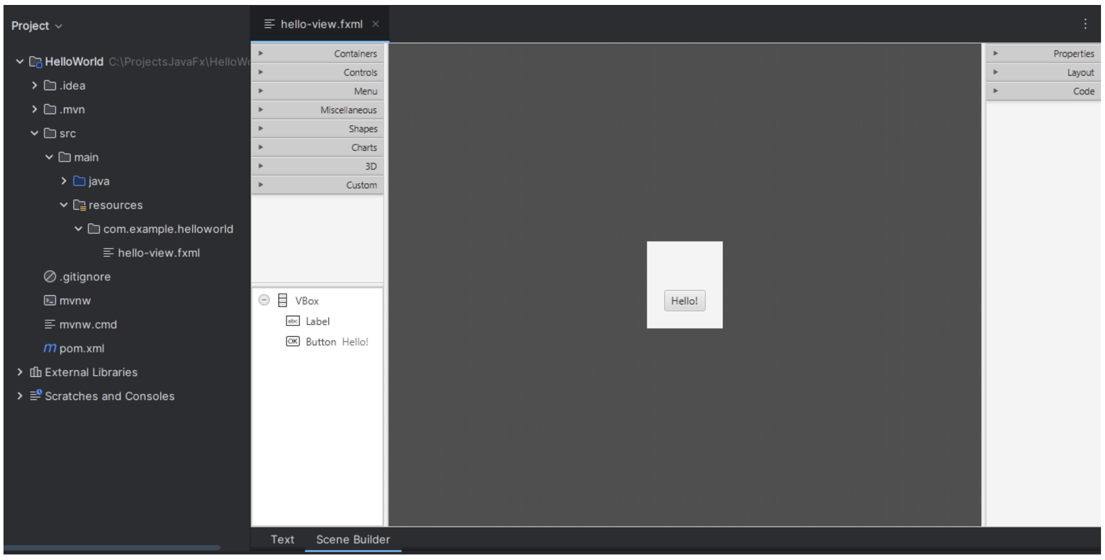


Makes it possible to run IDE plugins that use JavaFX APIs. Adds the JavaFX 11.0.14 runtime to the IDE. Please note that this plugin itself doesn't provide any functionality. It is only required to run plugins that use JavaFX APIs, and it will automatically be installed along with them.

Notes for plugin developers

If your plugin uses JavaFX APIs to implement its UI, please add this plugin as a direct or optional dependency of your plugin. To do so, add <depends>com.intellij.javafx</depends> to the plugin.xml file of your plugin.

Then the error will say download scene builder kit for JavaFx, install it and you are done.



Scene builder contains:

Left:

Containers: Containers used to organize other components, such as VBox, HBox, GridPane, etc.

Controls: User interface controls such as buttons, labels, text fields, etc.

Menu: Menu-related components such as MenuBar and MenuItem.

Miscellaneous: Contains miscellaneous components that do not fall into other categories.

Shapes: Geometric figures such as rectangles, circles, lines, among others, to add shapes to the interface.

Charts: Charts such as bar charts or pie charts.

3D: Components to add 3D objects.

Custom: JFoeniX components.

At the bottom the components that are in use.

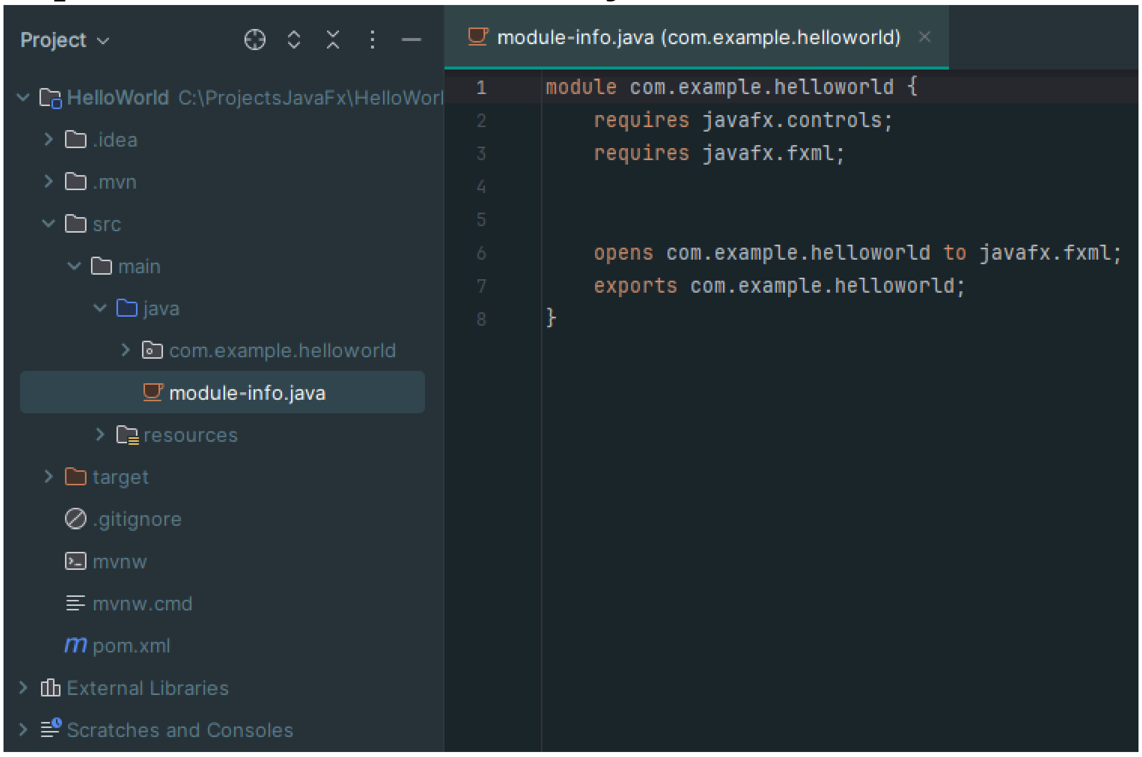
Derecha

Properties: gives direct access to the essential settings of each component, allowing you to quickly customize its behavior and appearance without the need to write additional code.

Layout: Allows you to modify the layout and positioning of a component, such as margins, alignment and size.

Code: Here you can link the component with the Java code, configuring the fx:id, event handlers such as onAction and other code-related properties.

Explicación archivo module-info.java



```java
module com.example.helloworld {
    requires javafx.controls;
    requires javafx.fxml;


    opens com.example.helloworld to javafx.fxml;
    exports com.example.helloworld;
}
```

module-info.java in JavaFX defines the dependencies and visibility of the project:

requires: Specifies the modules you need (such as javafx.controls for the UI and javafx.fxml for FXML files).

opens: Enables reflection access to the com.example.helloworld package (so that JavaFX can load controllers).

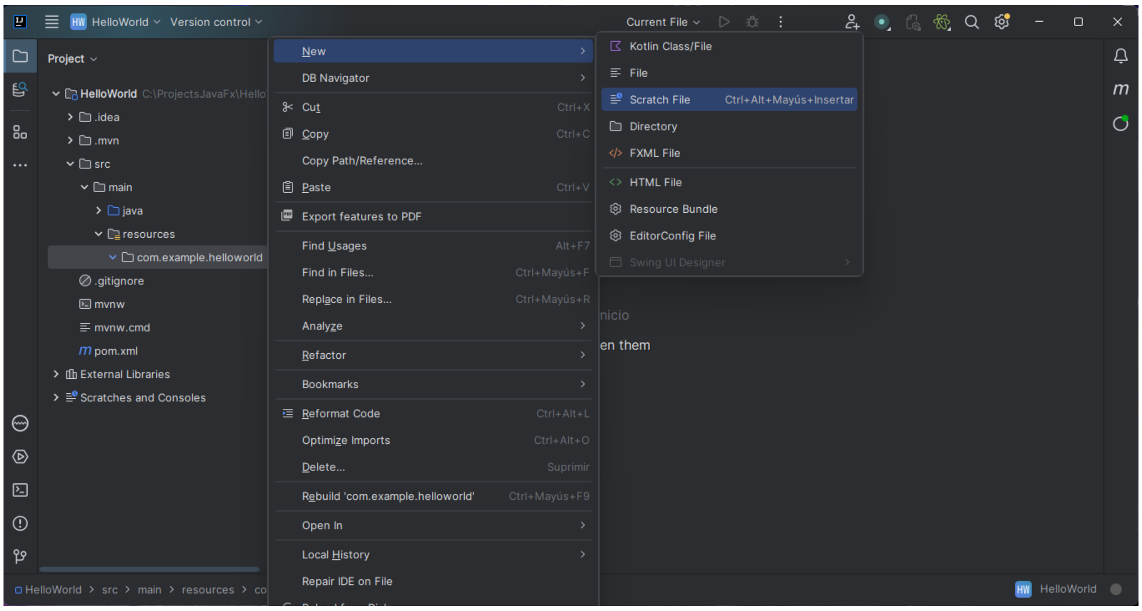
exports: Exposes the com.example.helloworld package so that other modules can use it.

Exercise 1: Hello World

First Hello World in JavaFx

In the previously created exercise we will delete javaresource hello-view.fxml that was created automatically and we will start to do it from 0, to learn

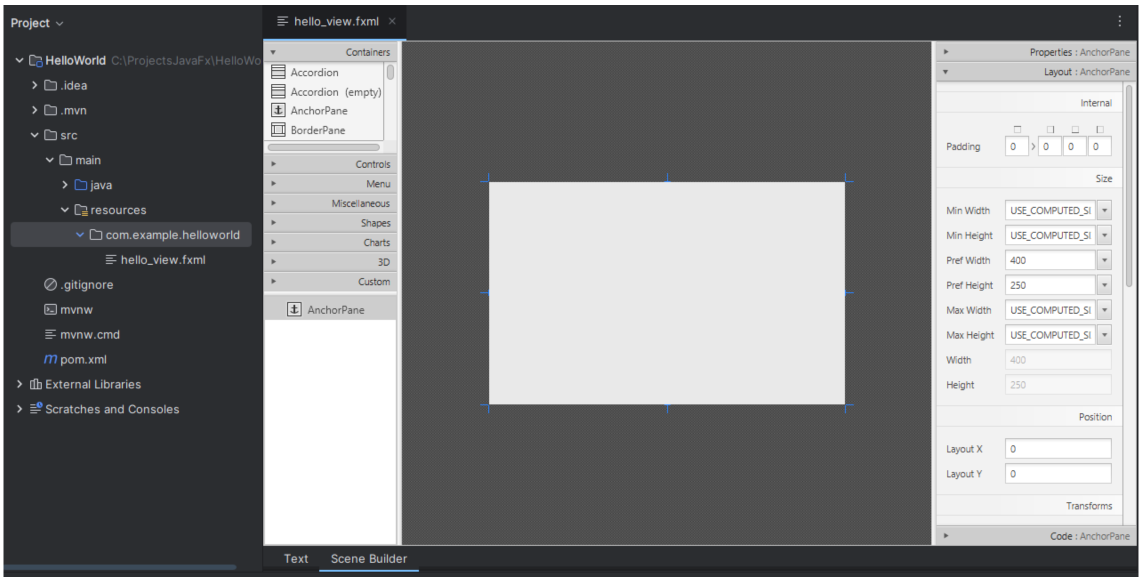
Create a new FXML file and name it hello_view



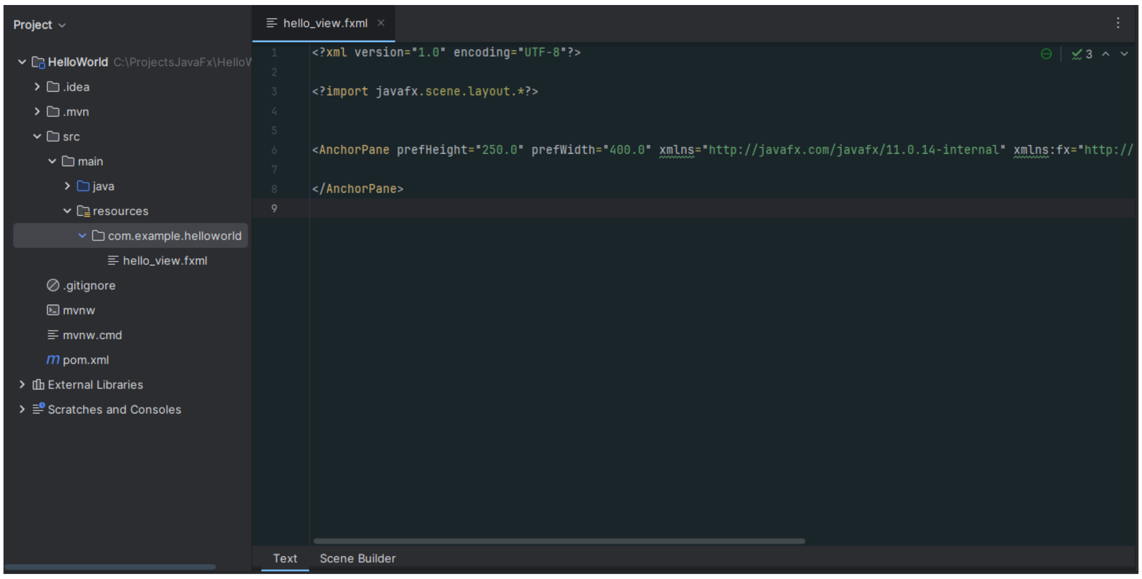
Remove from main “our package” HelloController

Open with scene builder the FXML file that we have just created.

From the configurations of the right, we deploy Layout and we can configure the size of our AnchorPane, we will leave it 400 x 250

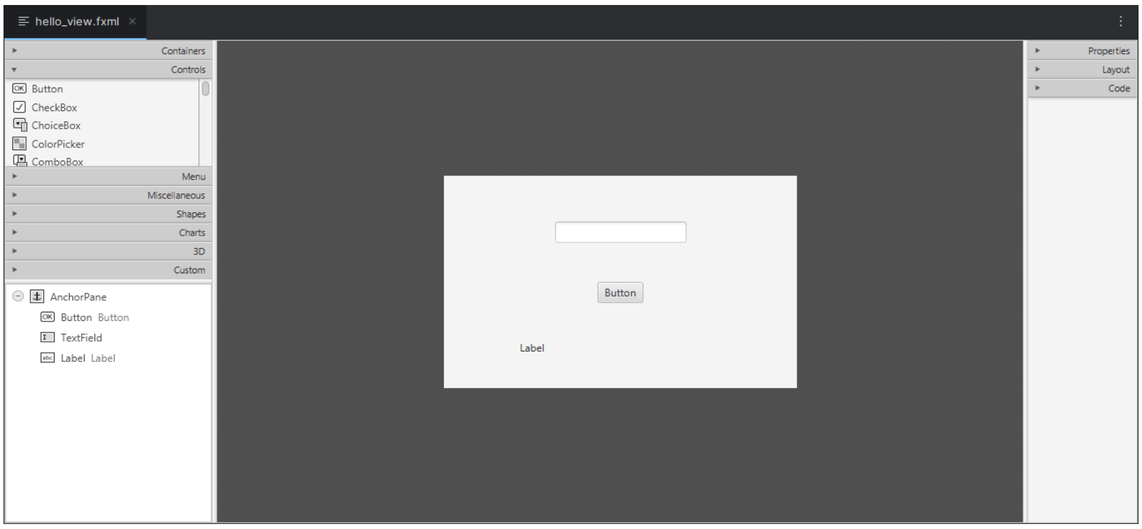


When saving in scene builder it is automatically updated in the intellij fxml.



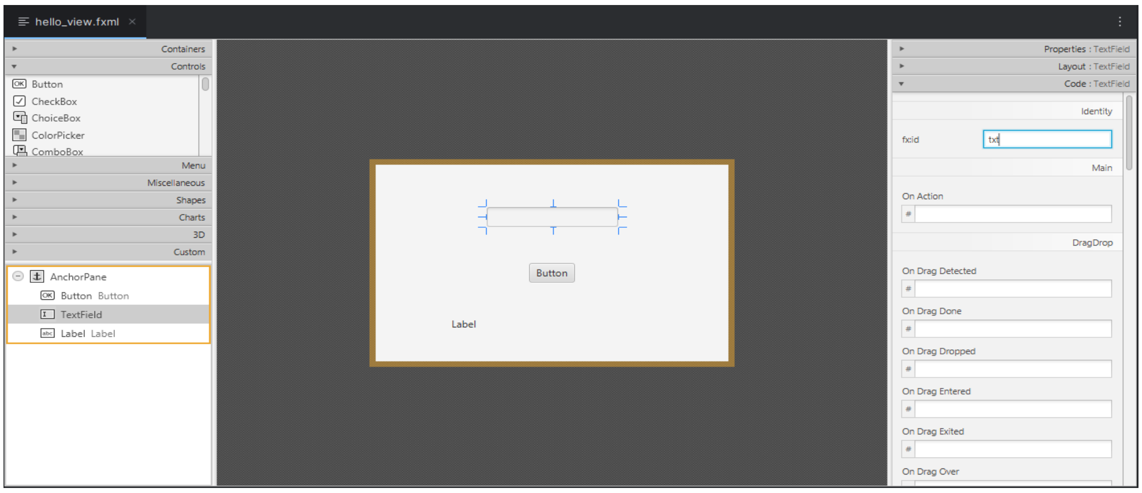
```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.*?>


<AnchorPane prefHeight="250.0" prefWidth="400.0" xmlns="http://javafx.com/javafx/11.0.14-internal" xmlns:fx="http://

</AnchorPane>
```

We will add 3 components, 1 button, 1 label and 1 txtField that are in controls



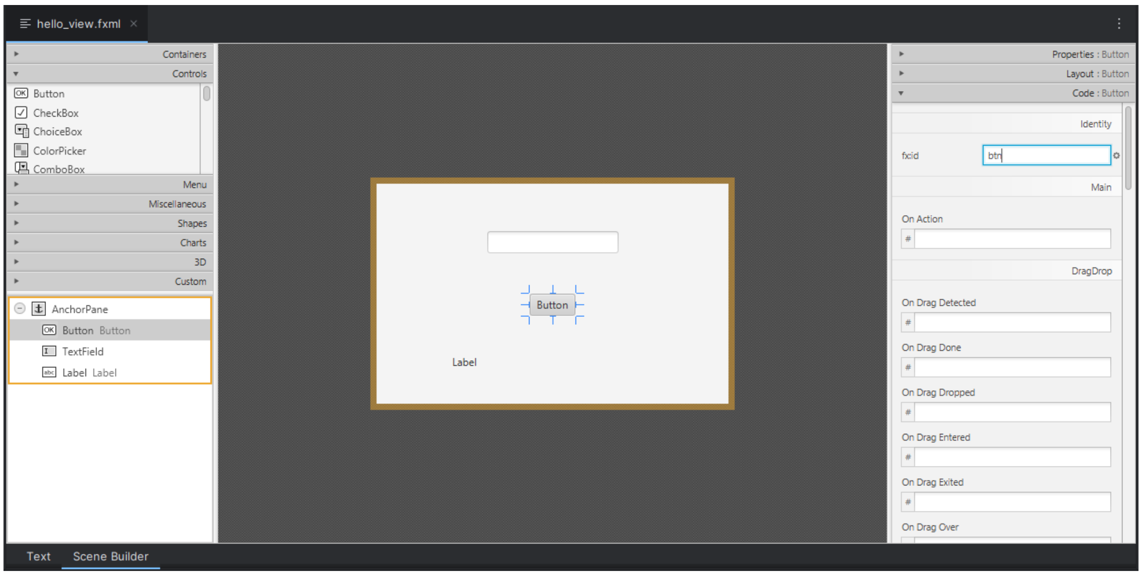
To each component we put an id, click on the component and then in the configurations on the right, in code we add the id

TxTfield



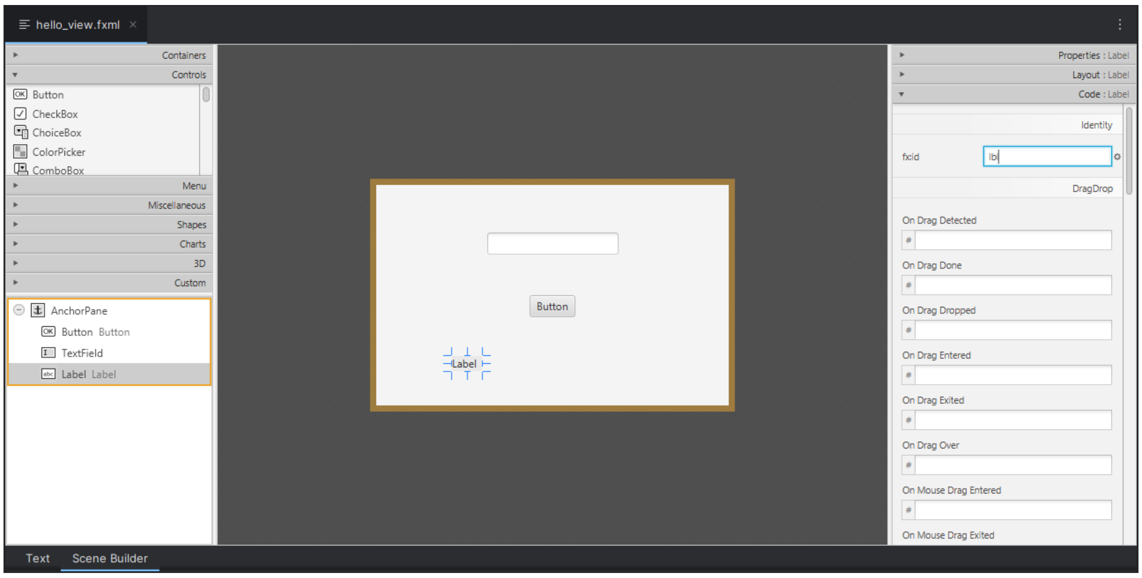
fx:id txt

Button



fx:id btn

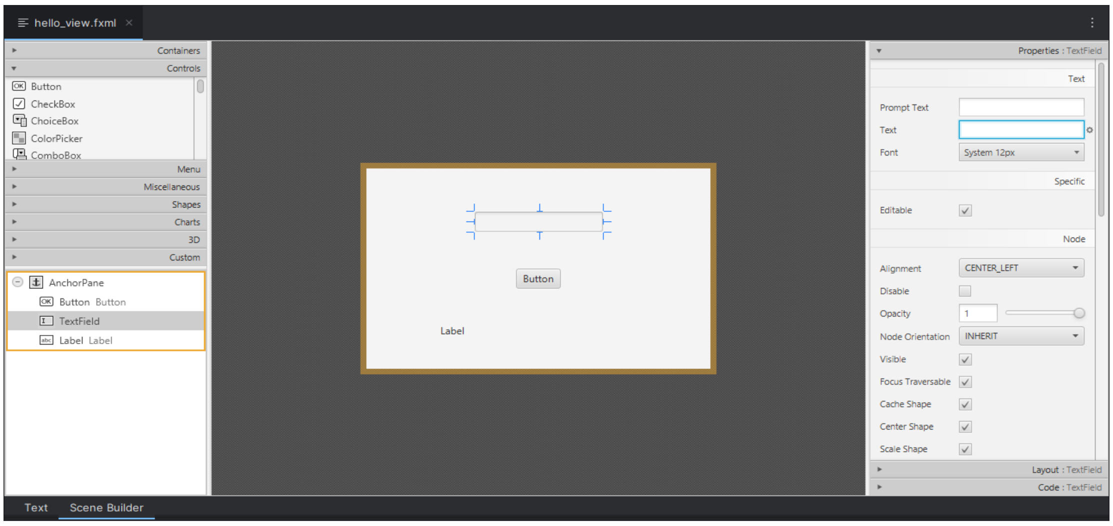
Label



fx:id lbl

In properties we configure what we want to show initially

The TxtField is left empty.



Properties : TextField

Text

Prompt Text
Text
Font System 12px

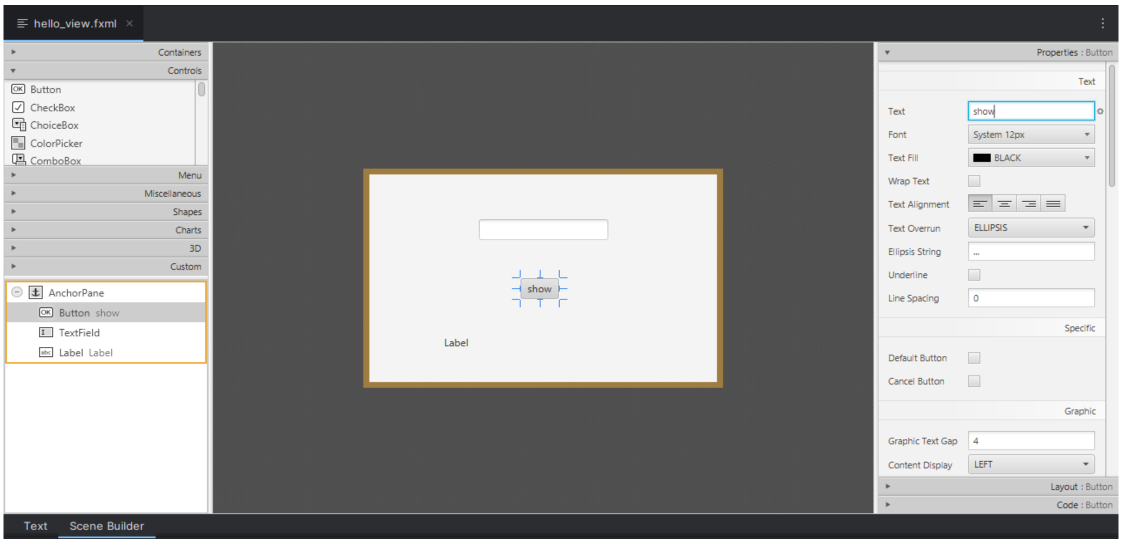
Specific

Editable ✓

Node

Alignment CENTER_LEFT
Disable
Opacity 1
Node Orientation INHERIT
Visible ✓
Focus Traversable ✓
Cache Shape ✓
Center Shape ✓
Scale Shape ✓

To the button we add show



Properties : Button

Text

Text show
Font System 12px
Text Fill BLACK
Wrap Text
Text Alignment
Text Overrun ELLIPSIS
Ellipsis String ...
Underline
Line Spacing 0

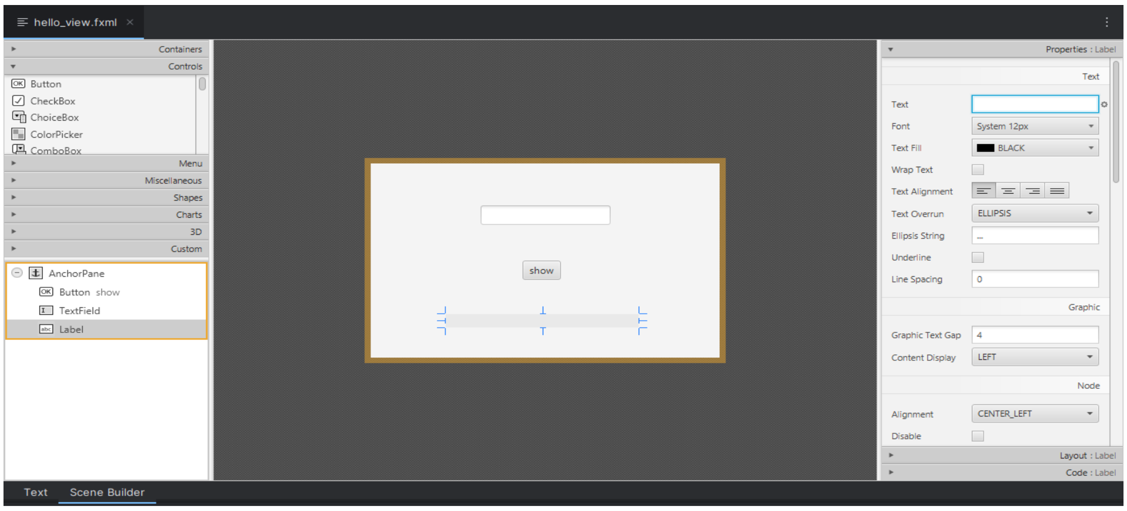
Specific

Default Button
Cancel Button

Graphic

Graphic Text Gap 4
Content Display LEFT

And the label is left empty



Properties : Label

Text

Text
Font System 12px
Text Fill BLACK
Wrap Text
Text Alignment
Text Overrun ELLIPSIS
Ellipsis String ...
Underline
Line Spacing 0

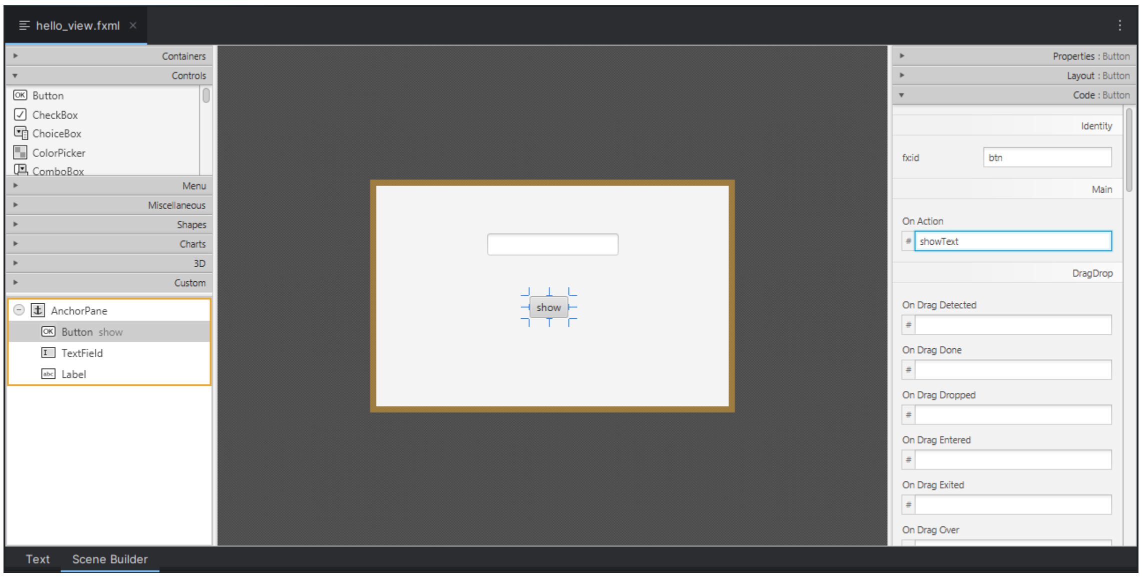
Graphic

Graphic Text Gap 4
Content Display LEFT

Node

Alignment CENTER_LEFT
Disable

We add action to the button in the settings on the right in code > On Action, to tell it what function to call when clicked.

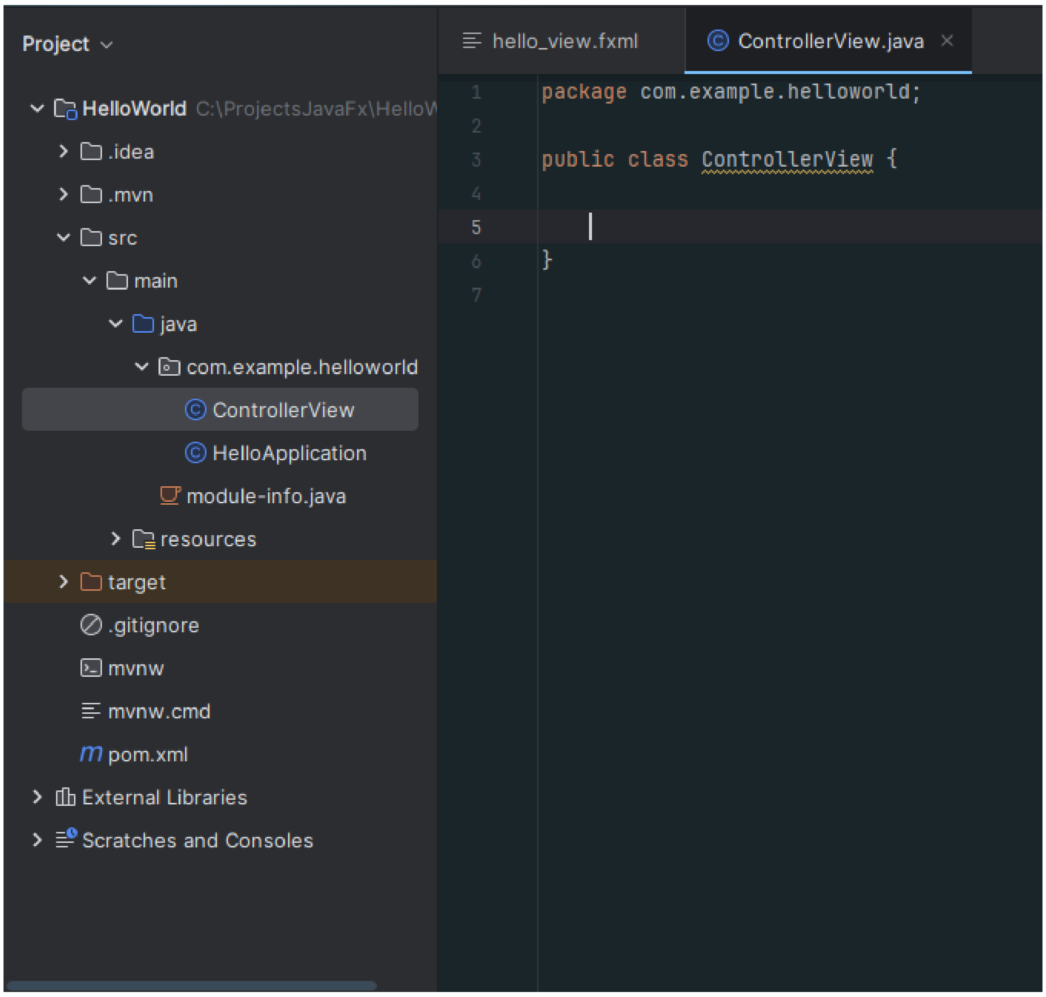


fx:id: btn

On Action: # showText

We finish the graphical part, now we will finish the configuration, following the MVC

We create a class called ControllerView in mainjava.example.helloworld



```java
package com.example.helloworld;

public class ControllerView {


}

```

Explanation of the controller

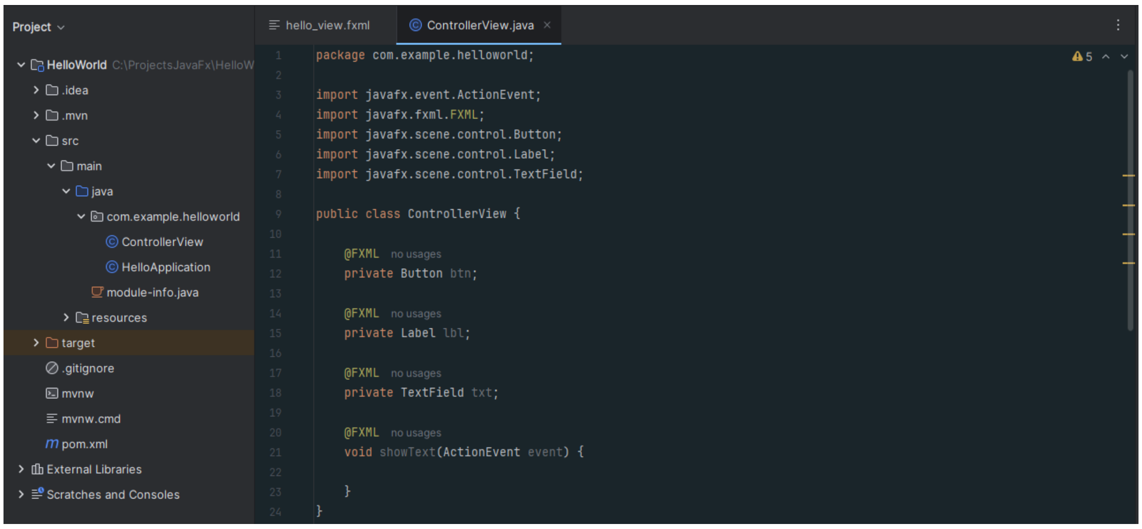
It is the bridge between the visual design described in the .fxml file and the Java application logic.

@FXML:

This annotation indicates that the field or method is accessible from the FXML file. The controller and the FXML file are linked through this annotation.

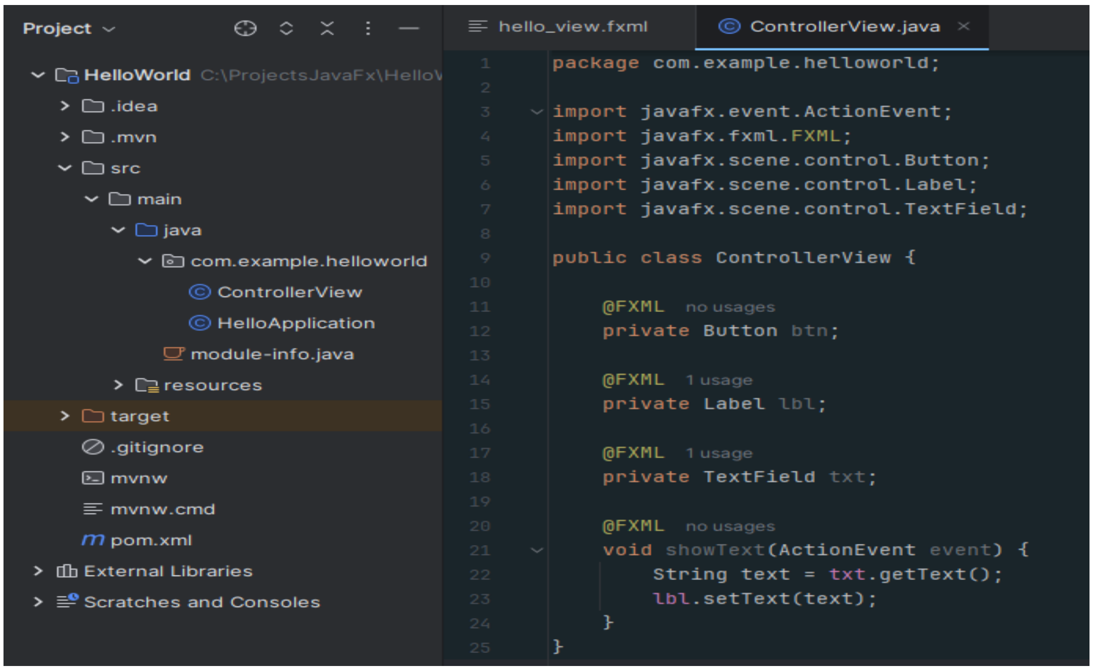
In attributes: When an attribute is annotated as @FXML, it means that this field refers to a GUI component that is declared in the FXML file, in methods it indicates that this method can be called directly from events defined in the FXML file, such as a button click.

Controller configuration



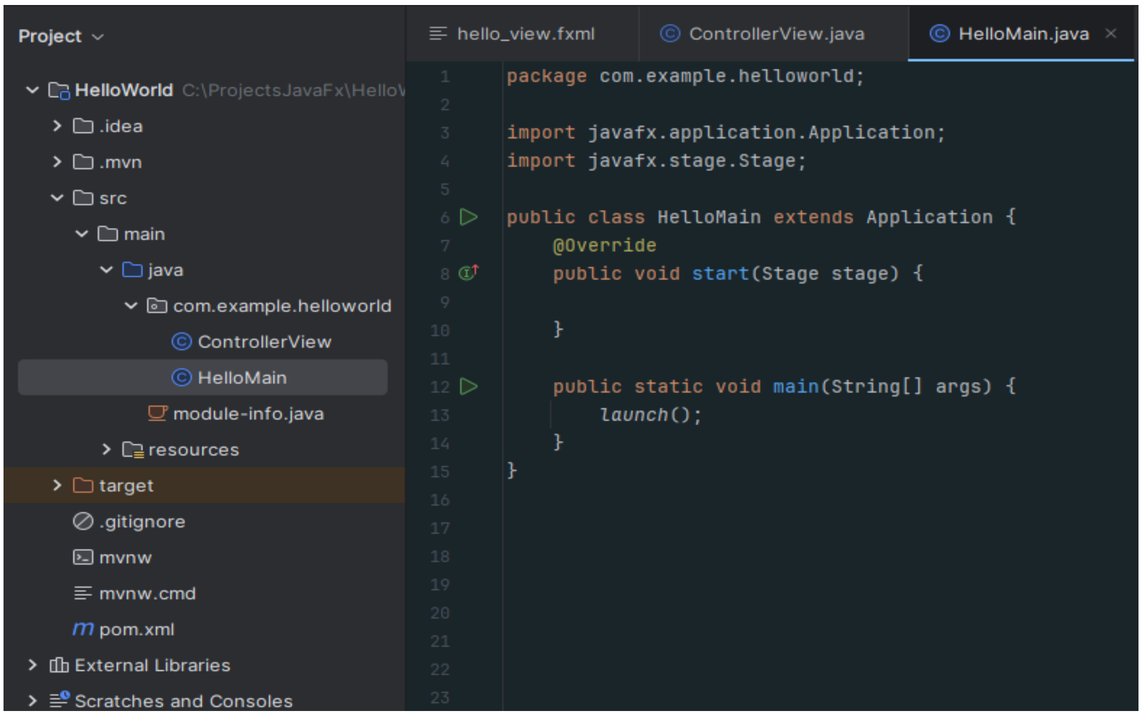
```java
package com.example.helloworld;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;

public class ControllerView {

    @FXML  no usages
    private Button btn;

    @FXML  no usages
    private Label lbl;

    @FXML  no usages
    private TextField txt;

    @FXML  no usages
    void showText(ActionEvent event) {

    }
}
```

In showText we do the logic of what the controller has to do, and it is very simple, it receives what the user sends in the text, and passes it to the label.



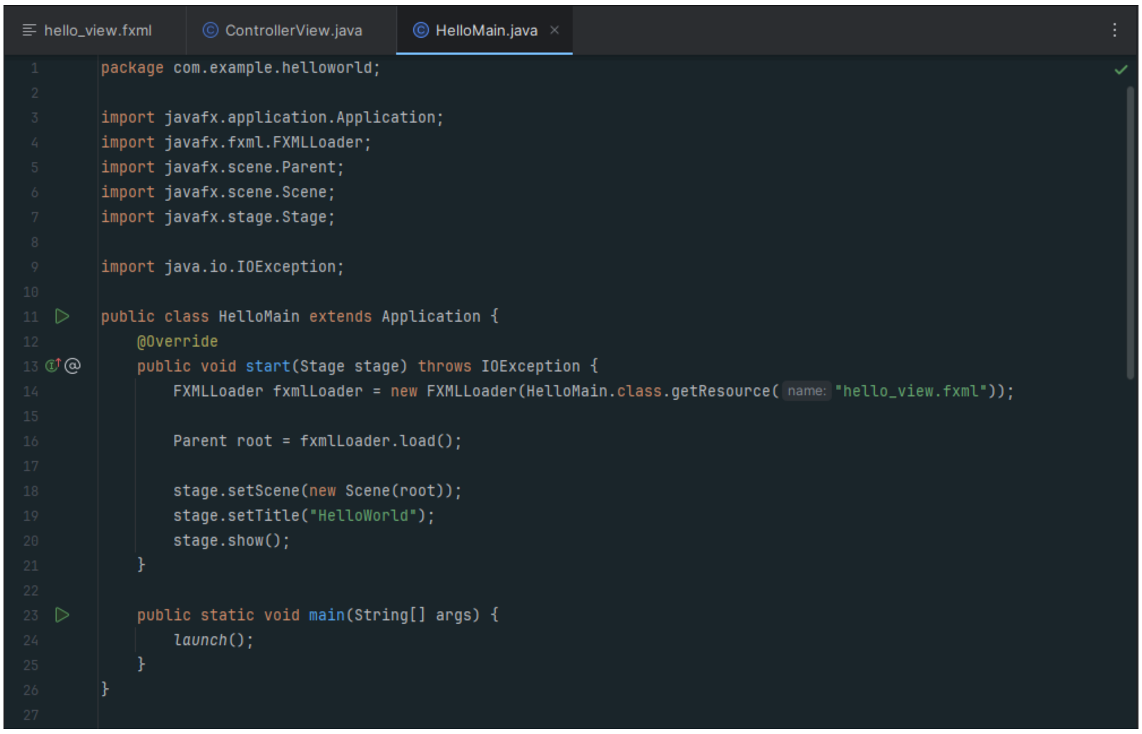
```java
package com.example.helloworld;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;

public class ControllerView {

    @FXML  no usages
    private Button btn;

    @FXML  1 usage
    private Label lbl;

    @FXML  1 usage
    private TextField txt;

    @FXML  no usages
    void showText(ActionEvent event) {
        String text = txt.getText();
        lbl.setText(text);
    }
}
```

Now we create our main class in main.example.helloworld, we extend Application meaning it is a JavaFX application, we implement the start method and add the main method



```java
package com.example.helloworld;

import javafx.application.Application;
import javafx.stage.Stage;

public class HelloMain extends Application {
    @Override
    public void start(Stage stage) {

    }

    public static void main(String[] args) {
        launch();
    }
}
```

With the code

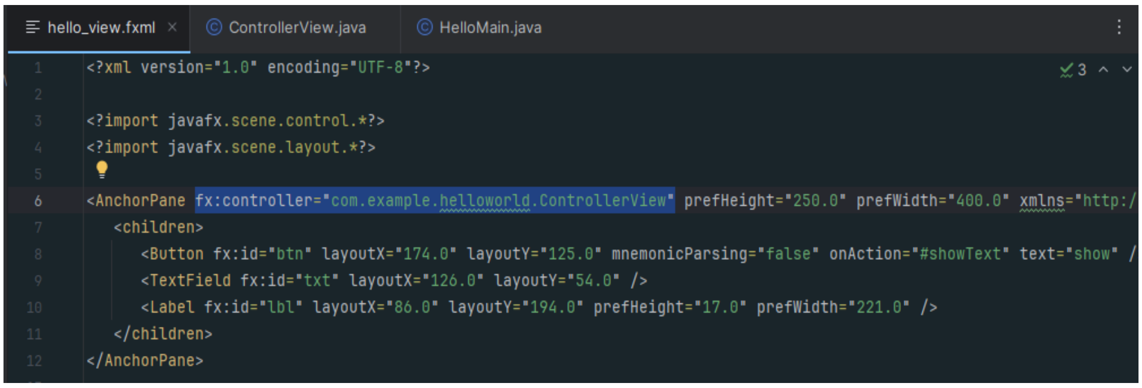


```java
package com.example.helloworld;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

public class HelloMain extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HelloMain.class.getResource( name: "hello_view.fxml"));

        Parent root = fxmlLoader.load();

        stage.setScene(new Scene(root));
        stage.setTitle("HelloWorld");
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

- **Start method:** This is the most important method in a JavaFX application. It is where we configure what will appear in the window.
- **Stage:** This is the main window of the application.

- **FXMLLoader:** We load a hello_view.fxml file, which defines the graphical interface.
- **Parent root:** Here the content of the interface that is defined in the FXML file is saved.
- **Scene:** A scene is the visual content of the window.
- **stage.show():** Shows the window on the screen. Without this, the window would not appear.
- **stage.setTitle():** Add name to the window.
- **Main method:** This method starts the application. It calls launch(), which is a special JavaFX method that starts the whole life cycle of the application.

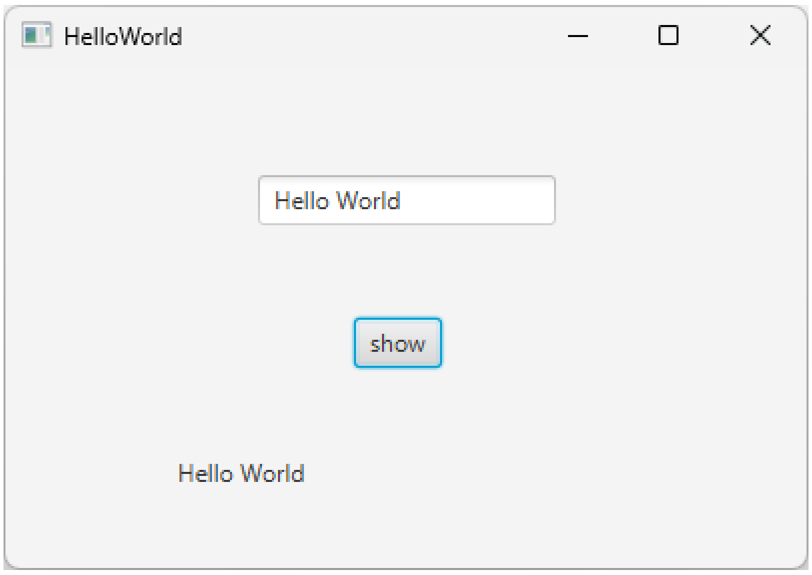
To finish linking the view with the controller we update the following code in the .fxml file



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

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane fx:controller="com.example.helloworld.ControllerView" prefHeight="250.0" prefWidth="400.0" xmlns="http:/
   <children>
      <Button fx:id="btn" layoutX="174.0" layoutY="125.0" mnemonicParsing="false" onAction="#showText" text="show" /
      <TextField fx:id="txt" layoutX="126.0" layoutY="54.0" />
      <Label fx:id="lbl" layoutX="86.0" layoutY="194.0" prefHeight="17.0" prefWidth="221.0" />
   </children>
</AnchorPane>
```

We place the name of the controller that we created

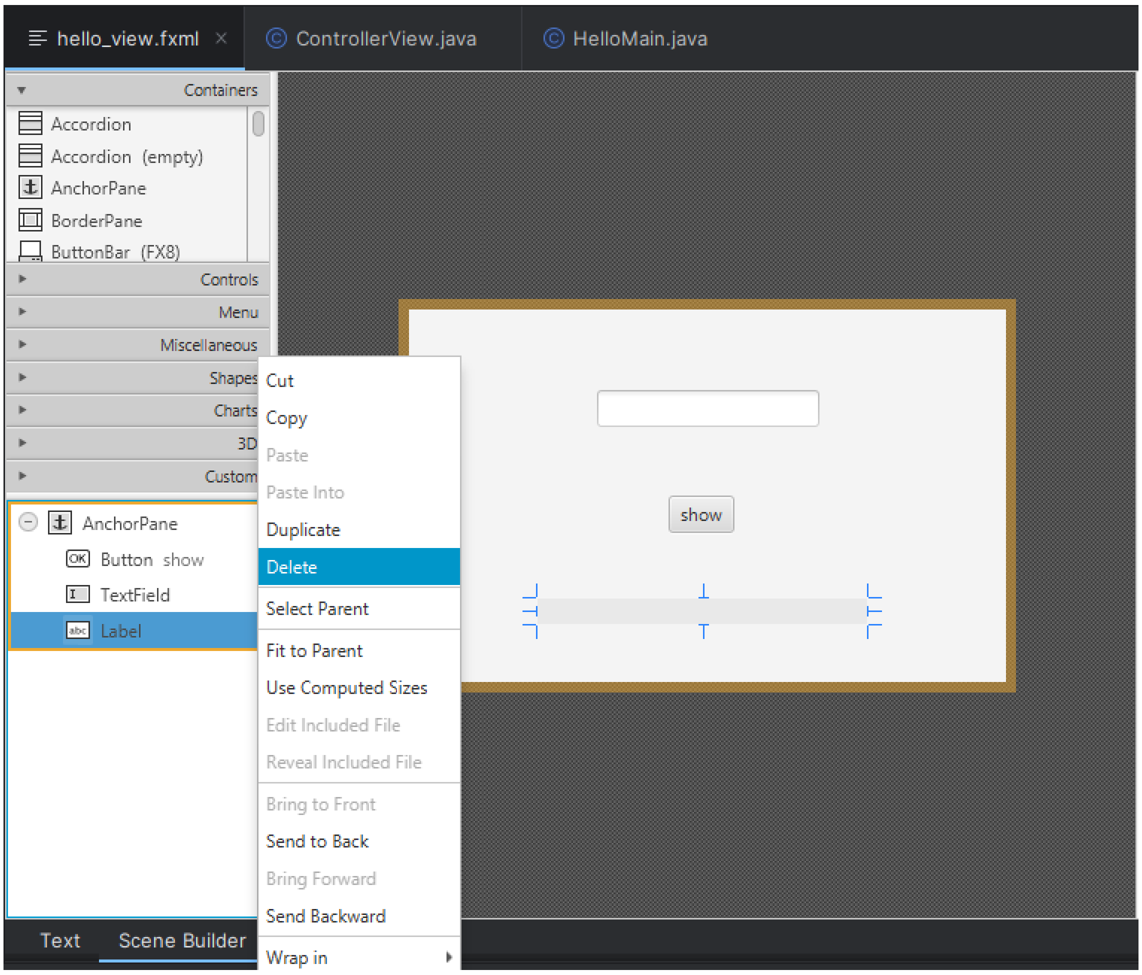
And when we run it, it would look like this



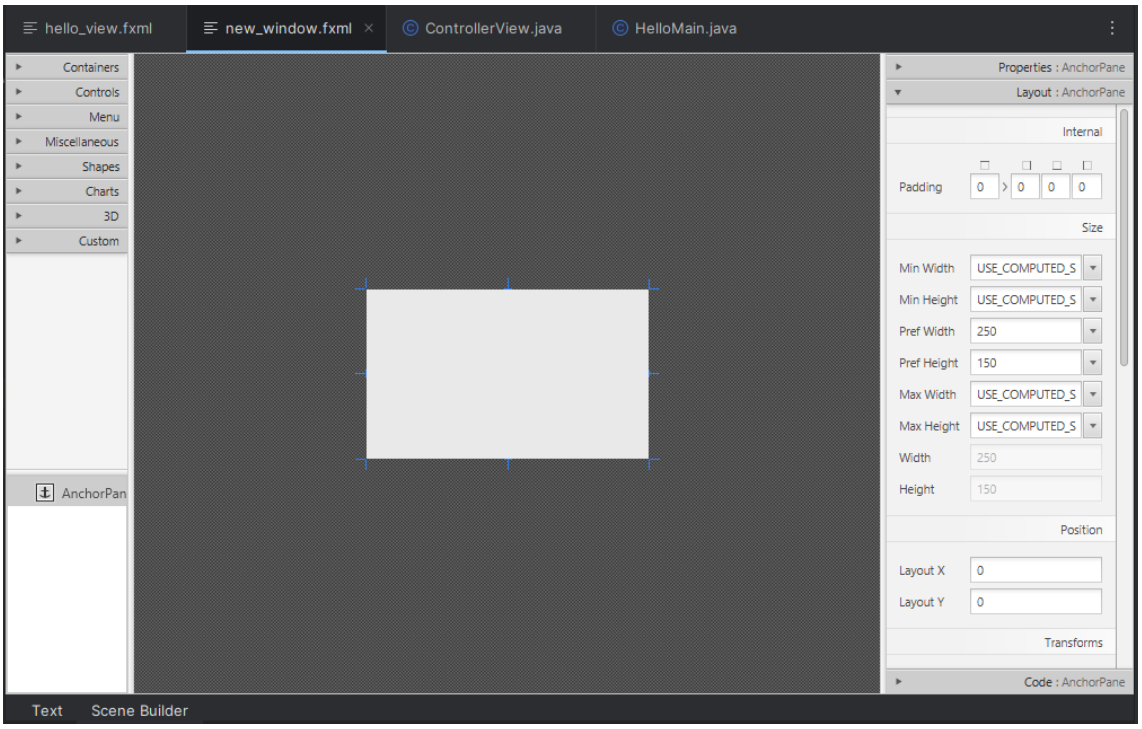
Adding more functionalities

New window containing the Hello World

Remove the label tag

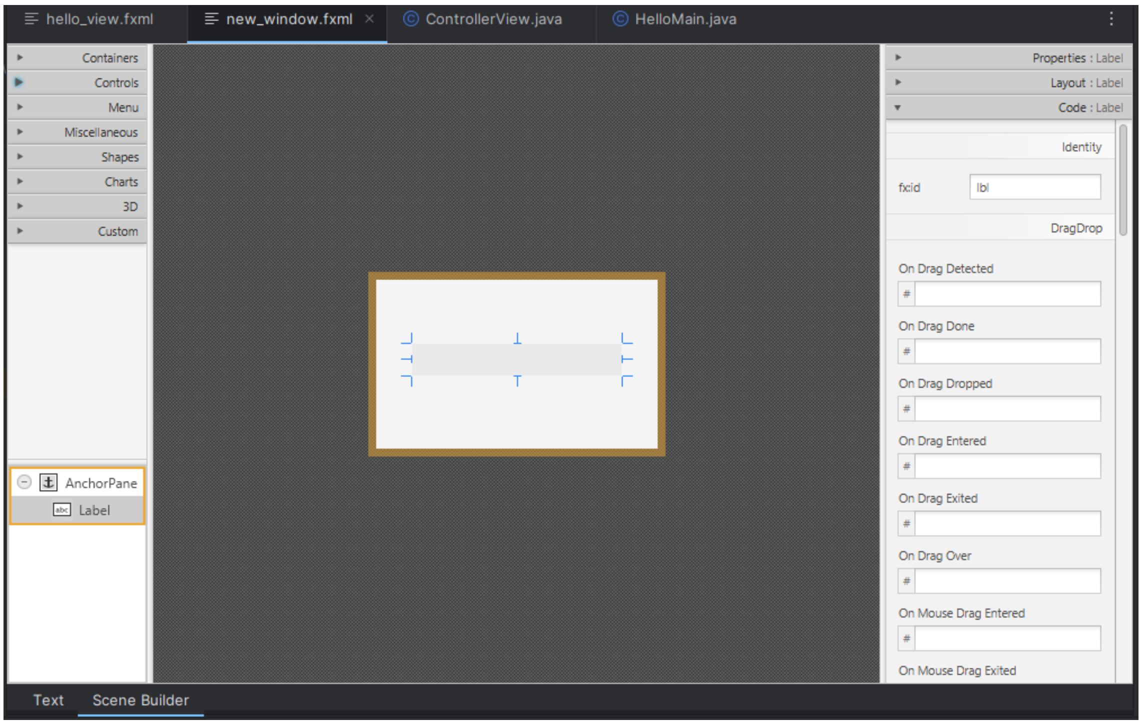


Create new .fxml file -> name it new_widow, open it with scene builder and adjust the size



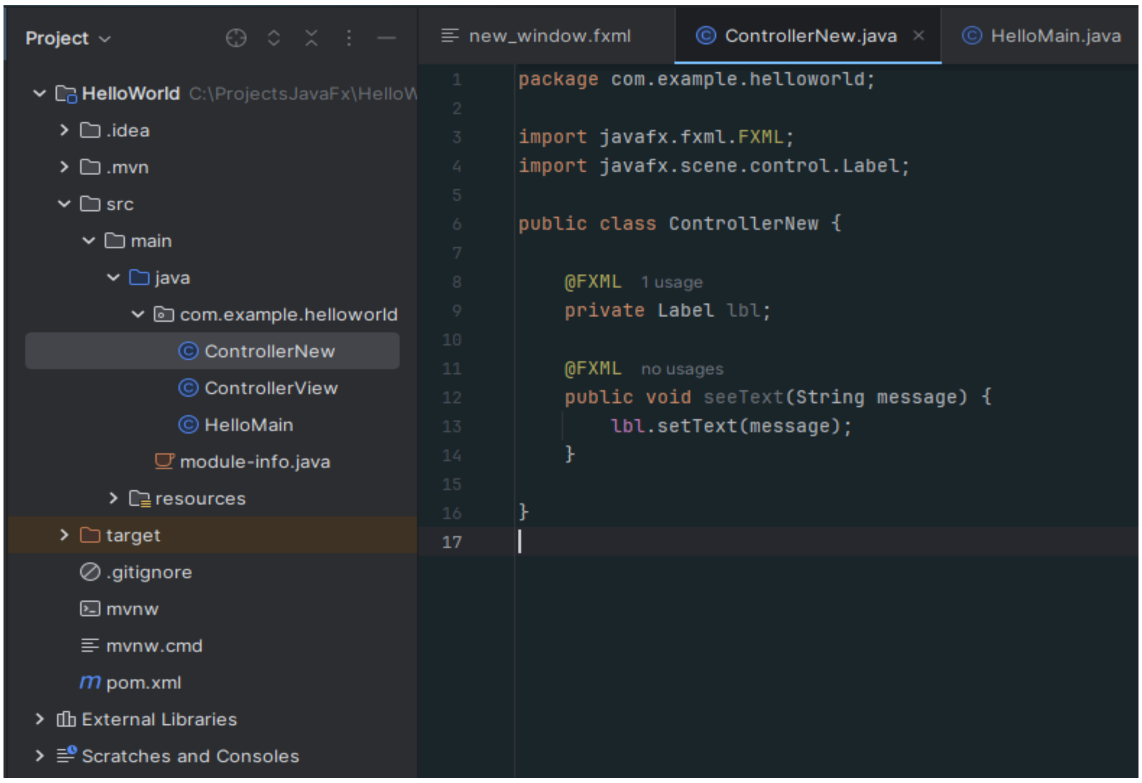
Padding 0 0 0 0
Min Width USE_COMPUTED_S
Min Height USE_COMPUTED_S
Pref Width 250
Pref Height 150
Max Width USE_COMPUTED_S
Max Height USE_COMPUTED_S
Width 250
Height 150
Layout X 0
Layout Y 0

Add a new label and remove the default text and add the id



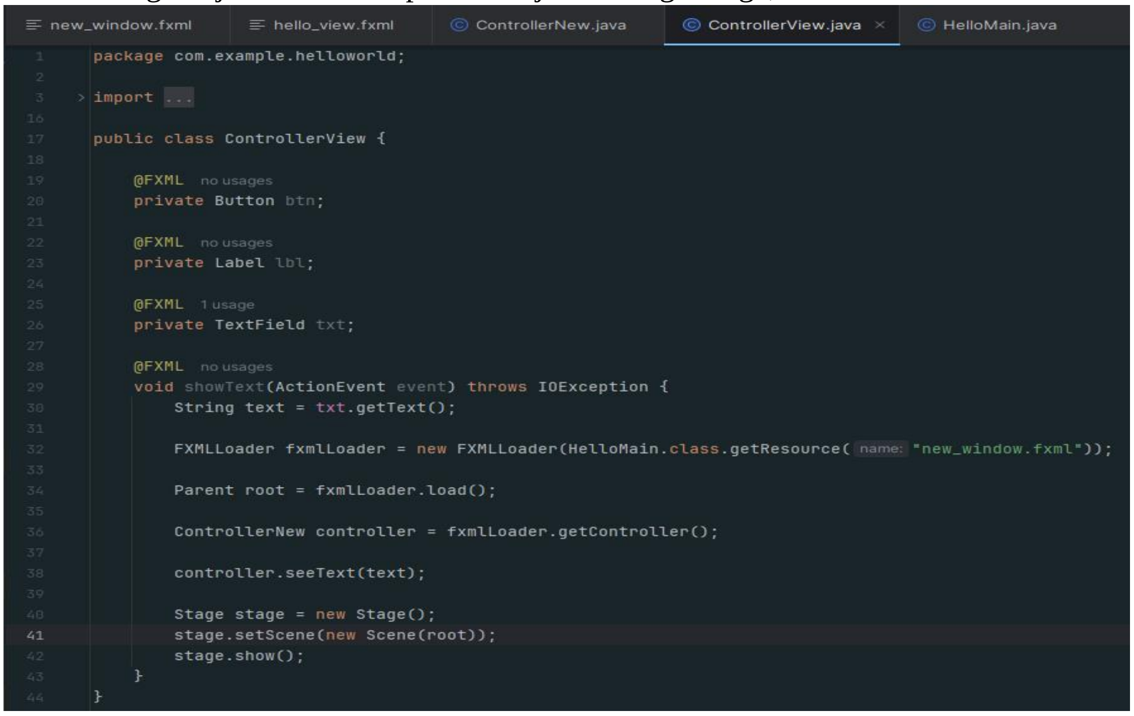
fx:id lbl

Create a new controller class called ControllerNew, add components with its annotation, add a new function called seeText and pass the message to it. Remember to update the name in the new_window.fxml file.



```java
package com.example.helloworld;

import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class ControllerNew {

    @FXML
    private Label lbl;

    @FXML
    public void seeText(String message) {
        lbl.setText(message);
    }

}
```

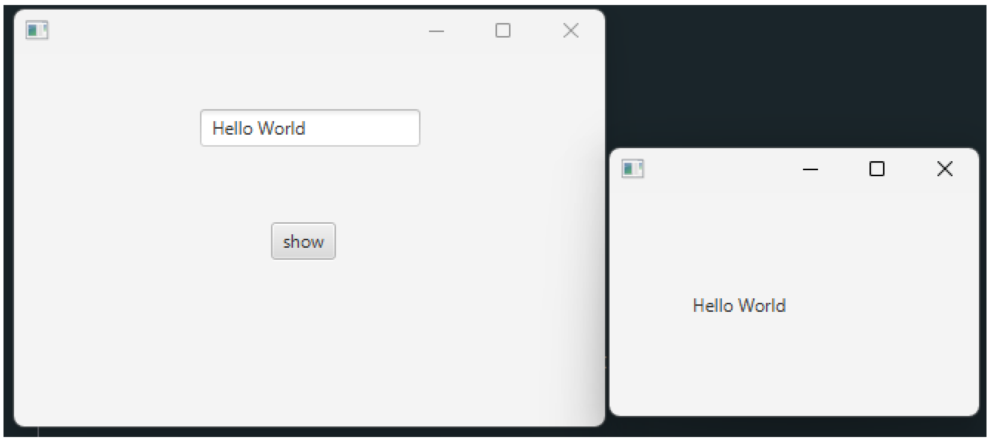
Modify the first controller called ControllerView, delete the line that we pass the text and add the code we have in the start method, update the path to the new arxhivo.fxml, create a stage object that we import from javafx.stage.Stage;



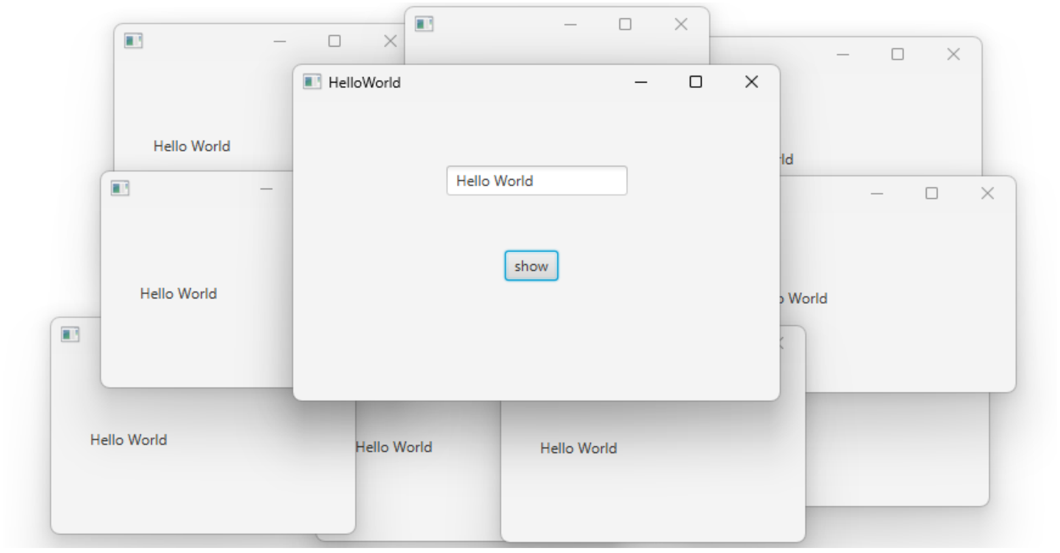
```
package com.example.helloworld;

import ...

public class ControllerView {

    @FXML  no usages
    private Button btn;

    @FXML  no usages
    private Label lbl;

    @FXML  1 usage
    private TextField txt;

    @FXML  no usages
    void showText(ActionEvent event) throws IOException {
        String text = txt.getText();

        FXMLLoader fxmlLoader = new FXMLLoader(HelloMain.class.getResource( name: "new_window.fxml"));

        Parent root = fxmlLoader.load();

        ControllerNew controller = fxmlLoader.getController();

        controller.seeText(text);

        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.show();
    }
}
```

- The button defined in the FXML file, associated with an ID of 'btn'. When the user clicks on this button, the 'showText' method is triggered.
- Label that can display some text in the user interface. It is linked to an ID of 'lbl' in the FXML file.
- TextField where the user can enter text. The value of this field is obtained in the 'showText' method.
- This method is activated when the button is clicked. Its function is to open a new window. It first gets the text from the text field 'txt', then loads a new FXML file named “new_window.fxml”, and finally, displays that new window.
- The action event that occurs when the button is clicked is passed as a parameter.
- If an error occurs when loading the FXML file, an IOException is thrown.
- The text entered by the user in the text field 'txt' is retrieved and stored in a variable.
- The new view is loaded from the FXML file “new_window.fxml”.
- The FXML file is loaded, and the corresponding interface structure is created.
- The controller of the new window is obtained so that the text can be passed to it.
- The text from the text field 'txt' is passed to the controller of the new window.
- A new window (Stage) is created to display the new scene.
- The new scene loaded from the FXML file is set in the new window.
- Finally, the new window is displayed on the screen.

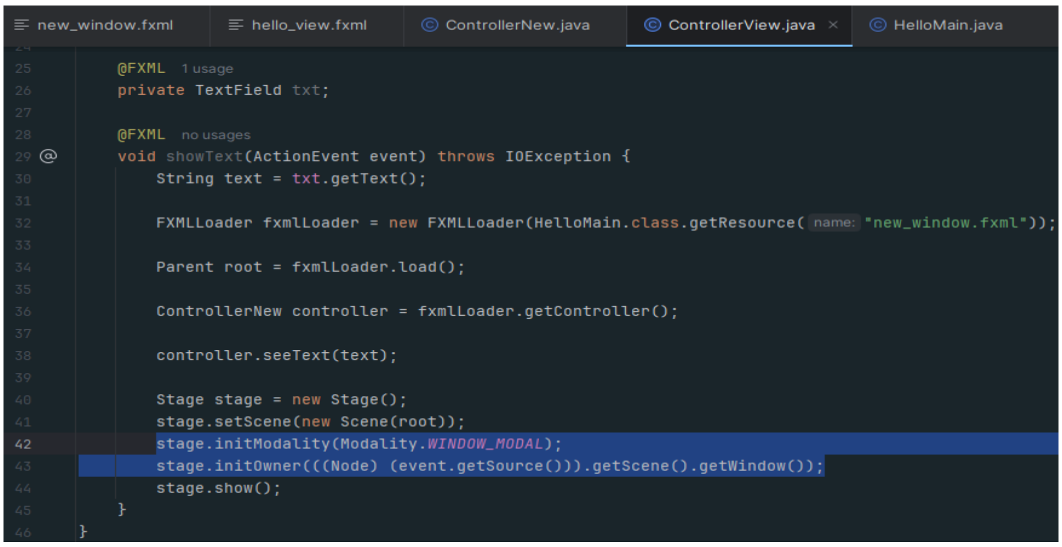
When running it, it already shows the windows passing the text



Fix that if we press the show button too many times it creates multiple Windows



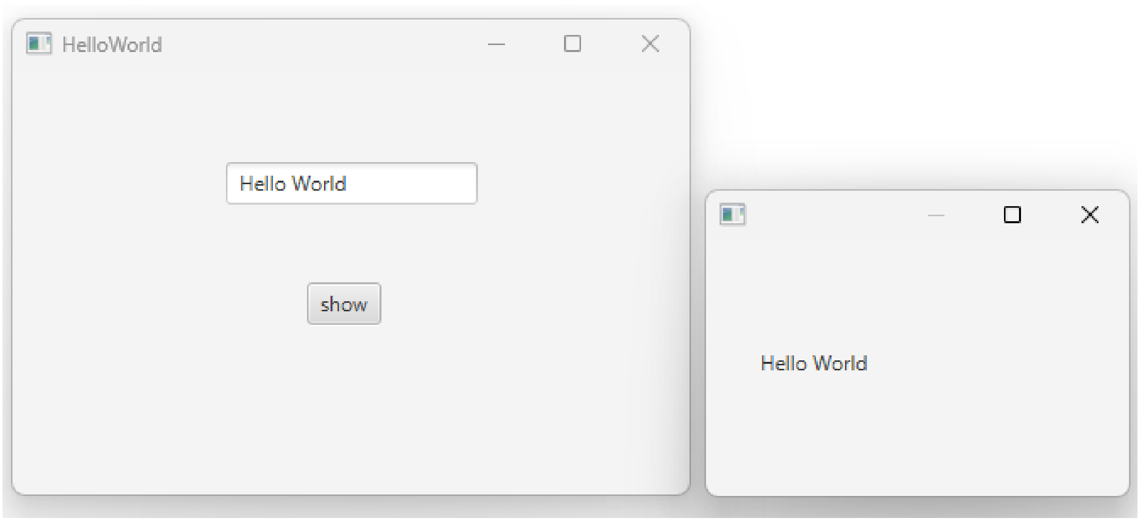
Enable modal window, so that when you create a new window, the main window is disabled.



```
new_window.fxml   hello_view.fxml   ControllerNew.java   ControllerView.java ×   HelloMain.java

    @FXML  1 usage
    private TextField txt;

    @FXML  no usages
    void showText(ActionEvent event) throws IOException {
        String text = txt.getText();

        FXMLLoader fxmlLoader = new FXMLLoader(HelloMain.class.getResource( name: "new_window.fxml"));

        Parent root = fxmlLoader.load();

        ControllerNew controller = fxmlLoader.getController();

        controller.seeText(text);

        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.initModality(Modality.WINDOW_MODAL);
        stage.initOwner(((Node) (event.getSource())).getScene().getWindow());
        stage.show();
    }
}
```

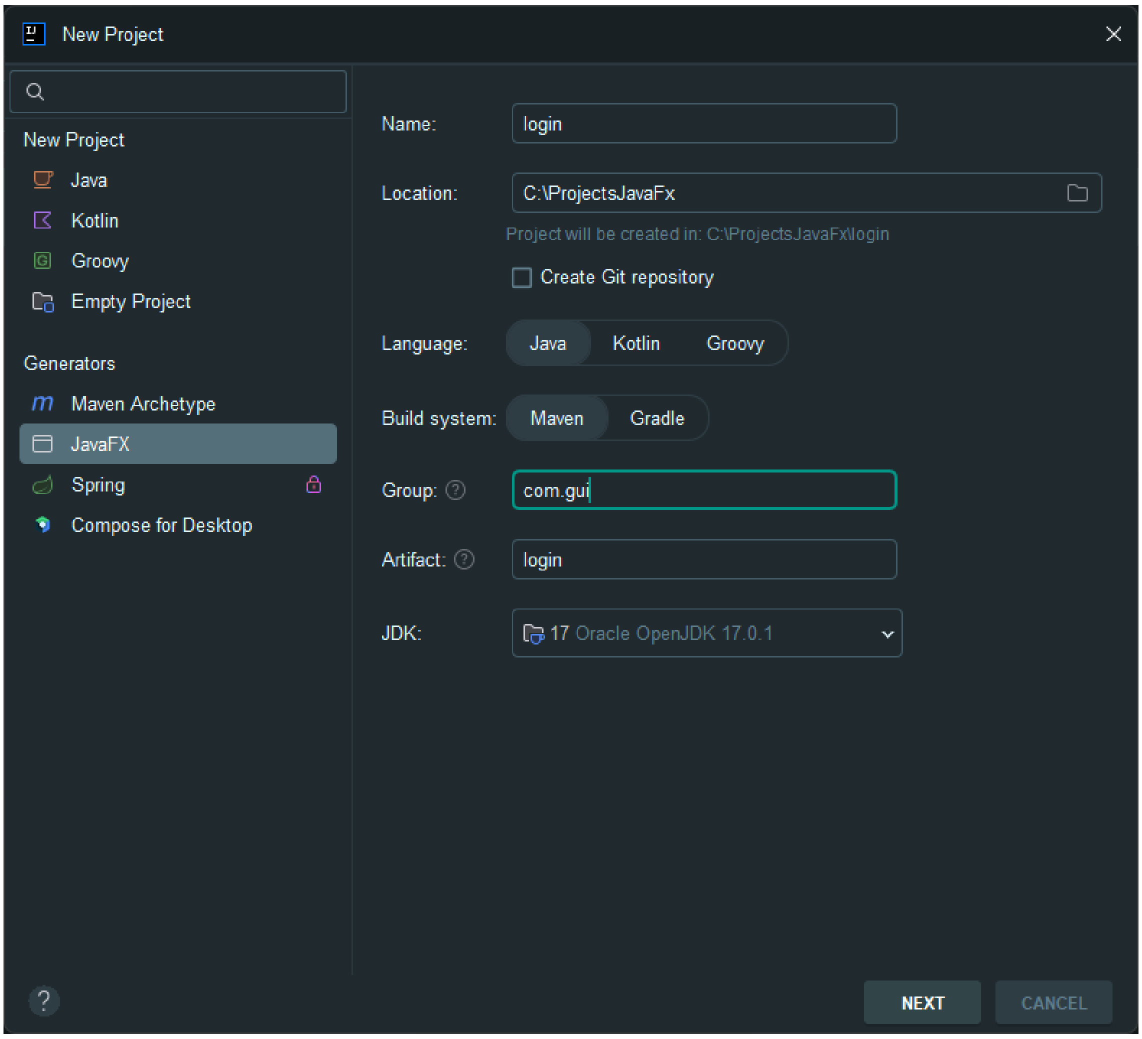
What was added serves to set the window as modal and sets the main window as its owner, blocking interaction with the main window until it is closed.

It no longer lets you interact with the main window



Exercise 2: Login

Create Project


New Project

Name: login

Location: C:\ProjectsJavaFx

Project will be created in: C:\ProjectsJavaFx\login

Create Git repository

Language: Java | Kotlin | Groovy

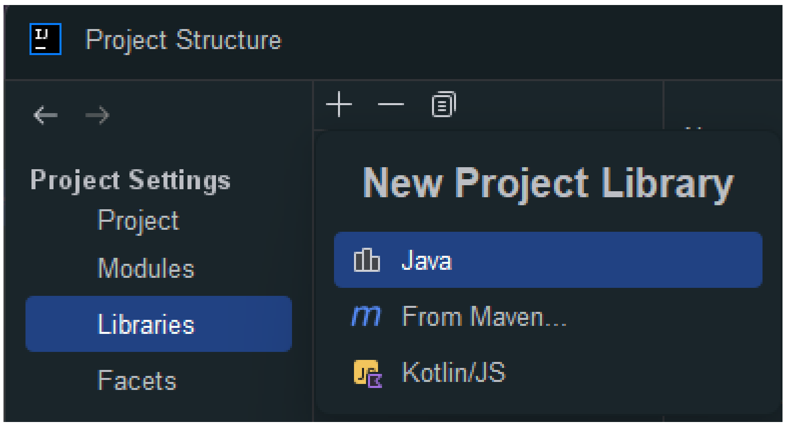
Build system: Maven | Gradle

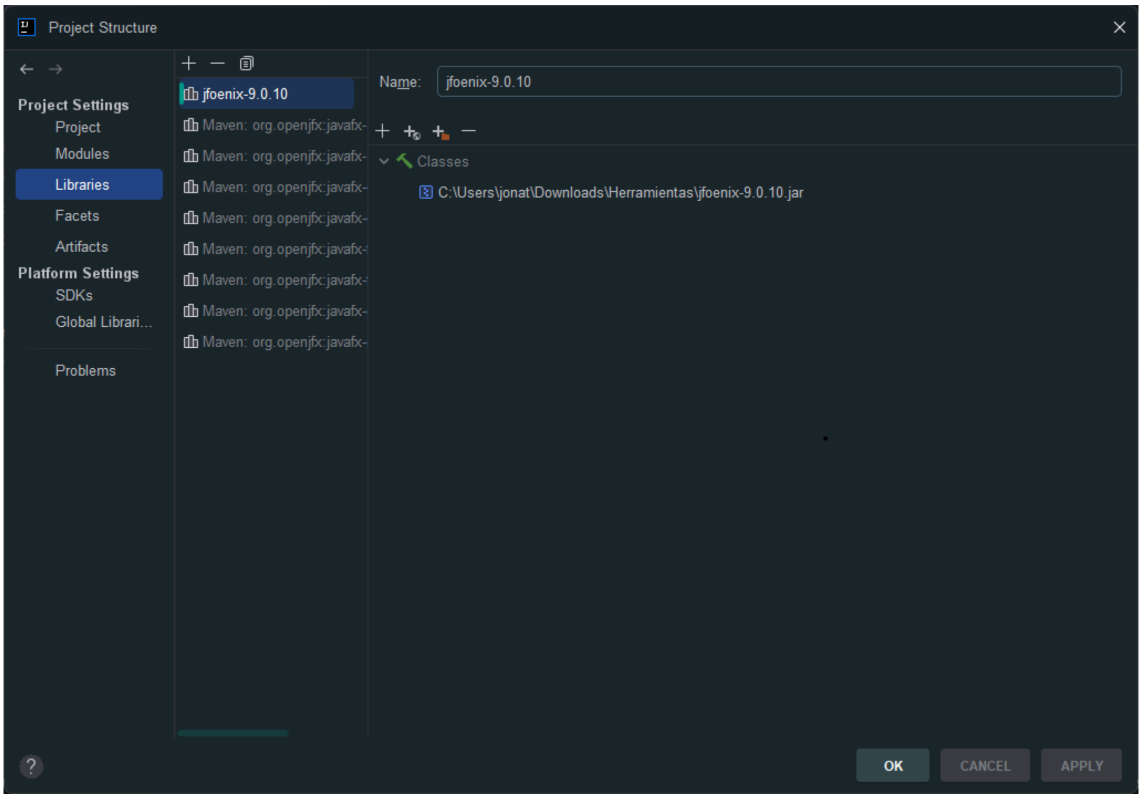
Group: com.gui

Artifact: login

JDK: 17 Oracle OpenJDK 17.0.1

Once created, go to File > Project Structure > Libraries -> + sign and add the J Foenix .jar.

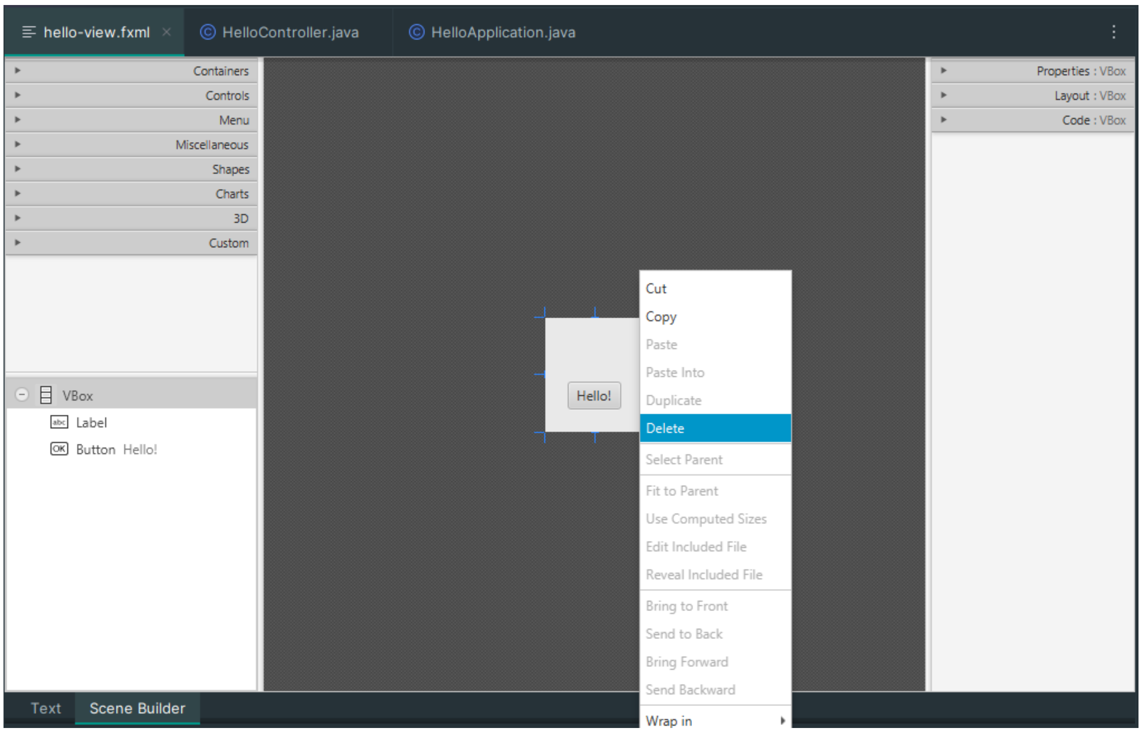



Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Librari...
Problems
jfoenix-9.0.10
Maven: org.openjfx:javafx-
Maven: org.openjfx:javafx-
Maven: org.openjfx:javafx-
Maven: org.openjfx:javafx-
Maven: org.openjfx:javafx-
Maven: org.openjfx:javafx-
Maven: org.openjfx:javafx-
Maven: org.openjfx:javafx-
Classes
OK
CANCEL
APPLY

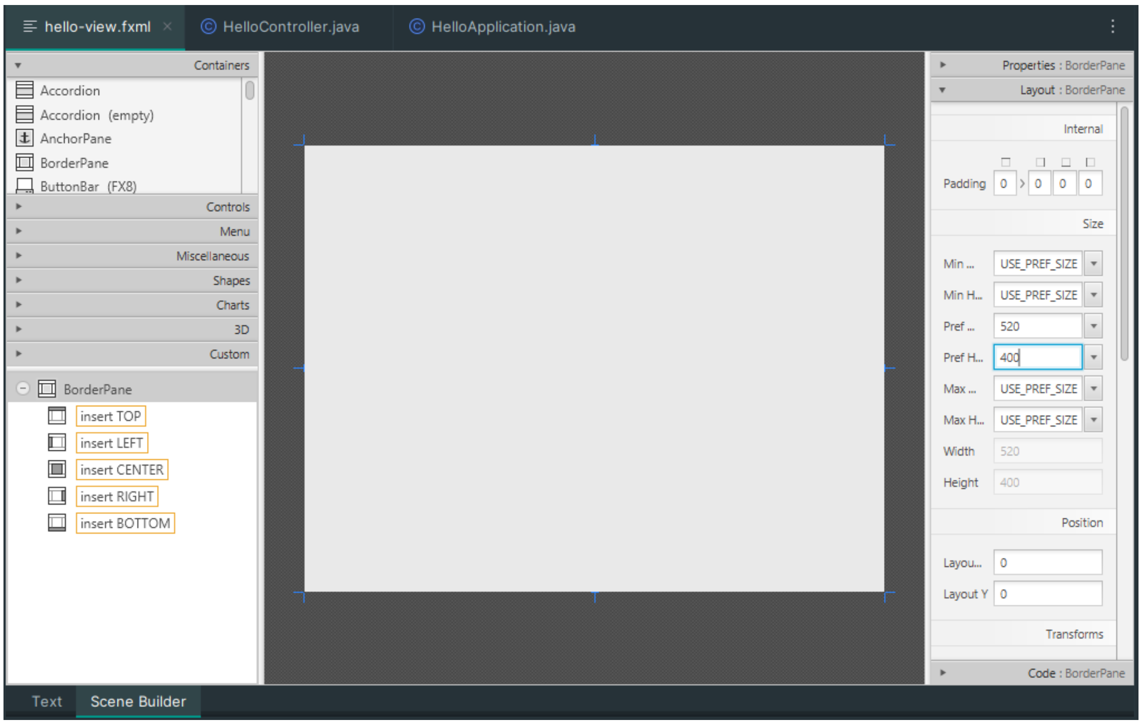
Name: jfoenix-9.0.10

C:\Users\jonat\Downloads\Herramientas\jfoenix-9.0.10.jar

Open the hello-view.fxml file with scene builder and delete the AnchorPane it contains

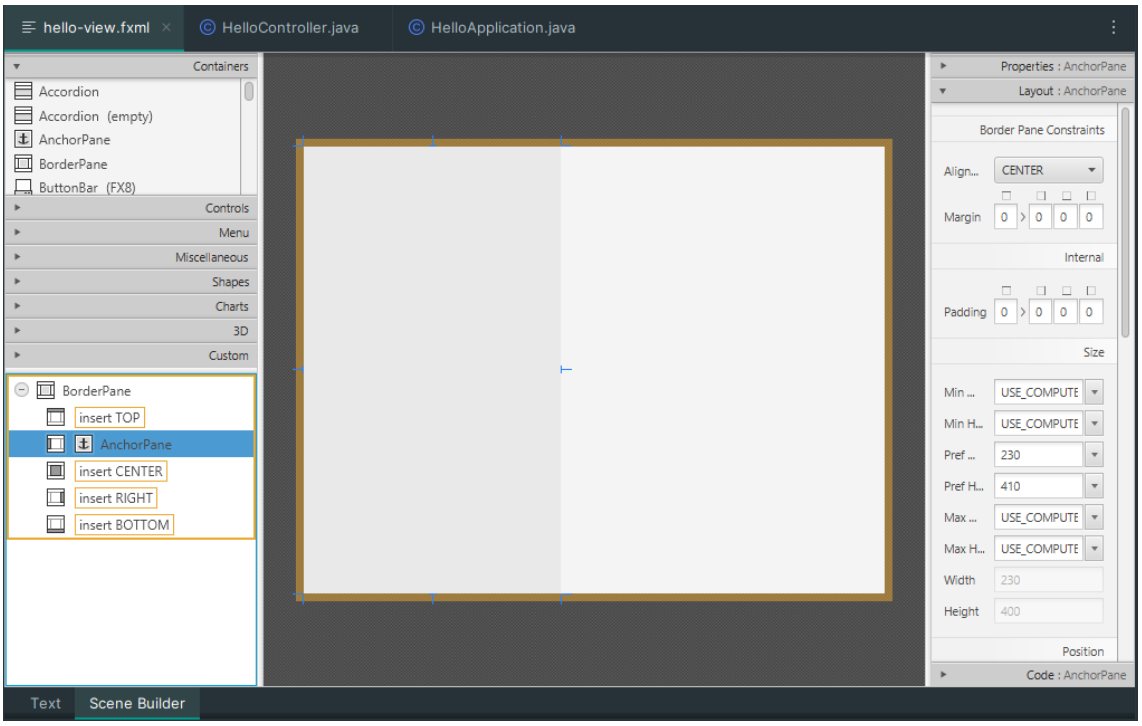


Add a BorderPane and configure it 520x400



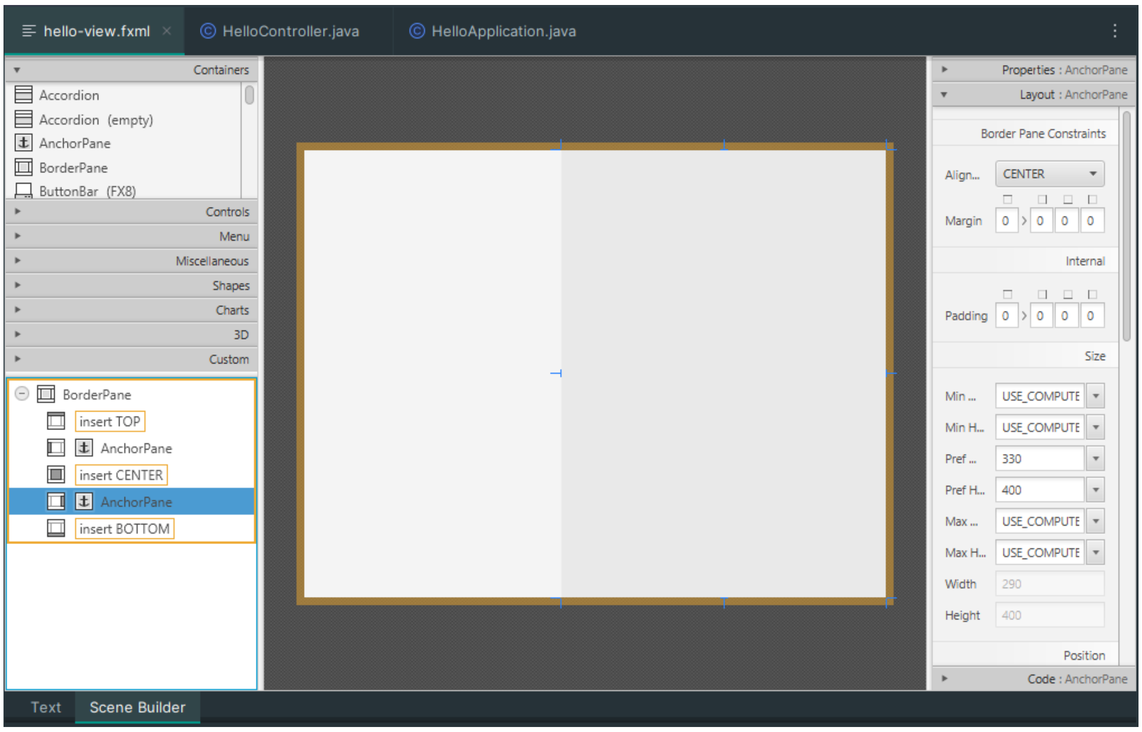
Padding 0 0 0 0
Min ... USE_PREF_SIZE
Min H... USE_PREF_SIZE
Pref ... 520
Pref H... 400
Max ... USE_PREF_SIZE
Max H... USE_PREF_SIZE
Width 520
Height 400
Layou... 0
Layout Y 0

Add AnchorPane by dragging on the left side to where it says insert LEFT and set it to 230x410



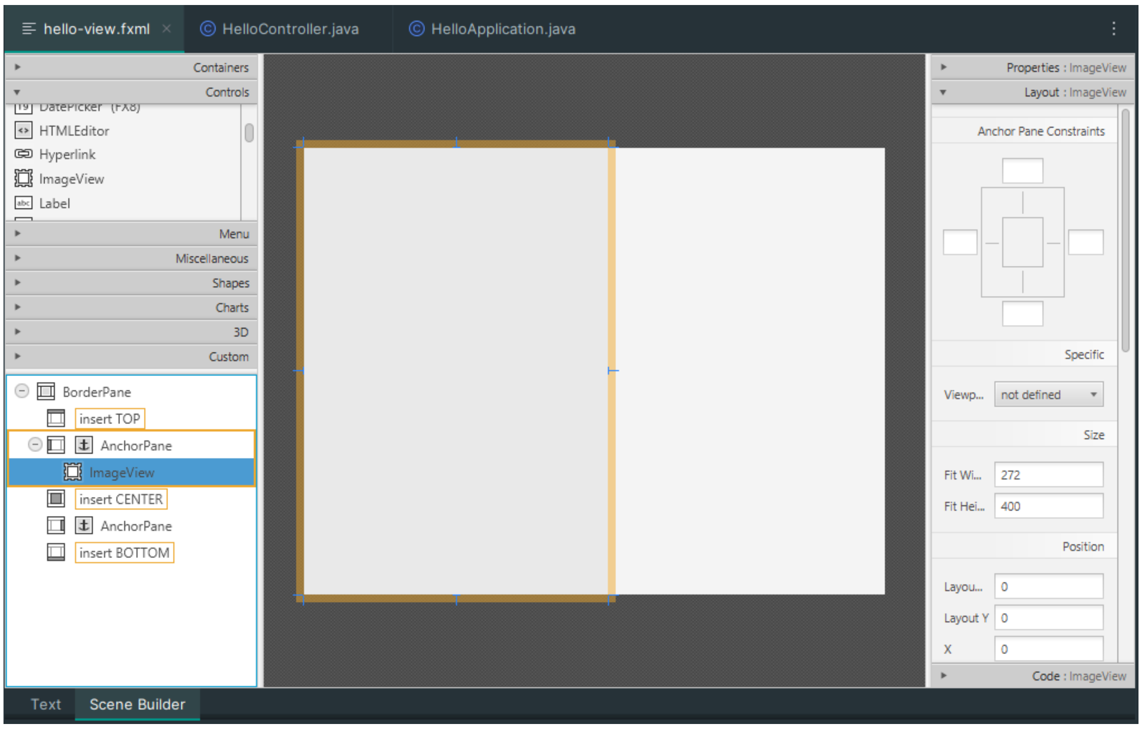
Align... CENTER
Margin 0 0 0 0
Padding 0 0 0 0
Min ... USE_COMPUTE
Min H... USE_COMPUTE
Pref ... 230
Pref H... 410
Max ... USE_COMPUTE
Max H... USE_COMPUTE
Width 230
Height 400

Add another AnchorPane by dragging to where it says insert RIGTH, set it to 330x400



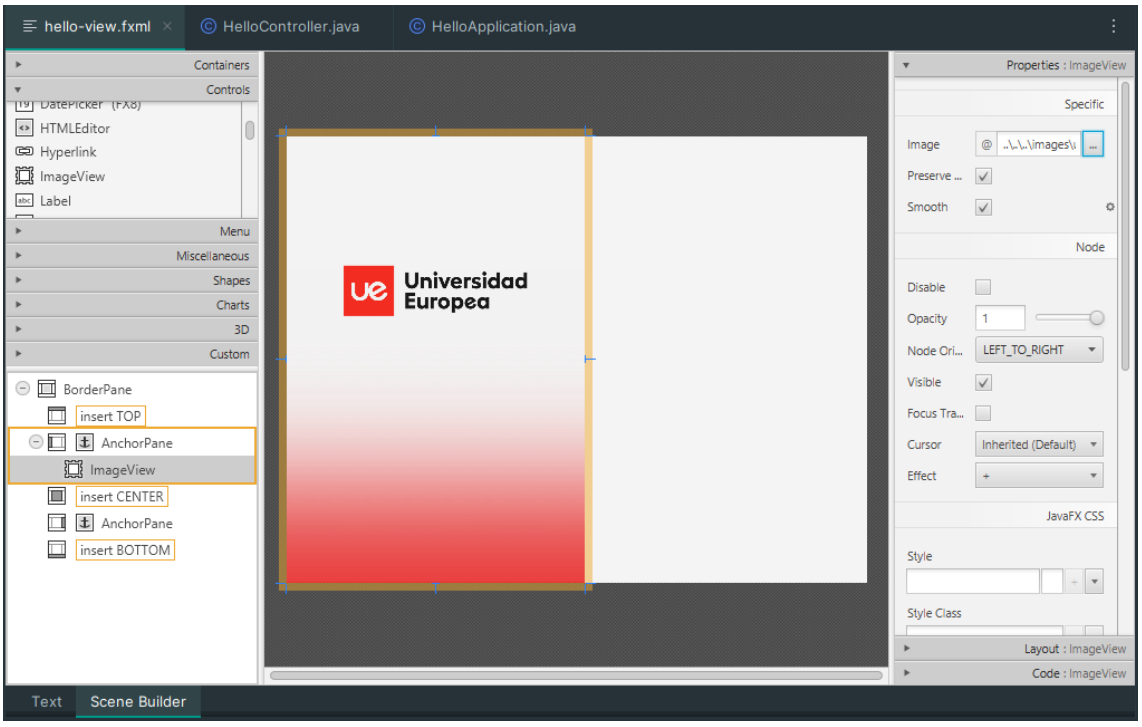
Align... CENTER
Margin 0 0 0 0
Padding 0 0 0 0
Min ... USE_COMPUTE
Min H... USE_COMPUTE
Pref ... 330
Pref H... 400
Max ... USE_COMPUTE
Max H... USE_COMPUTE
Width 290
Height 400

30ill30n ImageView and set its size to 272x400

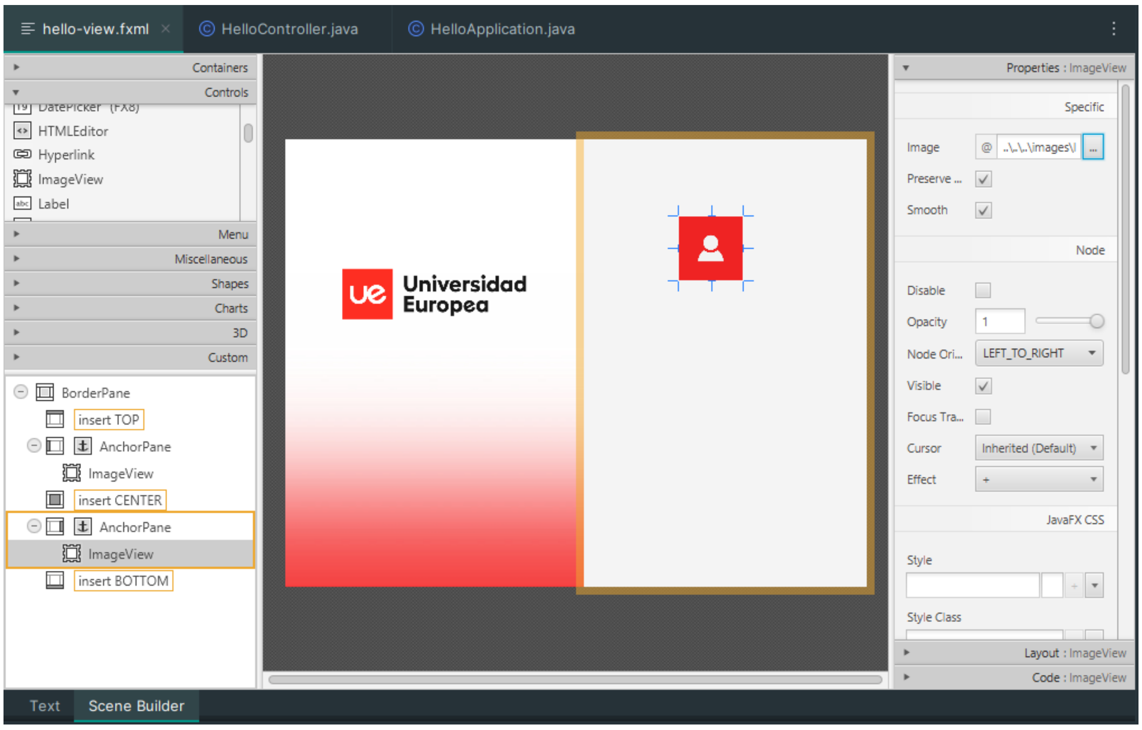


Viewp... not defined
Fit Wi... 272
Fit Hei... 400
Layou... 0
Layout Y 0
X 0

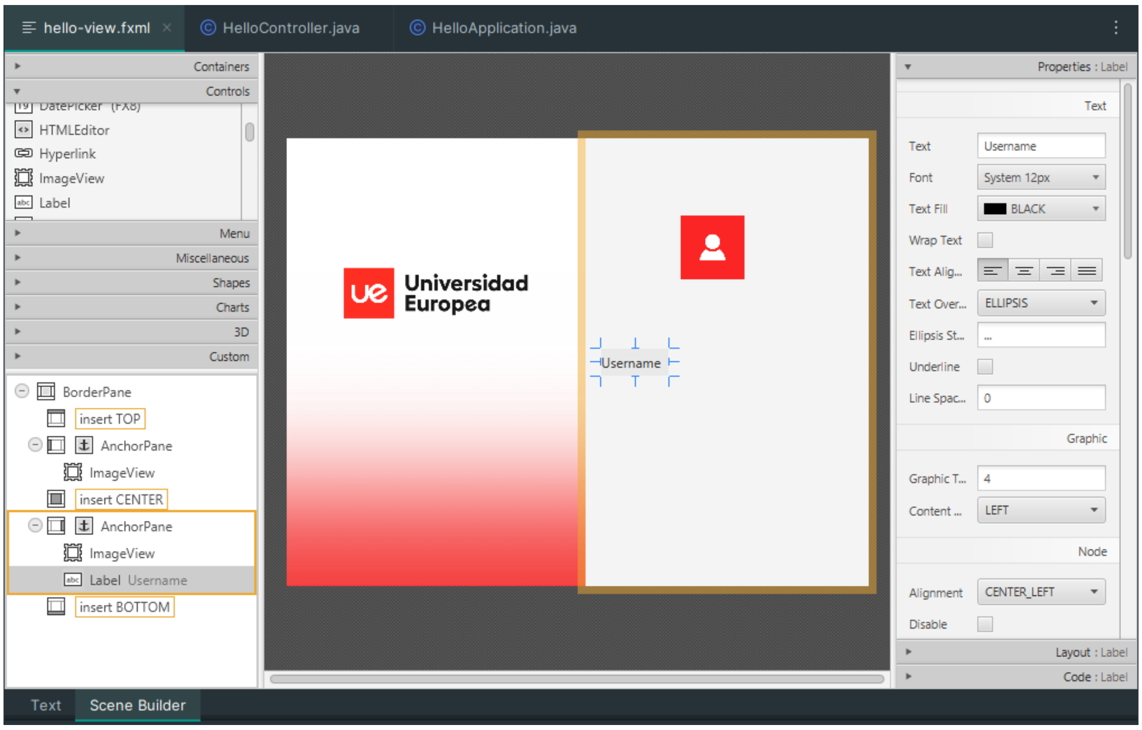
In properties > layout load image



Add another ImageView with the size of 84x57 and load image from properties > layout > image

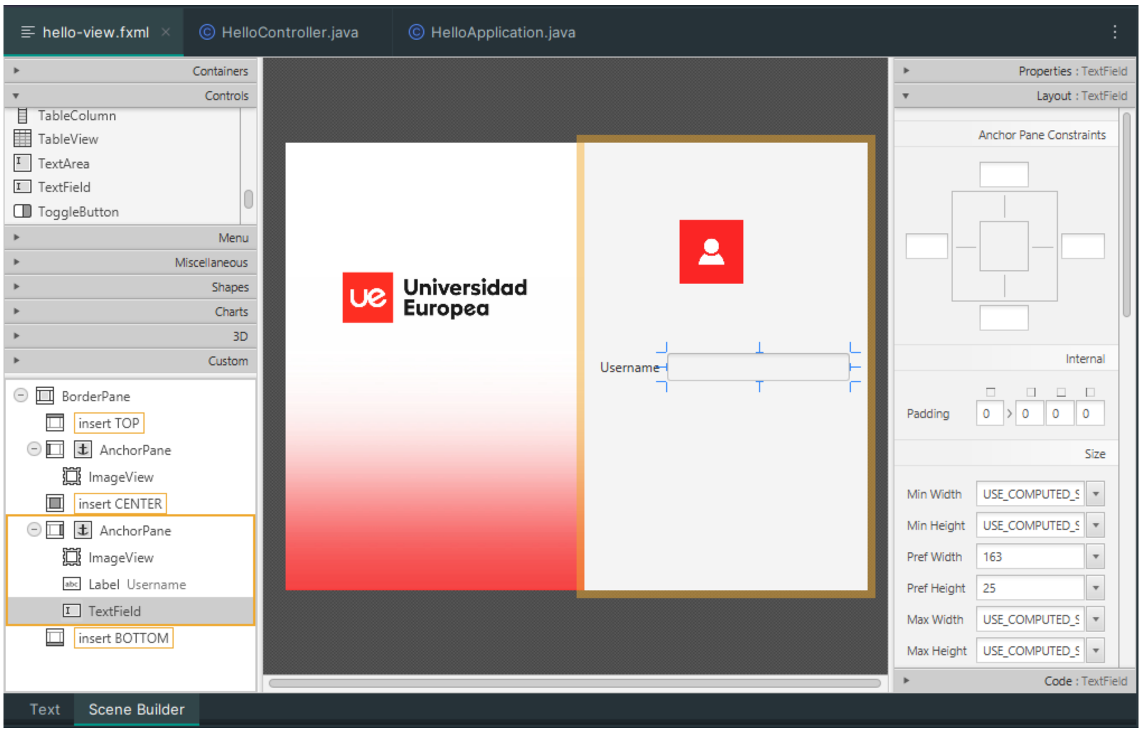


Add a new label and in properties > text = Username



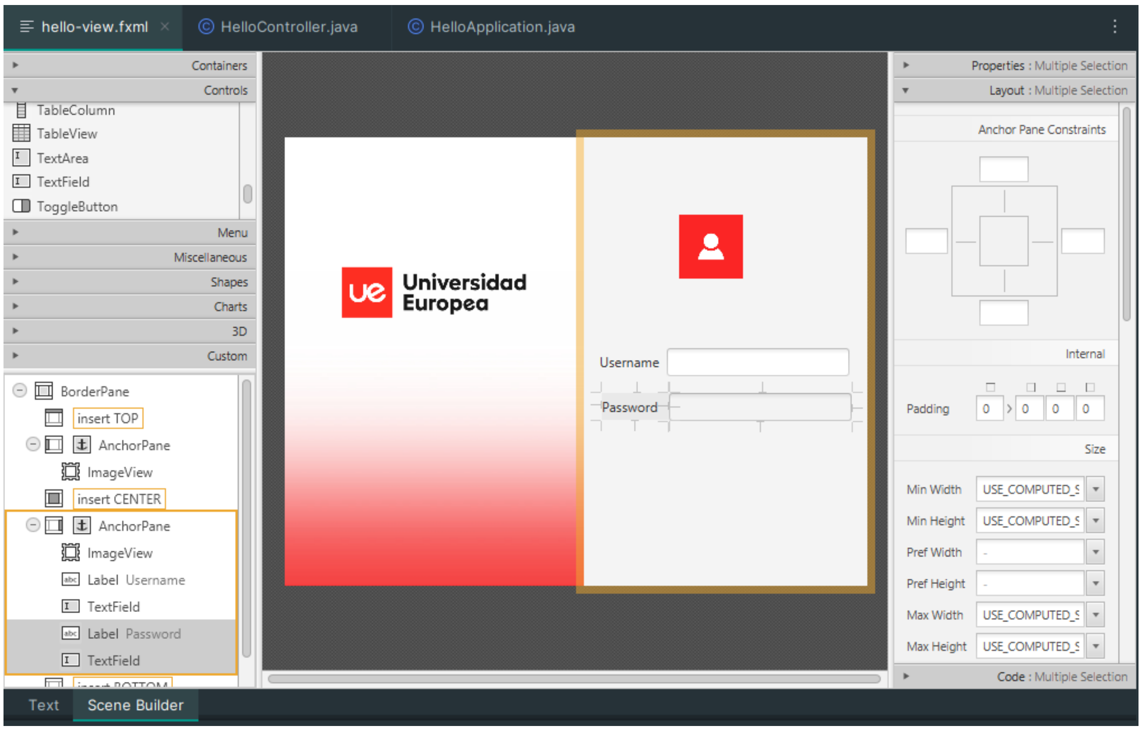
Text: Username
Font: System 12px
Text Fill: BLACK
Text Over...: ELLIPSIS
Ellipsis St...: ...
Line Spac...: 0
Graphic T...: 4
Content ...: LEFT
Alignment: CENTER_LEFT

Add a textField and configure it 163x25



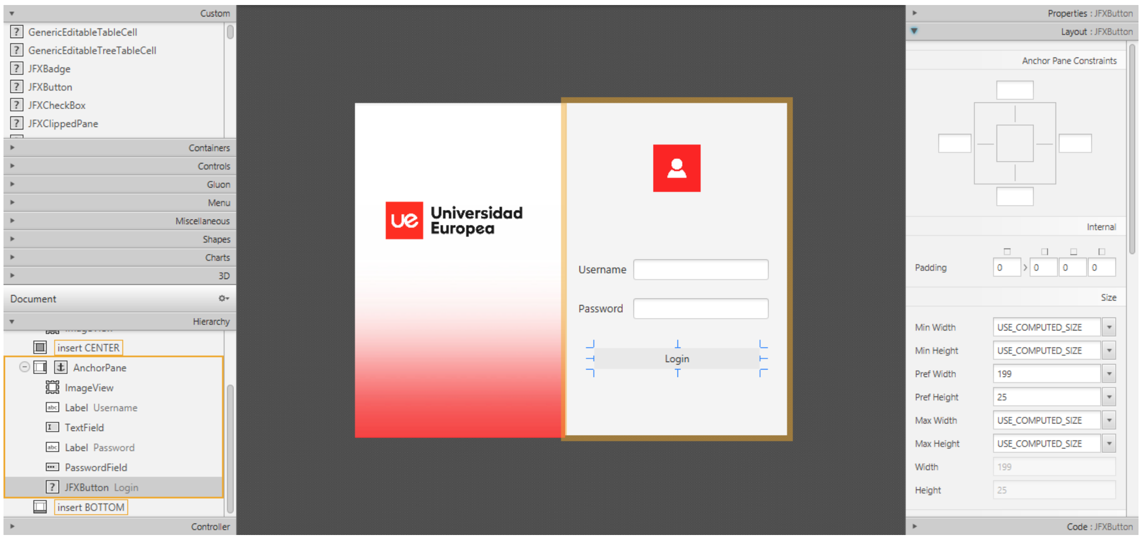
Padding: 0 0 0 0
Min Width: USE_COMPUTED_S
Min Height: USE_COMPUTED_S
Pref Width: 163
Pref Height: 25
Max Width: USE_COMPUTED_S
Max Height: USE_COMPUTED_S

Add another label and a Password field with the same settings as above



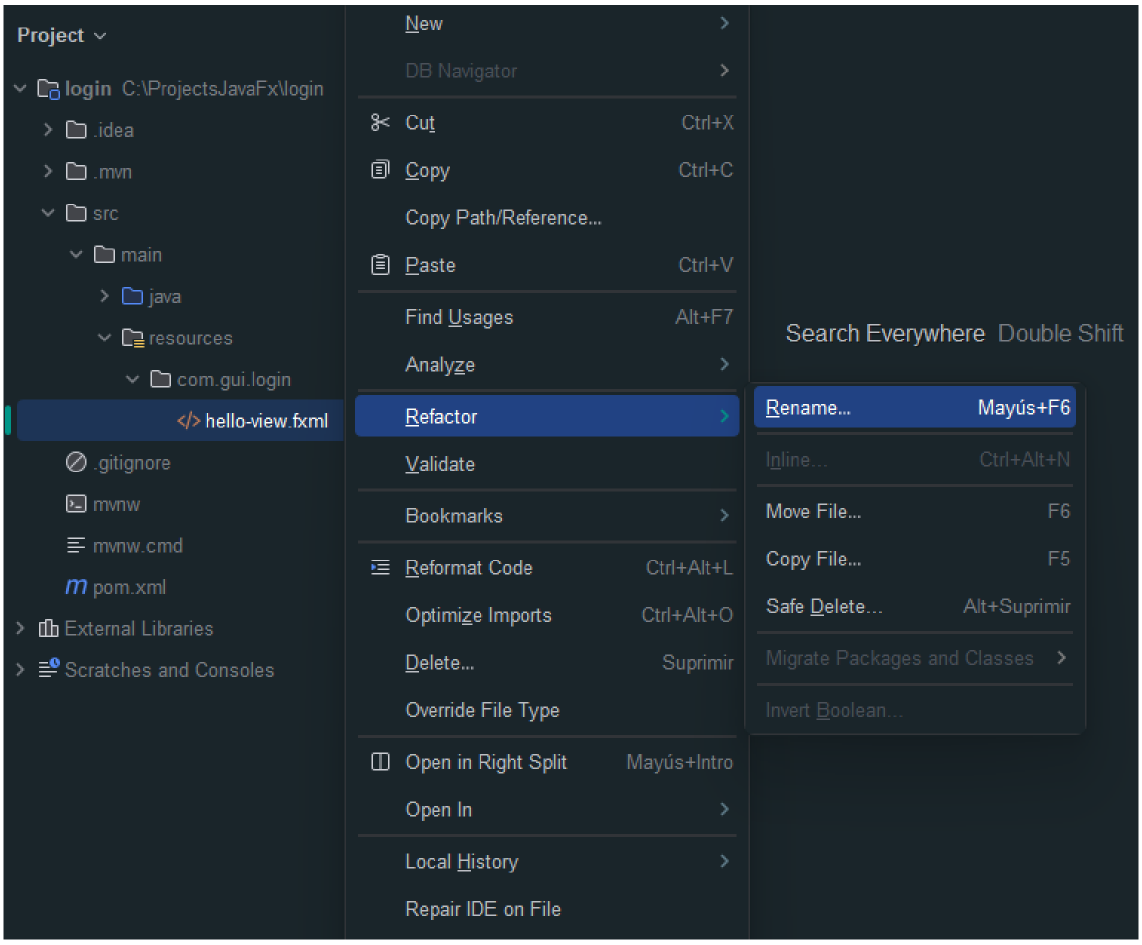
Padding 0 0 0 0
Min Width USE_COMPUTED_S
Min Height USE_COMPUTED_S
Pref Width -
Pref Height -
Max Width USE_COMPUTED_S
Max Height USE_COMPUTED_S

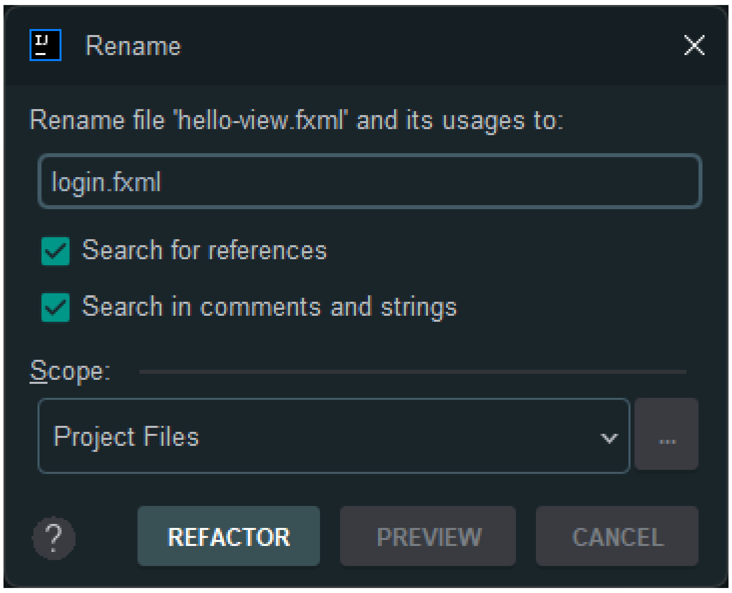
Add a JFXBUTTON, rename it in properties > text and set in Loyout 199x25



Padding 0 0 0 0
Min Width USE_COMPUTED_SIZE
Min Height USE_COMPUTED_SIZE
Pref Width 199
Pref Height 25
Max Width USE_COMPUTED_SIZE
Max Height USE_COMPUTED_SIZE
Width 199
Height 25

Rename the fxml file to login




Rename

Rename file 'hello-view.fxml' and its usages to:

login.fxml

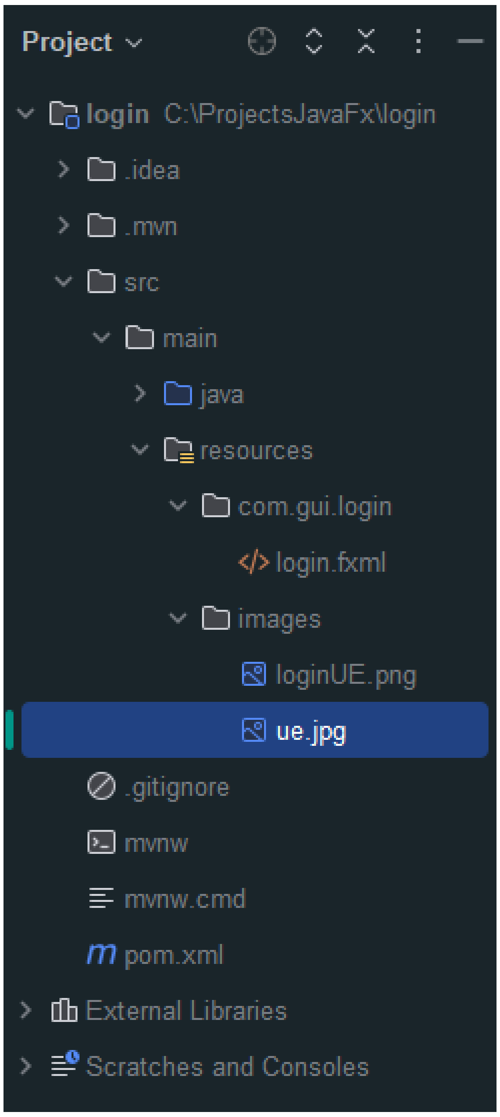
☑ Search for references

☑ Search in comments and strings

Scope:

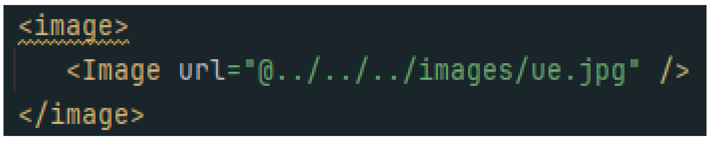
Project Files

Create a new folder in resource called images and place the images in it.

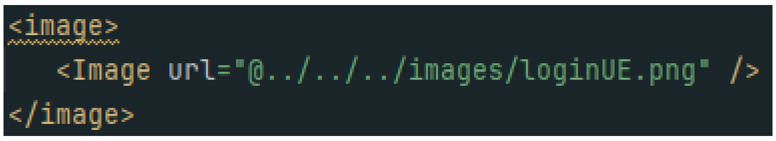


```
Project
login C:\ProjectsJavaFx\login
  .idea
  .mvn
  src
    main
      java
      resources
        com.gui.login
          login.fxml
        images
          loginUE.png
          ue.jpg
  .gitignore
  mvnw
  mvnw.cmd
  pom.xml
External Libraries
Scratches and Consoles
```

Update image paths in login.fxml file

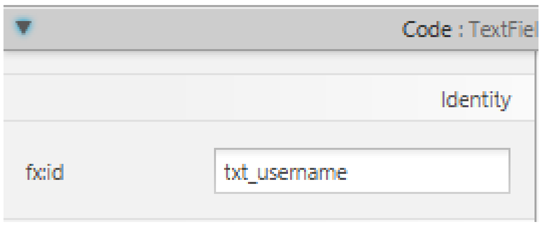


```
<image>
    <Image url="@../../../images/ue.jpg" />
</image>
```



```
<image>
    <Image url="@../../../images/loginUE.png" />
</image>
```

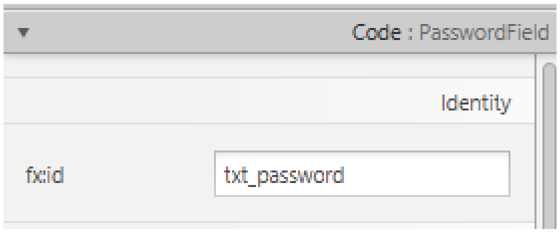
Add id to texts



Code : TextFiel

Identity

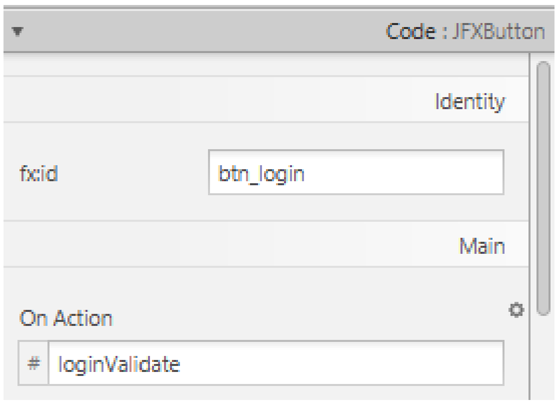
fx:id txt_username


Code : PasswordField

Identity

fx:id txt_password

Al button add id and OnAction


Code : JFXButton

Identity

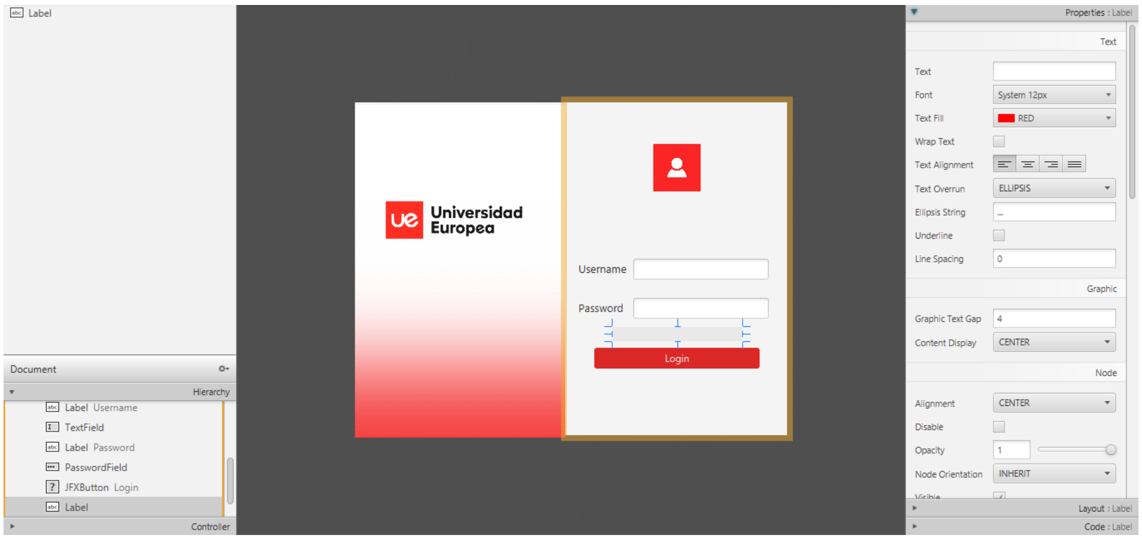
fx:id btn_login

Main

On Action

loginValidate

Add a label for validation that shows if the fields are empty, it 36ill show it in red



Document

Hierarchy

- Label Username
- TextField
- Label Password
- PasswordField
- JFXButton Login
- Label

Controller

Properties : Label

Text

Text	
Font	System 12px
Text Fill	🟥 RED
Wrap Text	☐
Text Alignment	
Text Overrun	ELLIPSIS
Ellipsis String	...
Underline	☐
Line Spacing	0

Graphic

Graphic Text Gap	4
Content Display	CENTER

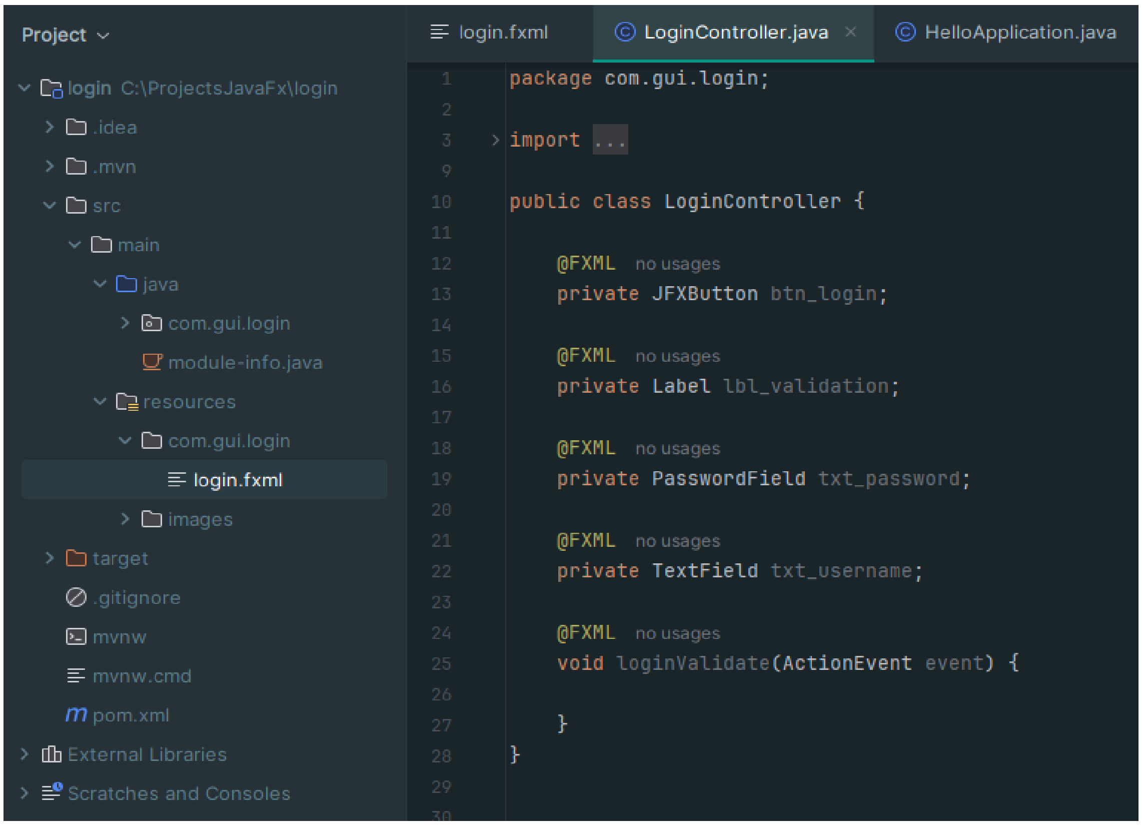
Node

Alignment	CENTER
Disable	☐
Opacity	1
Node Orientation	INHERIT

Layout : Label

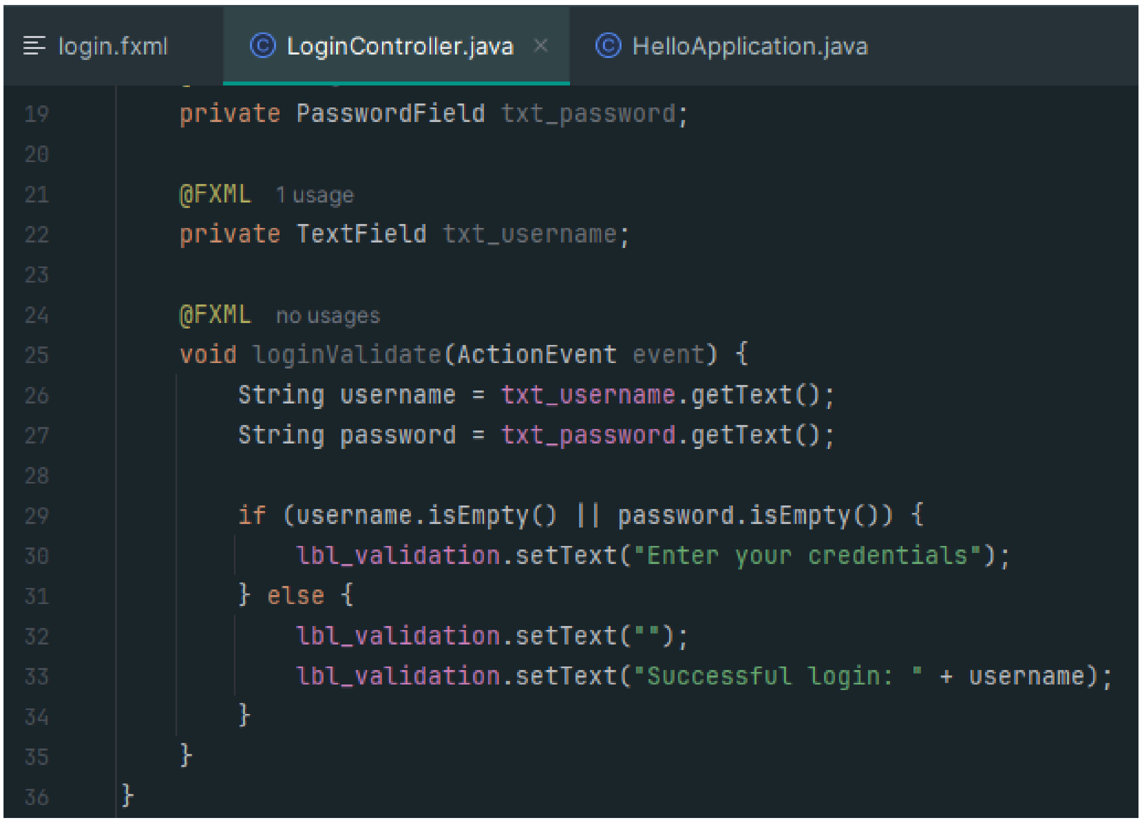
Code : Label

Add LoginController



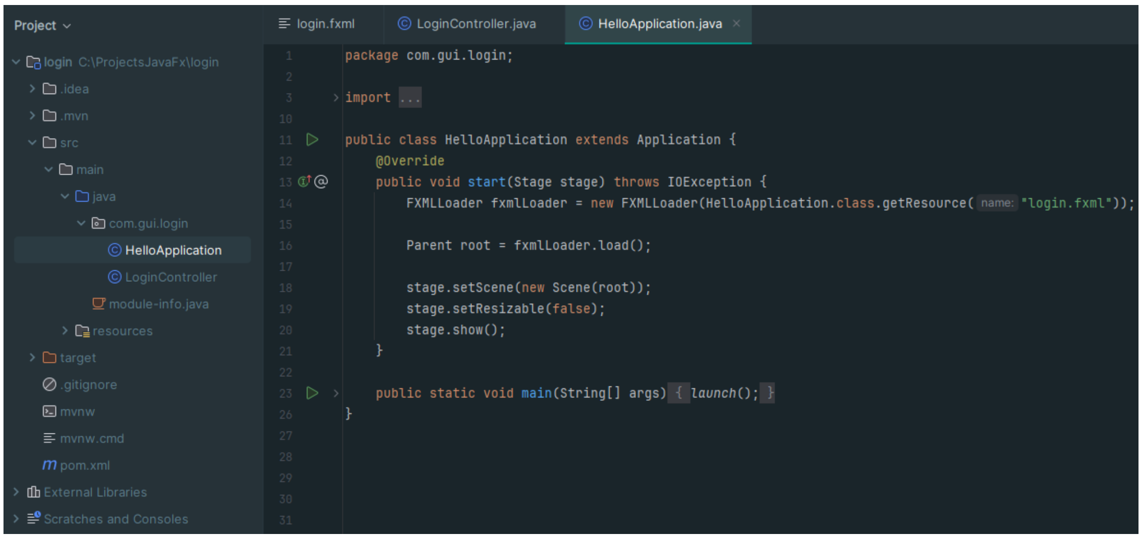
```java
package com.gui.login;

import ...

public class LoginController {

    @FXML  no usages
    private JFXButton btn_login;

    @FXML  no usages
    private Label lbl_validation;

    @FXML  no usages
    private PasswordField txt_password;

    @FXML  no usages
    private TextField txt_username;

    @FXML  no usages
    void loginValidate(ActionEvent event) {

    }
}
```

Adding logic in the controller to validate that no empty data is sent and assuming that you log on



```java
    private PasswordField txt_password;

    @FXML  1 usage
    private TextField txt_username;

    @FXML  no usages
    void loginValidate(ActionEvent event) {
        String username = txt_username.getText();
        String password = txt_password.getText();

        if (username.isEmpty() || password.isEmpty()) {
            lbl_validation.setText("Enter your credentials");
        } else {
            lbl_validation.setText("");
            lbl_validation.setText("Successful login: " + username);
        }
    }
}
```

Logic so that the screen cannot be accommodated.

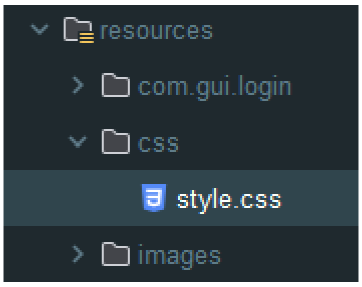


```java
package com.gui.login;

import ...

public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource( name: "login.fxml"));

        Parent root = fxmlLoader.load();

        stage.setScene(new Scene(root));
        stage.setResizable(false);
        stage.show();
    }

    public static void main(String[] args) { launch(); }
}
```

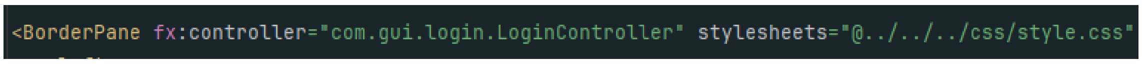
Using stage.setResizable(false); prevents the window from being resized by the user. By setting the value false, you disable the option to resize the window, which means that it 38ill remain at the dimensions you have defined without allowing the user to stretch or shrink it.

Apply css

Create folder in resources called css, and add new file style.css

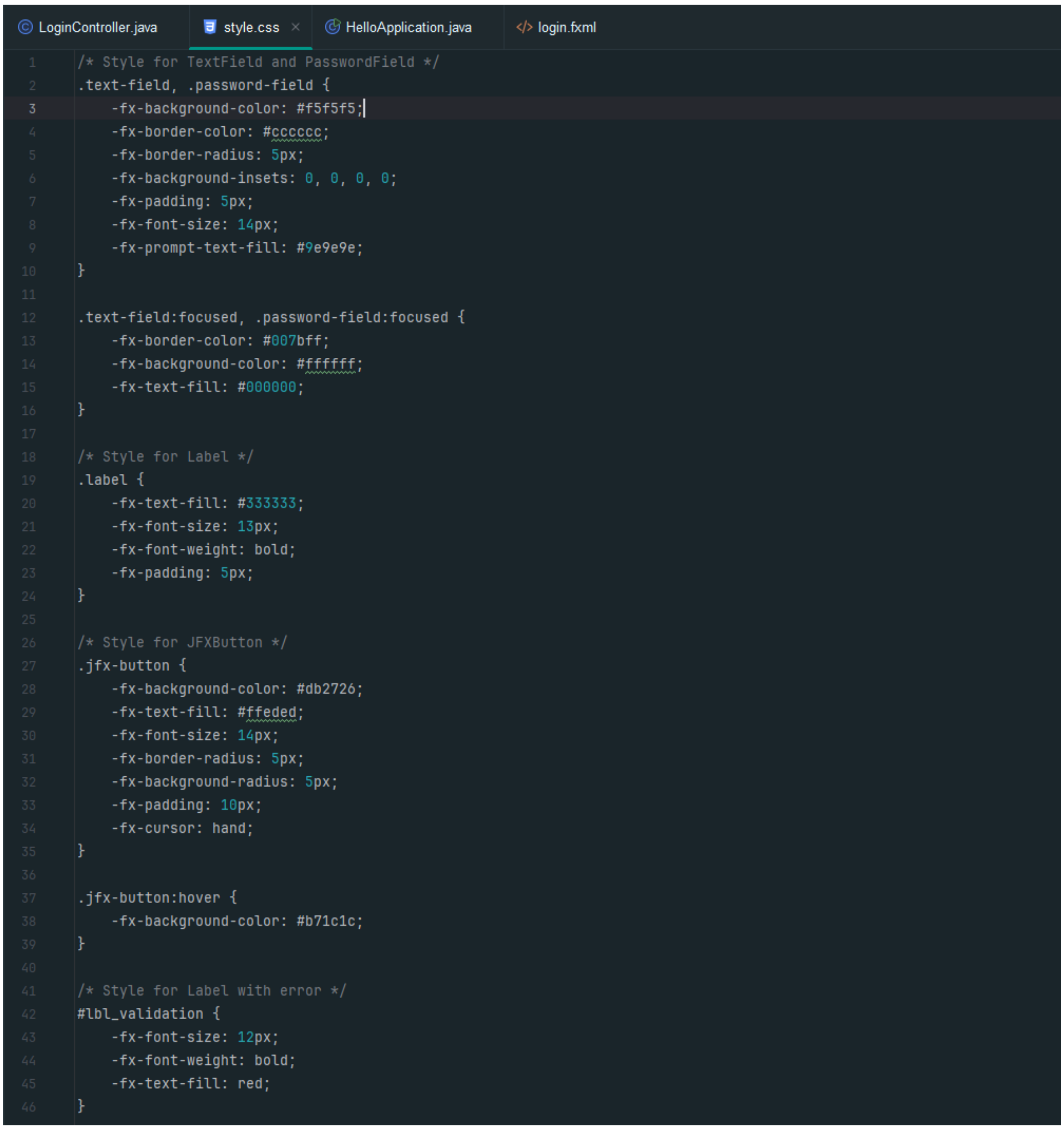


Link css file to fxml file



```xml
<BorderPane fx:controller="com.gui.login.LoginController" stylesheets="@../../../css/style.css"
```

css file



```css
/* Style for TextField and PasswordField */
.text-field, .password-field {
    -fx-background-color: #f5f5f5;
    -fx-border-color: #cccccc;
    -fx-border-radius: 5px;
    -fx-background-insets: 0, 0, 0, 0;
    -fx-padding: 5px;
    -fx-font-size: 14px;
    -fx-prompt-text-fill: #9e9e9e;
}

.text-field:focused, .password-field:focused {
    -fx-border-color: #007bff;
    -fx-background-color: #ffffff;
    -fx-text-fill: #000000;
}

/* Style for Label */
.label {
    -fx-text-fill: #333333;
    -fx-font-size: 13px;
    -fx-font-weight: bold;
    -fx-padding: 5px;
}

/* Style for JFXButton */
.jfx-button {
    -fx-background-color: #db2726;
    -fx-text-fill: #ffeded;
    -fx-font-size: 14px;
    -fx-border-radius: 5px;
    -fx-background-radius: 5px;
    -fx-padding: 10px;
    -fx-cursor: hand;
}

.jfx-button:hover {
    -fx-background-color: #b71c1c;
}

/* Style for Label with error */
#lbl_validation {
    -fx-font-size: 12px;
    -fx-font-weight: bold;
    -fx-text-fill: red;
}
```

TextField and PasswordField: Set a light background color, a gray border, and a rounded border radius. Add internal padding so that the text does not touch the borders. Fields change color when in focus to highlight the blue border.

Label: Sets the text color to a dark gray, a font size of 13px and a bold font weight to highlight the labels.

JFXButton: Customize the button with a red background color, white text, and a rounded border. The button changes color to a darker red when the mouse is hovered over it.

Label with error: Sets a smaller font size and bold font to display error messages in red.

Advantages of using CSS vs. styling elements from the Scene Builder's own options

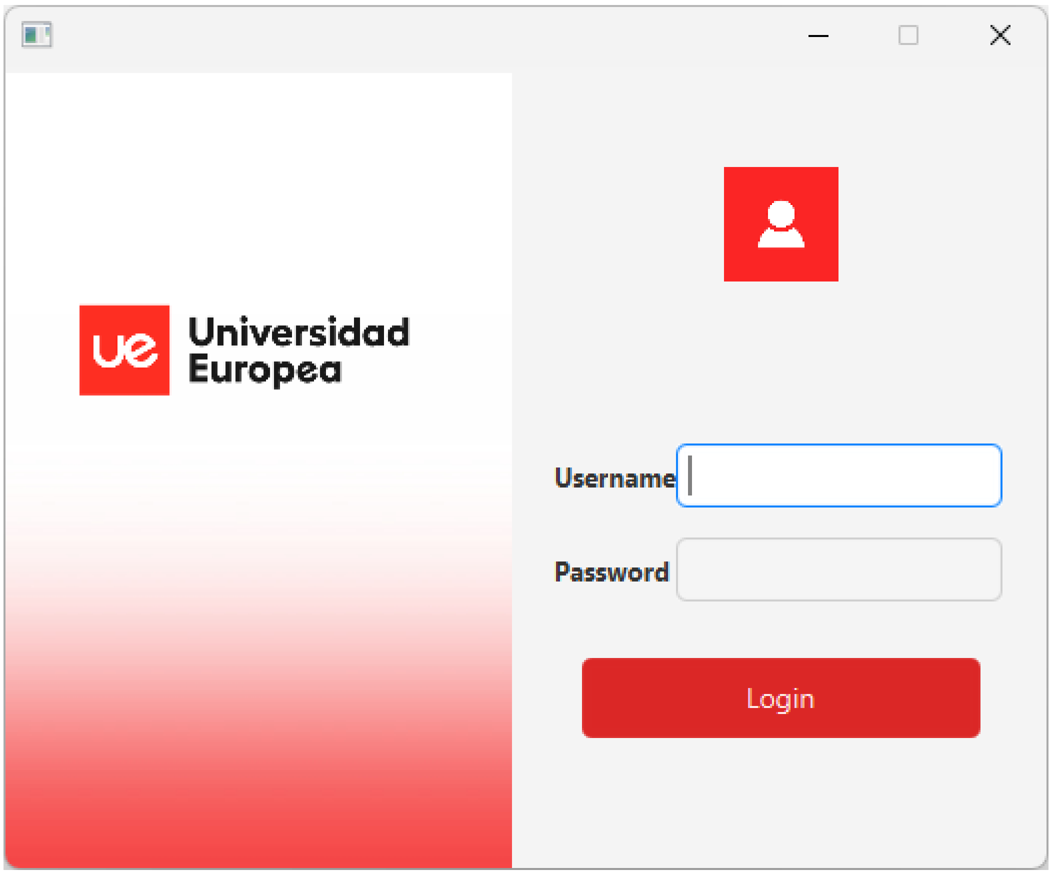
Advantages of using CSS:

- **Reusability:** Define styles once and apply them to multiple elements.
- **Flexibility:** Advanced customization and responsive designs.
- **Separation:** Separate CSS for better organization.

Disadvantages of using Scene Builder for layouts:

- **Limited:** Fewer customization options than CSS.
- **Maintainability:** Can be difficult to manage for complex layouts.

When running it



Notes

IntelliJ IDEA

If you install the latest available version of IntelliJ IDEA (2024.2.1), you may have problems when using Scene Builder, since the IDE suggests to install it, but it does not do it correctly and does not advance beyond that point. For this reason, we have chosen to work with an older version of IntelliJ IDEA (2023.3.8).

Java

If you are using Java 8, you are likely to encounter difficulties during installation in IntelliJ IDEA. Therefore, it has been decided to use JDK version 17.0.1.

Installation

Remember to save everything you download and install, such as JFoenix, JDK and other tools, in a specific folder created to manage these resources. This will make it easier for you to find the necessary paths when configuring your project, such as adding the JFoenix .jar file to the libraries and modules to use its components.