Practical Sheet nº7

Content

- Event-oriented Programming
- Introduction to JavaFX
- JavaFX Architecture, Controls & Containers
- JavaFX Scene Builder
- Skinning with CSS

In this class, we will create our first Graphical User Interface (GUI) using JavaFX and connect it to the Logic Layer developed in Scala.

7 – Event-oriented Programming with JavaFX

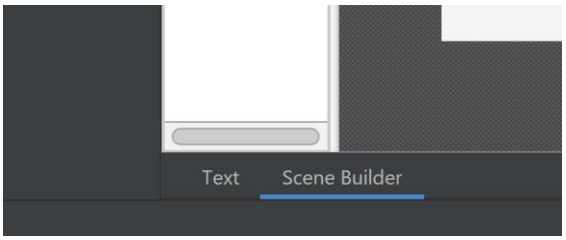
7.1 – Hello World application (tutorial)

To create a JavaFX GUI in an IDEA Scala project with the IntelliJ IDE, you should perform the following steps:

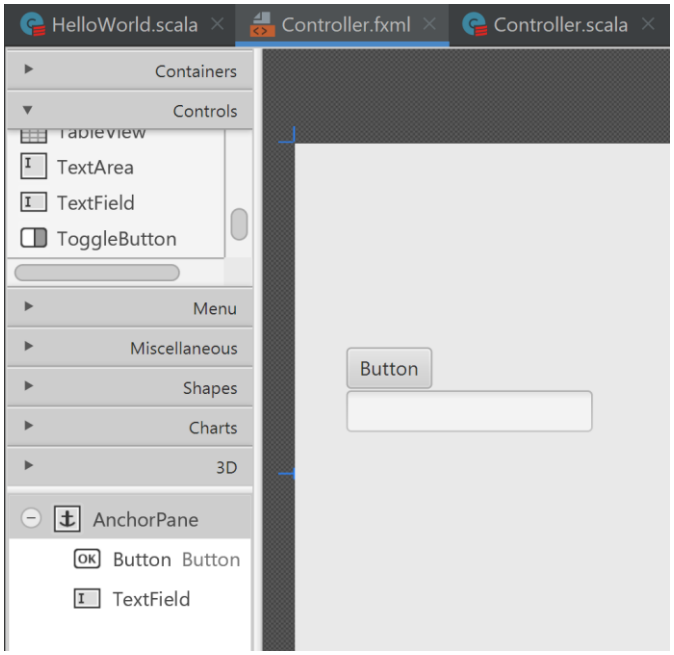
- Add the Scala SDK 2.13.12 (or higher i.e., 3.2.2) to the libraries:
  - File/Project Structure…/Libraries/+/Download Scala SDK/OK
- Download JavaFX 15.0.1 SDK (or higher, i.e., 20) from: https://gluonhq.com/products/javafx/
  - Add it to the libraries:
    - File/Project Structure…/Libraries/+/Java/"*path*\openjfx-15_windows-x64_bin-sdk\javafx-sdk-15\lib"
- Download JavaFX SceneBuilder from IntelliJ:
  - File/Settings/Plugins/ write JavaFX and select: "JavaFX Runtime for Plugins"
- Add a New Scala Class (HelloWorld.scala) in the *src* folder
- Add a New FXML file (Controller.fxml) in the *src* folder
- Add a New Scala Class (Controller.scala) in the *src* folder
- Create a method named *onButton1Clicked* in "Controller.scala"

```
class Controller {
    def onButton1Clicked(): Unit = {
      println("Hello World")
    }
}
```

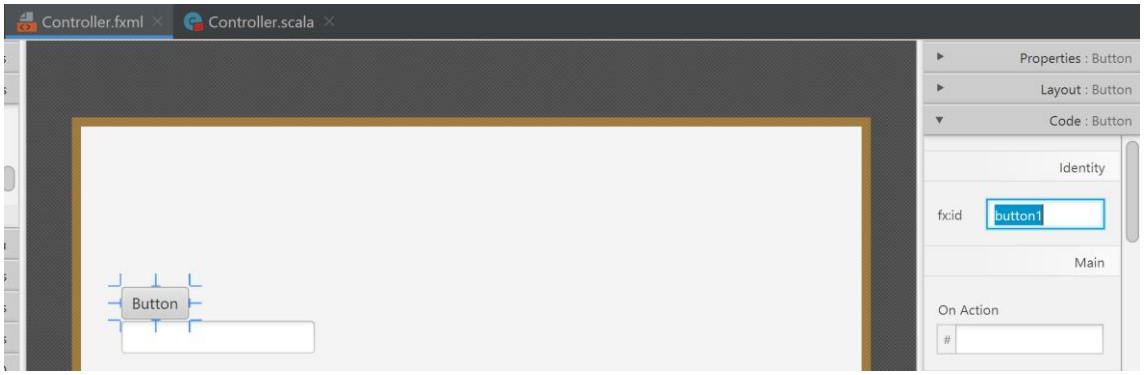
- Open the FXML file created (Controller.fxml) using the Scene Builder view (present at the bottom):



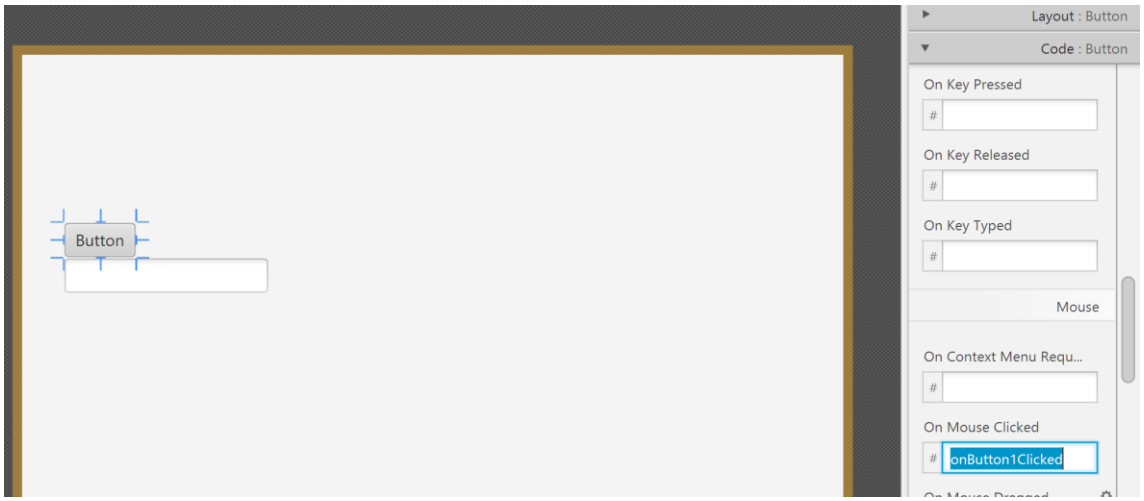
- Add a Button and a TextField (drag and drop) to the Scene (see below):



- Add *fx IDs* to the Button (fx ID: *button1*) and TextField (fx ID: *textField1*) created (see below):



- Add a name for the event handler (onButton1Clicked) when the mouse is clicked on the button (see below):



- Add attributes to the controller (Controller.scala) with the same name used in the ID fields of the FXML (see below):

```
import javafx.fxml.FXML
import javafx.scene.control.{Button, TextField}

class Controller {
  @FXML
  private var button1: Button = _
  @FXML
  private var textField1: TextField = _

  def onButton1Clicked(): Unit = {
     println("Hello World")
  }
}
```

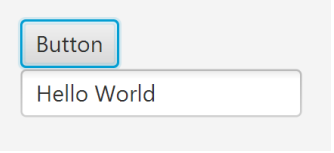
- Complete the HelloWorld class (see below):

```
import javafx.application.Application
import javafx.fxml.FXMLLoader
import javafx.scene.{Parent, Scene}
import javafx.stage.Stage
```

```
class HelloWorld extends Application {


  override def start(primaryStage: Stage): Unit = {
    primaryStage.setTitle("My Hello World App")
    val fxmlLoader =
        new FXMLLoader(getClass.getResource("Controller.fxml"))
    val mainViewRoot: Parent = fxmlLoader.load()
    val scene = new Scene(mainViewRoot)
    primaryStage.setScene(scene)
    primaryStage.show()
  }
}


object FxApp {
  def main(args: Array[String]): Unit = {
    Application.launch(classOf[HelloWorld], args: _*)
  }
}
```

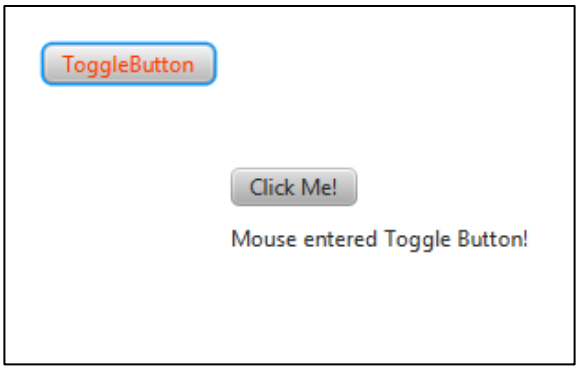
- Run the *FxApp* object
  - Click the button present in the GUI to see the "Hello World" printed in the console.
  - Change the implementation of the `onButton1Clicked` method to see the "Hello World" printed in the textField (see below):

```
def onButton1Clicked(): Unit = {
      textField1.setText("Hello World")
}
```



7.2 – Widget properties and Event handlers

1. Complete the source code provided to obtain the following GUI (see below). The *label* below the "Click Me!" button (with no associated behavior) should only be presented when the cursor enters the Toggle Button. In addition, clicking the *Toggle Button* should change the associated button text to "Clicked" and its opacity to 50%.



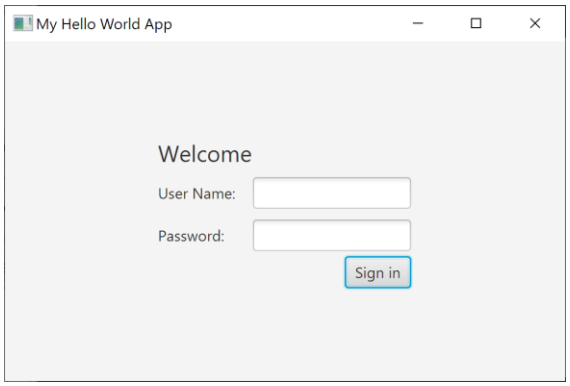
2. Create a GUI with the same appearance as in the figure below. The radio buttons used cannot be selected simultaneously (use *ToggleGroup*).
Hint: use "*-fx-background-color*" and "*GREEN*" in JavaFX CSS Style property to change the background color of a widget to green color.

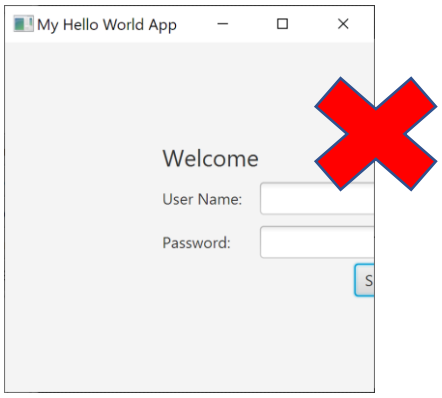
7.3 – Responsive GUI and connection to the logic layer

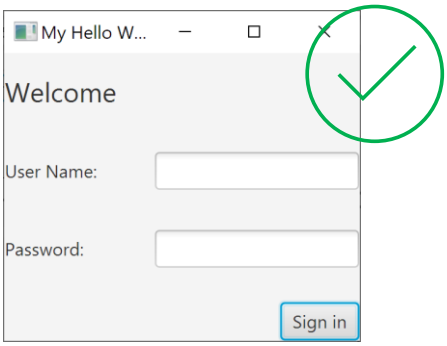
A responsive GUI is a graphical user interface that adjusts smoothly to various screen/window sizes to ensure content consistency across devices.

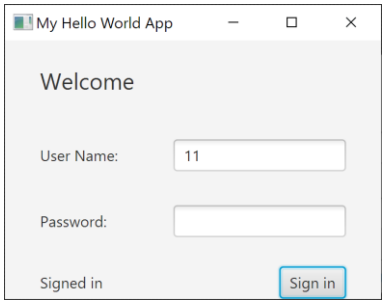
Create a responsive form (using a *GridPane*) with the appearance of the one in the figure below. A "signed in" message should be presented in case the user exists in the *Turma* (from Ficha5); otherwise, a "user does not exist" message should appear (create a new label for this purpose). To simplify the search, assume that the information introduced as User Name in the *textField1* is the student number and do not consider the password field (see figures below).

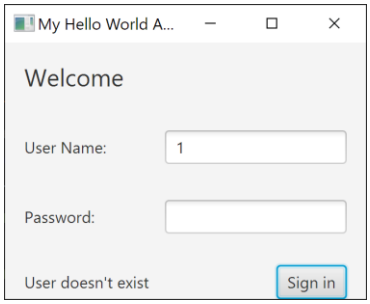
Start by opening the "Controller.fxml" in the Scene Builder view and interactively perform the changes.











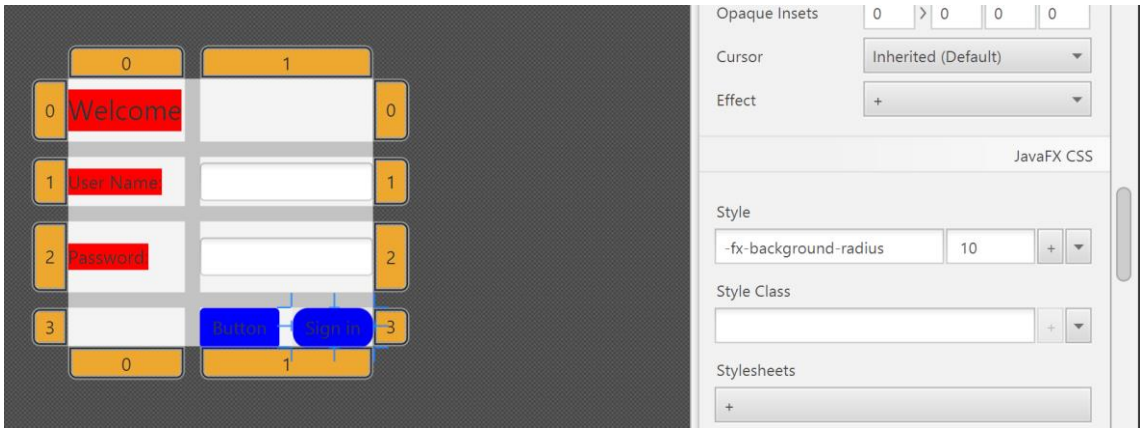
Update the *FxApp* object with a *Turma* object (see below) and the *onButton1Clicked* method present in the controller to work as expected.

```
object FxApp {

  var t: Turma = Turma("id22", List((11,"as",RegimeOPT.Ordinario, Some(1), Some(20))
,(12, "Jose", RegimeOPT.TrabEstud, Some(13), None)))

  def main(args: Array[String]): Unit = {

    Application.launch(classOf[HelloWorld], args: _*)

  }

}
```

7.4 – Skinning with CSS

The JavaFX UI controls used by Scene Builder are pre-styled with a default JavaFX look and feel. Scene Builder immediately renders this predefined JavaFX style when you drag the UI control from the Library panel to the Content or Hierarchy panel. You can customize the style used in your application by changing the component's properties via the Properties section of the Inspector panel or by defining your own styling rules in a CSS file.

1. In the Scene Builder view, click the Style *textField* property in the JavaFX CSS section of the Properties column, where you can find the syntax you can use for each JavaFX CSS property's value.
2. Select the button and select "-fx-background-radius" and add 10 as the value in the Style *textField* property in the JavaFX CSS section of the Properties column.
   Notice that the Button is now rounded around the corners (see the figure below).



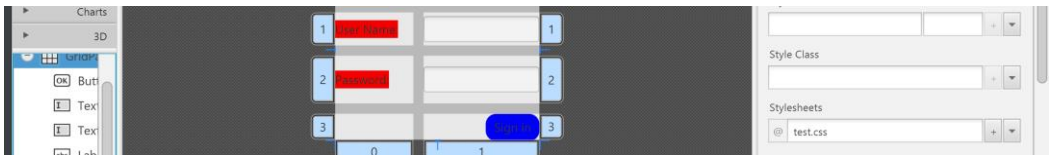
3. Create a CSS rule to apply the new button style to all the buttons you add to your current FXML layout (i.e., *GridPane*).

   a. Create an empty CSS file, e.g., test.css, in your project.

   b. Add the following CSS rule to test.css and save the file.

```
.button {
  -fx-background-radius: 10px;
}
```

   c. In the Properties section of the *GridPane*, click the + button in the Stylesheets and add the test.css file.



d. Try now to add a new button in the grid and see the result.

e. Add a new rule in the test.css file for the labels (e.g., to change the background color to RED) and see the result.