**Sofia University**
**Department of Mathematics and Informatics**

**Course : Applied OO Programming part 1**
**Date: April 21, 2020**
**Student Name:**

**Lab No. 9**

**Problem 1**

1. Open the Modular Java project **`DrawingBoard`** (see attached **`DrawingBoard.rar`**) in IntelliJ. Add module named **`model`** with package **`geometry`** to this project and create the following classes in package **`geometry`** (the module **requires** access to JavaFX controls):
   - Create a module-info descriptor for this module and add statements allowing access to JavaFX controls
   - Write a class **Point**. It **has** an array of two integer data members- the **x** and **y** coordinates. Define a **full set of constructors** (default, general purpose and a copy constructor), **set** and **get** methods for the class data members, a **set** method with a **Point argument** and a **get** method **returning a Point object**, as well as a **toString**() method.
   - Next, write a class **Line**. A **Line is a Point sPoint and has** a data member **Point ePoint denoting respectively the** starting and the ending **Point** of the line. Define a **full set of constructors** (default, general purpose and a copy constructor), **set** and **get** methods for the class data members, as well as a **toString**() method (reuse the **toString**() method defined for class Point). Write additionally a ***measure()*** method returning the length of the **Line**. Write additionally a ***`draw(Pane pane)`*** method allowing to draw ***this*** rectangle in the ***`Container`*** node of a JavaFX ***`Scene`***, referenced by ***`pane`***
   - Finally, write a class **Rectangle**. **Rectangle is a Point, which defines the *upper left corner* and has a Point** that **defines** the ***lower right corner*** of the rectangle. Define a **full set of constructors** (default, general purpose and a copy constructor), **set** and **get** methods for the class data members, as well as a **toString**() method (reuse the **toString**() method defined for class **Point**). Write additionally a ***measure()*** method returning the perimeter of the **Rectangle**. Write additionally a ***`draw(Pane pane)`*** method allowing to draw ***this*** **Rectangle** in the ***`Container`*** node of a JavaFX ***`Scene`***, referenced by ***`pane`***.

2. Add content to the event handler methods of buttons of ***`MainSceneController.java`*** in package ***`view`*** of module **`gui`** allowing to draw line and rectangle object that are instances of the respective classes in module

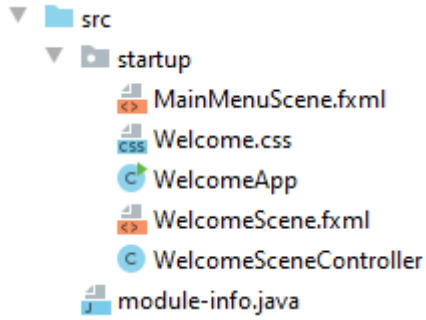
**model**. Draw all the geometric objects in the Pane (**pnlDrawingBoard)** on the right side of ***MainScene*** using coordinates for the Point object generated at random employing class Random (the coordinates must be selected to fall inside the current width and height of **pnlDrawingBoard)** .

**Problem 2 (*teaches Java modular projects, switching context inside a Stage and inside a Scene as well as using Stylesheets with Scene Builder*)**

Create a Java modular project having two modules **login** and **app,** create dependency of both modules to the JavaFX library

**A. Add content to module login**

Create package **startup** with two empty FXML files and two Java files as shown below



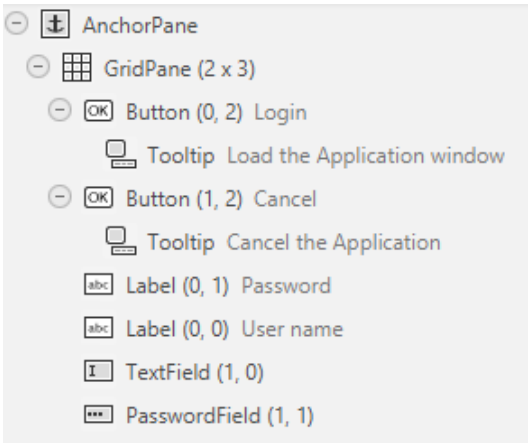

**Add** a **module descriptor** for **login** allowing to execute this FXML application as per the instructions in Lecture 9. Read the attached file **Styles.pdf** and learn to declare the different types of CSS selectors (class, built-in class, ID and pseudo classes) with JavaFX.

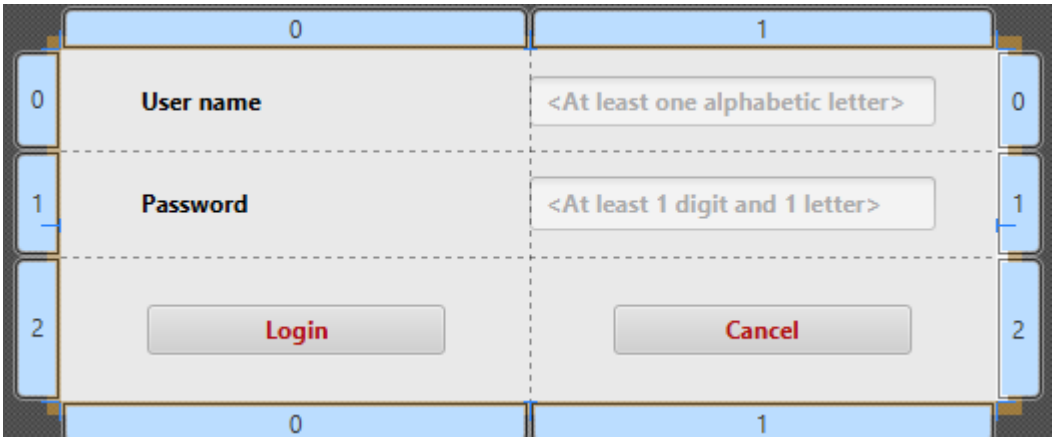
**Add a Stylesheet** to package **startup** named **Welcome.css** with the following data

```
.grid {
    -fx-background-color: #cbd3d0;
    -fx-padding: 8;
    -fx-hgap: 8;
    -fx-vgap: 8;
}
#label {
    -fx-font-size: 12px;
    -fx-font-weight: bold;
    -fx-padding: 0 0 0 40;
    -fx-text-fill: #000000;
}
#text-field {
    -fx-font-size: 12px;
    -fx-font-weight: bold;
    -fx-text-fill: #000000;
}
#password-field {
    -fx-font-size: 12px;
    -fx-font-weight: bold;
    -fx-text-fill: #000000;
}
#button {
    -fx-text-fill: #b80f0f;
    -fx-font-size: 12px;
```

```
    -fx-font-weight: bold;
    -fx-border-radius: 16;
}
```

Design a login window with JavaFX Scene and Controller defined respectively, by **WelcomeScene.fxml** and **WelcomeSceneController.java**. Reproduce the following design



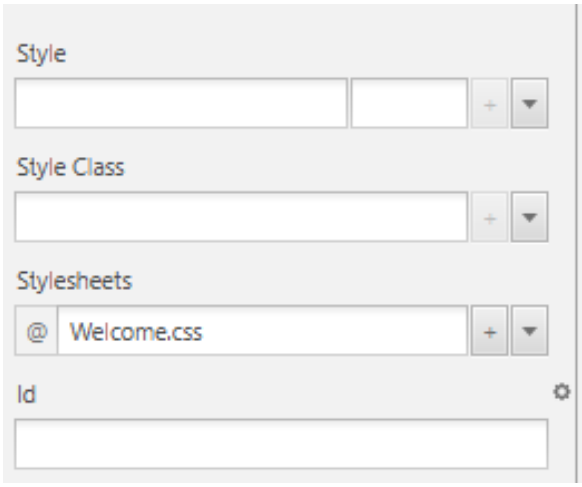




Button **Cancel** quits the application, Button **Login** validates the user input with regular expressions:

- Username must include only alphabetic letters(uppercase or lowercase), at least one
- Password must include only digits AND leters (at least 1 letter and at lleast 1 digit)

Write the respective validation rules in the **OnAction** event handler for button **Login**

Attach the **Welcome.css** stylesheet to the **GridPane** in Scene Builder

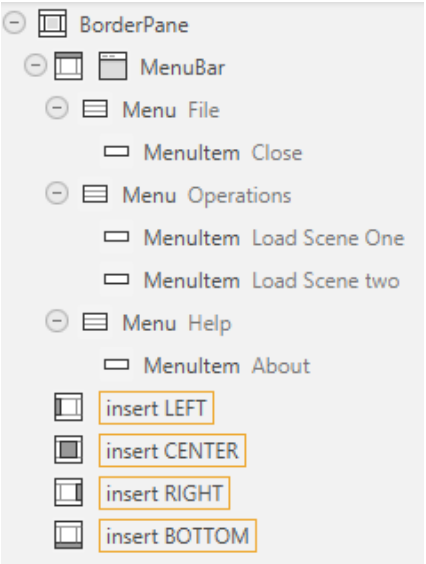


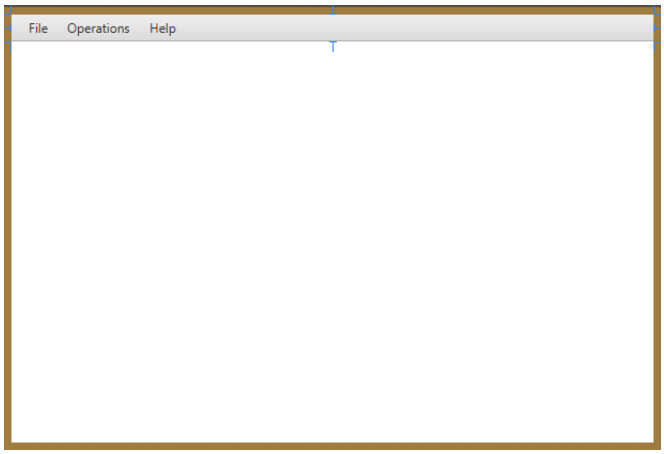
## B. Add content to module `app`

Create a package **gui** in module **app**. Design the JavaFX scene for the main application with **MainMenuScene.fxml**, where the respective controller **gui.MainMenuSceneController.java** is in package **gui.**

Add a **module descriptor** for `app` allowing to execute this FXML application as per the instructions in Lecture 9.

Reproduce the following design with **MainMenuScene.fxml**




Update the **OnAction** event handler for button **Login** in **WelcomeSceneController.java** to execute the following block of code in case the user input matches the regular expression rule **to switch the scene in the login window with the scene** of **MainMenuScene.fxml**

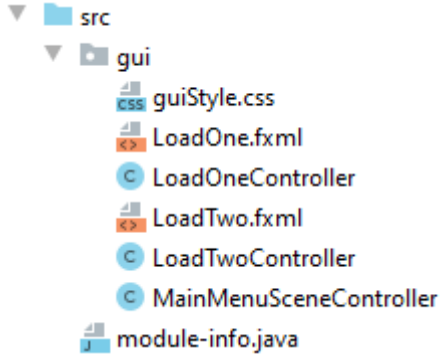
```
{// get current stage (window)
    mainMenu =  (Stage) txtPassword.getScene().getWindow();
    mainMenu.hide();  // close current stage, avoid blink
    // read new scene
    FXMLLoader loader = new FXMLLoader();
    Parent root = null;
    try {
        loader.setLocation(getClass()
              .getResource("/gui/MainMenuScene.fxml"));
        root = (Parent) loader.load(getClass()
                                .getResource("MainMenuScene.fxml")
                                .openStream());
    } catch (IOException e) {
        e.printStackTrace();
    }
    // Get the Controller from the FXML scene
     // setup the newly read scene
    MainMenuSceneController menuController =
                    (MainMenuSceneController) loader.getController();
    // switch scenes
    Scene scene = new Scene(root);
    mainMenu.setScene(scene);
    mainMenu.setTitle("Student dashboard");
    mainMenu.setResizable(false);
    mainMenu.show();
}
```

where `mainMenu` is a user defined datamember of class **MainMenuSceneController**

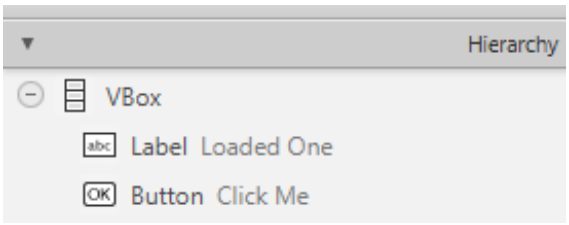
**Add a Stylesheet** to package **gui** named **guiStyle.css** with the following data

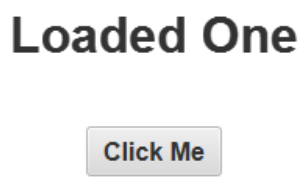
```
Label{
   -fx-font: bold 24pt "arial";
}

Button{
    -fx-font: bold 12pt "arial";
}
```

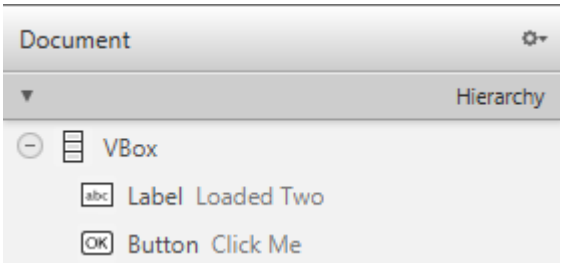
**Add two empty FXML files** and respective **controllers** to package **gui**

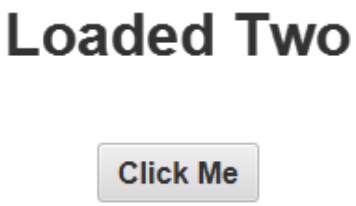



We use these FXML files to add simple content to the main application window. The design of these scenes has minor differences allowing just to distinguish them as follows:

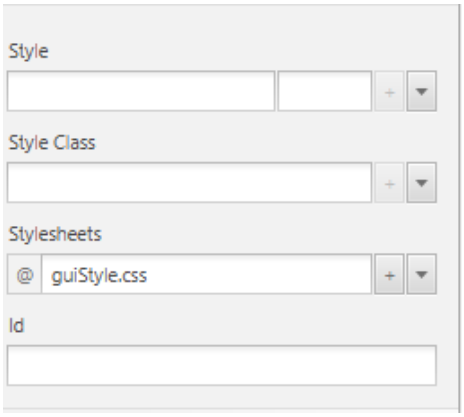


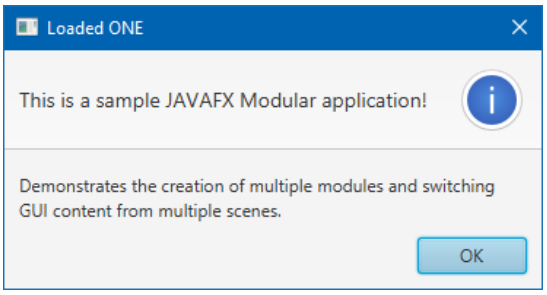



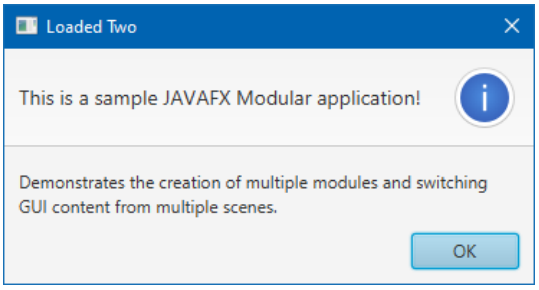




To achieve the above look- and- feel attach the **guiStyles.css** stylesheet to the **Vbox-es** of both FXML document from Scene Builder



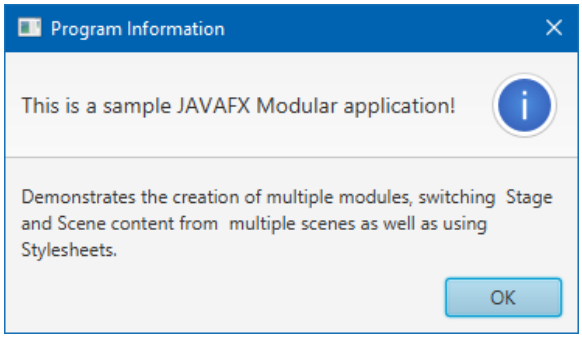
The **OnAction** event handlers for the **Click Me** buttons display **Alert** message box




Add **OnAction** event handlers for the menu items in **WelcomeSceneController.java as follows:**

- Click on **Close** must close the FXML application
- Click on **About** displays Alert message box




- Click on **Load Scene One** loads below the main menu the scene defined by LoadOne.fxml
- Click on **Load Scene Two** loads below the main menu the scene defined by LoadTwo.fxml

In order **to switch the content** below the Menubar in **MainMenuScene.fxml write the following OnAction** event handlers **for** MenuItems **Load Scene One** and **Load Scene Two**

```
loadContent("LoadOne.fxml");
```

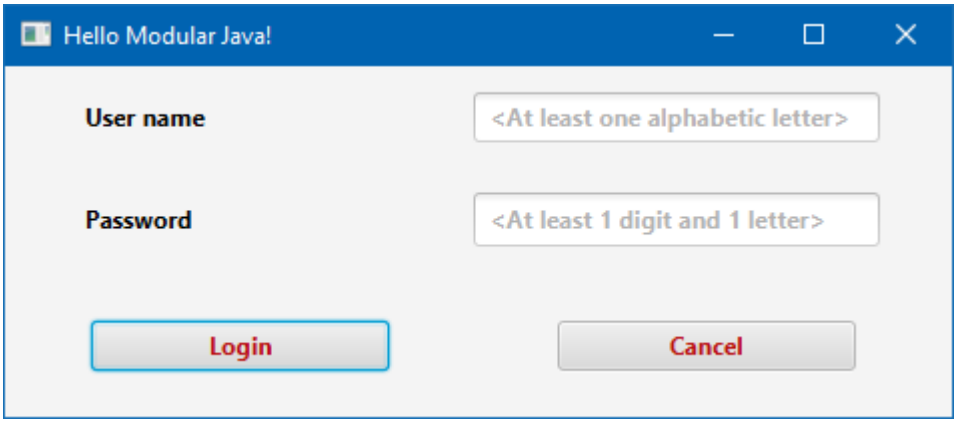
and

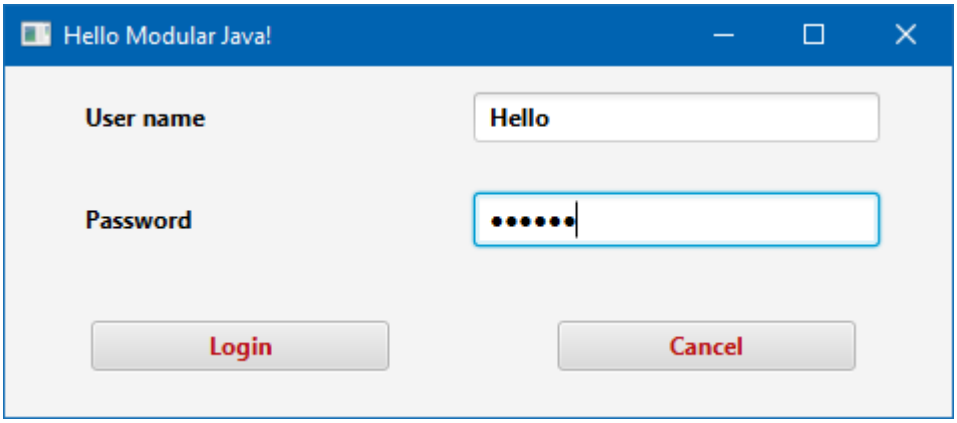
```
loadContent("LoadTwo.fxml");
```

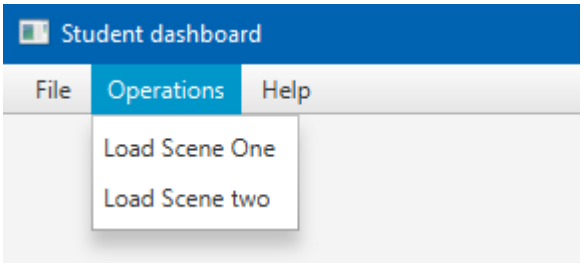
**where**

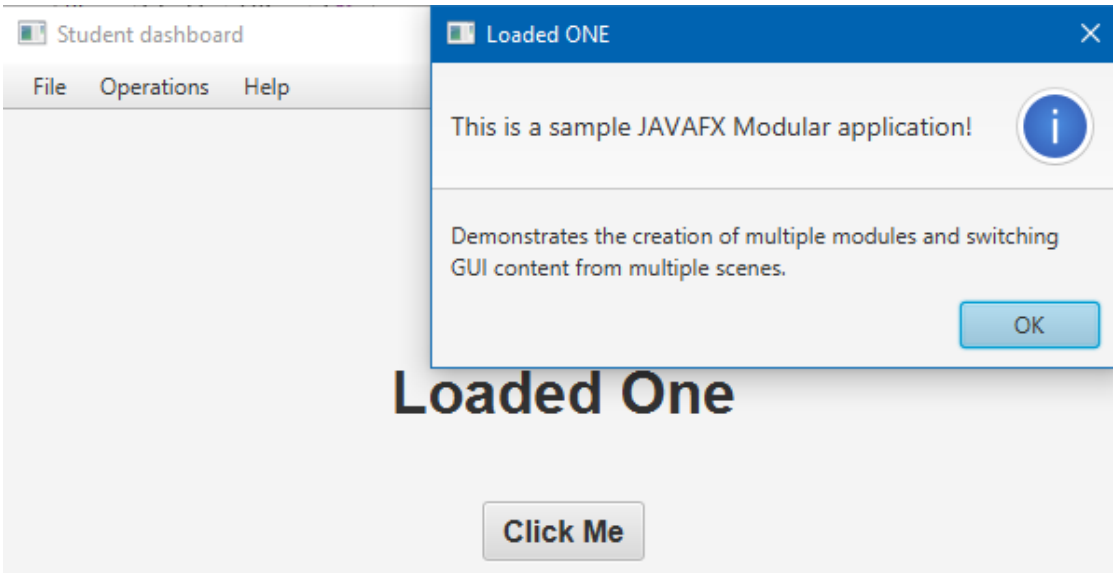
```
private void loadContent(String fxmlFileName) {
    FXMLLoader loader = new FXMLLoader();
    VBox root = null;
    try {
        loader.setLocation(getClass().getResource(fxmlFileName));
        root = (VBox) loader.load(getClass()
                                  .getResource(fxmlFileName)
                                  .openStream());
    } catch (IOException e) {
        e.printStackTrace();
    }
    bpContent.setCenter(root);
}
```

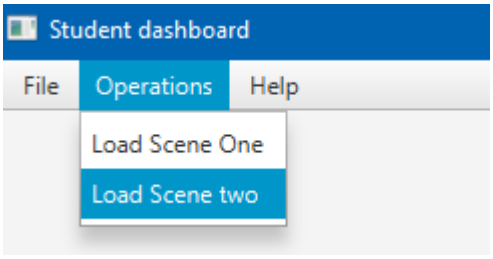
Run and test the Java modular application

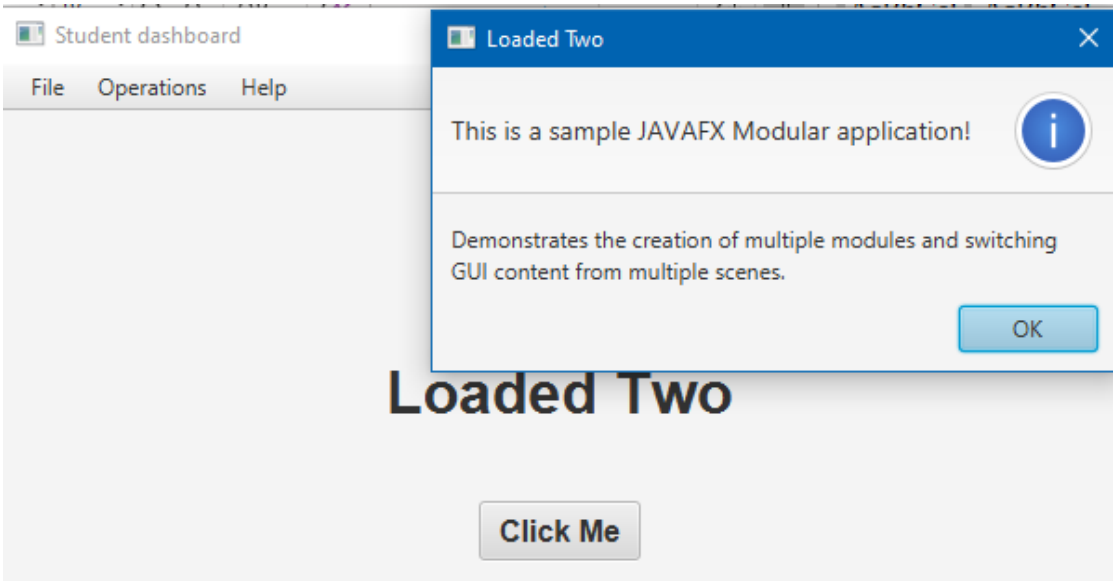

Hello Modular Java!
User name
<At least one alphabetic letter>
Password
<At least 1 digit and 1 letter>
Login
Cancel


Hello Modular Java!
User name
Hello
Password
Login
Cancel


Student dashboard
File
Operations
Help
Load Scene One
Load Scene two


Student dashboard
File
Operations
Help
Loaded ONE
This is a sample JAVAFX Modular application!
Demonstrates the creation of multiple modules and switching GUI content from multiple scenes.
OK
Loaded One
Click Me

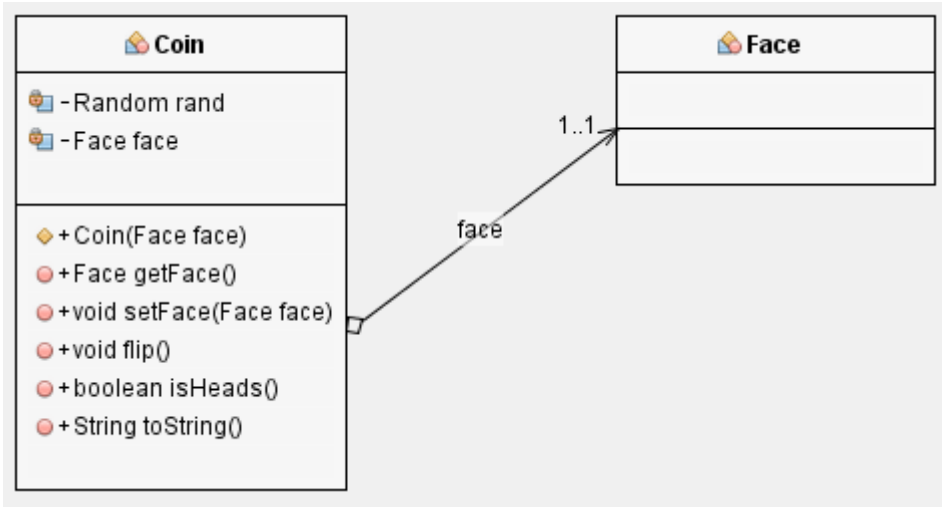



**Important**: **Use meaningful identifiers for FXML controls and event handlers according to the Modified Hungarian Notation i.e use well-known prefixes btn-, txt-, mnu- and so on for controls and suffix -OnAction for OnAction event handlers. Use comments**.

**Problem 3**

Given are class Coin and enum Face with constants HEAD and TAIL

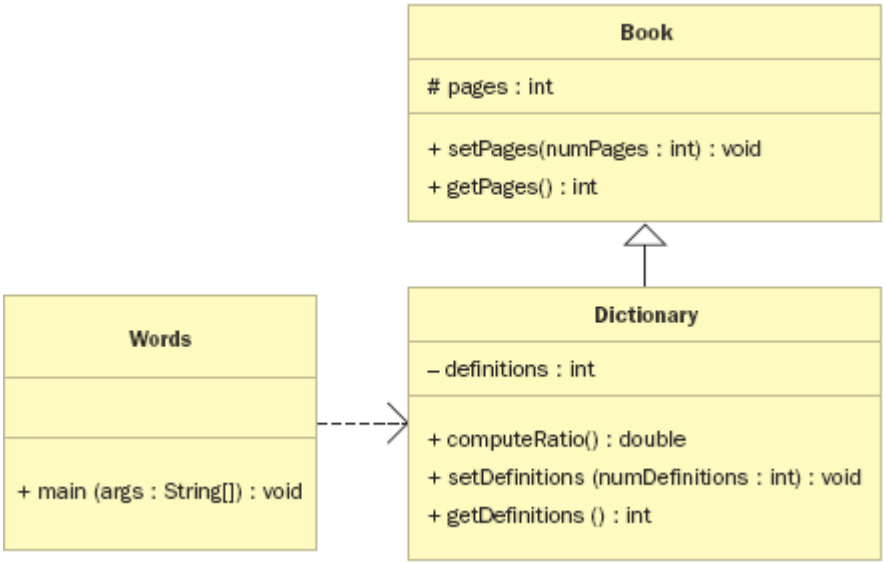



Write a class called MonetaryCoin that is derived from the Coin class. Store an integer in the MonetaryCoin that represents its value and add a method that returns its value. Create a separate class with a main method to instantiate and compute the sum of several

MonetaryCoin objects. Demonstrate that a monetary coin inherits its parent's ability to be flipped.

**Problem_4**

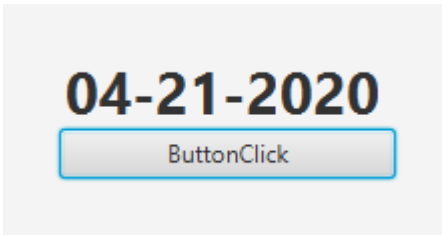
Implement the classes on the following UML diagram




**Problem No 5a**

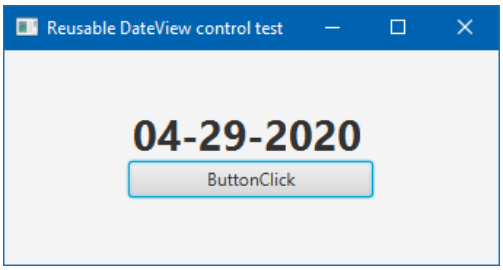
Create a Java FXML **non-modular** project having the following simple design:

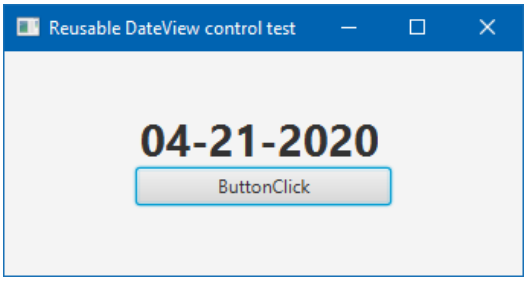

Where the Label displays the text 04-29-2020 by default. On clicking **ButtonClick** the Label makes use of **LocalDate** class to display the *current date* formatted by **DateFormatter** as "MM-dd-yyyy"




Package the JavaFX graph tree with root the AnchorPane as a resuable visual JavaFX control named **DateViewControl** in a JAR file. Add the control to the **Library** of Scene Builder.

**Create a new Java FXML *non*-modular project** with **BorderPane** node for a Parent node. Drag the control in the **Center** of the **BorderPane**, set up the **Application** template and run the FXML application




**Problem No 5b**

**Create a** calculator as a **non-modular JavaFXML resuable visual** ***component*** in Scene Builder. The **Calculator** should allow the user to input numbers in a textbox and choose an operation to perform on them (addition, multiplication, division, subtraction) with **Buttons** as it is done with a usual calculator (*see the design of the* **Calculator** *application in the Accessories Program group in the MS Windows environment*) . Design the **Layout** of the buttons and the textbox to execute these operations, as well as, add support for handling the following events:

a) to remember the currently displayed number (**M** operation)

b) to add the currently displayed number with the number stored in memory and display the result (**M+** operation)

c) to subtract the currently displayed number with the number stored in memory and display the result (**M-** operation)

d) to clear the memory (**MC-** operation

The methods performing the Calculator operations must be **public.** There should be also **two *public set* properties** for the user numeric input, necessary to complete the calculator operations**.** There should be a ***public get*** **property** for the Calculator result.

Package the resulable component as a JAR file and add it to the Scene Builder Library. The visual component must be accessible for *click- drag- and- drop* from the Custom section of Scene Builder

**Write a non-modular JavaFXML application** making use of the **reusable Calculator component**.