Week 9

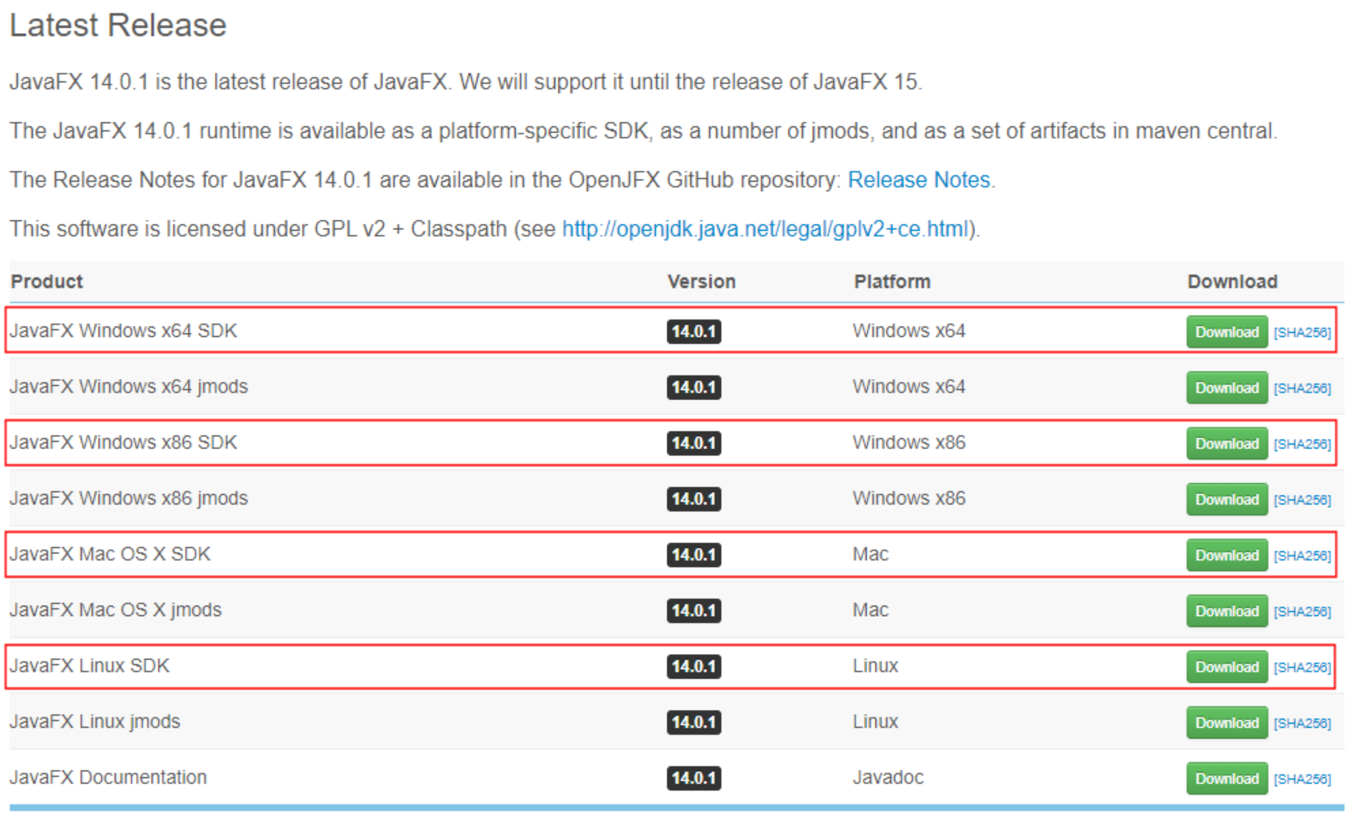
Comment

The purpose of this practical is to introduce you to developing graphical user interfaces (GUIs) with JavaFX. JavaFX is a library that allows the creation of graphical desktop applications in Java.

1 JavaFX Setup

Before Java 11, JavaFX was included with the JDK; however, now it must be installed as a standalone library. This comes with some additional steps in order to get up and running:

1. Download JavaFX from `https://gluonhq.com/products/javafx/`. Ensure you download the correct version for your operating system. (You may also download the JavaDoc for JavaFX if you would like easy access to it. It is the last link shown in the screenshot below.)



Latest Release

JavaFX 14.0.1 is the latest release of JavaFX. We will support it until the release of JavaFX 15.

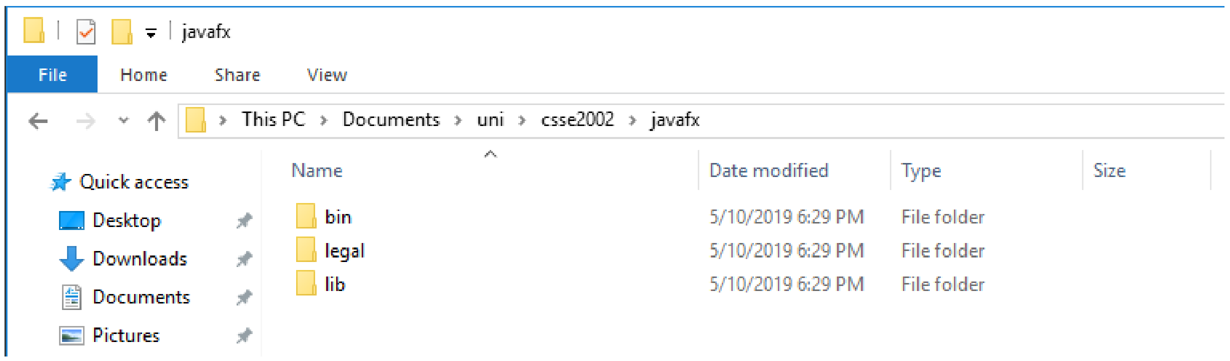
The JavaFX 14.0.1 runtime is available as a platform-specific SDK, as a number of jmods, and as a set of artifacts in maven central.

The Release Notes for JavaFX 14.0.1 are available in the OpenJFX GitHub repository: Release Notes.

This software is licensed under GPL v2 + Classpath (see http://openjdk.java.net/legal/gplv2+ce.html).

Product	Version	Platform	Download
JavaFX Windows x64 SDK	14.0.1	Windows x64	Download [SHA256]
JavaFX Windows x64 jmods	14.0.1	Windows x64	Download [SHA256]
JavaFX Windows x86 SDK	14.0.1	Windows x86	Download [SHA256]
JavaFX Windows x86 jmods	14.0.1	Windows x86	Download [SHA256]
JavaFX Mac OS X SDK	14.0.1	Mac	Download [SHA256]
JavaFX Mac OS X jmods	14.0.1	Mac	Download [SHA256]
JavaFX Linux SDK	14.0.1	Linux	Download [SHA256]
JavaFX Linux jmods	14.0.1	Linux	Download [SHA256]
JavaFX Documentation	14.0.1	Javadoc	Download [SHA256]

2. Once the download is complete, extract the SDK to a memorable location on your computer (On EAIT lab computers, put this in your H drive).



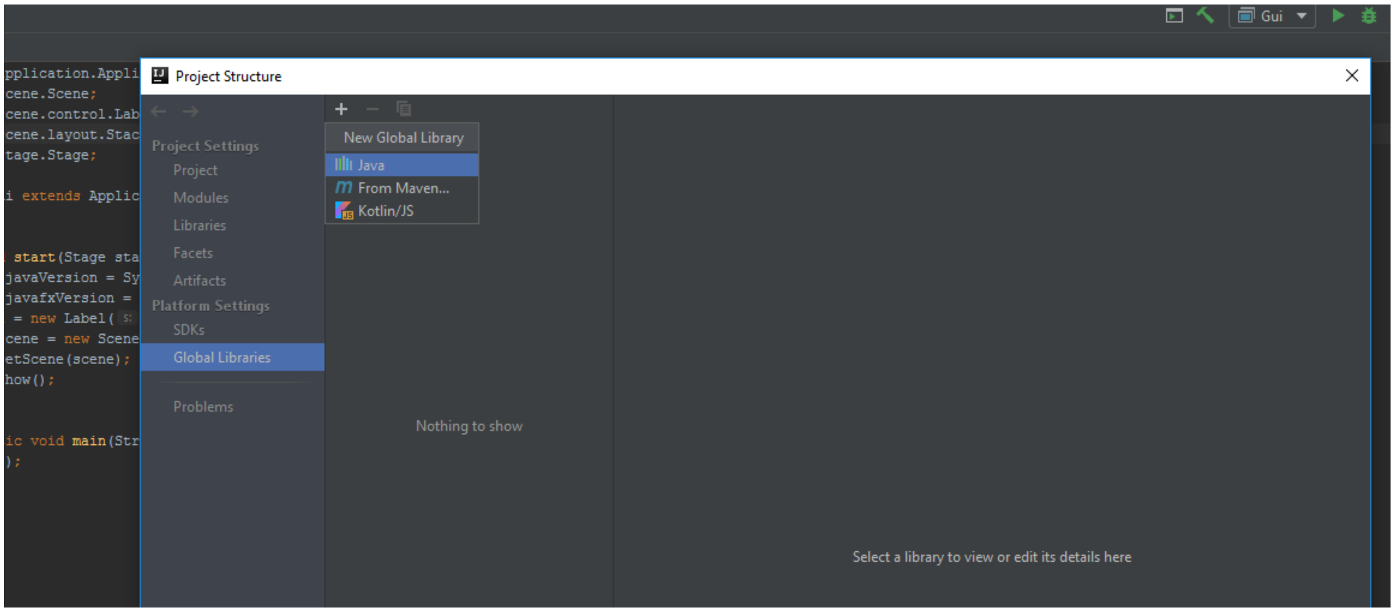
Name	Date modified	Type	Size
bin	5/10/2019 6:29 PM	File folder	
legal	5/10/2019 6:29 PM	File folder	
lib	5/10/2019 6:29 PM	File folder	

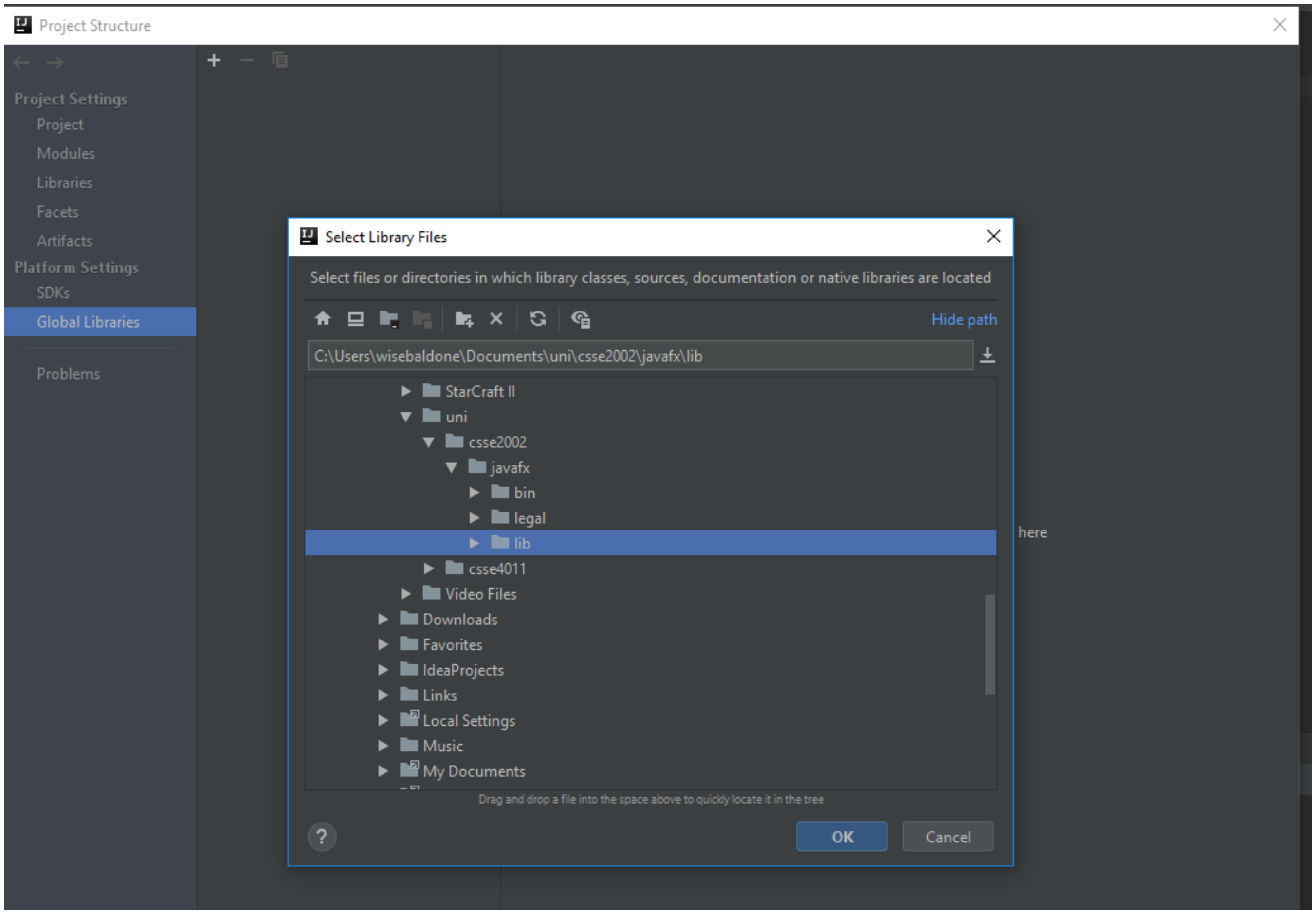
3. Create a new normal Java project in IntelliJ and create a class called `HelloFX` that contains the following code below:

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloFX extends Application {

    @Override
    public void start(Stage stage) {
        String javaVersion = System.getProperty("java.version");
        String javafxVersion = System.getProperty("javafx.version");
        Label l = new Label("Hello, JavaFX " +
            javafxVersion + ", running on Java " +
            javaVersion + ".");
        Scene scene = new Scene(new StackPane(l), 640, 480);
        stage.setScene(scene);
        stage.show();
    }

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

4. Add the libraries you extracted before as a global library for your new project by going to File -> Project Structure -> Global Libraries.

5. Add the library by clicking the '+' symbol and selecting Java as the New Global Library navigate to the `lib` folder within the OpenJFX SDK you extracted earlier.





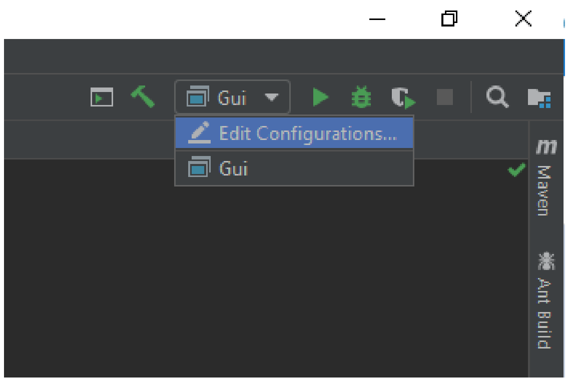
Select files or directories in which library classes, sources, documentation or native libraries are located

```
C:\Users\wisebaldone\Documents\uni\csse2002\javafx\lib
```

Drag and drop a file into the space above to quickly locate it in the tree

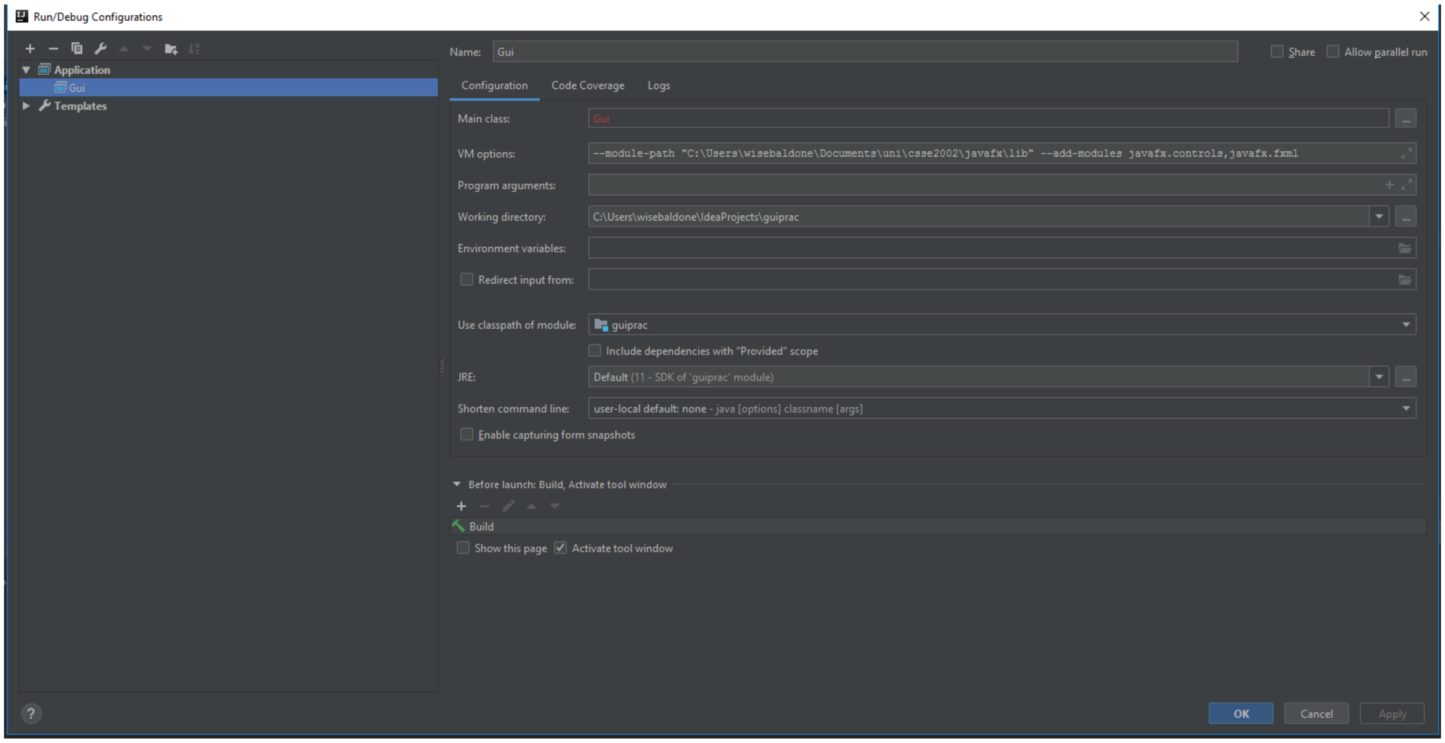
6. Click the play button for your `HelloFX` class. This should produce an error.

7. To fix this we need to edit the configuration of the program by selecting it from the "Run" dropdown menu and selecting "Edit Configurations...".



8. Add the below arguments to "VM Options":
   `--module-path "[location of SDK]" --add-modules javafx.controls,javafx.fxml`

   You will need to change the module path to the actual location of your JavaFX SDK. See the below screenshot for an example for Windows:



Name: Gui

Main class: Gui

VM options: `--module-path "C:\Users\wisebaldone\Documents\uni\csse2002\javafx\lib" --add-modules javafx.controls,javafx.fxml`

Program arguments:

Working directory: C:\Users\wisebaldone\IdeaProjects\guiprac

Environment variables:

☐ Redirect input from:

Use classpath of module: guiprac

☐ Include dependencies with "Provided" scope

JRE: Default (11 - SDK of 'guiprac' module)

Shorten command line: user-local default: none - java [options] classname [args]

☐ Enable capturing form snapshots

☐ Show this page ☑ Activate tool window

9. You should now be able to run the application and a window should display successfully.

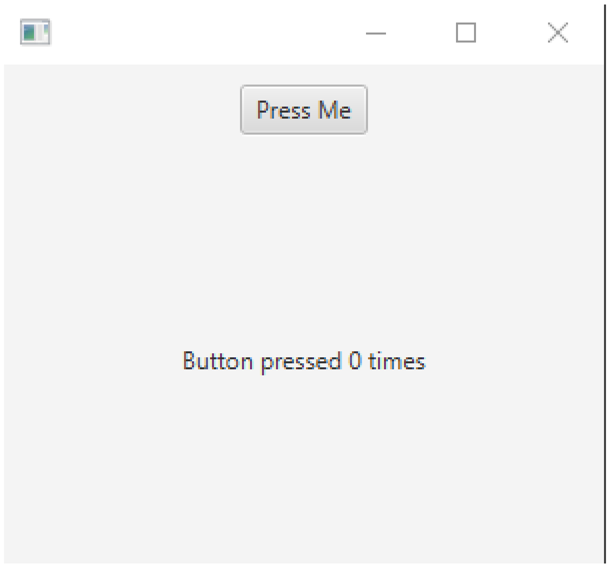
2 Simple Application

In this exercise you will create a small GUI (Graphical User Interface) which will contain a `Button` and `Label`.

The function of this GUI is simply to keep track of how many times the button is pressed. You may create the GUI however you like, using any layout of your choice.

JavaFX has several built-in layouts which you can use to create a JavaFX Application. A guide on how to use them can be found here: `https://docs.oracle.com/javase/8/javafx/layout-tutorial/builtin_layouts.htm` and the JavaDoc for the Java 14 version of JavaFX is located here: `https://openjfx.io/javadoc/14/`

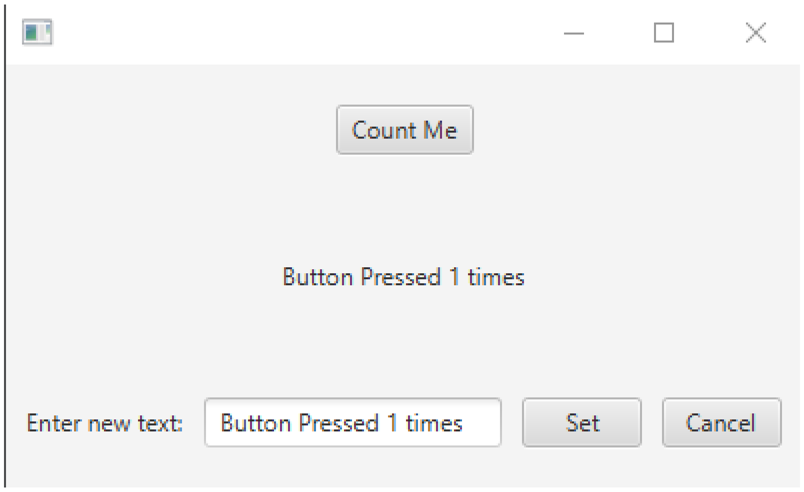
A possible layout for this GUI is given below - you may wish to replicate this or create your own.



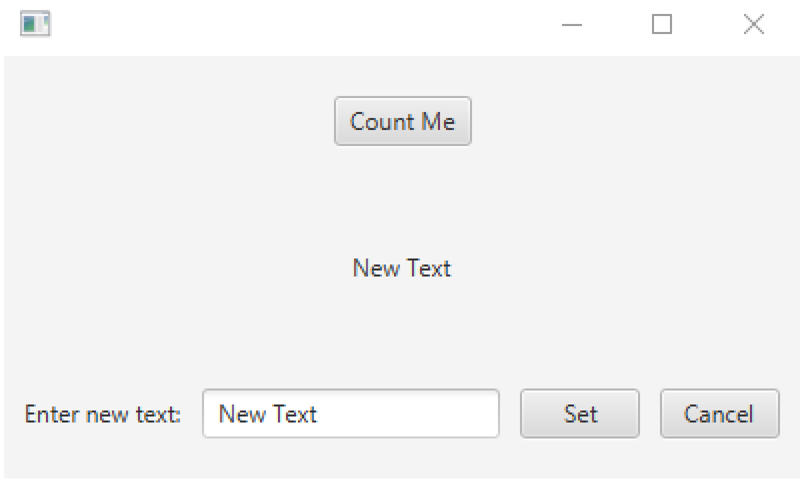
Each time you press the "Press Me" button, the label should be updated appropriately. For example, if the button is pressed 5 times since launch, then the label should read "Button pressed 5 times".

3 Extended Application

In this exercise you will extend the previous simple GUI application. We will now add a `TextField` into which a user can enter a comment and two `Button`s to set the `Label` displayed in the middle of the window.



Like in the previous exercise, each time you press the "Count Me" button, the label should be updated appropriately. Additionally, It should update the text field to display how many times the "Count Me" button has been pressed, and it should echo the output to the console.



A user may enter text into the text field. If they press the "Set" button, the label should be updated to display the text the user has entered. If they press the "Cancel" button, the contents of the textfield should be set to be "Cancelled" and the label should be changed to "Cancelled". The updated label contents should be echoed to the console.

Neither the "Set" or "Cancel" button should modify the count kept of how many times the "Count Me" button has been pressed.

4 MVVM

JavaFX provides a powerful concept known as "properties". These properties serve as wrappers for common types such as `String`, `Integer`, and `List`. Properties are particularly useful in GUI applications, where the appearance of a given graphical element depends on some internal state in the program.

For example, the content of a `Label` displaying the user's current score in a game could be linked to a score variable in the game model. JavaFX properties can be used to ensure this score label is automatically updated whenever the score variable changes, through a process called "binding".

Two JavaFX properties can be bound together, either uni-directionally or bi-directionally. With bi-directional binding, a change in the first property will be reflected instantly in the second, and vice versa.

There are many types of properties provided by JavaFX, a list can be found here: `https://openjfx.io/javadoc/14/javafx.base/javafx/beans/property/Property.html`

Convert the extended application from the previous question to use a `StringProperty`. `StringProperty` is an abstract class, so you will need to create an instance of one of its subclasses, such as `SimpleStringProperty`. You will need to bind this `StringProperty` to the label and the text field.

Try changing the binding to be bi-directional and see what effect it has on the label and text field. Try to understand why this happens.

Acknowledgements

This practical was original made by Abhi Jagtap in 2018 with adjustments and additions by the Course Staff.