Lab 07: GUI Design

Aims:

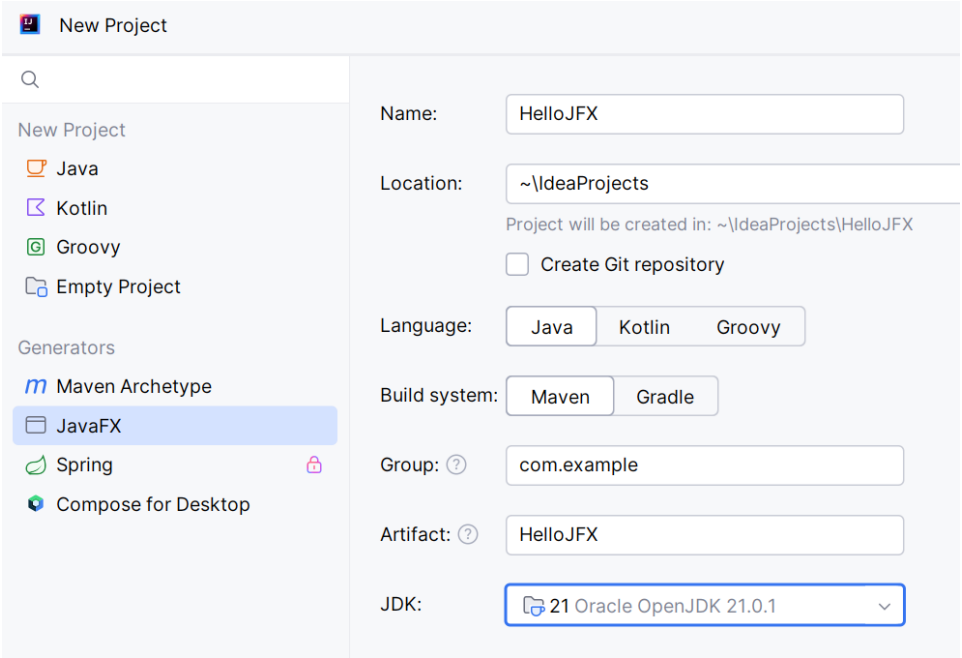
1. Exercise on IntelliJ with basic GUI designs, using JavaFX and Scene Builder.
2. Understand the key components of the JavaFX, including properties, event handling, event listener, etc.

---

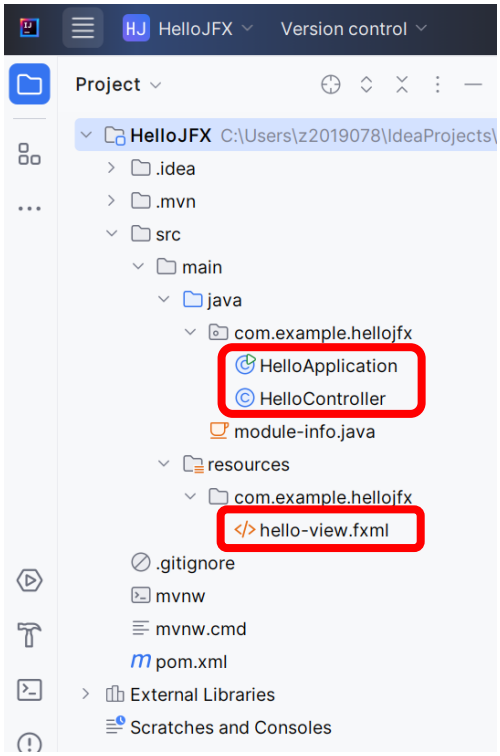
1. Set up a JavaFX project

Recall: in Lab 1, we have already installed and configured the JavaFX library and SceneBuilder that run with IntelliJ. If you haven't done so, please review the Lab1 material for installation instructions, both available on Moodle.

To start a JavaFX project in IntelliJ, go to "File" – "New Project" – "JavaFX", then input a project name such as HelloJFX, then complete the creation. Currently there is no need to select any dependencies.

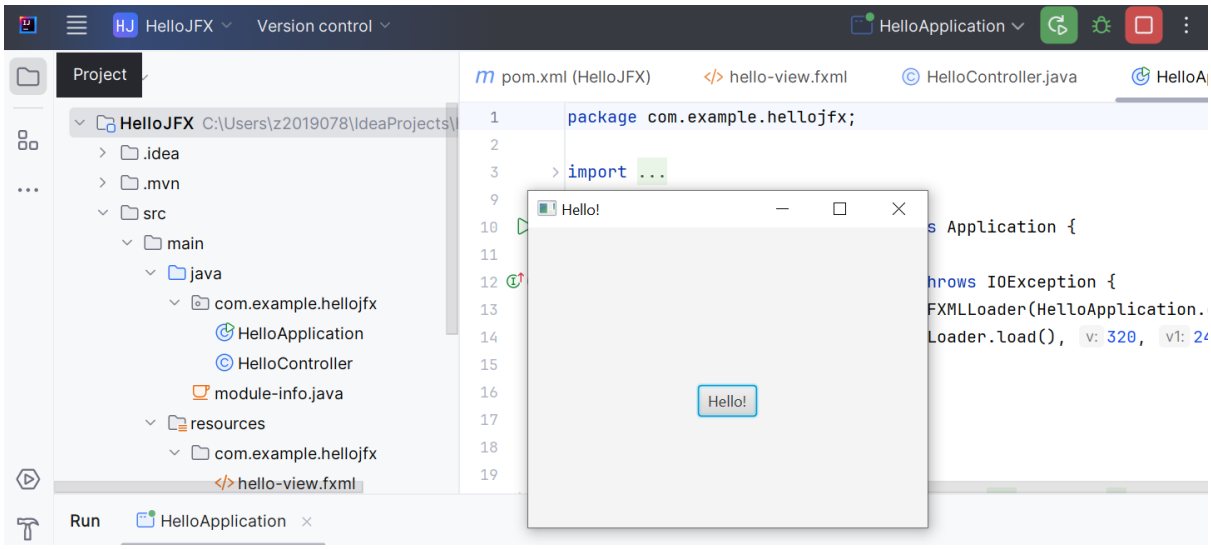


You should see that IntelliJ creates a 'java' directory and a 'resources' directory containing three files enclosed in red color below.

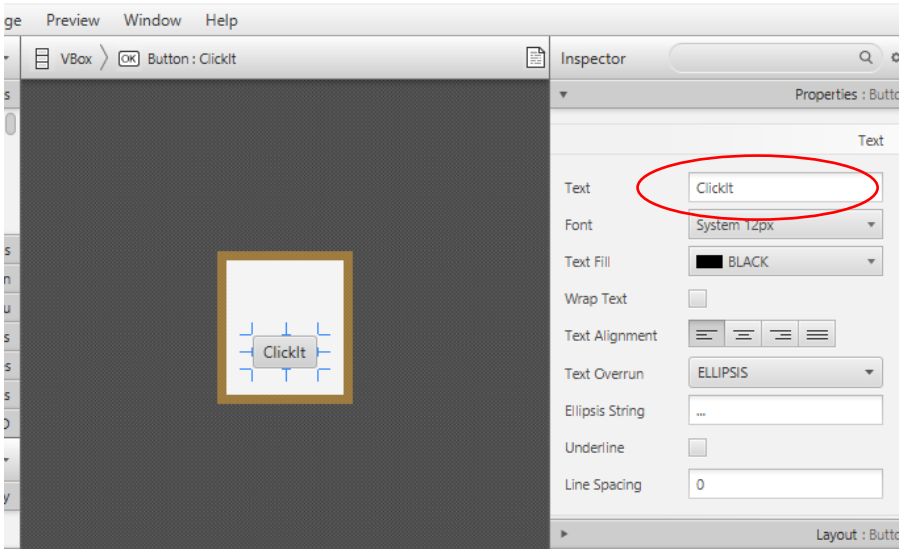


Observe that: This is already an MVC template with Model defined in 'HelloApplication', View defined in 'hello-view.fxml', and Controller defined in 'HelloController'. IntelliJ wants you to *maintain* your GUI-based projects. We will exercise on the MVC pattern in next week's lab.

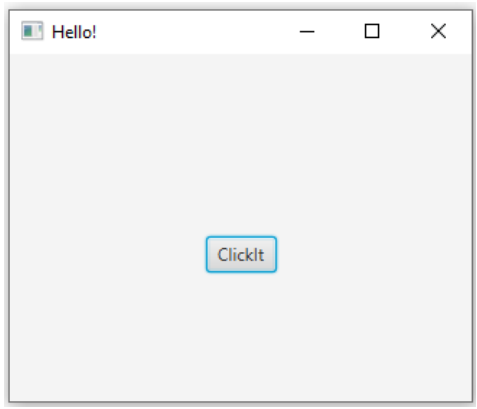
The default first run should give you a white pane with a button. On clicking the button, a message is shown as below. If you are unable to start running your JavaFX project, watch the Lab 1 installation video.



Open the SceneBuilder, and open the 'hello-view.fxml' file of this project. [Or right click the 'hello-view.fxml' file, and choose 'Open in SceneBuilder']. Edit it by renaming the button as 'ClickIt'.



Then save this layout by 'File -> Save', and come back to IntelliJ. You should find that the 'hello-view.fxml' file has been updated (Watch line 14, where now text="ClickIt"). Run the project, and make sure that this window shows up.



The preparation is all set now.

2. Key properties of a JavaFX project

EXECUTION SEQUENCE OF JAVAFX

The skeleton of a JavaFX project is as follows.

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
```

```
        /*
        Code for JavaFX application.
        (Stage, scene, scene graph)
        */
    }
    public static void main(String[] args){
        launch();
    }
}
```

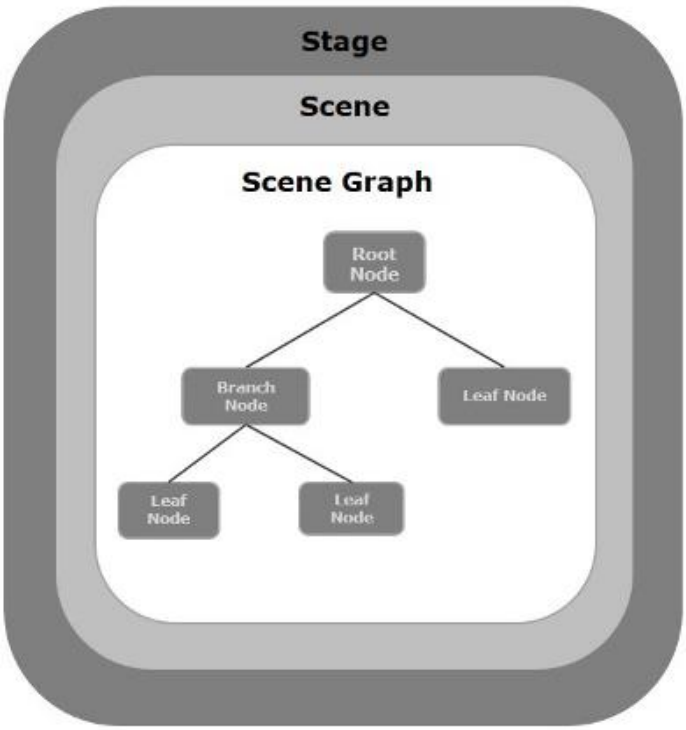
(In your Main.java (or HelloApplication.java) in IntelliJ, comment the auto-generated codes in the *start()* method, leaving it empty as above. We will insert some code into it later).

The **Application** class (e.g., in line 10 of HelloApplication.java) is the entry point of the application in JavaFX. To create a JavaFX application, you need to inherit this class and implement its abstract method *start()*. In this method, you need to write the entire code for the JavaFX graphics.

In the *main()* method, you have to launch the application using the *launch()* method. This method internally calls the *start()* method of the Application class.

THREE MAJOR COMPONENTS IN A JAVAFX APPLICATION

Stage, **Scene**, and **Nodes.**



A **stage** is essentially a window, with minimize, maximize, close buttons, and the title of the window at the top-left position. It contains all the objects of a JavaFX application. It is represented by **Stage** class of the package **javafx.stage** that is imported (e.g., in line 6 of HelloApplication.java). The primary stage is created by the platform itself. Note that it is passed as an argument to the ***start()*** method.

A **scene** contains all the contents of the JavaFX application. Specifically, it contains **nodes** organized in a scene graph. In the scene graph, a node can be a root node, a parent/branch node, or a leaf node. A leaf node may be geometrical objects (Circle, Rectangle, etc.), UI controls (buttons, test area, etc), Containers (Layout Panes), media elements, etc. A parent/root node can be a group node, which contains a collection of children nodes, which can be sub-parent nodes or leaf nodes.

Let's now insert the following code into the ***start()*** method.

```
BorderPane root = new BorderPane();
Button btn = new Button("Close App");
BorderPane.setAlignment(btn, Pos.CENTER);
root.setTop(btn);

stage.setTitle("Hello World");
stage.setScene(new Scene(root, 300, 275));
stage.show();
```

If run, a window should popup. Here, note this line which demonstrates the relationships (stage, scene, nodes) mentioned above:

```
stage.setScene(new Scene(root, 300, 275));
```

===

Task:

Download the JavaFX demonstration project, posted in Moodle by Bryan on Monday, and play with the various layouts and node types, such as TextField, Label, Button, Slider, etc. The code paths are 'Lec06-GUI-1/src/main/java/javafxapp/Main.java' and 'Lec06-GUI-1/src/main/java/javafxlayout/Main.java'.

Go to https://www.tutorialspoint.com/javafx/javafx_ui_controls.htm for a more complete list of UI controls, and try the demo code provided there.

===

EVENT HANDLING AND LISTENING

It could be confusing when you come across the words *event handler* and *event listener* in JavaFX. Here we provide rule-of-thumb guidance trying to clarify the concepts and distinguish their usages.

An *Event Listener* is essentially a **Property Change Listener** of a JavaFX class. What is a property? Observe the following lines (44 and 60-66) that can be found in 'Lec06-GUI-1/src/javafxapp/Main.java':

```
TextField textFieldInput = new TextField();
textFieldInput.textProperty().addListener(
            new ChangeListener<String>() {
    @Override
    public void changed(
            ObservableValue<? extends String> observableValue,
            String oldValue,
            String newValue) {
        labelOutput.setText(newValue);
    }
});
```

Notice the '*textProperty()*' method associated with this **TextField** class. According to the naming convention of the JavaFX, this 'textProperty()' method indicates that there is a field named '*text*' available in the TextField class. Associated with this 'text' field, there are 2+1 tool methods to access it, namely a getter '*getText()*', a setter '*setText()*', and a property getter, '*textProperty()*', which returns the StringProperty class defined in JavaFX. You can verify it yourself in IntelliJ.

The 'XXProperty' is available in a JavaFX class to describe its field named 'XX', but extends to more functionality compared to the normal getter and setter. One of the extension is: it can add listener to monitor the value change of this property. In other words, an event listener is usually used to monitor the change of the *property* value, defined in the class.

Example:

```
textFieldInput.textProperty().addListener(...);
```

In the parameter list `(...)`, you find two classes/interfaces, namely ChangeListener and ObservableValue:

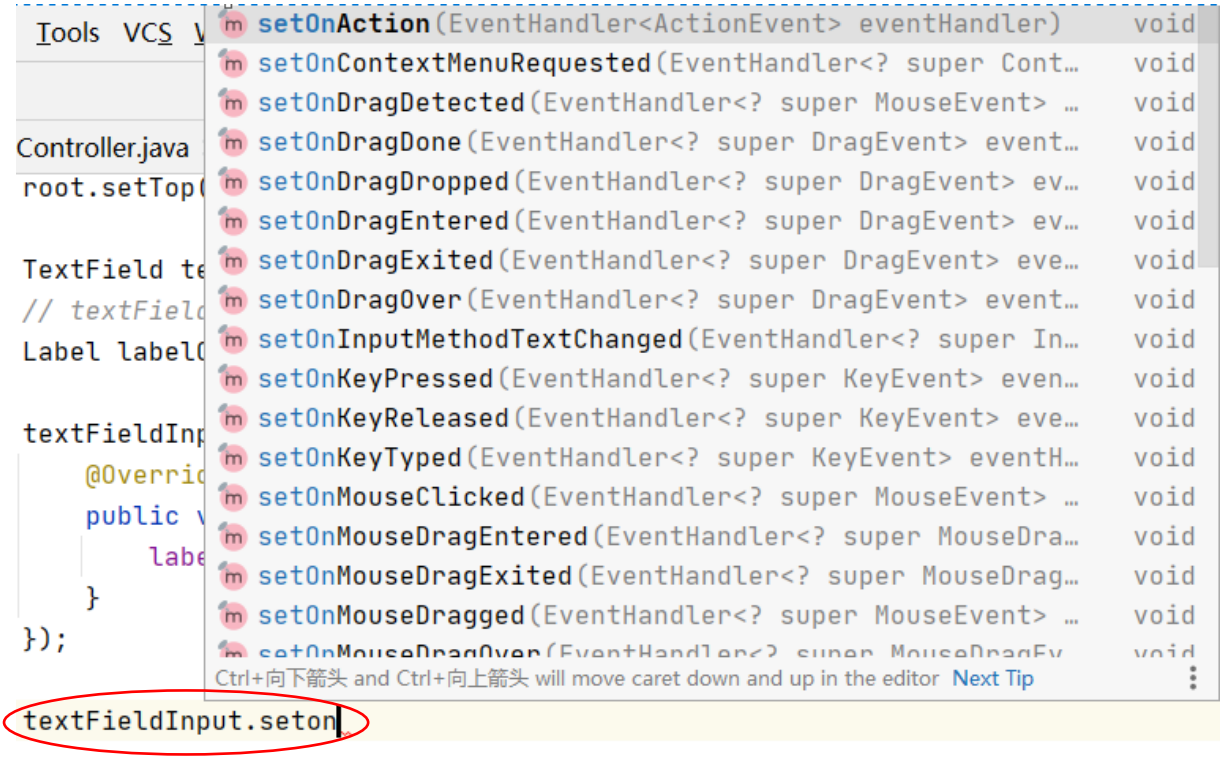
```
ChangeListener<String>
```
and
```
ObservableValue<? extends String>
```

In JavaFX, the **ObservableValue** interface fires the change notifications, and **ChangeListener** interfaces receive them. In the *addListener* method below, the ObservableValue wraps a string value and fires its changes to a newly registered ChangeListener object.

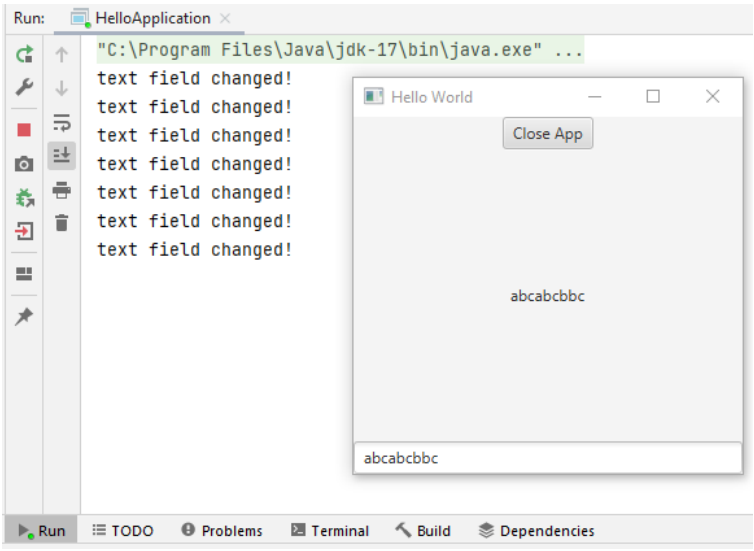
```
.addListener(new ChangeListener<String>() {
    @Override
    public void changed(
            ObservableValue<? extends String> observableValue,
            String oldValue, String newValue) {
        labelOutput.setText(newValue);
    }
})
```

Note that, the event listener follows the observer pattern. But JavaFX has hidden the pattern in its library/implementations, so that you don't have to recreate the wheel, but just follow the routine to create and use the observer/listener.

On the other hand, an *Event Handler* usually defines your customized method if a related action/event happens. JavaFX defines a wide range of events for a class that you can `@override` when defining the event handler. They are typically defined in *.setOnXXX()* methods shown below. In our HelloJFX project (HelloApplication.java), you can try to type the following red-circled statement to show the setOnXXX options.



When running the application, try typing whatever text in below input field, then pressing 'Enter' after you type something in the TextField. You should see the effect of 'setOnAction' defined in lines 52-57 in HelloApplication.java.



===

Task:

Try the sample code in 'Lec06-GUI-1/src/main/java/javafxapp/Main.java' to have a feeling of event listening and handling. You can also try the lambda expression there.

===

More details and examples available at:
https://docs.oracle.com/javase/8/javafx/properties-binding-tutorial/binding.htm#JFXBD107

https://www.tutorialspoint.com/javafx/javafx_event_handling.htm

===

Take-Home Task:

Try all the sample code that are not mentioned above in 'Lec06-GUI-1.zip'.

===