# Graphical User Interface (GUI) & Basic Listener

# Outlines

- History
- JavaFX components
- Starting using GUI
- Basic structure (stage, scene, scene graph, node)
- Layout
- Chart
- Scene builder
- FXML
- Style
- Binding properties
- Basic event handling

# History

- AWT is Java's original set of classes for building GUIs
  - Abstract Window Toolkit (AWT)
  - import java.awt.*
  - Uses peer components of the OS; heavyweight
  - Not truly portable: looks different and lays out inconsistently on different OSs
    - Due to OS's underlying display management system
- Swing is designed to solve AWT's problems
  - import javax.swing.*
  - Extends AWT
  - 99% java; lightweight components
  - Layout consistently on all OSs
  - Uses AWT event handling

# History (cont.)

- JavaFX
  - JAVA + FLASH + FLEX
  - An API included in Java SE 8 for UI development
  - The successor of Java Swing
  - 100% java; lightweight component
  - Swing Node (embed Swing in JavaFX)
  - More features
    - Data binding
    - FXML (mark-up language for designing UI)
    - CSS
    - Charts.
    - 3D Support
    - Etc.
- We will learn JavaFX in this class

# JavaFX components

- Containers
  - Anchor Pane, Stack Pane, Tab Pane, HBox, Vbox, …
- UI Controls
  - Accordion, Label, Button, RadioButton, CheckBox, TextField, TextArea, Slider, Tooltip, ComboBox, ProgressBar, DatePicker, ColorPicker, …
- Shapes
  - Line, Rectangle Ellipse, Path, Circle Arc, Polygon Polyline, Curve, Text
- Charts
  - LineChart, PieChart, AreaChart, BarChart, ScatterChart, BubbleChart

# JavaFX components (cont.)


Reference: http://docs.oracle.com/javafx/2/ui_controls/overview.htm

# Let's setup our Java FX

- 1) Download Java FX
- 2) One-time setup for Java FX
- 3) Permanent setup for Java FX
- 4) Scence Builder (optional)

# 1) Download JavaFX

– https://gluonhq.com/products/javafx/

GLUON Products Developers Pricing Services Insights Contact

- The OpenJFX page at openjfx.io is a great starting place to learn more about JavaFX.
- This software is licensed under GPL v2 + Classpath

## Downloads

Filter what you need

| JavaFX version | Operating System | Architecture | Type |
|---|---|---|---|
| 19 | Windows | x64 | SDK |

☐ Include older versions

| OS | Version | Architecture | Type | Download |
|---|---|---|---|---|
| Windows | 19 | x64 | SDK | Download [SHA256] |

Unzip it to the folder of your choice!!

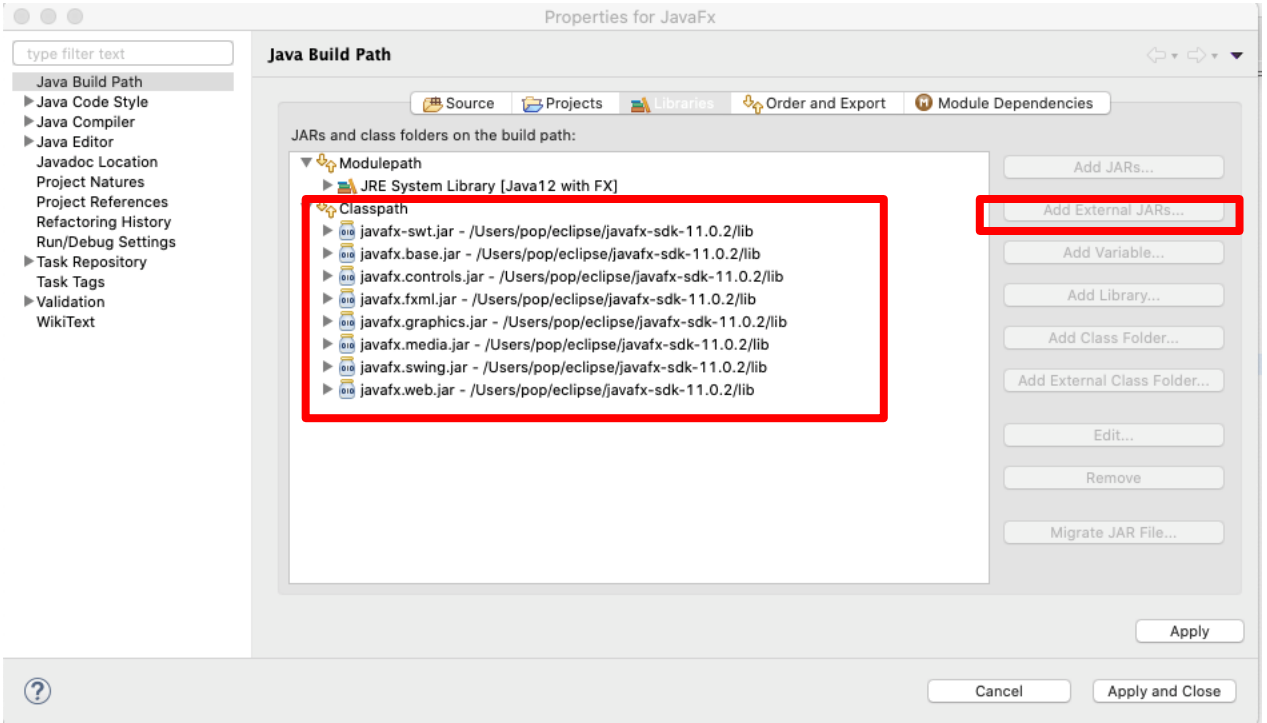
# 2) One-time setup for Java FX



Java™

# How to setup JavaFX project (cont.)

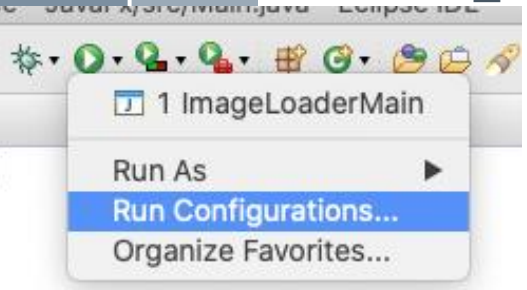
- Right-Click your project > Build Path… > Configure Build Path
- In the Libraries Tab, under Classpath, click Add External JAR…
- Navigate to the previously extracted JavaFX folder, go to the folder lib, and select every  jar file in there and click Open.

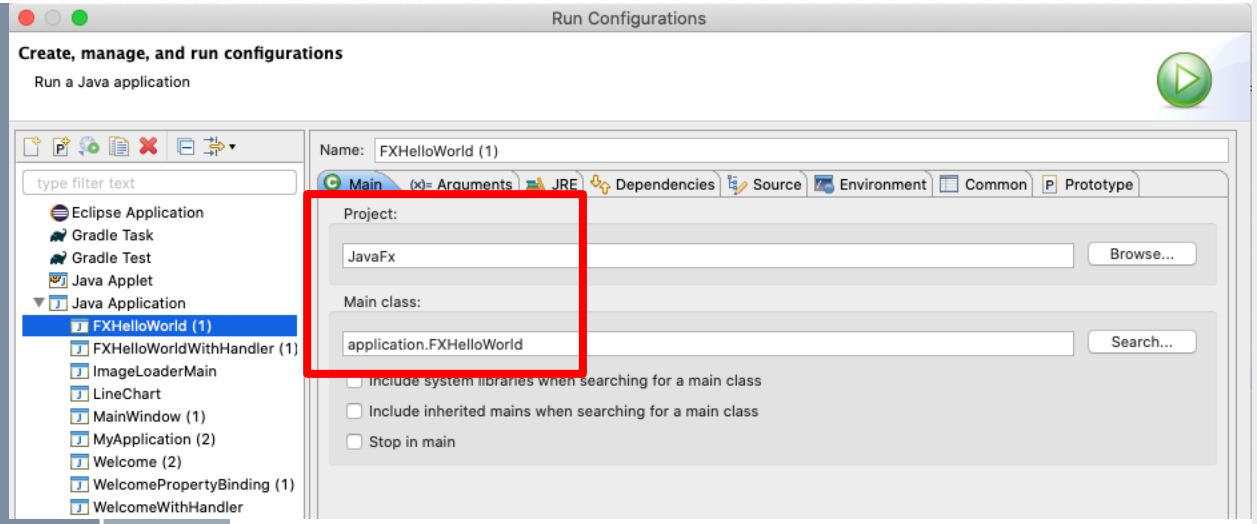


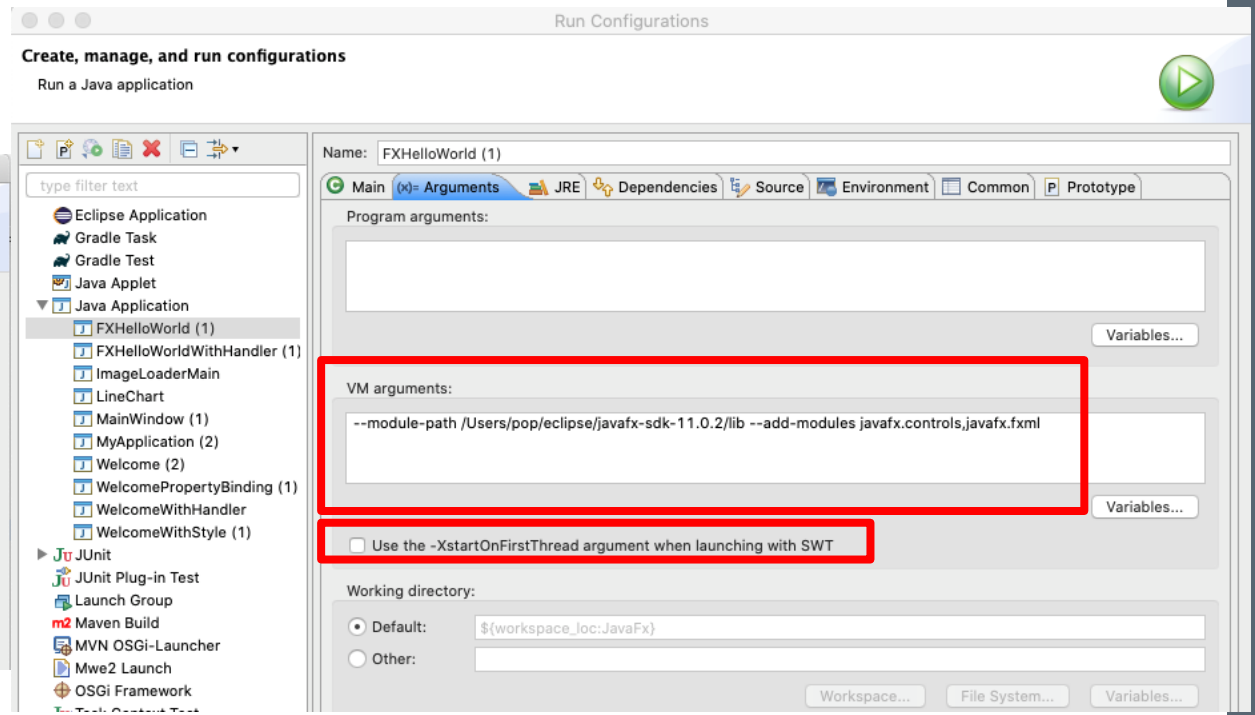
# How to setup JavaFX project (cont.)

- Modify Run Configurations
  - In the main tab, make sure the project & main class
  - In the arguments tab, add the following VM arguments & uncheck XstartOnFirstThread

--module-path <Your-JavaFX-Full-Path>/lib --add-modules javafx.controls,javafx.fxml



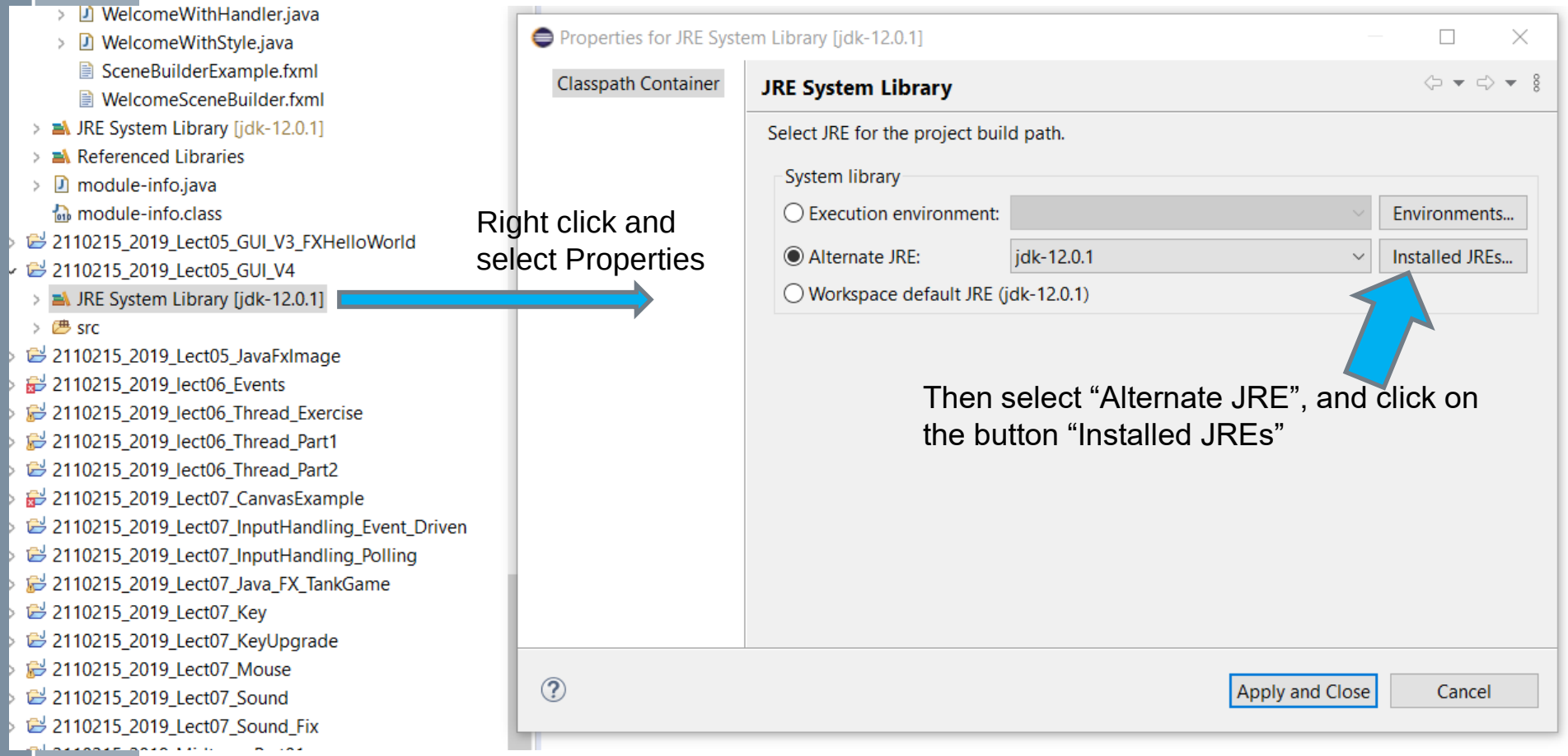



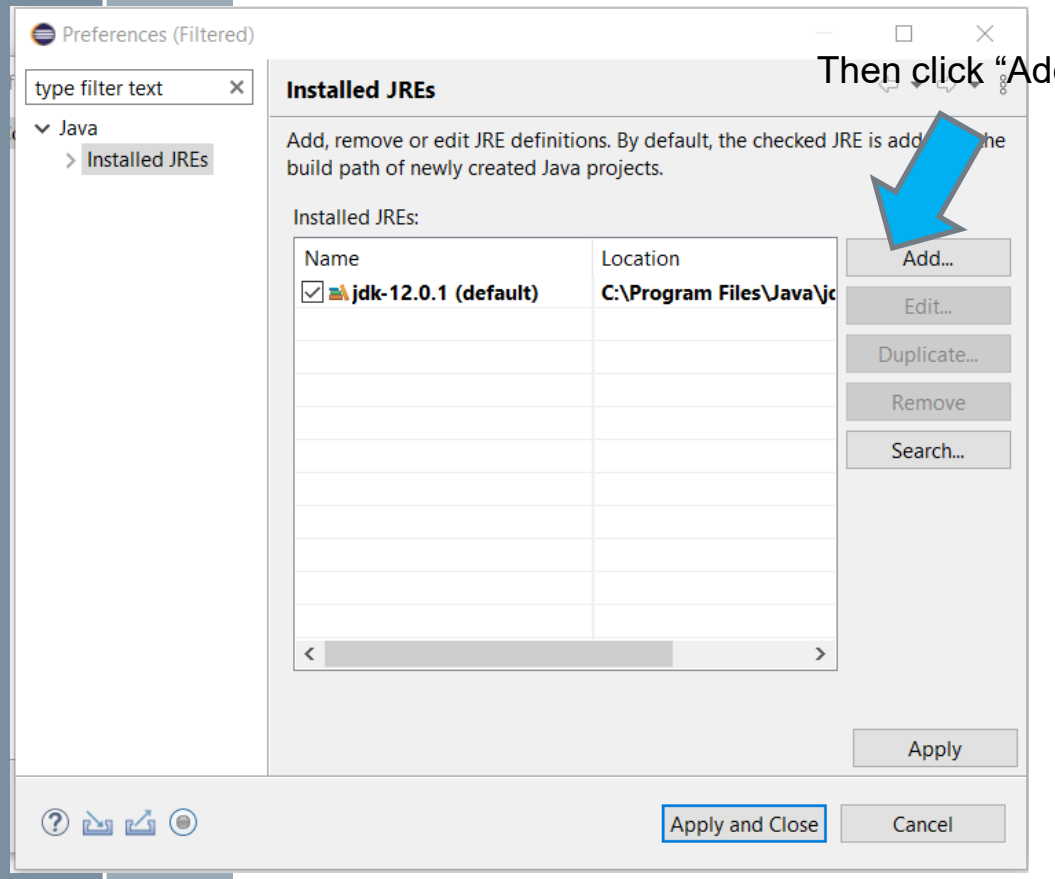
# 3) Permanent setup for Java FX



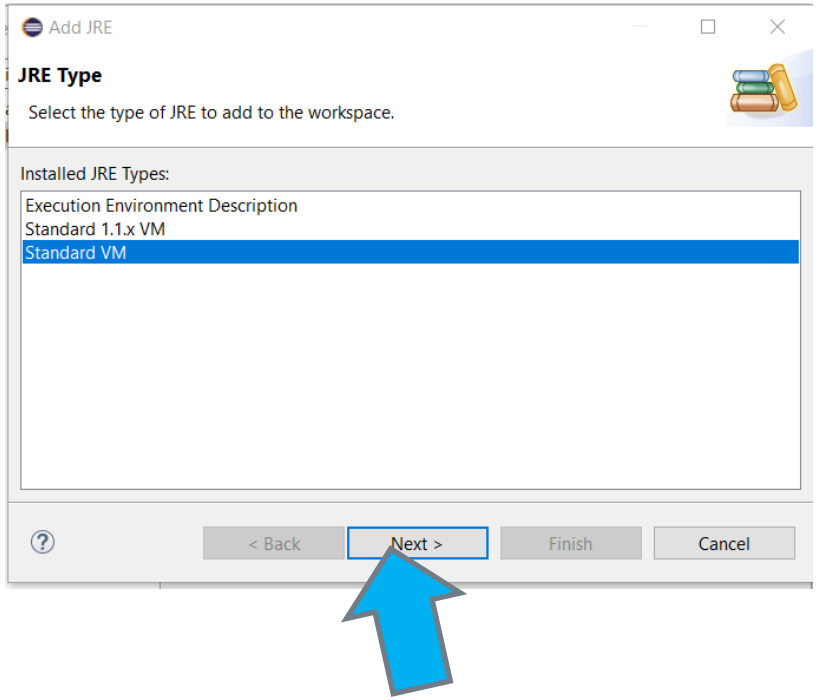
Java

- Install javaFX and note its lib folder location.
- Then in your project
  - Set JRE to include JavaFx by the following steps.



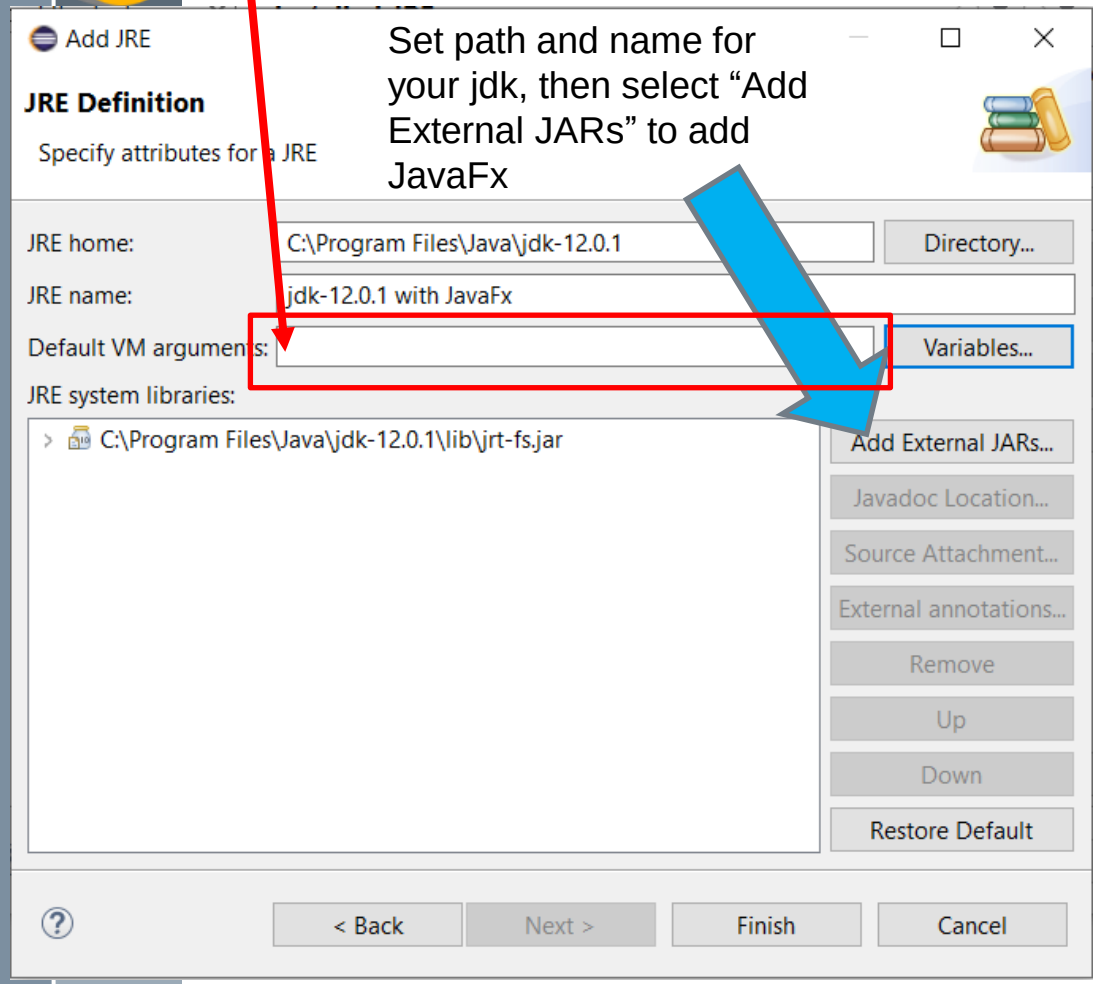


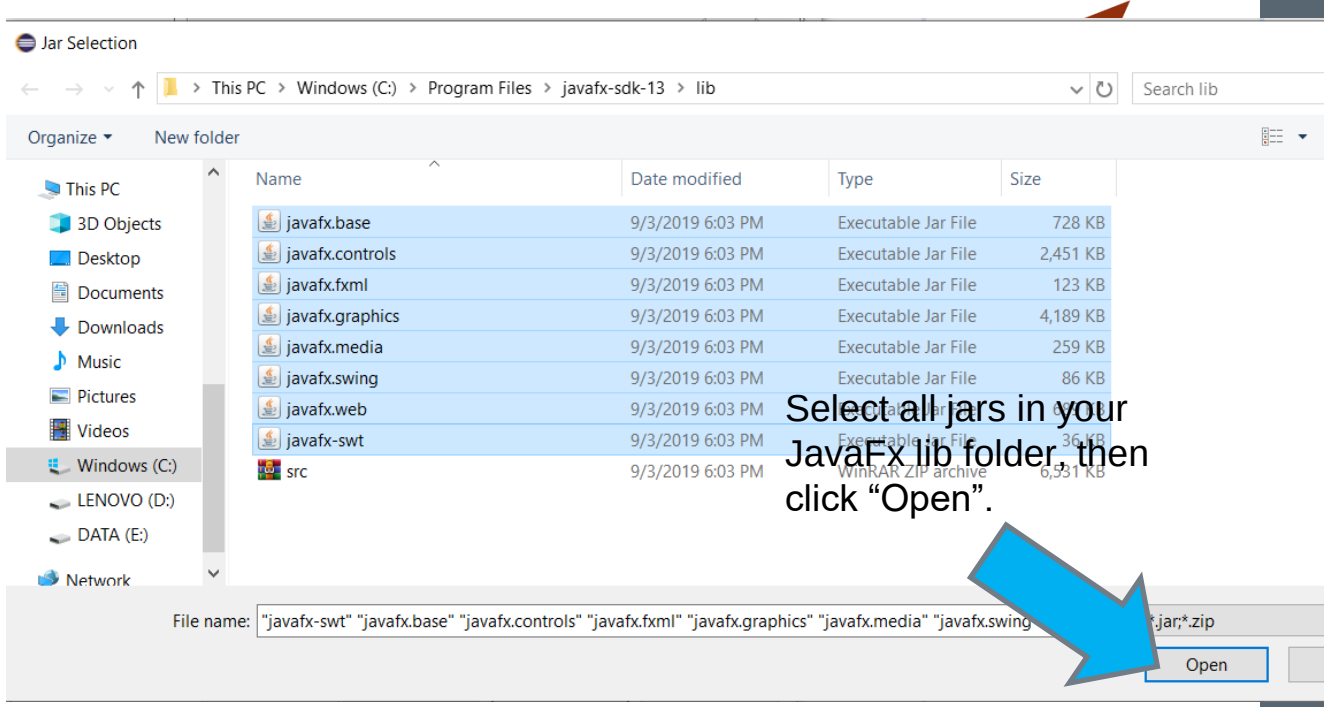

Then click “Add”

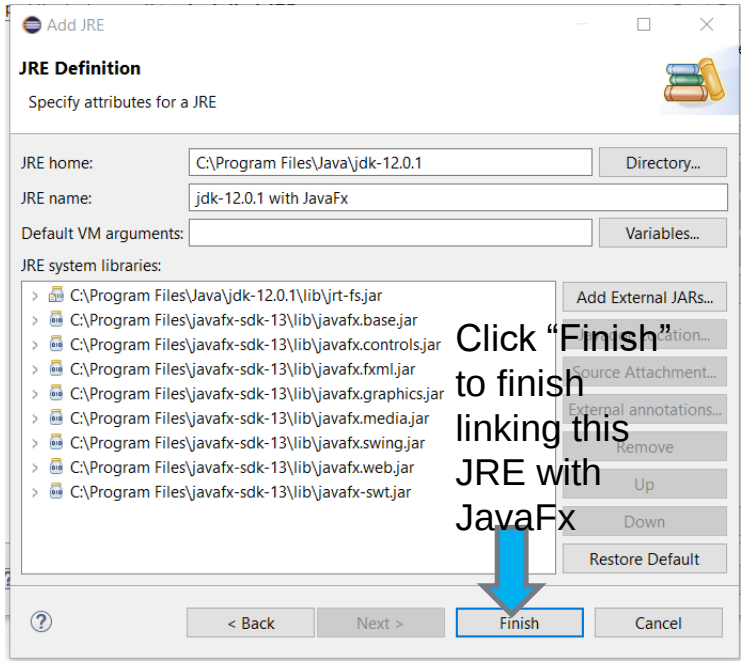


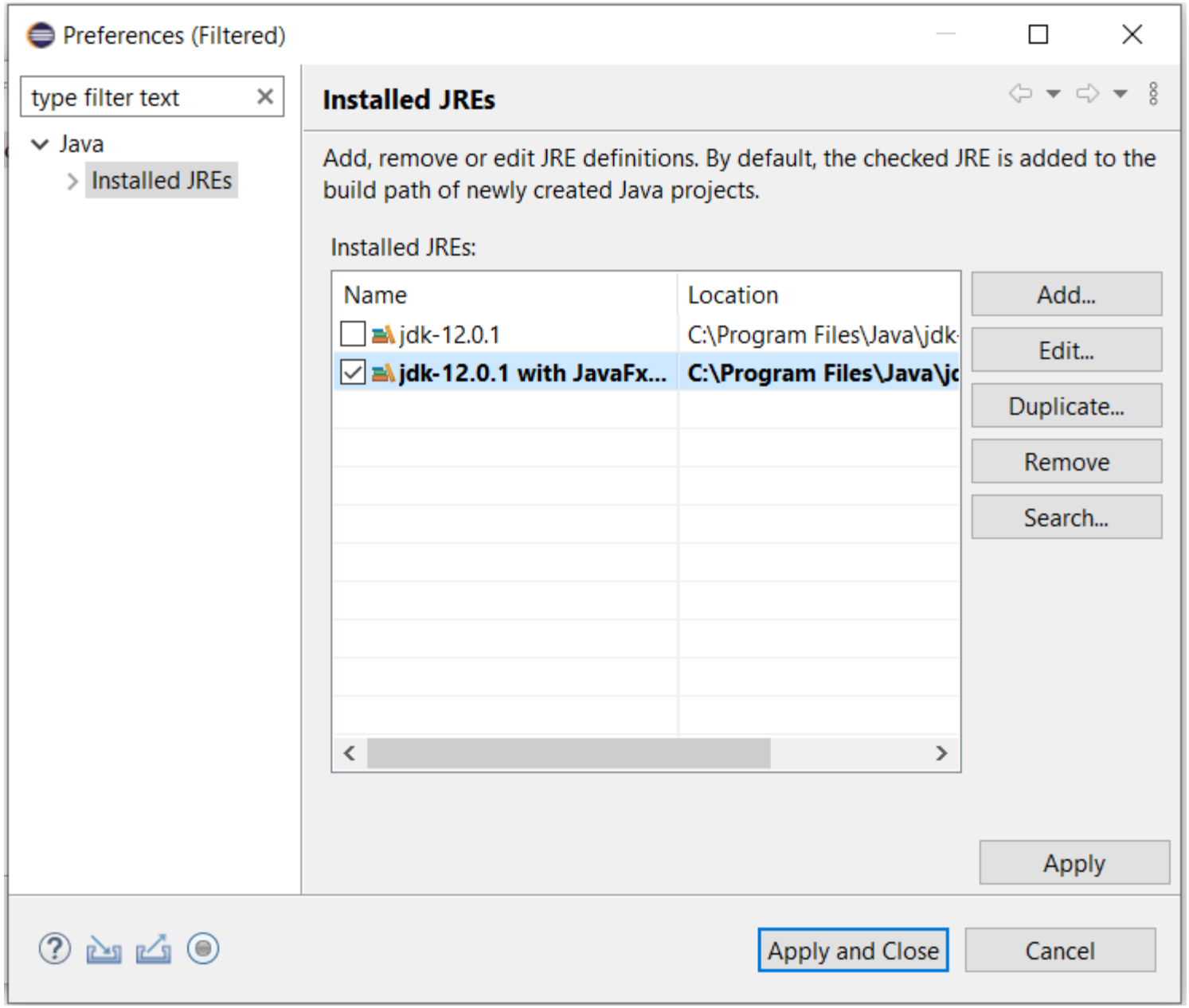

Make sure “Standard VM” is selected,
Then click “Next”

--module-path /Users/pop/eclipse/javafx-sdk-11.0.2/lib --add-modules javafx.controls,javafx.fxml






You can now select this new JRE for every project that uses JavaFx

# Now you can just

- Create a Java project.
- Create a Java class to run as your JavaFx application.

# Caution!

Exception : The type 'Button' is not API or javaFX.application not registered.

- Go into the project's build path and edited the JRE System Library, some execution environment was selected.
- Choose to use an “Alernate JRE” and make sure you select the correct JRE from here, then it will fix this error for you.

# 4) Scence Builder (optional)

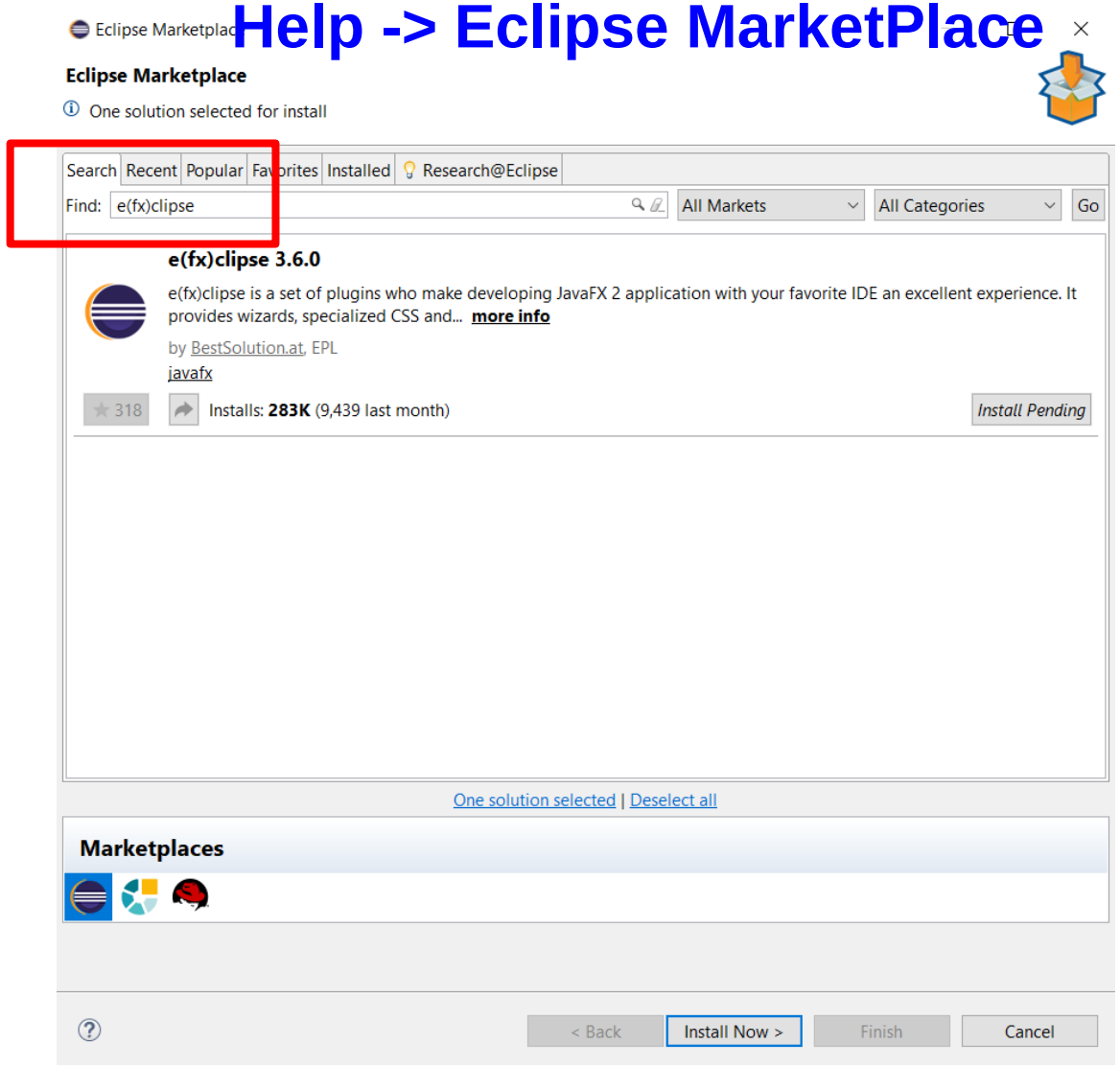


Java

# SceneBuilder

## Next step of the setup, install E(fx)clipse into Eclipse

## Help -> Eclipse MarketPlace



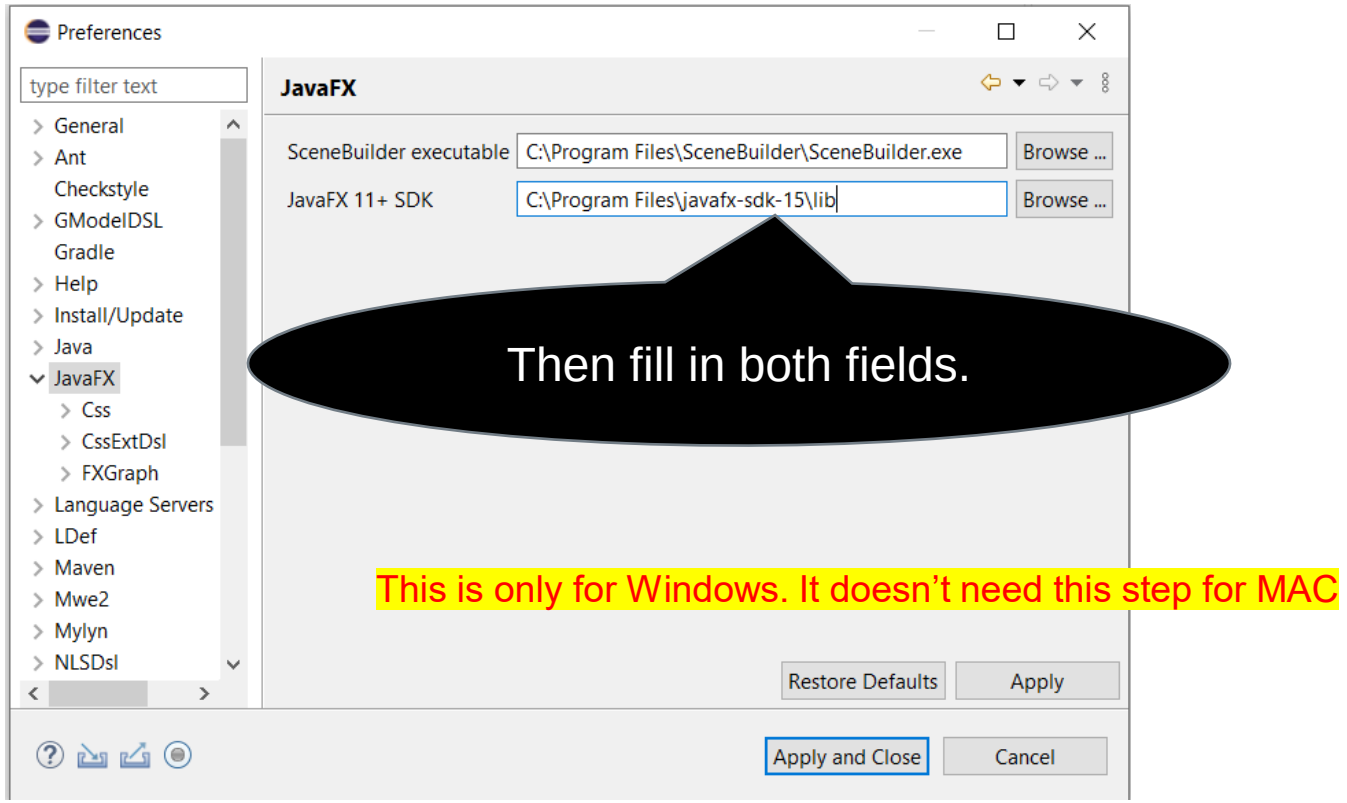

This plugin helps you

- open FXML editor with syntax highlighting.
- Link JavaFX, Eclipse, and Scene builder

# Scene builder (cont.)

- How to install JavaFX Scene Builder
  - Download and install JavaFX Scene Builder https://gluonhq.com/products/scene-builder/
  - Configuring Eclipse to use the Scene Builder “Window > Preferences”

Select this



Yeah! We now finish setup everything.

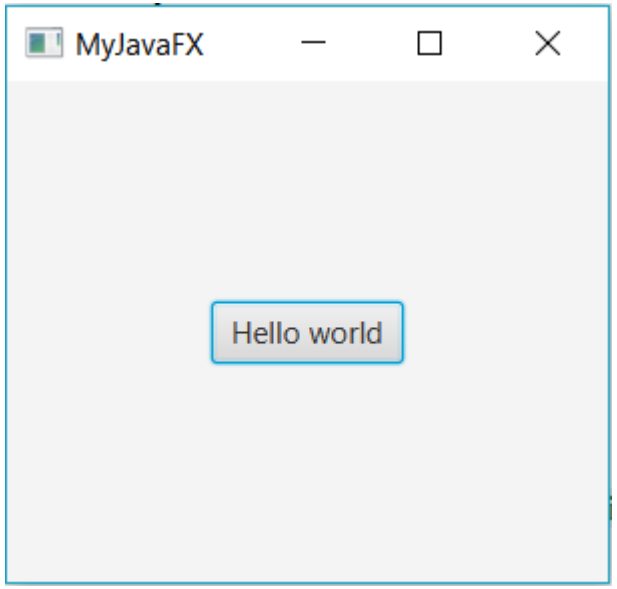
So, let's start our Java FX Project ☺

# JavaFX HelloWorld Example

## FXHelloWorld.java

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

public class FXHelloWorld extends Application {
	// Override the start method in the Application class
	@Override
	public void start(Stage primaryStage) {
		// Create a scene and place a button in the scene
		Button btn = new Button("Hello world");
		StackPane root = new StackPane();
		root.getChildren().add(btn);
		Scene scene = new Scene(root, 300, 250);
		primaryStage.setTitle("MyJavaFX"); // Set the stage title
		primaryStage.setScene(scene); // Place the scene
		primaryStage.show();
	}

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

**May not compile at all!**



# JavaFX HelloWorld Example (cont.)

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

public class FXHelloWorld extends Application {
	// Override the start method in the Application class
	@Override
	public void start(Stage primaryStage) {
		// Create a scene and place a button in the scene
		Button btn = new Button("Hello world");
		StackPane root = new StackPane();
		root.getChildren().add(btn);
		Scene scene = new Scene(root, 300, 250);
		primaryStage.setTitle("MyJavaFX"); // Set the stage title
		primaryStage.setScene(scene); // Place the scene
		primaryStage.show();
	}

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

To create JavaFX application,

- Extends Application (javafx.application.Application)

# JavaFX HelloWorld example (cont.)

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

public class FXHelloWorld extends Application {
	// Override the start method in the Application class
	@Override
	public void start(Stage primaryStage) {
		// Create a scene and place a button in the scene
		Button btn = new Button("Hello world");
		StackPane root = new StackPane();
		root.getChildren().add(btn);
		Scene scene = new Scene(root, 300, 250);
		primaryStage.setTitle("MyJavaFX"); // Set the stage title
		primaryStage.setScene(scene); // Place the scene
		primaryStage.show();
	}

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

To create JavaFX application,

- Extends Application (javafx.application.Application)
- Override the start() method

# JavaFX HelloWorld example (cont.)

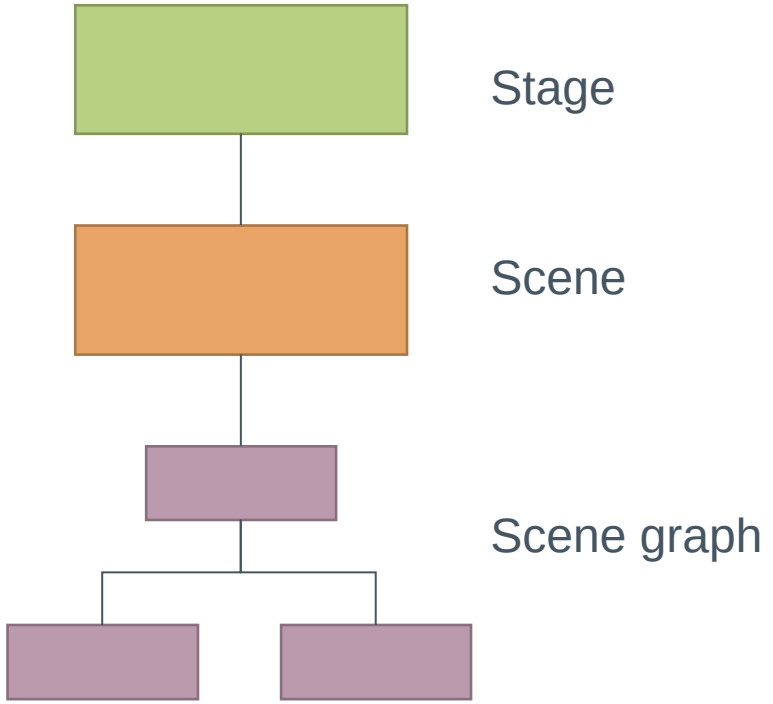
```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

public class FXHelloWorld extends Application {
      // Override the start method in the Application class
      @Override
      public void start(Stage primaryStage) {
            // Create a scene and place a button in the scene
            Button btn = new Button("Hello world");
            StackPane root = new StackPane();
            root.getChildren().add(btn);
            Scene scene = new Scene(root, 300, 250);
            primaryStage.setTitle("MyJavaFX"); // Set the stage title
            primaryStage.setScene(scene); // Place the scene
            primaryStage.show();
      }

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

To create JavaFX application,

- Extends Application (javafx.application.Application)
- Override the start() method
- Call launch() (Application.launch())
  - The framework internals call the start() method to start
  - Then, javafx.stage.Stage object is available to use

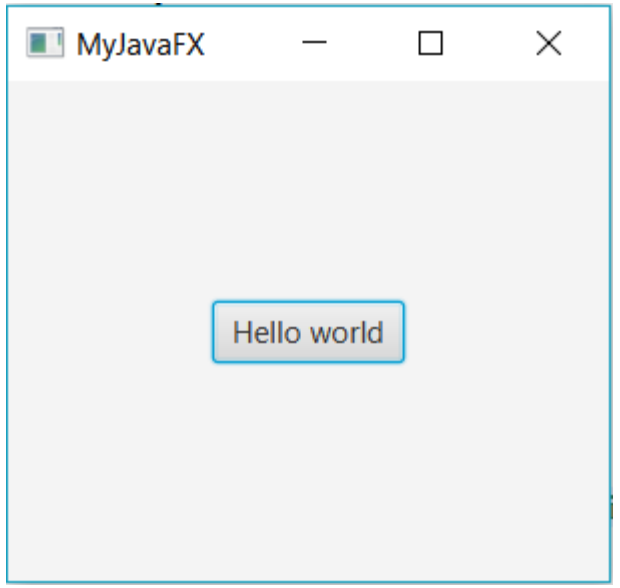
# Basic structure

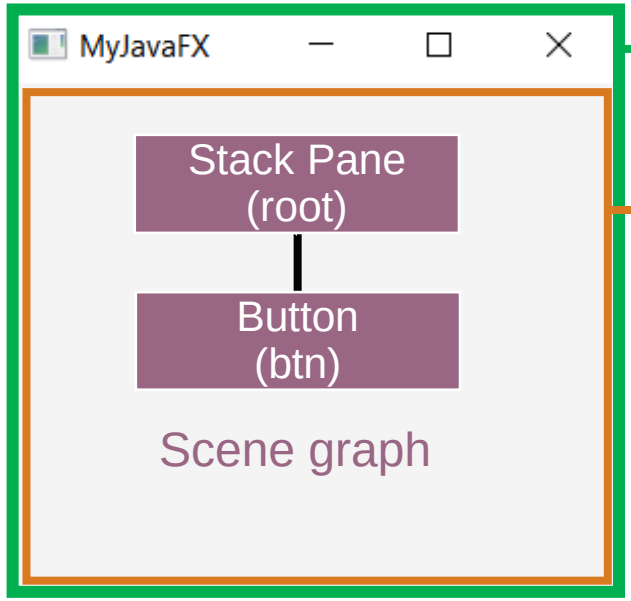
- JavaFX application contains one or more stages which corresponds to windows
- Each stage has a scene
- Each scene can have scene graph (hierarchical tree of nodes)
- Node (UI Components such as control, layout)



# JavaFX HelloWorld Example

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

public class FXHelloWorld extends Application {
      // Override the start method in the Application class
      @Override
      public void start(Stage primaryStage) {
            // Create a scene and place a button in the scene
            Button btn = new Button("Hello world");
            StackPane root = new StackPane();
            root.getChildren().add(btn);
            Scene scene = new Scene(root, 300, 250);
            primaryStage.setTitle("MyJavaFX"); // Set the stage title
            primaryStage.setScene(scene); // Place the scene
            primaryStage.show();
      }

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



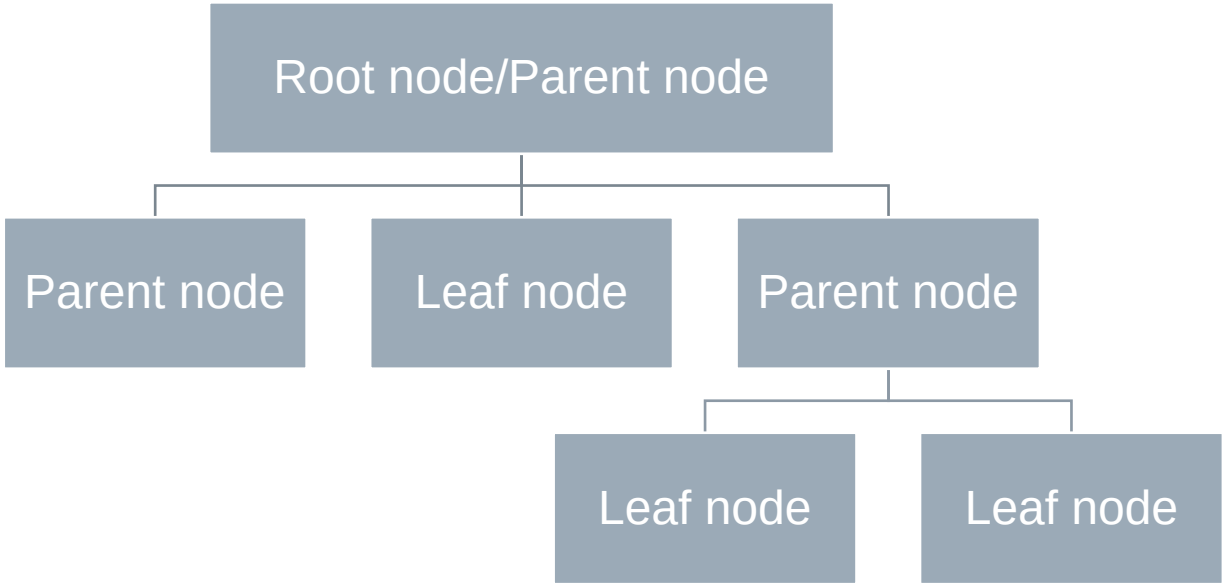




Stage
javafx.stage (window)

Scene
javafx.scene

# Scene graphs

- In JavaFX, contents (such as text, images, and UI controls) are organized using a tree-like data structure known as scene graph
- A scene graph is hierarchical tree of nodes



# Nodes

- GUI component object, such as geometric shapes, UI controls, layout panes, and 3D objects.
- 3 types of nodes
  - Root Node
    - Parent of all other nodes
    - Scene graph can have only one root node.
  - Parent Node (group of nodes)
    - Can have other nodes as children
  - Leaf Node
    - Cannot have children
    - Not container

# Nodes (cont.)

- Node can have the following:
  - ID
  - Style
  - Class
  - Bounding volume
  - Effects such as blurs and shadows
  - Event handlers (such as mouse, keyboard)
- Add nodes to parent

```
myParent.getChildren().add(childNode);
```

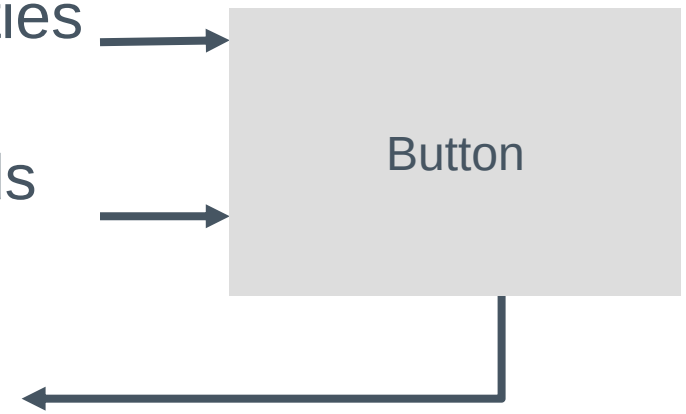
or

```
myParent.getChildren().addAll(childNode1, childNode2);
```

# Using GUI Component

- Java: GUI component = class

- Properties
- Methods
- Events




**Using a GUI component**

- 1. Create it

```
Button btn = new Button("Hello world");
```

- 2. Configure it

```
// using getter/setter to access properties (text)
btn.setText("Hello world"); // methods
```

- 3. Add it to parent

```
root.getChildren().add(btn);
```

- 4. Listen to it

Events: Listeners

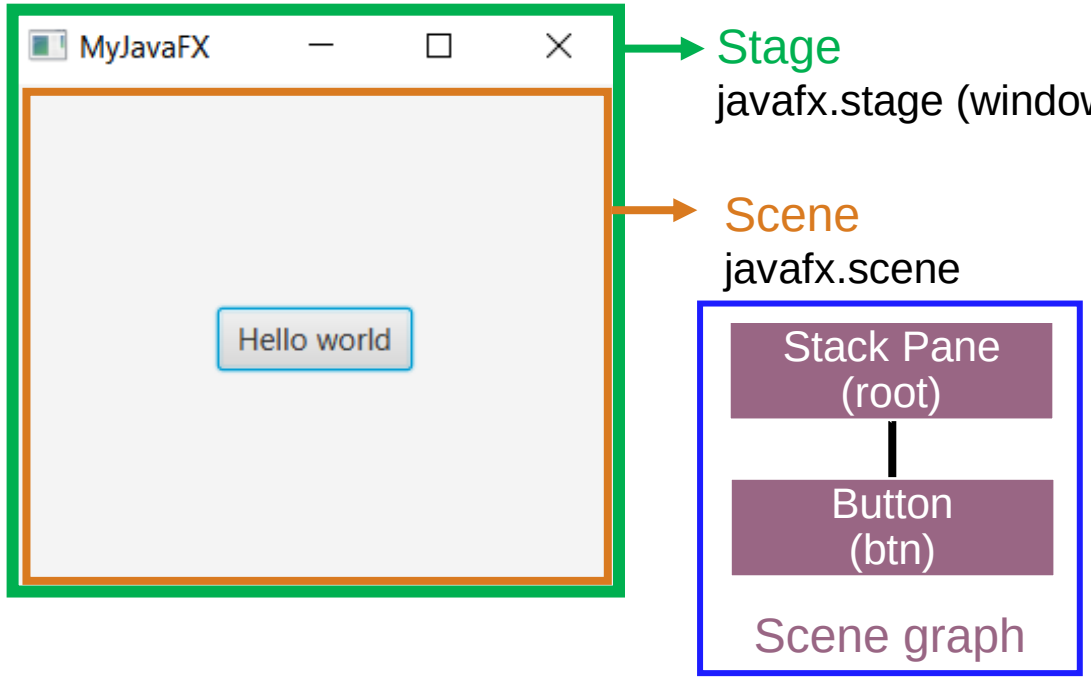
# Using a GUI Component

1. Create it
2. Configure it
3. Add children  (if root or parent node (container))
4. Add to parent  (if not root node)
5. Listen to it

order important

# JavaFX HelloWorld Example

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

public class FXHelloWorld extends Application {
	// Override the start method in the Application class
	@Override
	public void start(Stage primaryStage) {
		// Create a scene and place a button in the scene
		Button btn = new Button("Hello world");
		StackPane root = new StackPane();
		root.getChildren().add(btn);
		Scene scene = new Scene(root, 300, 250);
		primaryStage.setTitle("MyJavaFX"); // Set the stage title
		primaryStage.setScene(scene); // Place the scene
		primaryStage.show();
	}

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



# Scene

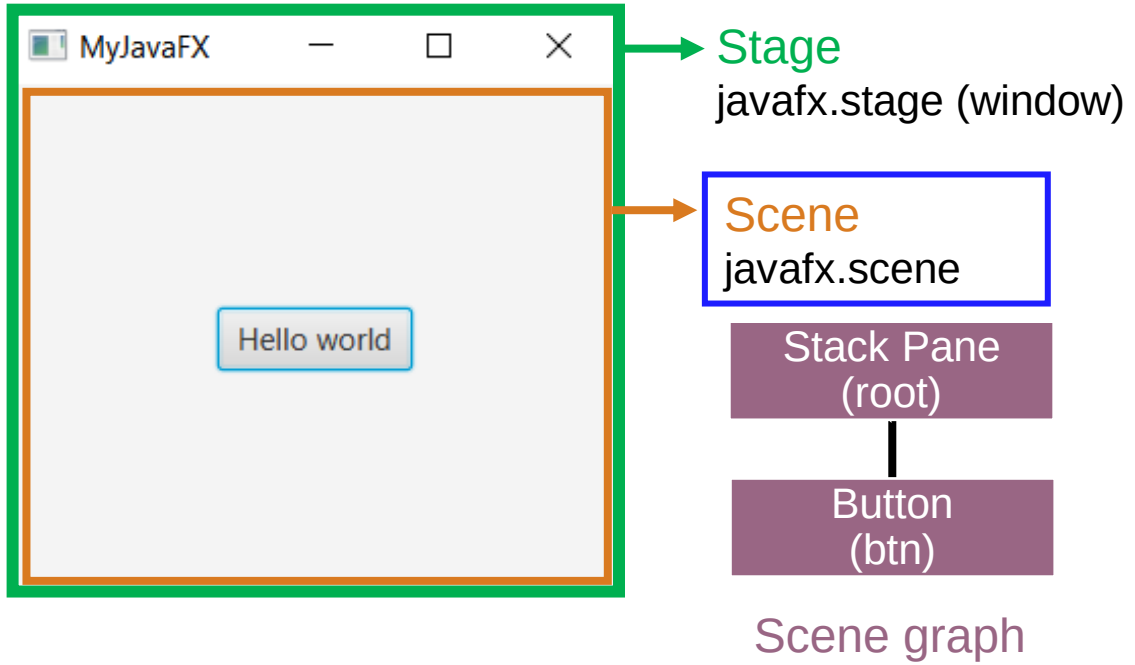
- Container for all contents in a scene graph
- Root node of the scene graph is required for creating Scene

```
Scene scene = new Scene(root, 300, 250);
```

- Be able to set size, color etc.
- If size is not specified, automatically compute based on its contents

# JavaFX HelloWorld Example

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

public class FXHelloWorld extends Application {
	// Override the start method in the Application class
	@Override
	public void start(Stage primaryStage) {
		// Create a scene and place a button in the scene
		Button btn = new Button("Hello world");
		StackPane root = new StackPane();
		root.getChildren().add(btn);
		Scene scene = new Scene(root, 300, 250);
		primaryStage.setTitle("MyJavaFX"); // Set the stage title
		primaryStage.setScene(scene); // Place the scene
		primaryStage.show();
	}

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



# Stage

- javafx.stage package
- Top level container of the application.
- Usually, OS Window.
- The main stage is created as part of the application launch and passed as an argument in start method

```
public void start(Stage primaryStage)
```

- Be able to set title, size, icon etc.
- Single application can have multiple stages

# Stage (cont.)

- Set Stage title

```
primaryStage.setTitle("MyJavaFX");
```

- Set scene to stage

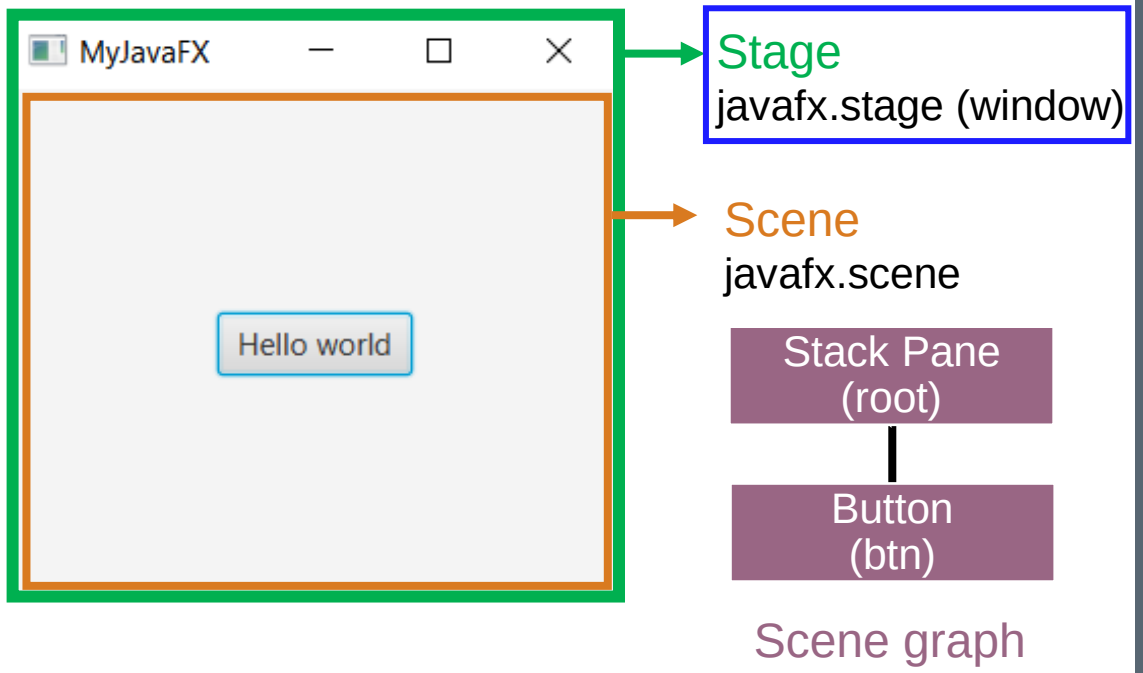
```
primaryStage.setScene(scene);
```

- Show the stage

```
primaryStage.show();
```

# JavaFX HelloWorld Example

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;

public class FXHelloWorld extends Application {
	// Override the start method in the Application class
	@Override
	public void start(Stage primaryStage) {
		// Create a scene and place a button in the scene
		Button btn = new Button("Hello world");
		StackPane root = new StackPane();
		root.getChildren().add(btn);
		Scene scene = new Scene(root, 300, 250);
		primaryStage.setTitle("MyJavaFX"); // Set the stage title
		primaryStage.setScene(scene); // Place the scene
		primaryStage.show();
	}

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

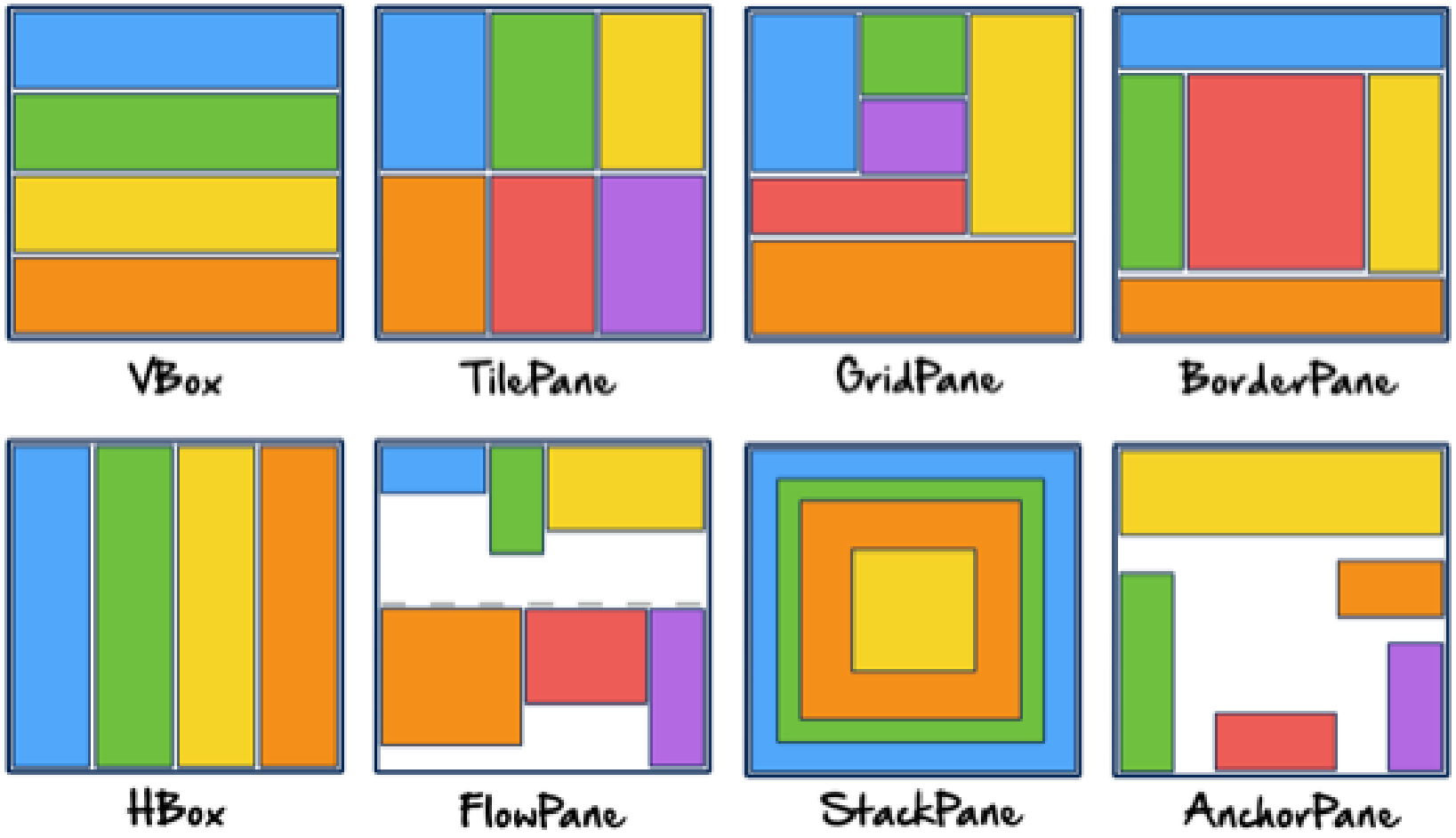


# Layout Pane

› JavaFX provides many types of panes for organizing nodes in a container.

| *Class* | *Description* |
|---|---|
| `Pane` | Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane. |
| `StackPane` | Places the nodes on top of each other in the center of the pane. |
| `FlowPane` | Places the nodes row-by-row horizontally or column-by-column vertically. |
| `GridPane` | Places the nodes in the cells in a two-dimensional grid. |
| `BorderPane` | Places the nodes in the top, right, bottom, left, and center regions. |
| `HBox` | Places the nodes in a single row. |
| `VBox` | Places the nodes in a single column. |

# Layout Pane (cont.)

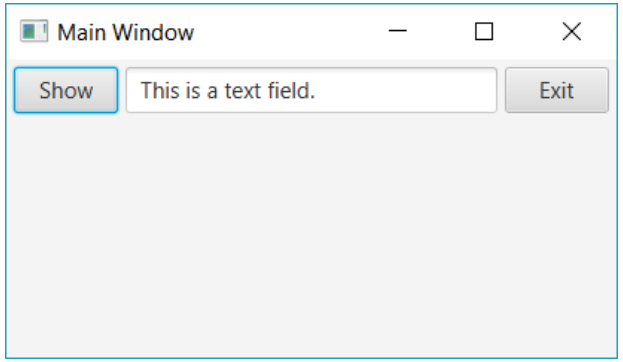


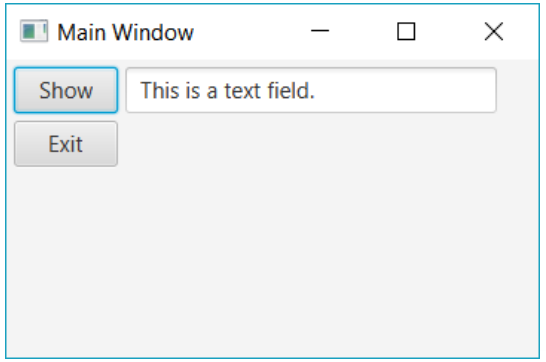

Reference: https://dzone.com/refcardz/javafx-8-1

# Examples

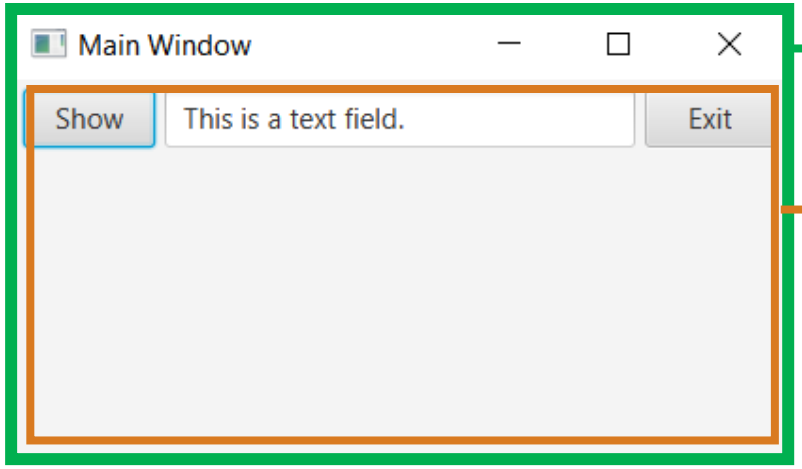
## MainWindow.java

```
package application;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;

public class MainWindow extends Application {
    @Override
    public void start(Stage primaryStage) {
    // create the flow pane as root node
    FlowPane root = new FlowPane();
    root.setPadding(new Insets(5));
    root.setHgap(5);
    root.setVgap(5);

        Button exitButton = new Button(" Exit ");
        exitButton.setPrefWidth(70);
        Button showButton = new Button(" Show ");
        showButton.setPrefWidth(70);

        TextField text = new TextField("This is a
                                text field.");
        text.setPrefWidth(250);

root.getChildren().addAll(showButton,text,exitBu
tton);

        Scene scene = new Scene(root, 410, 200);

        primaryStage.setTitle("Main Window");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



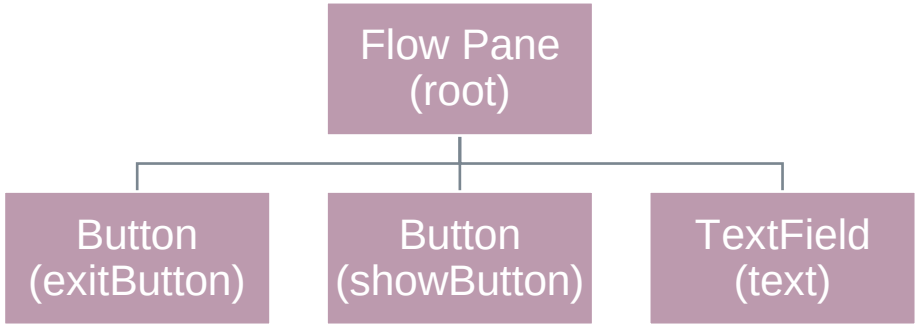



# Examples (cont.)




**Stage**
javafx.stage (window)

**Scene**
javafx.scene

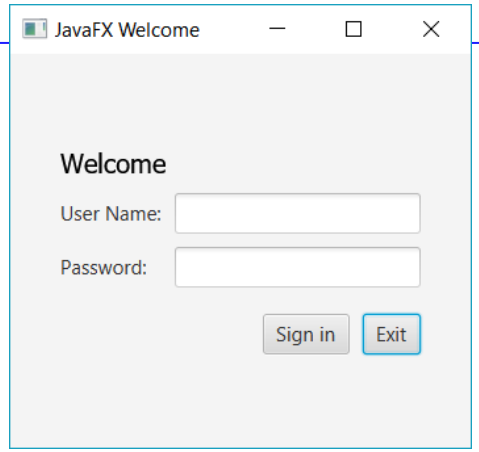



Scene graph

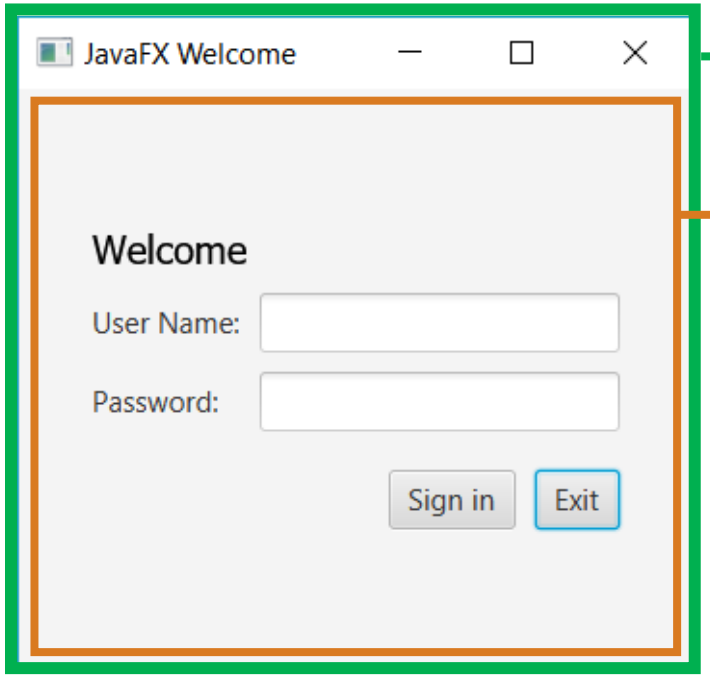
# Examples (cont.)

## Welcome.java

```
public class Welcome extends Application {

@Override
public void start(Stage primaryStage) {

      GridPane grid = new GridPane();
      grid.setAlignment(Pos.CENTER);
      grid.setHgap(10);
      grid.setVgap(10);
      grid.setPadding(new Insets(25, 25, 25, 25));

      Text scenetitle = new Text("Welcome");
      scenetitle.setFont(Font.font("Tahoma",
      FontWeight.NORMAL, 20));
      grid.add(scenetitle, 0, 0, 2, 1);

      Label userName = new Label("User Name:");
      grid.add(userName, 0, 1);

      TextField userTextField = new TextField();
      grid.add(userTextField, 1, 1);

      Label pw = new Label("Password:");
      grid.add(pw, 0, 2);

      PasswordField pwBox = new PasswordField();
      grid.add(pwBox, 1, 2);

      HBox hbBtn = new HBox(10);
      hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
      Button signinBtn = new Button("Sign in");
      Button exitBtn = new Button("Exit");
      hbBtn.getChildren().addAll(signinBtn,exitBtn);
      grid.add(hbBtn, 1, 4);

      Scene scene = new Scene(grid, 350, 300);

      primaryStage.setScene(scene);
      primaryStage.setTitle("JavaFX Welcome");
      primaryStage.show();
      }

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

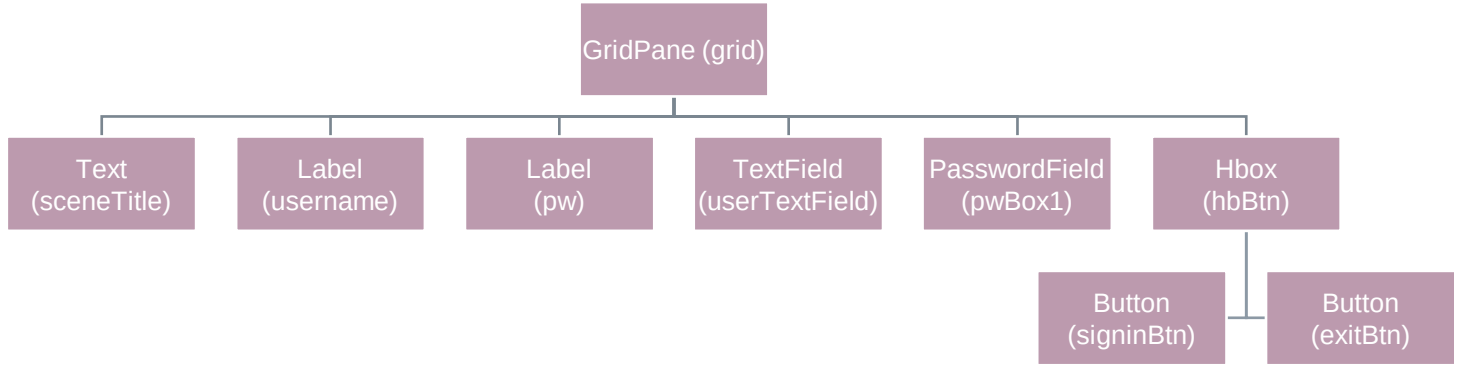


# Examples (cont.)



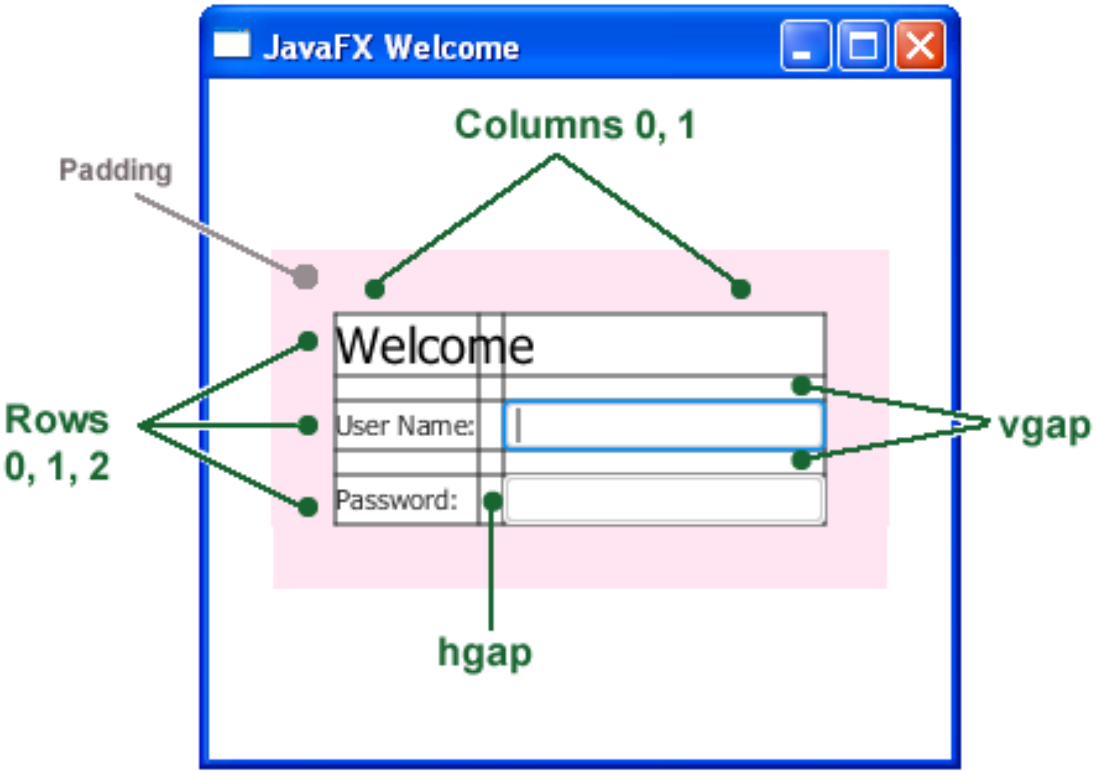

**Stage**
javafx.stage (window)

**Scene**
javafx.scene




Scene graph

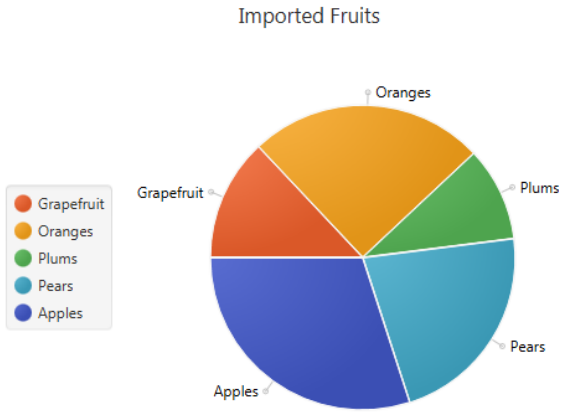
# Examples (cont.)

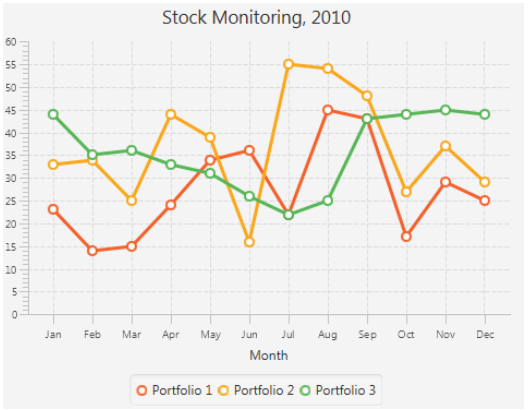


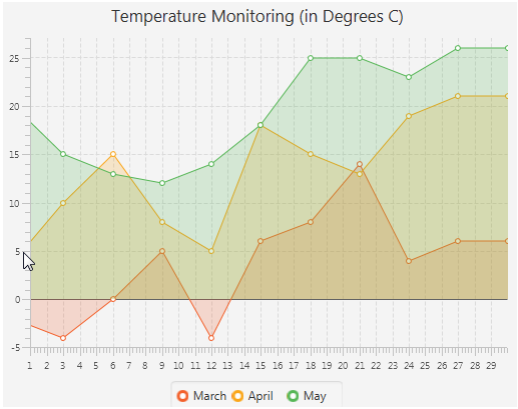

Reference: http://docs.oracle.com/javafx/2/get_started/form.htm

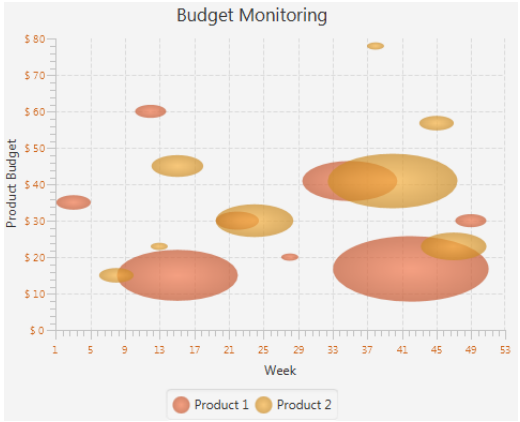
# Charts

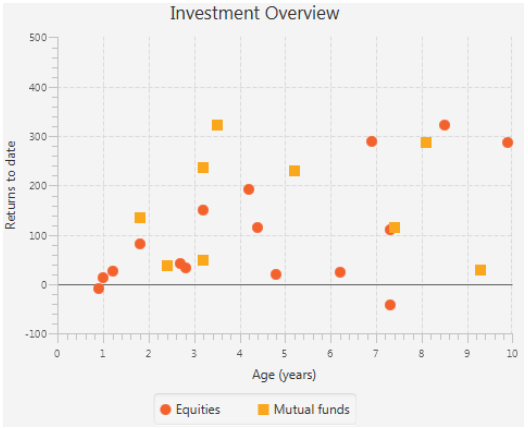
› javafx.scene.chart package

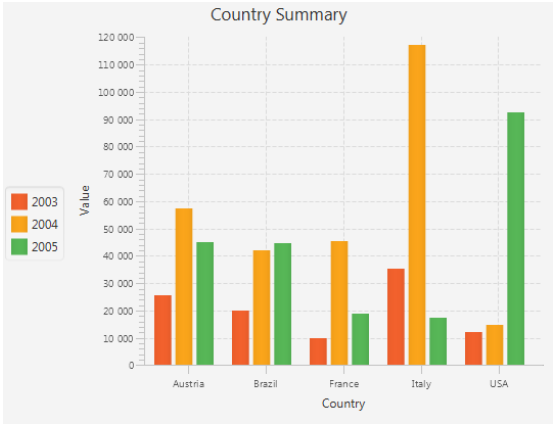


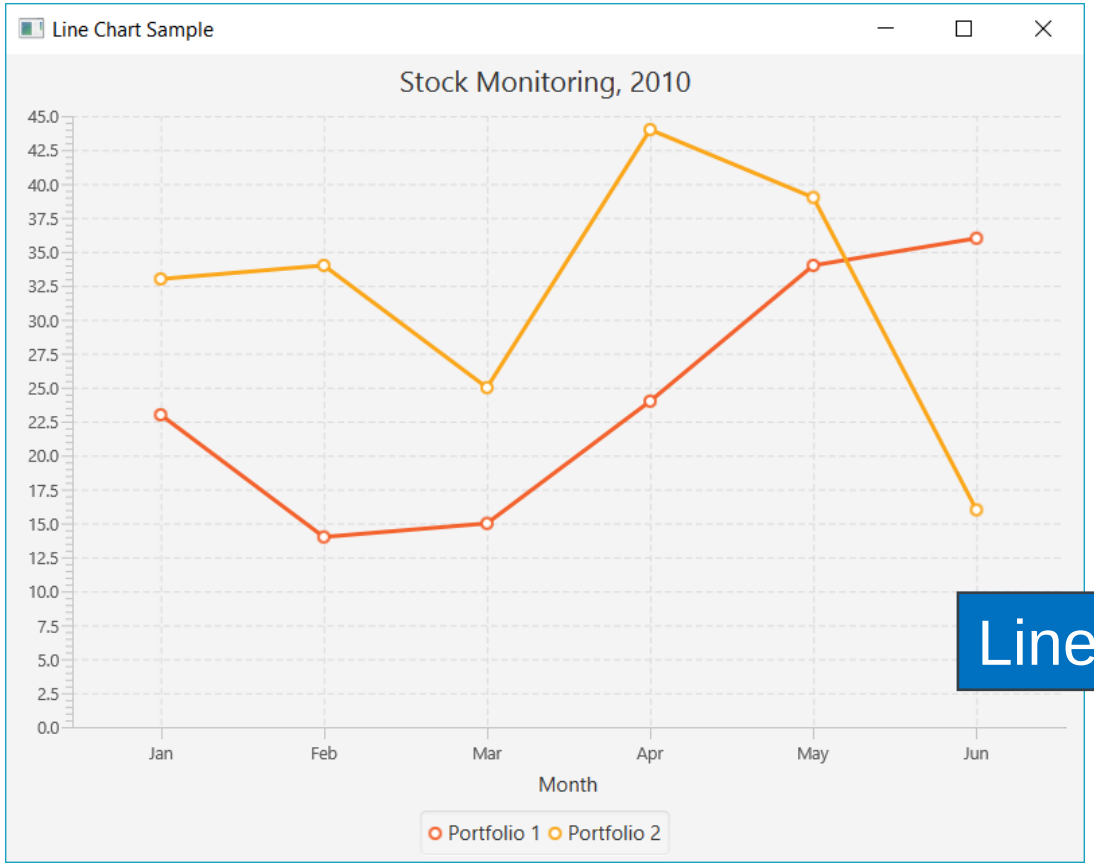



# Charts (cont.)



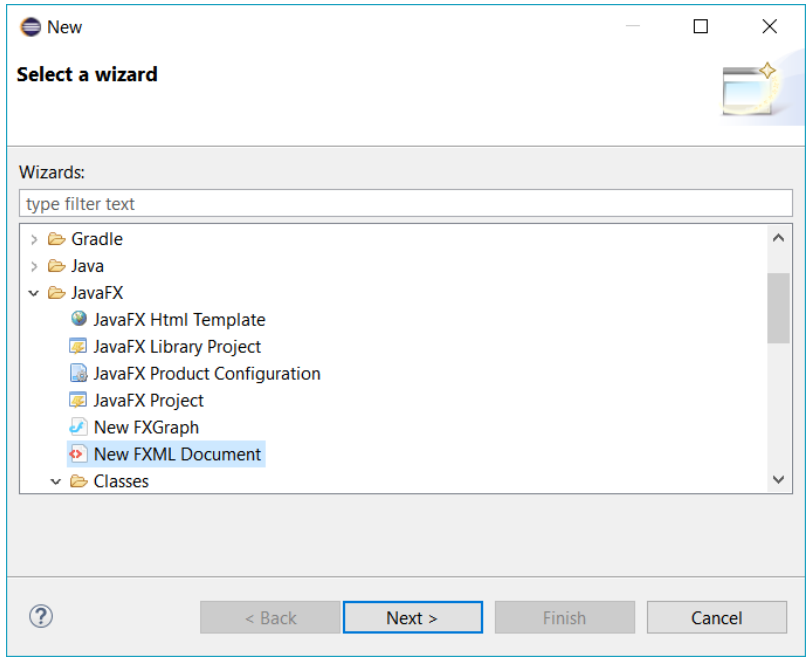

LineChartSample.java

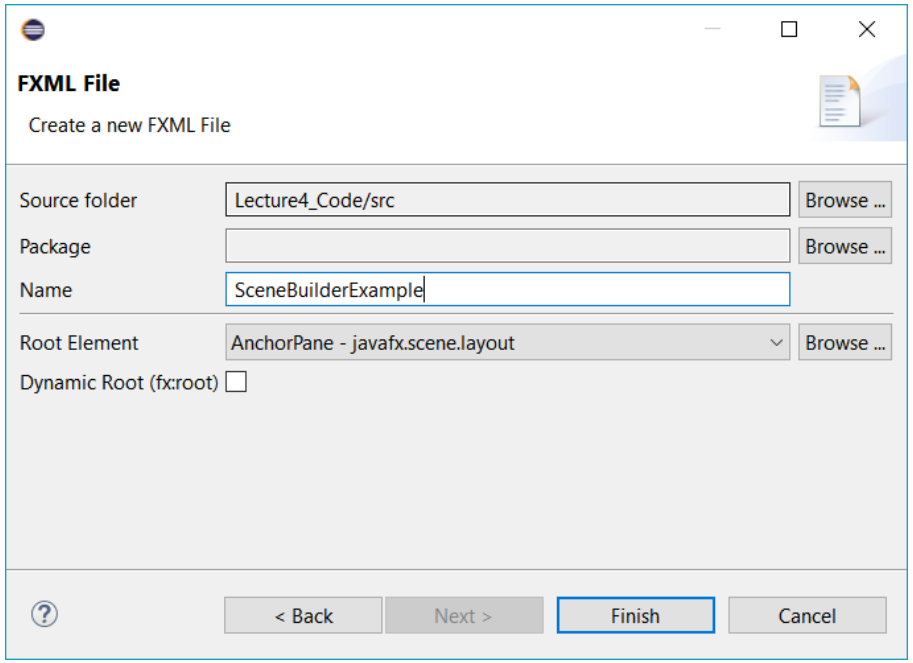
# Scene builder

- JavaFX Scene Builder is a visual layout tool that lets users quickly design JavaFX application user interfaces, without coding.
- FXML code for the layout that they are creating is automatically generated in the background.
- FXML file that can then be combined with a Java project by binding the UI to the application's logic

# Scene builder (to create a new .fxml)

› New > other > New FXML Document

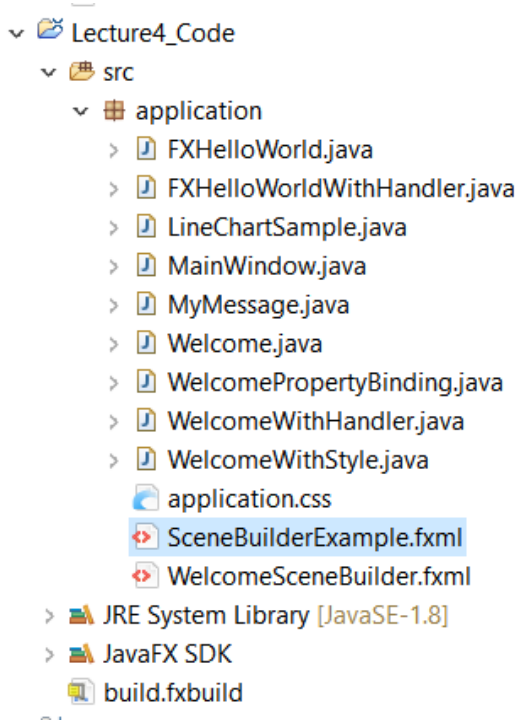


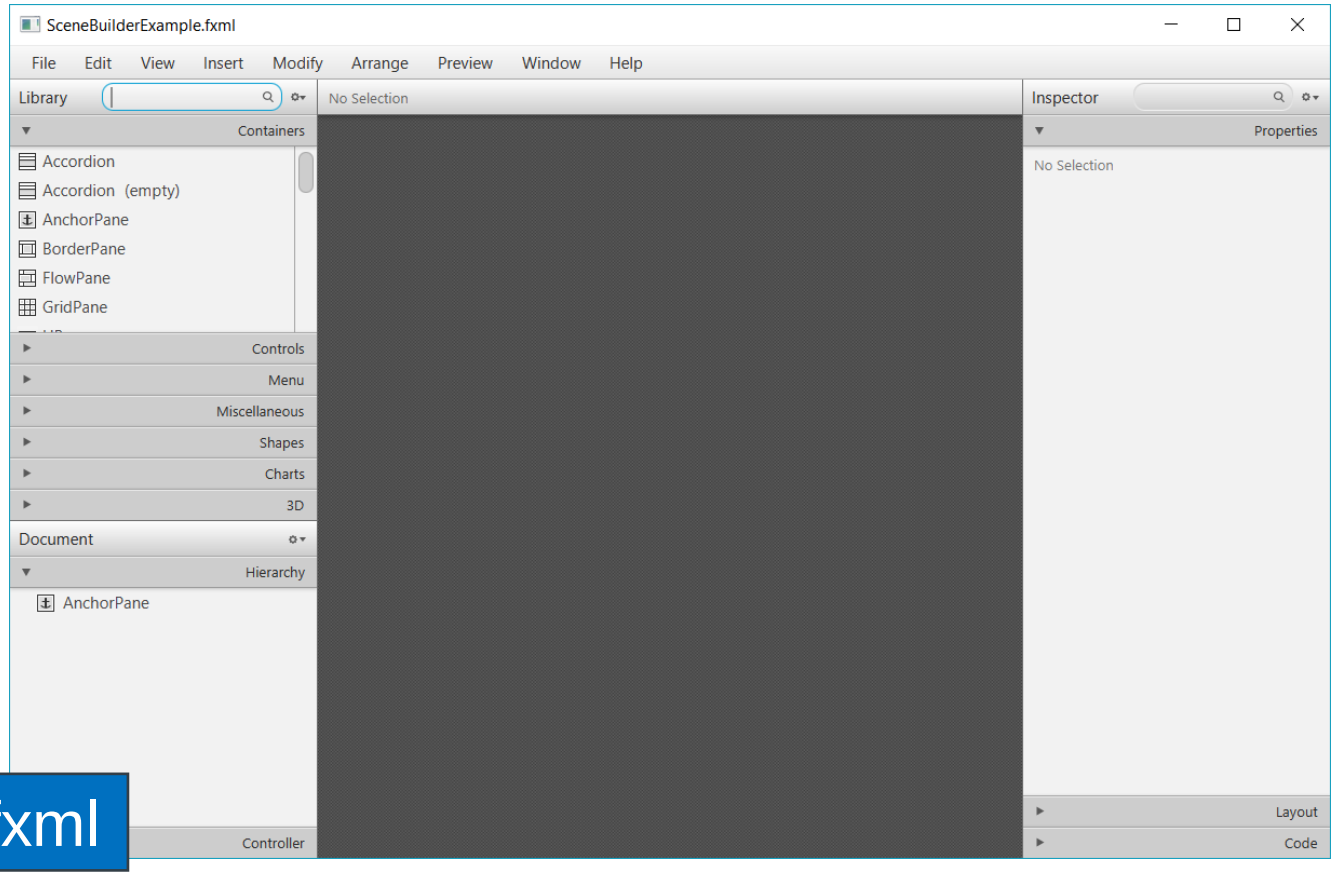




(To create a new .fxml, you can copy another .fxml file too)

# Scene builder (cont.)

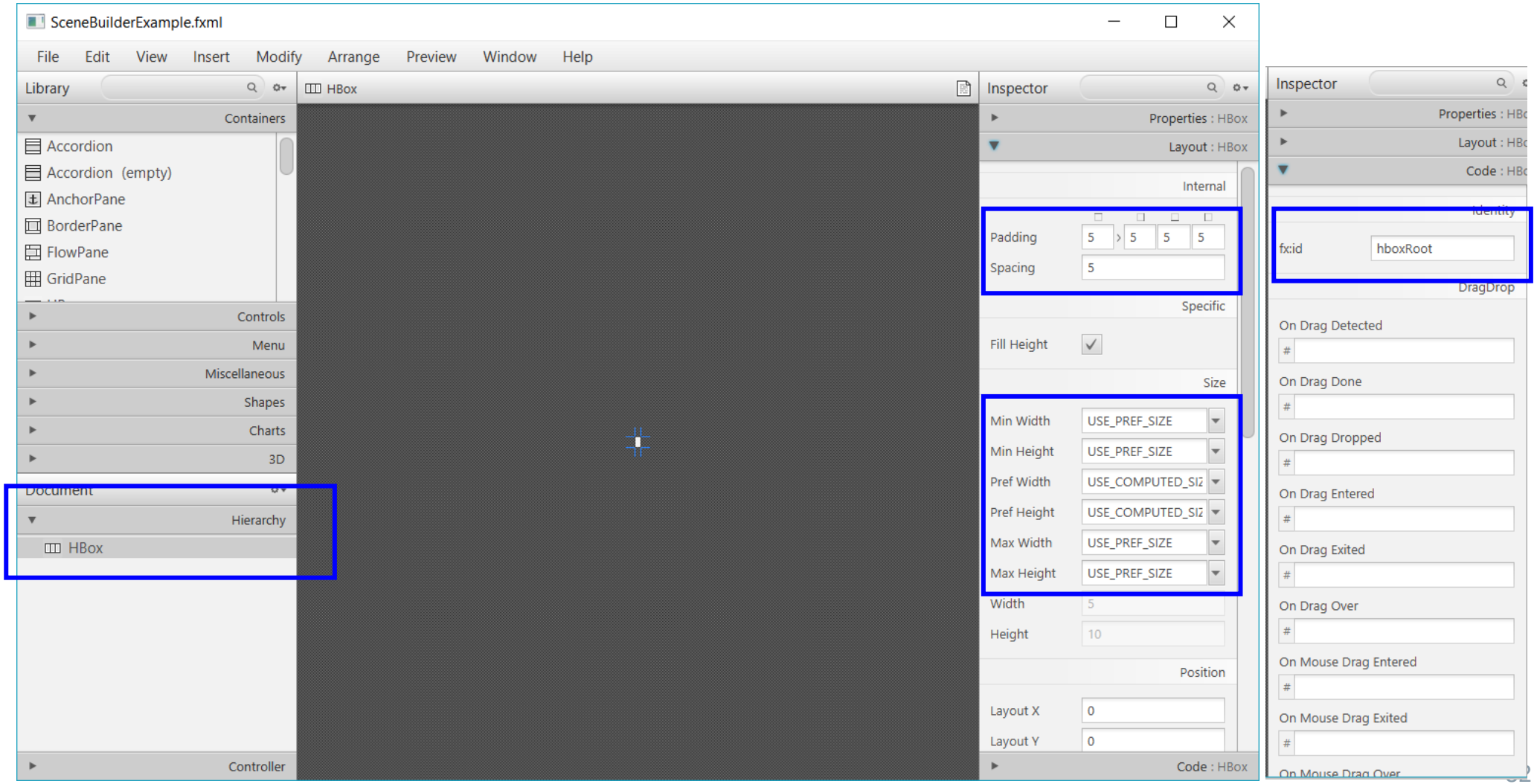
› Right click .fxml file > open with SceneBuilder



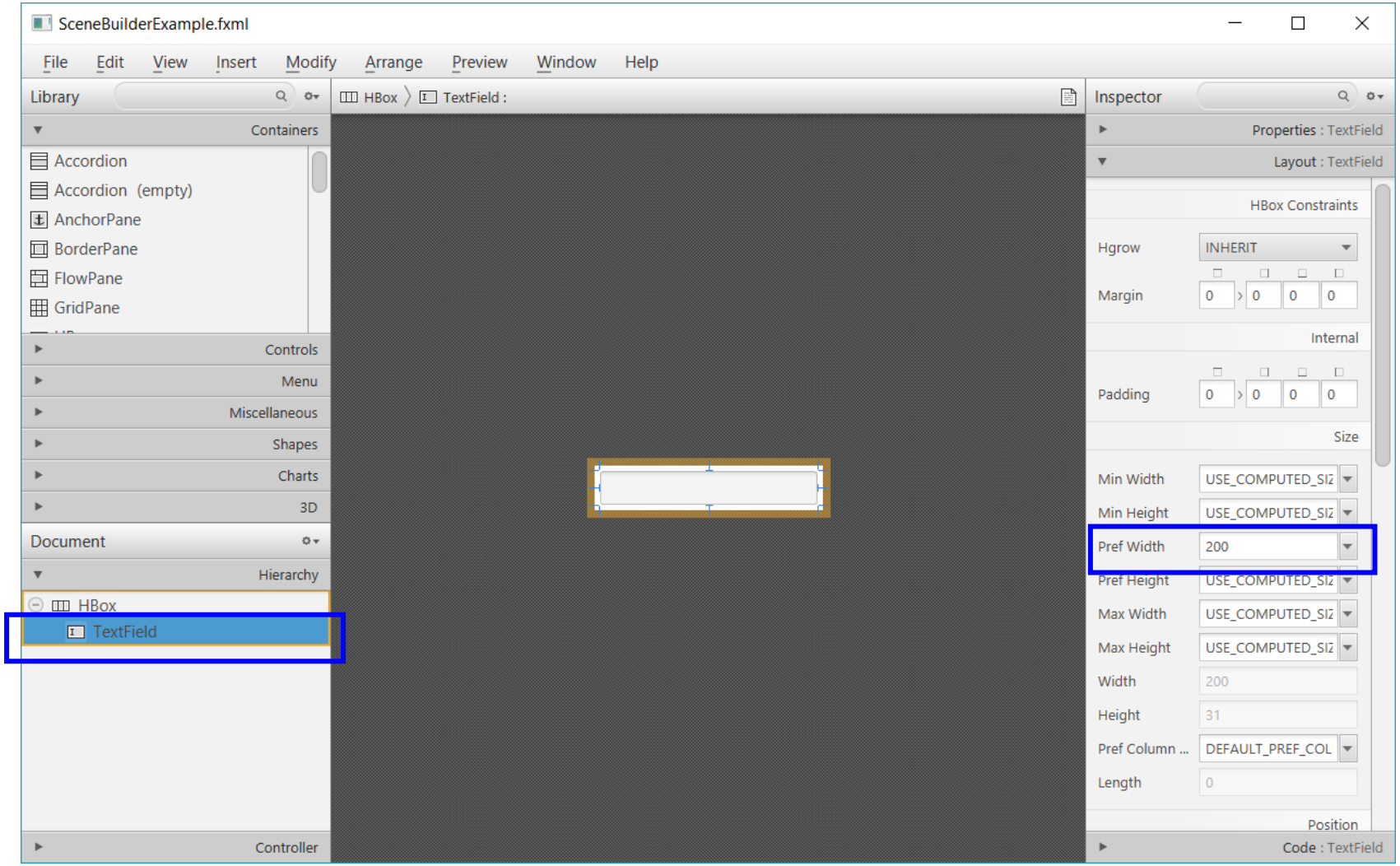



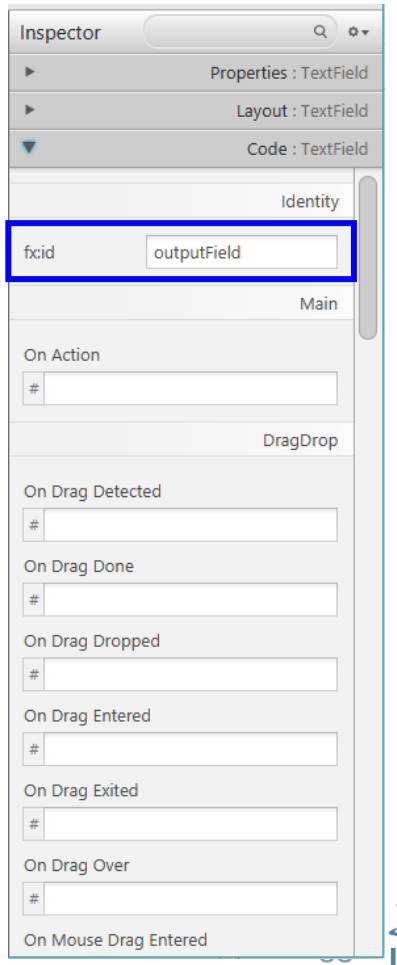

SceneBuilderExample.fxml

# Scene builder (cont.)

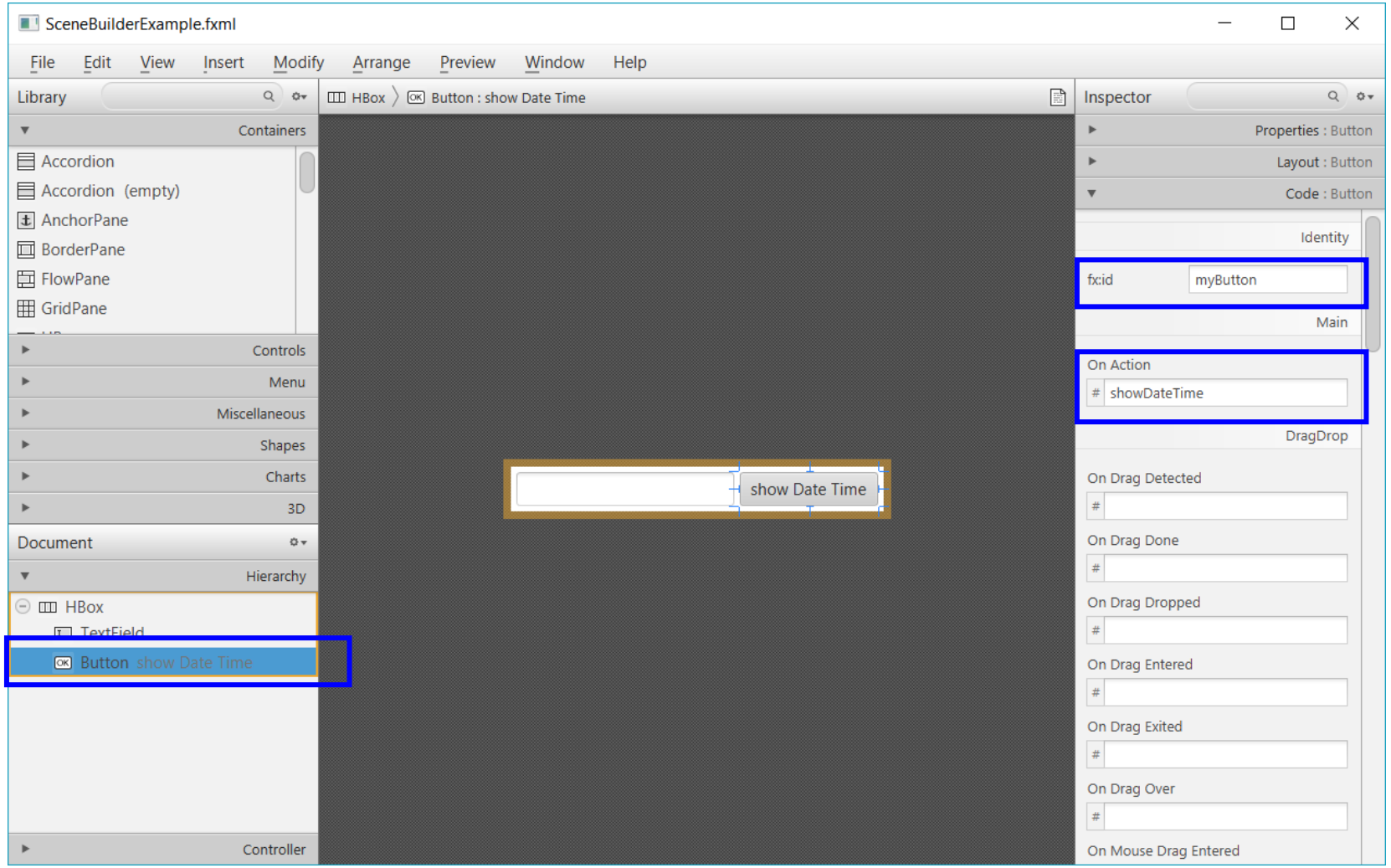


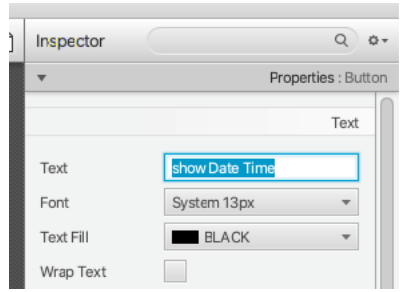
# Scene builder (cont.)



# Scene builder (cont.)



# Scene builder (cont.)

- Save
- Drag file to editor in eclipse to view FXML

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.geometry.*?>
4  <?import javafx.scene.control.*?>
5  <?import javafx.scene.text.*?>
6  <?import java.lang.*?>
7  <?import javafx.scene.layout.*?>
8  <?import javafx.scene.layout.AnchorPane?>
9
10 <HBox fx:id="hboxRoot" maxHeight="-Infinity"
11         maxWidth="-Infinity" minHeight="-Infinity"
12         minWidth="-Infinity" spacing="5.0"
13         xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">
14    <children>
15       <TextField fx:id="outputField" editable="false" prefWidth="200.0" />
16       <Button fx:id="myButton" mnemonicParsing="false" onAction="#showDateTime" text="show Date Time" />
17    </children>
18    <padding>
19       <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
20    </padding>
21 </HBox>
```

# Scene builder (cont.)

› Adding the attribute fx:controller to <Hbox>, the Controller will be useful to the Controls lying inside Hbox such as myButton and outputField.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import javafx.geometry.*?>
4  <?import javafx.scene.control.*?>
5  <?import javafx.scene.text.*?>
6  <?import java.lang.*?>
7  <?import javafx.scene.layout.*?>
8  <?import javafx.scene.layout.AnchorPane?>
9
10 <HBox fx:id="hboxRoot" maxHeight="-Infinity"
11         maxWidth="-Infinity" minHeight="-Infinity"
12         minWidth="-Infinity" spacing="5.0"
13         xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
14         fx:controller="application.MyController">
15    <children>
16       <TextField fx:id="outputField" editable="false" prefWidth="200.0" />
17       <Button fx:id="myButton" mnemonicParsing="false" onAction="#showDateTime" text="show Date Time
18    </children>
19    <padding>
20       <Insets bottom="5.0" left="5.0" right="5.0" top="5.0" />
21    </padding>
22 </HBox>
23
```

Must be public!

```
1  package application;
2
3  import java.net.URL;
4  import java.text.DateFormat;
5  import java.text.SimpleDateFormat;
6  import java.util.Date;
7  import java.util.ResourceBundle;
8
9  import javafx.event.ActionEvent;
10 import javafx.fxml.FXML;
11 import javafx.fxml.Initializable;
12 import javafx.scene.control.Button;
13 import javafx.scene.control.TextField;
14
15 public class MyController implements Initializable {
16
17    @FXML
18    private Button myButton;
19
20    @FXML
21    private TextField outputField;
22
23    @Override
24    public void initialize(URL location, ResourceBundle resources) {
25    }
26
27    // When user click on myButton
28    // this method will be called.
29    public void showDateTime(ActionEvent event) {
30        System.out.println("Button Clicked!");
31        Date now = new Date();
32        DateFormat df = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss.SSS");
33        // Model Data
34        String dateTimeString = df.format(now);
35        // Show in VIEW
36        outputField.setText(dateTimeString);
37
38    }
39
40 }
```



# Scene builder (cont.)

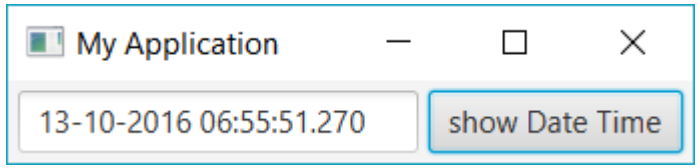
› Run “MyApplication”

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.fxml.FXMLLoader;
5  import javafx.scene.Parent;
6  import javafx.scene.Scene;
7  import javafx.stage.Stage;
8
9  public class MyApplication  extends Application {
10
11     @Override
12     public void start(Stage primaryStage) {
13         try {
14             // Read file fxml and draw interface.
15             Parent root = FXMLLoader.load(getClass()
16                     .getResource("SceneBuilderExample.fxml"));
17
18             primaryStage.setTitle("My Application");
19             primaryStage.setScene(new Scene(root));
20             primaryStage.show();
21
22         } catch(Exception e) {
23             e.printStackTrace();
24         }
25     }
26
27     public static void main(String[] args) {
28         launch(args);
29     }
30
31 }
```

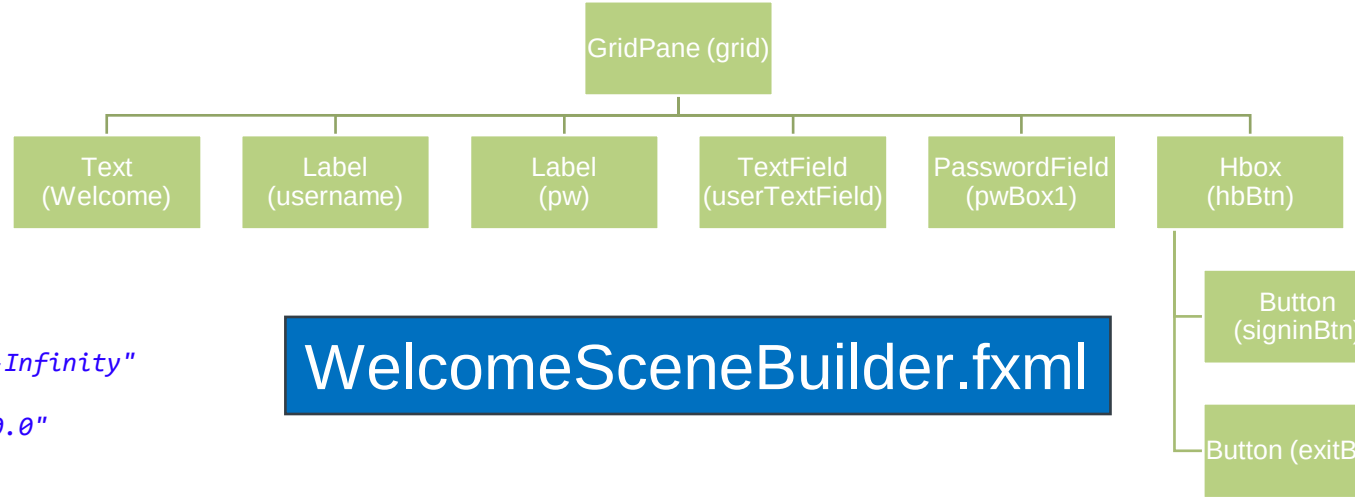
SceneBuilderExample.fxml

MyController.java

MyApplication.java



# FXML




## WelcomeSceneBuilder.fxml

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

<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>

<GridPane hgap="10.0" maxHeight="-Infinity" maxWidth="-Infinity"
          minHeight="-Infinity" minWidth="-Infinity"
          prefHeight="300.0" prefWidth="350.0" vgap="10.0"
          xmlns="http://javafx.com/javafx/8"
          xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Welcome">
         <font>
            <Font name="Tahoma" size="20.0" />
         </font>
      </Text>
      <Label text="User Name:" GridPane.rowIndex="1" />
      <Label text="Password:" GridPane.rowIndex="2" />
      <HBox alignment="BOTTOM_RIGHT" prefHeight="100.0"
            prefWidth="200.0" spacing="10.0" GridPane.columnIndex="1"
            GridPane.rowIndex="4">
         <children>
            <Button mnemonicParsing="false" text="Sign in" />
            <Button mnemonicParsing="false" text="Exit" />
         </children>
      </HBox>
      <TextField GridPane.columnIndex="1" GridPane.rowIndex="1" />
      <PasswordField GridPane.columnIndex="1" GridPane.rowIndex="2" />
   </children>
   <columnConstraints>
      <ColumnConstraints hgrow="SOMETIMES" maxWidth="263.0"
                         minWidth="10.0" prefWidth="87.0" />
      <ColumnConstraints hgrow="SOMETIMES" maxWidth="463.0"
                         minWidth="10.0" prefWidth="203.0" />
   </columnConstraints>
   <padding>
      <Insets bottom="25.0" left="25.0" right="25.0" top="25.0" />
   </padding>
   <rowConstraints>
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
   </rowConstraints>
</GridPane>
```

# CSS

- JavaFX provides styling by Cascading Style Sheets(CSS).
- CSS support is based on the W3C CSS version 2.1
- JavaFX CSS document: https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html

# CSS (cont.)

```
// set style
grid.setStyle("-fx-background-color:lightgray;");
scenetitle.setStyle("-fx-font-size: 32px;
                    -fx-font-family:\"Arial Black\";
                    -fx-fill: #555;");
signinBtn.setStyle("-fx-text-fill: white;
                    -fx-font-weight: bold;
                    -fx-font-family: \"Arial Narrow\";
                    -fx-background-color: darkgreen;");
exitBtn.setStyle("-fx-text-fill: white;
                    -fx-font-weight: bold;
                    -fx-font-family: \"Arial Narrow\";
                    -fx-background-color: darkred;");
```

WelcomeWithStyle.java


Remarks: you can set same style for more than one node using “css class” or writing the style in separated file (not covered in this class)

# Binding properties

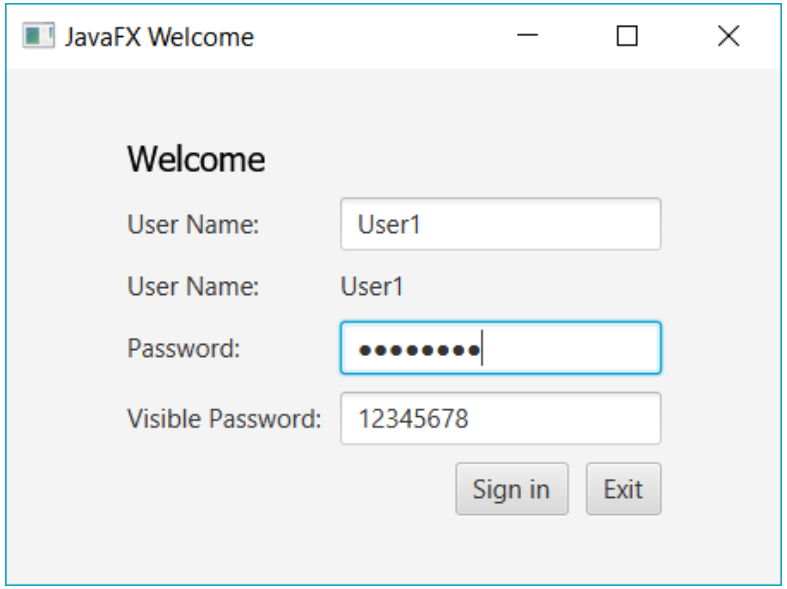
- JavaFX introduces a new concept called binding property
- Enables a target object to be bound to a source object.
- If the value in the source object changes, the target property is also changed automatically.
- The target object is simply called a binding object or a binding property.

# Binding Properties (cont.)

```
Label userName = new Label("User Name:");
grid.add(userName, 0, 1);
TextField userTextField = new TextField();
grid.add(userTextField, 1, 1);

Label userName1 = new Label("User Name:");
grid.add(userName1, 0, 2);
Label userNameOut = new Label();
grid.add(userNameOut, 1, 2);

// Unidirectional bindings
userNameOut.textProperty().bind(userTextField.textProperty());
```

```
Label pw1 = new Label("Password:");
grid.add(pw1, 0, 3);
PasswordField pwBox1 = new PasswordField();
grid.add(pwBox1, 1, 3);

Label pw2 = new Label("Visible Password:");
grid.add(pw2, 0, 4);
TextField pwBox2 = new TextField();
grid.add(pwBox2, 1, 4);

// Bidirectional bindings
pwBox1.textProperty().bindBidirectional(pwBox2.textProperty());
```

WelcomePropertyBinding.java



# Event Handling

- To make the program response to an action, you need to create a listener object that waits for a particular event to handle and modified the correspondence method.
- There are many events on GUI:
  - ActionEvent, InputEvent, ScrollToEvent, WindowEvent, WebEvent, MouseEvent, KeyEvent, …
- JavaFX event is an instance of the javafx.event.Event class or its subclass

# Event Handling

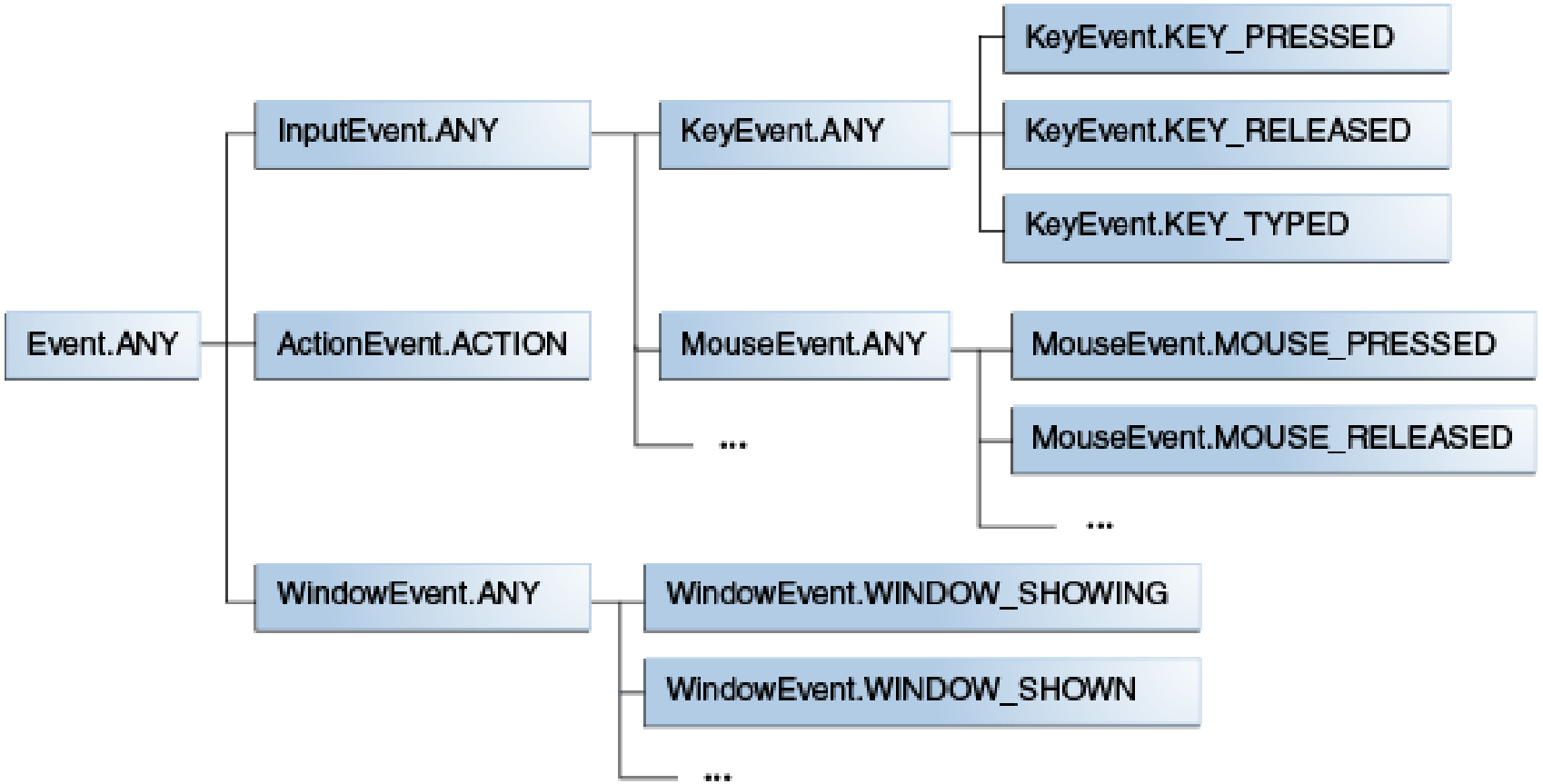
- Use the setOnXXX methods to register event handlers

  setOnEvent-type(EventHandler<? super event-class> value)

  - Event-type is the type of event that the handler processes,
    setOnKeyTyped for Key Typed events
    setOnMouseClicked for Mouse Clicked events.
  - event-class is the class that defines the event type,
    KeyEvent for events related to keyboard input
    MouseEvent for events related to mouse input.

- Override handle method

# Event Handling (cont.)




Event type hierarchy
Reference: http://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

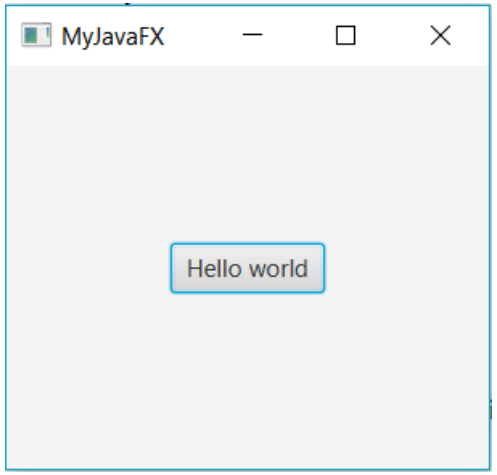
# Event Handling (cont.)

FXHelloWorldWithHandler.java

```
package application;

import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.control.Button;
. . .

public class FXHelloWorld extends Application {
	// Override the start method in the Application class
	@Override
	public void start(Stage primaryStage) {
		// Create a scene and place a button in the scene
		Button btn = new Button("Hello world");
		. . .
		StackPane root = new StackPane();
		root.getChildren().add(btn);
		Scene scene = new Scene(root, 300, 250);
		primaryStage.setTitle("MyJavaFX"); // Set the stage title
		primaryStage.setScene(scene); // Place the scene
		primaryStage.show();
	}

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

```
import javafx.event.ActionEvent;

import javafx.event.EventHandler;
```

```
// set event handler

btn.setOnAction(new EventHandler<ActionEvent>() {

	public void handle(ActionEvent event) {

		System.out.println("Hello World");

	}

});
```




Problems @ Javadoc Declaration

<terminated> FXHelloWorldWithHandler

```
Hello World
Hello World
Hello World
```

# Event Handling (cont.)

- setOnAction() method is used to register an event handler.
- handle() method in the event handler is called when user clicks the button and it print "Hello World" to the console.

# Event Handling (cont.)

- Clear User Name when press ESC
- Change button width if mouse is over
- Popup welcome dialog when click Sign in
- Close application when click Exit

WelcomeWithHandler.java

# Common Event-Handling Problem

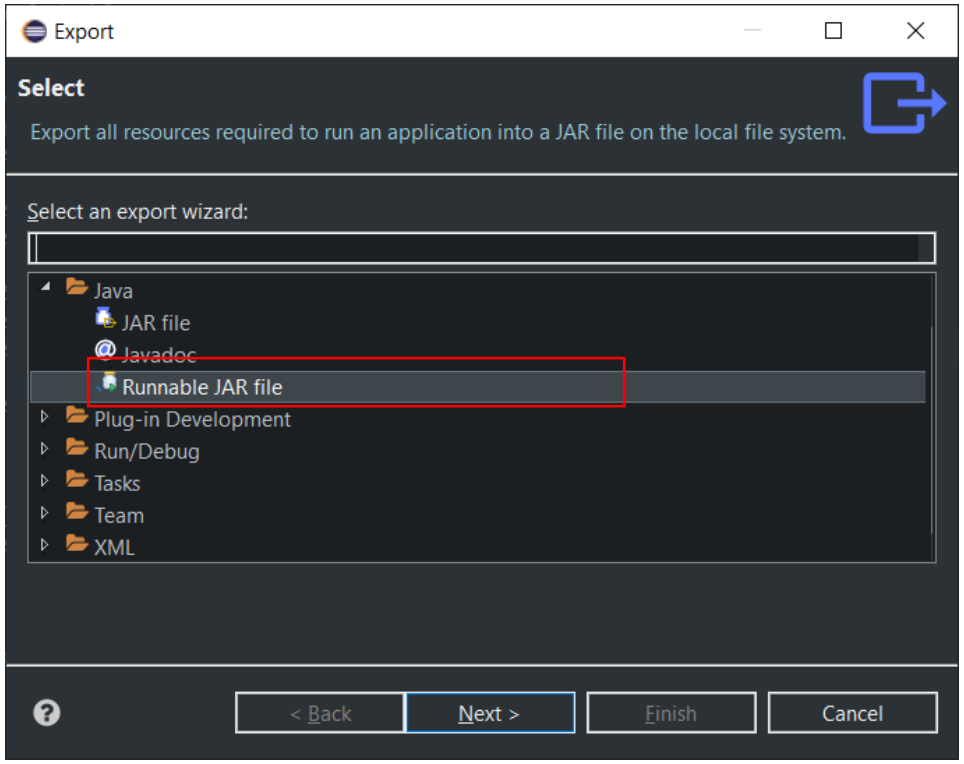
- A component does not generate the events it should.
  - Did you register the right kind of listener to detect the events?
  - Did you register the listener to the right object?
  - Did you implement the event handler correctly?

# Export Jar

- We've managed to create our Java FX Application
- Let's try out our application as an executable JAR

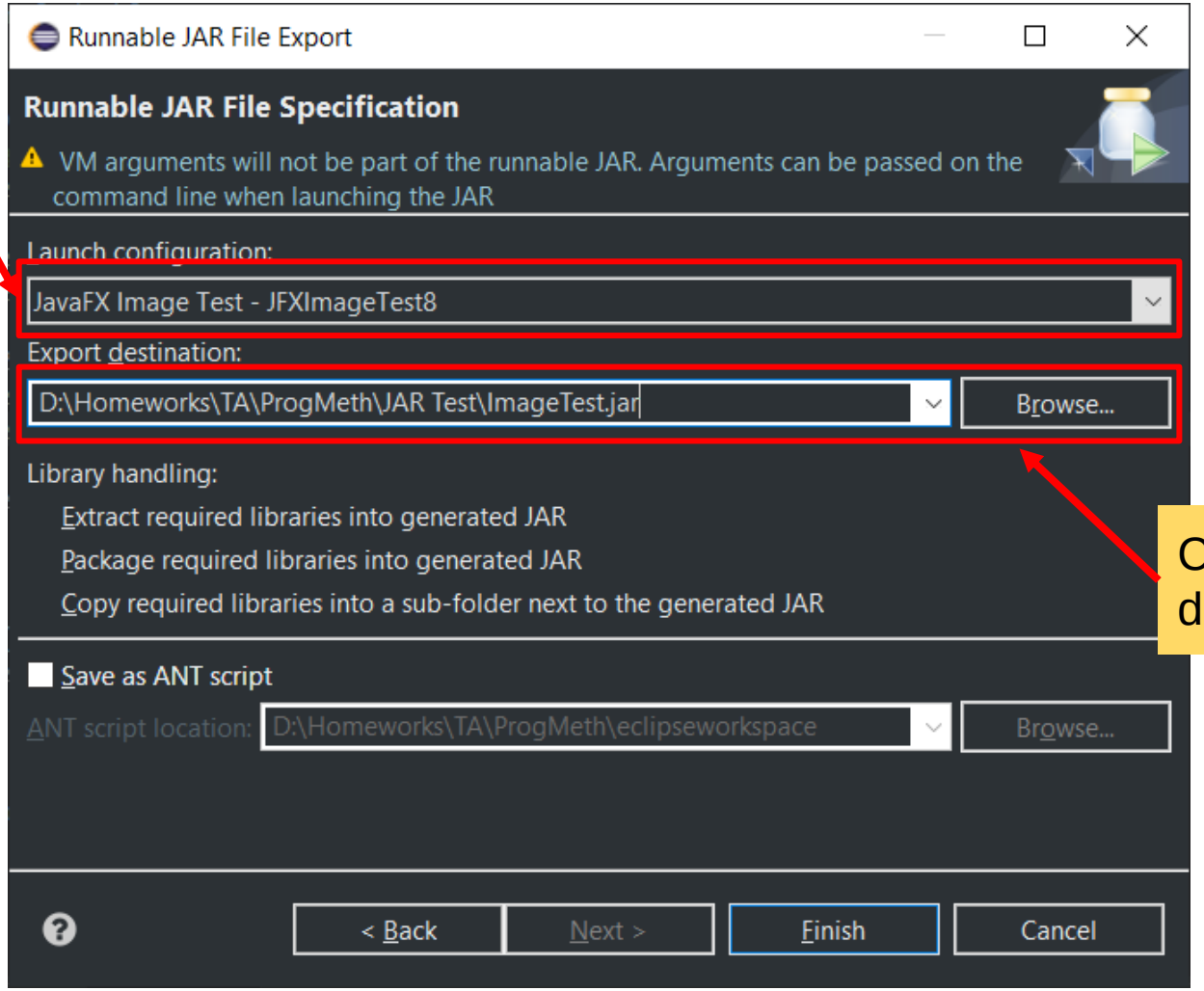
# How to export a runnable jar file

- Click File > Export
- Choose Java > Runnable JAR File



# How to export a runnable jar file

Be sure to pick correct Launch configuration for the project!




Choose the export destination here.

# Using VM Arguments to Run .jar files outside the IDE

1. Export the .jar file.
2. Open cmd in the folder your .jar file is in
3. Type the following into your command line:

```
java -jar --module-path "(your javafx libpath here)" --add-modules
javafx.controls,javafx.fxml (your jar file name).jar

Example:
java -jar --module-path "C:\Program Files\Java\javafx-sdk-12.0.2\lib" --
add-modules javafx.controls,javafx.fxml ImageLoader.jar

Note that in MacOS, you should NOT use quotes. You should also use slash
(/) instead of backslash (\)
```

Your program should run now!

C:\Windows\System32\cmd.exe

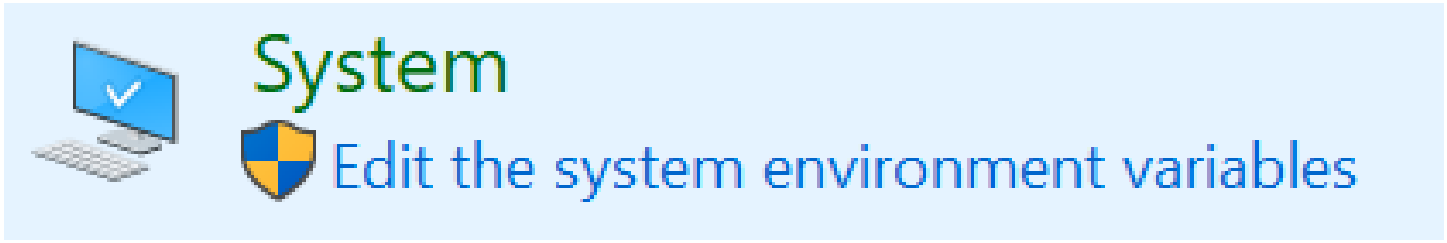
```
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\Users\RamBanjo\eclipse-workspace\JAVA_FX_Image>java -jar --module-path "C:\Program Files\Java\javafx-sdk-12.0.2\lib"
--add-modules javafx.controls,javafx.fxml ImageLoader.jar

D:\Users\RamBanjo\eclipse-workspace\JAVA_FX_Image>
```

# Setting System Variable to Save Time

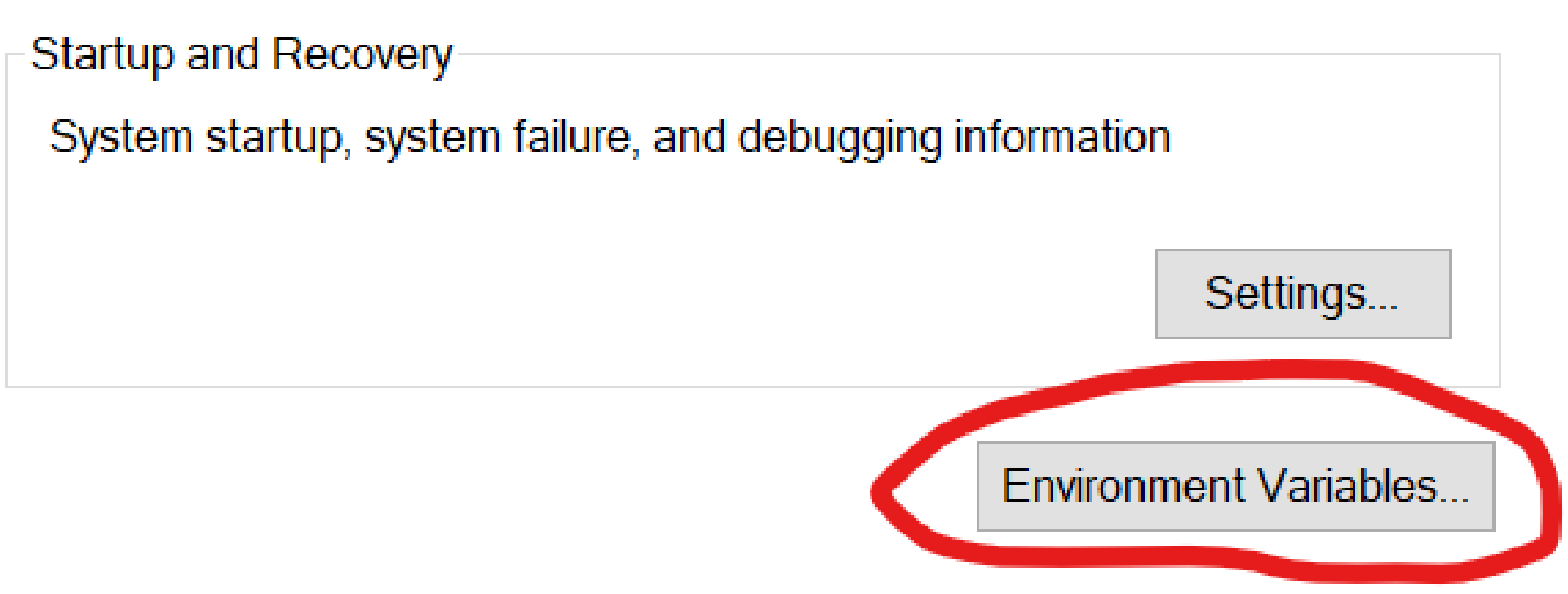
Note: for windows users only!

for mac users, please visit the tutorial here:
http://osxdaily.com/2015/07/28/set-enviornment-variables-mac-os-x/

1. Go to your Control Panel
2. Search for “System Variable” and click on this:





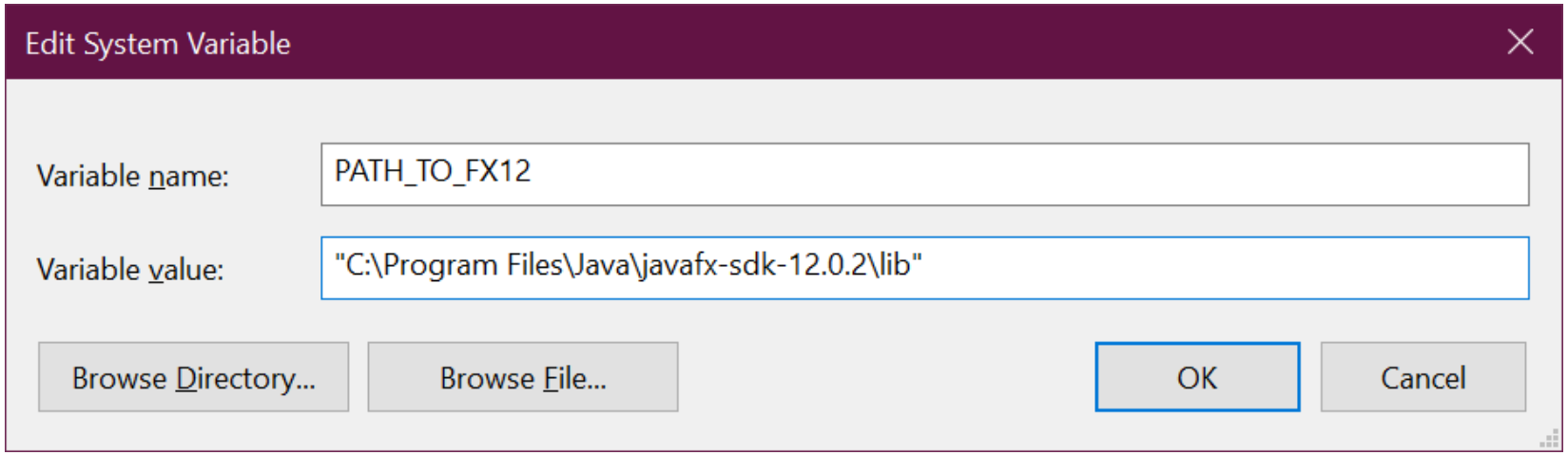

Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...

# Setting System Variable to Save Time

3.) Under “System Variables”, click New…

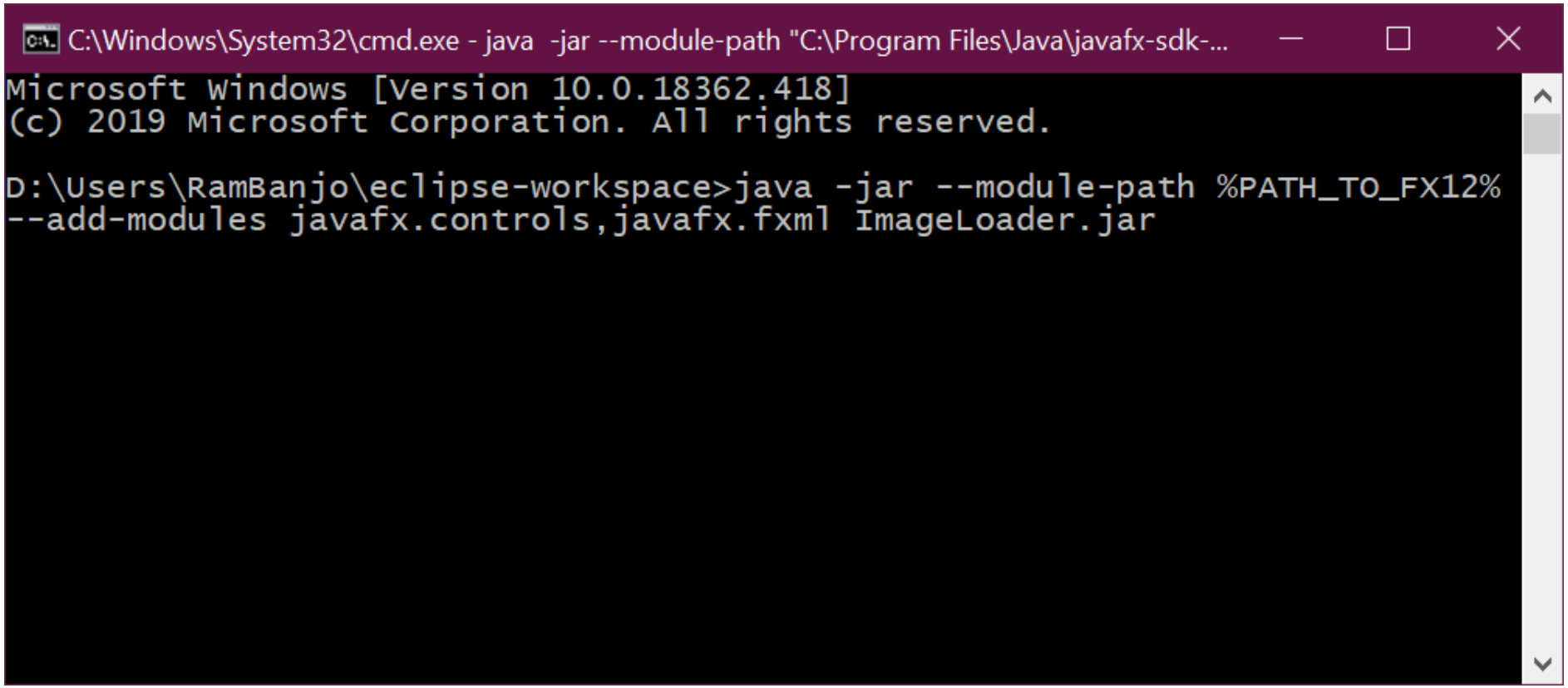
4.) Put in your JavaFX lib path (with quotes), and put the name you want to use

5.) Click OK, then Apply and Close the Environment Variables window



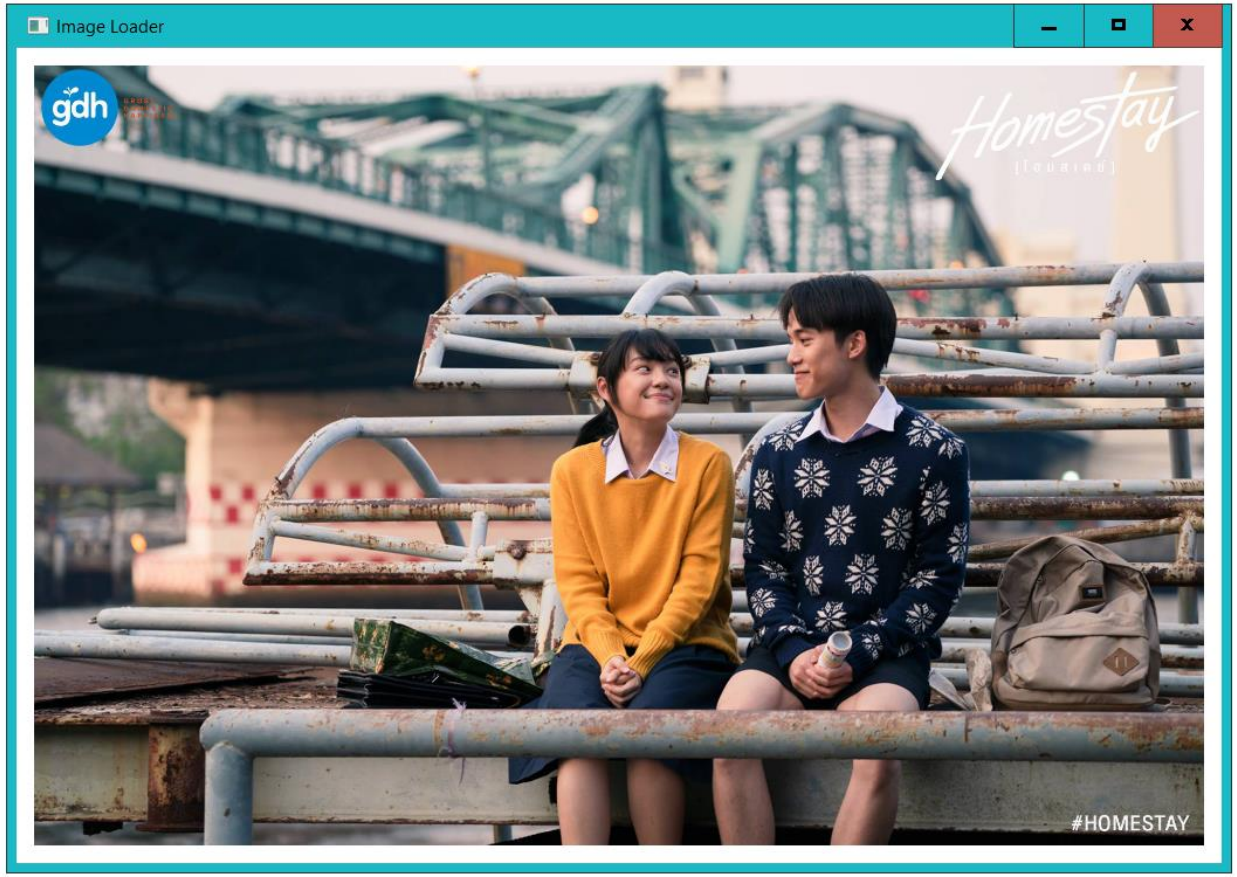
# Setting System Variable to Save Time

You should now be able to use that variable instead of typing the entire path



# How to export Jar with picture

› Run -> JAVA_FX_Image/ImageLoader.jar



# How to export Jar with picture (cont.)

- Let's copy our ImageLoader.jar to somewhere
- Run -> JAVA_FX_Image/Test_Jar/1_only_jar/run.jar

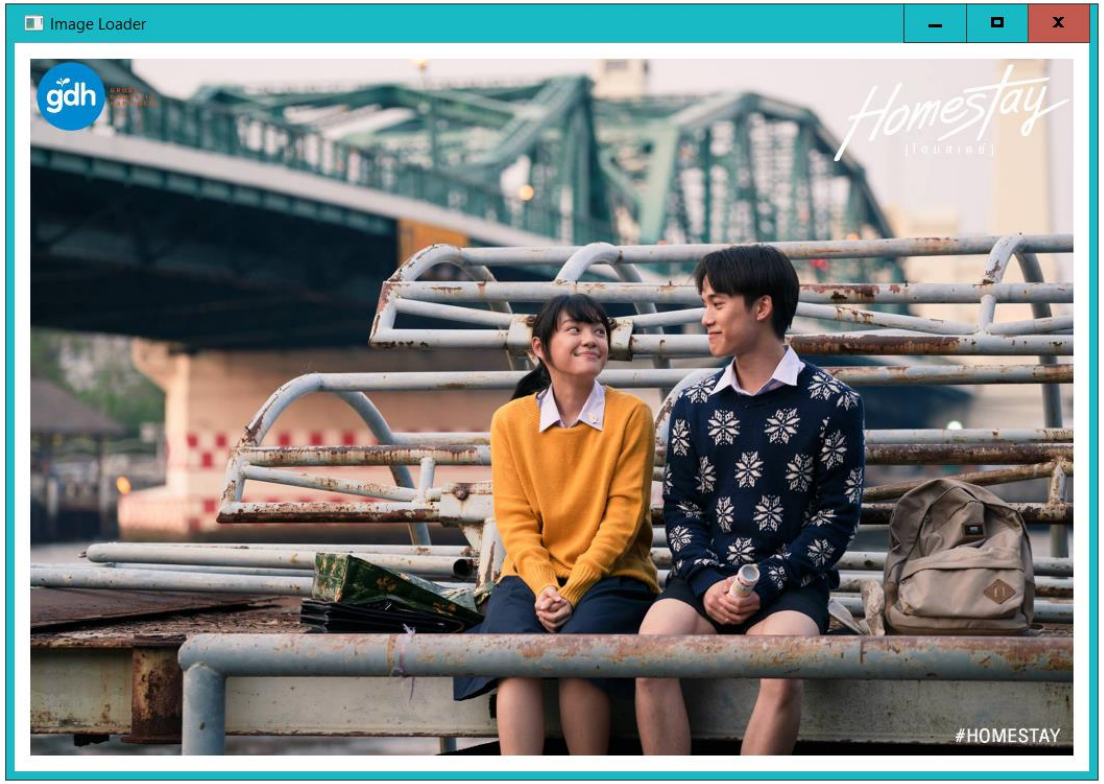

Our Image doesn't appear anymore

# Export Jar with res folder

- Let's take a look at how we load our image
  - ImageView imageView = new ImageView(new Image("file:res/images/homestay.jpg"));
- The image must be in the same directory as our JAR
  - Let's try again

# Export Jar with res folder (cont.)

- Run -> JAVA_FX_Image/Test_Jar/2_jar_with_res_folder/run.jar
- It works !!!

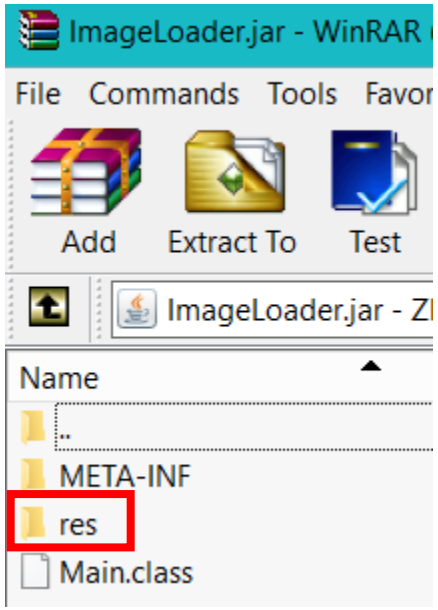


# Export Jar containing res folder

- Keeping resource beside our JAR makes it work.
- But it would be better if we can store all our resources into our JAR

# Export Jar containing res folder (cont.)

› Run -> JAVA_FX_Image/Test_Jar/3_jar_contain_res_folder/run.jar





Our Image still doesn't appear

# Export Jar containing res folder (cont.)

- Why?
  - Because ImageView imageView = new ImageView(new Image("file:res/images/homestay.jpg"));
  - Can get resource from file only
- How to fix it?

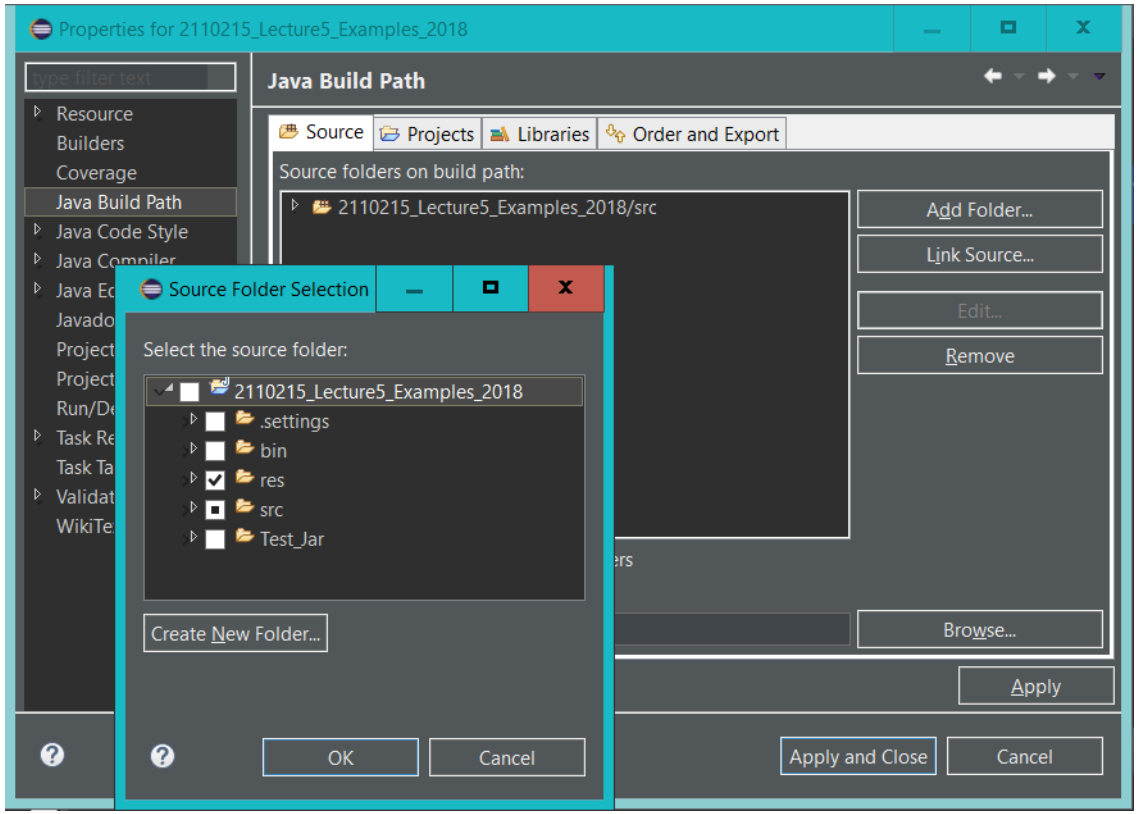
# Export Jar containing res folder - ClassLoader

- Use ClassLoader to help loading our image
  - A path to our resource related to our .class file directory
- ClassLoader.*getSystemResource(String filePath)*
  - Return as URL

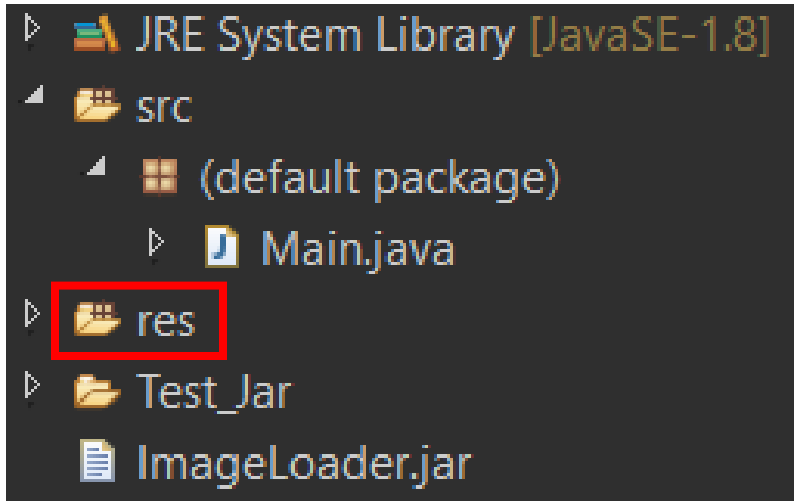
- Example:
  - String image_path = **ClassLoader.*getSystemResource("images/homestay.jpg").toString();***
  - ImageView imageView = **new ImageView(**new Image(image_path)***);***

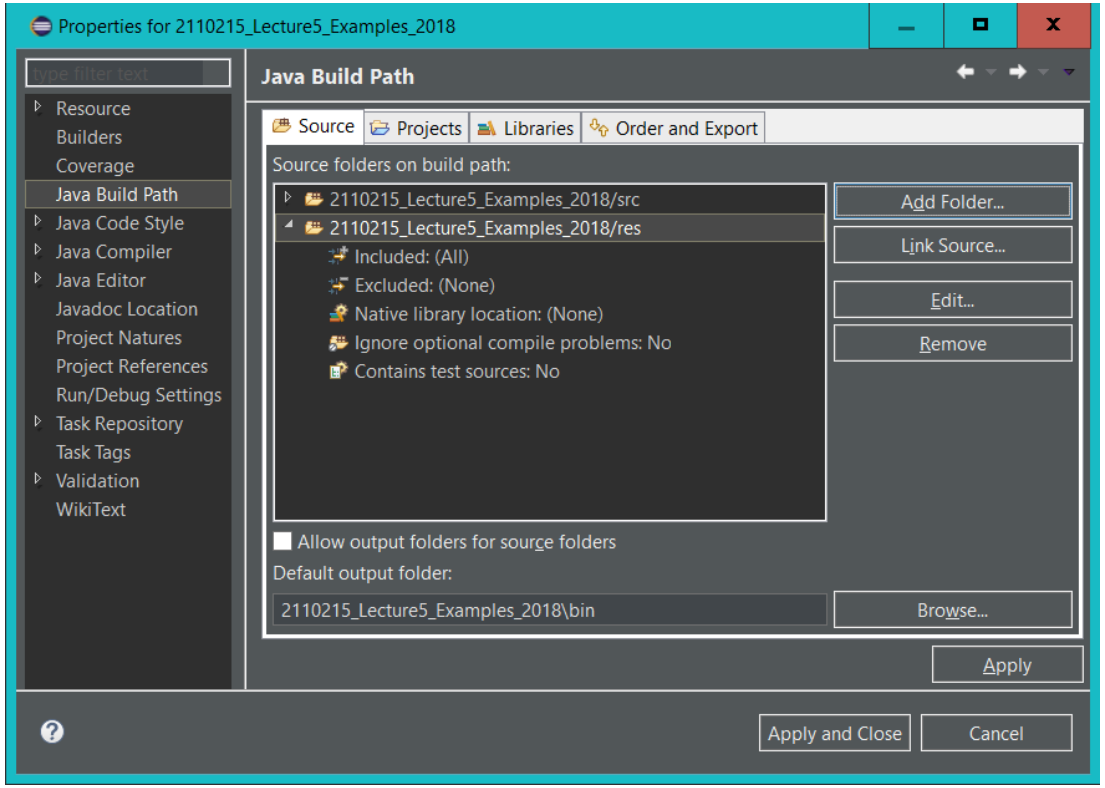
# Export Jar containing res folder - BuildPath

- For add resoure folder, Build Path
- -> Configure Build Path
- -> Source (Tab)
- -> Add Folder
- -> Select Folder res



# Export Jar containing res folder - BuildPath



# Export Jar containing res folder (cont.)

- Run -> JAVA_FX_Image/Test_Jar/4_jar_fixed/run.jar
- This works because it read resource from our jar file.