




# Application Program Development

Segment : GUI with JavaFx



# Outcomes

- Understanding User Interfaces
  - What is a model?
  - A GUI (Graphical User Interface)
  - An Application
  - Window Component
  - Container
- 

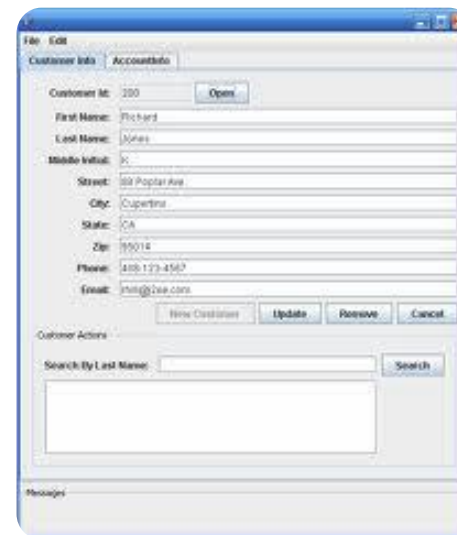
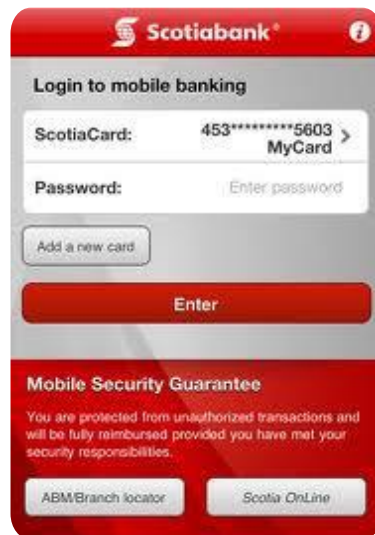
# Understanding user interfaces

- All applications require some kind of **user interface** which allows the user to interact with the underlying program/software.
- Most user interfaces have the ability to take-in information from the user and also to provide visual or audible information back to the user.
- In the *real* world, an interface often has physical interactive components. For example, there are two obvious ways to interact with (or *interface* with) our bank account.
  - We may go up to a teller at the bank and perform some transactions, or we may use an ATM.



# Understanding user interfaces

- In the *virtual* world, we may interact with our bank account electronically by using a web browser, or phone app or dedicated stand-alone software from the bank.
- In this case, the interaction is through software menus, buttons, text fields, lists, etc..
- Up until this point, our programs/applications did not really have any interactive user interface. That is, we defined classes, wrote programs and then simply ran them inside of the IDE that we were using. The results of our program were displayed as text output in the JAVA console window.



- When writing programs that bring up a main interactive window (or those that run on a phone/tablet or in a browser), it is important to understand that more is "going on behind the scenes".

## Model

- The **model** of an application consists of all classes that represent the "business logic" part of the application ... the underlying system on which a user interface is attached.
  - The model is always developed separately from the user interface.
  - It should not assume any knowledge about the user interface at all (e.g., model classes should not assume that **System.out.println()** is available).

## User Interface


- The **user interface** is the part of the application that is attached to the model which handles interaction with the user and does NOT deal with the business logic.
  - The user interface always makes use of the model classes and often causes the model to change according to user interaction.
  - The changes to the model are often reflected back (visually) on the user interface as a form of immediate feedback.

## GUI

- A **graphical user interface (GUI)** is a user interface that makes use of one or more windows to interact with the user.

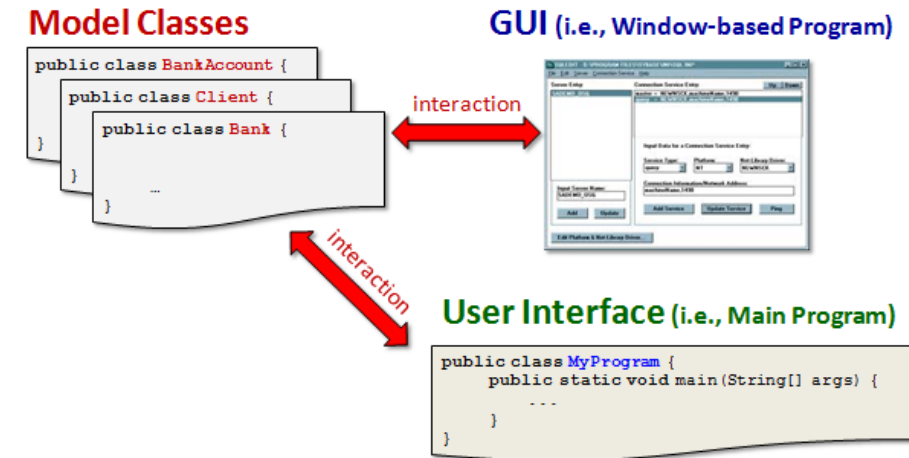


# What is GUI?

- A **graphical user interface (GUI)** presents a user-friendly mechanism for interacting with an app.
  - A GUI gives an app a distinctive “look-and-feel.”
  - GUIs are built from **GUI components**—also called controls or widgets (short for window gadgets).
  - A GUI component is an object with which the user interacts via the mouse, the keyboard or another form of input, such as voice recognition.
- 

# GUI Conti...

- A GUI is often preferred over text-based user interfaces because it is more natural ... more like real-world applications that we are used to using. Imagine, for example, if the internet was only text-based with no buttons, text fields, drop-down lists, images, etc..
- So, it is important to understand that there should always be a separation in your code between the **model** classes and the **user interface** classes.
- Such a separation allows you to share the same model classes with different interfaces.
- *An **application (or app)** is a computer program with a graphical user interface which can interact with user to perform tasks & calculations, obtain & visualize information, and potentially interact with the real world through sensors and hardware.*




# One slide History of GUI with Java

- Java's original GUI library was the Abstract Window Toolkit (AWT).
- Swing was added to the platform in Java SE 1.2.
- Swing was the primary Java GUI technology.
- Swing will remain part of Java and is still widely used.
- **JavaFX** is Java's GUI (current), graphics and multimedia API of the future.
- Sun Microsystems (acquired by Oracle in 2010) announced JavaFX in 2007 as a competitor to Adobe Flash and Microsoft Silverlight.





# What is JavaFx?

- JavaFX is an open source Java-based framework for developing rich client applications.
  - Currently it is comparable to other frameworks on the market such as Adobe AIR and Microsoft Blazor.
  - The JavaFX library is available as a public Java application programming interface (API)
- 

# JavaFx vs Swing

- JavaFX is easier to use—it provides one API for client functionality, including GUI, graphics and multimedia (images, animation, audio and video).
  - JavaFX Scene Builder can be used standalone or integrated with many IDEs and it produces the same code regardless of the IDE.
  - JavaFX gives you complete control over a JavaFX GUI's look-and-feel via Cascading Style Sheets (CSS).
  - JavaFX has better threading support, which is important for getting the best application performance on today's multi-core systems.
  - JavaFX uses the GPU (graphics processing unit) for hardware-accelerated rendering.
  - JavaFX provides multiple upgrade paths for enhancing existing GUIs.
  - JavaFX capabilities may be embedded into Swing apps via class JFXPanel.
- Swing is only for GUIs, so you need to use other APIs for graphics and multimedia apps.
  - With Swing, many IDEs provided GUI design tools for dragging and dropping components onto a layout; however, each IDE produced different code (such as different variable and method names).
  - Though Swing components could be customized, application performance on today's multi-core systems.
  - Swing GUI capabilities may be embedded into JavaFX apps via class SwingNode.

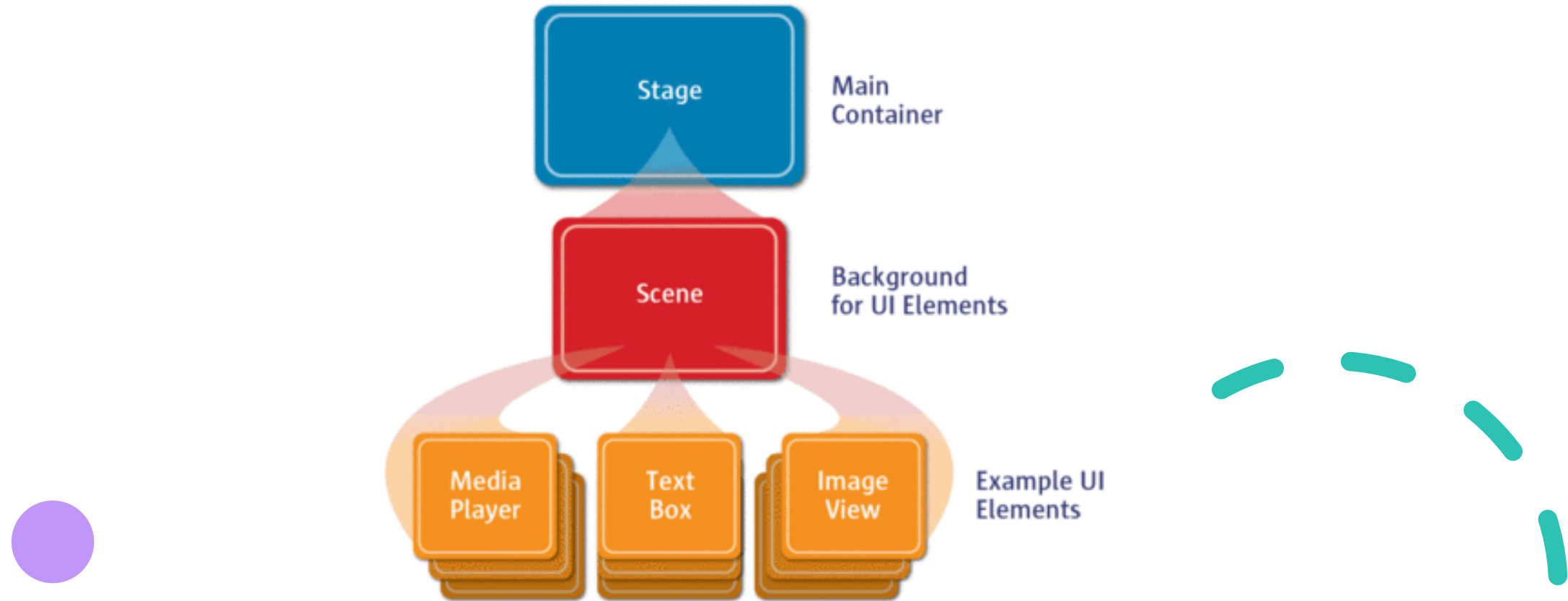
# Java Scene Builder

- The **Scene Builder** tool is a standalone JavaFX GUI visual layout tool that can also be used with various IDEs.
- JavaFX Scene Builder enables you to create GUIs by dragging and dropping GUI components from Scene Builder's library onto a design area.
- Also provides you the opportunity of modifying and styling the GUI's.
- Skinning, a concept which you can achieve in Scene Builder by using **Cascading Style Sheets (CSS)** to change the entire look-and-feel of your GUI.

**Note:** Follow the instruction in the lab on how to install Scene Builder and embed it with your IDE

# FXML (FX Markup Language)

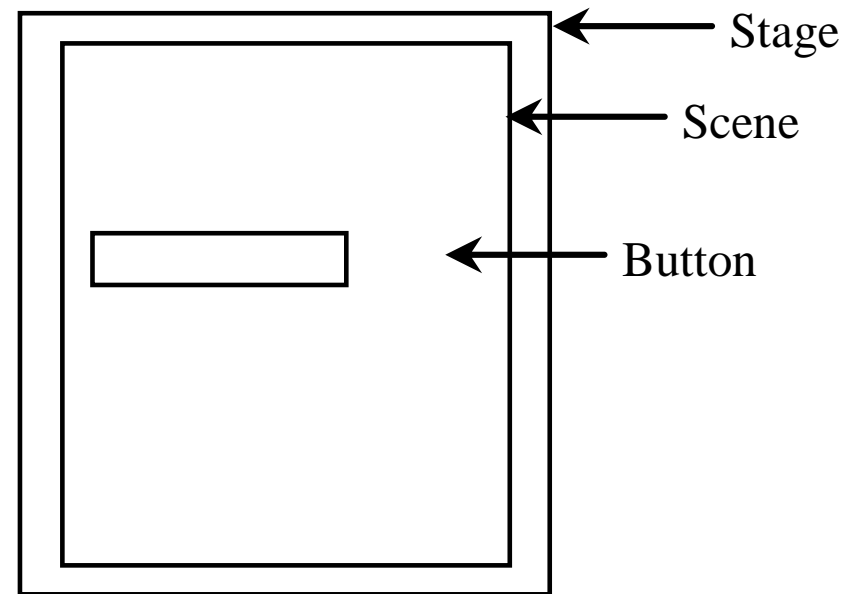
- JavaFX Scene Builder generates **FXML (FX Markup Language)**—an XML vocabulary for defining and arranging JavaFX GUI controls.
- In JavaFX, FXML concisely describes GUI, graphics and multimedia elements.
- JavaFX Scene Builder completely hides the FXML details from you, so you can focus on defining *what* the GUI should contain without specifying *how* to generate it — this is an example of *declarative style programming*.
- This separation of the interface (the GUI) from the implementation (the Java code) makes it easier to debug, modify and maintain JavaFX GUI apps.



# JavaFx Windows Structure

# Basic Structure of JavaFx

- Application
- Override the start(Stage) method
- Stage, Scene, and Nodes



# Basic Structure of JavaFx – Example

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class JavaFXBasic extends Application {
7      @Override // Override the start method in the Application class
8      public void start(Stage primaryStage) {
9          // Create a button and place it in the scene
10         Button btOK = new Button("OK");
11         Scene scene = new Scene(btOK, 200, 250);
12         primaryStage.setTitle("JavaFX Basic Structure"); // Set the stage title
13         primaryStage.setScene(scene); // Place the scene in the stage
14         primaryStage.show(); // Display the stage
15     }
16
17     /**
18      * The main method is only needed for the IDE with limited
19      * JavaFX support. Not needed for running from the command line.
20      */
21     public static void main(String[] args) {
22         launch(args);
23     }
24 }
```

# Display a simple shape in JavaFx – Example

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.stage.Stage;
7
8  public class ShowCircle extends Application {
9      @Override
10     public void start(Stage primaryStage) {
11         // Create a circle and set its properties
12         Circle circle = new Circle();
13         circle.setCenterX(100);
14         circle.setCenterY(100);
15         circle.setRadius(50);
16         circle.setStroke(Color.BLACK); // Set circle stroke color
17         circle.setFill(Color.WHITE);
18
19         // Create a pane to hold the circle
20         Pane pane = new Pane();
21         pane.getChildren().add(circle);
22
23         // Create a scene and place it in the stage
24         Scene scene = new Scene(pane, 200, 200);
25         primaryStage.setTitle("ShowCircle");
26         primaryStage.setScene(scene);
27         primaryStage.show();
28     }
```

```
29
30     /**
31      * The main method is only needed for the IDE with limited
32      * JavaFX support. Not needed for running from the command line.
33      */
34     public static void main(String[] args) {
35         launch(args);
36     }
37 }
```



# Components of JavaFX

- **Stage**

- The window in which a JavaFX app's GUI is displayed is known as the stage and is an instance of class *Stage* (package `javafx.stage`).

- **Scene**

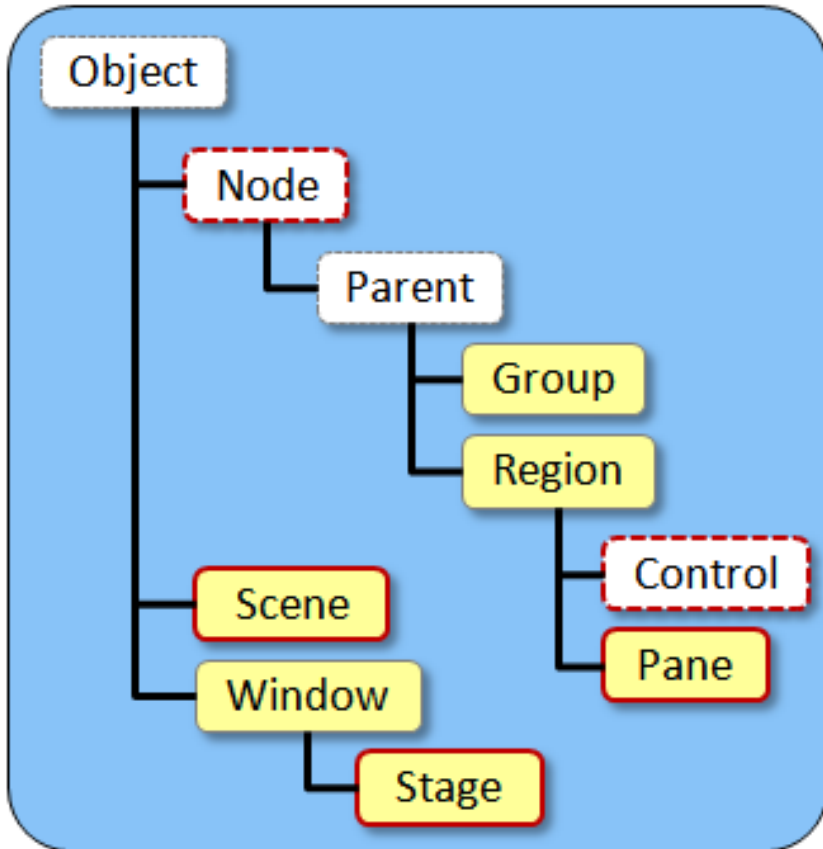
- The stage contains one active scene that defines the GUI as a scene graph
- A tree data structure of an app's visual elements, such as GUI controls, shapes, images, video, text and more. The scene is an instance of class *Scene* (package `javafx.scene`).

- **Window**

- A window component is an object with a visual representation that is placed on a window and usually allows the user to interact with it.
  - In JavaFX, typical window components (e.g., buttons, textFields, lists) are **Control** objects ... and they represent the various components of the window.
  - Components are often grouped together, much like adding elements to an array.

- **Node**

- Each visual element in the scene graph is a node.
- An instance of a subclass of **Node** (package `javafx.scene`)

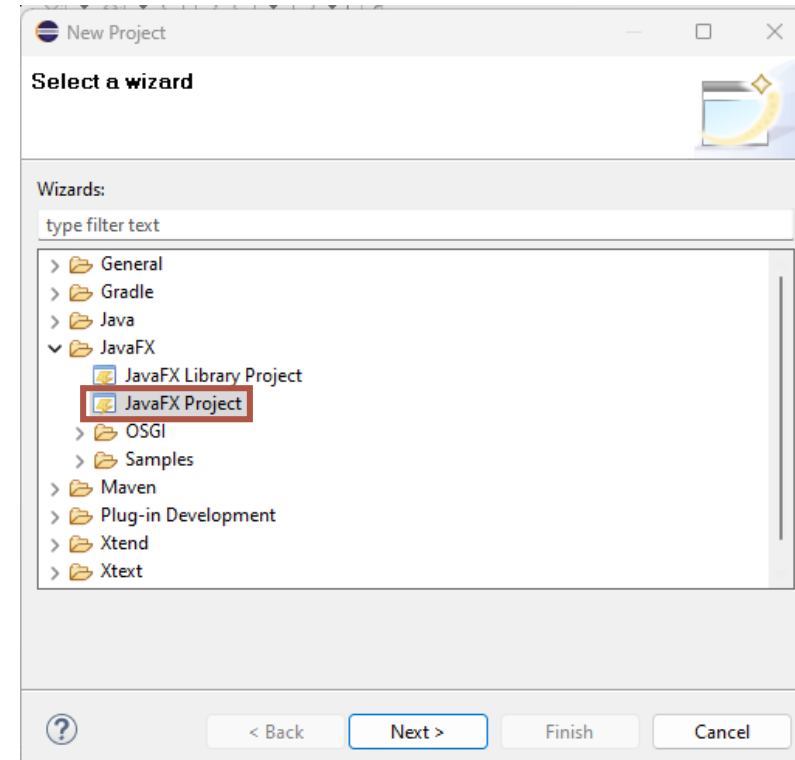


# Components of JavaFX

- **Container**
  - Is an object that contains components and/or other containers.
  - The most easily understood container in JavaFX is the **Pane** class.
  - It is the base class for its various subclasses that are used to automatically manage the layout of the window components.
  - Its variety of subclasses allow the components to be resized and repositioned automatically, while maintaining a particular arrangement/layout on the window.
- **Layout Containers**
  - Nodes that have children are typically **layout containers** that arrange their child nodes in the scene.
- **Controls**
  - Are GUI components, such as Labels that display text, TextFields that enable a program to receive text typed by the user, Buttons that initiate actions and more.
- **Control class**
  - An FXML GUI's event handlers are define in a so-called controller class.
- **Event handler**
  - Is a method that responds to a user interaction.

# Welcome Application

- Let us build our first Welcome to JavaFx example using eclipse.
- Create a new JavaFx project using eclipse.
- File → New → Project



# Welcome Application

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: WelcomeToFx

☒ Use default location

Location: C:\Users\Mahboob Ali\eclipse-workspace\Winter2022\JavaFxInstallation\WelcomeToFx [Browse...](#)

JRE

☒ Use an execution environment JRE: JavaSE-17

☐ Use a project specific JRE: jdk-19

☐ Use default JRE 'jdk-19' and workspace compiler preferences [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets [New...](#)

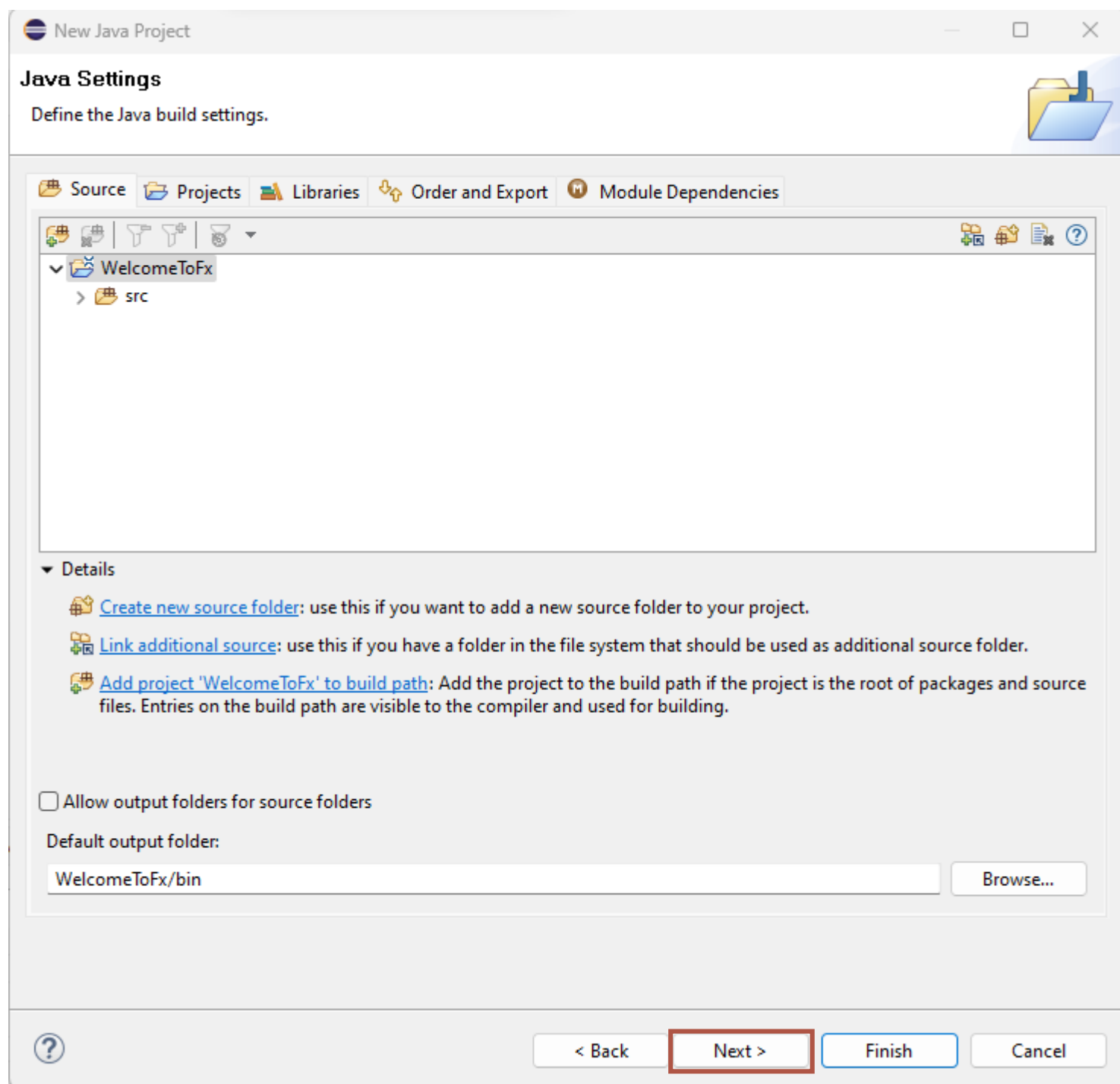
Working sets: [Select...](#)

Module

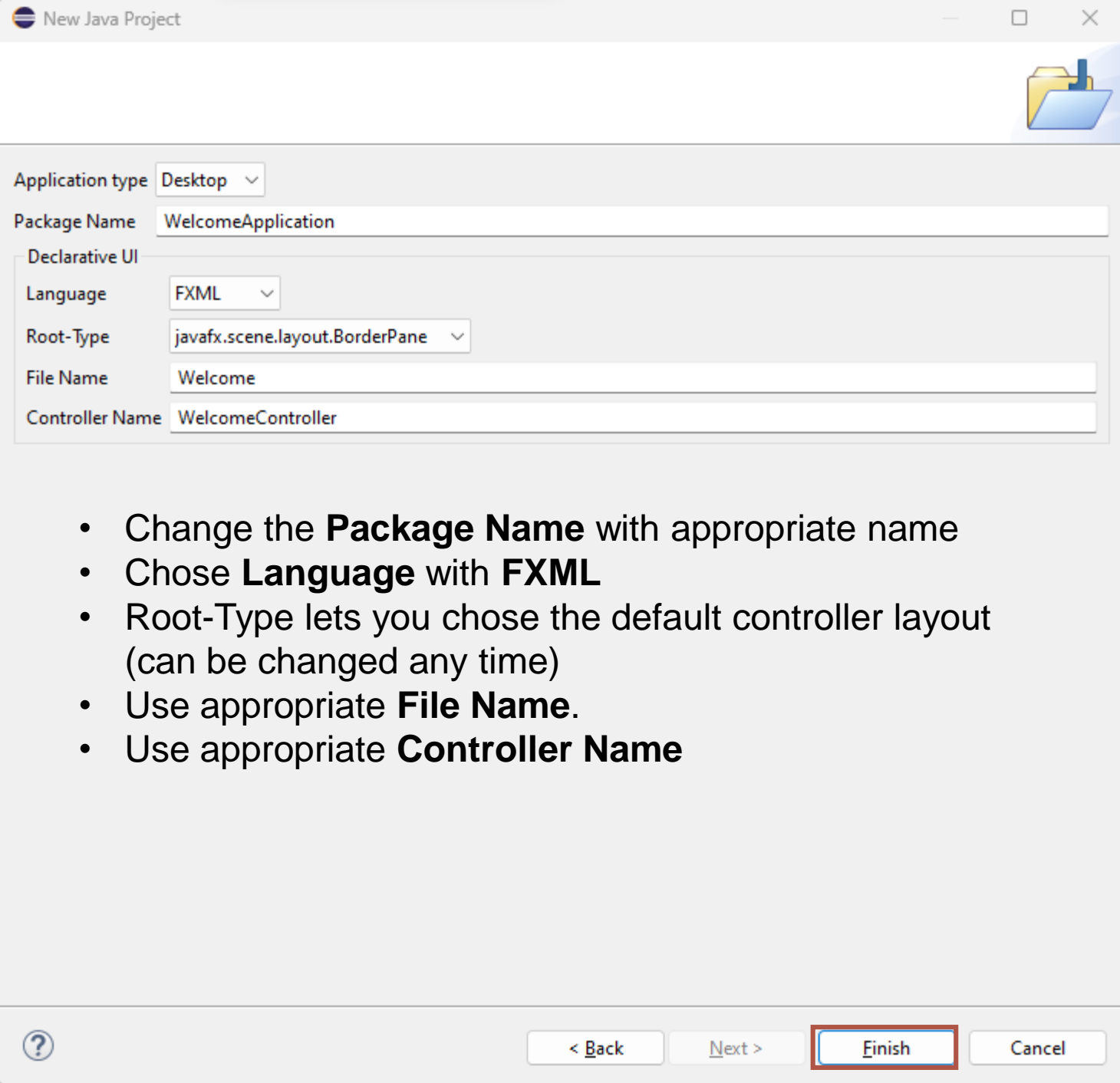
☒ Create module-info.java file

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

# Welcome Application



# Welcome Application



New Java Project

Application type: Desktop

Package Name: WelcomeApplication

Declarative UI

Language: FXML

Root-Type: javafx.scene.layout.BorderPane

File Name: Welcome

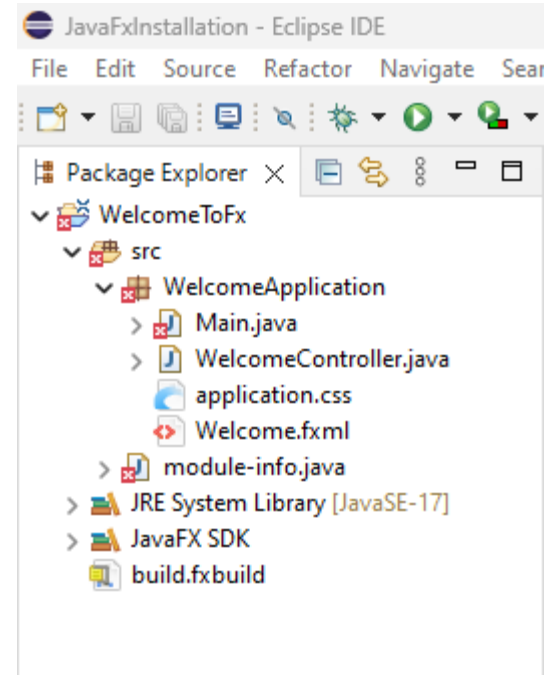
Controller Name: WelcomeController

- Change the **Package Name** with appropriate name
- Chose **Language** with **FXML**
- Root-Type lets you chose the default controller layout (can be changed any time)
- Use appropriate **File Name**.
- Use appropriate **Controller Name**

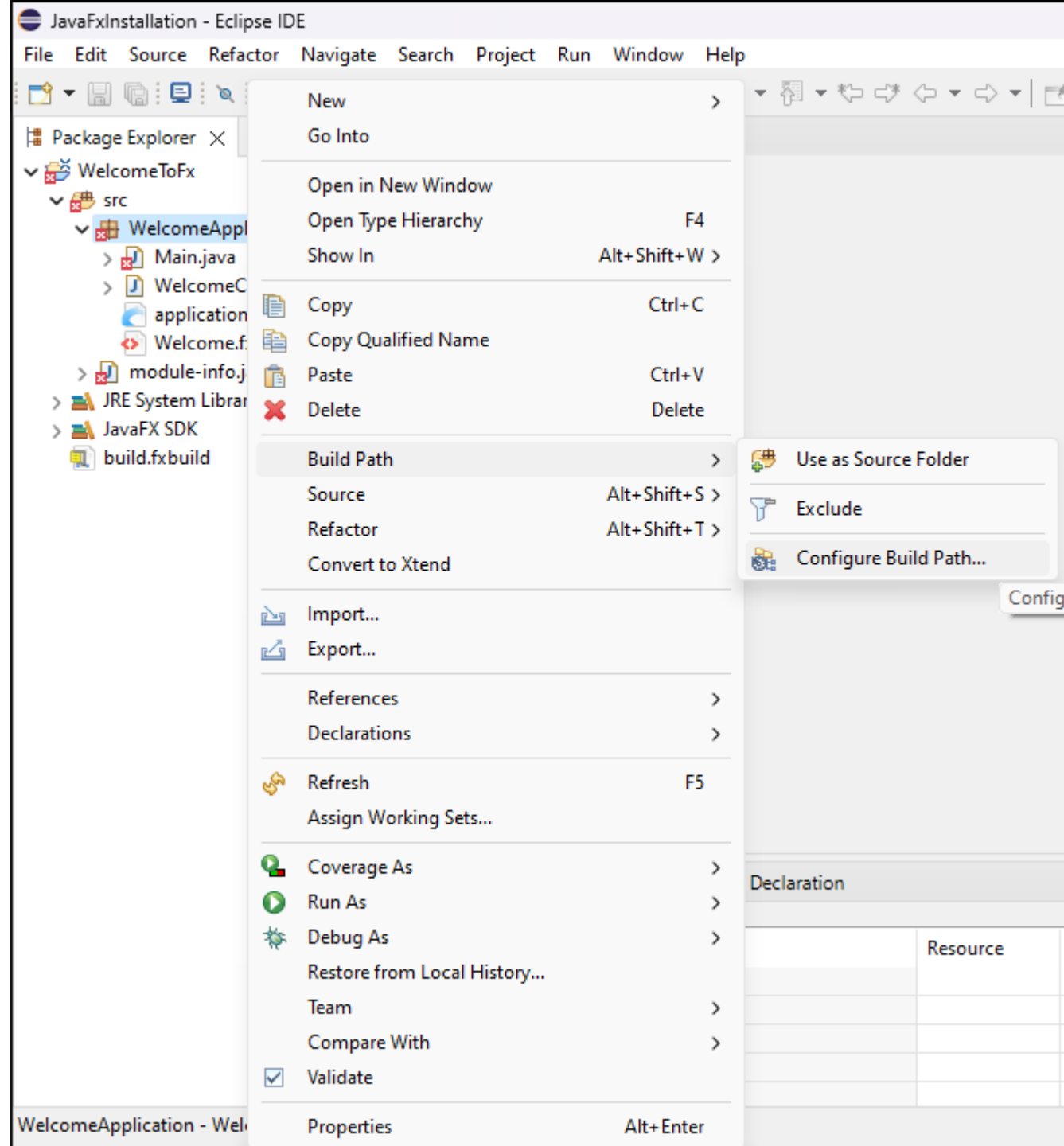
< Back Next > Finish Cancel

# Welcome Application

- Removing the error from the project.
- Need to add the Fx Library (Use the Environment Setup from the lab to create the Fx library)
- `Welcome.fxml` – This file contains the FXML markup for GUI.
- `Main.java` – This is the main class that creates the GUI from the FXML file and displays the GUI in a window
- `WelcomeController.java` – This is the class in which you'd define the GUI's event handlers/ triggers that allow the app to respond to user interaction with the GUI

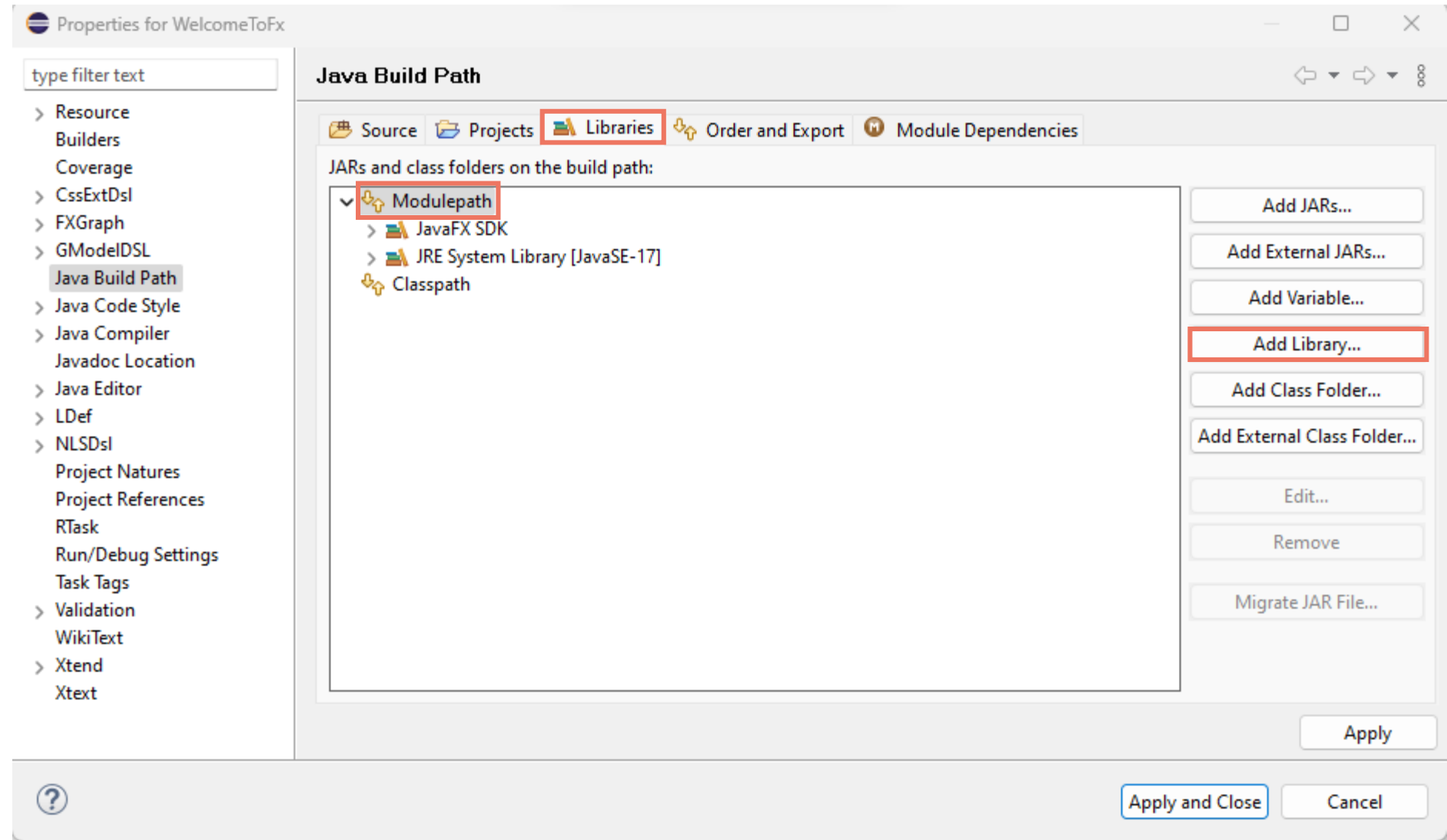


# Welcome Application

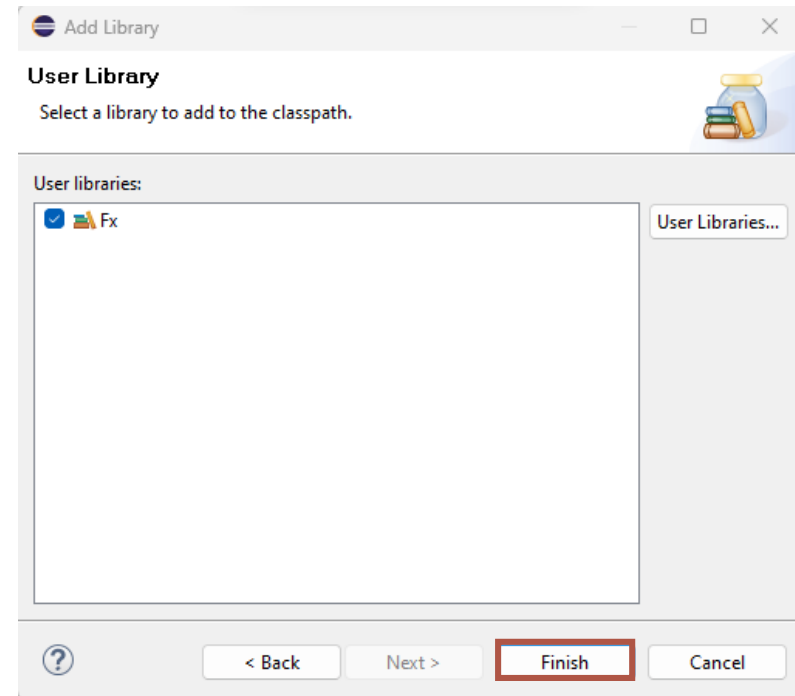
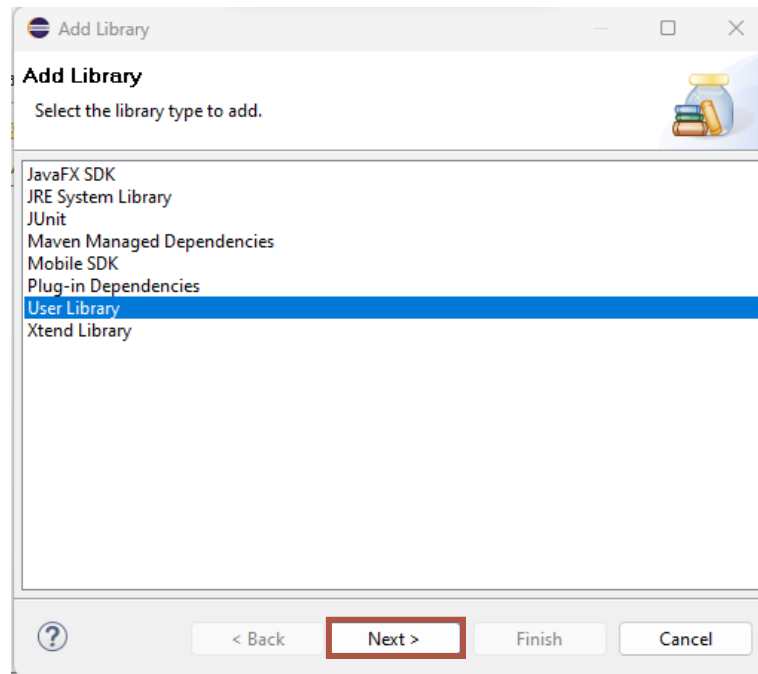




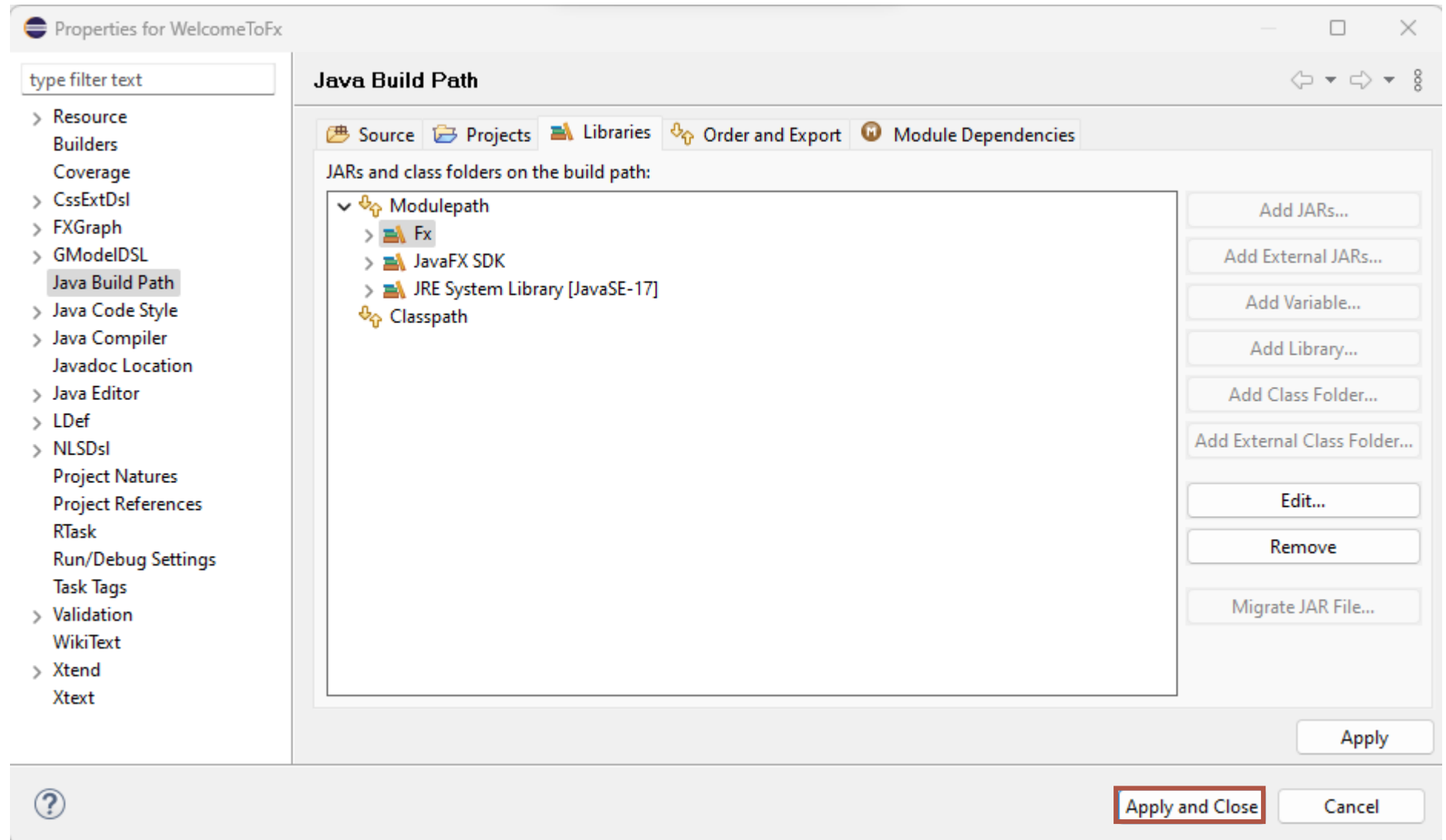
# Welcome Application



# Welcome Application

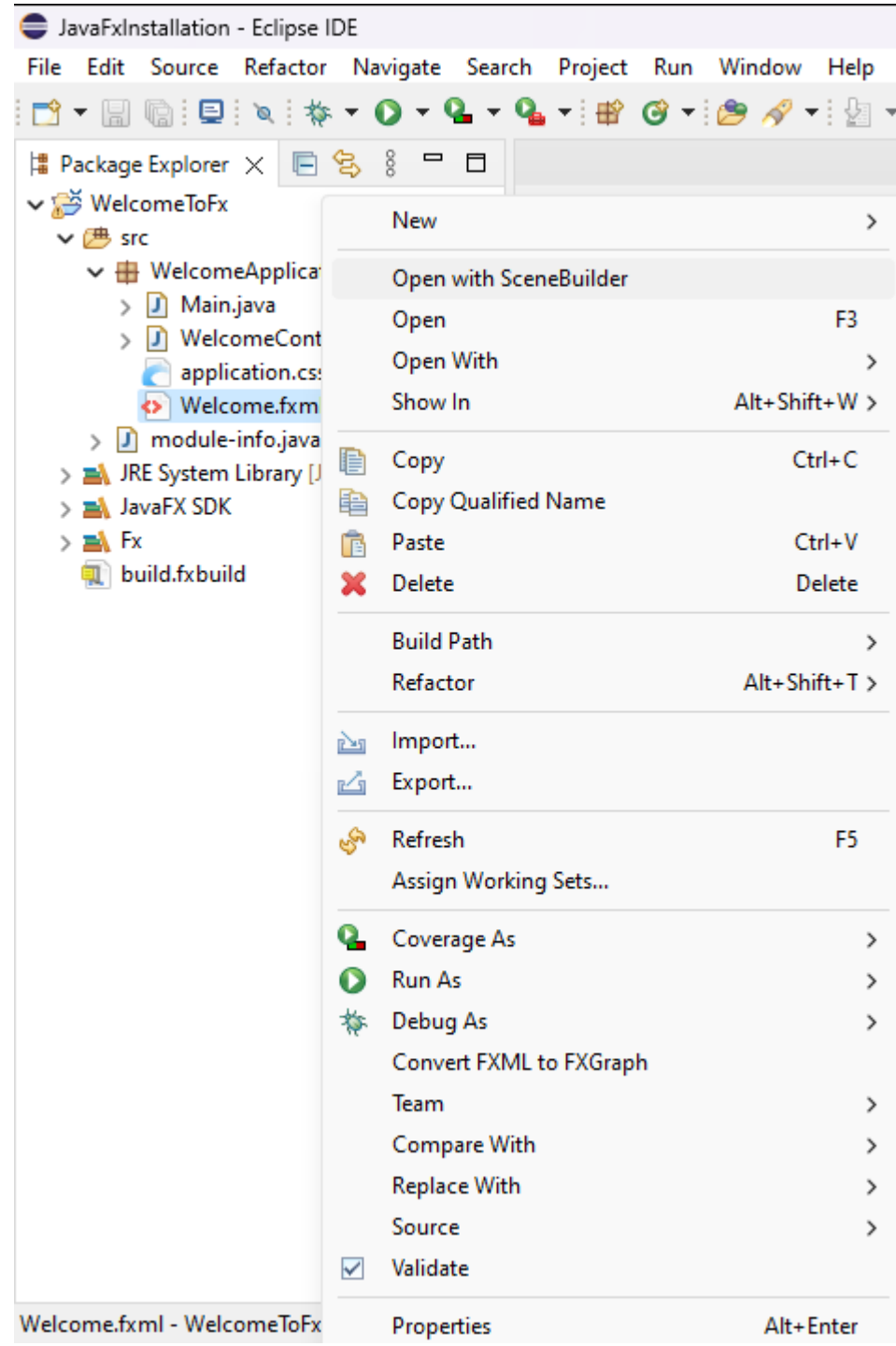


# Welcome Application

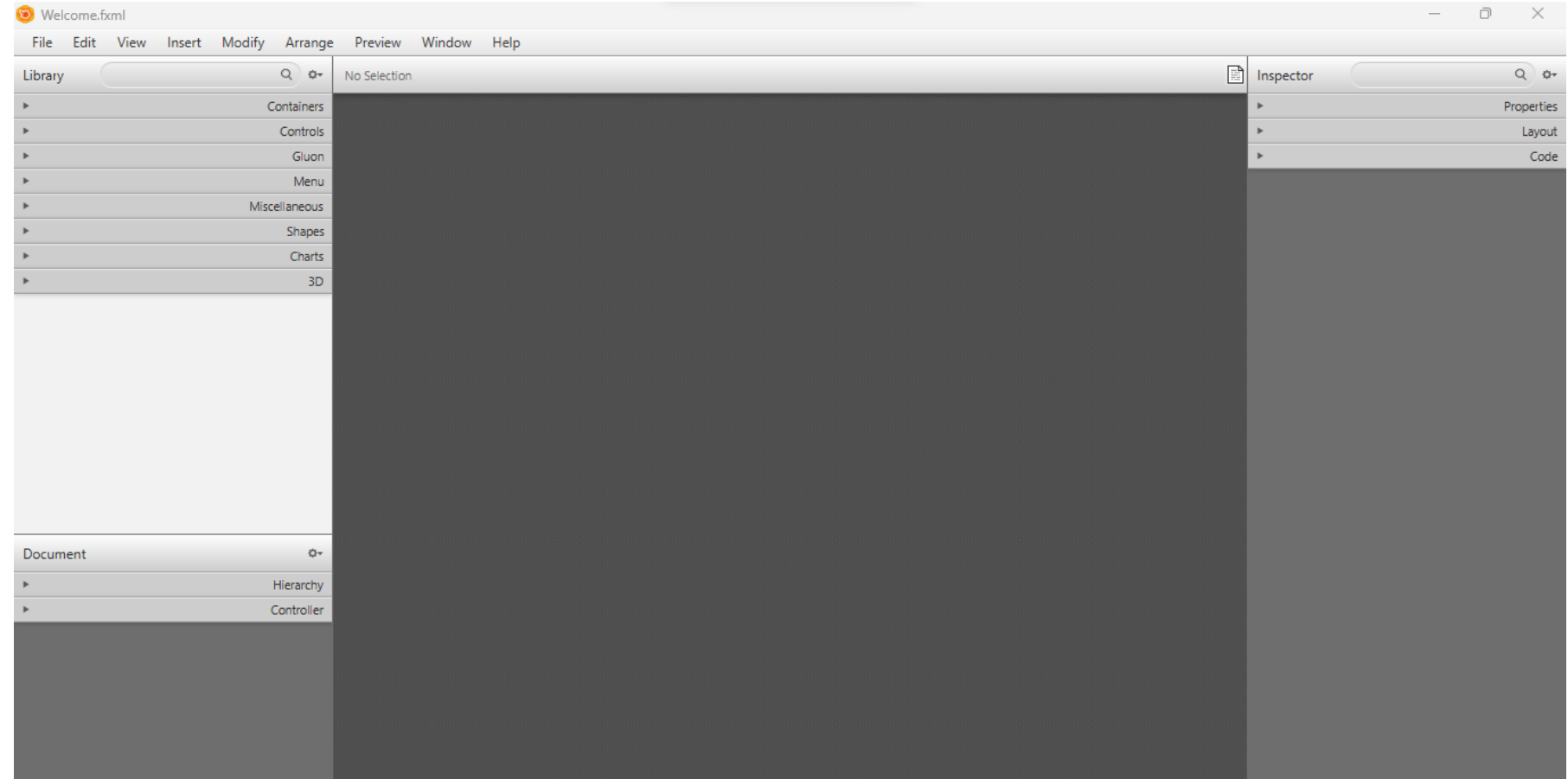


# Welcome Application

- Right Click on your Welcome.fxml and open it in Scene Builder

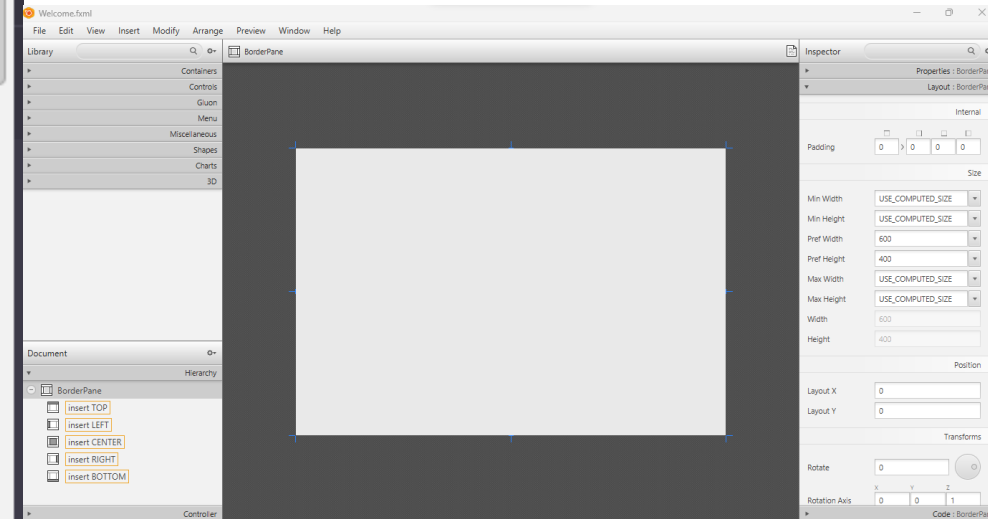
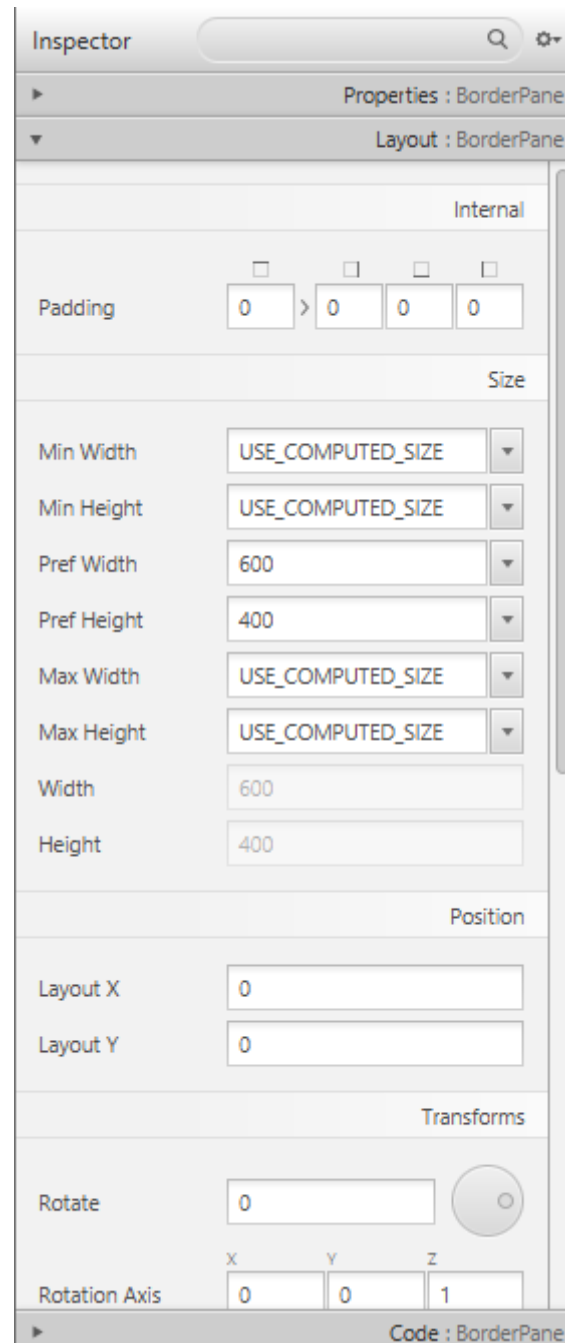


# Welcome Application



# Welcome Application

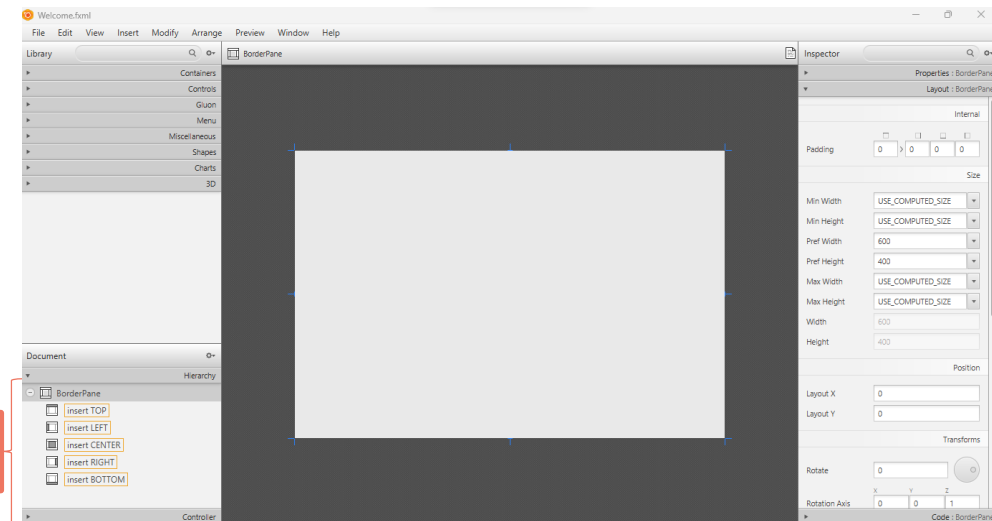
- We already started with a default **BorderPane** type controller layout.
- If you change the **Pref Width** to 600 and **Pref Height** to 400 you will see the initial design window of the border layout



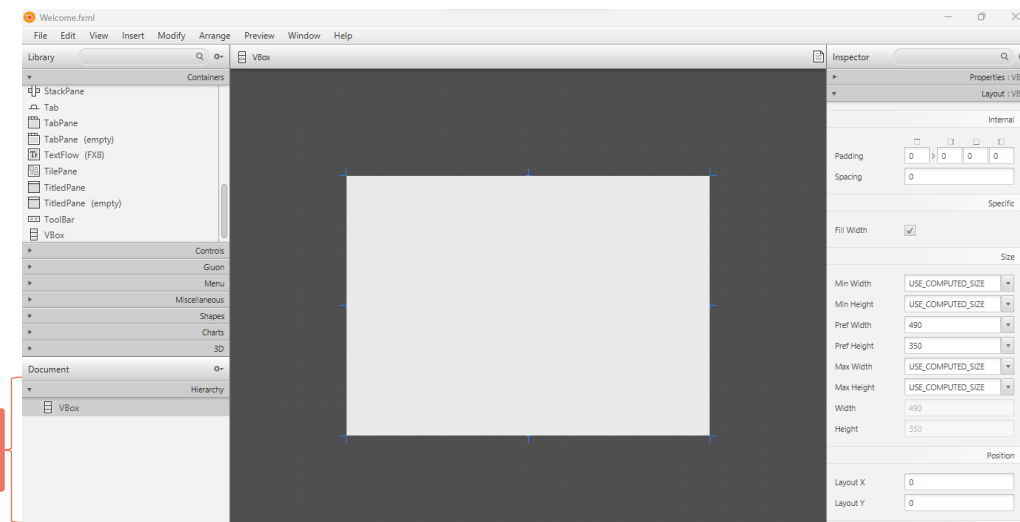
# Welcome Application

- For this example, we would like to work with a `Label` and an `ImageView` in a **VBox layout container** (package `javafx.scene.layout`)
- **Adding a VBox to the Default Layout.** Drag a VBox from the **Library** window's **Containers** section onto the default `BorderPane` in Scene Builder's content panel.
- **Making the VBox the Root Layout.** Select `Edit > Trim Document to Selection` to make the `VBox` the root layout and remove the `BorderPane`.

Before

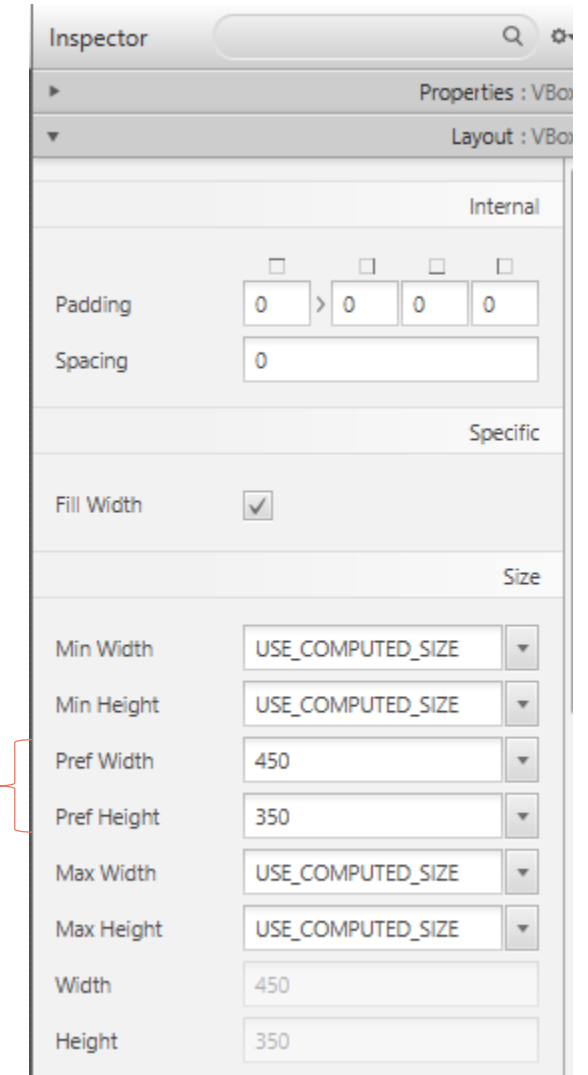
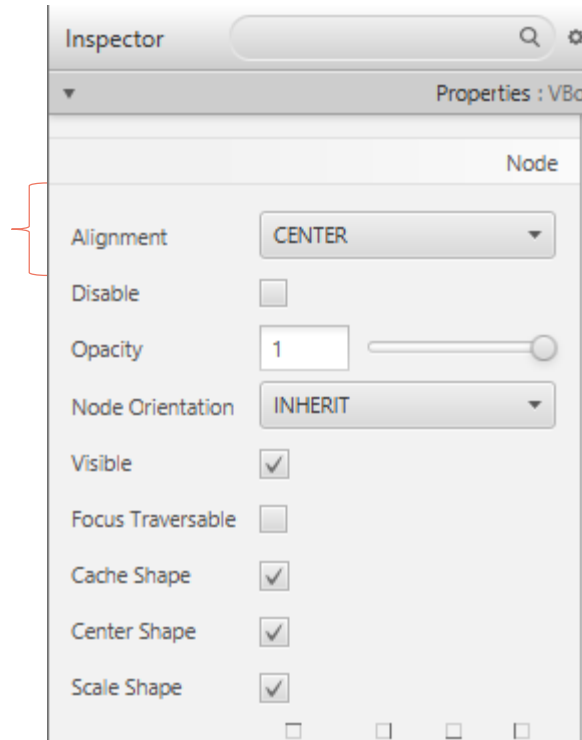


After



# Welcome Application

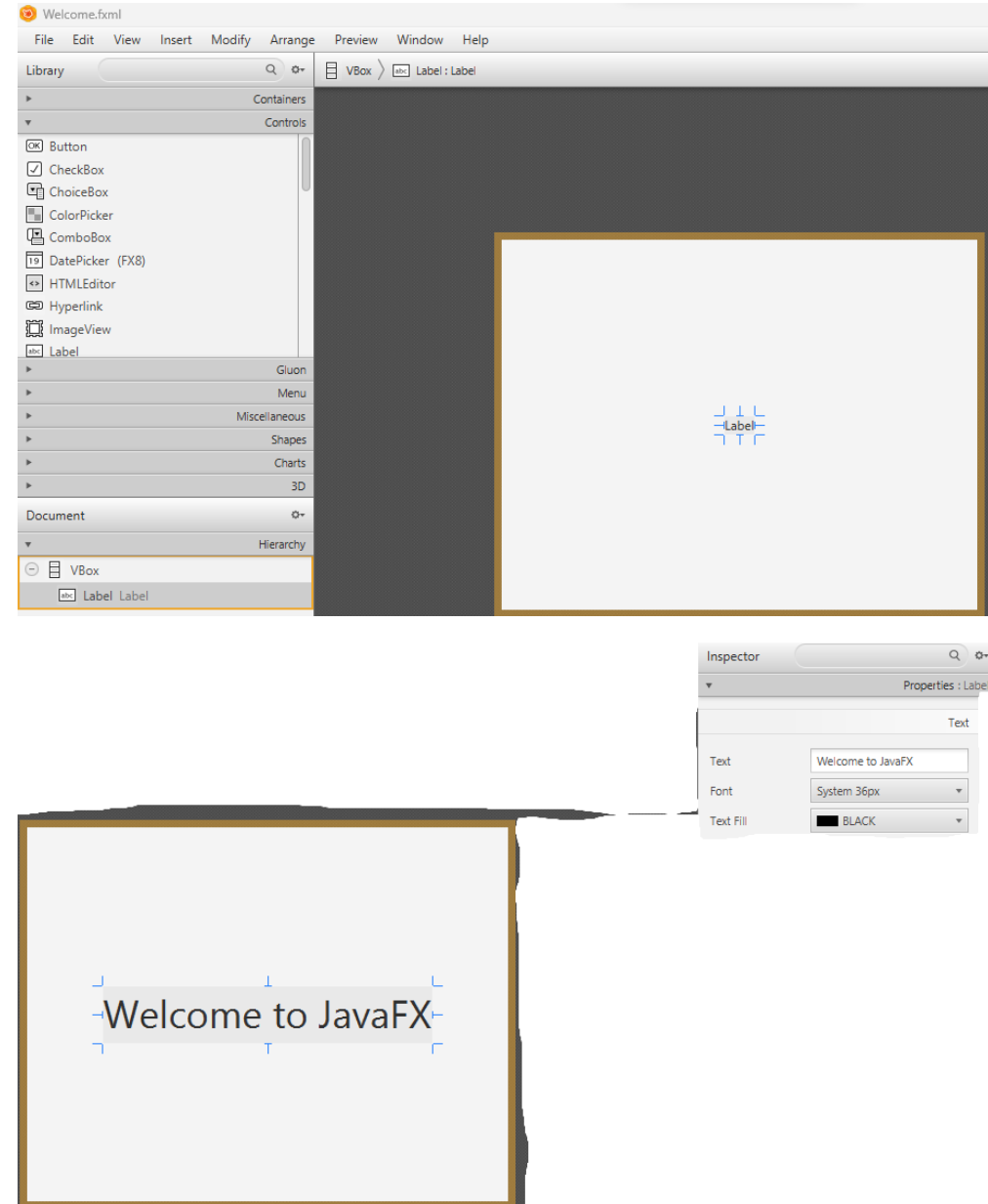
- **Specifying the VBox's Alignment.** A VBox's alignment determines the layout positioning of the VBox's children. Select the VBox, then in the Inspector's Properties section, set the Alignment property to CENTER.
- **Specifying the VBox's Preferred Size.** The preferred size (width and height) of the scene graph's root node is used by the scene to determine its window size when the app begins executing. Select the VBox, then in the Inspector's Layout section, set the Pref Width property to 450 and the Pref Height property to 300.





# Welcome Application

- **Adding a Label to the VBox.** Drag a Label from the Library window's Controls section onto the VBox. The Label is automatically centered in the VBox.
- **Changing the Label's text.** You can set a Label's text either by double clicking it and typing the text, or by selecting the Label and setting its Text property in the Inspector's Properties section. Set the Label's text to "Welcome to JavaFX!".
- **Changing the Label's font.**
  - Set the Label to display in a large bold font.
  - Select the Label, then in the Inspector's Preferences section, click the value to the right of the **Font** property.
  - Set the style property to **Bold** and the **size** property to 36.



# Welcome Application

- **Adding an ImageView to the VBox.** Drag an *ImageView* from the Library window's Controls section onto the VBox. The ImageView is automatically placed below the Label and is also automatically centered in the VBox.
- **Setting the ImageView's image.** To set the image to display, select the ImageView and click the ellipsis (...) button to the right of the Image property in the Inspector's Properties section.
- **Changing the ImageView's size.** We'd like to display the image in its original size. To do so, you must delete the default values for the ImageView's Fit Width and Fit Height properties.



# Welcome Application

- You can run an application multiple ways.
- Right click on the Main.java and go to Run As and then Click Java Application.
- Source Code for the example  
[WelcomeToFx](#)



# Basic Structure of JavaFx – Example

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class JavaFXBasic extends Application {
7      @Override // Override the start method in the Application class
8      public void start(Stage primaryStage) {
9          // Create a button and place it in the scene
10         Button btOK = new Button("OK");
11         Scene scene = new Scene(btOK, 200, 250);
12         primaryStage.setTitle("JavaFX Basic Structure"); // Set the stage title
13         primaryStage.setScene(scene); // Place the scene in the stage
14         primaryStage.show(); // Display the stage
15     }
16
17     /**
18      * The main method is only needed for the IDE with limited
19      * JavaFX support. Not needed for running from the command line.
20      */
21     public static void main(String[] args) {
22         launch(args);
23     }
24 }
```

# Display a simple shape in JavaFx – Example

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.stage.Stage;
7
8  public class ShowCircle extends Application {
9      @Override
10     public void start(Stage primaryStage) {
11         // Create a circle and set its properties
12         Circle circle = new Circle();
13         circle.setCenterX(100);
14         circle.setCenterY(100);
15         circle.setRadius(50);
16         circle.setStroke(Color.BLACK); // Set circle stroke color
17         circle.setFill(Color.WHITE);
18
19         // Create a pane to hold the circle
20         Pane pane = new Pane();
21         pane.getChildren().add(circle);
22
23         // Create a scene and place it in the stage
24         Scene scene = new Scene(pane, 200, 200);
25         primaryStage.setTitle("ShowCircle");
26         primaryStage.setScene(scene);
27         primaryStage.show();
28     }
```

```
29
30     /**
31      * The main method is only needed for the IDE with limited
32      * JavaFX support. Not needed for running from the command line.
33      */
34     public static void main(String[] args) {
35         launch(args);
36     }
37 }
```