

GUIs with JavaFX and Scene Builder

School of Computing
Clemson University
Clemson, SC 29634 USA

CPSC 2150: Wednesday, April 19, 2023

Slide Sources:

- 1 Java How to Program, 11/e
- 2 ZyBook - Introduction to Programming in Java

Last Class

- 1 Programs with GUIs
- 2 Java Swing
 - ▶ [JFrame](#)
 - ▶ [ActionListener](#)
 - ▶ [ActionEvent](#)
- 3 Java Swing Widgets
 - ▶ Adding Widgets to Our Screen
 - ▶ Instance Variables
 - ▶ Layout Managers
- 4 Fundamentals: [DemoGUI](#)
- 5 Constructor for View Class
- 6 Recap: MVC Pattern
- 7 Threads

Homework Assignment

- 1 Programming assignment 5 is posted on your lab Canvas page.

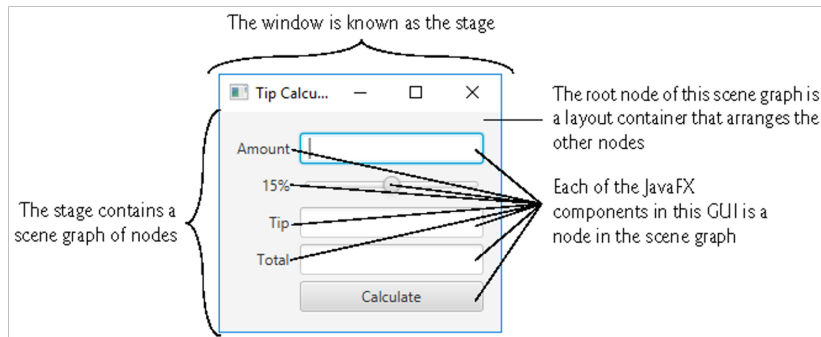
Comparison Between Java Swing and JavaFX

- ▶ <https://www.educba.com/java-swing-vs-java-fx/>

JavaFX App Window Structure

- ▶ A JavaFX app window consists of several parts
- ▶ **Controls** are GUI components, such as `Labels` that display text, `TextFields` that enable a program to receive user input, `Buttons` that users click to initiate actions, and more.
- ▶ 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`).
- ▶ 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.
- ▶ The scene is an instance of class `Scene` (`package javafx.scene`).

JavaFX App Window Structure



JavaFX App Window Structure

- ▶ Each visual element in the scene graph is a **node** – an instance of a subclass of `Node` (package `javafx.scene`), which defines common attributes and behaviors for all nodes
- ▶ With the exception of the first node in the scene graph – the **root node** – each node in the scene graph has one parent.
- ▶ Nodes can have transforms (e.g., moving, rotating and scaling), opacity (whether a node is transparent, partially transparent or opaque), effects (e.g., drop shadows, blurs, reflection and lighting) and more.

JavaFX App Window Structure

- ▶ Nodes with children are typically **layout containers** that arrange their child nodes in the scene.
- ▶ The nodes arranged in a layout container are a combination of controls and, in more complex GUIs, possibly other layout containers.
- ▶ When the user interacts with a control, the control generates an event.
- ▶ Programs can respond to these events – known as event handling – to specify what should happen when each user interaction occurs.
- ▶ An **event handler** is a method that responds to a user interaction. An FXML GUI's event handlers are defined in a **controller class**.

Class Application

- ▶ The class responsible for launching a JavaFX app is a subclass of `Application` (`package javafx.application`).
- ▶ When the subclass's `main` method is called:
 - ▶ Method `main` calls class `Application`'s `static` `launch` method to begin executing the app.
 - ▶ The `launch` method, in turn, causes the JavaFX runtime to create an object of the `Application` subclass and call its `start` method.
 - ▶ The `Application` subclass's `start` method creates the GUI, attaches it to a `Scene` and places it on the `Stage` that `start` receives as an argument.

Arranging JavaFX Components with a `GridPane`

- ▶ A `GridPane` (`package javafx.scene.layout`) arranges JavaFX components into columns and rows in a rectangular grid.
- ▶ Each cell in a `GridPane` can be empty or can hold one or more JavaFX components, including layout containers that arrange other controls.
- ▶ Each component in a `GridPane` can span multiple columns or rows, though we did not use that capability in this GUI.
- ▶ When you drag a `GridPane` onto Scene Builder's content panel, Scene Builder creates the `GridPane` with two columns and three rows by default.

Technologies Overview

Class Application

- ▶ GUIs are event driven
 - ▶ When the user interacts with a GUI component, the interaction – known as an event – drives the program to perform a task.
- ▶ The code that performs a task in response to an event is called an event handler.
- ▶ For certain events you can link a control to its event-handling method by using the **Code** section of Scene Builder's **Inspector** window.
- ▶ You'll create the event handler entirely in code.
- ▶ JavaFX applications in which the GUI is implemented as FXML adhere to the Model-View-Controller (MVC) design pattern, which separates an app's data (contained in the model) from the app's GUI (the view) and the app's processing logic (the controller).

Technologies Overview

- ▶ The controller implements logic for processing user inputs.
- ▶ The view presents the data stored in the model.
- ▶ When a user provides input, the controller modifies the model with the given input.
- ▶ When the model changes, the controller updates the view to present the changed data.
- ▶ In a simple app, the model and controller are often combined into a single class.
- ▶ In a JavaFX FXML app, you define the app's event handlers in a controller class.

Technologies Overview

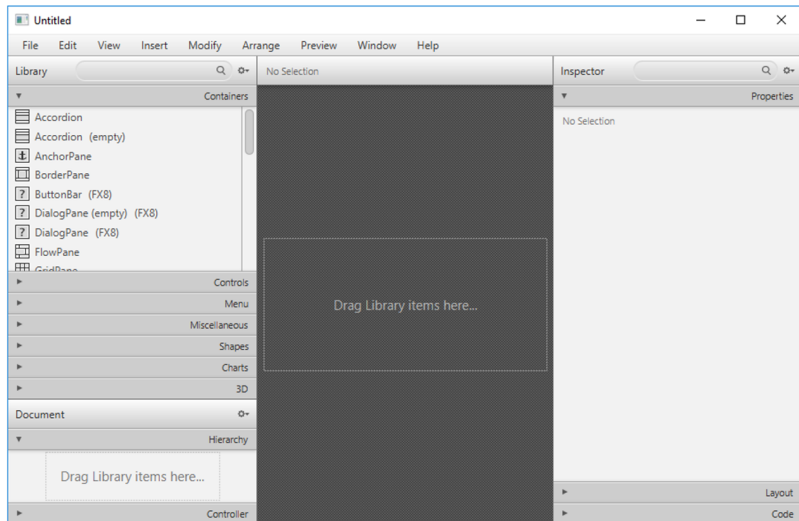
- ▶ GUI's can be built programmatically as well as done with Swing
- ▶ Or they can be built graphically using Scene Builder

```
14 public class SalaryApp extends Application {
15     private Label wageLabel;    // Label for hourly salary
16     private Label sallLabel;    // Label for yearly salary
17     private TextField salField; // Displays hourly salary
18     private TextField wageField; // Displays yearly salary
19     private Button calcButton;  // Triggers salary calculation
20
21     @Override
22     public void start(Stage applicationStage) {
23         Scene scene = null;      // Scene contains all content
24         GridPane gridPane = null; // Positions components within scene
25
26         gridPane = new GridPane(); // Create an empty pane
27         scene = new Scene(gridPane); // Create scene containing the grid pane
28
29         // Set hourly and yearly salary
30         wageLabel = new Label("Hourly wage:");
31         sallLabel = new Label("Yearly salary:");
32
33         wageField = new TextField();
34         wageField.setPrefColumnCount(15);
35         wageField.setEditable(true);
36         wageField.setText("");
```

JavaFX Scene Builder

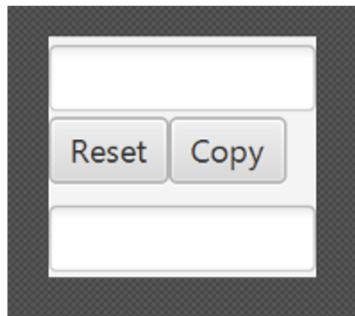
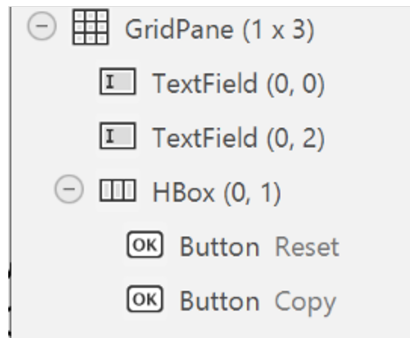
- ▶ **JavaFX Scene Builder** is a standalone JavaFX GUI visual layout tool that can also be used with various IDEs.
- ▶ JavaFX Scene Builder enables you to create GUIs by dragging and dropping GUI components from Scene Builder's library onto a design area, then modifying and styling the GUI – all without writing any code.
- ▶ JavaFX Scene Builder generates FXML (FX Markup Language) – an XML vocabulary for defining and arranging JavaFX GUI controls without writing any Java code.
- ▶ The FXML code is separate from the program logic that's defined in Java source code – 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 Scene Builder



- ▶ If a control or layout will be manipulated programmatically in the controller class, you must provide a name for that control or layout. Each object's name is specified via its `fx:id` property. You can set this property's value by selecting a component in your scene, then expanding the **Inspector** window's **Code** section – the `fx:id` property appears at the top.

Building the Demo GUI



Building the Demo GUI

DemoGuiApp

```
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  /**
10   * Demo GUI using a version of the Model-View-Controller (MVC) design pattern.
11   *
12   * @author Bruce W. Weide
13   *
14   */
15  public class DemoGuiApp extends Application {
16      @Override
17      public void start(Stage stage) throws Exception {
18          Parent root =
19              FXMLLoader.load(getClass().getResource("DemoGUI.fxml"));
20
21          Scene scene = new Scene(root); // attach scene graph to scene
22          stage.setTitle("DemoGUI"); // displayed in window's title bar
23          stage.setScene(scene); // attach scene to stage
24          stage.show(); // display the stage
25      }
```

Building the Demo GUI

DemoGUI.fmx1

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>

<GridPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" xmlns=
"http://javafx.com/javafx/8.0.171" xmlns:fx="http://javafx.com/fxml/1" fx:controller="DemoController">
  <columnConstraints>
    <ColumnConstraints halignment="CENTER" hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
  </columnConstraints>
  <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>
  <children>
    <TextField fx:id="myInputField" />
    <TextField fx:id="myOutputField" GridPane.rowIndex="2" />
    <HBox prefHeight="100.0" prefWidth="200.0" GridPane.rowIndex="1">
      <children>
        <Button fx:id="myResetButton" mnemonicParsing="false" onAction="#ResetCallback" text="Reset" />
        <Button fx:id="myCopyButton" mnemonicParsing="false" onAction="#CopyCallback" text="Copy" />
      </children>
    </HBox>
  </children>
</GridPane>
```

Building the Demo GUI

DemoGuiApp

- ▶ Lines 3-7 show the imports
- ▶ Method `start` (lines 18-24) creates the GUI, attaches it to a `Scene` and places it on the `Stage` that method `start` receives as an argument.
- ▶ Lines 18-19 use class `FXMLLoader`'s `static` method `load` to create the GUI's scene graph. This method:
 - ▶ Returns a `Parent` (`package javafx.scene`) reference to the scene graph's root node – the GUI's `GridPane` in this app.
 - ▶ Creates an object of the `DemoController` class that we specified in the FXML file.
 - ▶ Initializes the controller's instance variables for the components that are manipulated programmatically.
 - ▶ Registers the event handlers specified in the FXML to the appropriate controls.
 - ▶ Enables the controls to call the corresponding methods when the user interacts with the app.

Building the Demo GUI

DemoGuiApp

- ▶ To display the GUI, you must attach it to a **Scene**, then attach the **Scene** to the **Stage** that method **start** receives as an argument.
- ▶ To attach the GUI to a **Scene**, line 21 creates a **Scene**, passing root (the scene graph's root node) as an argument to the constructor.
 - ▶ By default, the **Scene**'s size is determined by the size of the scene graph's root node.
- ▶ Line 22 uses **Stage** method **setTitle** to specify the text that appears in the **Stage** window's title bar.
- ▶ Line 23 calls **Stage** method **setScene** to place the **Scene** onto the **Stage**.
- ▶ Line 24 calls **Stage** method **show** to display the **Stage** window.

Building the Demo GUI

DemoController Class

Lines 2-9 shows `DemoController` imports

```
2  import java.net.URL;
3  import java.util.ResourceBundle;
4
5  import javafx.event.ActionEvent;
6  import javafx.fxml.FXML;
7  import javafx.scene.control.Button;
8  import javafx.scene.control.TextField;
9  import javafx.scene.*;
```

Building the Demo GUI

DemoController Class

- ▶ The `@FXML` annotation preceding an instance variable indicates that the variable's name can be used in the FXML file that describes the app's GUI.
- ▶ The variable names that you specify in the controller class must precisely match the `fx:id` values you specified when building the GUI.
- ▶ When the `FXMLLoader` loads an FXML file to create a GUI, it also initializes each of the controller's instance variables that are declared with `@FXML` to ensure that they refer to the corresponding GUI components in the FXML file.

```
15 @FXML
16 private ResourceBundle resources;
17
18 @FXML
19 private URL location;
20
21 @FXML
22 private TextField myInputField;
23
24 @FXML
25 private TextField myOutputField;
26
27 @FXML
28 private Button myResetButton;
29
30 @FXML
31 private Button myCopyButton;
```

Building the Demo GUI

DemoController Class

- ▶ The `@FXML` annotation preceding a method indicates that the method can be used to specify a control's event handler in the FXML file that describes the app's GUI.

```
33  @FXML
34  void CopyCallback(ActionEvent event) {
35      model.setInput(myInputField.getText());
36      model.setOutput(myInputField.getText());
37      /*
38       * Update view to reflect changes in model
39       */
40      myOutputField.setText(model.getCounter()+" : "+model.output());
41  }
42
43  @FXML
44  void ResetCallback(ActionEvent event) {
45      model.setInput("");
46      model.setOutput("");
47      model.resetCounter();
48      /*
49       * Update view to reflect changes in model
50       */
51      myOutputField.setText(model.output());
52      myInputField.setText(model.input());
53  }
54  }
```

Building the Demo GUI

DemoController Class

- ▶ When the FXMLLoader loads `DemoGUI.fxml` to create the GUI, it creates and registers an event handler for the Reset & Copy Buttons' `ActionEvent`.
- ▶ The event handler for this event must implement interface `EventHandler<ActionEvent>`
- ▶ This interface contains a `handle` method that returns `void` and receives an `ActionEvent` parameter.
- ▶ This method's body, in turn, calls method `copyButtonCallback` and `resetButtonCallback` when the user clicks the button.
- ▶ `FXMLLoader` performs similar tasks for every event listener you specify via the Scene Builder **Inspector** window's **Code** section.

Building the Demo GUI

DemoController Class

- ▶ When the `FXMLLoader` creates an object of a controller class, it determines whether the class contains an `initialize` method with no parameters and, if so, calls that method to initialize the controller.
- ▶ This method can be used to configure the controller before the GUI is displayed.

```
69         @FXML
70         void initialize() {
71
72         }
73     }
```

JavaFX main vs. Java Swing main

```
36     public static void main(String[] args) {  
37  
38         // create a TipCalculator object and call its start method  
39         launch(args);  
40  
41         // the view is created by the start method above  
42  
43         // the controller is created implicitly by the view  
44         // the observers are attached directly to the screen objects in the controller  
45  
46         // the model is created explicitly in the controller
```

```
public static void main(String[] args) {  
    DemoModel model = new DemoModel();  
    DemoView view = new DemoView();  
    DemoController controller = new DemoController(model, view);  
  
    view.registerObserver(controller);  
}
```

Additional JavaFX Capabilities

JavaFX Resources and JavaFX in the Real World

- ▶ Visit the following website for a lengthy and growing list of JavaFX resources: <http://bit.ly/JavaFXResources>

Exploring further: JavaFX overview, tutorials, and references from Oracle's Java Documentation: <https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm>

Homework Assignment

- 1 Programming assignment 5 is posted on your lab Canvas page.

Summary

- 1 JavaFX App Window Structure
- 2 Technologies Overview
- 3 JavaFX Scene Builder
- 4 Building the Demo GUI