**Application Program Development: Event Handling**

**Instructor: Mahboob Ali**

**Learning Outcomes**

1. **Understanding of Events in JavaFX**: Learn what events are and how they work in JavaFX.
2. **Event Source, Target, and Types**: Identify the source, target, and types of events.
3. **Event Processing Mechanism**: Understand how events are processed.
4. **Event Filters and Handlers**: Learn how to use event filters and handlers.

**What is an Event?**

An event in JavaFX is an occurrence of interest, typically triggered by user interaction. Examples include clicking the mouse or pressing a key. Events are essential for making applications interactive.

**Key Definitions**

1. **Event**: An occurrence that happens in the program based on user interaction, such as pressing a key or clicking a mouse button.
2. **Event Source**: The component for which the event is generated (e.g., a Button).
3. **Event Handler**: A procedure that contains code to execute when a specific event occurs.

**Event Classes in JavaFX**

In JavaFX, all events are represented by distinct classes. There are many kinds of events, each with its own unique class. For example:

* **MouseEvent**: Represents mouse actions.
* **KeyEvent**: Represents keyboard actions.
* **ActionEvent**: Represents actions performed by buttons and other controls.

**Understanding Event Flow**

Events are generated when the user interacts with the user interface:

1. The user triggers an event by interacting with the UI (e.g., clicking a button).
2. The event is generated.
3. The appropriate event handler is called.
4. The event handling code modifies the model.
5. The user interface is updated to reflect changes in the model.

**Handling Events in JavaFX**

To handle events in JavaFX, follow these steps:

1. **Identify Event Types**: Determine which events you want to handle.
2. **Write Event Handlers**: Implement appropriate event handlers for each event.
3. **Register Event Handlers**: Connect event handlers to event sources.

**EventHandler Interface**

The **EventHandler** interface in JavaFX is used to handle events:

java

Copy code

* public interface EventHandler<T extends Event> extends EventListener {
* void handle(T event);
* }

**Example: Simple Button Press**

Let's consider an application that handles a simple button press.

**Explanation**

The application creates a button and connects an event handler to it. When the button is pressed, a message is printed to the console.

**Code**

java

Copy code

import javafx.application.Application;

import javafx.event.\*;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.Pane;

import javafx.stage.Stage;

public class SimpleEventTest extends Application {

public void start(Stage primaryStage) {

Pane aPane = new Pane();

Button aButton = new Button("Press Me");

aButton.relocate(100, 10);

aButton.setPrefSize(100, 30);

aPane.getChildren().add(aButton);

// Connect the event handler to the button

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

public void handle(ActionEvent event) {

System.out.println("That felt good");

}

});

primaryStage.setTitle("Making a Handler");

primaryStage.setScene(new Scene(aPane, 300, 50));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Example: Multiple Buttons**

Handling multiple buttons can be done using a single event handler.

**Explanation**

This application creates a keypad with multiple buttons, all sharing the same event handler. When a button is pressed, its label is printed to the console.

**Code**

java

Copy code

import javafx.application.Application;

import javafx.event.\*;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.Pane;

import javafx.stage.Stage;

public class MultipleButtonsApp extends Application {

Button[][] buttons; // This will store all the Buttons

public void start(Stage primaryStage) {

Pane aPane = new Pane();

buttons = new Button[4][3];

String[] buttonLabels = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "\*", "0", "#"};

for (int row = 0; row < 4; row++) {

for (int col = 0; col < 3; col++) {

buttons[row][col] = new Button(buttonLabels[row \* 3 + col]);

buttons[row][col].relocate(10 + col \* 70, 10 + row \* 70);

buttons[row][col].setPrefSize(65, 65);

buttons[row][col].setOnAction(new EventHandler<ActionEvent>() {

public void handle(ActionEvent actionEvent) {

System.out.println("Button " +

((Button) actionEvent.getSource()).getText() + " was pressed.");

}

});

aPane.getChildren().add(buttons[row][col]);

}

}

primaryStage.setTitle("Keypad");

primaryStage.setScene(new Scene(aPane, 225, 295));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Example: Calculator Application**

This example handles the event of pressing a "Compute" button to calculate the square and square root of a number.

**Explanation**

The application consists of text fields for input and output, and a button to perform the computation. When the "Compute" button is pressed, the entered number is squared and square-rooted, and the results are displayed in the respective fields.

**Code**

java

Copy code

import javafx.application.Application;

import javafx.event.\*;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.Pane;

import javafx.stage.Stage;

public class CalculatorApp extends Application {

// Text fields to hold the user data and the computed data

TextField valueField, squareField, rootField;

public void start(Stage primaryStage) {

Pane aPane = new Pane();

// Add the value label and text field

Label label = new Label("Value:");

label.relocate(10, 10);

label.setPrefSize(100, 30);

aPane.getChildren().add(label);

valueField = new TextField();

valueField.relocate(100, 10);

valueField.setPrefSize(150, 30);

aPane.getChildren().add(valueField);

// Add the compute button

Button computeButton = new Button("Compute");

computeButton.relocate(10, 50);

computeButton.setPrefSize(240, 30);

aPane.getChildren().add(computeButton);

// Add the square label and text field

label = new Label("Square:");

label.relocate(10, 100);

label.setPrefSize(100, 30);

aPane.getChildren().add(label);

squareField = new TextField();

squareField.relocate(100, 100);

squareField.setPrefSize(150, 30);

squareField.setEditable(false);

aPane.getChildren().add(squareField);

// Add the square root label and text field

label = new Label("Square Root:");

label.relocate(10, 150);

label.setPrefSize(100, 30);

aPane.getChildren().add(label);

rootField = new TextField();

rootField.relocate(100, 150);

rootField.setPrefSize(150, 30);

rootField.setEditable(false);

aPane.getChildren().add(rootField);

// Connect the event handlers to the buttons

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

public void handle(ActionEvent actionEvent) {

if (valueField.getText().length() > 0) {

float value = Float.parseFloat(valueField.getText());

squareField.setText("" + value \* value);

rootField.setText("" + Math.sqrt(value));

}

}

});

primaryStage.setTitle("Calculator");

primaryStage.setResizable(false);

primaryStage.setScene(new Scene(aPane, 248, 178));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Summary**

* **Events in JavaFX**: Understand what events are and how they work.
* **Event Source and Handler**: Identify the source and write event handlers.
* **Event Flow**: Learn the event flow from user interaction to UI update.
* **Examples**: Implement examples to handle button presses and create interactive applications.

By mastering event handling in JavaFX, you can create interactive and responsive applications.