**Lab Manual: Introduction to JavaFX - User Interface Components**

**Objective:**

The objective of this lab is to introduce students to JavaFX, the Java framework for building graphical user interfaces (GUIs). This manual will cover common JavaFX UI components such as labels, buttons, text fields, checkboxes, radio buttons, combo boxes, and images. Students will learn how to use these components in JavaFX applications and then create a full-fledged user interface for an application.

**1. Introduction to JavaFX:**

JavaFX is a modern framework for developing rich graphical user interfaces in Java. It provides a wide range of UI components and allows for the development of cross-platform applications. JavaFX applications are typically structured as follows:

1. **Main Application Class (Application)**: The entry point of the JavaFX application that extends the Application class and overrides the start() method to define the main UI components.
2. **FXML File**: A markup file used to define the user interface layout in an XML-like format. It helps separate UI design from application logic.
3. **Controller Class**: This class contains methods to handle user actions (like button clicks) and bind UI components to the backend logic.

**2. JavaFX UI Components Explained**

**A. Label**

A **Label** is a simple text element used to display static text. It is often used for headings, instructions, or displaying messages to users.

**Code Example:**

<Label fx:id="welcomeText" text="Welcome to JavaFX!" />

**B. Button**

A **Button** is an interactive element that performs an action when clicked. Buttons are used to trigger events such as submitting forms or switching between screens.

**Code Example:**

<Button text="Click Me!" onAction="#onButtonClick" />

In the controller:

@FXML

protected void onButtonClick() {

welcomeText.setText("Button Clicked!");

}

**C. TextField**

A **TextField** allows users to input a single line of text. It is commonly used for entering data such as usernames, search queries, or form fields.

**Code Example:**

<TextField fx:id="userNameField" promptText="Enter your name" />

In the controller:

@FXML

private TextField userNameField;

@FXML

protected void onGreetButtonClick() {

String userName = userNameField.getText();

welcomeText.setText("Hello, " + userName + "!");

}

**D. TextArea**

A **TextArea** is similar to a **TextField** but allows multi-line text input. It is commonly used for comments, descriptions, or other longer form entries.

**Code Example:**

<TextArea fx:id="commentsArea" promptText="Enter your comments here" />

**E. CheckBox**

A **CheckBox** allows the user to select or deselect an option. It is useful when the user can select multiple options from a list.

**Code Example:**

<CheckBox text="I agree to the terms and conditions" />

In the controller:

@FXML

private CheckBox agreeCheckBox;

@FXML

protected void onAgreeCheckBoxAction() {

if (agreeCheckBox.isSelected()) {

// Handle agreement logic

}

}

**F. RadioButton**

A **RadioButton** allows users to select one option from a set of mutually exclusive options. Multiple RadioButtons can be grouped together using a **ToggleGroup**.

**Code Example:**

<RadioButton text="Option 1" />

<RadioButton text="Option 2" />

<RadioButton text="Option 3" />

In the controller:

ToggleGroup group = new ToggleGroup();

radioButton1.setToggleGroup(group);

radioButton2.setToggleGroup(group);

radioButton3.setToggleGroup(group);

**G. ComboBox**

A **ComboBox** provides a drop-down list of items, allowing the user to select one item from the list.

**Code Example:**

<ComboBox fx:id="comboBox">

<items>

<FXCollections fx:factory="observableArrayList">

<String fx:value="Option 1"/>

<String fx:value="Option 2"/>

<String fx:value="Option 3"/>

</FXCollections>

</items>

</ComboBox>

In the controller:

@FXML

private ComboBox<String> comboBox;

@FXML

protected void onComboBoxSelect() {

String selectedOption = comboBox.getValue();

welcomeText.setText("You selected: " + selectedOption);

}

**H. ImageView**

The **ImageView** component is used to display images. It can scale and fit images to the desired dimensions.

**Code Example:**

<ImageView fx:id="imageView" fitWidth="200" fitHeight="200">

<image>

<Image url="file:resources/image.jpg" />

</image>

</ImageView>

**3. Additional JavaFX Components**

The following components are essential for modern UI applications and will enhance the capabilities of JavaFX applications.

**A. TabPane**

A TabPane allows developers to create tabbed interfaces where multiple views or components can be organized under tabs.

**Code Example:**

<TabPane fx:id="tabPane" tabClosingPolicy="UNAVAILABLE">

<tabs>

<Tab text="Home">

<content>

<Label text="Welcome to the Home Tab" />

</content>

</Tab>

<Tab text="Settings">

<content>

<VBox alignment="CENTER" spacing="10">

<Label text="Settings Page" />

<CheckBox text="Enable Notifications" />

</VBox>

</content>

</Tab>

</tabs>

</TabPane>

**B. Slider**

A Slider is a UI control for selecting a numeric value within a specified range by sliding a knob.

**Code Example:**

<Slider fx:id="volumeSlider" min="0" max="100" value="50" showTickLabels="true" showTickMarks="true" />

In the controller:

@FXML

private Slider volumeSlider;

@FXML

protected void onSliderChange() {

System.out.println("Current Slider Value: " + volumeSlider.getValue());

}

**C. ListView**

A ListView displays a scrollable list of items, allowing single or multiple selections.

**Code Example:**

<ListView fx:id="itemListView">

<items>

<FXCollections fx:factory="observableArrayList">

<String fx:value="Item 1" />

<String fx:value="Item 2" />

<String fx:value="Item 3" />

</FXCollections>

</items>

</ListView>

In the controller:

@FXML

private ListView<String> itemListView;

@FXML

protected void onItemSelected() {

String selectedItem = itemListView.getSelectionModel().getSelectedItem();

System.out.println("Selected Item: " + selectedItem);

}

**D. ProgressBar**

A ProgressBar visually represents the progress of a task.

**Code Example:**

<ProgressBar fx:id="progressBar" progress="0.5" />

In the controller:

@FXML

private ProgressBar progressBar;

@FXML

protected void updateProgress() {

progressBar.setProgress(0.75); // Update the progress to 75%

}

**4. Putting It All Together - Full Example:**

The following example combines all the components, both original and newly added, into a single user interface.

**FXML Layout (extended-layout.fxml)**

<VBox xmlns:fx="http://javafx.com/fxml" alignment="CENTER" spacing="20.0">

<Label fx:id="welcomeText" text="Welcome to the Extended JavaFX UI!" />

<!-- TabPane -->

<TabPane fx:id="mainTabPane">

<tabs>

<Tab text="Main">

<content>

<VBox spacing="10">

<TextField fx:id="nameField" promptText="Enter your name" />

<Button text="Submit" onAction="#onSubmit" />

</VBox>

</content>

</Tab>

<Tab text="Options">

<content>

<VBox spacing="10">

<Slider fx:id="optionsSlider" min="0" max="10" value="5" />

<ProgressBar fx:id="optionsProgressBar" progress="0.3" />

</VBox>

</content>

</Tab>

</tabs>

</TabPane>

<!-- ListView -->

<ListView fx:id="itemListView">

<items>

<FXCollections fx:factory="observableArrayList">

<String fx:value="Option A" />

<String fx:value="Option B" />

<String fx:value="Option C" />

</FXCollections>

</items>

</ListView>

<Button text="Clear" onAction="#onClear" />

</VBox>

**Controller (ExtendedController.java)**

package org.example.fxextended;

import javafx.fxml.FXML;

import javafx.scene.control.\*;

public class ExtendedController {

@FXML

private Label welcomeText;

@FXML

private TextField nameField;

@FXML

private Slider optionsSlider;

@FXML

private ProgressBar optionsProgressBar;

@FXML

private ListView<String> itemListView;

@FXML

protected void onSubmit() {

String name = nameField.getText();

welcomeText.setText("Hello, " + name + "!");

}

@FXML

protected void onClear() {

nameField.clear();

optionsSlider.setValue(0);

optionsProgressBar.setProgress(0);

itemListView.getSelectionModel().clearSelection();

}

}

**5. Student Task:**

**Task Description:** Create a comprehensive user interface for a "Course Registration System" using JavaFX, integrating all components covered in this lab.

**Requirements:**

1. Use TabPane to organize sections such as "Student Details," "Course Selection," and "Summary."
2. Use Slider to adjust the number of courses.
3. Use ListView to display available courses and allow the user to select multiple courses.
4. Use ProgressBar to indicate the percentage of registration completion.
5. Include a "Submit" button to display a summary of selected options and a "Clear" button to reset the form.

**Deliverables:**

1. FXML layout file (course-registration.fxml).
2. Controller class to handle all actions.
3. A screenshot or preview of the working application.