UNIT 5

Label

- What is a Label?
- A Label in JavaFX is a non-editable text control used to display short text or an image. It's one of the most basic UI components.
- Label simpleLabel = new Label("Welcome to JavaFX!");
- Label styledLabel = new Label("Styled Text");
- styledLabel.setFont(Font.font("Arial", FontWeight.BOLD, 20));
- Displays read-only text
- Can include images/icons
- Supports text wrapping
- Styleable with CSS
- Basic property
- label.setText("New text"); // Change text
- label.setFont(Font.font(16)); // Set font size
- label.setTextFill(Color.RED); // Change text color
- label.setWrapText(true);
- Button Class: javafx.scene.control.Button is a UI component that triggers actions when clicked.

Creating Buttons

- Button simpleButton = new Button("Click Me");
- Common Properties
- button.setText("New Text"); // Change button text
- button.setDisable(true); // Disable the button
- button.setPrefSize(100, 50); // Set preferred size
- button.setWrapText(true); // Enable text wrapping

Property	Description	Example
`text`	Displayed text	`btn.setText("Save")`
`graphic`	Icon (ImageView)	`btn.setGraphic(new ImageView())`
`disable`	Enable/disable interaction	`btn.setDisable(true)`
`style`	CSS styling	`btn.setStyle("-fx-color: red")`

CheckBox

- Purpose: A tri-state selection control (checked/unchecked/indeterminate)
- Data Binding: Supports property observation via selectedProperty()
- CheckBox checkBox = new CheckBox("Accept Terms"); // With label
- CheckBox emptyBox = new CheckBox();

Radio Buttons

- Exclusive Selection: RadioButtons allow single-choice selection within a group
- ToggleGroup Requirement: Must be grouped using ToggleGroup for mutual exclusivity
-

Key Properties & Methods

Property/Method	Description	Example
`isSelected()`	Check selection state	`rb1.isSelected()`
`setSelected()`	Programmatically select	`rb2.setSelected(true)`
`selectedProperty()`	Observable selection property	For data binding
`setUserData()`	Attach custom data	`rb1.setUserData("admin")`

TextField

- Purpose: Single-line text input control
- Best For: Short text input (names, numbers, search queries)

Common Methods

Method	Description
`setText()` / `getText()`	Set/retrieve content
`clear()`	Clear the field
`setEditable(false)`	Make read-only
`setFont()`	Change text font
`selectAll()`	Select all text

Text Area (Multi-line Input)

- Purpose: Multi-line text input with scrolling
- Best For: Long text (comments, descriptions, code)

Common Methods

Method	Description
`setWrapText()`	Enable/disable text wrapping
`setScrollTop()`	Control vertical scroll
`setPrefRowCount()`	Set visible rows
`positionCaret()`	Move cursor position

Example:

```
import javafx.application.Application;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
```

```
public class forms extends Application {
   @Override
   public void start(Stage stage) {
      // Create controls
      TextField nameField = new TextField();
      nameField.setPromptText("Your name");
      TextArea noteArea = new TextArea();
      noteArea.setPromptText("Your notes");
      noteArea.setMaxHeight(100);
      ToggleGroup group = new ToggleGroup();
      RadioButton opt1 = new RadioButton("Option 1");
      RadioButton opt2 = new RadioButton("Option 2");
      opt1.setToggleGroup(group);
      opt2.setToggleGroup(group);
      Button submit = new Button("Submit");
      // Layout
      VBox root = new VBox(10, nameField, noteArea, opt1, opt2,
submit);
      // Show window
      stage.setScene(new Scene(root, 250, 250));
      stage.setTitle("Mini Form");
      stage.show();
   }

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

ComboBox

- In JavaFX, a ComboBox is a versatile UI control that presents a dropdown list of options from which users can select one. It extends the ComboBoxBase class and allows for both predefined selections and, optionally,
- Items List: The ComboBox has an items property, functioning similarly to the ListView items property. This property holds the list of options displayed to the user upon clicking the ComboBox.
- Value Property: The value property represents the currently selected item or the user-entered value
- Example:
- import javafx.collections.FXCollections;
- import javafx.scene.control.ComboBox;

```
// Create a ComboBox with predefined items
ComboBox<String> comboBox = new ComboBox<>(FXCollections.observableArrayList("Option 1", "Option 2", "Option 3"));

// Alternatively, create an empty ComboBox and add items later
ComboBox<String> comboBox = new ComboBox<>();
comboBox.getItems().addAll("Option 1", "Option 2", "Option 3");
```

ListView

- A ListView is a UI control that displays a vertical or horizontal list of items from which users can select one or more options. It's commonly used for presenting a scrollable list of choices in JavaFX
- Example:

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.ListView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ListViewExample extends Application {

    @Override
    public void start(Stage stage) {
        // Creating an ObservableList of items
        ObservableList<String> items = FXCollections.observableArrayList(
            "Option 1", "Option 2", "Option 3"
        );

        // Creating the ListView and setting its items
        ListView<String> listView = new ListView<>(items);

        // Setting the selection mode to single selection
        listView.getSelectionModel().setSelectionMode(javafx.scene.control.SelectionMode.SINGLE);
```

```
        // Handling selection changes
        listView.getSelectionModel().selectedItemProperty().addListener((observable, oldValue, newValue) -> {
            System.out.println("Selected item: " + newValue);
        });

        // Setting up the scene and stage
        VBox vbox = new VBox(listView);
        Scene scene = new Scene(vbox, 300, 200);
        stage.setScene(scene);
        stage.setTitle("ListView Example");
        stage.show();
    }

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

ScrollBar

In JavaFX, a ScrollBar is a control that enables users to navigate through a range of values by interacting with a draggable thumb along a horizontal or vertical track. It includes increment and decrement buttons for fine-tuned adjustments. ScrollBars are commonly integrated into other controls, such as ScrollPane and ListView

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.ScrollBar;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.geometry.Orientation;

public class ScrollBarExample extends Application {

    @Override
    public void start(Stage stage) {
        ScrollBar scrollBar = new ScrollBar();
        scrollBar.setOrientation(Orientation.VERTICAL);
        scrollBar.setMin(0);
        scrollBar.setMax(100);
```

```
        scrollBar.setValue(50);

        VBox vbox = new VBox(scrollBar);
        Scene scene = new Scene(vbox, 200, 200);
        stage.setScene(scene);
        stage.setTitle("ScrollBar Example");
        stage.show();
    }

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

Slider

- In JavaFX, a Slider is a UI control that allows users to select a numeric value from a specified range by moving a thumb along a track. It's commonly used for settings like volume control, brightness adjustment, or any scenario requiring input within a continuous or discrete range.
- Track: The bar representing the range of values.
- Thumb: The draggable element that moves along the track to select a value.
- Tick Marks and Labels (Optional): Visual indicators along the track denoting specific values, enhancing user precision.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Slider;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class SliderExample extends Application {

    @Override
    public void start(Stage stage) {
        // Creating a slider with a range from 0 to 100 and an initial
value of 50
        Slider slider = new Slider(0, 100, 50);

        // Enabling tick marks and labels
```

```
        slider.setShowTickMarks(true);
        slider.setShowTickLabels(true);
        slider.setMajorTickUnit(25);
        slider.setMinorTickCount(4);
        slider.setBlockIncrement(10);

        // Setting up the scene and stage
        VBox vbox = new VBox(slider);
        Scene scene = new Scene(vbox, 300, 100);
        stage.setScene(scene);
        stage.setTitle("JavaFX Slider Example");
        stage.show();
    }

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

- Slider Initialization: A Slider is instantiated with a minimum value of 0, a maximum of
- Tick Marks and Labels: Tick marks and labels are enabled to display major ticks at intervals of 25 and minor ticks dividing each major interval into 4 parts.
- Block Increment: Sets the increment value for keyboard navigation and mouse clicks on the track.

images and videos

- In JavaFX, you can incorporate images and videos into your applications using the ImageView and MediaView classes, respectively. Below are simple examples demonstrating how to use each
- An Image is loaded from a file named "example.jpg". Ensure the file path is correct.
- An ImageView is created to display the image.
- The setFitWidth method sets the width of the image to 300 pixels, and setPreserveRatio(true) maintains the image's aspect ratio.
- A StackPane is used to center the ImageView in the scene.
- A Media object is created with the URL of the media file. Ensure the file path is correct.
- A MediaPlayer is created to control the playback of the media.
- A MediaView is created to display the media.

- The setFitWidth method sets the width of the media to 400 pixels, and setPreserveRatio(true) maintains the media's aspect ratio.
- A StackPane is used to center the MediaView in the scene.
- The mediaPlayer.play() method starts the playback of the media.

Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import javafx.stage.Stage;

public class MediaViewExample extends Application {

    @Override
    public void start(Stage stage) {
        // Load the media file
        String mediaUrl = "file:example.mp4"; // Ensure the path is correct
        Media media = new Media(mediaUrl);
        // Create a MediaPlayer to control playback of the media
        MediaPlayer mediaPlayer = new MediaPlayer(media);
        // Create a MediaView to display the media
        MediaView mediaView = new MediaView(mediaPlayer);
        // Optional: Set properties for the MediaView
        mediaView.setFitWidth(400); // Set the width to 400 pixels
        mediaView.setPreserveRatio(true); // Preserve the aspect ratio
        // Create a layout pane to hold the MediaView
        StackPane root = new StackPane(mediaView);
        // Create a scene with the layout pane
        Scene scene = new Scene(root, 600, 400);
        // Set the stage title and scene, then show the stage
        stage.setTitle("MediaView Example");
        stage.setScene(scene);
        stage.show();

        // Start playing the media
        mediaPlayer.play();
    }

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

```
    }
}
```