**Ripunjay Narula (19BCE0470)**

**Java Theory Digital Assignment**

**Submitted to- Priya G**

***JAVAFX UI CONTROLS:***

JavaFX provides several classes in the package **javafx.scene.control**. To create various GUI components (controls), JavaFX supports several controls. Each control is represented by a class. You can create a control by instantiating its respective class.

Following is the list of commonly used controls while the GUI is designed using JavaFX:

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Control** | **Description** |
| 1 | [Label](https://www.javatpoint.com/javafx-label) | Label is a component that is used to define a simple text on the screen. Typically, a label is placed with the node, it describes. |
| 2 | [Button](https://www.javatpoint.com/javafx-button) | Button is a component that controls the function of the application. Button class is used to create a labelled button. |
| 3 | [RadioButton](https://www.javatpoint.com/javafx-radiobutton) | The Radio Button is used to provide various options to the user. The user can only choose one option among all. A radio button is either selected or deselected. |
| 4 | [CheckBox](https://www.javatpoint.com/javafx-checkbox) | Check Box is used to get the kind of information from the user which contains various choices. User marked the checkbox either on (true) or off(false). |
| 5 | [TextField](https://www.javatpoint.com/javafx-textfield) | Text Field is basically used to get the input from the user in the form of text. javafx.scene.control.TextField represents TextField |
| 6 | [PasswordField](https://www.javatpoint.com/javafx-passwordfield) | PasswordField is used to get the user's password. Whatever is typed in the passwordfield is not shown on the screen to anyone. |
| 7 | [HyperLink](https://www.javatpoint.com/javafx-hyperlink) | HyperLink are used to refer any of the webpage through your appication. It is represented by the class **javafx.scene.control.HyperLink** |
| 8 | [Slider](https://www.javatpoint.com/javafx-slider) | Slider is used to provide a pane of options to the user in a graphical form where the user needs to move a slider over the range of values to select one of them. |
| 9 | [ProgressBar](https://www.javatpoint.com/javafx-progressbar) | Progress Bar is used to show the work progress to the user. It is represented by the class **javafx.scene.control.ProgressBar**. |
| 10 | [ProgressIndicator](https://www.javatpoint.com/javafx-progress-indicator) | Instead of showing the analogue progress to the user, it shows the digital progress so that the user may know the amount of work done in percentage. |
| 11 | [ScrollBar](https://www.javatpoint.com/javafx-scrollbar) | JavaFX Scroll Bar is used to provide a scroll bar to the user so that the user can scroll down the application pages. |
| 12 | [Menu](https://www.javatpoint.com/javafx-menu) | JavaFX provides a Menu class to implement menus. Menu is the main component of any application. |
| 13 | [ToolTip](https://www.javatpoint.com/javafx-tooltip) | JavaFX ToolTip is used to provide hint to the user about any component. It is mainly used to provide hints about the text fields or password fields being used in the application. |

***JAVAFX UI MENU:***

A menu is a list of options or commands presented to the user. In JavaFX a menu is represented by the **javafx.scene.control.Menu** class, you can create a menu by instantiating this class.

While instantiating, you can pass the title of the menu as a parameter to its constructor. The Menu class contains an observable list that holds the contents of the menu (menu items).

## **Menu items and menu bar**

The menu items are represented by the ****javafx.scene.control.MenuItem**** class, a superclass of the Menu class. You can display a text or a graphic as a menu item and add the desired cation to it.

To create a menu −

* Instantiate the Menu class.
* Create a required number of menu items by instantiating the MenuItem class.
* Add the created menu items to the observable list of the menu.

***JavaFX Event Handling:***

JavaFX provides us the flexibility to create various types of applications such as Desktop Applications, Web applications and graphical applications. In the modern day applications, the users play a vital role in the proper execution of the application. The user need to interact with the application in most of the cases.

In JavaFX, An event is occurred whenever the user interacts with the application nodes. There are various sources by using which, the user can generate the event. For example, User can make the use of mouse or it can press any button on the keyboard or it can scroll any page of the application in order to generate an event. Hence,we can say that the events are basically the notifications that tell us that something has occurred at the user's end.

A perfect Application is the one which takes the least amount of time in handling the events. This part of the tutorial This part of the tutorial describes the way in which the events are handled in JavaFX.

**Types of Events**

In general, the events are mainly classified into the following two types.

1. Foreground Events

Foreground events are mainly occurred due to the direct interaction of the user with the GUI of the application. Such as clicking the button, pressing a key, selecting an item from the list, scrolling the page, etc.

1. Background Events

Background events doesn't require the user's interaction with the application. These events are mainly occurred to the operating system interrupts, failure, operation completion, etc.

**Processing Events in JavaFX**

In JavaFX, events are basically used to notify the application about the actions taken by the user. JavaFX provides the mechanism to capture the events, route the event to its target and letting the application handle the events.

JavaFX provides the class javafx.event.Event which contains all the subclasses representing the types of Events that can be generated in JavaFX. Any event is an instance of the class Event or any of its subclasses.

There are various events in JavaFX i.e. MouseEvent, KeyEvent, ScrollEvent, DragEvent, etc. We can also define our own event by inheriting the class javafx.event.Event.

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Property** | **Description** |
| 1 | Event Type | It is the type of the event that is being generated. It is basically the instance of EventType class. It is hierarchical. The instance of EventType class is further classified into various type of events for example KeyEvent class contains KEY\_PRESSED, KEY\_RELEASED, and KEY\_TYPED types. | |
| 2 | Source | It represents source of the event i.e. the origin which is responsible to generate the event. | |
| 3 | Target | It is the node on which the event is generated. It remains unchanged for the generated event. It is the instance of any of the class that implements the EventTarget interface. | |

The properties of an event is described in the following table:

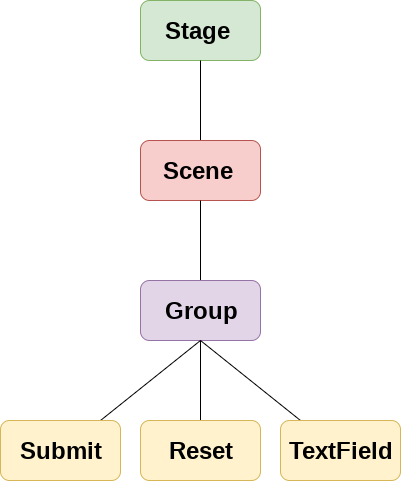
**Event Delivery Process**

The following steps need to be followed in order to handle the events.

1. **Route Construction**

An Event Dispatch Chain is created in order to determine the default route of the event, whenever it is generated. The Event dispatch chain contains the path from the stage to the Node on which the event is generated.

An event dispatch chain is created in the following image for the event generated on one of the scene graph node.



1. **Event Capturing Phase**

Once the Event Dispatch Chain is created, the event is dispatched from the source node of the event. All the nodes are traversed by the event from top to bottom. If the event filter is registered with any of these nodes, then it will be executed. If any of the nodes are not registered with the event filter then the event is transferred to the target node. The target node processes the event in that case.

1. **Event Bubbling**

Once the event is processed by the target node or by any of the registered filter, the event traverses all the nodes again from the bottom to the stage node. If any of these nodes are registered with the event filter, then it will get executed otherwise the process gets completed.

1. **Event Handlers and Filters**

Event Handlers and filters contains application logic to process an event. A node can be registered to more than one Event Filters. The interface javafx.event.EventHandler must be implemented by all the event handlers and filters.

1. **Convenience Methods**

JavaFX provides the convenience methods which can be used to handle events within our JavaFX application. They provide an easy way to create and register event handlers to respond to KeyEvent, MouseEvent, Action Event, Drag & Drop Events and many more.

Node class contains various Event Handler properties which can be set to the User defined Event Handlers using the setter methods defined in the class.

Setting, EventHandler properties of the Node class, to the user defined event handlers, automatically registers the handlers to receive the corresponding event types.

\*\*The EventHandler properties along with their setter methods (convenience methods) are described in the following table:

|  |  |  |
| --- | --- | --- |
| **EventHandler Property** | **Description** | **Setter Methods** |
| onContextMenuRequested | This is of the type EventHandler of ContextMenuEvent. This is assigned to a function which is to be called when the context menu is requested on the node. | setOnContextMenuRequested(EventHandler value ) |
| onDragDetected | This is of the type EventHandler of MouseEvent. This indicates a function which is to be called when the drag gesture is detected. | setOnDragDetected(EventHandler value ) |
| onDragDone | This is of the type EventHandler of DragEvent. | setOnDragDone(EventHandler value ) |
| onDragDropped | This is of the type EventHandler of DragEvent. This is assigned to a function which is to be called when the mouse is released during a drag and drop operation. | setOnDragDropped(EventHandler value ) |
| onDragEntered | This is of the type EventHandler of DragEvent. This is assigned to a function which is to be called when the drag gesture enters the node. | setOnDragEntered(EventHandler value ) |
| onDragExited | This is of the type EventHandler of DragEvent. This is assigned to a function which is to be called when the drag gesture exits the node. | setOnDragExited(EventHandler value ) |
| onDragOver | This is of the type EventHandler of DragEvent.This is assigned to a function which is to be called when the drag gesture progresses within the node. | setOnDragOver(EventHandler value ) |
| onInputMethodTextChanged | This is of the type EventHandler of InputMethodEvent. This is assigned to a function which is to be called when the Node has focus and the input method text has changed. | setOnInputMethodTextChanged(EventHandler value ) |
| onKeyPressed | This is of the type EventHandler of KeyEvent. This is assigned to a function which is to be called when the Node has focus and key is pressed. | setOnKeyPressed(EventHandler value ) |
| onKeyReleased | This is of the type EventHandler of KeyEvent. This is assigned to a function which is to be called when the Node has focus and key is released. | setOnKeyReleased(EventHandler value ) |
| onKeyTyped | This is of the type EventHandler of KeyEvent.This is assigned to a function which is to be called when the Node has focus and key is typed. | setOnKeyTyped(EventHandler value ) |
| onMouseClicked | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is clicked on the node. | setOnMouseClicked(EventHandler value ) |
| onMouseDragEntered | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture enters the node. | setOnMouseDragEntered(EventHandler value ) |
| onMouseDragExited | This is of the type EventHandler of MouseDragEvent.This is assigned to a function which is to be called when a full press drag release gesture exits the node. | setOnMouseDragExited(EventHandler value ) |
| onMouseDragged | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when the mouse button is pressed and dragged on the node. | setOnMouseDragged(EventHandler value ) |
| onMouseDragOver | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture progresses within the node. | setOnMouseDragOver(EventHandler value ) |
| onMouseDragReleased | This is of the type EventHandler of MouseDragEvent. This is assigned to a function which is to be called when a full press drag release gesture ends within the node. | setOnMouseDragReleased(EventHandler value ) |
| onMouseEntered | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse enters the node. | setOnMouseEntered(EventHandler value ) |
| onMouseExited | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse exits the node. | setOnMouseExited(EventHandler value ) |
| onMouseMoved | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse moves within the node and no button has been pushed. | setOnMouseMoved(EventHandler value ) |
| onMousePressed | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is pressed on the node. | setOnMousePressed(EventHandler value ) |
| onMouseReleased | This is of the type EventHandler of MouseEvent. This is assigned to a function which is to be called when the mouse button is released on the node. | setOnMouseReleased(EventHandler value ) |
| onRotate | This is of the type EventHandler of RotateEvent. This is assigned to a function which is to be called when the rotation action is performed on the node. | setOnRotate(EventHandler value ) |
| onRotationFinished | This is of the type EventHandler of RotateEvent. This is assigned to a function which is to be called when the rotation gesture ends. | setOnRotationFinished(EventHandler value ) |
| onRotationStarted | This is of the type EventHandler of RotateEvent. This is assigned to a function which is to be called when the rotation gesture is first detected. | setOnRotationStarted(EventHandler value ) |
| onScrollFinished | This is of the type EventHandler of ScrollEvent. This is assigned to a function which is to be called when the scroll gesture ends. | setOnScrollFinished(EventHandler value ) |
| onScroll | This is of the type EventHandler of ScrollEvent. This is assigned to a function which is to be called when the scroll action is performed. | setOnScroll(EventHandler value ) |
| onScrollStarted | This is of the type EventHandler of ScrollEvent. This is assigned to a function which is to be called when the scrolling gesture is detected. | setOnScrollStarted(EventHandler value ) |
| onSwipeDown | This is of the type EventHandler of SwipeEvent. This is assigned to a function which is to be called when the downwards swipe happens on the node. | setOnSwipeDown(EventHandler value ) |
| onSwipeUP | This is of the type EventHandler of SwipeEvent. This is assigned to a function which is to be called when the upwards swipe happens on the node. | setOnSwipeUP(EventHandler value ) |
| onSwipeLeft | This is of the type EventHandler of SwipeEvent. This is assigned to a function which is to be called when the leftwards swipe happens on the node. | setOnSwipeLeft(EventHandler value ) |
| onSwipeRight | This is of the type EventHandler of SwipeEvent. This is assigned to a function which is to be called when the Rightwards swipe happens on the node. | setOnSwipeRight(EventHandler value ) |
| onTouchMoved | This is of the type EventHandler of TouchEvent. This is assigned to a function which is to be called when the touch point is moved on the node. | setOnTouchMoved(EventHandler value ) |
| onTouchReleased | This is of the type EventHandler of TouchEvent. This is assigned to a function which is to be called when the touch point is released on the node. | setOnTouchReleased(EventHandler value ) |
| onTouchStationary | This is of the type EventHandler of TouchEvent. This is assigned to a function which is to be called when the touch point is pressed and stays still | setOnTouchStationary(EventHandler value ) |
| onZoomFinished | This is of the type EventHandler of ZoomEvent. This is assigned to a function which is to be called when the zooming gesture ends. | setOnZoomFinished(EventHandler value ) |
| onZoom | This is of the type EventHandler of ZoomEvent. This is assigned to a function which is to be called when the zooming gesture is performed. | setOnZoom(EventHandler value ) |
| onZoomStarted | This is of the type EventHandler of ZoomEvent. This is assigned to a function which is to be called when the zooming gesture is detected. | setOnZoomStarted(EventHandler value ) |

***GUI CREATION***

The GUI is created using **IntelliJ IDEA**. The application is a To-do List which has the following functions:

1. Adding Tasks
2. Deleting Tasks
3. Saving the final list to a text file (tasks.txt)e

The files of the GUI with their respective contents are as follows:

**Main.java:**

package sample;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application {

@Override

public void start(Stage primaryStage) throws Exception{

Parent root = FXMLLoader.load(getClass().getResource("project.fxml"));

Scene primaryscene=new Scene(root, 300, 275);

primaryStage.setTitle("19BCE0470 To-do List");

primaryStage.setScene(primaryscene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

**Controller.java:**

package sample;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.ListView;

import javafx.scene.control.TextField;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.net.URL;

import java.util.List;

import java.util.ResourceBundle;

import java.util.Scanner;

public class Controller implements Initializable {

@FXML private TextField newtask;

@FXML private ListView<String> tasklist;

private String filepath="tasks.txt";

private File data= new File(filepath);

public void addnewtask(ActionEvent e){

String text= newtask.getText();

if (!text.equals("")){

tasklist.getItems().add(text);

newtask.setText("");

} else{

System.out.println("NO INPUT PRESENT ");

}

//System.out.println(text);

}

public void exitprogram(ActionEvent e){

List<String> currenttasks=tasklist.getItems();

try {

FileWriter writer = new FileWriter(filepath);

for (String text: currenttasks){

text+="\n";

writer.write(text);

}

writer.close();

} catch (IOException ioException) {

ioException.printStackTrace();

}

System.exit(0);

}

public void deletetask(ActionEvent e){

String selected= tasklist.getSelectionModel().getSelectedItem();

if (selected!=null){

tasklist.getItems().remove(selected);

} else {

System.out.println("NO SELECTION MADE");

}

}

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

try {

Scanner myReader= new Scanner(data);

while (myReader.hasNextLine()){

String text=myReader.nextLine();

tasklist.getItems().add(text);

}

myReader.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

}

**project.fxml:**

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

<?import javafx.geometry.\*?>

<?import javafx.scene.control.\*?>

<?import javafx.scene.layout.\*?>

<?import javafx.scene.text.\*?>

<GridPane alignment="center" hgap="10" vgap="10" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controller">

<children>

<AnchorPane id="bgpane" prefHeight="270.0" prefWidth="300.0" stylesheets="@../../out/Styles/styles.css" />

<VBox prefHeight="200.0" prefWidth="100.0">

<children>

<HBox prefHeight="47.0" prefWidth="300.0" spacing="10.0">

<padding>

<Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />

</padding>

<children>

<Button id="addbtn" mnemonicParsing="false" onAction="#addnewtask" prefHeight="26.0" prefWidth="90.0" stylesheets="@../../out/Styles/styles.css" text="Add Task" textFill="WHITE">

<font>

<Font name="System Bold Italic" size="12.0" />

</font></Button>

<Button id="dltbtn" mnemonicParsing="false" onAction="#deletetask" prefHeight="26.0" prefWidth="90.0" stylesheets="@../../out/Styles/styles.css" text="Delete Task" textFill="WHITE">

<HBox.margin>

<Insets />

</HBox.margin>

<font>

<Font name="System Bold" size="12.0" />

</font>

</Button>

<Button id="exitbtn" mnemonicParsing="false" onAction="#exitprogram" prefWidth="90.0" stylesheets="@../../out/Styles/styles.css" text="Exit" textFill="WHITE">

<font>

<Font name="System Bold" size="12.0" />

</font>

</Button>

</children>

</HBox>

<TextField id="txt" fx:id="newtask" prefWidth="280.0" promptText="Add Task">

<VBox.margin>

<Insets left="10.0" right="10.0" />

</VBox.margin>

<opaqueInsets>

<Insets />

</opaqueInsets>

</TextField>

<ListView id="txt" fx:id="tasklist" prefHeight="200.0" prefWidth="200.0">

<opaqueInsets>

<Insets />

</opaqueInsets>

<VBox.margin>

<Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />

</VBox.margin>

</ListView>

</children></VBox>

</children>

<columnConstraints>

<ColumnConstraints />

</columnConstraints>

<rowConstraints>

<RowConstraints />

</rowConstraints>

</GridPane>

