JavaFX Events

Basic of JavaFX Events:

A GUI based applications are mostly driven by Events. Events are the actions that the user performs and the responses the application generates.

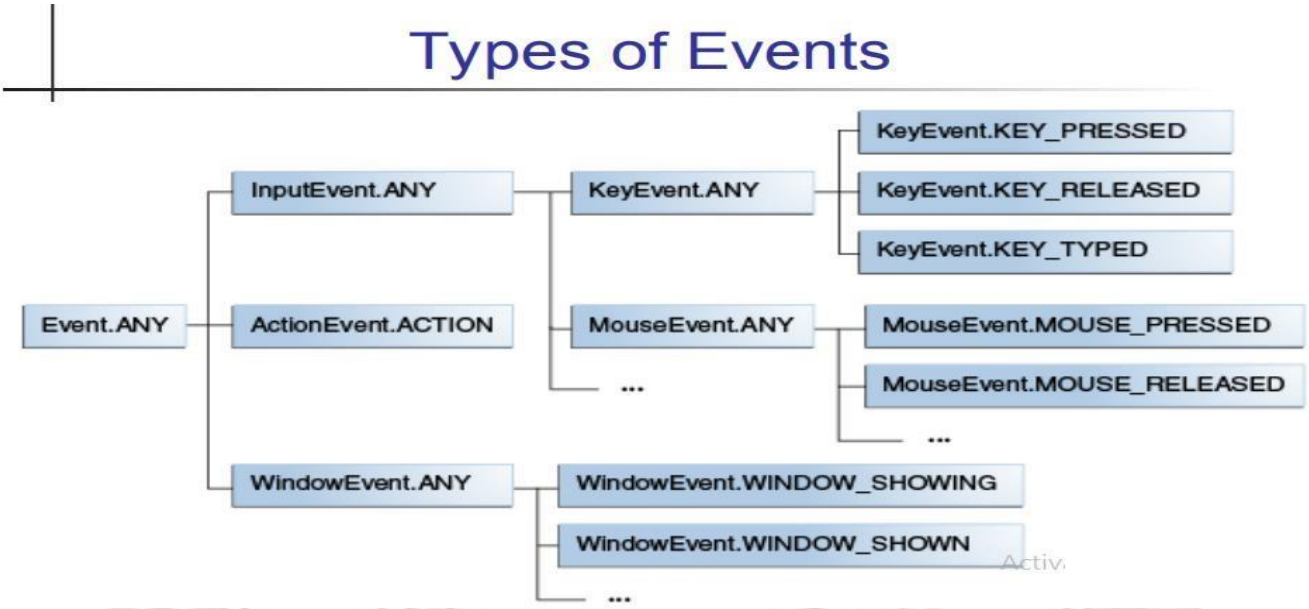
Example: Button clicks by user, key press on the application etc.

An event is a notification about a change. It encapsulates the state changes in the event source. Registered event filters and event handlers within the application receive the event and provide a response.

- JavaFX provides support to handle events through the base class "Event" which is available in the package javafx.event.

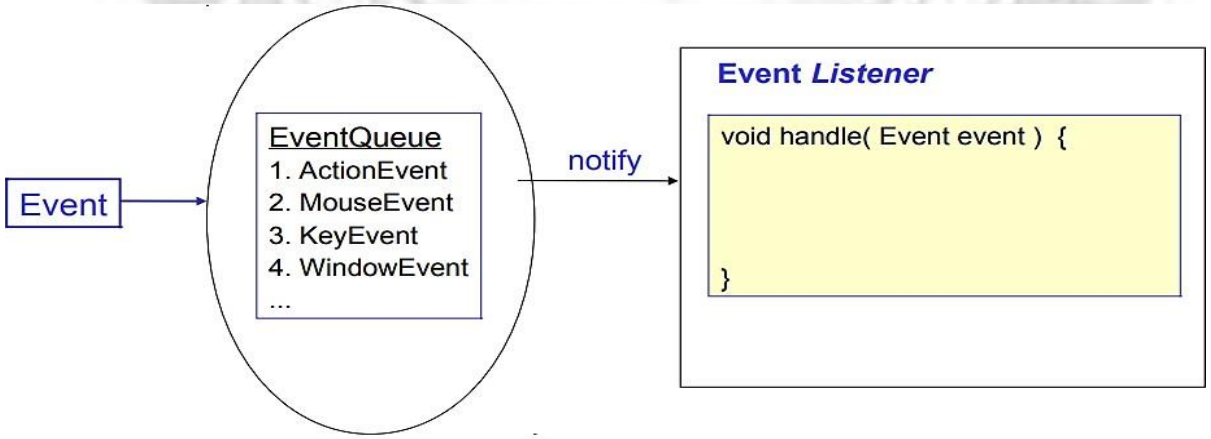
Examples of Events:

- **Action Event** — widely used to indicate things like when a button is pressed.
  Class:- ActionEvent
  Actions:- button pressed.
- **Mouse Event** — occurs when mouse is clicked
  Class:- MouseEvent
  Actions:- mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target.
- **Drag Event** — occurs when the mouse is dragged.
  Class:- DragEvent
  Actions:- drag entered, drag dropped, drag entered target, drag exited target, drag over.
- **Key Event** — indicates that a keystroke has occurred.
  Class:- KeyEvent
  Actions:- Key pressed, key released and key typed.
- **Window Event:**
  Class:- WindowEvent
  Actions:- window hiding, window shown, window hidden, window showing.
- **Scroll Event** — indicates scrolling by mouse wheel, track pad, touch screen, etc...
- **TouchEvent** — indicates a touch screen action



: Event Handling:

Event handling is the mechanism that controls the event and decides what should happen, if an event occurs. It has the code which is known as Event Handler that is executed when an event occurs.



Event Handling in JavaFX is done by Event Filters and Event Handlers. They contain the event handling logic to process a generated event.
Every event in JavaFX has three properties:

1. Event source
2. Event target
3. Event type

S.N	Property	Description
1	**Event Source**	It denotes source of the event i.e. the origin which is responsible for generating the event.
2	**Event Target**	It denotes the node on which the event is created. It remains unaffected for the generated event. Event Target is the instance of any of the class that implements the java interface "EventTarget".
3	**Event Type**	It is the type of the event that is being generated. It is basically the instance of EventType class. Example: KeyEvent class contains KEY_PRESSED, KEY_RELEASED, and KEY_TYPED types.

Phases of Event Handling in JavaFX:

Whenever an event is generated, JavaFX undergoes the following phases:

1. Target Selection – Depends on the particular event type.
2. Route Construction – Specified by the event target.
3. Event Capturing – Event travels from the stage to the event target.
4. Event Bubbling – Event travel back from the target to the stage.

1. **Target Selection:**

   The first step to process an event is the selection of the event target. Event target is the node on which the event is created. Event target is selected based in the Event Type.

   - For key events, the target is the node that has key focus.
   - The node where the mouse cursor is located is the target for mouse events.

2. **Route Construction:**

   Usually, an event travels through the event dispatchers in order in the event dispatch chain. An Event Dispatch Chain is created to determine the default route of the event whenever an event is generated. It contains the path from the stage to the node on which the event is generated.

3. **Event Capturing:**

   In this phase, an event is dispatched by the root node and passed down in the Event Dispatch Chain to the target node.

   Event Handlers will not be invoked in this phase.

   If any node in the chain has registered the event filter for the type of event that occurred, then the filter on that node is called. When the filter completes, the

event is moved down to the next node in the Dispatch Chain. If no event filters consumes the event, then the event target receives and processes the generated event.

4. **Event Bubbling:**

   In this phase, a event returns from the target node to the root node along the event dispatch chain.

   Events handlers will be invoked in this phase.

   If any node in the chain has a handler for the generated event, that handler is executed. When the handler completes, the event is bubbled up in the chain. If the handler is not registered for a node, the event is returned to the bubbled up to next node in the route. If no handler in the path consumed the event, the root node consumes the event and completes the processing.

Three methods for Event Handling:

1. **Convenience Methods:**
   - ✓ setOnKeyPressed(eventHandler);
   - ✓ setOnMouseClicked(eventHandler);
2. **Event Handler/Filter Registration Methods:**
   - ✓ addEventHandler(eventType, eventHandler);
   - ✓ addEventFilter(eventType, eventFilter);
3. **Event Dispatcher Property (lambda expression).**

Event Filters:

- ❖ Event Filters provides the way to handle the events generated by the Keyboard Actions, Mouse Actions, Scroll Actions and many more event sources.
- ❖ They process the events during Event Capturing Phase.
- ❖ A node must register the required event filters to handle the generated event on that node. **handle()** method contains the logic to execute when the event is triggered.

- ❖ **Adding Event-Filter to a node:**

  To register the event filter for a node, **addEventFilter()** method is used.

Syntax:

```
node.addEventFilter (<Event_Type>, new EventHandler<Event-Type>()
{
public void handle(Event-Type)
{
//Actual logic
});
```

Where,
First argument is the type of event that is generated.
Second argument is the filter to handle the event.

❖ **Removing Event-Filter:**

We can remove an event filter on a node using **removeEventFilter()** method.

Syntax:

node.removeEventFilter(<Input-Event>, filter);

Event Handlers:

❖ Event Filters provides the way to handle the events generated by the Keyboard Actions, Mouse Actions, Scroll Actions and many more event sources.
❖ They are used to handle the events during Event Bubbling Phase.
❖ A node must register the event handlers to handle the generated event on that node. **handle()** method contains the logic to execute when the event is triggered.

❖ **Adding Event-Handler to a node:**

To register the event handler for a node, **addEventHandler()** method is used.

Syntax:

```
node.addEventHandler (<Event_Type>, new EventHandler<Event-Type>()
{
public void handle(<Event-Type> e)
{
//Handling Code
});
```

Where,
First argument is the type of event that is generated.
Second argument is the filter to handle the event.

❖ **Removing Event-Filter:**

We can remove an event handler on a node using **removeEventHandler()** method.

Syntax:

node.removeEventHandler(<EventType>, handler);

A node can register for more than one Event Filters and Handlers.
The interface javafx.event.EventHanler must be implemented by all the event filters and event handlers.

5.3: Handling Key Events and Mouse Events

HANDLING KEY EVENTS

Key Event – It is an input event that indicates the key stroke occurred on a node.

✓ It is represented by the class named KeyEvent.

- ✓ This event includes actions like key pressed, key released and key typed.

Types of Key Event in Java

1. KEY_PRESSED – When a key on the keyboard is pressed, this event will be triggered.
2. KEY_RELEASED – When the pressed key on the keyboard is released, this event will be executed.
3. KEY_TYPED – This event will be triggered when a Unicode character is entered

Methods in the KeyEvent class to get the key details

- KeyCode **getCode()** – This method returns the key information or the KeyCode enum constant linked with the pressed or released key.
- String **getText()** – This method returns a String description of the KeyCode linked with the KEY_PRESSED and KEY_RELEASED events.
- String **getCharacter()** – This method returns a string representing a character or a sequence of characters connected with the KEY_TYPED event.

Example:

```
/* Program to handle KeyTyped and KeyPressed Events.
   Whenever a key is pressed in TextFiled1, it will be displayed in TextFiled2.
   Whenever BackSpace key is pressed in TextFiled1, last character in TextFiled2 will
   be erased.
   If you attempt to type a character in TextField2, alert box will be displaying  */
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.*;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.scene.input.*;
import javafx.scene.control.Alert.*;
public class NewFXMain  extends Application {

  @Override
  public void start(Stage primaryStage)
{
     TextField tf1=new TextField();
     TextField tf2=new TextField();
     Label l1=new Label("Text Pressed : ");
     EventHandler<KeyEvent> handler1=new EventHandler<KeyEvent>() {
       String str="",str1="";
       int d;
              public void handle(KeyEvent event)
              {
                 if(event.getCode()== KeyCode.BACK_SPACE)
                 {
```

```
                str=str.substring(0,str.length()-1);
                tf2.setText(str);
                }


                else
                {
                str+=event.getText();
                tf2.setText(str);
                }
              }
            };

    EventHandler<KeyEvent> handler2=new EventHandler<KeyEvent>(){
      public void handle(KeyEvent event)
      {
        Alert a=new Alert(AlertType.WARNING);
        a.setContentText("Sorry! Dont Type Anything Here!!");
        a.show();
      }
    };

    tf1.setOnKeyPressed(handler1);
    tf2.setOnKeyTyped(handler2);
    GridPane root = new GridPane();
    root.addRow(1,tf1);
    root.addRow(2,l1);
    root.addRow(3,tf2);
    Scene scene = new Scene(root, 300, 250);
    primaryStage.setTitle("KeyEvent-Demo");
    primaryStage.setScene(scene);
    primaryStage.show();
  }

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

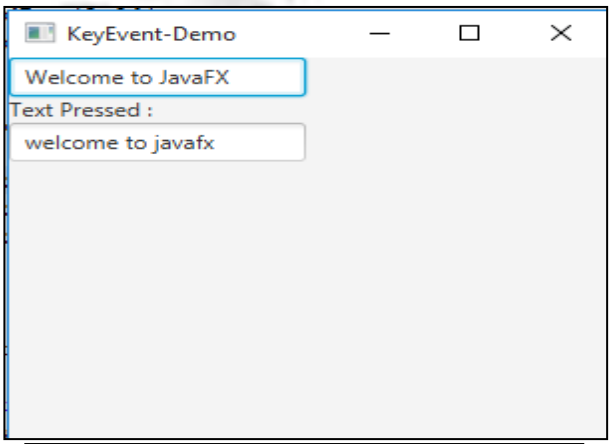


Figure 1: When a key is pressed in TextField 1

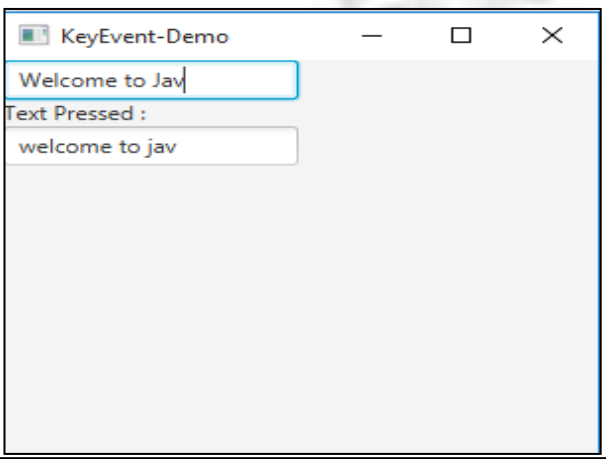
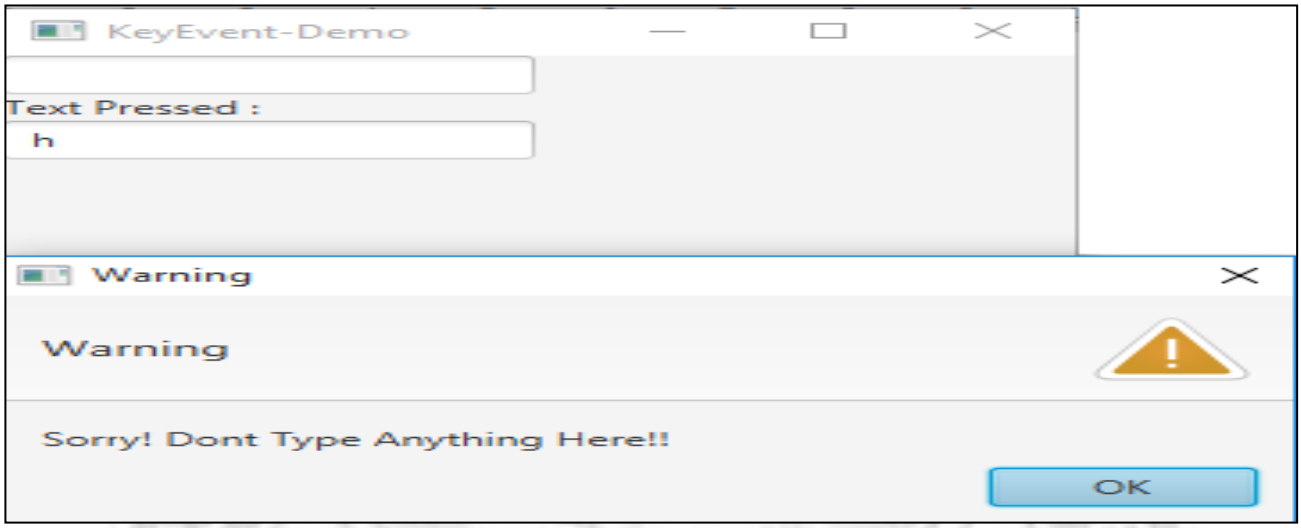


Figure 2: When backspace key is pressed in TextField 1



HANDLING MOUSE EVENTS

JavaFX Mouse Events are used to handle mouse events. The MouseEvents works when you Clicked, Dragged, or Pressed and etc. An object of the MouseEvent class represents a mouse events.

Types of Mouse Events in JavaFX

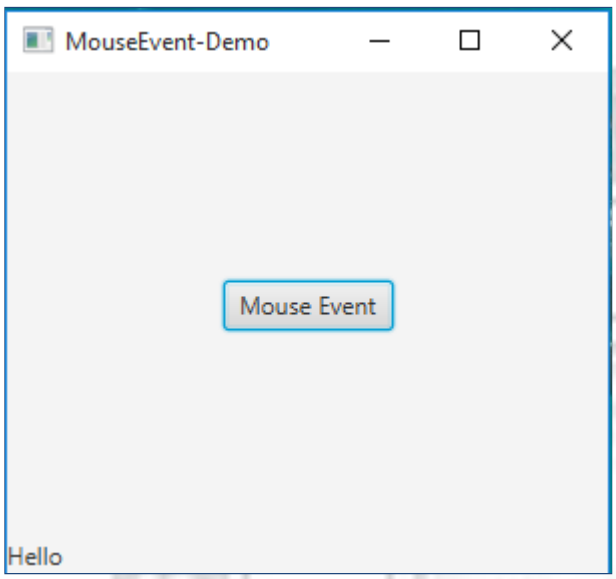
- **ANY** – This mouse event type is known as the supertype of all mouse event types. If you want your node to receive all types of events. This event type would be used for your handlers.
- **MOUSE_PRESSED** – When you press a mouse button, this event is triggered. The MouseButton enum defines three constants that represent a mouse button: NONE, PRIMARY, and SECONDARY. The MouseEvent class's getButton() method returns the mouse button that is responsible for the event.
- **MOUSE_RELEASED** – The event is triggered if you pressed and released a mouse button in the same node.
- **MOUSE_CLICKED** – This event will occur when you pressed and released a node.
- **MOUSE_MOVED** – Simply move your mouse without pressing any mouse buttons to generate this type of mouse event.
- **MOUSE_ENTERED** – This event occurs when the mouse or cursor enters the target node.
- **MOUSE_EXITED** – This event occurs when the mouse or cursor leaves or moved outside the target node.
- **MOUSE_DRAGGED** – This event occurs when you move the mouse with a pressed mouse button to a target node.

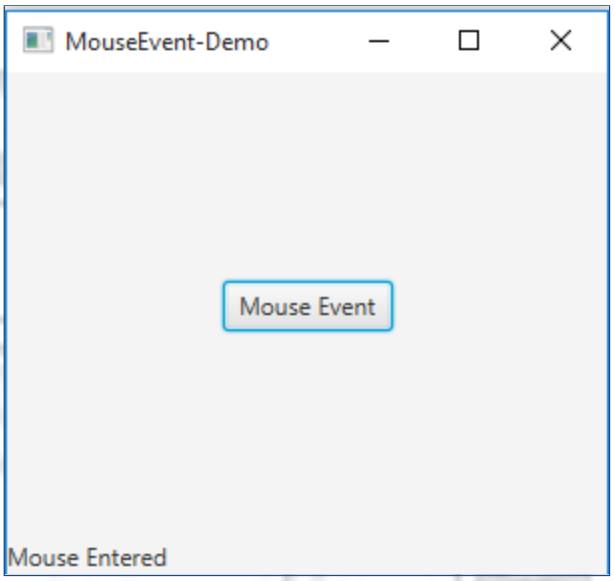
Example:

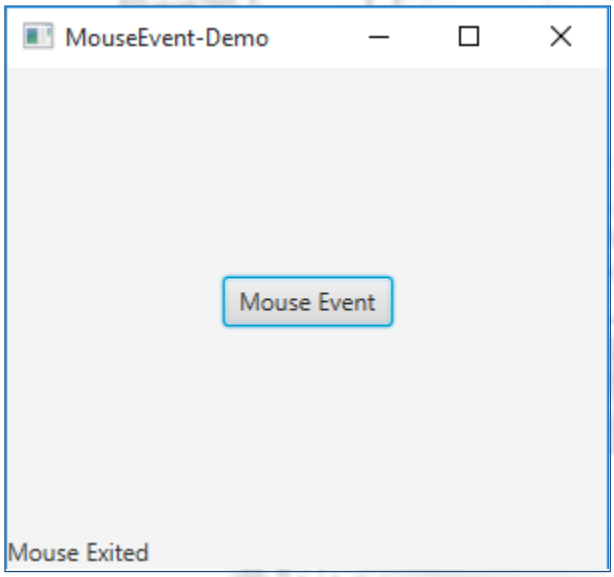
```
import javafx.application.Application;
import javafx.event.Event.*;
import javafx.scene.*;
import javafx.event.EventHandler;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.*;
```

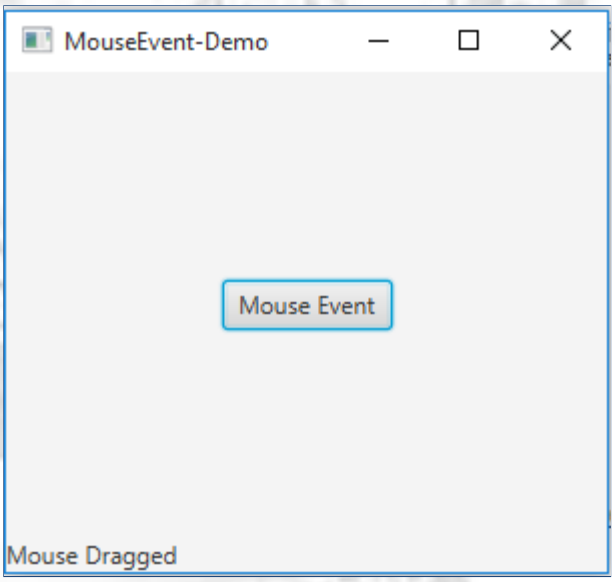
```
import javafx.stage.Stage;
import javafx.scene.control.*;
import java.util.*;
public class MouseEvents extends Application {
   @Override
   public void start(Stage primaryStage) {
      Button btn = new Button();
      Label status=new Label();
      btn.setText("Mouse Event");
      status.setText("Hello");
      btn.setOnMousePressed(new EventHandler<MouseEvent>() {
         public void handle(MouseEvent me) {
            status.setText("Mouse pressed");
         }
      });
      btn.setOnMouseEntered(e-> {
         status.setText("Mouse Entered");
      });
      btn.setOnMouseExited(e-> {
         status.setText("Mouse Exited");
      });
      btn.setOnMouseReleased(e-> {
         status.setText("Mouse Released");
      });
      BorderPane bp = new BorderPane();
      bp.setCenter(btn);
      bp.setBottom(status);
      Scene scene = new Scene(bp, 300, 250);
      scene.setOnMouseDragged(e-> {
         status.setText("Mouse Dragged");
      });
      primaryStage.setTitle("MouseEvent-Demo");
      primaryStage.setScene(scene);
      primaryStage.show();
   }
   public static void main(String[] args) {
      launch(args);
   }
}
```

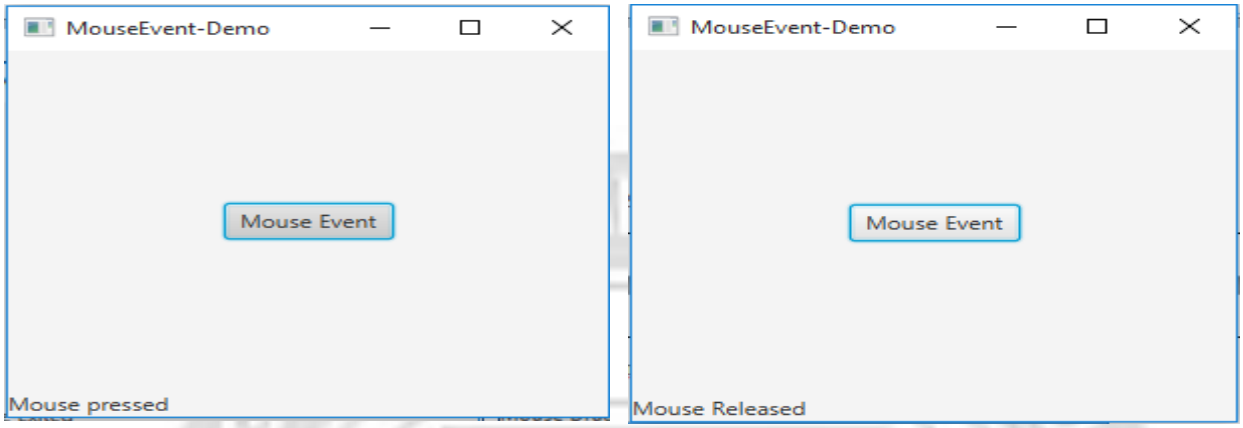
OUTPUT


MouseEvent-Demo
Mouse Event
Hello


MouseEvent-Demo
Mouse Event
Mouse Entered


MouseEvent-Demo
Mouse Event
Mouse Exited

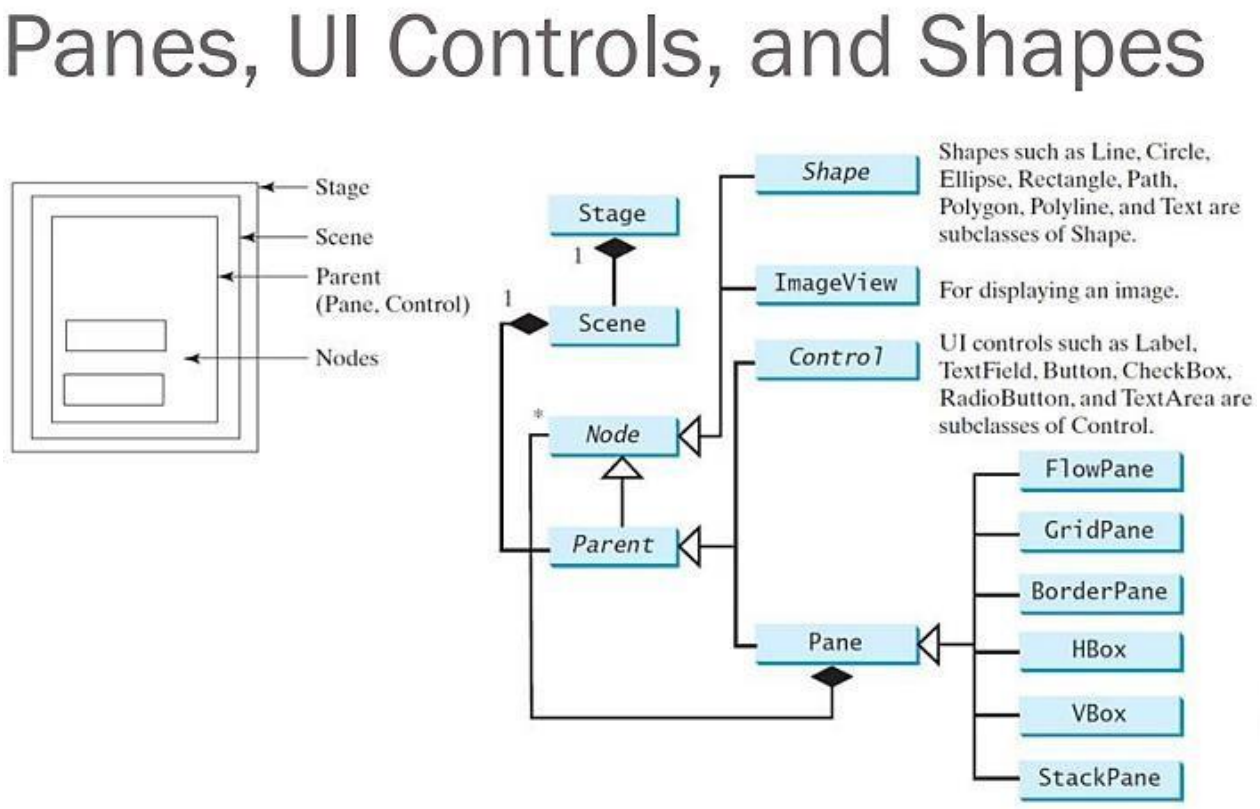

MouseEvent-Demo
Mouse Event
Mouse Dragged



5.4: JavaFX UI Controls

Every user interface considers the following three main aspects –

1. UI elements – These are the core visual elements which the user eventually sees and interacts with.
2. Layouts – They define how UI elements should be organized on the screen.
3. Behavior – These are events which occur when the user interacts with UI elements.



- JavaFX provides several classes in the package **javafx.scene.control.**

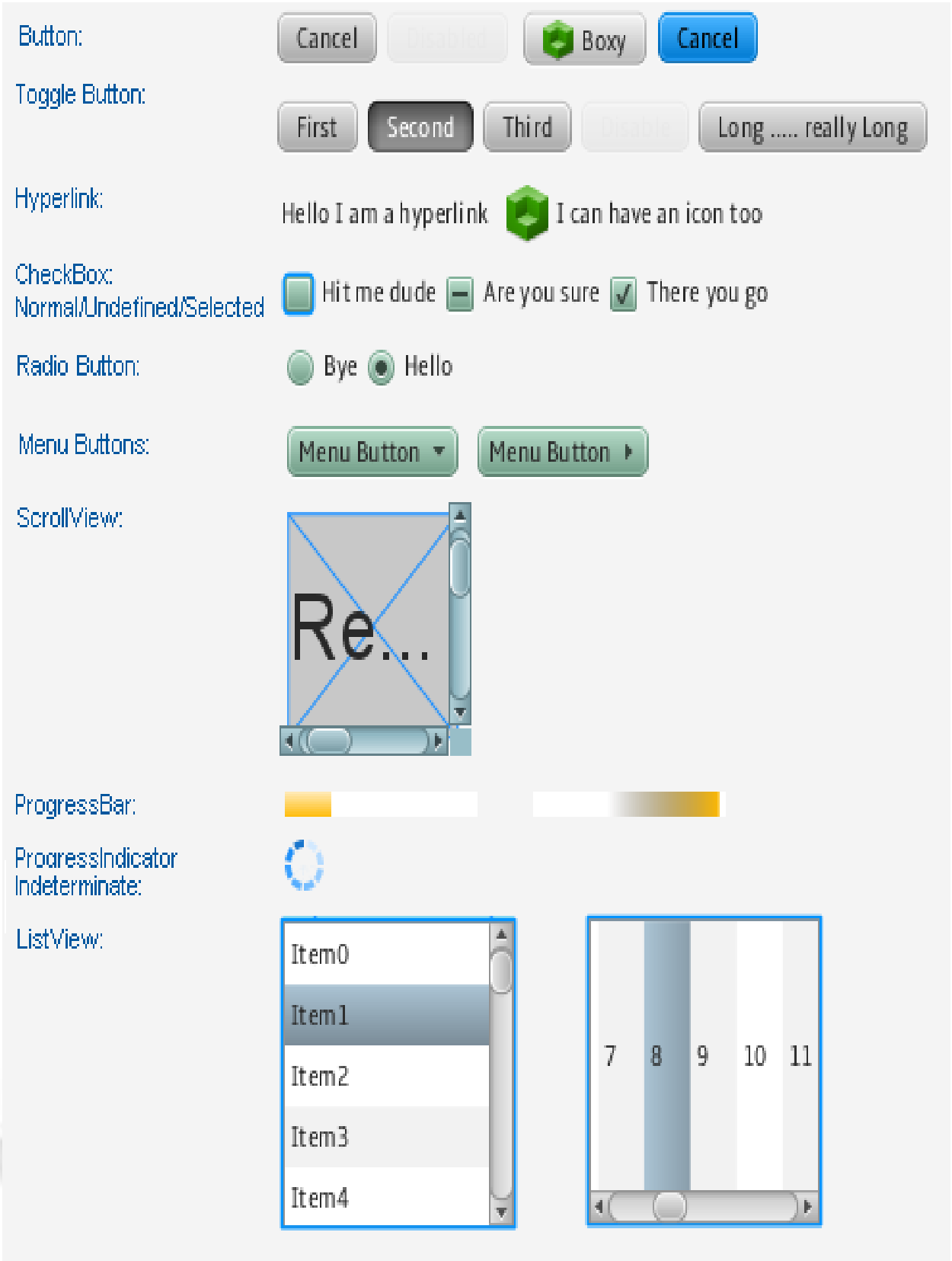


Figure: JavaFX UI Controls

S. No.	UI Control	Description	Constructors
1.	Label	Component that is used to define a simple text on the screen. It is an not editable text control.	new Label() new Label(String S, Node n) new Label(String s)
2.	TextField	Used to get the input from the user in the form of text. Allows to enter a limited quantity of text.	New TextField()
3.	CheckBox	Used to get the kind of information from the user which contains various choices. User marked the checkbox either on (true) or off(false).	new CheckBox() new CheckBox(String s)
4.	RadioButton	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.	new RadioButton() new RadioButton(String s)
5.	Button	Component that controls the function of the application.	new Button() new Button(String s)
6.	ComboBox	Shows a list of items out of which user can select at most one item	new ComboBox new ComboBox(ObservableList i)
7.	ChoiceBox	Shows a set of items and allows the user to select a single choice and it will show the currently selected item on the top. ChoiceBox by default has no selected item unless otherwise selected.	new ChoiceBox new ChoiceBox(ObservableList i)
8.	ListView	Enables users to choose one or more options from a predefined list of choices.	new ListView();
9.	ScrollPane	It provides a scrollable view of UI Elements. It is a container that has two scrollbars around the component it contains if the component is larger than the visible area of the ScrollPane. The scrollbars enable the user to scroll around the component shown inside the ScrollPane	new ScrollPane();
10.	ToggleButton	Special control having the ability to be selected. Basically, ToggleButton is rendered similarly to a Button but these two are the different types of Controls. A Button is a “command” button that invokes a function when clicked. But a ToggleButton is a control with a Boolean indicating whether it is selected.	new ToggleButton newToggleButton(String txt) new ToggleButton(String txt, Node graphic)

Selected User Actions and Handlers

User Action	*Source Object*	*Event Type Fired*	*Event Registration Method*
Click a button	`Button`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Press Enter in a text field	`TextField`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`RadioButton`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`CheckBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Select a new item	`ComboBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Mouse pressed	`Node, Scene`	`MouseEvent`	`setOnMousePressed(EventHandler<MouseEvent>)`
Mouse released			`setOnMouseReleased(EventHandler<MouseEvent>)`
Mouse clicked			`setOnMouseClicked(EventHandler<MouseEvent>)`
Mouse entered			`setOnMouseEntered(EventHandler<MouseEvent>)`
Mouse exited			`setOnMouseExited(EventHandler<MouseEvent>)`
Mouse moved			`setOnMouseMoved(EventHandler<MouseEvent>)`
Mouse dragged			`setOnMouseDragged(EventHandler<MouseEvent>)`
Key pressed	`Node, Scene`	`KeyEvent`	`setOnKeyPressed(EventHandler<KeyEvent>)`
Key released			`setOnKeyReleased(EventHandler<KeyEvent>)`
Key typed			`setOnKeyTyped(EventHandler<KeyEvent>)`

Example : JavaFX program for Simple Registration form using UI Controls:

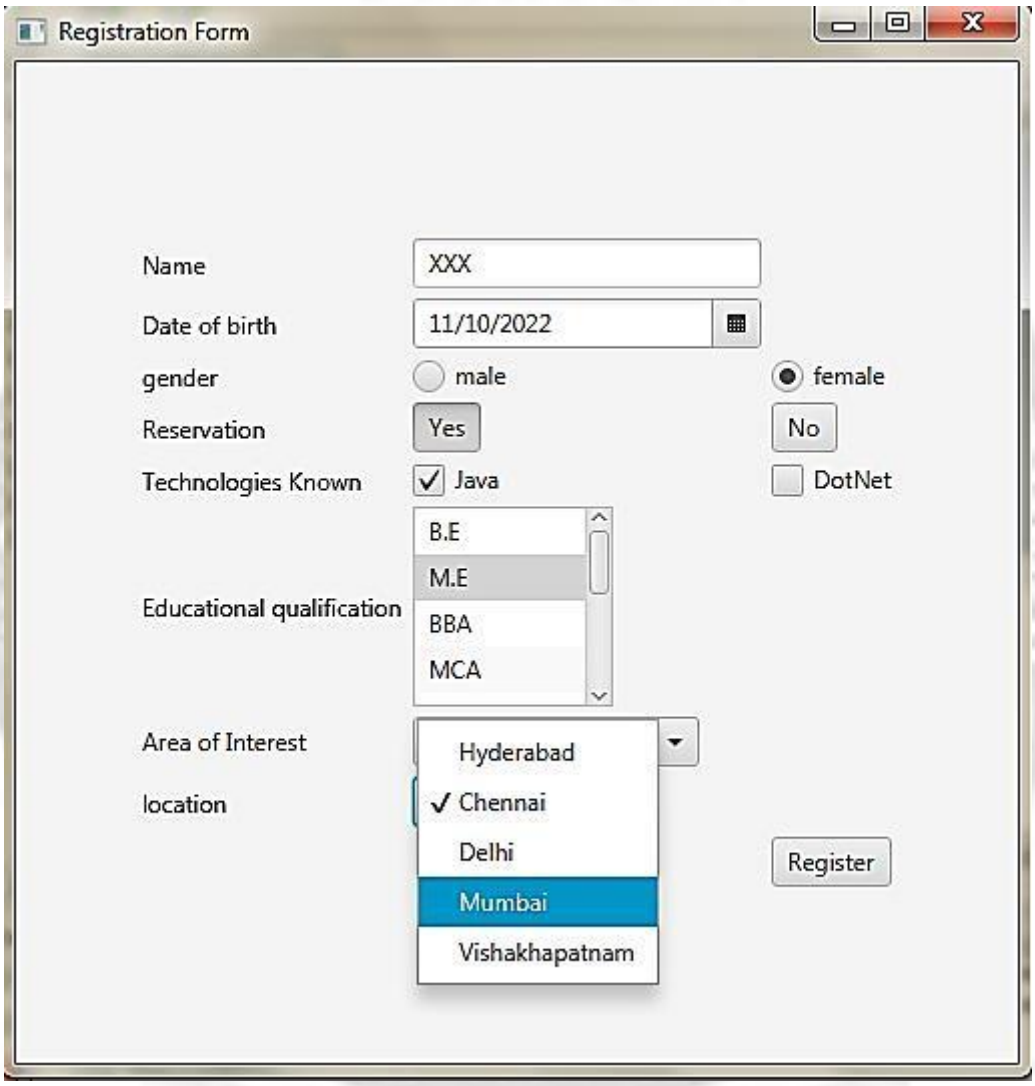
```
import javafx.application.Application;
import javafx.collections.*;

import javafx.geometry.Insets;
import javafx.geometry.Pos;

import javafx.scene.image.*;

import javafx.scene.Scene;
import javafx.scene.control.*;

import javafx.scene.layout.*;
import javafx.scene.text.Text;

import javafx.stage.Stage;

public class JavaFXControlDemo extends Application {
  @Override
  public void start(Stage stage)
{
```

```
//Label for name
Text nameLabel = new Text("Name");

//Text field for name
TextField nameText = new TextField();

//Label for date of birth
Text dobLabel = new Text("Date of birth");

//date picker to choose date
DatePicker datePicker = new DatePicker();

//Label for gender
Text genderLabel = new Text("gender");

//Toggle group of radio buttons
ToggleGroup groupGender = new ToggleGroup();
RadioButton maleRadio = new RadioButton("male");
maleRadio.setToggleGroup(groupGender);
RadioButton femaleRadio = new RadioButton("female");
femaleRadio.setToggleGroup(groupGender);

//Label for reservation
Text reservationLabel = new Text("Reservation");

//Toggle button for reservation
ToggleButton yes = new ToggleButton("Yes");
ToggleButton no = new ToggleButton("No");
ToggleGroup groupReservation = new ToggleGroup();
yes.setToggleGroup(groupReservation);
no.setToggleGroup(groupReservation);

//Label for technologies known
Text technologiesLabel = new Text("Technologies Known");

//check box for education
CheckBox javaCheckBox = new CheckBox("Java");
javaCheckBox.setIndeterminate(false);

//check box for education
CheckBox dotnetCheckBox = new CheckBox("DotNet");
javaCheckBox.setIndeterminate(false);

//Label for education
Text educationLabel = new Text("Educational qualification");
```

```
    //list View for educational qualification
    ObservableList<String> names = FXCollections.observableArrayList(
      "B.E","M.E","BBA","MCA", "MBA", "Vocational", "M.TECH", "Mphil",
"Phd");
    ListView<String> educationListView = new ListView<String>(names);
    educationListView.setMaxSize(100, 100);

educationListView.getSelectionModel().setSelectionMode(SelectionMode.MU
LTIPLE);

    Label interest=new Label("Area of Interest");
    ComboBox AoI=new ComboBox();
    AoI.getItems().addAll("Android App. Dev.", "IoS App. Dev.", "FUll Stack
Dev.", "Azure FrmWork", "AWS", "Web Dev.", "Ui/Ux Design");
    AoI.setVisibleRowCount(3);

    //Label for location
    Text locationLabel = new Text("location");

    //Choice box for location
    ChoiceBox locationchoiceBox = new ChoiceBox();
    locationchoiceBox.getItems().addAll
      ("Hyderabad", "Chennai", "Delhi", "Mumbai", "Vishakhapatnam");

    //Label for register
    Button buttonRegister = new Button("Register");

    //Creating a Grid Pane
    GridPane gridPane = new GridPane();

    //Setting size for the pane
    gridPane.setMinSize(500, 500);

    //Setting the padding
    gridPane.setPadding(new Insets(10, 10, 10, 10));

    //Setting the vertical and horizontal gaps between the columns
    gridPane.setVgap(5);
    gridPane.setHgap(5);

    //Setting the Grid alignment
    gridPane.setAlignment(Pos.CENTER);

    //Arranging all the nodes in the grid
    gridPane.add(nameLabel, 0, 0);
    gridPane.add(nameText, 1, 0);

    gridPane.add(dobLabel, 0, 1);
```

```
                gridPane.add(datePicker, 1, 1);

                gridPane.add(genderLabel, 0, 2);
                gridPane.add(maleRadio, 1, 2);
                gridPane.add(femaleRadio, 2, 2);
                gridPane.add(reservationLabel, 0, 3);
                gridPane.add(yes, 1, 3);
                gridPane.add(no, 2, 3);

                gridPane.add(technologiesLabel, 0, 4);
                gridPane.add(javaCheckBox, 1, 4);
                gridPane.add(dotnetCheckBox, 2, 4);

                gridPane.add(educationLabel, 0, 5);
                gridPane.add(educationListView, 1, 5);

                gridPane.add(interest,0,6);
                gridPane.add(AoI,1,6);

                gridPane.add(locationLabel, 0, 7);
                gridPane.add(locationchoiceBox, 1, 7);

                gridPane.add(buttonRegister, 2, 8);

                Scene scene = new Scene(gridPane);

                //Setting title to the Stage
                stage.setTitle("Registration Form");

                //Adding scene to the stage
                stage.setScene(scene);

                //Displaying the contents of the stage
                stage.show();
    }

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

OUTPUT:


Registration Form
Name
XXX
Date of birth
11/10/2022
gender
male
female
Reservation
Yes
No
Technologies Known
Java
DotNet
B.E
M.E
BBA
MCA
Educational qualification
Area of Interest
location
Hyderabad
Chennai
Delhi
Mumbai
Vishakhapatnam
Register