Unit – 5: JAVAFX EVENT HANDLING, CONTROLS AND COMPONENTS

UNIT V JAVAFX EVENT HANDLING, CONTROLS AND COMPONENTS

JAVAFX Events and Controls: Event Basics – Handling Key and Mouse Events. Controls: Checkbox, ToggleButton – RadioButtons – ListView – ComboBox – ChoiceBox – Text Controls – ScrollPane. Layouts – FlowPane – HBox and VBox – BorderPane – StackPane – GridPane. Menus – Basics – Menu – Menu bars – MenuItem.

5.1: Introduction to JavaFX

What is JavaFX?

JavaFX is a set of graphics and media packages that enable developers to design, create, test, debug, and deploy desktop applications and Rich Internet Applications (RIA) that operate consistently across diverse platforms. The applications built in JavaFX can run on multiple platforms including Web, Mobile, and Desktops.

Features of JavaFX:

Feature	Description
Java Library	It consists of many classes and interfaces that are written in Java.
FXML	FXML is the XML based Declarative markup language. The coding can be done in FXML to provide the more enhanced GUI to the user.
Scene Builder	Scene Builder generates FXML mark-up which can be ported to an IDE.
Web view	Web View uses WebKitHTML technology to embed web pages into the Java Applications.
Built in UI controls	Built-in controls are not dependent on operating system. The UI components are used to develop a full featured application.
CSS like styling	JavaFX code can be embedded with the CSS to improve the style and view of the application.
Swing interoperability	The JavaFX applications can be embedded with swing code using the Swing Node class. We can update the existing swing application with the powerful features of JavaFX.
Canvas API	Canvas API provides the methods for drawing directly in an area of a JavaFX scene.
Rich Set of APIs	JavaFX provides a rich set of API's to develop GUI applications.

Integrated Graphics Library	It is provided to deal with 2D and 3D graphics.
Graphics Pipeline	JavaFX graphics are based on Graphics rendered pipeline(prism). It offers smooth graphics which are hardware accelerated.
High Performance Media Engine	The media pipeline supports the playback of web multimedia on a low latency. It is based on a Gstreamer Multimedia framework.
Self-contained application deployment model	Self-Contained application packages have all of the application resources and a private copy of Java and JavaFX Runtime.

5.1.1: JavaFX Application Structure:

A JavaFX application will have three major components namely

1) Stage
2) Scene and
3) Nodes

as shown in the following diagram.

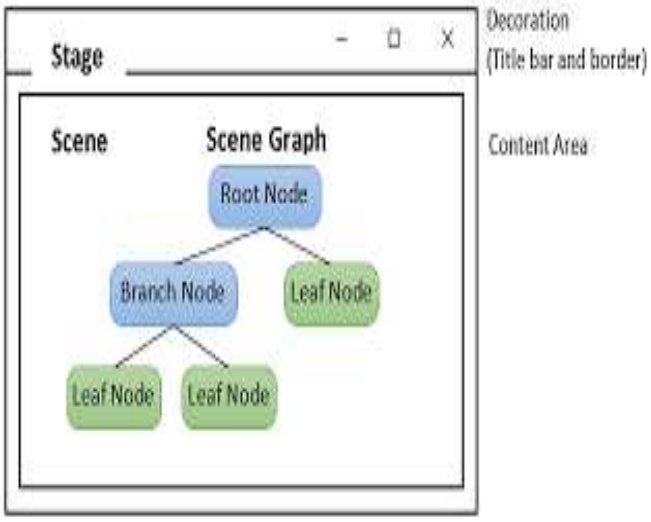


Figure: JavaFX App.lication Structure

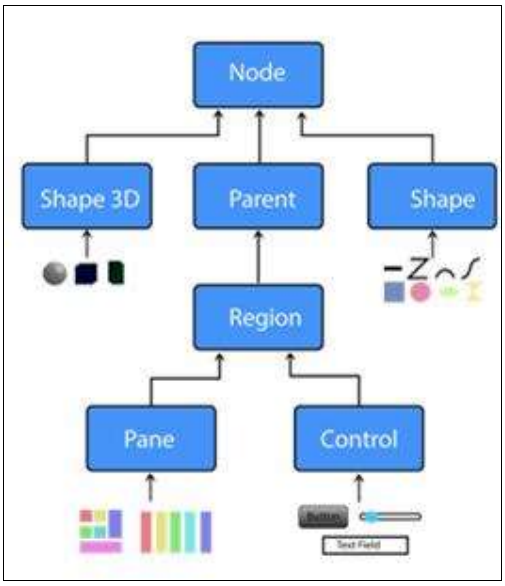


Figure: Scene Graph and Nodes

1) Stage

- ✓ Stage(a window) in a JavaFX application is similar to the Frame in a Swing Application. It acts like a container for all the JavaFX objects.
- ✓ Primary Stage is created internally by the platform. Other stages can further be created by the application.

- ✓ A stage has two parameters determining its position namely **Width** and **Height**. It is divided as Content Area and Decorations (Title Bar and Borders).
- ✓ There are five types of stages available –
  - Decorated
  - Undecorated
  - Transparent
  - Unified
  - Utility
- ✓ We have to call the **show()** method to display the contents of a stage.

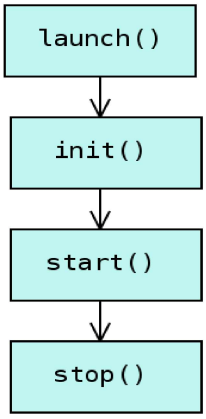
2) Scene

- ✓ A scene represents the physical contents of a JavaFX application. It contains all the contents of a scene graph.
- ✓ The class **Scene** of the package **javafx.scene** represents the scene object. At an instance, the scene object is added to only one stage.

3) Scene Graph and Nodes

- ✓ A **scene graph** is a tree-like data structure (hierarchical) representing the contents of a scene. In contrast, a **node** is a visual/graphical object of a scene graph.
- ✓ A node may include –
  - **Geometrical** (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon, etc.
  - **UI Controls** such as – Button, Checkbox, Choice Box, Text Area, etc.
  - **Containers** (Layout Panes) such as Border Pane, Grid Pane, Flow Pane, etc.
  - **Media elements** such as Audio, Video and Image Objects.
- ✓ A node is of three types –
  - **Root Node –** The first Scene Graph is known as the Root node.
  - **Branch Node/Parent Node –** the node with child nodes are known as branch/parent nodes. The parent nodes will be of the following types –
    - **Group** – A group node is a collective node that contains a list of children nodes. Whenever the group node is rendered, all its child nodes are rendered in order. Any transformation, effect state applied on the group will be applied to all the child nodes.
    - **Region –** It is the base class of all the JavaFX Node based UI Controls, such as Chart, Pane and Control.
    - **WebView –** This node manages the web engine and displays its contents.
  - **Leaf Node –** The node without child nodes is known as the leaf node.

5.1.2: Lifecycle of a JavaFX Application:



The JavaFX Application class has three life cycle methods, which are –

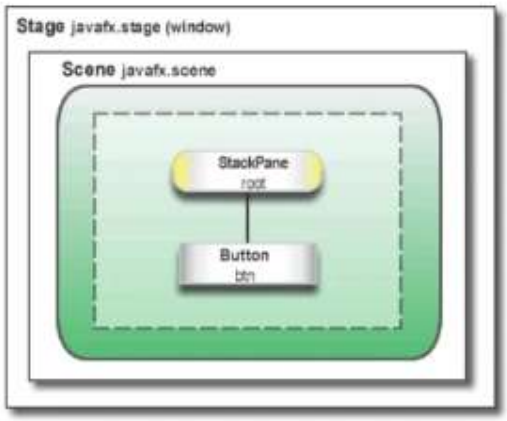
1) **launch() -** to launch JavaFX application.
2) **init() –** An empty method which can be overridden, but you cannot create a stage or scene in this method.
3) **start() –** The entry point method where the JavaFX graphics code is to be written.
4) **stop() –** An empty method which can be overridden, here we can write the logic to stop the application.

General Rules for writing JavaFX Application:

- ✓ A JavaFX Application must extend javafx.application.Application.
- ✓ The main() method should call Application.launch()
- ✓ The start() method is the main entry point for all JavaFX applications
  - Start() is called when a Stage is connected to the Operating System's window
- ✓ The content of the scene is represented as a hierarchical scene graph of nodes:
  - Stage is the top-level JavaFX Container
  - Scene is the container for all content

Minimal example

```
public class HelloWorld extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello World!");

        StackPane root = new StackPane();

        Button btn = new Button();
        btn.setText("Say 'Hello World'");

        root.getChildren().add(btn);

        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```



5.2 JavaFX Events

5.2.1: 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.

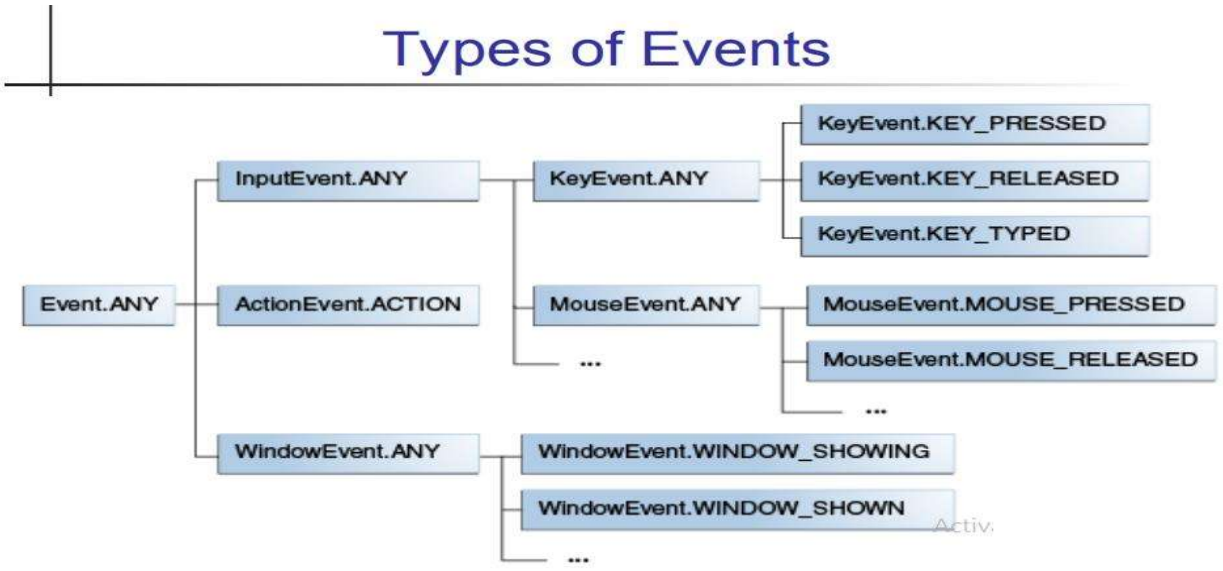
Example: Button clicks by user, key press on the application etc.

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.

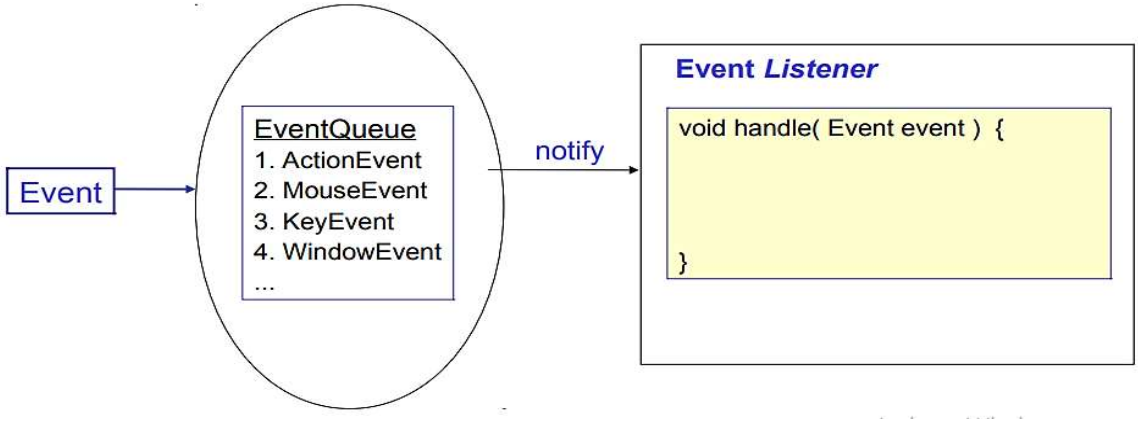
Examples of Events:

- **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



5.2.2: 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 executes. 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

5.3.1: 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);
    }
}
```

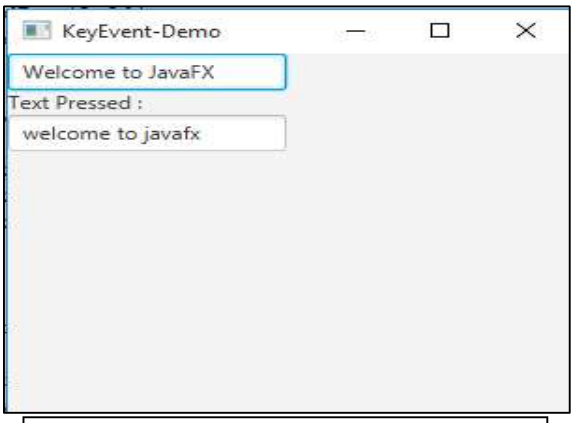


Figure 1: When a key is pressed in TextField 1

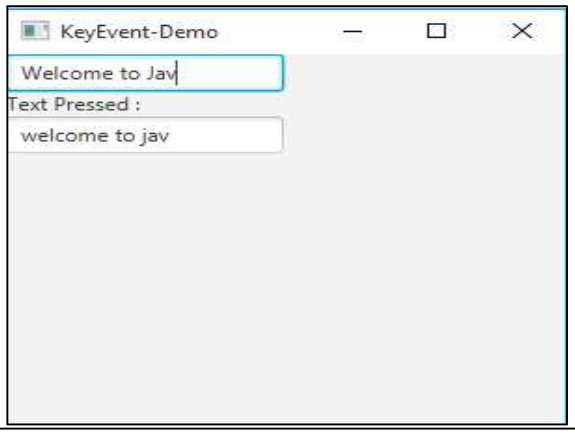


Figure 2: When backspace key is pressed in TextField 1

5.3.2: 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.

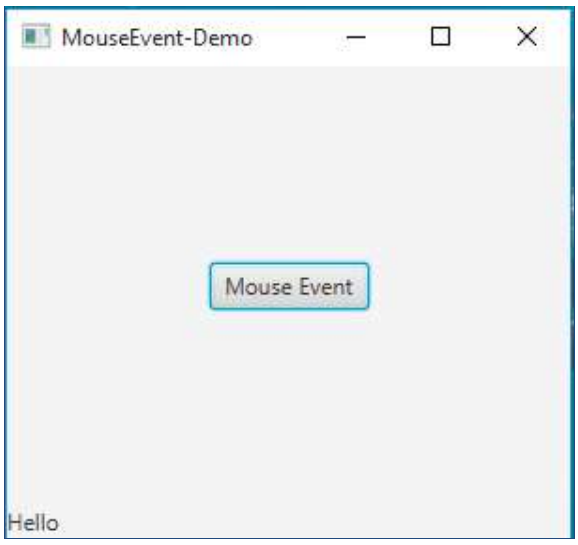
Example:

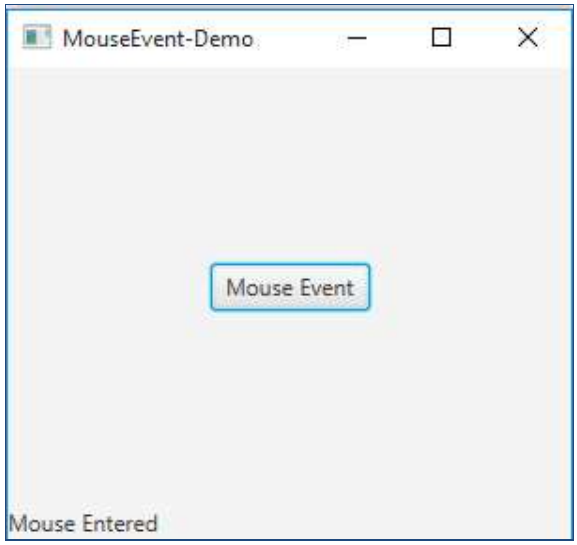
```
import javafx.application.Application;
import javafx.event.Event.*;
import javafx.scene.*;
import javafx.event.EventHandler;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.*;
import javafx.stage.Stage;
import javafx.scene.control.*;
import java.util.*;
```

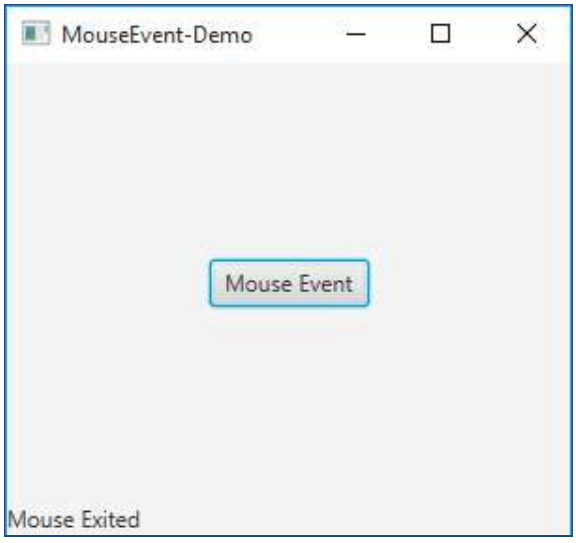
```
public class MouseEvents extends Application {

   @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();
   }
```

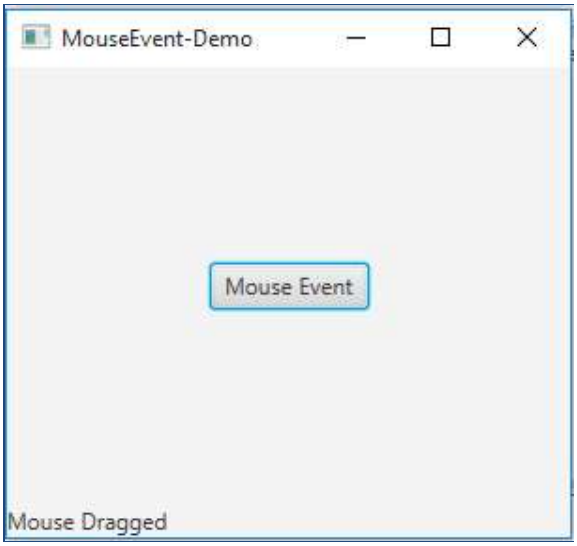
```
    public static void main(String[] args) {
        launch(args);
    }
}
```

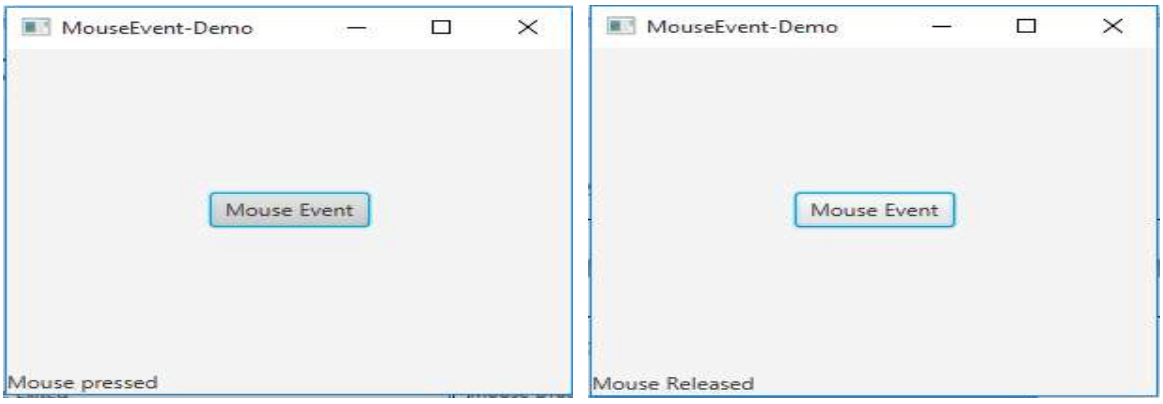
OUTPUT







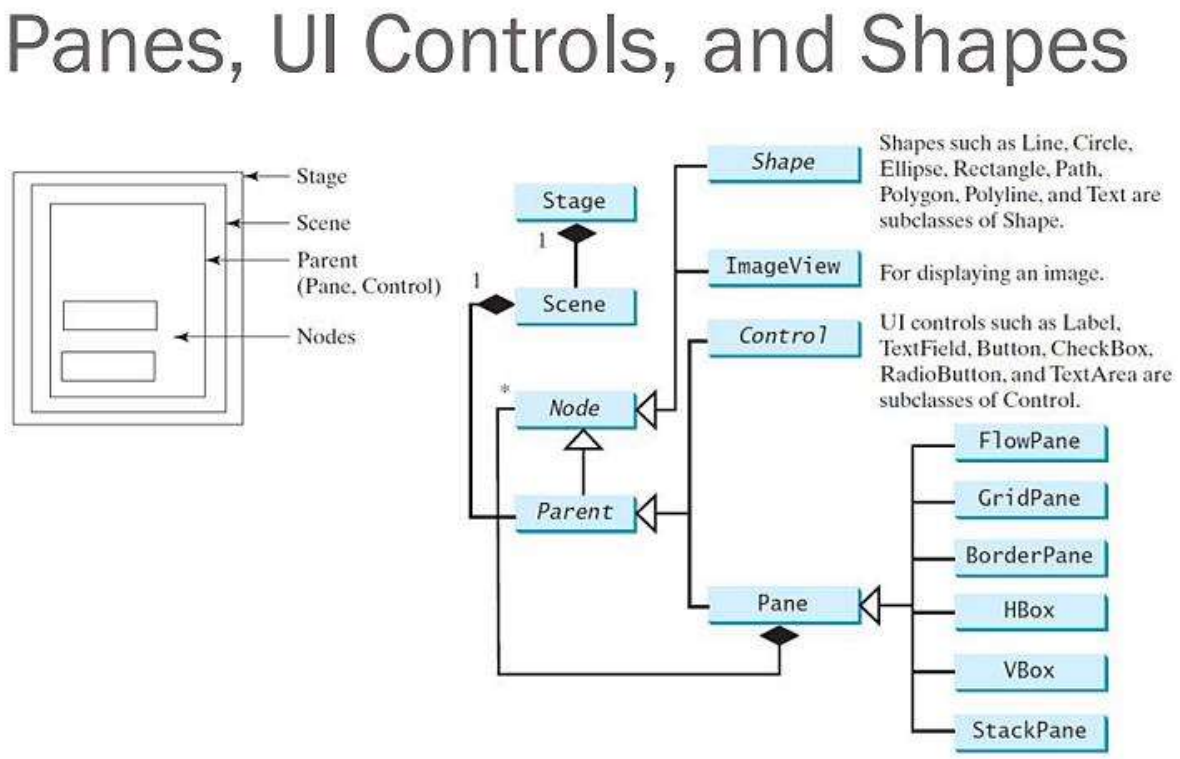




5.4: JavaFX UI Controls

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.**

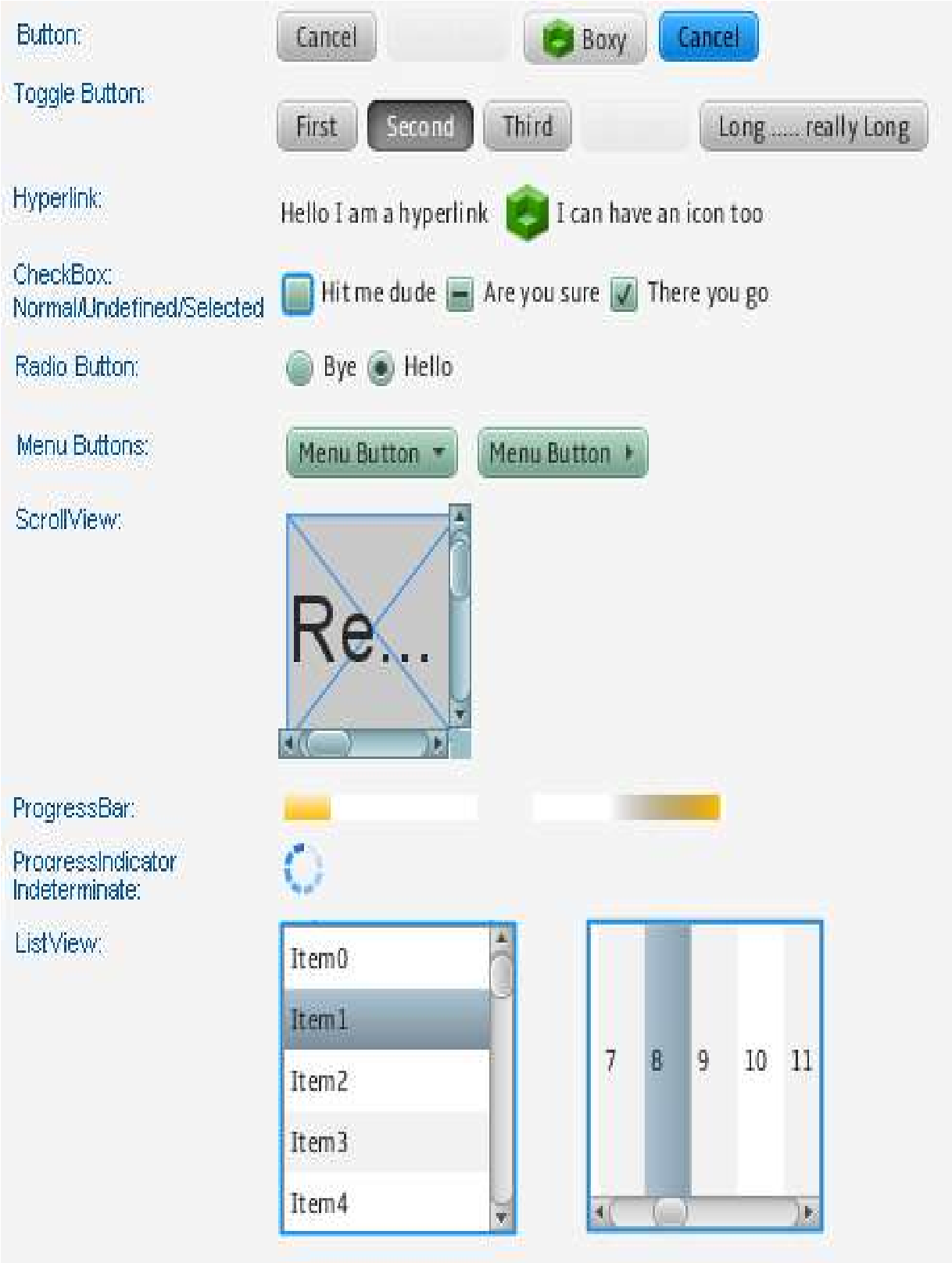


Figure: JavaFX UI Controls

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)
{
```

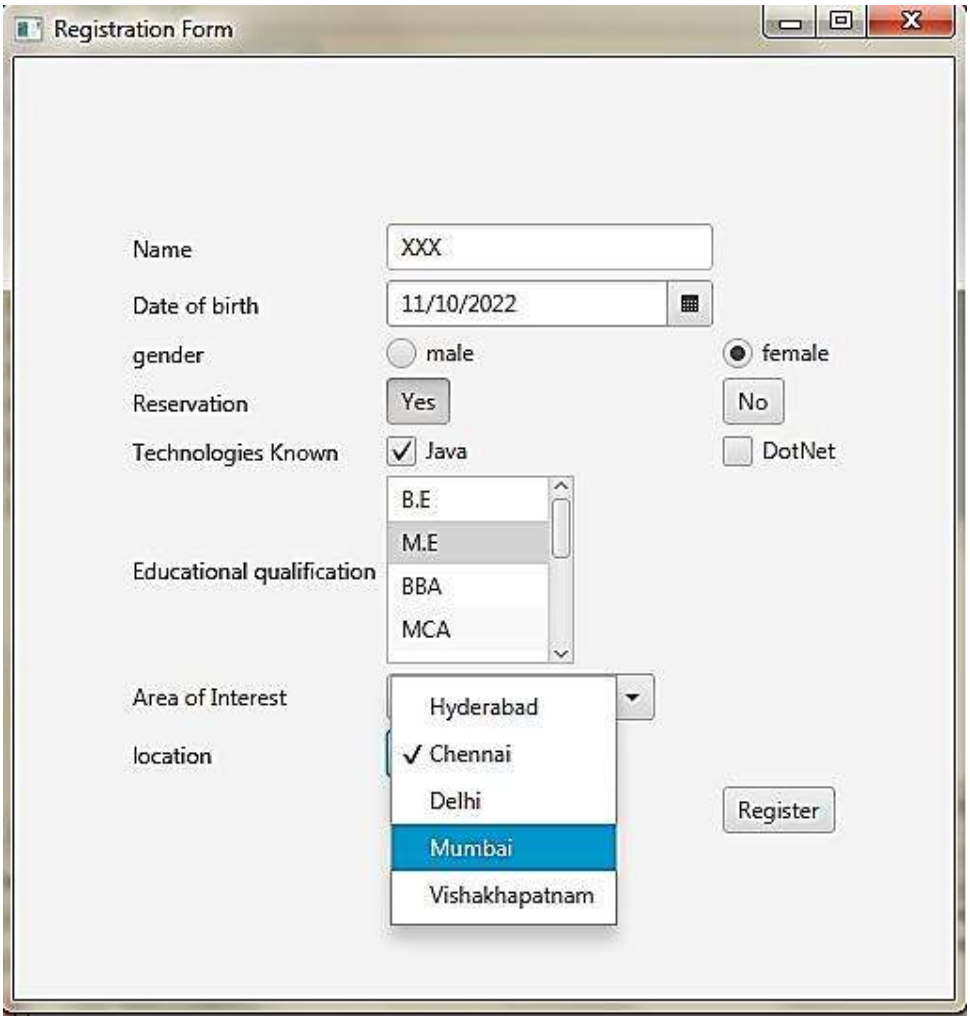
```
//Label for name
Text nameLabel = new Text("Name");

//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:



5.5: Layouts – FlowPane – HBox and VBox – BorderPane – StackPane – GridPane.

In JavaFX, Layout defines the way in which the components are to be seen on the stage. It basically organizes the scene-graph nodes.

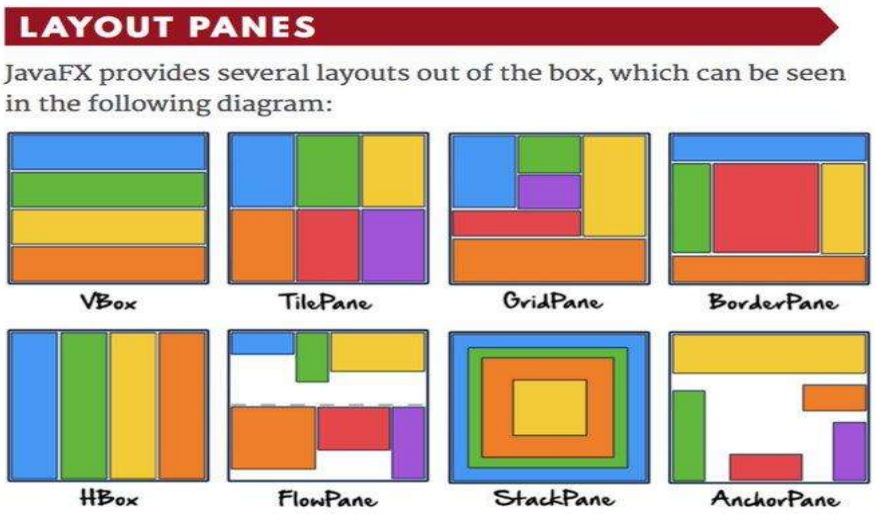
Layout Panes: Layout panes are containers which are used for flexible and dynamic arrangements of UI controls within a scene graph of a JavaFX application.

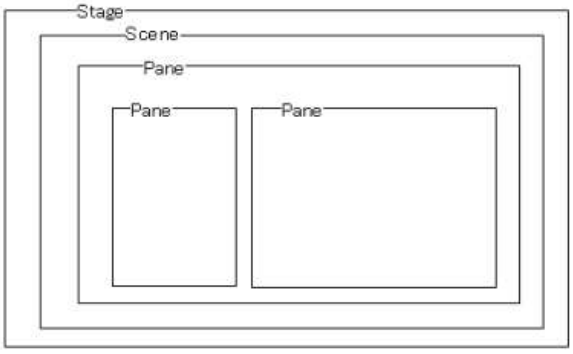
Package used: javaFX.scene.layout package

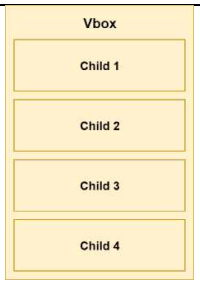
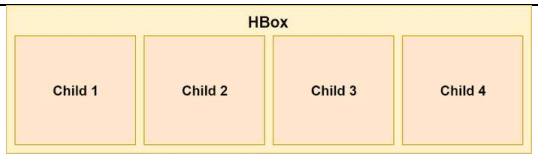
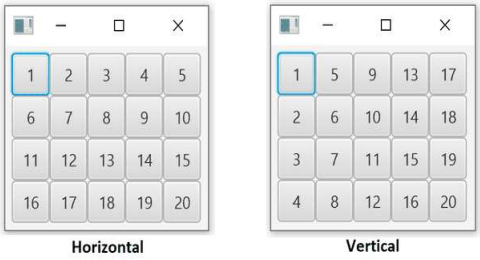
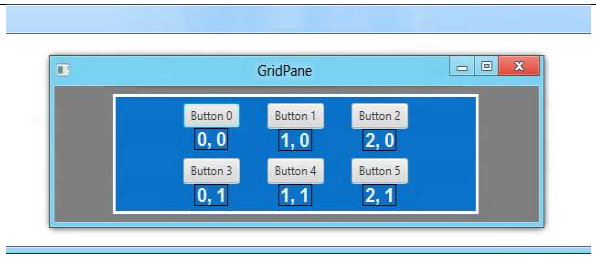
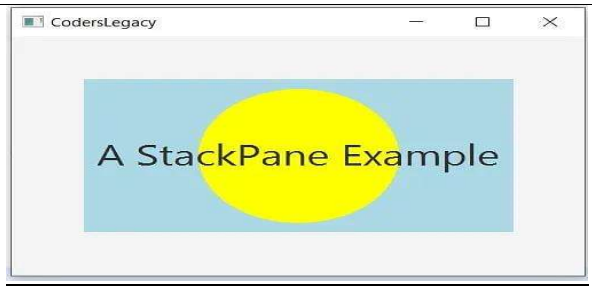
JavaFX provides various built-in Layouts that are

1. Pane
2. VBox
3. HBox
4. BorderPane
5. FlowPane
6. GridPane
7. StackPane.

JavaFX provides many types of panes for organizing nodes in a container:



Class	Description	Representation
Pane	Base class for layout panes. It containes the getChildren() method for returning a list of nodes in the pane.	

VBox	Places the nodes in a single column	
HBox	Places the nodes in a single row	
BorderPane	Places the nodes in the top, right, bottom, left and center regions	
FlowPane	Places the nodes row-by-row horizontally or column-by-column vertically	
GridPane	Places the nodes in the cells in a two-dimensional grid(like matrix)	
StackPane	Places the nodes on top of each other in the center of the pane	

Methods and Properties of different layouts:

<table>
<tr><th>Layout</th><th>Constructors</th><th>Methods/Properties</th></tr>
<tr><td>VBox</td><td>1. VBox() : creates layout with 0 spacing
2. Vbox(Double spacing) : creates layout with a spacing value of double type
3. Vbox(Double spacing, Node? children) : creates a layout with the specified spacing among the specified child nodes
4. Vbox(Node? children) : creates a layout with the specified nodes having 0 spacing among them</td><td>
<table>
<tr><th>Property</th><th>Description</th><th>Setter Methods</th></tr>
<tr><td>Alignment</td><td>This property is for the alignment of the nodes.</td><td>setAlignement(Double)</td></tr>
<tr><td>FillWidth</td><td>This property is of the boolean type. The Widtht of resizeable nodes can be made equal to the Width of the VBox by setting this property to true.</td><td>setFillWidth(boolean)</td></tr>
<tr><td>Spacing</td><td>This property is to set some spacing among the nodes of VBox.</td><td>setSpacing(Double)</td></tr>
</table>
</td></tr>
<tr><td>HBox</td><td>1. new HBox() : create HBox layout with 0 spacing
2. new Hbox(Double spacing) : create HBox layout with a spacing value</td><td>
<table>
<tr><th>Property</th><th>Description</th><th>Setter Methods</th></tr>
<tr><td>Alignment</td><td>This represents the alignment of the nodes.</td><td>setAlignment(Double)</td></tr>
<tr><td>fillHeight</td><td>This is a boolean property. If you set this property to true the height of the nodes will become equal to the height of the HBox.</td><td>setFillHeight(Double)</td></tr>
<tr><td>spacing</td><td>This represents the space between the nodes in the HBox. It is of double type.</td><td>setSpacing(Double)</td></tr>
</table>
</td></tr>
<tr><td>BorderPane</td><td>1. BorderPane() :- create the empty layout
2. BorderPane(Node Center):- create the layout with the center node
3. BorderPane(Node Center, Node top, Node right, Node bottom, Node left) :- create the layout with all the nodes</td><td>
<table>
<tr><th>Type</th><th>Property</th><th>Setter Methods</th><th>Description</th></tr>
<tr><td>Node</td><td>Bottom</td><td>setBottom()</td><td>Add the node to the bottom of the screen</td></tr>
<tr><td>Node</td><td>Centre</td><td>setCentre()</td><td>Add the node to the centre of the screen</td></tr>
<tr><td>Node</td><td>Left</td><td>setLeft()</td><td>Add the node to the left of the screen</td></tr>
<tr><td>Node</td><td>Right</td><td>setRight()</td><td>Add the node to the right of the screen</td></tr>
<tr><td>Node</td><td>Top</td><td>setTop()</td><td>Add the node to the top of the screen</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td>FlowPane</td>
<td>1. FlowPane()
2. FlowPane(Double Hgap, Double Vgap)
3. FlowPane(Double Hgap, Double Vgap, Node? children)
4. FlowPane(Node... Children)
5. FlowPane(Orientation orientation)
6. FlowPane(Orientation orientation, double Hgap, Double Vgap)</td>
<td>
<table>
<tr><th>Property</th><th>Description</th><th>Setter Methods</th></tr>
<tr><td>alignment</td><td>The overall alignment of the flowpane's content.</td><td>setAlignment(Pos value)</td></tr>
<tr><td>columnHalignment</td><td>The horizontal alignment of nodes within the columns.</td><td>setColumnHalignment(HPos Value)</td></tr>
<tr><td>hgap</td><td>Horizontal gap between the columns.</td><td>setHgap(Double value)</td></tr>
<tr><td>orientation</td><td>Orientation of the flowpane</td><td>setOrientation(Orientation value)</td></tr>
<tr><td>prefWrapLength</td><td>The preferred height or width where content should wrap in the horizontal or vertical flowpane.</td><td>setPrefWrapLength(double value)</td></tr>
<tr><td>rowValignment</td><td>The vertical alignment of the nodes within the rows.</td><td>setRowValignment(VPos value)</td></tr>
<tr><td>vgap</td><td>The vertical gap among the rows</td><td>setVgap(Double value)</td></tr>
</table>
</td>
</tr>
<tr>
<td>GridPane</td>
<td>Public GridPane(): creates a gridpane with 0 hgap/vgap.</td>
<td>
<table>
<tr><th>Property</th><th>Description</th><th>Setter Methods</th></tr>
<tr><td>alignment</td><td>Represents the alignment of the grid within the GridPane.</td><td>setAlignment(Pos value)</td></tr>
<tr><td>gridLinesVisible</td><td>This property is intended for debugging. Lines can be displayed to show the gidpane's rows and columns by setting this property to true.</td><td>setGridLinesVisible(Boolean value)</td></tr>
<tr><td>hgap</td><td>Horizontal gaps among the columns</td><td>setHgap(Double value)</td></tr>
<tr><td>vgap</td><td>Vertical gaps among the rows</td><td>setVgap(Double value)</td></tr>
</table>
</td>
</tr>
<tr>
<td>StackPane</td>
<td>1. StackPane()
2. StackPane(Node? Children)</td>
<td>
<table>
<tr><th>Property</th><th>Description</th><th>Setter Methods</th></tr>
<tr><td>alignment</td><td>It represents the default alignment of children within the StackPane's width and height</td><td>setAlignment(Node child, Pos value)</td></tr>
</table>
</td>
</tr>
</table>

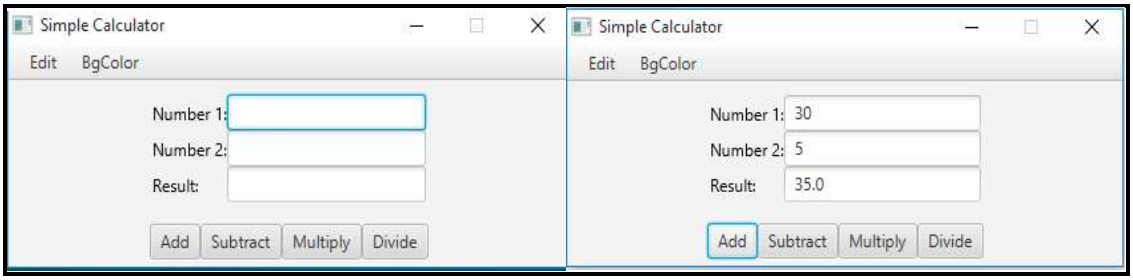
Example: Program for Layouts, Menus and MenuBars, Event Handling

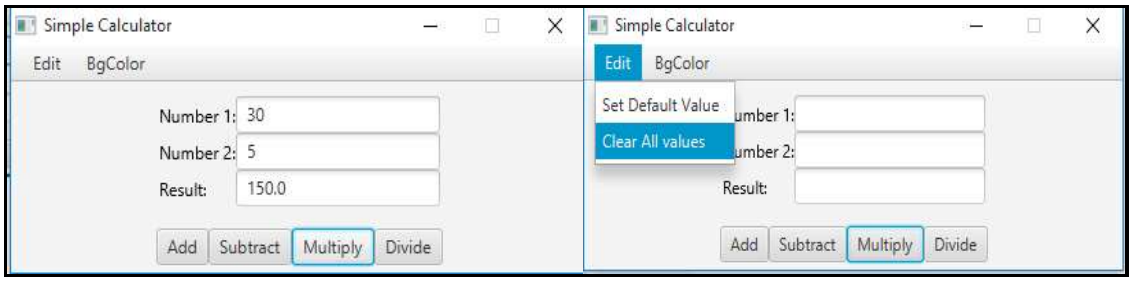
```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class JavaFXApplication1 extends Application {
  @Override//from ww w . ja v a2s .co m
  public void start(Stage primaryStage) {
    // Create the top section of the UI
    Text tNumber1 = new Text("Number 1:");
    Text tNumber2 = new Text("Number 2:");
    Text tResult = new Text("Result:");
    TextField tfNumber1 = new TextField();
    TextField tfNumber2 = new TextField();
    TextField tfResult = new TextField();
    tfResult.setEditable(false);

    Menu me=new Menu("Edit");
    // create menuitems
     MenuItem m1 = new MenuItem("Set Default Value");
     MenuItem m2 = new MenuItem("Clear All values");

     // add menu items to menu
     me.getItems().add(m1);
     me.getItems().add(m2);

    Menu mc=new Menu("Bg_Color");
    MenuItem c1 = new MenuItem("Red");
    MenuItem c2 = new MenuItem("Green");

    mc.getItems().addAll(c1,c2);
    MenuBar mb = new MenuBar();
```

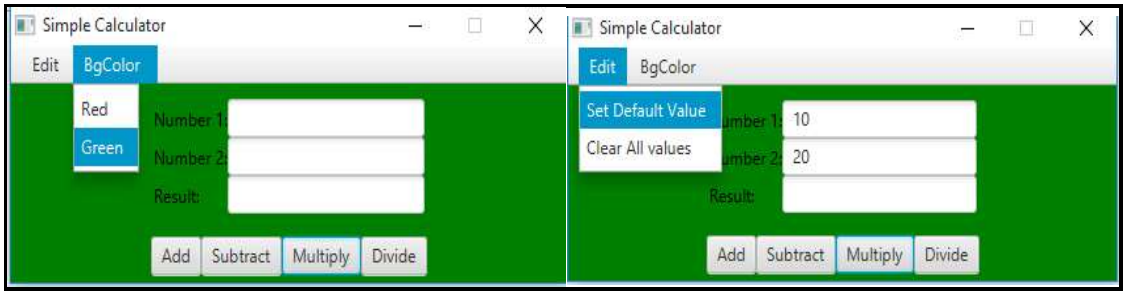
```
    // add menu to menubar
  mb.getMenus().add(me);
  mb.getMenus().add(mc);

  VBox vb=new VBox(mb);

  m1.setOnAction(e -> {
     tfNumber1.setText("10");
     tfNumber2.setText("20");
  });

  m2.setOnAction(e ->{
     tfNumber1.setText("");
     tfNumber2.setText("");
     tfResult.setText("");
  });

 // Create the bottom section of the UI
  Button btAdd = new Button("Add");
  Button btSubtract = new Button("Subtract");
  Button btMultiply = new Button("Multiply");
  Button btDivide = new Button("Divide");

  // Add top and bottom UI to HBox containers

  GridPane calcTop = new GridPane();
  calcTop.setAlignment(Pos.CENTER);
  calcTop.setPadding(new Insets(5));
  calcTop.add(tNumber1, 0, 0);
  calcTop.add(tfNumber1, 1, 0);
  calcTop.add(tNumber2, 0, 1);
  calcTop.add(tfNumber2, 1, 1);
  calcTop.add(tResult, 0, 2);
  calcTop.add(tfResult, 1, 2);


  FlowPane calcBottom = new FlowPane();
  calcBottom.setAlignment(Pos.CENTER);
  calcBottom.setPadding(new Insets(5));
```

```
    calcBottom.getChildren().addAll(btAdd, btSubtract, btMultiply, btDivide);

    // Add HBox containers to a BorderPane
    BorderPane pane = new BorderPane();
    pane.setTop(vb);
    pane.setCenter(calcTop);
    pane.setBottom(calcBottom);

    c1.setOnAction(e -> {
       pane.setBackground(new Background(new BackgroundFill(Color.RED,null,null)));
    });
    c2.setOnAction(e -> {
       pane.setBackground(new Background(new
BackgroundFill(Color.GREEN,null,null)));
    });

    // Register event handlers for buttons
    btAdd.setOnAction(e -> {
      double a = getDoubleFromTextField(tfNumber1);
      double b = getDoubleFromTextField(tfNumber2);
      tfResult.setText(String.valueOf(a + b));
    });

    btSubtract.setOnAction(e -> {
      double a = getDoubleFromTextField(tfNumber1);
      double b = getDoubleFromTextField(tfNumber2);
      tfResult.setText(String.valueOf(a - b));
    });

    btMultiply.setOnAction(e -> {
      double a = getDoubleFromTextField(tfNumber1);
      double b = getDoubleFromTextField(tfNumber2);
      tfResult.setText(String.valueOf(a * b));
    });

    btDivide.setOnAction(e -> {
      double a = getDoubleFromTextField(tfNumber1);
      double b = getDoubleFromTextField(tfNumber2);
      tfResult.setText(b == 0 ? "NaN" : String.valueOf(a / b));
```

```
  });

  Scene scene = new Scene(pane);
  primaryStage.setTitle("Simple Calculator");
  primaryStage.setScene(scene);
  primaryStage.setResizable(false);
  primaryStage.show();
}

private static double getDoubleFromTextField(TextField t) {
  return Double.parseDouble(t.getText());
}

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







5.6: Menus – Basics – Menu – Menu bars – MenuItem.

5.6.1. JavaFX Menus, MenuItem and MenuBar:

Menu is a popup menu that contains several menu items that are displayed when the user clicks a menu. The user can select a menu item after which the menu goes into a hidden state.

MenuBar is usually placed at the top of the screen which contains several menus. JavaFX MenuBar is typically an implementation of a menu bar.

Constructor of the MenuBar class are:

1. **MenuBar():** creates a new empty menubar.
2. **MenuBar(Menu... m):** creates a new menubar with the given set of menu.

Constructor of the Menu class are:

1. **Menu():** creates an empty menu
2. **Menu(String s):** creates a menu with a string as its label
3. **Menu(String s, Node n):**Constructs a Menu and sets the display text with the specified text and sets the graphic Node to the given node.
4. **Menu(String s, Node n, MenuItem... i):**Constructs a Menu and sets the display text with the specified text, the graphic Node to the given node, and inserts the given items into the items list.

Commonly used methods:

Method	Explanation
getItems()	returns the items of the menu
hide()	hide the menu
show()	show the menu
getMenus()	The menus to show within this MenuBar.
isUseSystemMenuBar()	Gets the value of the property useSystemMenuBar
setUseSystemMenuBar(boolean v)	Sets the value of the property useSystemMenuBar.
setOnHidden(EventHandler v)	Sets the value of the property onHidden.

setOnHiding(EventHandler v)	Sets the value of the property onHiding.
setOnShowing(EventHandler v)	Sets the value of the property onShowing.
setOnShown(EventHandler v	Sets the value of the property onShown.

JavaFX Menu

- ✓ In the JavaFX application, in order to create a menu, menu items, and menu bar, Menu, MenuItem, and MenuBar class is used. The menu allows us to choose options among available choices in the application.
- ✓ All methods needed for this purpose are present in the javafx.scene.control.Menu class.

Example: **Java program to create a menu bar and add menu to it and also add menuitems to the menu**

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuBar;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.VBox;
public class MenuUI extends Application {
@Override
    public void start(Stage primaryStage) throws Exception
    {
            Menu newmenu = new Menu("File");
            Menu newmenu2 = new Menu("Edit");

            MenuItem m1 = new MenuItem("Open");
            MenuItem m2 = new MenuItem("Save");
            MenuItem m3 = new MenuItem("Exit");
            MenuItem m4 = new MenuItem("Cut");
            MenuItem m5 = new MenuItem("Copy");
```

```
            MenuItem m6 = new MenuItem("Paste");
            newmenu.getItems().add(m1);
            newmenu.getItems().add(m2);
            newmenu.getItems().add(m3);
            newmenu2.getItems().add(m4);
            newmenu2.getItems().add(m5);
            newmenu2.getItems().add(m6);
            MenuBar newmb = new MenuBar();
            newmb.getMenus().add(newmenu);
            newmb.getMenus().add(newmenu2);
            VBox box = new VBox(newmb);
            Scene scene = new Scene(box,400,200);
            primaryStage.setScene(scene);
            primaryStage.setTitle("JavaFX Menu Example");
            primaryStage.show();
    }
    public static void main(String[] args)
    {
            Application.launch(args);
    }
}
```

Output: