# Chapter-6
# GUI and JavaFX

## Introduction

- A GUI provides user-friendly human interaction
- Traditionally, we have used AWT and Swing toolkits for creating GUI in java.
- JavaFX is considered as modern Graphical User Interface (GUI) toolkit that helps create desktop applications.
- After the emergence of JavaFX, programmers rely on this for developing GUI with rich content.
- Moreover, it also supports in all operating systems and devices like Linux, iOS, Windows, Mac, Android etc. JavaFX allows developers to rapidly build rich cross-platform applications.
- JavaFX supports modern GPUs through hardware-accelerated graphics.

## Why JavaFx

- There are several reasons why JavaFX is a great GUI application platform.
  1. Java is still one of the most popular programming languages in the world, with a large set of standard classes, and a rich set of open source toolkits developed by the Java developer community.
  2. JavaFX can run on all of the following OS'es and devices:
     - Windows
     - Linux
     - Mac
     - iOS
     - Android / Chromebook

  This makes JavaFX a versatile cross OS and cross device application toolkit.
- JavaFX comes with a rich set of GUI controls, and open source toolkits add even more tools to the total ecosystem.
- JavaFX comes with a large set of built-in GUI components, like buttons, text fields, tables, trees, menus, charts, Accordion, togglebutton, 2D and 3D Shapes and much more. JavaFX can be styled via CSS and / or programmatically. JavaFX comes with a built-in chart library you can use for simple charts. JavaFX has support for 2D and 3D Graphics. JavaFX has a WebView which can display modern web applications.

## JavaFx vs Swing

- Swing and AWT were older Java frameworks replaced by the JavaFX platform for developing rich Internet applications in JDK8.
- JavaFX is meant to produce applications with such subtle GUI options as animation, web views, audio and video playback, and designs supported by Cascading Style Sheets (CSS).
- JavaFX has intrinsic support for signature gestures resembling scrolling, swiping, rotating, and zooming. but, Swing lacks any support for modern touch devices.
- JavaFX has many eye-catching controls that Swing doesn't have.
- In JavaFX, layouts are subclasses of the Node category similar to controls and shapes. In Swing, a layout is related to a JPanel.
- Events in JavaFX are higher thought-out and additional consistent than their equivalents in Swing. Events in Swing aren't that consistent
- In comparison with JavaFX, Swing is a complex collection of UI components that keep on adding a few more bugs every time new features are included into an already unmanageable design
- JavaFX is by far better than Swing at handling graphics because it uses a hardware-accelerated graphics pipeline called Prism to do the rendering job.
- Javafx support code based as well as CSS based styling but swing support only code based stylling.
- Javafx is vector based but swing is pixel based.
- Javafx has built in support for 3D graphics but swing doesn't have.
- Javafx has webview that can render modern web page but swing doesn't have.
- We can launch javafx with or without main method but swing application must have main method.

## Basic Structure of JavaFx Application

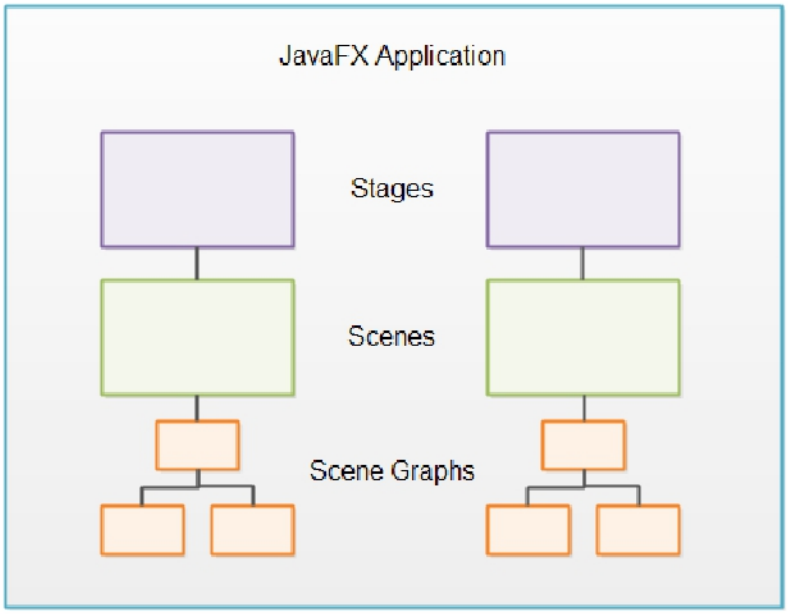


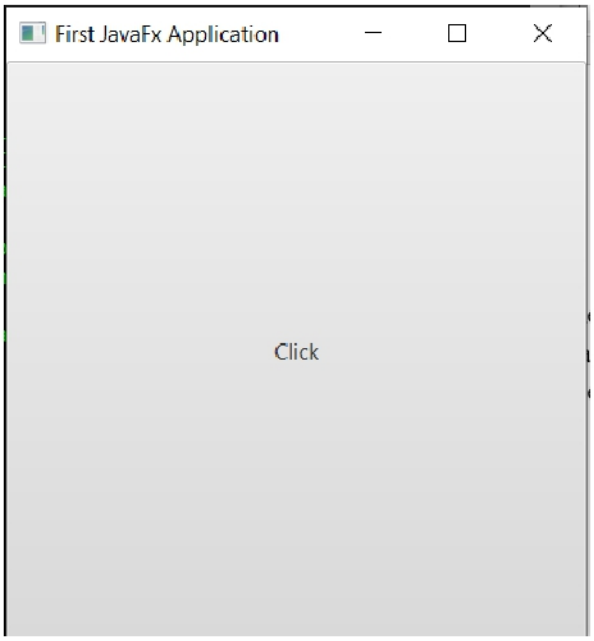

Fig: Structure of JavaFx Application

- Any JavaFX application needs a primary launch class. Any class that extends the Javafx.application.Application class is the primary entry point for any javafx application.
- This subclass must override the the abstract start() method of the Application class
- The start() method is called when the JavaFX application is started.
- The start() method takes a single parameter of the type Stage .
- **Stage** is top level container in javafx application. The stage is where all the visual parts of the JavaFX application are displayed. The Stage object is created for you by the JavaFX runtime. We can set title of thus created stage using **setTitle() method** and then calls **show()** on it to display . That will make the JavaFX application visible in a window with the title visible in the top bar of the window.
- To display something inside the JavaFX application window you must add **a Scene** to the Stage object. Scene is a container class for all the content in scene graph in the stage. A stage can display multiple scenes, just like in a theater play. Similarly, a computer game could have a menu scene, a game scene, a game over scene, a high score scene etc.
- All visual components (controls, layouts etc.) must be attached to a scene to be displayed, and that scene must be attached to a stage for the whole scene to be visible. The total object graph of all the controls, layouts etc. attached to a scene is called the **scene graph.**
- All components attached to the scene graph are called nodes. All nodes are subclasses of a JavaFX class called javafx.scene.Node .
- Finally we can launch a JavaFX application with or without a main() method. But, if you want to pass command line parameters to the application we need to add a main () method.

- **The following program demonstrates the concept.**

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class Demo extends Application
```

```
{
        public void start(Stage primaryStage)
        {
                Button btn_ok=new Button("Click");
                Scene sc=new Scene(btn_ok,400,400);//create a scene and add button in the scene
                primaryStage.setScene(sc);//place the scene in the stage
                primaryStage.setTitle("First JavaFx Application");//set the stage title
                primaryStage.show();//display the stage

        }
        public static void main(String [] args)
        {
                launch(args);//start the application
        }
}
```




## JavaFx Layout

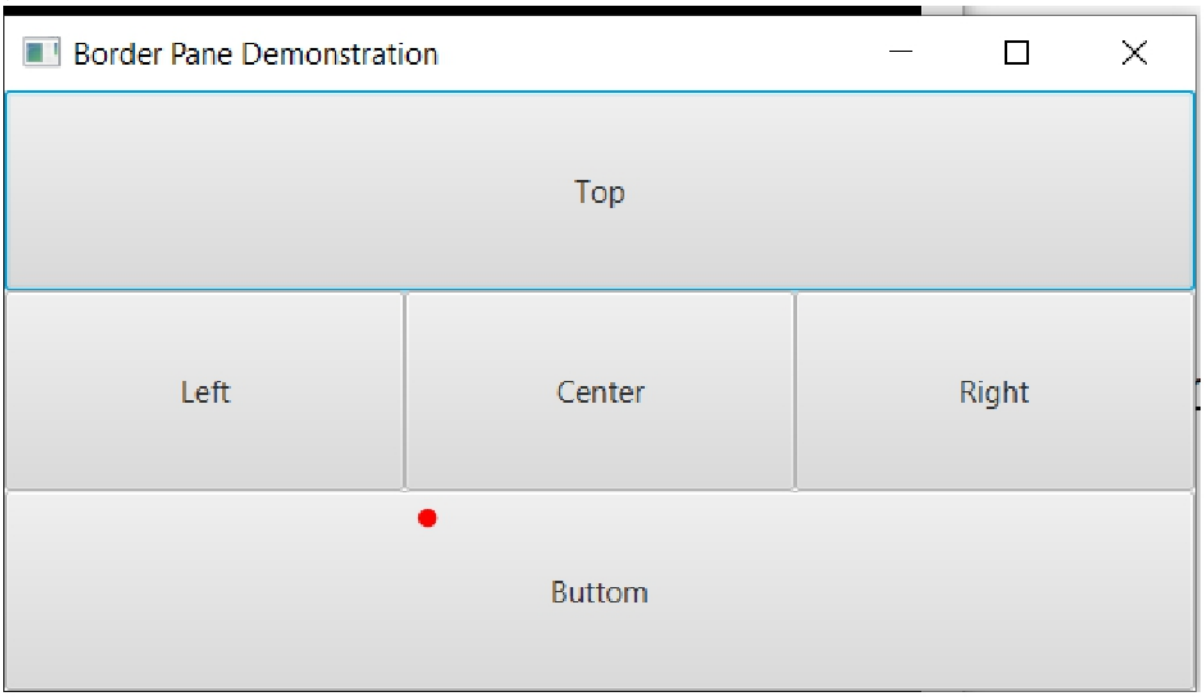
- Layouts are particularly useful to have consistent arrangements of a UI control such as Buttons, Texts, Shapes, and so on within the viewable area, even on resize.
- JavaFX layouts are simpler and more intuitive to implement that Swing layouts.
- Some common layouts are:

1. Border Pane
    - In this layout, the nodes are arranged in the top, center, bottom, left, and, right positions.
    - We can create a border pane in your application by instantiating the **javafx.scene.layout.BorderPane** class.
    - Constructor for creating border pane is

      BorderPane bp = new BorderPane();
    - There are five properties of this class (Node type) specifying the positions in the pane, namely, top, bottom, right, left, center. You can set nodes as values to these properties using the **setTop(), setBottom(), setRight(), setleft() and, setCenter()**.
    - We can set the size of the border pane using the **setPrefSize()** method.
    - The following program demonstrates the concept.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
public class Demo extends Application
{
  public void start(Stage stage)
  {
    //Creating buttons
    Button left = new Button("Left");
    left.setPrefSize(200, 100);
    Button right = new Button("Right");
    right.setPrefSize(200, 100);
    Button top = new Button("Top");
    top.setPrefSize(595, 100);
    Button bottom = new Button("Buttom");
    bottom.setPrefSize(595, 100);
    Button center = new Button("Center");
    center.setPrefSize(200, 100);

    //Creating the border pane
    BorderPane pane = new BorderPane();

    //Setting the top, bottom, center, right and left nodes to the pane
    pane.setTop(top);
    pane.setBottom(bottom);
    pane.setLeft(left);
    pane.setRight(right);
    pane.setCenter(center);

    //Setting the Scene
    Scene scene = new Scene(pane, 595, 300);
    stage.setTitle("Border Pane Demonstration");
```

```
    stage.setScene(scene);
    stage.show();
  }
  public static void main(String args[])
  {
    launch(args);
  }
}
```




2. Flow Pane
   - A JavaFX FlowPane is a layout component which lays out its child components either vertically or horizontally, and which can wrap the components onto the next row or column if there is not enough space in one row.
   - The JavaFX FlowPane layout component is represented by the class **javafx.scene.layout.FlowPane**
   - **Constructor for creating flowlayout pane is**

     FlowPane flowpane = new FlowPane();
   - We can add children to a FlowPane by obtaining its **child collection using getChildren() method** and add adding the components to it we want the FlowPane to layout.
   - To make a FlowPane visible you must add it to the JavaFX scene graph. To do so we must add the FlowPane instance to a Scene object, or add the FlowPane to a layout component which is added to a Scene object.
   - The following program demonstrates the concept.

```
import javafx.application.Application;
import javafx.geometry.Orientation;
import javafx.scene.Scene;
```

```
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;
public class Demo extends Application
{
   public void start(Stage primaryStage) throws Exception
   {
      primaryStage.setTitle("FLowPane Demonstration");
      Button btn1 = new Button("Button1");
      Button btn2 = new Button("Button2");
      Button btn3 = new Button("Button3");

      FlowPane flowpane = new FlowPane();
         //flowpane.setHgap(10);//Set Horizontal Gap between each components
         //flowpane.setVgap(10); //Set Vertical Gap between each components
         //flowpane.setOrientation(Orientation.VERTICAL);//Set the orienttion, default one is HORIZONTAL
      flowpane.getChildren().add(btn1);
      flowpane.getChildren().add(btn2);
      flowpane.getChildren().add(btn3);

      Scene scene = new Scene(flowpane, 500, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
   public static void main(String[] args)
  {
      Application.launch(args);
   }
}
```

3. GridPane:
   - A JavaFX GridPane is a layout component which Lays out UI controls in a tabular fashion, in grids of rows and columns
   - The size of the cells in the grid depends on the components displayed in the GridPane, but there are some rules. All cells in the same row will have the same height, and all cells in the same column will have the same width. Different rows can have different heights and different columns can have different widths
   - The JavaFX GridPane layout component is represented by the class javafx.scene.layout.GridPane
   - The Constructor for creating gridpane is

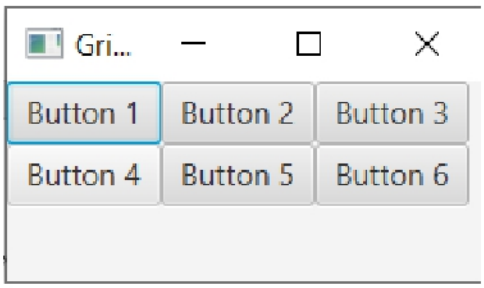
     GridPane gridPane = new GridPane();
   - We can add children to gridpane using add() method.

     Add(component,column_index, row_ind, column_span, row_span);

     Where, component=node to be added, column_index is the column index , row_ind is the row index of the cell in which the component should be displayed, column_span is the number of column the columns should extend to and row_span is the number of row the component should extends to.
   - The following demonstrates the conept.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
public class Demo extends Application
{
   public void start(Stage primaryStage) throws Exception
   {
      primaryStage.setTitle("GridPane Demonstration");
      Button button1 = new Button("Button 1");
      Button button2 = new Button("Button 2");
      Button button3 = new Button("Button 3");
      Button button4 = new Button("Button 4");
      Button button5 = new Button("Button 5");
      Button button6 = new Button("Button 6");

      GridPane gridPane = new GridPane();

      gridPane.add(button1, 0, 0, 1, 1);
      gridPane.add(button2, 1, 0, 1, 1);
      gridPane.add(button3, 2, 0, 1, 1);
      gridPane.add(button4, 0, 1, 1, 1);
```

```
        gridPane.add(button5, 1, 1, 1, 1);
        gridPane.add(button6, 2, 1, 1, 1);

        Scene scene = new Scene(gridPane, 240, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args)
    {
        Application.launch(args);
    }
}
```

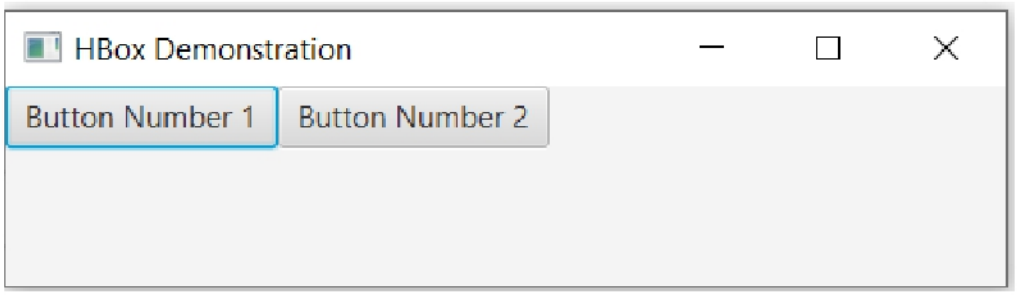



4. Hbox
   - The JavaFX HBox component is a layout component which positions all its child nodes (components) in a horizontal row.
   - The Java HBox component is represented by the class javafx.scene.layout.HBox
   - The constructor for creating Hbox is

     HBox hbox = new HBox();
   - The following program demonstrates the concept

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class Demo extends Application
{
 public void start(Stage primaryStage) throws Exception
 {
  primaryStage.setTitle("HBox Demonstration");
```

```
    Button button1 = new Button("Button Number 1");
    Button button2 = new Button("Button Number 2");

    HBox hbox = new HBox();
    //hbox.setSpacing(50);//sets spacing between each components
    hbox.getChildren().add(button1);
    hbox.getChildren().add(button2);

    Scene scene = new Scene(hbox, 200, 100);
    primaryStage.setScene(scene);
    primaryStage.show();
  }

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

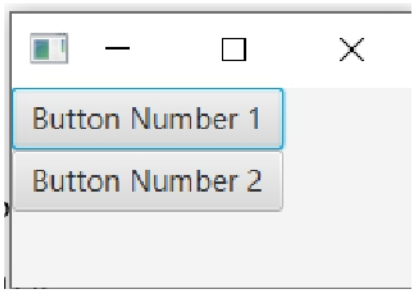



5. Vbox
   - The JavaFX VBox component is a layout component which positions all its child nodes (components) in a vertical column - on top of each other.
   - The JavaFX VBox component is represented by the class javafx.scene.layout.VBox.
   - The constructor for creating Vbox is

     VBox vbox = new VBox();
   - The following program demonstrates the concept.

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

```
public class Demo extends Application
{
 public void start(Stage primaryStage) throws Exception
 {
  primaryStage.setTitle("VBox Demonstration");

  Button button1 = new Button("Button Number 1");
  Button button2 = new Button("Button Number 2");

  VBox vbox = new VBox();
  //vbox.setSpacing(50);//sets spacing between each components
  vbox.getChildren().add(button1);
  vbox.getChildren().add(button2);

  Scene scene = new Scene(vbox, 200, 100);
  primaryStage.setScene(scene);
  primaryStage.show();
 }
 public static void main(String[] args)
{
  Application.launch(args);
 }
}
```




## JavaFX UI Controls

### Label

- The JavaFX Label control can display a text or image label inside a JavaFX GUI.
- The label control must be added to the scene graph to be visible.
- The JavaFX Label control is represented by the class javafx.scene.control.Label .
- The constructor for creating label is

Label label = new Label(String text);

**TextField**

- A JavaFX TextField control enables users of a JavaFX application to enter text which can then be read by the application.
- The JavaFX TextField control is represented by the class javafx.scene.control.TextField .
- The constructor for creating label is

TextField textField = new TextField();

**Button**

- A JavaFX Button control enables a JavaFX application to have some action executed when the application user clicks the button.
- The JavaFX Button control is represented by the class javafx.scene.control.Button .
- The constructor for creating button is

Button button = new Button(String label);

- In order to respond to the click of a button you need to attach an event listener to the Button object. Here is how that looks:

```
button.setOnAction(new EventHandler()
{
   public void handle(ActionEvent actionEvent)
  {
      //... do something in here.
   }
});
```

## Event Handling in javafx application

- In JavaFX, we can develop GUI applications, web applications and graphical applications. In such applications, whenever a user interacts with the application (nodes), an event is said to have been occurred.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.
- JavaFX provides support to handle a wide varieties of events. The class named Event of the package javafx.event is the base class for an event. JavaFX provides a wide variety of events. Some of them are are MouseEvent, DragEvent, ActionEvent, WindowEvent etc.
- Event handler for each event type can be defined consistently as

```
EventHandler<ActionEvent> event = new EventHandler<ActionEvent>()
    {
        public void handle(ActionEvent e)
       {
            //handle event here
       }
};
```

- To add an event to a node/Source/Component, we need to register it as below

btn_ok.setOnAction(event);

**Create a GUI with Javafx to input and display sum of two numbers. Use GridPane for layout management.**

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
public class Demo extends Application
{
   public void start(Stage primaryStage) throws Exception
   {
        TextField txt_fn=new TextField();
        TextField txt_sn=new TextField();
        TextField txt_res=new TextField();
        Label lbl_fn=new Label("First Number :");
        Label lbl_sn=new Label("Second Number :");
        Label lbl_res=new Label("Result  :");
        Button btn_ok=new Button("OK");
        EventHandler<ActionEvent> event = new EventHandler<ActionEvent>()
         {
            public void handle(ActionEvent e)
        {
                        int fn=Integer.parseInt(txt_fn.getText());
                        int sn=Integer.parseInt(txt_sn.getText());
                        int res=fn+sn;
                        txt_res.setText(Integer.toString(res));
        }
      };
      btn_ok.setOnAction(event);//register ActionEvent
```

```
        GridPane gridPane = new GridPane();
        gridPane.add(lbl_fn, 0, 0, 1, 1);
        gridPane.add(txt_fn, 1, 0, 1, 1);
        gridPane.add(lbl_sn, 0, 1, 1, 1);
        gridPane.add(txt_sn, 1, 1, 1, 1);
        gridPane.add(lbl_res, 0, 2, 1, 1);
        gridPane.add(txt_res, 1, 2, 1, 1);
        gridPane.add(btn_ok, 0, 3, 1, 1);

        Scene scene = new Scene(gridPane, 240, 300);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args)
    {
        Application.launch(args);
    }
}
```

**Create a GUI with Javafx to input and display sum of two numbers. Use GridPane for layout management. The program must prompt user to quit if he/she closes the window.**

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import javafx.event.ActionEvent;
import javafx.stage.WindowEvent;
import javafx.event.EventHandler;
import javax.swing.*;
```

```
public class Demo extends Application
{
   public void start(Stage primaryStage) throws Exception
   {
        TextField txt_fn=new TextField();
        TextField txt_sn=new TextField();
        TextField txt_res=new TextField();
        Label lbl_fn=new Label("First Number :");
        Label lbl_sn=new Label("Second Number :");
        Label lbl_res=new Label("Result  :");
        Button btn_ok=new Button("OK");
        EventHandler<ActionEvent> event = new EventHandler<ActionEvent>()
        {
           public void handle(ActionEvent e)
        {
                        int fn=Integer.parseInt(txt_fn.getText());
                        int sn=Integer.parseInt(txt_sn.getText());
                        int res=fn+sn;
                        txt_res.setText(Integer.toString(res));
        }
      };
        EventHandler<WindowEvent> event1 = new EventHandler<WindowEvent>()
        {
           public void handle(WindowEvent e)
        {
                int res=JOptionPane.showConfirmDialog(null,"Are you sure");
                if(res==0)
                {
                        System.exit(0);
                }
                else
                {
                        e.consume();//if other options pressed then cancel the current event
```

```
                }
            }
        };

            btn_ok.setOnAction(event);//register ActionEvent
            primaryStage.setOnCloseRequest(event1);//Equivalent to onclosing() in swing
        GridPane gridPane = new GridPane();
        gridPane.add(lbl_fn, 0, 0, 1, 1);
        gridPane.add(txt_fn, 1, 0, 1, 1);
        gridPane.add(lbl_sn, 0, 1, 1, 1);
        gridPane.add(txt_sn, 1, 1, 1, 1);
        gridPane.add(lbl_res, 0, 2, 1, 1);
        gridPane.add(txt_res, 1, 2, 1, 1);
        gridPane.add(btn_ok, 0, 3, 1, 1);

        Scene scene = new Scene(gridPane, 240, 300);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args)
    {
        Application.launch(args);
    }
}
```

**RadioButton**

- A JavaFX RadioButton is a button that can be selected or not selected.
- The JavaFX RadioButton is represented by the class javafx.scene.control.RadioButton.
- The constructor for creating RadioButton is

  RadioButton radioButton1 = new RadioButton(Strign Lavel);
- The RadioButton class has a method named isSelected which lets you determine if the RadioButton is selected or not. The isSelected() method returns a boolean with the value true if the RadioButton is selected, and false if not. Here is an example:

  boolean isSelected = radioButton1.isSelected();
- We can group JavaFX RadioButton instances into a ToggleGroup. A ToggleGroup allows at most one RadioButton to be selected at any time.

  ToggleGroup radioGroup = new ToggleGroup();

  To add a radio button to this group we have to use setToggleGroupMethod() of RadioButton class.

For example:

radioButton1.setToggleGroup(radioGroup);

- The following program demonstrates the concept.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class Demo extends Application
 {
   public void start(Stage primaryStage) throws Exception
        {
      primaryStage.setTitle("Radio Button Demonstration");
      RadioButton radioButton1 = new RadioButton("Left");
      RadioButton radioButton2 = new RadioButton("Right");
      RadioButton radioButton3 = new RadioButton("Up");
      RadioButton radioButton4 = new RadioButton("Down");

      ToggleGroup radioGroup = new ToggleGroup();
      radioButton1.setToggleGroup(radioGroup);
      radioButton2.setToggleGroup(radioGroup);
      radioButton3.setToggleGroup(radioGroup);
      radioButton4.setToggleGroup(radioGroup);

      HBox hbox = new HBox(radioButton1, radioButton2, radioButton3, radioButton4);
      Scene scene = new Scene(hbox, 200, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
   public static void main(String[] args)
 {
      Application.launch(args);
```

```
    }
}
```

**CheckBox**

- A JavaFX CheckBox is a button which can be in three different states: Selected, not selected and unknown (indeterminate).
- The JavaFX CheckBox control is represented by the class javafx.scene.control.CheckBox.
- The constructor for creating Checkbox is

  CheckBox checkBox1 = new CheckBox(Strign label);
- The following program demonstrates the concept

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class Demo extends Application
{
 public void start(Stage primaryStage) throws Exception
 {
      primaryStage.setTitle("CheckBox Demonstration");

      CheckBox checkBox1 = new CheckBox("Green");
        CheckBox checkBox2 = new CheckBox("Red");
      HBox hbox = new HBox(checkBox1,checkBox2);
      Scene scene = new Scene(hbox, 200, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
   public static void main(String[] args)
 {
      Application.launch(args);
   }
}
```

**Tooltip**

- The JavaFX Tooltip class (javafx.scene.control.Tooltip) can display a small popup with explanatory text when the user hovers the mouse over a JavaFX control.
- A Tooltip is a well-known feature of modern desktop and web GUIs.
- A Tooltip is useful to provide extra help text in GUIs where there is not space enough available to have an explanatory text visible all the time, e.g. in the button text.
- The constructor for creating tooltip is

Tooltip tooltip1 = new Tooltip(String text);

- The following program demonstrates adding tooltip text in JavaFx components

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import javafx.scene.control.Tooltip;


public class Demo extends Application
{
   public void start(Stage primaryStage) throws Exception
 {
      primaryStage.setTitle("Tooltip Demonstration");
     Tooltip tooltip1 = new Tooltip("Click To Browse Text File");
     Button btn=new Button("Browse");
     btn.setTooltip(tooltip1);
      HBox hbox = new HBox(btn);
      Scene scene = new Scene(hbox, 200, 100);
      primaryStage.setScene(scene);
      primaryStage.show();
   }
   public static void main(String[] args)
 {
      Application.launch(args);
   }
```

```
}
```

**Hyperlink**

- The JavaFX Hyperlink control is a text that functions as a button, meaning you can configure a Hyperlink to perform some action when the user clicks it.
- Just like a hyperlink in a web page.
- The JavaFX Hyperlink control is represented by the class javafx.scene.control.Hyperlink .
- The constructor for creating hyperlink is

  Hyperlink link = new Hyperlink(String label);
- The following program demonstrates the conept

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Hyperlink;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class Demo extends Application
{
   public static void main(String[] args)
   {
      launch(args);
   }
   public void start(Stage primaryStage)
 {
      primaryStage.setTitle("Hyperlink Demonstration");
      Hyperlink link = new Hyperlink("Click Me!");
      VBox vBox = new VBox(link);
      Scene scene = new Scene(vBox, 960, 600);
      primaryStage.setScene(scene);
      primaryStage.show();
   }

}
```

**FileChooser.**

- A JavaFX FileChooser class (javafx.stage.FileChooser) is a dialog that enables the user to select one or more files via a file explorer from the user's local computer.
- The JavaFX FileChooser is implemented in the class javafx.stage.FileChooser.
- The constructor for creating file chooser dialog is

  FileChooser fileChooser = new FileChooser();
- Showing the JavaFX FileChooser dialog is done by calling its showOpenDialog() method as below

  File selectedFile = fileChooser.showOpenDialog(stage);

  To filter out what files are shown in the filechooser dialog we can add filter as below

```
FileChooser fileChooser = new FileChooser();
fileChooser.getExtensionFilters().addAll(
    new FileChooser.ExtensionFilter("Text Files", "*.txt")
   ,new FileChooser.ExtensionFilter("HTML Files", "*.htm")
);
```

- The following program demonstrates the concept

```
import java.io.File;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
public class Demo extends Application
 {
   public static void main(String[] args)
{
      launch(args);
   }
   public void start(Stage primaryStage)
{
      primaryStage.setTitle("File Chooser Demonstration");
      FileChooser fileChooser = new FileChooser();
      Button btn_browse=new Button("Browse");
         EventHandler<ActionEvent> event = new EventHandler<ActionEvent>()
```

```
            {


             public void handle(ActionEvent e)
            {
                    File f= fileChooser.showOpenDialog(primaryStage);
            }
        };


          btn_browse.setOnAction(event);//register ActionEvent
        VBox vBox = new VBox(btn_browse);
        Scene scene = new Scene(vBox, 960, 600);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

**Assignment**
**Explain Javafx menu button with its constructor. Also write suitable program to demonstrate it.**