GUI with JavaFX

- JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.
- JavaFX is a Java library used to develop Desktop applications as well as Rich Internet Applications (RIA). The applications built in JavaFX, can run on multiple platforms including Web, Mobile and Desktops.
- JavaFX is intended to replace swing in Java applications as a GUI framework. However, It provides more functionalities than swing. Like Swing, JavaFX also provides its own components and doesn't depend upon the operating system. It is lightweight and hardware accelerated. It supports various operating systems including Windows, Linux and Mac OS.

Rich Internet Applications are those web applications which provide similar features and experience as that of desktop applications. They offer a better visual experience when compared to the normal web applications to the users. These applications are delivered as browser plug-ins or as a virtual machine and are used to transform traditional static applications into more enhanced, fluid, animated and engaging applications.

Features of JavaFX

JavaFX is an open-source framework based on Java, used for advancing rich client applications. JavaFX is recognized as the replacement or successor of the Java Swing in the field of graphical user interface (GUI) development technology in the platform of Java. The JavaFX library is available as a public Java application programming interface (API).

The JavaFX library comprises numerous peculiarities that make it a handpicked option for developers to develop rich client applications. These features are as follows:

1. **Java Library –** JavaFX is a Java library, which allows the user to gain the support of all the Java characteristics such as multithreading, generics, lambda expressions, and many more. The user can also use any of the Java editors or IDE's of their choice, such as Eclipse, NetBeans, to write, compile, run, debug, and package their JavaFX application.

2. **Platform Independent –** The rich internet applications made using JavaFX are platform-independent. The JavaFX library is open for all the scripting languages that can be administered on a JVM, which comprise – Java, Groovy, Scala, and JRuby.

3. **FXML –** JavaFX emphasizes an HTML-like declarative markup language known as FXML. FXML is based on extensible markup language (XML). The sole objective of this markup language is to specify a user interface (UI). In FXML, the programming can be done to accommodate the user with an improved GUI.

4. **Scene Builder –** JavaFX also implements a tool named Scene Builder, which is a visual editor for FXML. Scene Builder generates FXML mark-ups that can be transmitted to the IDE's like Eclipse and NetBeans, which further helps the user to combine the business logic to their applications. The users can also use the drag and drop design interface, which is used to design FXML applications (just like the Drag & Drop feature of the Java Swing and DreamWeaver Applications).

5. **Hardware-accelerated Graphics Pipeline –** The graphics of the JavaFX applications are based on the hardware-accelerated graphics rendering pipeline, commonly known as Prism. The Prism engine offers smooth JavaFX graphics that can be rendered quickly when utilized with a supported graphics card or graphics processing unit (GPU). In the case where the system does not hold the graphic cards, then the prism engine defaults to the software rendering stack.

6. **WebView –** JavaFX applications can also insert web pages. To embed web pages, Web View of JavaFX uses a new HTML rendering engine technology known as WebKitHTML. WebView is used to make it possible to insert web pages within a

JavaFX application. JavaScript running in WebView can call Java APIs and vice-versa.

7. **Built-in UI controls –** JavaFX comprises all the major built-in UI controls that help in developing well-featured applications. These built-in UI components are not operating system-dependent. In simple words, these controls do not depend on any of the Operating systems like Windows, iOS, Android, etc. These built-in controls are single-handedly ample to perform a whole implementation of the applications.

8. **CSS Styling –** Just like websites use CSS for styling, JavaFX also provides the feature to integrate the application with CSS styling. The users can enhance the styling of their applications and can also improve the outlook of their implementation by having simple knowledge and understanding of CSS styling.

9. **Rich set of APIs –** JavaFX library also presents a valuable collection of APIs that helps in developing GUI applications, 2D and 3D graphics, and many more. This collection of APIs also includes all the characteristics of the Java platform. Hence, working with this API, a user can access the specialties of Java languages such as Generics, Annotations, Multithreading, and Lambda Expressions, and many other features as well. In JavaFX, the popular Java Collections library was also improved, and notions like lists and maps were introduced. Using these APIs, the users can witness the variations in the data models.

10. **High-Performance media engine –** Like the graphics pipeline, JavaFX also possesses a media pipeline that advances stable internet multimedia playback at low latency. This high-performance media engine or media pipeline is based on a multimedia framework known as Gstreamer.

Swing Vs JavaFx

- Swing is the standard toolkit for Java developer in creating GUI, whereas JavaFX provides platform support for creating desktop applications.
- Swing has a more sophisticated set of GUI components, whereas JavaFX has a decent number of UI components available but lesser than what Swing provides.

- Swing is a legacy library that fully features and provide pluggable UI components, whereas JavaFX has UI components that are still evolving with a more advanced look and feel.
- Swing can provide UI components with a decent look and feel, whereas JavaFX can provide rich internet application having a modern UI.
- Swing related classes can be found in the Java API guide with complete documentation, whereas JavaFX doc is available in various format with comprehensive detailing and file support.
- Swing, since its advent, can create UI component using standard Java component classes, whereas Java FX initially uses a declarative language called JavaFX Script.
- Swing has a UI component library and act as a legacy, whereas JavaFX has several components built over Swing.
- Swing has support for MVC, but it is not consistent across a component, whereas JavaFX support is very friendly with MVC.
- Swing has various IDEs, which offer a tool for rapid development, whereas JavaFX has also support from various IDEs as well, but it is not as mature as Swing.
- A swing was renamed from Java Foundation Classes, and sun microsystems announced it in the year 1997, whereas JavaFX was initially released in December 2008 by Sun microsystem and now acquired by Oracle

Basis of Comparison Between Java Swing vs Java FX	**Java Swing**	**Java FX**
Components	Swing has a number of components to it	Less component as compared to legacy Swing APIs
User Interface	Standard UI components can be designed with Swing	Rich GUI components can be created with an advanced look and feel
Development	Swing APIs are being used to write UI components	JavaFX scripts and fast UI development with screen builder
Functionality	No new functionality introduction for future	JavaFX has a rich new toolkit, expected to grow in future
Category	Legacy UI library fully featured	Up and coming to feature-rich UI components
MVC Support	MVC support across components lack consistency	Friendly with MVC pattern

Important packages in JavaFx

- JavaFX provides a complete API with a rich set of classes and interfaces to build GUI applications with rich graphics. The important packages of this API are −
  - *javafx.animation* − Contains classes to add transition based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes.
  - *javafx.application* − Contains a set of classes responsible for the JavaFX application life cycle.
  - *javafx.css* − Contains classes to add CSS–like styling to JavaFX GUI applications.
  - *javafx.event* − Contains classes and interfaces to deliver and handle JavaFX events.
  - *javafx.geometry* − Contains classes to define 2D objects and perform operations on them.
  - *javafx.stage* − This package holds the top level container classes for JavaFX application.
  - *javafx.scene* − This package provides classes and interfaces to support the scene graph. In addition, it also provides sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc. There are several components that support this rich API of JavaFX.

JavaFX Application Structure

- JavaFX application is divided hierarchically into three main components known as Stage, Scene and nodes. We need to import **javafx.application.Application** class in every JavaFX application. This provides the following life cycle methods for JavaFX application.
  - ***public void init()***
  - ***public abstract void start(Stage primaryStage)***
  - ***public void stop()***
- In order to create a basic JavaFX application, we need to:
  - Import ***javafx.application.Application*** into our code.
  - Inherit ***Application*** into our class.
  - Override ***start()*** method of ***Application*** class.
- In addition to these, it provides a static method named ***launch()*** to launch JavaFX application.

- Since the **launch()** method is static, you need to call it from a static context (main generally). Whenever a JavaFX application is launched, the following actions will be carried out (in the same order).
    - An instance of the application class is created.
    - *init()* method is called.
    - The *start()* method is called.
    - The launcher waits for the application to finish and calls the *stop()* method.

Stage

- Stage 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. The object of primary stage is passed to start method. We need to call show method on the primary stage object in order to show our primary stage. Initially, the primary Stage looks empty. However, we can add various objects to this primary stage. The objects can only be added in a hierarchical way i.e. first, scene graph will be added to this primaryStage and then that scene graph may contain the nodes. A node may be any object of the user's interface like text area, buttons, shapes, media, etc.

Scene

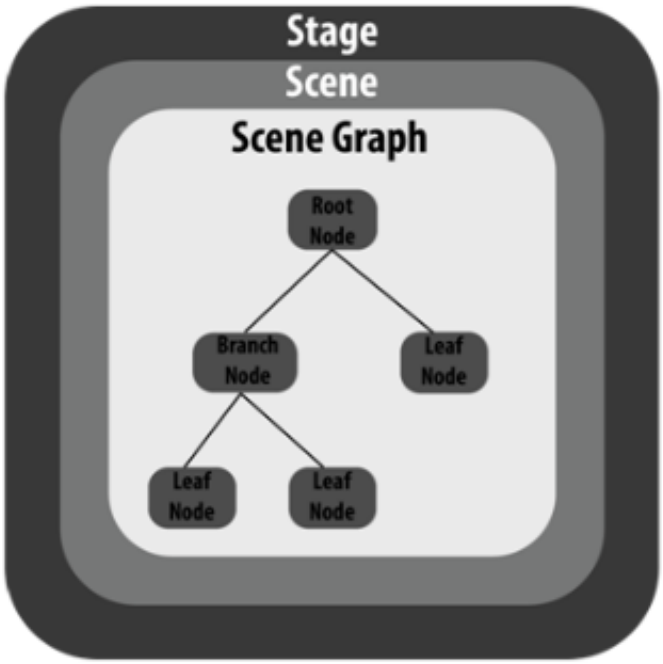
- Scene actually holds all the physical contents (nodes) of a JavaFX application. ***Javafx.scene.Scene*** class provides all the methods to deal with a scene object. Creating scene is necessary in order to visualize the contents on the stage.
- At one instance, the scene object can only be added to one stage. In order to implement Scene in our JavaFX application, we must import ***javafx.scene*** package in our code. The Scene can be created by creating the Scene class object and passing the layout object into the Scene class constructor..

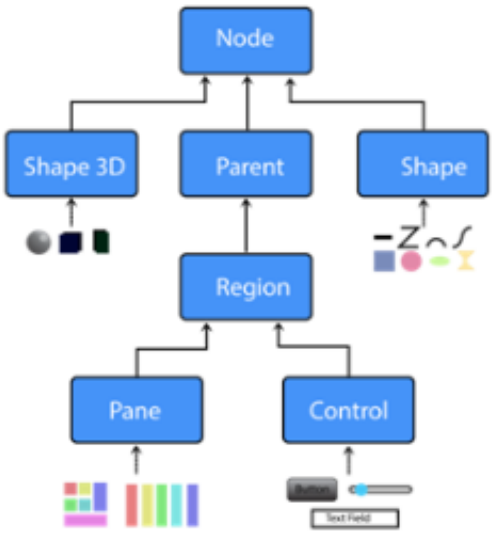
Scene Graph

— Scene Graph exists at the lowest level of the hierarchy. It can be seen as the collection of various nodes. A **node** is the element which is visualized on the stage. It can be any button, text box, layout, image, radio button, check box, etc.

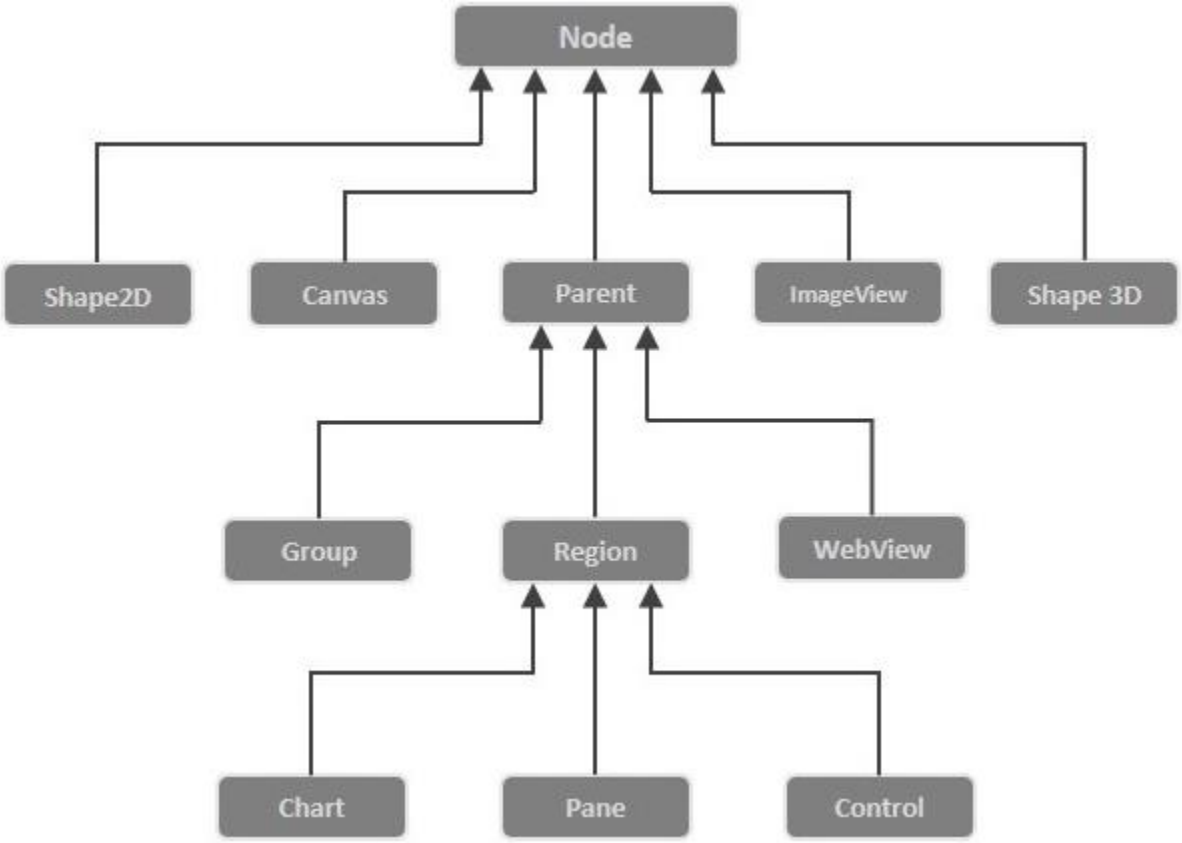
— The nodes are implemented in a tree kind of structure. There is always one root in the scene graph. This will act as a parent node for all the other nodes present in the scene graph. However, this node may be any of the layouts available in the JavaFX system.

— The leaf nodes exist at the lowest level in the tree hierarchy. Each of the node present in the scene graphs represents classes of ***javafx.scene*** package therefore we need to import the package into our application in order to create a full featured **javafx** application.





☞ Following is a diagram representing the node class hierarchy of JavaFX.

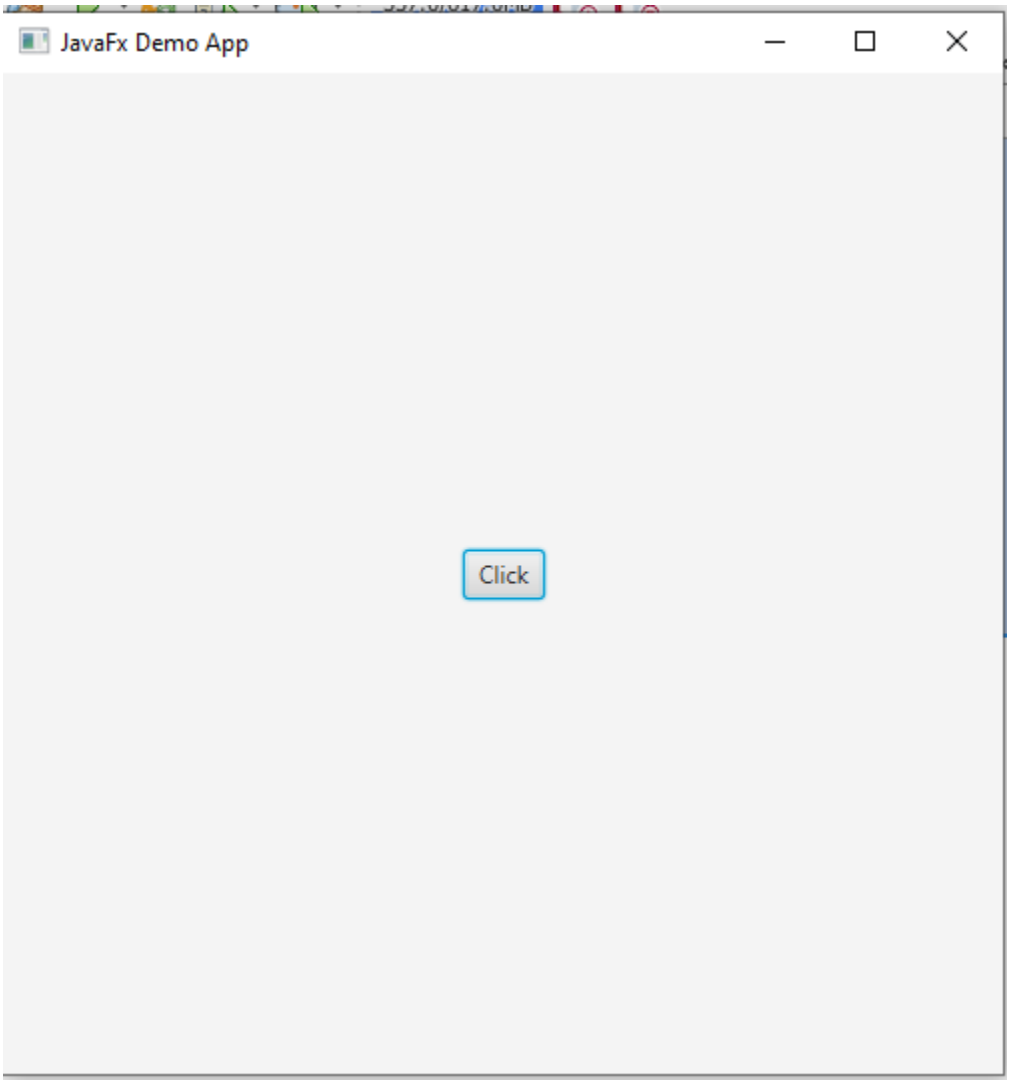


First JavaFX Application

```
package newgui;

import javafx.application.Application;

import static javafx.application.Application.launch;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.layout.StackPane;

import javafx.stage.Stage;

/**

 *
```

```
 * @author Bipin
 */
//Step 1: Extend javafx.application.Application
//and override start()
public class Demo extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
        // Step 2 create a button
        Button btn=new Button("Click");
        //Step 3: Create a layout and add button to it
        //here layout used is StackPane
        StackPane root=new StackPane();
        root.getChildren().add(btn);
        //Step 4: Create a Scene
        Scene scene=new Scene(root,500,500);
        //Step 5: Set the title of the Stage
        primaryStage.setTitle("JavaFx Demo App");
        //Step 6:Adding scene to the stage
        primaryStage.setScene(scene);
        //Step 7: Show the contents of Stage
        primaryStage.show();
    }
```

```
    //Step 8: Launch the app
    public static void main(String args[]){
       launch(args);
    }
}
```



JavaFX Layouts

- This arrangement of the components within the container is called the Layout of the container. We can also say that we followed a layout as it includes placing all the components at a particular position within the container.
- Layouts are the top level container classes that define the UI styles for scene graph objects. Layout can be seen as the parent node to all the other nodes. JavaFX provides various layout panes that support different styles of layouts.
- In JavaFX, Layout defines the way in which the components are to be seen on the stage. It basically organizes the scene-graph nodes. We have several built-in layout panes in JavaFX that are HBox, VBox, StackPane, FlowBox, AnchorPane, etc. Each Built-in layout is represented by a separate class which needs to be instantiated in order to implement that particular layout pane.
- All these classes belong to javafx.scene.layout package. javafx.scene.layout.Pane class is the base class for all the built-in layout classes in JavaFX.

Class	Description
BorderPane	Organizes nodes in top, left, right, centre and the bottom of the screen.
FlowPane	Organizes the nodes in the horizontal rows according to the available horizontal spaces. Wraps the nodes to the next line if the horizontal space is less than the total width of the nodes
GridPane	Organizes the nodes in the form of rows and columns.
HBox	Organizes the nodes in a single row.
Pane	It is the base class for all the layout classes.
StackPane	Organizes nodes in the form of a stack i.e. one onto another
VBox	Organizes nodes in a vertical column.

Steps to create layout

- In order to create the layouts, we need to follow the following steps.
- Instantiate the respective layout class, for example, *HBox root = new HBox();*
- Setting the properties for the layout, for example, *root.setSpacing(20);*
- Adding nodes to the layout object,

- for example r*oot.getChildren().addAll(<NodeObjects>);*

JavaFX BorderPane

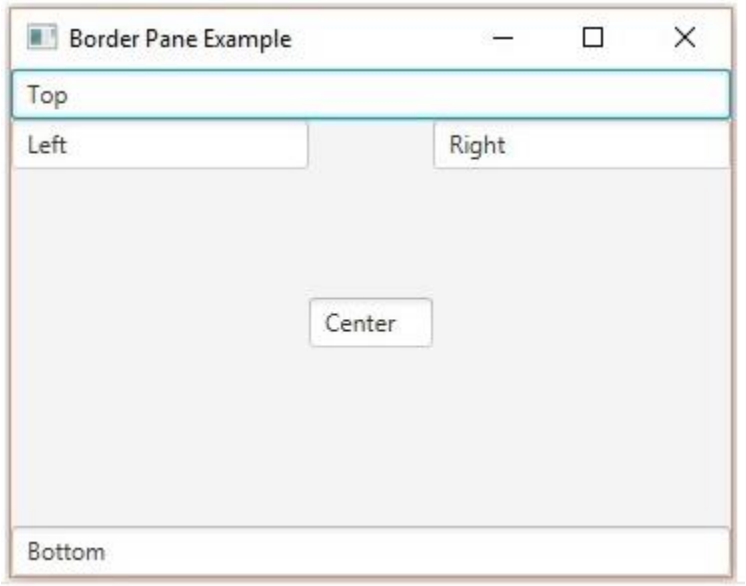
- BorderPane arranges the nodes at the left, right, centre, top and bottom of the screen. It is represented by **javafx.scene.layout.BorderPane** class.
- This class provides various methods like **setRight(), setLeft(), setCenter(), setBottom()** and **setTop()** which are used to set the position for the specified nodes.
- We need to instantiate BorderPane class to create the **BorderPane** layout.
- The properties of BorderPane class along with their setter methods are given in the table below.

Type	Property	Setter Methods	Description
Node	Bottom	setBottom()	Add the node to the bottom of the screen
Node	Centre	setCentre()	Add the node to the centre of the screen
Node	Left	setLeft()	Add the node to the left of the screen
Node	Right	setRight()	Add the node to the right of the screen
Node	Top	setTop()	Add the node to the top of the screen

- There are the following constructors in the class.
  - *BorderPane() :* create the empty layout
  - *BorderPane(Node Center) :* create the layout with the center node
  - *BorderPane(Node Center, Node top, Node right, Node bottom, Node left)* : create the layout with all the nodes

Example

```
import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class BorderPaneExample extends Application {
   @Override
   public void start(Stage stage) {
      //Instantiating the BorderPane class
      BorderPane bPane = new BorderPane();

      //Setting the top, bottom, center, right and left nodes to the pane
      bPane.setTop(new TextField("Top"));
      bPane.setBottom(new TextField("Bottom"));
      bPane.setLeft(new TextField("Left"));
      bPane.setRight(new TextField("Right"));
      bPane.setCenter(new TextField("Center"));

      //Creating a scene object
      Scene scene = new Scene(bPane);

      //Setting title to the Stage
      stage.setTitle("BorderPane Example");

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

      //Displaying the contents of the stage
      stage.show();
   }
   public static void main(String args[]){
      launch(args);
   }
}
```


Border Pane Example
Top
Left
Right
Center
Bottom

JavaFX StackPane

- FlowPane layout pane organizes the nodes in a flow that are wrapped at the flowpane's boundary.
- The horizontal flowpane arranges the nodes in a row and wrap them according to the flowpane's width.
- The vertical flowpane arranges the nodes in a column and wrap them according to the flowpane's height.
- FlowPane layout is represented by **javafx.scene.layout.FlowPane** class. We just need to instantiate this class to create the flowpane layout.
- There are various properties of the class which are described in the table below.

Property	Description	Setter Methods
alignment	The overall alignment of the flowpane's content.	setAlignment(Pos value)
columnHalignment	The horizontal alignment of nodes within the columns.	setColumnHalignment(HPos Value)
hgap	Horizontal gap between the columns.	setHgap(Double value)
orientation	Orientation of the flowpane	setOrientation(Orientation value)
prefWrapLength	The preferred height or width where content should wrap in the horizontal or vertical flowpane.	setPrefWrapLength(double value)
rowValignment	The vertical alignment of the nodes within the rows.	setRowValignment(VPos value)
vgap	The vertical gap among the rows	setVgap(Double value)

- There are 8 constructors in the class that are given below.
    - FlowPane()
    - FlowPane(Double Hgap, Double Vgap)
    - FlowPane(Double Hgap, Double Vgap, Node? children)
    - FlowPane(Node... Children)
    - FlowPane(Orientation orientation)
    - FlowPane(Orientation orientation, double Hgap, Double Vgap)
    - FlowPane(Orientation orientation, double Hgap, Double Vgap, Node? children)
    - FlowPane(Orientation orientation, Node... Children)

```
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Sphere;
import javafx.stage.Stage;

public class FlowPaneExample extends Application {
  @Override
  public void start(Stage stage) {
    //Creating button1
    Button button1 = new Button("Button1");

    //Creating button2
    Button button2 = new Button("Button2");

    //Creating button3
    Button button3 = new Button("Button3");

    //Creating button4
    Button button4 = new Button("Button4");

    //Creating a Flow Pane
    FlowPane flowPane = new FlowPane();

    //Setting the horizontal gap between the nodes
    flowPane.setHgap(25);

    //Setting the margin of the pane
    flowPane.setMargin(button1, new Insets(20, 0, 20, 20));

    //Retrieving the observable list of the flow Pane
    ObservableList list = flowPane.getChildren();
```

```
    //Adding all the nodes to the flow pane
    list.addAll(button1, button2, button3, button4);

    //Creating a scene object
    Scene scene = new Scene(flowPane);

    //Setting title to the Stage
    stage.setTitle("Flow Pane Example");

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

    //Displaying the contents of the stage
    stage.show();
  }
  public static void main(String args[]){
    launch(args);
  }
}
```

JavaFX StackPane

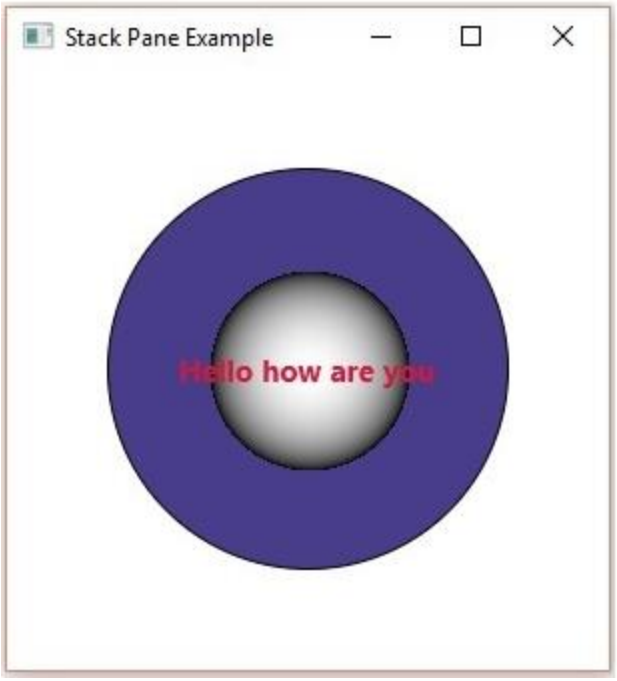
- The StackPane layout pane places all the nodes into a single stack where every new node gets placed on the top of the previous node.
- It is represented by **javafx.scene.layout.StackPane** class.
- We just need to instantiate this class to implement **StackPane** layout into our application.
- The class contains only one property that is given below along with its setter method.

Property	Description	Setter Method
alignment	It represents the default alignment of children within the StackPane's width and height	setAlignment(Node child, Pos value) setAlignment(Pos value)

- The class contains two constructors that are given below.
  - StackPane()
  - StackPane(Node? Children)

```
import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;

import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Sphere;

import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class StackPaneExample extends Application {
  @Override
  public void start(Stage stage) {
    //Drawing a Circle
    Circle circle = new Circle(300, 135, 100);
    circle.setFill(Color.DARKSLATEBLUE);
    circle.setStroke(Color.BLACK);

    //Drawing Sphere
```

```
        Sphere sphere = new Sphere(50);

        //Creating a text
        Text text = new Text("Hello how are you");

        //Setting the font of the text
        text.setFont(Font.font(null, FontWeight.BOLD, 15));

        //Setting the color of the text
        text.setFill(Color.CRIMSON);

        //setting the position of the text
        text.setX(20);
        text.setY(50);

        //Creating a Stackpane
        StackPane stackPane = new StackPane();

        //Setting the margin for the circle
        stackPane.setMargin(circle, new Insets(50, 50, 50, 50));

        //Retrieving the observable list of the Stack Pane
        ObservableList list = stackPane.getChildren();

        //Adding all the nodes to the pane
        list.addAll(circle, sphere, text);

        //Creating a scene object
        Scene scene = new Scene(stackPane);

        //Setting title to the Stage
        stage.setTitle("Stack Pane Example");

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

        //Displaying the contents of the stage
        stage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```



JavaFX HBox

- HBox layout pane arranges the nodes in a single row.
- It is represented by **javafx.scene.layout.HBox** class.
- We just need to instantiate **HBox** class in order to create HBox layout.
- The Properties of the class along with their setter methods are given in the table below.

Property	Description	Setter Methods
alignment	This represents the alignment of the nodes.	setAlignment(Double)
fillHeight	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.	setFillHeight(Double)
spacing	This represents the space between the nodes in the HBox. It is of double type.	setSpacing(Double)

- The HBox class contains two constructors that are given below.
    - HBox() : creates HBox layout with 0 spacing
    - Hbox(Double spacing) : creates HBox layout with a spacing value

Example

```
import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import javafx.scene.layout.HBox;

public class HBoxExample extends Application {
  @Override
  public void start(Stage stage) {
    //creating a text field
    TextField textField = new TextField();

    //Creating the play button
    Button playButton = new Button("Play");

    //Creating the stop button
    Button stopButton = new Button("stop");

    //Instantiating the HBox class
    HBox hbox = new HBox();

    //Setting the space between the nodes of a HBox pane
    hbox.setSpacing(10);

    //Setting the margin to the nodes
    hbox.setMargin(textField, new Insets(20, 20, 20, 20));
    hbox.setMargin(playButton, new Insets(20, 20, 20, 20));
    hbox.setMargin(stopButton, new Insets(20, 20, 20, 20));

    //retrieving the observable list of the HBox
    ObservableList list = hbox.getChildren();

    //Adding all the nodes to the observable list (HBox)
    list.addAll(textField, playButton, stopButton);

    //Creating a scene object
    Scene scene = new Scene(hbox);
```

```
    //Setting title to the Stage
    stage.setTitle("Hbox Example");

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

    //Displaying the contents of the stage
    stage.show();
  }
  public static void main(String args[]){
    launch(args);
  }
}
```

JavaFX VBox

- Instead of arranging the nodes in horizontal row, Vbox Layout Pane arranges the nodes in a single vertical column.
- It is represented by **javafx.scene.layout.VBox** class which provides all the methods to deal with the styling and the distance among the nodes.
- This class needs to be instantiated in order to implement VBox layout in our application.
- This class Provides various properties which are described in the table below.

Property	Description	Setter Methods
Alignment	This property is for the alignment of the nodes.	setAlignement(Double)
FillWidth	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.	setFillWidth(boolean)

Spacing	This property is to set some spacing among the nodes of VBox.	setSpacing(Double)

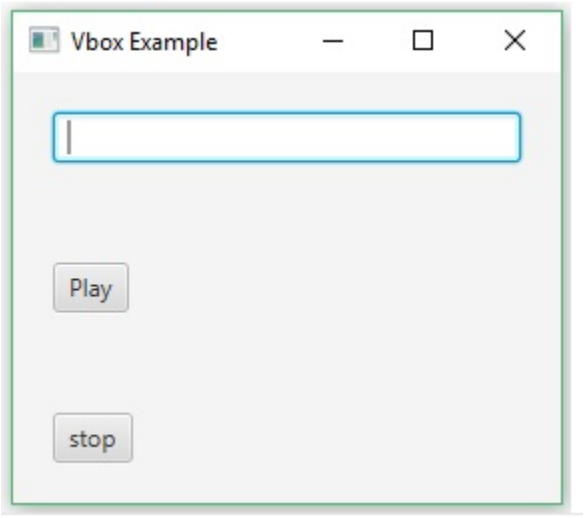
Constructors

- VBox() : creates layout with 0 spacing
- Vbox(Double spacing) : creates layout with a spacing value of double type
- Vbox(Double spacing, Node? children) : creates a layout with the specified spacing among the specified child nodes
- Vbox(Node? children) : creates a layout with the specified nodes having 0 spacing among them

Example

```
import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import javafx.scene.layout.VBox;

public class VBoxExample extends Application {
  @Override
  public void start(Stage stage) {
    //creating a text field
    TextField textField = new TextField();

    //Creating the play button
    Button playButton = new Button("Play");

    //Creating the stop button
    Button stopButton = new Button("stop");

    //Instantiating the VBox class
    VBox vBox = new VBox();

    //Setting the space between the nodes of a VBox pane
    vBox.setSpacing(10);

    //Setting the margin to the nodes
    vBox.setMargin(textField, new Insets(20, 20, 20, 20));
    vBox.setMargin(playButton, new Insets(20, 20, 20, 20));
```

```
        vBox.setMargin(stopButton, new Insets(20, 20, 20, 20));

        //retrieving the observable list of the VBox
        ObservableList list = vBox.getChildren();

        //Adding all the nodes to the observable list
        list.addAll(textField, playButton, stopButton);

        //Creating a scene object
        Scene scene = new Scene(vBox);

        //Setting title to the Stage
        stage.setTitle("Vbox Example");

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

        //Displaying the contents of the stage
        stage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```



JavaFX GridPane

- GridPane Layout pane allows us to add the multiple nodes in multiple rows and columns.
- It is seen as a flexible grid of rows and columns where nodes can be placed in any cell of the grid. It is represented by **javafx.scence.layout.GridPane** class.

- We just need to instantiate this class to implement **GridPane**.
- The properties of the class along with their setter methods are given in the table below.

Property	Description	Setter Methods
alignment	Represents the alignment of the grid within the GridPane.	setAlignment(Pos value)
gridLinesVisible	This property is intended for debugging. Lines can be displayed to show the gidpane's rows and columns by setting this property to true.	setGridLinesVisible(Boolean value)
hgap	Horizontal gaps among the columns	setHgap(Double value)
vgap	Vertical gaps among the rows	setVgap(Double value)

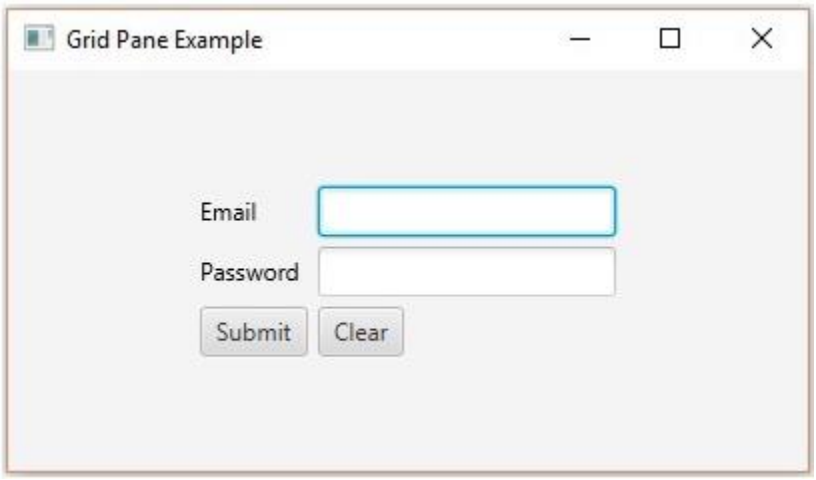
- The class contains only one constructor that is given below.

  Public GridPane(): creates a gridpane with 0 hgap/vgap

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Text;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class GridPaneExample extends Application {
  @Override
  public void start(Stage stage) {
    //creating label email
    Text text1 = new Text("Email");
```

```
//creating label password
Text text2 = new Text("Password");

//Creating Text Filed for email
TextField textField1 = new TextField();

//Creating Text Filed for password
TextField textField2 = new TextField();

//Creating Buttons
Button button1 = new Button("Submit");
Button button2 = new Button("Clear");

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

//Setting size for the pane
gridPane.setMinSize(400, 200);

//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(text1, 0, 0);
gridPane.add(textField1, 1, 0);
gridPane.add(text2, 0, 1);
gridPane.add(textField2, 1, 1);
gridPane.add(button1, 0, 2);
gridPane.add(button2, 1, 2);

//Creating a scene object
Scene scene = new Scene(gridPane);

//Setting title to the Stage
stage.setTitle("Grid Pane Example");

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

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

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



JavaFX UI Controls

- The UI elements are the one which are actually shown to the user for interaction or information exchange. Layout defines the organization of the UI elements on the screen. Behaviour is the reaction of the UI element when some event is occurred on it.
- However, the package **javafx.scene.control** provides all the necessary classes for the UI components like Button, Label, etc. Every class represents a specific UI control and defines some methods for their styling.

SN	Control	Description
1	Label	Label is a component that is used to define a simple text on the screen. Typically, a label is placed with the node, it describes.
2	Button	Button is a component that controls the function of the application. Button class is used to create a labelled button.

3	RadioButton	The Radio Button is used to provide various options to the user. The user can only choose one option among all. A radio button is either selected or deselected.
4	CheckBox	Check Box is used to get the kind of information from the user which contains various choices. User marked the checkbox either on (true) or off(false).
5	TextField	Text Field is basically used to get the input from the user in the form of text. javafx.scene.control.TextField represents TextField
6	PasswordField	PasswordField is used to get the user's password. Whatever is typed in the passwordfield is not shown on the screen to anyone.
7	HyperLink	HyperLink are used to refer any of the webpage through your appication. It is represented by the class **javafx.scene.control.HyperLink**
8	Slider	Slider is used to provide a pane of options to the user in a graphical form where the user needs to move a slider over the range of values to select one of them.
9	ProgressBar	Progress Bar is used to show the work progress to the user. It is represented by the class **javafx.scene.control.ProgressBar**.
10	ProgressIndicator	Instead of showing the analogue progress to the user, it shows the digital progress so that the user may know the amount of work done in percentage.
11	ScrollBar	JavaFX Scroll Bar is used to provide a scroll bar to the user so that the user can scroll down the application pages.
12	Menu	JavaFX provides a Menu class to implement menus. Menu is the main component of any application.
13	ToolTip	JavaFX ToolTip is used to provide hint to the user about any component. It is mainly used to provide hints about the text fields or password fields being used in the application.

JavaFX Label

- **javafx.scene.control.Label** class represents label control. As the name suggests, the label is the component that is used to place any text information on the screen. It is mainly used to describe the purpose of the other components to the user.
- You cannot set a focus on the label using the Tab key.

Constructors:

- ☞ Label(): creates an empty Label
- ☞ Label(String text): creates Label with the supplied text
- ☞ Label(String text, Node graphics): creates Label with the supplied text and graphi cs

JavaFX Button

- ☞ JavaFX button control is represented by **javafx.scene.control.Button** class.
- ☞ A button is a component that can control the behaviour of the Application. An event is generated whenever the button gets clicked.
- ☞ Button can be created by instantiating Button class. Use the following line to create button object.

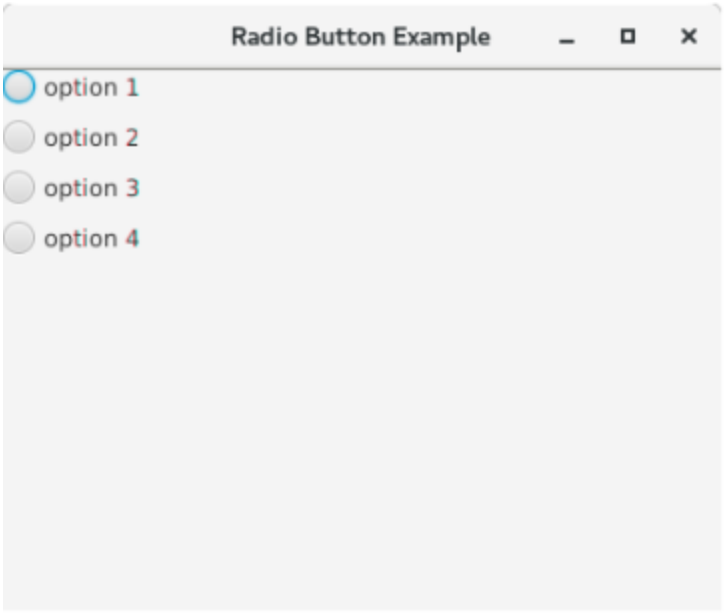
```
Button btn = new Button("My Button");
```

RadioButton

- ☞ The Radio Button is used to provide various options to the user. The user can only choose one option among all. A radio button is either selected or deselected. It can be used in a scenario of multiple choice questions in the quiz where only one option needs to be chosen by the student.
- ☞ The following code shows how one radio button is selected from a toggle group.

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.RadioButton;
```

```
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class RadioButtonTest extends Application {


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

@Override
public void start(Stage primaryStage) throws Exception {

    ToggleGroup group = new ToggleGroup();
    RadioButton button1 = new RadioButton("option 1");
    RadioButton button2 = new RadioButton("option 2");
    RadioButton button3 = new RadioButton("option 3");
    RadioButton button4 = new RadioButton("option 4");
    button1.setToggleGroup(group);
    button2.setToggleGroup(group);
    button3.setToggleGroup(group);
    button4.setToggleGroup(group);
    VBox root=new VBox();
    root.setSpacing(10);
    root.getChildren().addAll(button1,button2,button3,button4);
    Scene scene=new Scene(root,400,300);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Radio Button Example");
    primaryStage.show();
}
}
```



JavaFX CheckBox

- The Check Box is used to provide more than one choices to the user. It can be used in a scenario where the user is prompted to select more than one option or the user wants to select multiple options.
- It is different from the radiobutton in the sense that, we can select more than one checkboxes in a scenerio.
- Instantiate javafx.scene.control.CheckBox class to implement CheckBox.
- Use the following line in the code to create a blank CheckBox.

```
CheckBox checkbox = new CheckBox();
```

- Use the following line to attach a label with the checkbox.

```
CheckBox checkbox = new CheckBox("Label Name");
```

- We can change the CheckBox Label at any time by calling an instance method *setText("text").* We can make it selected by calling *setSelected("true")*

The following code implements CheckBox into our application.

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class CheckBoxTest extends Application {

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

@Override
public void start(Stage primaryStage) throws Exception {

   Label l = new Label("What do you listen:     ");
   CheckBox c1 = new CheckBox("Radio one");
   CheckBox c2 = new CheckBox("Radio Mirchi");
   CheckBox c3 = new CheckBox("Red FM");
   CheckBox c4 = new CheckBox("FM GOLD");
   HBox root = new HBox();
   root.getChildren().addAll(l,c1,c2,c3,c4);
   root.setSpacing(5);
   Scene scene=new Scene(root,800,200);
   primaryStage.setScene(scene);
   primaryStage.setTitle("CheckBox Example");
   primaryStage.show();
}
}
```

JavaFX HyperLink

- In JavaFx, we can use hyper-links to refer the web pages. It is similar to anchor links in HTML. javafx.scene.control.HyperLink class provides all the necessary methods to deal with JavaFX hyper-links.
- The following code implements HyperLink into our application.

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Hyperlink;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class HyperLinkTest extends Application {

public static void main(String[] args) {
launch(args);
}
@Override
public void start(Stage primaryStage) throws Exception {

   Hyperlink hp = new Hyperlink("http://www.cdcsit.com");
   StackPane root = new StackPane();
   hp.setOnAction(e -> System.out.println("Link Clicked"));
   root.getChildren().add(hp);
   Scene scene=new Scene(root,400,300);
   primaryStage.setScene(scene);
   primaryStage.setTitle("Hyperlink Example");
   primaryStage.show();
}
}
```

JavaFX TextField

- Text Field is basically used to get the input from the user in the form of text. **javafx.scene.control.TextField** represents **TextField**.
- It provides various methods to deal with textfields in **JavaFX. TextField** can be created by instantiating **TextField** class.
- **TextField** class provides an instance method **getText()** to retrieve the textfield data. It returns String object which can be used to save the user details in database.

JavaFX Menu

- JavaFX provides a Menu class to implement menus. Menu is the main component of a any application. In JavaFX, javafx.scene.control.Menu class provides all the methods to deal with menus. This class needs to be instantiated to create a Menu.
- The following sample of code shows the implementation of JavaFX menu.

```
ManuBar menubar = new MenuBar(); //creating MenuBar
Menu MenuName = new Menu("Menu Name"); //creating Menu
MenuItem MenuItem1 = new MenuItem("Menu Item 1 Name"); //creating Menu
Item
MenuName.getItems().add(MenuItem1); //adding Menu Item to the Menu
menubar.getMenus().add(MenuName); //adding Menu to the MenuBar
```

Example

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
public class MenuExample extends Application {
   public static void main(String[] args) {
   launch(args);
      }

   @Override
   public void start(Stage primaryStage) throws Exception {
      // TODO Auto-generated method stub
      BorderPane root = new BorderPane();
      Scene scene = new Scene(root,200,300);
      MenuBar menubar = new MenuBar();
      Menu FileMenu = new Menu("File");
      MenuItem filemenu1=new MenuItem("new");
      MenuItem filemenu2=new MenuItem("Save");
      MenuItem filemenu3=new MenuItem("Exit");
      Menu EditMenu=new Menu("Edit");
      MenuItem EditMenu1=new MenuItem("Cut");
      MenuItem EditMenu2=new MenuItem("Copy");
      MenuItem EditMenu3=new MenuItem("Paste");
```

```
        EditMenu.getItems().addAll(EditMenu1,EditMenu2,EditMenu3);
        root.setTop(menubar);
        FileMenu.getItems().addAll(filemenu1,filemenu2,filemenu3);
        menubar.getMenus().addAll(FileMenu,EditMenu);
        primaryStage.setScene(scene);
        primaryStage.show();

    }
}
```

JavaFX FileChooser

- JavaFX File chooser enables users to browse the files from the file system. **javafx.stage.FileChooser** class represents **FileChooser**. It can be created by instantiating **FileChooser** class.
- As we see in the modern day applications, there are two types of dialogues shown to the user, one is for opening the file and the other is for saving the files. In each case, the user needs to browse a location for the file and give the name to the file.
- The FileChooser class provides two types of methods,
    - showOpenDialog()
    - showSaveDialog()

The following code implements showSaveDialog() method

```
package application;
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.HBox;
import javafx.stage.FileChooser;
```

```
import javafx.stage.Stage;
public class FileChooserExample extends Application{
   @Override
   public void start(Stage primaryStage) throws Exception {
      FileChooser file = new FileChooser();
      file.setTitle("Open File");
      file.showOpenDialog(primaryStage);
      HBox root = new HBox();
      root.setSpacing(20);
      Scene scene = new Scene(root,350,100);
      primaryStage.setScene(scene);
      primaryStage.setTitle("FileChooser Example");
      primaryStage.show();
   }
   public static void main(String[] args) {
      launch(args);
   }
}
```

JavaFX Tooltip

- JavaFX Tool tip is used to provide hint to the user about any component. It is mainly used to provide hints about the text fields or password fields being used in the application.

- It can be created by instantiating javafx.scene.control.Tooltip class. The following code implements Tooltip about a password field to the user.

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.PasswordField;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class ProgressBarTest extends Application {
public void start(Stage primaryStage) throws Exception {
   PasswordField pf = new PasswordField();
   Tooltip tool=new Tooltip();
   StackPane root = new StackPane();
   tool.setText("Information");
   pf.setTooltip(tool);
   root.getChildren().add(pf);
   Scene scene = new Scene(root,300,200);
   primaryStage.setScene(scene);
   primaryStage.setTitle("ToolTip Example");
   primaryStage.show();
```

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