

CHAPTER 7

JAVAFX Basic ,Event Driven Programming and Animations



SUBJECT: OOP-I
CODE: 3140705

PREPARED BY:
ASST. PROF. NENSI KANSAGARA
(CSE DEPARTMENT, ACET)

PART-I JAVAFX Basic

Basic Structure of JAVAFX Program

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.

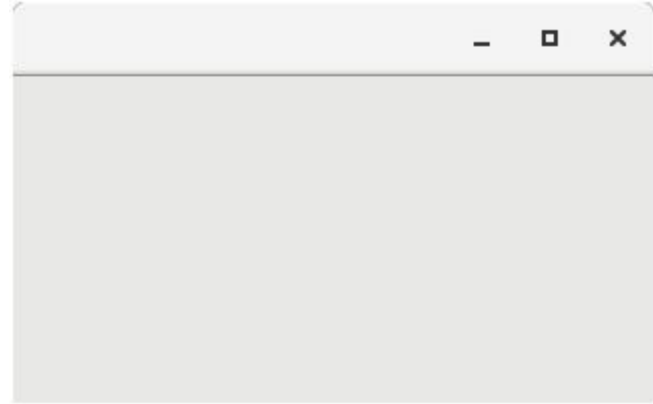
- public void init()
- public abstract void start(Stage primaryStage)
- public void stop()

in order to create a basic JavaFX application, we need to:

1. Import **javafx.application.Application** into our code.
2. Inherit **Application** into our class.
3. Override **start()** method of Application class.

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 like following.

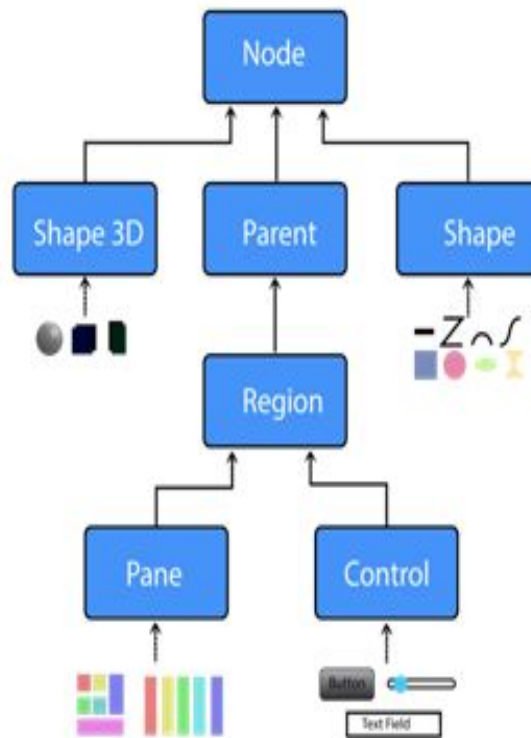
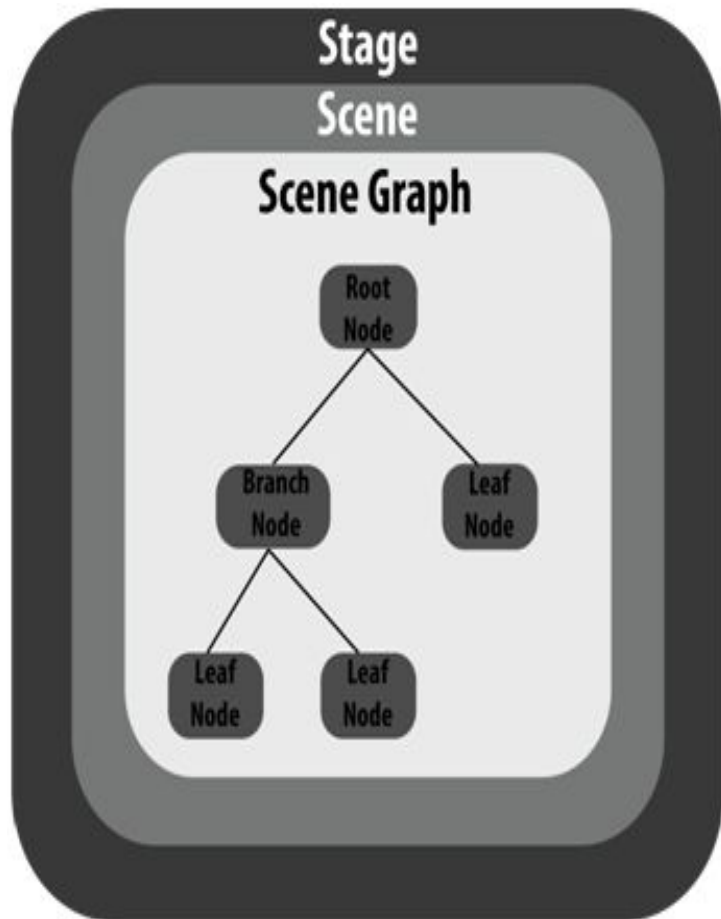


Scene

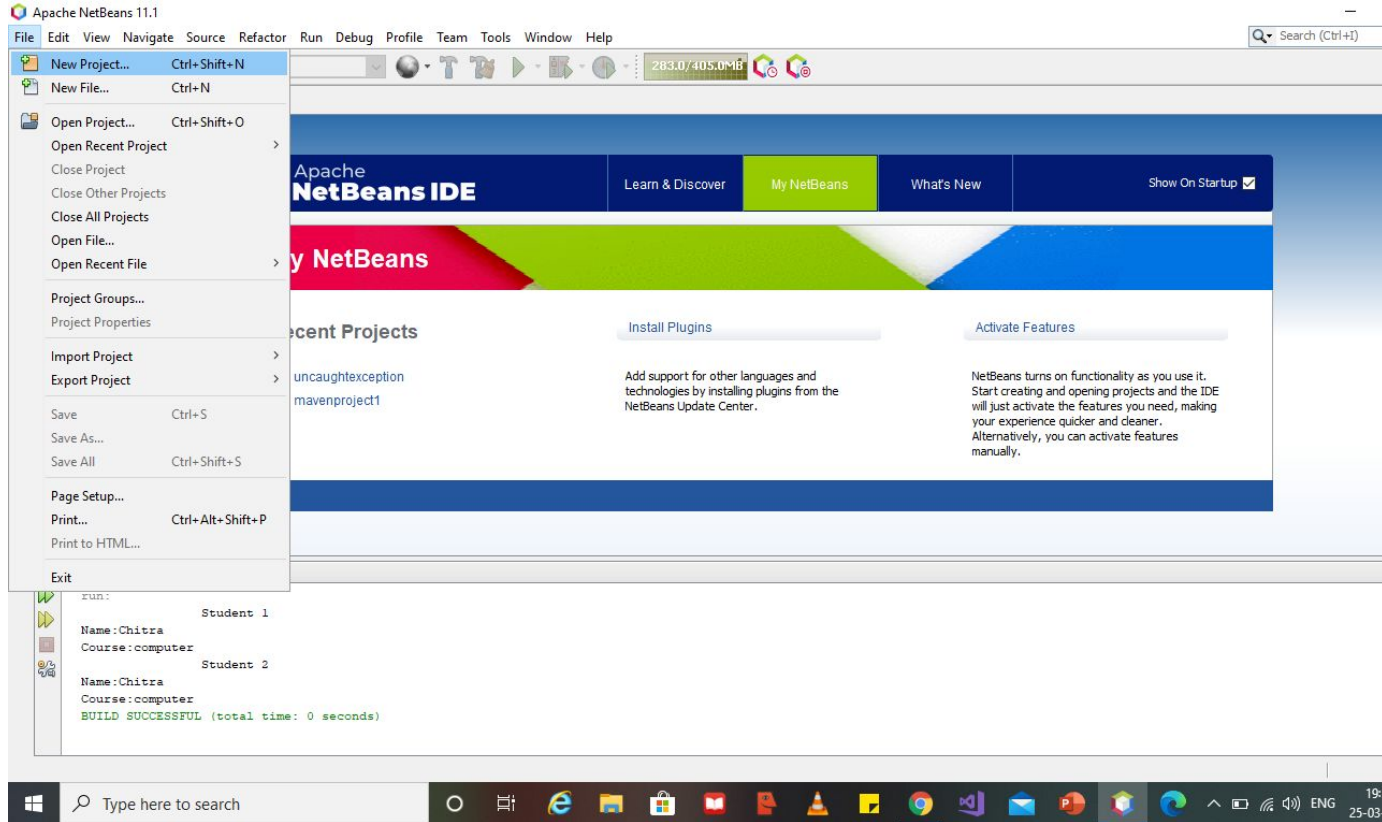
- ★ Scene actually holds all the physical contents (nodes) of a JavaFX application. **javafx.scene.Scene** class provides all the methods to deal with a scene object. Creating scene is necessary in order to visualize the contents on the stage.
- ★ At one instance, the scene object can only be added to 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. We will discuss Scene class and its method later in detail.

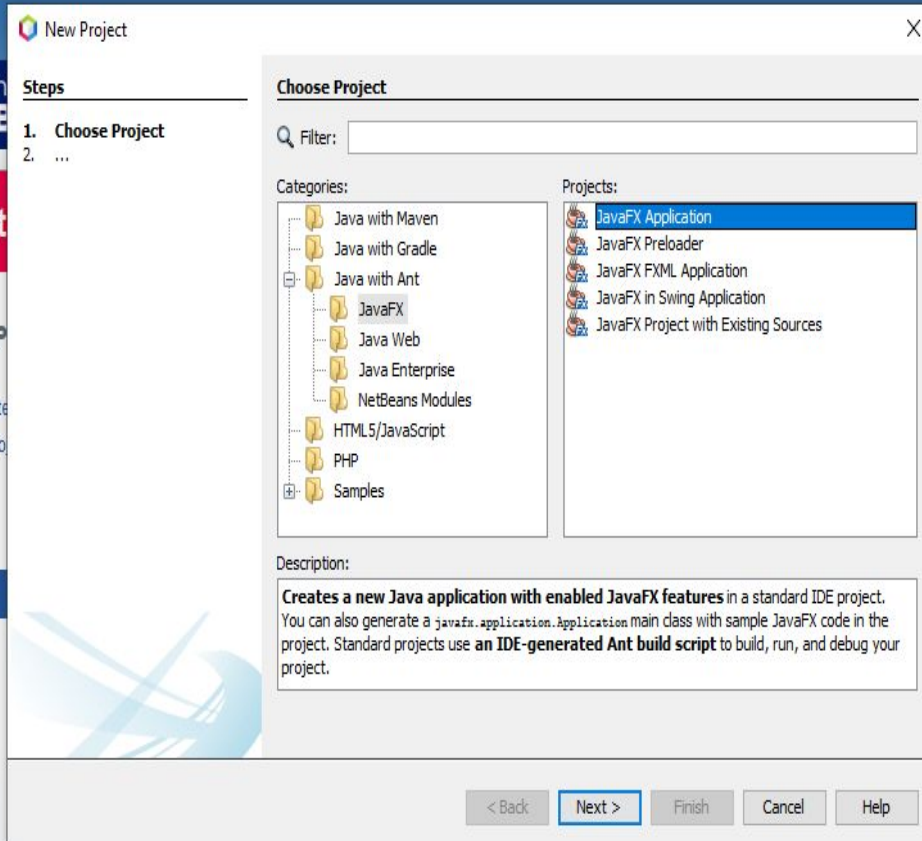
Scene Graph

- ★ 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.



Writing a First JAVAFX Program





New JavaFX Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name:

Project Location:

Project Folder:

JavaFX Platform:

☐ Create Custom Preloader

Project Name:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☒ Create Application Class

```
package myfirstjavafxapplication;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.control.Label;

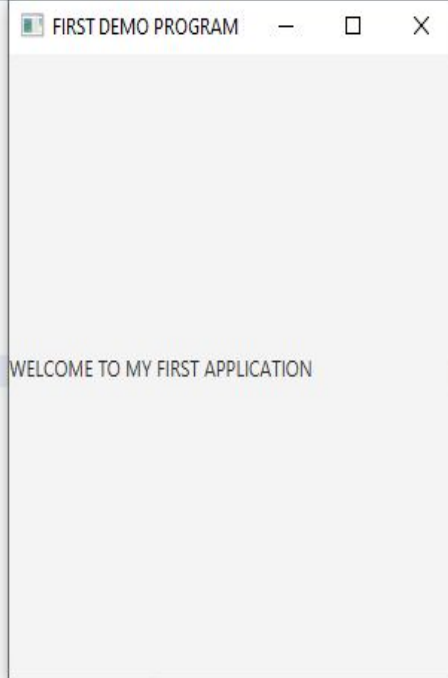
public class MyfirstJavaFXApplication extends Application {

    @Override
    public void start(Stage primaryStage) {
        Label L = new Label("WELCOME TO MY FIRST APPLICATION");

        Scene scene = new Scene(L, 300, 350);

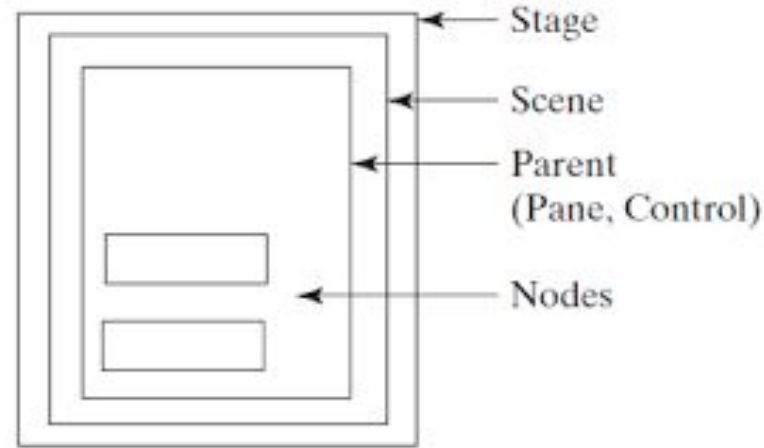
        primaryStage.setTitle("FIRST DEMO PROGRAM");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



Panes, UI Control and Shapes

- ★ Pane is a container class using which the UI components can be placed at any desired location with any desired size.
- ★ Node is any visual component such as UI Controls, Shapes, or a image view.
- ★ UI Controls refer to label, button, checkbox, radio button and so on.
- ★ Shapes refer to lines, rectangle, circle and so on.
- ★ A Scene can be displayed in a stage.



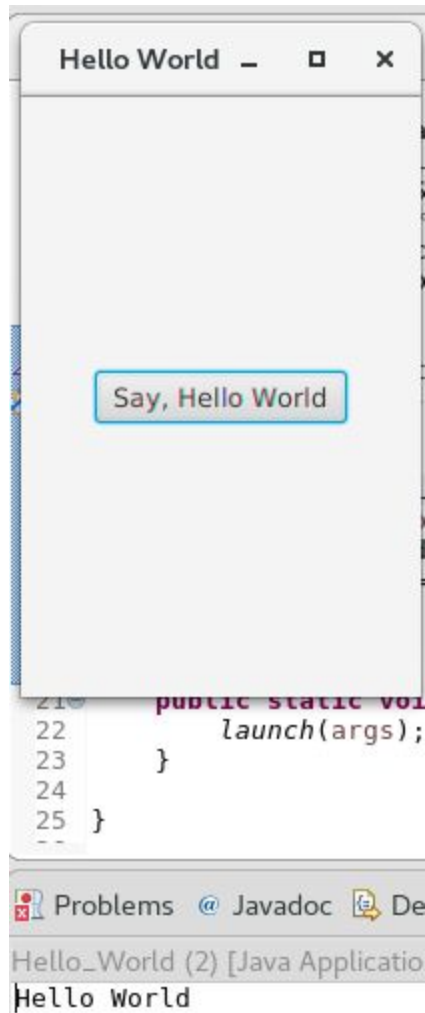
```

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class JavaFXApplicationButton extends Application{
    @Override
    public void start(Stage primaryStage) throws Exception {
        // TODO Auto-generated method stub
        Button btn1=new Button("Say, Hello World");
        btn1.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent arg0) {
                // TODO Auto-generated method stub
                System.out.println("hello world");
            }
        });
        StackPane root=new StackPane();
        root.getChildren().add(btn1);
        Scene scene=new Scene(root,600,400);
        primaryStage.setTitle("First JavaFX Application");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main (String[] args)
    {
        launch(args);
    }
}

```

OUTPUT:



Property Binding

- ★ JAVAFX introduces a new concept called property binding that enables a target object to be bound to a source object.
- ★ The target object is simply called a binding object or a binding property.
- ★ Method:

`target.bind(source)`

- ★ The bind method defines in `javafx.beans.property.Property`
- ★ A binding property(target) is an instance of `javafx.beans.property.Property`
- ★ A source object is an instance of the `javafx.beans.value.ObservableValue`

```
package javafxpropertybinding;

import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class JavaFXPropertyBinding {

    public static void main(String[] args) {
        DoubleProperty num1 =new SimpleDoubleProperty(10);
        DoubleProperty num2 = new SimpleDoubleProperty(20);
        num1.bind(num2);
        System.out.println("Num1 =" +num1.getValue());
        System.out.println("Num2 =" +num1.getValue());
        num2.setValue(555);
        System.out.println("Num1 =" +num1.getValue());
        System.out.println("Num1 =" +num2.getValue());
    }
}
```


The Color and the Font Class

- ★ In JavaFX, we can fill the shapes with the colors. We have the flexibility to create our own color using the several methods and pass that as a Paint object into the **setFill()** method. Let's discuss the several methods of creating color in JavaFX.
- ★ **RGB Color**
- ★ RGB color system is the most popular method to create a color in graphics. It consists of three components named as RED \rightarrow R, GREEN \rightarrow G and BLUE \rightarrow B. Each component uses 8 Bits that means every component can have the integer value from 0 to $2^8 - 1 = 255$.
- ★ The computer screen can be seen as the collection of pixels. The set (R,G,B) actually represents the emission of their respective LEDs on the screen.

- ★ If the value of **RED** is set to 0 then it means that the Red LED is turned off while the value 255 indicates that the full emission of LED is being there. The combination of (0,0,0) represents the black color while (255,255,255) represents the white color. The middle values in that range can represent different colors.
- ★ Using the superimposition of RGB, we can represent $255 \times 255 \times 255$ different colors. In JavaFX, the class `javafx.scene.paint.Color` class represents colors.
- ★ There is a static method named as **rgb()** of Color class. It accepts three integer arguments as Red, Green, Blue and one optional double argument called alpha. The value of alpha is proportional to the opacity of the color. The alpha value 0 means that the color is completely transparent while the value 1 means that the color is completely opaque.

```

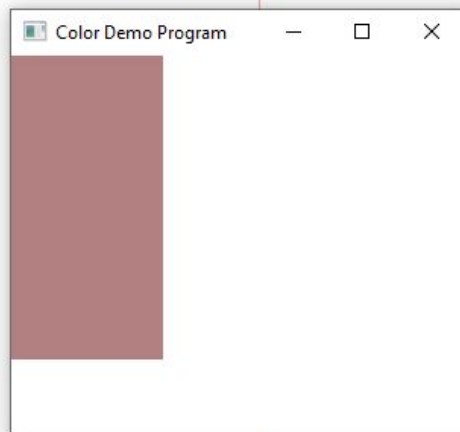
package javafxcolor;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.Shadow;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
public class JavaFXColor extends Application {

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        primaryStage.setTitle("Color Demo Program");
        Rectangle rect = new Rectangle(100,200);

        int red=100;
        int green=0;
        int blue=0;
        rect.setFill(Color.rgb(red, green, blue,0.5));
        root.getChildren().add(rect);
        Scene scene = new Scene(root,300,250);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

```



The font class/text class:

we need to provide the text based information on the interface of our application. JavaFX library provides a class named **javafx.scene.text.Text** for this purpose. This class provides various methods to alter various properties of the text. We just need to instantiate this class to implement text in our application.

Property	Description	Setter Methods
boundstype	This property is of object type. It determines the way in which the bounds of the text is being calculated.	setBoundsType(TextBoundsType value)
font	Font of the text.	setFont(Font value)
fontsmoothingType	Defines the requested smoothing type for the font.	setFontSmoothingType(FontSmoothingType value)
linespacing	Vertical space in pixels between the lines. It is double type property.	setLineSpacing(double spacing)
strikethrough	This is a boolean type property. We can put a line through the text by setting this property to true.	setStrikeThrough(boolean value)
textalignment	Horizontal Text alignment	setTextAlignment(TextAlignment value)
textorigin	Origin of text coordinate system in local coordinate system.	setTextOrigin(VPos value)
text	It is a string type property. It defines the text string which is to be displayed.	setText(String value)
underline	It is a boolean type property. We can underline the text by setting this property to true.	setUnderLine(boolean value)
wrappingwidth	Width limit for the text from where the text is to be wrapped. It is a double type property.	setWidth(double value)

```

package javafxtext1;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
import javafx.scene.text.Text;

public class JavaFXText1 extends Application {

    @Override
    public void start(Stage primaryStage) {
        Text text = new Text();
        text.setText("Hello !! Welcome to First JAVAFX TEXT");
        StackPane root = new StackPane();
        Scene scene = new Scene(root, 300, 400);
        root.getChildren().add(text);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Text Demo program");
        primaryStage.show();
    }

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

```



Font and position of the text

- ★ JavaFX enables us to apply various fonts to the text nodes. We just need to set the property **font** of the Text class by using the setter method **setFont()**. This method accepts the object of **Font** class.
- ★ The class **Font** belongs to the package **javafx.scene.text**. It contains a static method named **font()**. This returns an object of **Font** type which will be passed as an argument into the **setFont()** method of Text class. The method **Font.font()** accepts the following parameters.
- ★ **Family:** it represents the family of the font. It is of string type and should be an appropriate font family present in the system.

cont..

- ★ **Weight:** this Font class property is for the weight of the font. There are 9 values which can be used as the font weight. The values are **FontWeight.BLACK, BOLD, EXTRA_BOLD, EXTRA_LIGHT, LIGHT, MEDIUM, NORMAL, SEMI_BOLD, THIN.**
- ★ **Posture:** this Font class property represents the posture of the font. It can be either **FontPosture.ITALIC** or **FontPosture.REGULAR.**
- ★ **Size:** this is a double type property. It is used to set the size of the font.

The Syntax of the method setFont() is given below.

```
<text_object>.setFont(Font.font(<String font_family>, <FontWeight>,  
<FontPosture>, <FontSize>))
```



```

package javafxtext2;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.scene.paint.Color;

public class JavaFXText2 extends Application {

    @Override
    public void start(Stage primaryStage) {
        Text text = new Text();

        text.setX(100);
        text.setY(20);
        text.setFont(Font.font("comic sans MS", FontWeight.BOLD, FontPosture.ITALIC, 20));
        text.setFill(Color.RED);
        text.setStroke(Color.BLACK); // setting the stroke for the text
        text.setStrokeWidth(1); // setting stroke width to 2
        text.setText("Welcome to First JAVA FX FONT CLASS");
        Group root = new Group();
        Scene scene = new Scene(root, 500, 200);
        root.getChildren().add(text);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Text DEMO Program");
        primaryStage.show();
    }
}

```



The image and image-view class:

- ★ You can load and modify images using the classes provided by JavaFX in the package `javafx.scene.image`. JavaFX supports the image formats like Bmp, Gif, Jpeg, Png.

Loading an Image

- ★ You can load an image in JavaFX by instantiating the class named `Image` of the package `javafx.scene.image`.
- ★ To the constructor of the class, you have to pass either of the following –
 - An `InputStream` object of the image to be loaded or,
 - A string variable holding the URL for the image.

//Passing FileInputStream object as a parameter

```
FileInputStream inputstream = new FileInputStream("C:\\images\\image.jpg");
```

```
Image image = new Image(inputstream);
```

//Loading image from URL

```
//Image image = new Image(new FileInputStream("url for the image));
```

After loading the image, you can set the view for the image by instantiating the ImageView class and passing the image to its constructor as follows –

```
ImageView imageView = new ImageView(image);
```

```
package javafximage;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

public class JavaFXImage extends Application {

    @Override
    public void start(Stage stage) throws FileNotFoundException {
        //Creating an image
        Image image = new Image(new FileInputStream("C:\\\\Users\\Lenovo\\Desktop\\oop lecture\\2.jpg"));

        //Setting the image view
        ImageView imageView = new ImageView(image);

        //Setting the position of the image
        imageView.setX(50);
        imageView.setY(25);

        //setting the fit height and width of the image view
        imageView.setFitHeight(455);
        imageView.setFitWidth(500);
    }
}
```

```
//Setting the preserve ratio of the image view
imageView.setPreserveRatio(true);

//Creating a Group object
Group root = new Group(imageView);

//Creating a scene object
Scene scene = new Scene(root, 600, 500);

//Setting title to the Stage
stage.setTitle("Image DEMO program");

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

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

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    launch(args);
}
```

Image DEMO program



Layout Panes and Shapes

LAYOUT PANES:

- ★ 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.

Layout Classes:

★ **javafx.scene.layout** Package provides various classes that represents the layouts. The classes are described in the table below.

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.

1. Instantiate the respective layout class, for example, **HBox root = new HBox();**
2. Setting the properties for the layout, for example, **root.setSpacing(20);**
3. Adding nodes to the layout object, for example,
root.getChildren().addAll(<NodeObjects>);

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)

HBox Constructor:

The HBox class contains two constructors that are given below.

1. **new HBox()** : create HBox layout with 0 spacing
2. **new Hbox(Double spacing)** : create HBox layout with a spacing value

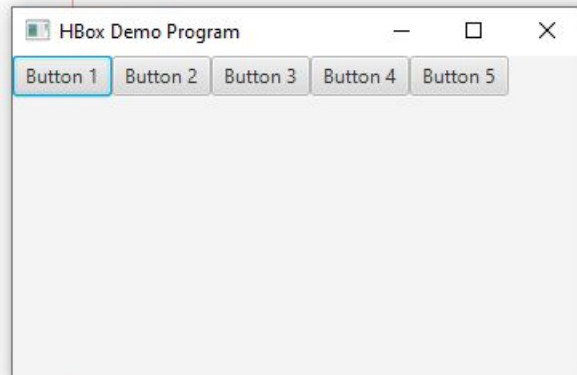
```
package javafxhbox;

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

public class JavaFXHBox extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception
    {
        Button btn1 = new Button("Button 1");
        Button btn2 = new Button("Button 2");
        Button btn3 = new Button("Button 3");
        Button btn4 = new Button("Button 4");
        Button btn5 = new Button("Button 5");
        HBox root = new HBox();
        Scene scene = new Scene(root,200,200);
        root.getChildren().addAll(btn1,btn2,btn3,btn4,btn5);
        primaryStage.setScene(scene);
        primaryStage.setTitle("HBox Demo Program");
        primaryStage.show();
    }

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



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 Method Provides various properties which are described in the table below.

Property	Description	Setter Methods
Alignment	This property is for the alignment of the nodes.	<code>setAlignment(Double)</code>
FillWidth	This property is of the boolean type. The Width of resizable nodes can be made equal to the Width of the VBox by setting this property to true.	<code>setFillWidth(boolean)</code>
Spacing	This property is to set some spacing among the nodes of VBox.	<code>setSpacing(Double)</code>

VBox Constructors:

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

```

package javafxapplicationvbox;

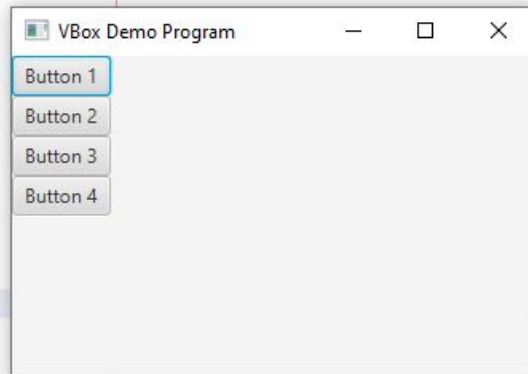
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class JavaFXApplicationVBox extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("Button 1");
        Button btn2 = new Button("Button 2");
        Button btn3 = new Button("Button 3");
        Button btn4 = new Button("Button 4");
        VBox root = new VBox();
        Scene scene = new Scene(root, 200, 200);
        root.getChildren().addAll(btn1, btn2, btn3, btn4);
        primaryStage.setTitle("VBox Demo Program");
        primaryStage.setScene(scene);
        primaryStage.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 two constructors that are given below.

1. StackPane()
2. StackPane(Node? Children)

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)

```

package javafxapplicationstackpane;

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

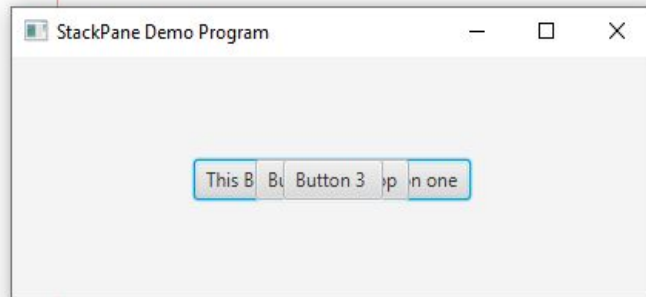
public class JavaFXApplicationStackPane extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("This Button is with caption one");
        Button btn2 = new Button("Button 2 on top");
        Button btn3 = new Button("Button 3 ");
        // Button btn4 = new Button("Button 4");

        StackPane root = new StackPane();
        Scene scene = new Scene(root, 100, 150);
        root.getChildren().addAll(btn1, btn2, btn3);
        primaryStage.setScene(scene);
        primaryStage.setTitle("StackPane Demo Program");
        primaryStage.show();
    }

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

```



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.scene.layout.GridPane** class. We just need to instantiate this class to implement GridPane.
- ★ The class contains only one constructor that is given below

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

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 gridpane'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)

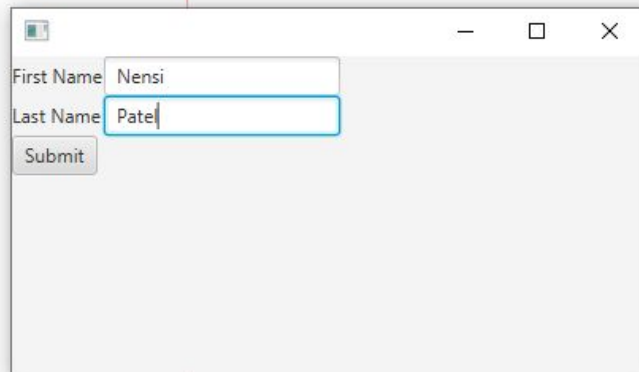
```
package javafxapplicationgridpane;

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;

public class JavaFXApplicationGridPane extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        Label first_name=new Label("First Name");
        Label last_name=new Label("Last Name");
        TextField tf1=new TextField();
        TextField tf2=new TextField();
        Button Submit=new Button ("Submit");
        GridPane root=new GridPane();
        Scene scene = new Scene(root,400,200);
        root.addRow(0, first_name,tf1); //addRow(int rowIndex,Node..,Childern)
        root.addRow(1, last_name,tf2);
        root.addRow(2, Submit);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



First Name Nensi

Last Name Pate

Submit

FlowPane

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

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)

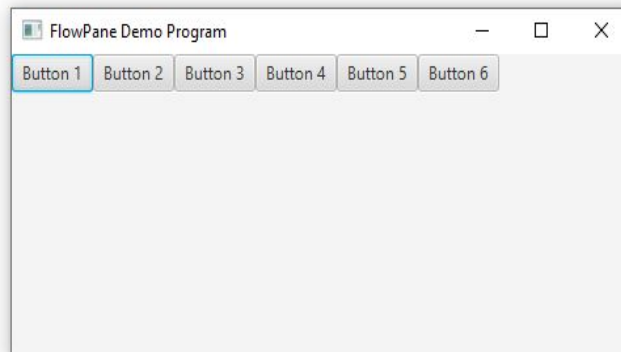
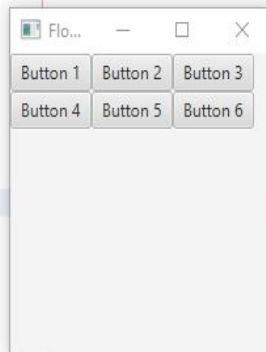
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;

import javafx.stage.Stage;
public class JavaFXApplicationFlowPane extends Application {
```

```
    @Override
    public void start(Stage primaryStage) {
        Button btn1 = new Button("Button 1");
        Button btn2 = new Button("Button 2");
        Button btn3 = new Button("Button 3");
        Button btn4 = new Button("Button 4");
        Button btn5 = new Button("Button 5");
        Button btn6 = new Button("Button 6");

        FlowPane root = new FlowPane();
        Scene scene = new Scene(root, 200, 200);
        root.getChildren().addAll(btn1, btn2, btn3, btn4, btn5, btn6);
        primaryStage.setTitle("FlowPane Demo Program");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

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



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.
- ★ 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

```
package javafxapplicationboredrpane;

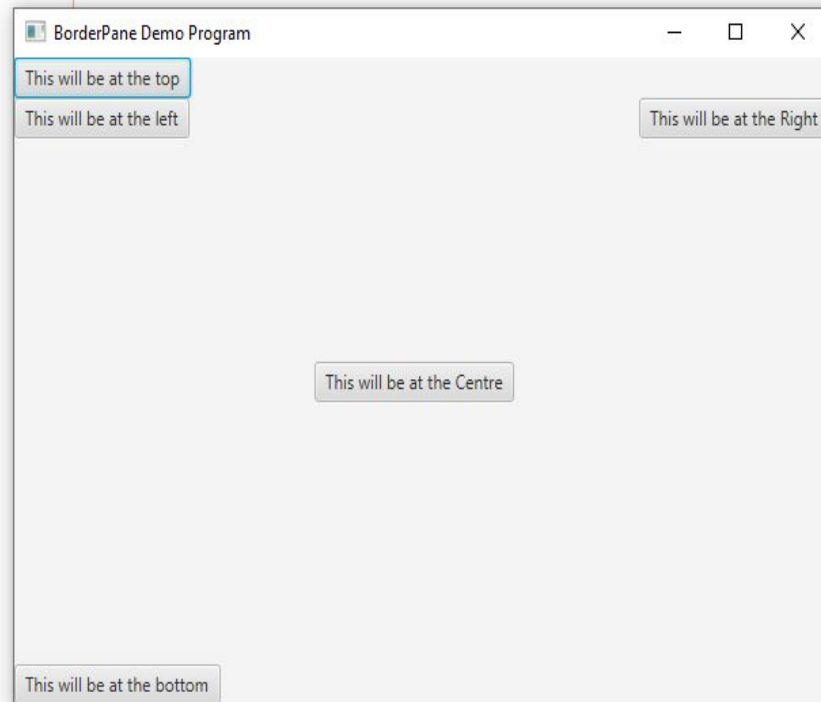
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class JavaFXApplicationBoredrPane extends Application {

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

        BorderPane BPane = new BorderPane();
        BPane.setTop(new Button("This will be at the top"));
        BPane.setLeft(new Button("This will be at the left"));
        BPane.setRight(new Button("This will be at the Right"));
        BPane.setCenter(new Button("This will be at the Centre"));
        BPane.setBottom(new Button("This will be at the bottom"));
        Scene scene = new Scene(BPane, 600, 400);
        primaryStage.setScene(scene);
        primaryStage.setTitle("BorderPane Demo Program");
        primaryStage.show();
    }

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



Shapes

- ★ In some of the applications, we need to show two dimensional shapes to the user. However, JavaFX provides the flexibility to create our own 2D shapes on the screen .
- ★ There are various classes which can be used to implement 2D shapes in our application. All these classes resides in **javafx.scene.shape** package.
- ★ This package contains the classes which represents different types of 2D shapes. There are several methods in the classes which deals with the coordinates regarding 2D shape creation.

Line:

- ★ In general, Line can be defined as the geometrical structure which joins two points (X1,Y1) and (X2,Y2) in a X-Y coordinate plane. JavaFX allows the developers to create the line on the GUI of a JavaFX application. JavaFX library provides the class **Line** which is the part of **javafx.scene.shape** package.
- ★ How to create a Line?
- ★ Follow the following instructions to create a Line.
 - Instantiate the class **javafx.scene.shape.Line**.
 - set the required properties of the class object.
 - Add class object to the group

Properties:

Property	Description	Setter Methods
endX	The X coordinate of the end point of the line	setEndX(Double)
endY	The y coordinate of the end point of the line	setEndY(Double)
startX	The x coordinate of the starting point of the line	setStartX(Double)
startY	The y coordinate of the starting point of the line	setStartY(Double)

```
package javafxapplicationline;

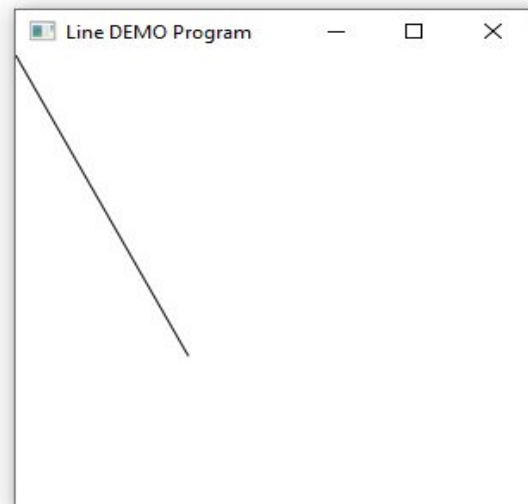
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Line;
import javafx.stage.Stage;

public class JavaFXApplicationLine extends Application {

    @Override
    public void start(Stage primaryStage) {
        // TODO Auto-generated method stub
        Line line = new Line(); //instantiating Line class
        line.setStartX(0); //setting starting X point of Line
        line.setStartY(0); //setting starting Y point of Line
        line.setEndX(100); //setting ending X point of Line
        line.setEndY(200); //setting ending Y point of Line
        Group root = new Group(); //Creating a Group
        root.getChildren().add(line); //adding the class object //to the group
        Scene scene = new Scene(root,300,300);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Line DEMO Program");
        primaryStage.show();
    }

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

}
```



```

package javafxapplicationline;

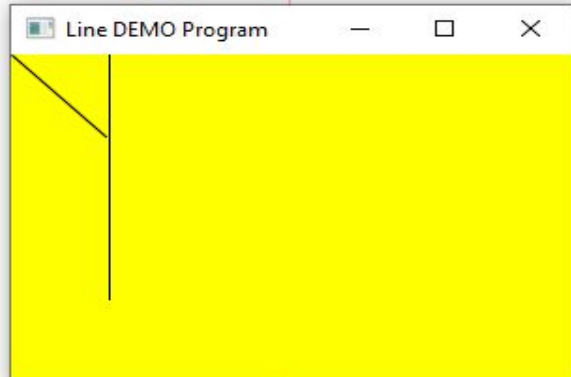
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.shape.Line;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class JavaFXApplicationLine extends Application {

    @Override
    public void start(Stage primaryStage) {
        // TODO Auto-generated method stub
        HBox root=new HBox();
        Line line1 = new Line(0,0,50,50); //instantiating Line class
        Line line2= new Line(50,50,50,200);
        // Group root = new Group(); //Creating a Group
        root.getChildren().addAll(line1,line2); //adding the class object //to the group
        Scene scene = new Scene(root,300,200,Color.YELLOW);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Line DEMO Program");
        primaryStage.show();
    }

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

```



Rectangle:

In general, Rectangles can be defined as the geometrical figure consists of four sides, out of which, the opposite sides are always equal and the angle between the two adjacent sides is 90 degree. A Rectangle with four equal sides is called square.

JavaFX library allows the developers to create a rectangle by instantiating **javafx.scene.shape.Rectangle** class

Properties

Property	Description	Setter Method
ArcHeight	Vertical diameter of the arc at the four corners of rectangle	setArcHeight(Double height)
ArcWidth	Horizontal diameter of the arc at the four corners of the rectangle	setArcWidth(Double Width)
Height	Defines the height of the rectangle	setHeight(Double height)
Width	Defines the width of the rectangle	setWidth(Double width)
X	X coordinate of the upper left corner	setX(Double X-value)
Y	Y coordinate of the upper left corner	setY(Double(Y-value)

```
package javafxapplicationrectangle;
```

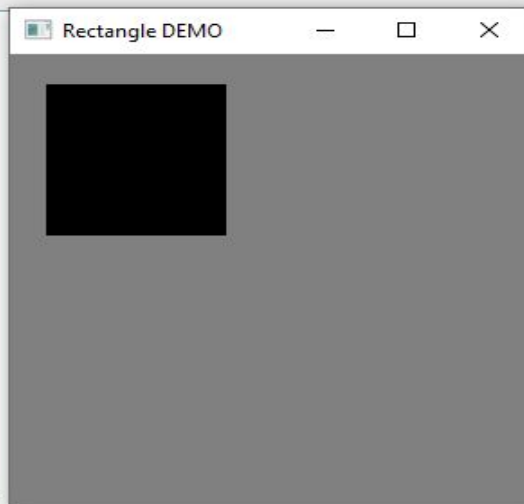
```
import javafx.application.Application;  
import javafx.scene.Group;  
import javafx.scene.Scene;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
import javafx.stage.Stage;
```

```
public class JavaFXApplicationRectangle extends Application {
```

```
    @Override
```

```
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("Rectangle DEMO");  
        Group group = new Group(); //creating Group  
        Rectangle rect=new Rectangle(); //instantiating Rectangle  
        rect.setX(20); //setting the X coordinate of upper left //corner of rectangle  
        rect.setY(20); //setting the Y coordinate of upper left //corner of rectangle  
        rect.setWidth(100); //setting the width of rectangle  
        rect.setHeight(100); // setting the height of rectangle  
        group.getChildren().addAll(rect); //adding rectangle to the //group  
        Scene scene = new Scene(group,200,300,Color.GRAY);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }
```

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



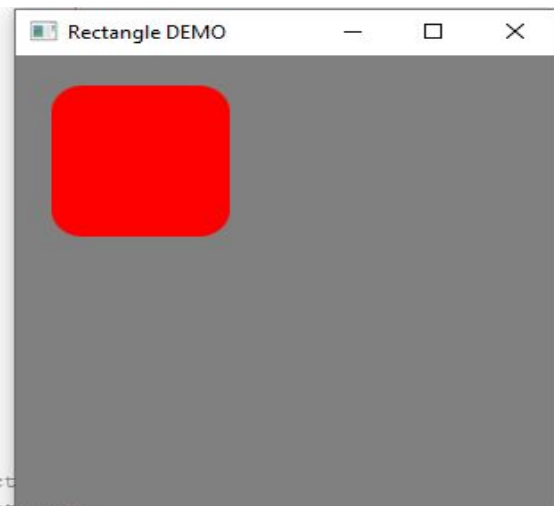
```
package javafxapplicationrectangle;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class JavaFXApplicationRectangle extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Rectangle DEMO");
        Group group = new Group(); //creating Group
        Rectangle rect=new Rectangle(); //instantiating Rectangle
        rect.setX(20); //setting the X coordinate of upper left //corner of rect
        rect.setY(20); //setting the Y coordinate of upper left //corner of rectangle
        rect.setWidth(100);
        rect.setHeight(100);
        rect.setArcHeight(35);
        rect.setArcWidth(35);
        rect.setFill(Color.RED);
        group.getChildren().addAll(rect);
        Scene scene = new Scene(group,200,300,Color.GRAY);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



Circle:

A circle is a special type of ellipse with both of the focal points at the same position. Its horizontal radius is equal to its vertical radius. JavaFX allows us to create Circle on the GUI of any application by just instantiating **javafx.scene.shape.Circle** class. Just set the class properties by using the instance setter methods and add the class object to the Group.

Properties

The class properties along with the setter methods and their description are given below in the table.

Property	Description	Setter Methods
centerX	X coordinate of the centre of circle	setCenterX(Double value)
centerY	Y coordinate of the centre of circle	setCenterY(Double value)
radius	Radius of the circle	setRadius(Double value)

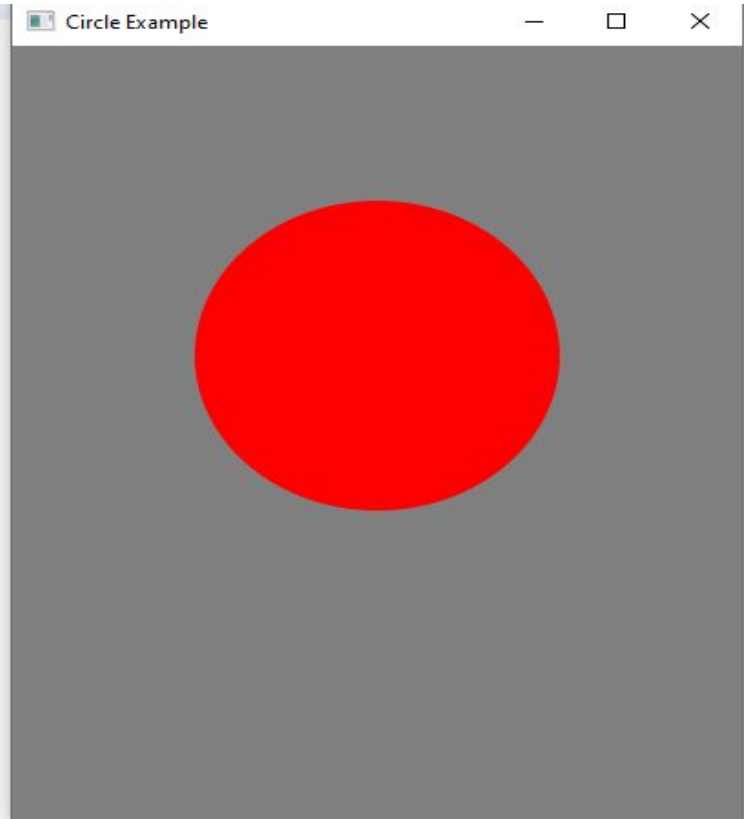
```
package javafxapplicationcircle;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class JavaFXApplicationCircle extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Circle Example");
        Group group = new Group();
        Circle circle = new Circle();
        circle.setCenterX(200);
        circle.setCenterY(200);
        circle.setRadius(100);
        circle.setFill(Color.RED);
        group.getChildren().addAll(circle);
        Scene scene = new Scene(group, 400, 500, Color.GRAY);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



Ellipse

In general, ellipse can be defined as the geometrical structure with the two focal points. The focal points in the ellipse are chosen so that the sum of the distance to the focal points is constant from every point of the ellipse.

In JavaFX, the class **javafx.scene.shape.Ellipse** represents Ellipse. This class needs to be instantiated in order to create ellipse. This class contains various properties which needs to be set in order to render ellipse on a XY place.

Properties

Property	Description	Setter Methods
CenterX	Horizontal position of the centre of eclipse	setCenterX(Double X-value)
CenterY	Vertical position of the centre of eclipse	setCenterY(Double Y-value)
RadiusX	Width of Eclipse	setRadiusX(Double X-Radius Vaue)
RadiusY	Height of Eclipse	setRadiusY(Double Y-Radius Value)

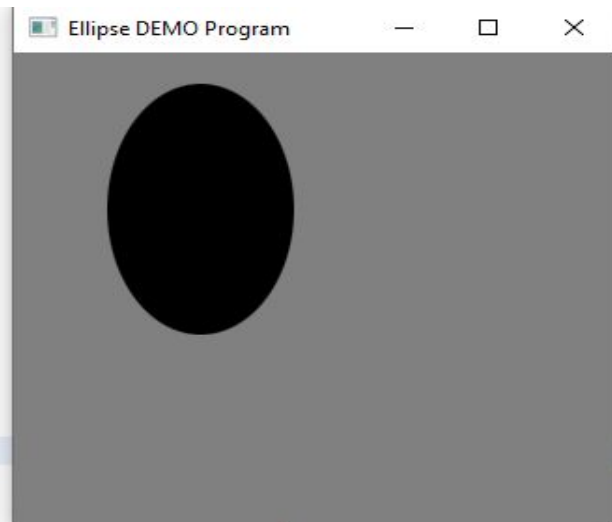
```
package javafxapplicationellipse;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Ellipse;
import javafx.stage.Stage;

public class JavaFXApplicationEllipse extends Application {

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Ellipse DEMO Program");
        Group group = new Group();
        Ellipse ellipse = new Ellipse();
        ellipse.setCenterX(100);
        ellipse.setCenterY(100);
        ellipse.setRadiusX(50);
        ellipse.setRadiusY(80);
        group.getChildren().addAll(ellipse);
        Scene scene = new Scene(group, 200, 300, Color.GRAY);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



Arc

In general, Arc is the part of the circumference of a circle or ellipse. It needs to be created in some of the JavaFX applications wherever required. JavaFX allows us to create the Arc on GUI by just instantiating **javafx.scene.shape.Arc** class. Just set the properties of the class to the appropriate values to show arc as required by the Application.

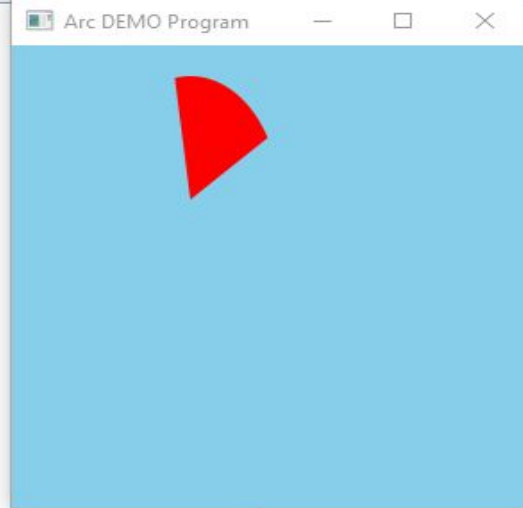
Properties

JavaFX Arc properties and their setter method are given in the table below.

Property	Description	Method
CenterX	X coordinate of the centre point	serCenterX(Double value)
CenterY	Y coordinate of the centre point	setCenterY(Double value)
Length	Angular extent of the arc in degrees	setLength(Double value)
RadiusX	Full width of the ellipse of which, Arc is a part.	setRadiusX(Double value)
RadiusY	Full height of the ellipse of which, Arc is a part	setRadiusY(Double value)
StartAngle	Angle of the arc in degrees	setStartAngle(Double value)
type	Type of Arc : OPEN, CHORD, ROUND	setType(Double value)

```
package javafxapplicationarc;
```

```
] import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcType;
import javafx.stage.Stage;
- import javafx.stage.Stage;
public class JavaFXApplicationArc extends Application {
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Arc DEMO Program");
        Group group = new Group();
        Arc arc = new Arc();
        arc.setCenterX(100);
        arc.setCenterY(100);
        arc.setRadiusX(50);
        arc.setRadiusY(80);
        arc.setStartAngle(30);
        arc.setLength(70);
        arc.setType(ArcType.ROUND);
        arc.setFill(Color.RED);
        group.getChildren().addAll(arc);
        Scene scene = new Scene(group, 200, 300, Color.SKYBLUE);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
- }
[ public static void main(String[] args) {
    launch(args);
- }
```



Polygons:

- ★ Polygon can be defined as a plain figure with at least three straight sides forming a loop. In the case of polygons, we mainly consider the length of its sides and the interior angles. Triangles, squares, Pentagons, Hexagons, etc are all polygons.
- ★ In JavaFX, Polygon can be created by instantiating **javafx.scene.shape.Polygon** class. We need to pass a Double array into the class constructor representing X-Y coordinates of all the points of the polygon. The syntax is given below.

```
Polygon poly = new Polygon(DoubleArray);
```

- ★ We can also create polygon by anonymously calling **addAll()** method on the reference returned by calling **getPoints()** method which is an instance method of Polygon class. However, we need to pass the double array into this method, which represents X-Y coordinates of the polygon. The syntax is given below.

```
Polygon polygon_object = new Polygon();
```

```
Polygon_Object.getPoints().addAll(Double_Array);
```

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Polygon;
import javafx.stage.Stage;

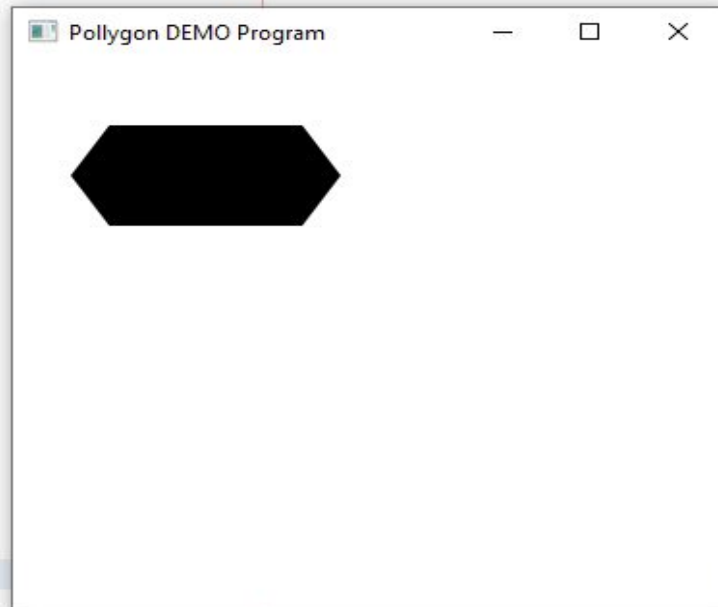
public class JavaFXApplicationPolygons extends Application {

    @Override
    public void start(Stage primaryStage) {
        Group root = new Group();
        primaryStage.setTitle("Pollygon DEMO Program");

        Polygon polygon = new Polygon();
        polygon.getPoints().addAll(new Double[]{
            50.0, 40.0,
            30.0, 70.0,
            50.0, 100.0,
            150.0, 100.0,
            170.0, 70.0,
            150.0, 40.0});

        root.getChildren().add(polygon);
        Scene scene = new Scene(root, 300, 400);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



PART II-Event Driven Programming

Events and Events Sources:

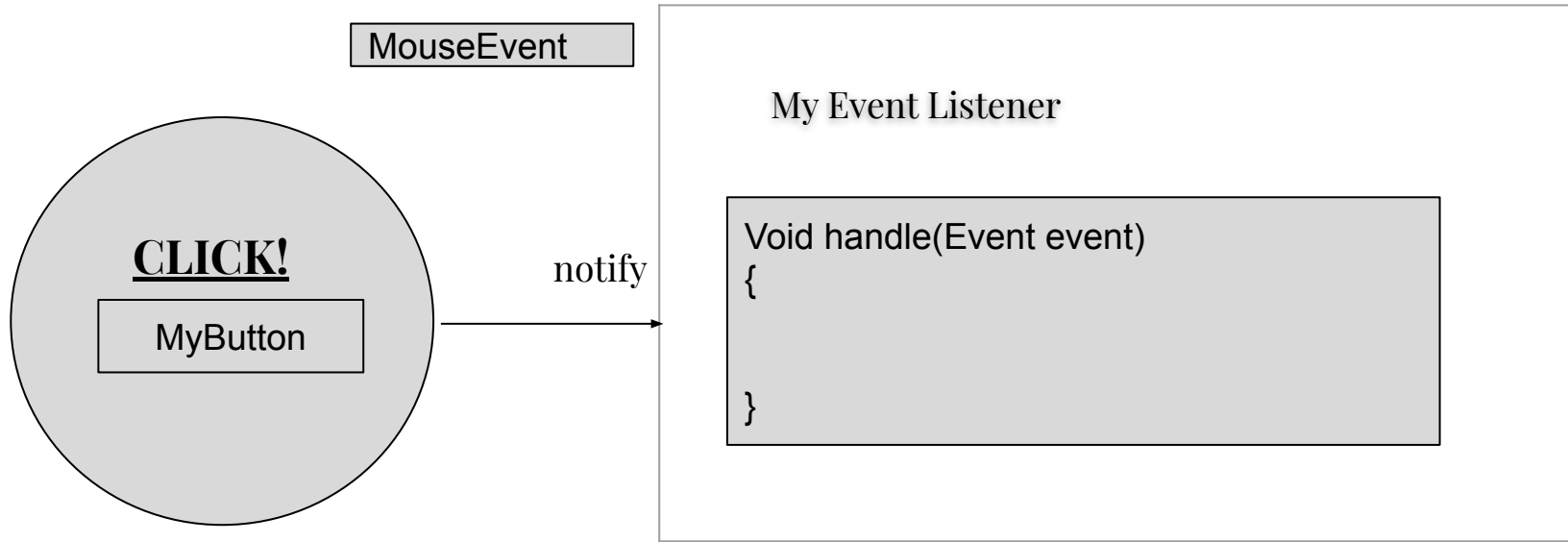
Event: Event means any activity that interrupts the current ongoing activity. For Example: When user clicks button then it generates an event. To respond to button click we need to write the code to process the button clicking action.

Event Source Object: The object generates the event is called event source object. For Example: if the event gets generated on clicking the button, then button is the event source object.

cont..

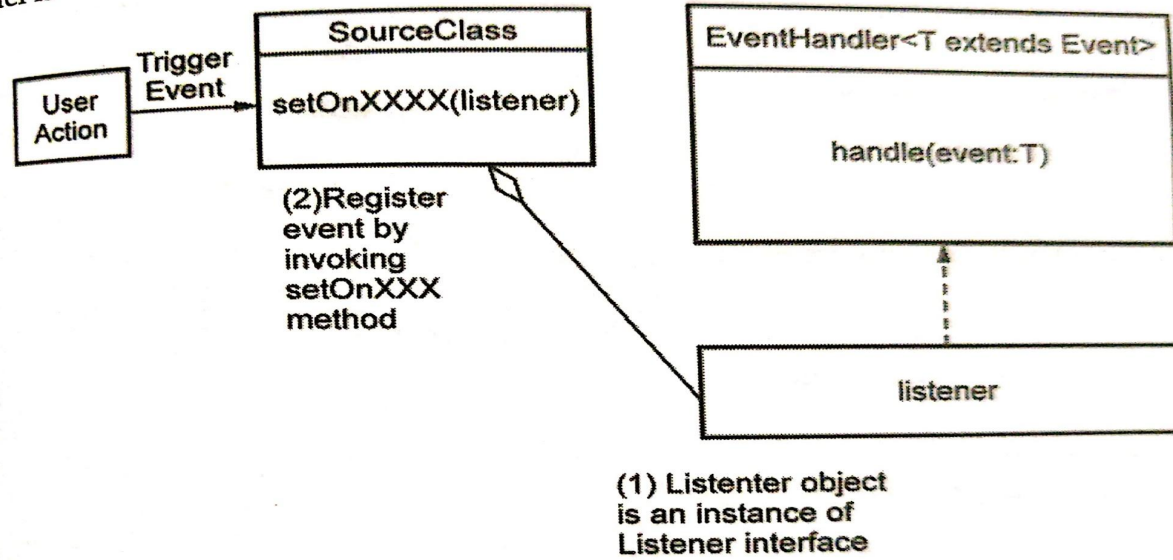
Event Handler: The event handling code written to process the generated event is called event handler.

Event Listener: The task of handling an event is carried out by event listener. When an event occurs, first of all an event object of the appropriate type is created. This object is then passed to a Listener. A listener must implement the interface that has the method for event handling.



User Action	Source Object	Event Type Fired	Event Registration Method
Click button	Button	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Enter text in textfield	TextField	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Click radio button for check or uncheck	RadioButton	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Check or uncheck checkbox	Checkbox	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Select an item	ComboBox	ActionEvent	setOnAction(EventHandler<ActionEvent>)

- Java makes use of delegation based model for event handling and registration. This model is as shown below.



The event handling process is a two step process. It performs following functionalities -

- (1) The handler object is an instance of appropriate **EventHandler** interface. The **EventHandler** interface is defined as **EventHandler<T extends Event>**. It contains **handle()** function for processing the event.

Example:

```
Button btn = new Button();  
btn.setText("Click Me");  
btn.setOnAction(new EventHandler<ActionEvent>() {  
  
    @Override  
    public void handle(ActionEvent event) {  
        System.out.println("Hello World!");  
    }  
}
```

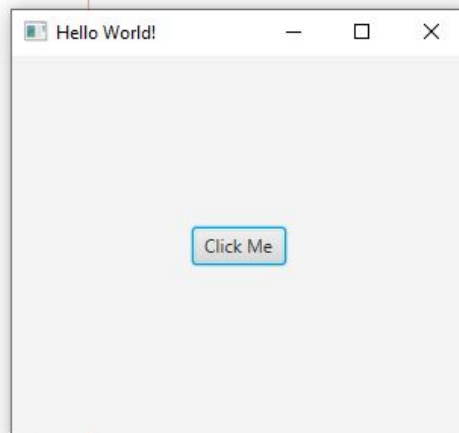
```
package javafxapplicationeventhandlingdemo;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class JavaFXApplicationEventHandlingDEMO extends Application {

    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Click Me");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                System.out.println("Button is Clicked!!!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



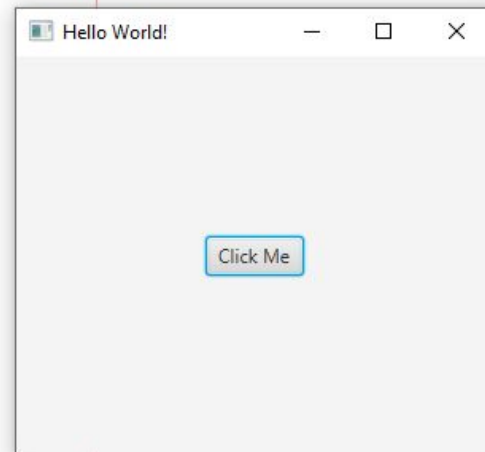

```

package javafxapplicationeventhandlingdemo;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class JavaFXApplicationEventHandlingDEMO extends Application {

    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Click Me");
        btn.setOnAction(new EventHandler<ActionEvent>() {

```



```

        @Override
        out - JavaFXApplicationEventHandlingDEMO (jfxsa-run) x
        jar:
        -check-concurrent-runs:
        -create-temp-run-dir:
            [copy] Copying 3 files to C:\Users\Lenovo\Documents\NetBeansProjects\JavaFXApplicationEventHandlingDEMO\dist\run178280864
        jfx-project-run:
            [echo] Executing C:\Users\Lenovo\Documents\NetBeansProjects\JavaFXApplicationEventHandlingDEMO\dist\run178280864\JavaFXApplicationEven
            [java] Button is Clicked!!!
    
```

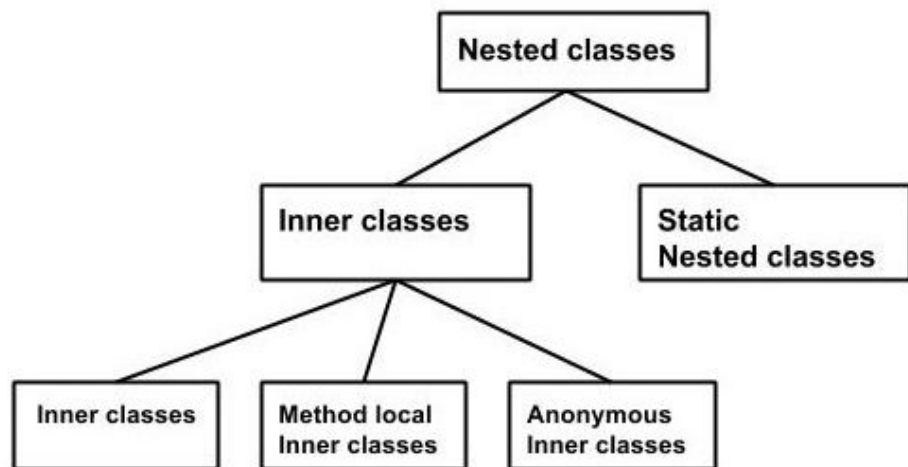
Inner Classes:

- ★ **Java inner class** or nested class is a class which is declared inside the class or interface.
- ★ We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
- ★ Additionally, it can access all the members of outer class including private data members and methods.
- ★ Syntax:

```
Acess_modifier class Java_Outer_class
{
    //code

    Acess_modifier class Java_Inner_class
    {
        //code
    }
}
```

- **Non-static nested classes** – These are the non-static members of a class.
- **Static nested classes** – These are the static members of a class.



Static Member Class

- ★ A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.
- ★ It can access static data members of outer class including private.
- ★ Static nested class cannot access non-static (instance) data member or method.

```
class TestOuter1{  
    static int data=30;  
    static class Inner{  
        void msg(){System.out.println("data is "+data);}  
    }  
    public static void main(String args[]){  
        TestOuter1.Inner obj=new TestOuter1.Inner();  
        obj.msg();  
    }  
}
```

Member Inner Classes

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer{
```

```
//code
```

```
class Inner{
```

```
//code
```

```
}
```

Local Inner Classes

- ★ A class i.e. created inside a method is called local inner class in java. If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

```
public class localInner1{  
    private int data=30;//instance variable  
    void display(){  
        class Local{  
            void msg(){System.out.println(data);}  
        }  
        Local l=new Local();  
        l.msg();  
    }  
    public static void main(String args[]){  
        localInner1 obj=new localInner1();  
        obj.display();  
    }  
}
```


Anonymous Inner Class

A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface. Java Anonymous inner class can be created by two ways:

1. Class (may be abstract or concrete).
2. Interface

```
abstract class Person{  
    abstract void eat();  
}  
  
class TestAnonymousInner{  
    public static void main(String args[]){  
        Person p=new Person(){  
            void eat(){System.out.println("nice fruits");}  
        };  
        p.eat();  
    }  
}
```

Anonymous Inner Class Handlers

- ★ Anonymous inner class is an inner class without a name. It combines two things- defining an inner class and creating an instance of the class into one step.
- ★ Syntax:

```
new superclassName_OR_InterfaceName()  
  
{  
  
    //overriding method in superclass or Interface  
  
}
```

```

package javafxapplicationanonymousinnerclass;

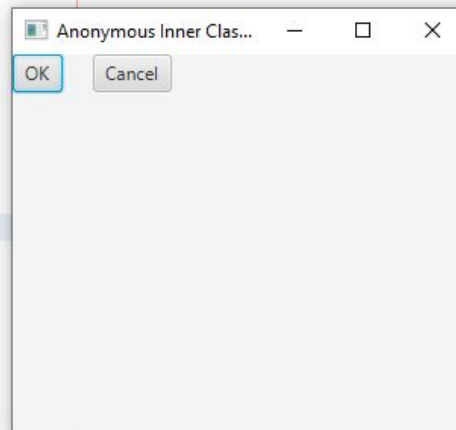
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class JavaFXApplicationAnonymousInnerClass extends Application {
    @Override
    public void start(Stage primaryStage) {

        HBox root= new HBox();
        root.setSpacing(20);
        Button ok_button = new Button("OK");
        Button cancel_button = new Button("Cancel");
        //create and registerd event
        ok_button.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Okay Button is Clicked!!!");
            }
        });

        cancel_button.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Cancel Button is Clicked!!!");
            }
        });
    }
}

```



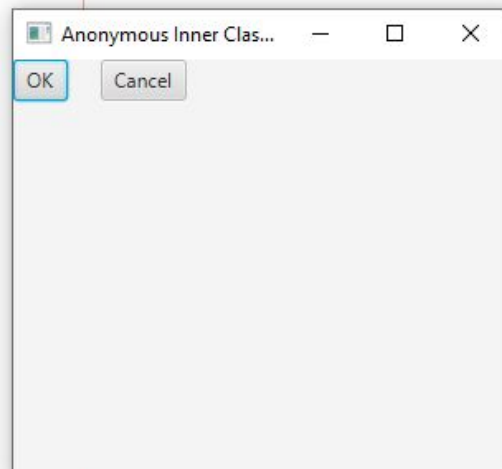
```

    public void handle(ActionEvent event) {
        System.out.println("Cancel Button is Clicked!!!");
    }
});

    root.getChildren().addAll(ok_button, cancel_button);
    Scene scene = new Scene(root, 300, 250);
    primaryStage.setTitle("Anonymous Inner Class DEMO");
    primaryStage.setScene(scene);
    primaryStage.show();
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    launch(args);
}

```



JavaFXApplicationAnonymousInnerClass (jfxsa-run) X

tr:

check-concurrent-runs:

create-temp-run-dir:

[copy] Copying 3 files to C:\Users\Lenovo\Documents\NetBeansProjects\JavaFXApplicationAnonymousInnerClass\dist\run1201214912

fx-project-run:

[echo] Executing C:\Users\Lenovo\Documents\NetBeansProjects\JavaFXApplicationAnonymousInnerClass\dist\run1201214912\JavaFXApplicatic

[java] Okay Button is Clicked!!!

```

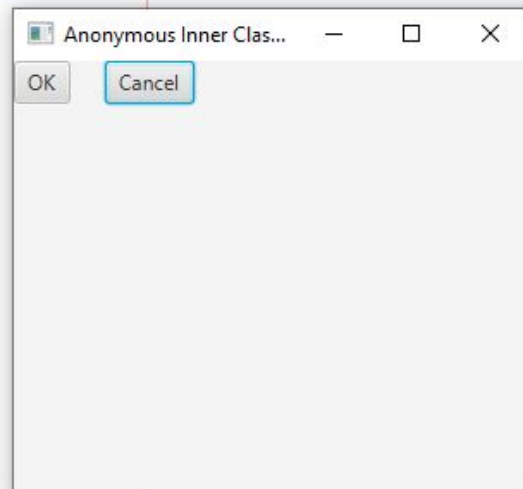
});

cancel_button.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Cancel Button is Clicked!!!");
    }
});

root.getChildren().addAll(ok_button, cancel_button);
Scene scene = new Scene(root, 300, 250);
primaryStage.setTitle("Anonymous Inner Class DEMO");
primaryStage.setScene(scene);
primaryStage.show();
}

/**
 * @param args the command line arguments
 */
}

```



- JavaFXApplicationAnonymousInnerClass (jfxsa-run) X

jar:

-check-concurrent-runs:

-create-temp-run-dir:

[copy] Copying 3 files to C:\Users\Lenovo\Documents\NetBeansProjects\JavaFXApplicationAnonymousInnerClass\dist\run688853054

jfx-project-run:

[echo] Executing C:\Users\Lenovo\Documents\NetBeansProjects\JavaFXApplicationAnonymousInnerClass\dist\run688853054\JavaFXApplicationAnon
 [java] Cancel Button is Clicked!!!

Lamda Method

Anonymous inner class event handler

```
ok_button.setOnAction{  
  
    new EventHandler <ActionEvent>(){  
  
        @override  
  
        Public void handle(ActionEvent event)  
  
        {  
  
            System.out.println("OK Button is clicked!!!");  
  
        }  
  
    });
```

Lamda expression event handler

```
ok_button.setOnAction((ActionEvent e)->  
  
    {  
  
        System.out.println("OK Button is clicked!!!");  
  
    })
```

Mouse and Key Events:

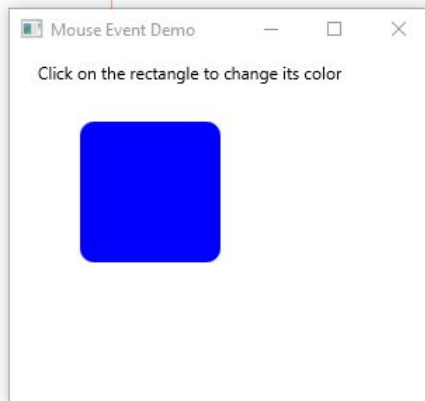
User Action	Source Object	Event Type Fired	Event Registration Method
Mouse Pressed	Node,Scene	Mouse Event	setOnMousePressed(EventHandler<MouseEvent>)
Mouse Released			setOnMouseReleased(EventHandler<MouseEvent>)
Mouse Clicked			setOnMouseClicked(EventHandler<MouseEvent>)
Mouse Moved			setOnMouseMoved(EventHandler<MouseEvent>)
Mouse Dragged			setOnMouseDragged(EventHandler<MouseEvent>)
Mouse entered			setOnMouseEntered(EventHandler<MouseEvent>)
Mouse exited			setOnMouseExited(EventHandler<MouseEvent>)

MouseEvent Class

Method	Description
MouseButton getButton	It represent which mount button is pressed
int getClickCount()	It returns the number of mouse clicks.
double getX()	It returns the x-coordinates of the mouse point in the event source node.
double getY()	It returns the y-coordinates of the mouse point in the event source node.
double getSceneX()	It returns the x-coordinates of the mouse point in the Scene
double getSceneY()	It returns the y-coordinates of the mouse point in the Scene
double getscreenX()	It returns the x-coordinates of the mouse point in the Screen

double getscreenY()	It returns the y-coordinates of the mouse point in the Screen
boolean isAltDown()	It checks if the ALT Key is pressed or not.
boolean isControlDown()	It checks if the Control Key is pressed or not.
boolean isShiftDown()	It checks if the shift key is pressed or not.

```
Source History | [Icons]
6 package javaFXApplicationMouseEventDemo;
7
8 import javafx.scene.paint.Color;
9 import javafx.application.Application;
10 import javafx.scene.input.MouseEvent;
11 import javafx.scene.shape.Rectangle;
12 import javafx.scene.text.Text;
13 import javafx.event.EventHandler;
14 import javafx.scene.Scene;
15 import javafx.scene.Group;
16 import javafx.scene.layout.StackPane;
17 import javafx.stage.Stage;
18
19 public class JavaFXApplicationMouseEventDemo extends Application {
20
21     @Override
22     public void start(Stage primaryStage) {
23
24         Rectangle rect= new Rectangle(50,50,100,100);
25         rect.setArcWidth(20);
26         rect.setArcHeight(20);
27         rect.setFill(Color.BLUE);
28         Text text= new Text(20,20,"Click on the rectangle to change its color");
29         rect.setOnMouseClicked(e->{rect.setFill(Color.RED);
30
31         });
32         Group root = new Group(rect,text);
33         Scene scene = new Scene(root, 300, 250);
34         primaryStage.setTitle("Mouse Event Demo");
35         primaryStage.setScene(scene);
36         primaryStage.show();
37     }
```



```
package javafx.application.mouseeventdemo;

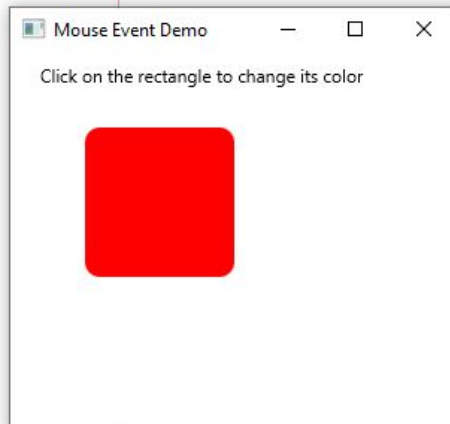
import javafx.scene.paint.Color;
import javafx.application.Application;
import javafx.scene.input.MouseEvent;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class JavaFXApplicationMouseEventDemo extends Application {

    @Override
    public void start(Stage primaryStage) {

        Rectangle rect= new Rectangle(50,50,100,100);
        rect.setArcWidth(20);
        rect.setArcHeight(20);
        rect.setFill(Color.BLUE);
        Text text= new Text(20,20,"Click on the rectangle to change its color");
        rect.setOnMouseClicked(e->{rect.setFill(Color.RED);

    });
    Group root = new Group(rect,text);
    Scene scene = new Scene(root, 300, 250);
    primaryStage.setTitle("Mouse Event Demo");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```



Keyboard Events:

When any key on the keyboard is pressed, released, or typed on a node then the keyboard event occurs.

User Action	Source Object	Event Type Fired	Event Registration Method
Key Pressed	Node, Scene	KeyEvent	setOnKeyPressed(EventHandler<KeyEvent>)
Key Released			setOnKeyReleased(EventHandler<KeyEvent>)
Key typed			setOnKeyTyped(EventHandler<KeyEvent>)

KeyEvent Class

Method	Description
String getCharacter()	It returns the character associated with the key pressed.
KeyCode getCode()	Returns the key code associated with the key in the event
String getText()	Returns the string describing the key code.
boolean isAltDown()	It returns true if the ALT Key is down
boolean isControlDown()	It returns true if the Control Key is down.
boolean isShiftDown()	It returns true if the shift key is pressed.
boolean isMetaDown()	It returns true if Meta button is pressed.

```

package javafxapplicationkeyboardevents;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class JavaFXApplicationKeyboardEvents extends Application {

    @Override
    public void start(Stage primaryStage) {
        Text msg= new Text(20,20,"Type something here...");
        Text text= new Text(50,50," ");
        text.setFont(Font.font("Arial",FontWeight.BOLD,40));
        text.setOnKeyPressed(e->{text.setText(e.getText());
        });
        Group root = new Group(msg,text);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("KeyBoard Event Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

```

Listeners for Observable Objects

- ★ A listener can be added to process a value change in an observable object.
- ★ Observable Object is basically an instance of class Observable.
- ★ Using the `addListener(InvalidationListener listener)` method we can add the Listener.
- ★ The Method `public void invalidated(Observable)` is an overriding method, that helps to note the change in value.


```
package javafx.application.observable;

import javafx.application.Application;

import javafx.beans.Observable;

import javafx.beans.property.IntegerProperty;

import javafx.beans.property.SimpleIntegerProperty;

public class JavaFXApplicationObservable extends Application {

    public static void main(String[] args) {

        IntegerProperty count= new SimpleIntegerProperty();

        count.addListener((Observable ov)->{

            System.out.println("The new value is"+count.intValue());

        });

        count.set(100);}}
```

Animation

- ★ In general, the animation can be defined as the transition which creates the myth of motion for an object. It is the set of transformations applied on an object over the specified duration sequentially so that the object can be shown as it is in motion.
- ★ This can be done by the rapid display of frames. In JavaFX, the package **javafx.animation** contains all the classes to apply the animations onto the nodes. All the classes of this package extend the class **javafx.animation.Animation**.
- ★ JavaFX provides the classes for the transitions like RotateTransition, ScaleTransition, TranslateTransition, FadeTransition, FillTransition, StrokeTransition, etc.

SN	Transition	Description
1	Rotate Transition	Rotate the Node along one of the axes over the specified duration.
2	Scale Transition	Animate the scaling of the node over the specified duration.
3	Translate Transition	Translate the node from one position to another over the specified duration.
4	Fade Transition	Animate the opacity of the node. It keeps updating the opacity of the node over a specified duration in order to reach a target opacity value
5	Fill Transition	Animate the node's fill color so that the fill color of the node fluctuates between the two color values over the specified duration.
6	Stroke Transition	Animate the node's stroke color so that the stroke color of the node fluctuates between the two color values over the specified duration.
7	Perform the list of transitions on a node in the sequential order.	
8	Parallel Transition	Perform the list of transitions on a node in parallel.
9	Path Transition	Move the node along the specified path over the specified duration.

Path Transition

- ★ It allows the node to animate through a specified path over the specified duration. In JavaFX, the path is defined by instantiating the class `javafx.scene.shape.Path`.
- ★ The translation along this path is done by updating the x and y coordinate of the node at the regular intervals. The rotation can only be done in the case when the orientation is set to be `OrientationType.ORTHOGONAL_TO_TANGENT`.
- ★ In JavaFX, the class `javafx.animation.PathTransition` represents the path transition. We need to instantiate this class in order to create an appropriate path transition.

Property	Description	Setter Methods
duration	This property is an object type of the class Duration. This represents the lifespan of the transition.	setDuration(Duration duration)
node	This is an object of the class Node. This represents the node onto which the transition will be applied.	setNode(Node node)
orientation	This is a object type property referenced by PathTransition.OrientationType . It represents the upright orientation of the node along the path.	SetOrientation(PathTransition.Orientation orientation-type)
path	This is an object type property of the class Shape. It specified the shape through which the outline of the animated path undergoes.	setPath(Shape shape)

Constructor:

There are three constructors in the class.

1. **public PathTransition()** : Creates the instance of the Path Transition with the default parameters
2. **public PathTransition(Duration duration, Shape path)** : Creates the instance of path transition with the specified duration and path
3. **public PathTransition(Duration duration, Shape path, Node node)** :
Creates the instance of the PathTransition with the specified duration, path and the node.

```

package javafxapplicationpathtransition;

import javafx.animation.PathTransition;
import javafx.animation.PathTransition.OrientationType;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class JavaFXApplicationPathTransition extends Application {

    @Override
    public void start(Stage primaryStage) {

        // TODO Auto-generated method stub
        //Creating Circle
        Circle circle = new Circle(50,50,20);

        //Setting Colour and Stroke properties for the Circle
        circle.setFill(Color.LIMEGREEN);
        circle.setStroke(Color.BLACK);
        //Instantiating PathTransition class
        PathTransition pathTransition = new PathTransition();
    }
}

```

```
//Setting duration for the PathTransition
pathTransition.setDuration(Duration.millis(1000));

//Setting Node on which the path transition will be applied
pathTransition.setNode(circle);
Rectangle rect=new Rectangle(350,100,100,100);
//setting path for the path transition
pathTransition.setPath(rect);

//setting orientation for the path transition
pathTransition.setOrientation(OrientationType.ORTHOGONAL_TO_TANGENT);

//setting up the cycle count
pathTransition.setCycleCount(20);

//setting auto reverse to be true
pathTransition.setAutoReverse(true);

//Playing path transition
pathTransition.play();

//Configuring group and scene
Group root = new Group();
root.getChildren().addAll(circle);
Scene scene = new Scene(root, 700, 400);

primaryStage.setTitle("Path Transition Example");
primaryStage.setScene(scene);
primaryStage.show();
}
```




Path Transition Example



Fade Transition

- ★ It animates the opacity of the node so that the fill color of the node becomes dull. This can be done by keep decreasing the opacity of the fill color over a specified duration in order to reach a target opacity value.
- ★ In JavaFX, the class **`javafx.animation.FadeTransition`** represents FadeTransition. We need to instantiate this class in order to create the appropriate Fade Transition.

Property	Description	Setter Methods
byValue	It is a double type property. It represents the incremented stop opacity value of the Fade transition.	setByValue(double property)
Duration	This is an object type property of the class Duration. It represent the duration of this fade transition.	setDuration(Duration duration)
fromValue	This is a double type property. It represents the start opacity for the fade transition.	setFromValue(double value)
node	This is an object type property of the class Node. This represents the node onto which, the transition is to be applied.	setNode(Node node)
toValue	This is a double type property. This represents the stop value of the opacity for the fade transition.	setToValue(double value)

Constructor:

The class contains three Constructors.

1. **public TranslateTransition()** : creates the new instance of TranslateTransition with the default parameters.
2. **public TranslateTransition(Duration duration)** : creates the new instance of TranslateTransition with the specified duration.
3. **public TranslateTransition(Duration duration, Node node)** : creates the new instance of Translate Transition with the specified duration and node.

JavaFXApplicationFadeTransition - Apache NetBeans IDE 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

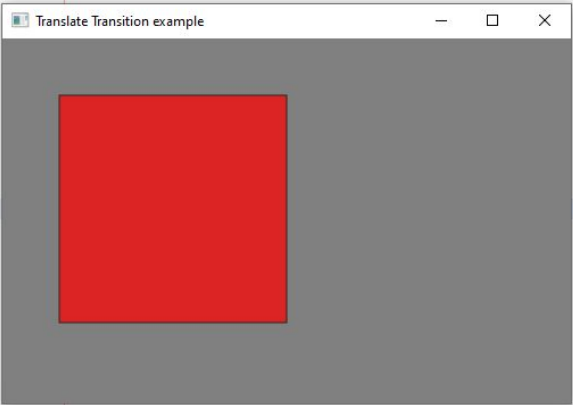
298.4/456.0-MB

JavaFXApplicationMouseEventDemo.java x JavaFXApplicationKeyboardEvents.java x JavaFXApplicationAnonymousInnerClass.java x JavaFXApplicationPathTransition.java x JavaFXApplicationFadeTransition.java x

Source History

```
1 package javafxapplicationfadetransition;
2
3
4 import javafx.animation.FadeTransition;
5 import javafx.application.Application;
6 import javafx.scene.Group;
7 import javafx.scene.Scene;
8 import javafx.scene.paint.Color;
9 import javafx.scene.shape.Circle;
10 import javafx.scene.shape.Rectangle;
11 import javafx.stage.Stage;
12 import javafx.util.Duration;
13 public class JavaFXApplicationFadeTransition extends Application {
14
15     @Override
16     public void start(Stage primaryStage) {
17         // TODO Auto-generated method stub
18         //Creating the circle
19         Rectangle rect = new Rectangle(50,50,200,200);
20
21         //setting color and stroke of the circle
22         rect.setFill(Color.RED);
23         rect.setStroke(Color.BLACK);
24
25         //Instantiating FadeTransition class
26         FadeTransition fade = new FadeTransition();
27
28
29         //setting the duration for the Fade transition
30         fade.setDuration(Duration.millis(3000));
31         //setting Circle as the node onto which the transition will be applied
32         fade.setNode(rect);
```

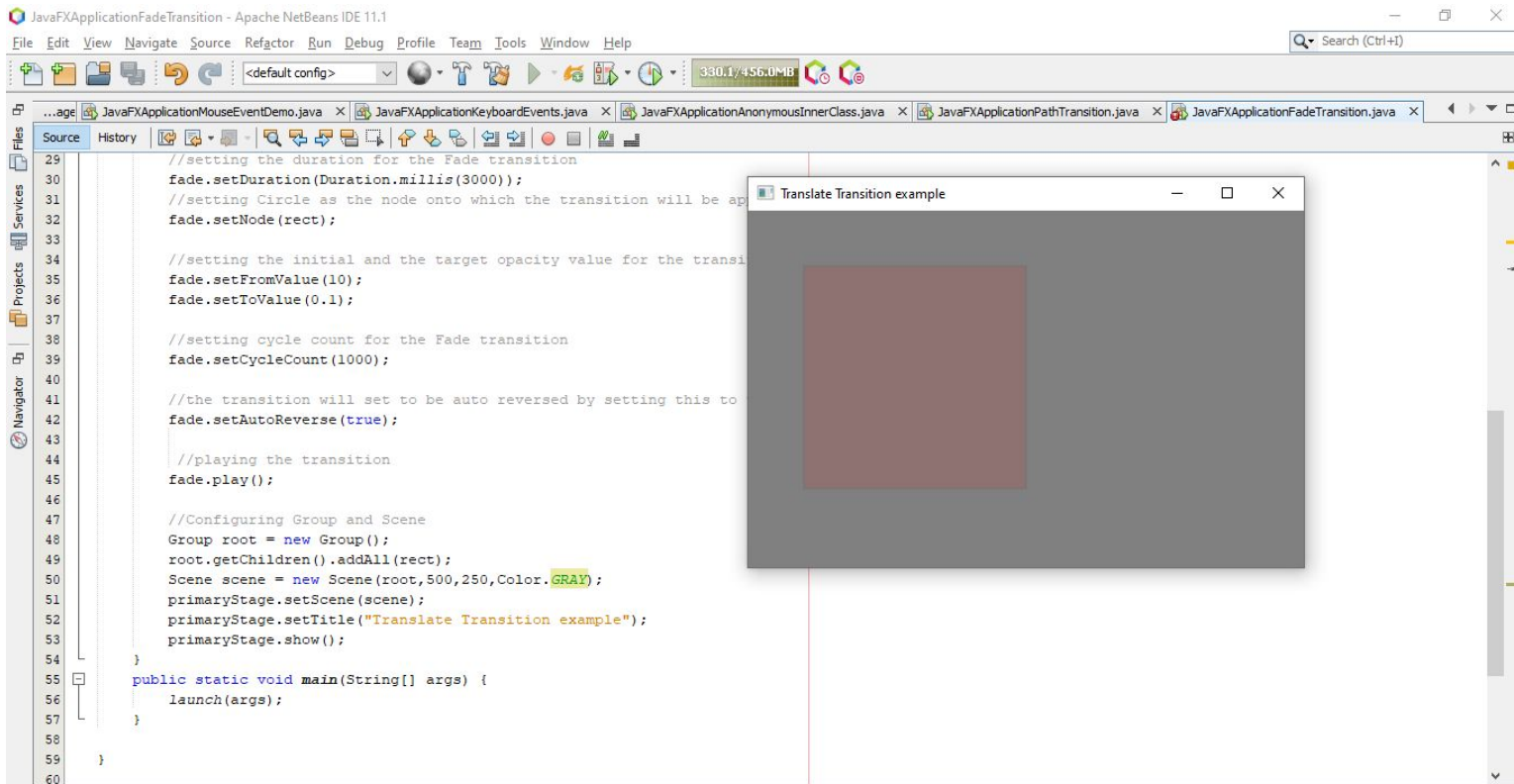
Translate Transition example



JavaFXApplicationFadeTransition (jfxsa-run) running... 12:29 INS

Type here to search

19:35 02-04-2020



Timeline:

- ★ An animation is driven by its associated properties, such as size, location, and color etc. Timeline provides the capability to update the property values along the progression of time. JavaFX supports key frame animation. In key frame animation, the animated state transitions of the graphical scene are declared by start and end snapshots (key frames) of the state of the scene at certain times. The system can automatically perform the animation. It can stop, pause, resume, reverse, or repeat movement when requested.

```

import javaix.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

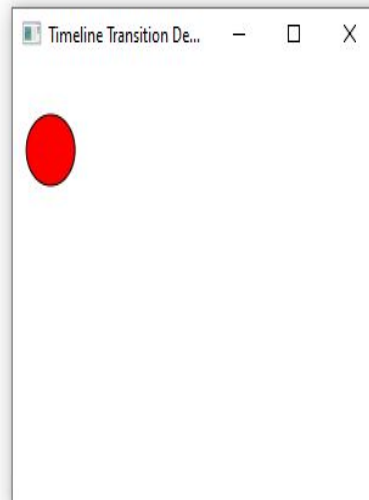
public class JavaFXApplicationTimeline extends Application {

    @Override
    public void start(Stage primaryStage) {
        Circle circle=new Circle(20,50,20);
        circle.setFill(Color.RED);
        circle.setStroke(Color.BLACK);
        Timeline tm = new Timeline(); //Instantiating timeline class

        KeyValue kv = new KeyValue(circle.translateXProperty(),200);
        KeyFrame kf= new KeyFrame(Duration.millis(2000),kv);
        tm.getKeyFrames().addAll(kf); //
        tm.setCycleCount(50);//setting up the cycle count
        tm.setAutoReverse(true);
        tm.play();
        Group root=new Group();
        root.getChildren().addAll(circle);
        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Timeline Transition Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}

```




```
import javafx.animation.KeyValue;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class JavaFXApplicationTimeline extends Application {

    @Override
    public void start(Stage primaryStage) {
        Circle circle=new Circle(20,50,20);
        circle.setFill(Color.RED);
        circle.setStroke(Color.BLACK);
        Timeline tm = new Timeline(); //Instantiating timeline class

        KeyValue kv = new KeyValue(circle.translateXProperty(),200);
        KeyFrame kf= new KeyFrame(Duration.millis(2000),kv);
        tm.getKeyFrames().addAll(kf); //
        tm.setCycleCount(50);//setting up the cycle count
        tm.setAutoReverse(true);
        tm.play();
        Group root=new Group();
        root.getChildren().addAll(circle);
        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Timeline Transition Demo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

