UNIT-7 JAVAFX basics and Event-driven programming and animations

(Chapter-14) Basic structure of JAVAFX program, Panes, UI control and shapes, Property binding, the Color and the Font class, the Image and Image-View class, layout panes and shapes, **(Chapter-15)**Events and Events sources, Registering Handlers and Handling Events, Inner classes, anonymous inner class handlers, mouse and key events, listeners for observable objects, animation

7.1 Basic structure of JAVAFX program

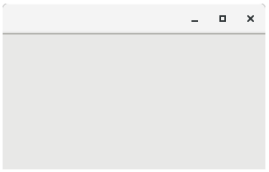
- JavaFX application is divided hierarchically into three main components known as Stage, Scene and nodes.
- We need to import **javafx.application.Application** class in every JavaFX application.
- This provides the following life cycle methods for JavaFX application.
  - public void **init**()
  - 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** 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.



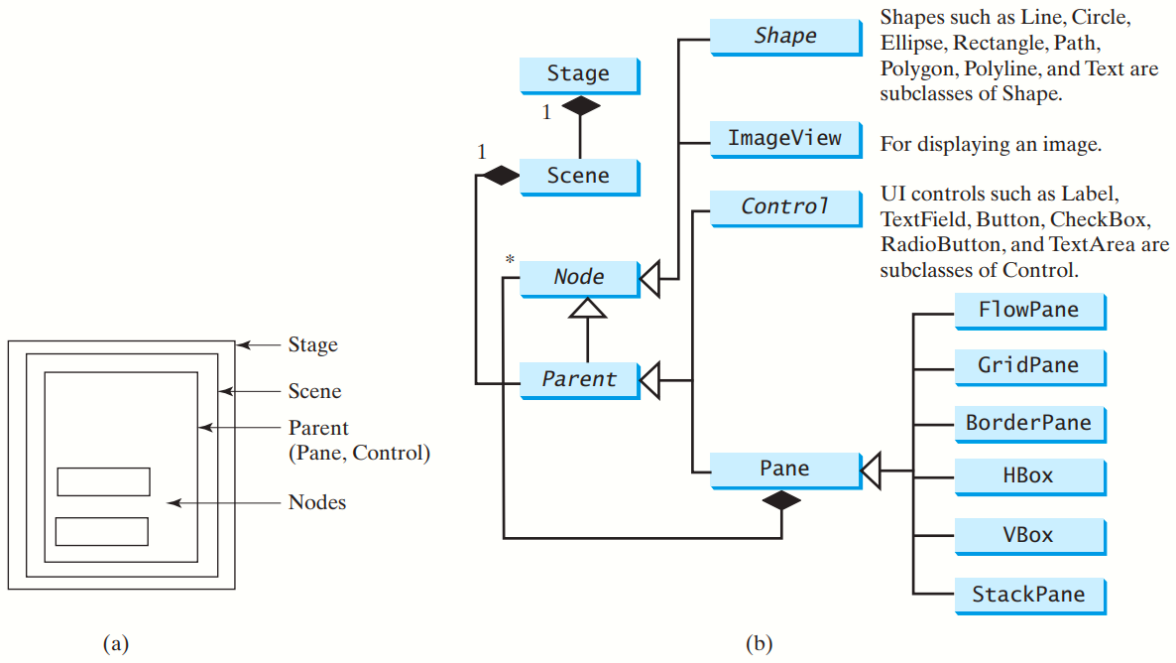
The objects can only be added in a hierarchical way. First, scene graph will be added to this primaryStage and then that scene graph may contain the nodes.

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.

Scene Graph (Contains A parent node and other nodes)

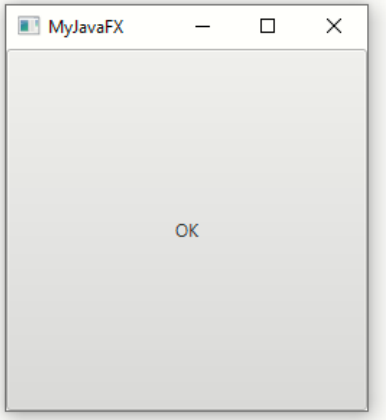
- Scene Graph exists at the lowest level of the hierarchy.
- It can be seen as the collection of various nodes.
- A node is the element which is visualized on the stage. It can be any button, text box, layout, image, radio button, check box, etc.
- The nodes are implemented in a tree kind of structure.
- 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.
- Each of the node present in the scene graphs represents classes of **javafx.scene** package.



The abstract javafx.application.Application class defines the essential framework for writing JavaFX programs.

Example: JavaFX program that illustrates the basic structure of a JavaFX.

```
1  package application;
2
3  import javafx.application.Application;
4   import javafx.scene.Scene;
5   import javafx.scene.control.Button;
6   import javafx.stage.Stage;
7
8   public class MyJavaFx1 extends Application {
9   @Override // Override the start method in the Application class
10  public void start(Stage primaryStage) {
11  // Create a scene and place a button in the scene
12  Button btOK = new Button("OK");
13  Scene scene = new Scene(btOK, 250, 250);
14  primaryStage.setTitle("MyJavaFX"); // Set the stage title
15  primaryStage.setScene(scene); // Place the scene in the stage
16  primaryStage.show(); // Display the stage
17  }
18
19  /**
20  * The main method is only needed for the IDE with limited
21  * JavaFX support. Not needed for running from the command line.
22  */
23  public static void main(String[] args) {
24  Application.launch(args);
25  }
26  }
```



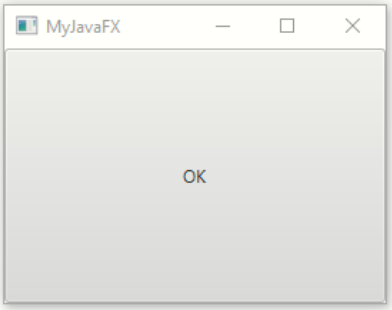
- The **launch** method (line 24) is a static method defined in the Application class for launching a stand-alone JavaFX application. When you run a JavaFX application without a main method, JVM automatically invokes the launch method to run the application.
- The main class overrides the **start** method defined in javafx.application.Application (line 10). JVM constructs an instance of the class using its no-arg constructor and invokes its start method. The start method normally places UI controls in a scene and displays the scene in a stage

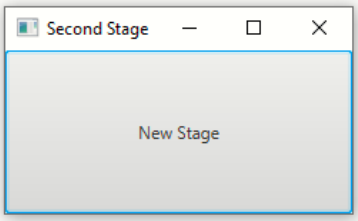
- A Scene object can be created using the constructor Scene(node, width, height). This constructor specifies the width and height of the scene and places the node in the scene.

- A Stage object is a window. A Stage object called primary stage is automatically created by the JVM when the application is launched. Line 13 sets the scene to the primary stage and line 14 displays the primary stage

Example: JavaFX program to show multiple stages in JavaFX.

```
1  package application;
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Button;
5  import javafx.stage.Stage;
6
7  public class MultipleStageDemo2 extends Application {
8  @Override // Override the start method in the Application class
9  public void start(Stage primaryStage) {
10 // Create a scene and place a button in the scene
11 Scene scene = new Scene(new Button("OK"), 200, 250);
12 primaryStage.setTitle("MyJavaFX"); // Set the stage title
13 primaryStage.setScene(scene); // Place the scene in the stage
14 primaryStage.show(); // Display the stage
15
16 Stage stage = new Stage(); // Create a new stage
17 stage.setTitle("Second Stage"); // Set the stage title
18 // Set a scene with a button in the stage
19 stage.setScene(new Scene(new Button("New Stage"), 100, 100));
20 stage.show(); // Display the stage
21 }
22 public static void main(String[] args) {
23         launch(args);
24     }
25 }
```





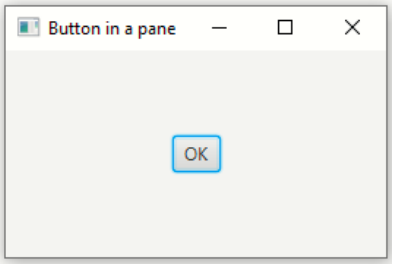
By default, the user can resize the stage. To prevent the user from resizing the stage, invoke **stage.setResizable(false)**.

7.2 Panes, UI control and shapes

- A Scene can contain a Control or a Pane, but not a Shape or an ImageView.
- A Pane can contain any subtype of Node (Control,Pane, Shape,ImageView)
- You can create a Scene using the constructor **Scene(Parent, width, height)** or **Scene(Parent).**
- Every subclass of Node has a no-arg constructor for creating a default node

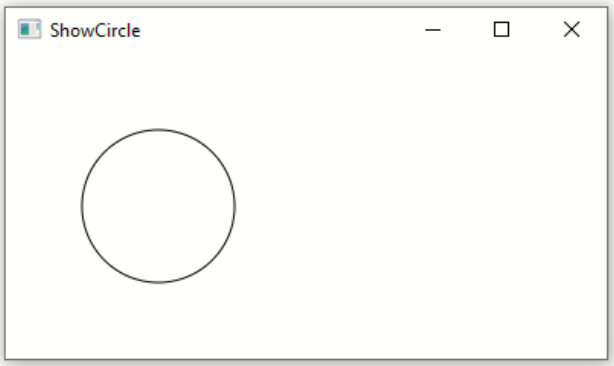
Example: Write a simple JavaFX program to show a control **Button** in pane **StackPane.**

```
1  package application;
2
3  import javafx.application.Application;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Button;
6  import javafx.stage.Stage;
7  import javafx.scene.layout.StackPane;
8
9  public class ButtonInPane3 extends Application {
10 @Override // Override the start method in the Application class
11 public void start(Stage primaryStage) {
12 // Create a scene and place a button in the scene
13 StackPane pane = new StackPane();
14 pane.getChildren().add(new Button("OK"));
15 Scene scene = new Scene(pane, 200, 50);
16 primaryStage.setTitle("Button in a pane"); // Set the stage title
17 primaryStage.setScene(scene); // Place the scene in the stage
18 primaryStage.show(); // Display the stage
19 }
20 public static void main(String[] args) {
21     System.out.println("launch application");
22     Application.launch(args);
23     }
24 }
```



Example: JavaFX program to draw a shape **Circle.**

```
1  package application;
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.layout.Pane;
5  import javafx.scene.paint.Color;
6  import javafx.scene.shape.Circle;
7  import javafx.stage.Stage;
8  public class ShowCircle extends Application {
9  @Override // Override the start method in the Application class
10 public void start(Stage primaryStage) {
11 // Create a circle and set its properties
12 Circle circle = new Circle();
13 circle.setCenterX(100);
14 circle.setCenterY(100);
15 circle.setRadius(50);
16 circle.setStroke(Color.BLACK);
17 circle.setFill(Color.WHITE);
18 // Create a pane to hold the circle
19 Pane pane = new Pane();
20  pane.getChildren().add(circle);
21   // Create a scene and place it in the stage
22  Scene scene = new Scene(pane, 200, 200);
23  primaryStage.setTitle("ShowCircle"); // Set the stage title
24  primaryStage.setScene(scene); // Place the scene in the stage
25  primaryStage.show(); // Display the stage
26  }
27 public static void main(String[] args) {
28     System.out.println("launch application");
29     Application.launch(args);
30     }
```



Note that the coordinates of the upper left corner of the pane is (0, 0) in the Java coordinate system, as shown in Figure, as opposed to the conventional coordinate system where (0, 0) is at the center of the window.

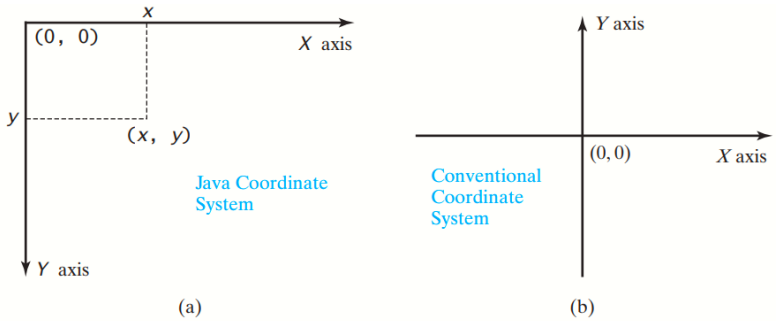


FIGURE 14.6 The Java coordinate system is measured in pixels, with (0, 0) at its upper-left corner.

In order to display the circle centered as the window resizes, the x- and y-coordinates of the circle center need to be reset to the center of the pane. This can be done by using **property binding**, introduced in the next section.

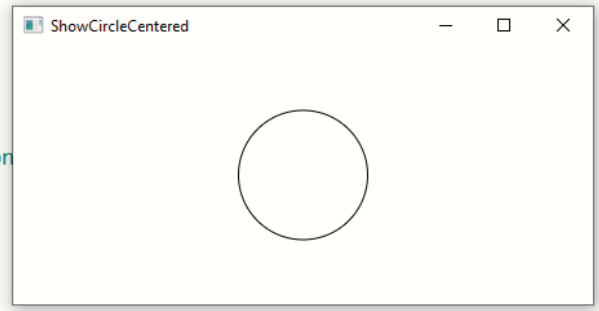
7.3 Property binding

- A target object can be bounded to a source object.
- A change in the source object will be automatically reflected in the target object.
- The **target** object is called a **binding** object or a binding property and the **source** object is called a bindable object or **observable** object.

In order to display the circle centered as the window resizes, the x- and y-coordinates of the circle center need to be reset to the center of the pane. This can be done by binding the centerX with pane's width/2 and centerY with pane's height/2.

Example: Write a simple JavaFX program to draw a shape **Circle**(at centre) using **property binding.**

```
1  package Chapter13;
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.layout.Pane;
5  import javafx.scene.paint.Color;
6  import javafx.scene.shape.Circle;
7  import javafx.stage.Stage;
8  public class ShowCircleCentered extends Application {
9  @Override // Override the start method in the Application
10 public void start(Stage primaryStage) {
11 // Create a pane to hold the circle
12 Pane pane = new Pane();
13 // Create a circle and set its properties
14 Circle circle = new Circle();
15 circle.centerXProperty().bind(pane.widthProperty().divide(2));
16 circle.centerYProperty().bind(pane.heightProperty().divide(2));
17 circle.setRadius(50);
18 circle.setStroke(Color.BLACK);
19 circle.setFill(Color.WHITE);
20 pane.getChildren().add(circle); // Add circle to the pane
21 // Create a scene and place it in the stage
22 Scene scene = new Scene(pane, 200, 200);
23 primaryStage.setTitle("ShowCircleCentered"); // Set the stage title
24 primaryStage.setScene(scene); // Place the scene in the stage
25 primaryStage.show(); // Display the stage
26 }
27 public static void main(String[] args) {
28     System.out.println("launch application");
29     Application.launch(args);
30     }
31 }
```



A target binds with a source using the bind method as follows: **target.bind(source);**

The **bind** method is defined in the **javafx.beans.property.Property** interface.
A source object is an instance of the **javafx.beans.value.ObservableValue** interface.

The naming convention for this method is the property name followed by the word **Property**. For example, the property getter method for **centerX** is **centerXProperty().** We call the **getCenterX()** method as the value getter method, the **setCenterX(double)** method as the value setter method, and**centerXProperty()** as the property getter method. Note that **getCenterX()** returns a double value and **centerXProperty()**returns an object of the **DoubleProperty** type.

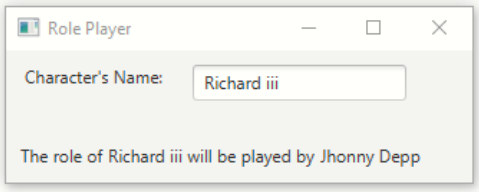
Two types of binding are supported:

- **Unidirectional binding:** With unidirectional binding, the binding works in just one direction.
  Bidirectional binding: With bidirectional binding, the two property values are synchronized.

Every property has a bind and a bindBiDirectional method. To set up a binding, simply call this method, specifying the property you want to bind to as the argument.

Practice Problem: JavaFX program to **bind textfield** value with a **label** to be displayed.

```
8  public class RolePlayer extends Application
9  {
10     TextField txtCharacter;
11     @Override public void start(Stage primaryStage)
12     {
13         // Create the Character label
14         Label lblCharacter = new Label("Character's Name:");
15         lblCharacter.setMinWidth(100);
16         lblCharacter.setAlignment(Pos.BOTTOM_RIGHT);
17         // Create the Character text field
18         txtCharacter = new TextField();
19         txtCharacter.setPromptText("Enter the name of the character here.");
20         // Create the Role labels
21         Label lblRole1 = new Label("The role of ");
22         Label lblRole2 = new Label();
23         Label lblRole3 = new Label(" will be played by ");
24         Label lblRole4 = new Label("Jhonny Depp");
25         // Create the Character pane
26         HBox paneCharacter = new HBox(20, lblCharacter, txtCharacter);
27         paneCharacter.setPadding(new Insets(10));
28         // Create the Role pane
29         HBox paneRole = new HBox(lblRole1, lblRole2, lblRole3, lblRole4);
30         paneRole.setPadding(new Insets(10));
31         // Add the Character and Actor panes to a VBox
32         VBox pane = new VBox(10, paneCharacter, paneRole);
33         // Create the bindings
34         lblRole2.textProperty().bind(txtCharacter.textProperty());
35         // Set the stage
36         Scene scene = new Scene(pane);
37         primaryStage.setScene(scene);
38         primaryStage.setTitle("Role Player");
39         primaryStage.show();    }
40 }
```



7.4 The Color and Font class

❖ The Color 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.
- The javafx.scene.paint.Color is a concrete subclass of Paint
- Lets discuss the several methods of creating color in JavaFX.

```
Color c = Color.BLUE;     //use the blue constant
// standard constructor, use 0->1.0 values, explicit alpha of 1.0
Color c = new Color(0,0,1,1.0);

Color c = Color.color(0,0,1.0); //use 0->1.0 values. implicit alpha of 1.0
Color c = Color.color(0,0,1.0,1.0); //use 0->1.0 values, explicit alpha of 1.0

Color c = Color.rgb(0,0,255); //use 0->255 integers, implicit alpha of 1.0
Color c = Color.rgb(0,0,255,1.0); //use 0->255 integers, explicit alpha of 1.0

//hue = 270, saturation & value = 1.0. inplicit alpha of 1.0
Color c = Color.hsb(270,1.0,1.0);
//hue = 270, saturation & value = 1.0, explicit alpha of 1.0
Color c = Color.hsb(270,1.0,1.0,1.0);
```

❖ The Font class

A font describes font name, weight and size.

Constructors of the class:

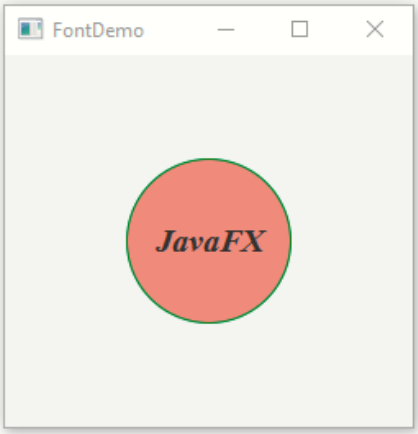
1. **Font(double s)** : Creates a font object with specified size.
2. **Font(String n, double s) :** Creates a font object with specified name and size.

Commonly Used Methods:

Method	Explanation
font(double s)	Creates a font object with specified size.
font(String f)	Creates a font object with specified family name.
font(String f, FontWeight w, FontPosture p, double s)	Creates a font object with specified family name, fontweight, font posture and size.
getDefault()	Returns the default font.
getFamilies()	Gets all the font families installed on the user's system.
getFamily()	Returns the family of the font.
getFontNames()	Gets the names of all fonts that are installed on the users system.
getFontNames(String f)	Gets the names of all fonts in the specified font family that are installed on the users system.
getName()	The full font name.
getSize()	The point size for this font.

Example: JavaFX program to use ***Color*** class to fill a circle and ***Font*** class to display text in it using ***StackPane***.

```
1  package Chapter13;
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.layout.*;
5  import javafx.scene.paint.Color;
6  import javafx.scene.shape.Circle;
7  import javafx.scene.text.*;
8  import javafx.scene.control.*;
9  import javafx.stage.Stage;
10 // The Font class and Color class demo
11 public class ColorDemo extends Application {
12     @Override // Override the start method in the Application class
13 public void start(Stage primaryStage) {
14 // Create a pane to hold the circle
15 Pane pane = new StackPane();
16
17 // Create a circle and set its properties
18 Circle circle = new Circle();
19       circle.setRadius(50);
20       circle.setStroke(Color.GREEN);
21       circle.setFill(new Color(0.74,0.3,0.3,0.63));
22 pane.getChildren().add(circle); // Add circle to the pane
23
24 // Create a label and set its properties
25 Label label = new Label("JavaFX");
26       label.setFont(Font.font("Times New Roman",FontWeight.BOLD, FontPosture.ITALIC, 20));
27 pane.getChildren().add(label);
28
29 // Create a scene and place it in the stage
30       Scene scene = new Scene(pane);
31       primaryStage.setTitle("FontDemo"); // Set the stage title
32       primaryStage.setScene(scene); // Place the scene in the stage
33       primaryStage.show(); // Display the stage
34     }
35 public static void main(String[] args) {
36     launch(args);
37   }
38   }
```



7.5 The Image and Image-View class

- The **Image** class represents a graphical image
- The **ImageView** class can be used to display an image.
- The **javafx.scene.image.Image** class represents a graphical image and is used for loading an image from a specified filename or a URL.
- The **javafx.scene.image.ImageView** is a node for displaying an image. An ImageView can be created from an Image object.

For example, the following code creates an ImageView from an image file

```
Image image = new Image("image/us.gif");
ImageView imageView = new ImageView(image);
```

OR

```
ImageView imageView = new ImageView("image/us.gif");
```

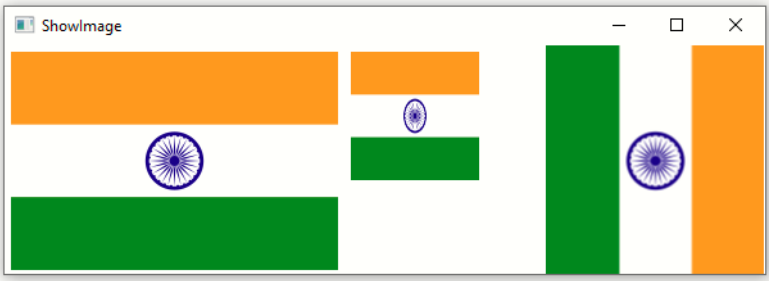
OR

```
// load the image
```

```
Image image = new Image("flower.png");
// simple displays ImageView the image as is
ImageView iv1 = new ImageView();
iv1.setImage(image);
```

Example: Write a JavaFX program to display an **image** three image views placed in an ***HBox*** pane.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
public class ShowImage extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
    // Create a pane to hold the image views
    Pane pane = new HBox(10);
    pane.setPadding(new Insets(5, 5, 5, 5));
    Image image = new Image("image/india.png");
    pane.getChildren().add(new ImageView(image));

    ImageView imageView2 = new ImageView(image);
    imageView2.setFitHeight(100);
    imageView2.setFitWidth(100);
    pane.getChildren().add(imageView2);

     ImageView imageView3 = new ImageView(image);
     imageView3.setRotate(90);
     pane.getChildren().add(imageView3);

     // Create a scene and place it in the stage
     Scene scene = new Scene(pane);
     primaryStage.setTitle("ShowImage"); // Set the stage title
     primaryStage.setScene(scene); // Place the scene in the stage
     primaryStage.show(); // Display the stage
    }
    public static void main(String[] args) {
        launch(args);
      }
```



7.6 Layout panes

JavaFX provides many types of panes for automatically laying out nodes in a desired location and size.

Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

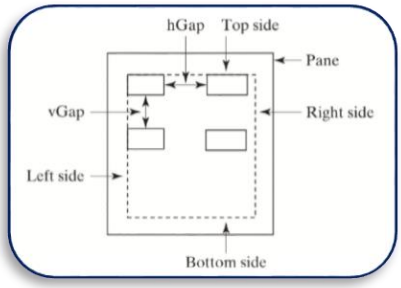
❖ **StackPane: Refer example of Font and Color class**

❖ FlowPane

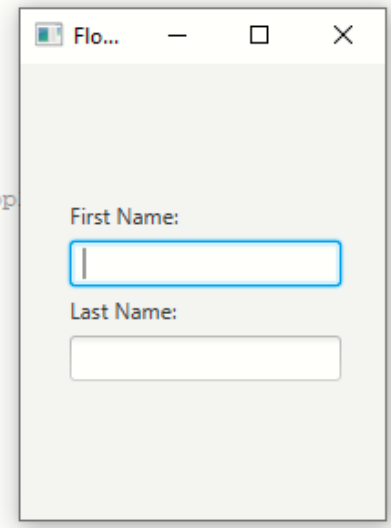
FlowPane arranges the nodes in the pane horizontally from left to right or vertically from top to bottom in the order in which they were added. When one row or one column is filled, a new row or column is started.

Example: JavaFX program to use **FlowPane** to display nodes.

- The program creates a FlowPane and sets its padding property with an Insets object.
- An Insets object specifies the size of the border of a pane.
- The constructor Insets**(11, 12, 13, 14)** creates an Insets with the border sizes for top (11), right (12), bottom (13), and left (14) in pixels, as shown in Figure.
- You can also use the constructor Insets**(value)** to create an Insets with the same value for all four sides.
- The **hGap** and **vGap** properties specify the horizontal gap and vertical gap between two nodes in the pane, as shown in Figure.




```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class ShowFlowPane extends Application {
    @Override // Override the start method in the App
        public void start(Stage primaryStage) {
        // Create a pane and set its properties
        FlowPane pane = new FlowPane();
        pane.setPadding(new Insets(11, 12, 13, 14));
        pane.setHgap(5);
        pane.setVgap(5);
        pane.setAlignment(Pos.CENTER);
        pane.setOrientation(Orientation.VERTICAL);

        // Place nodes in the pane
        pane.getChildren().addAll(new Label("First Name:"),new TextField(),
                                  new Label("Last Name:"),new TextField()
        );

        // Create a scene and place it in the stage
        Scene scene = new Scene(pane, 200, 250);
        primaryStage.setTitle("FlowPane"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }
    public static void main(String[] args) {
        launch(args);
      }
  }
```

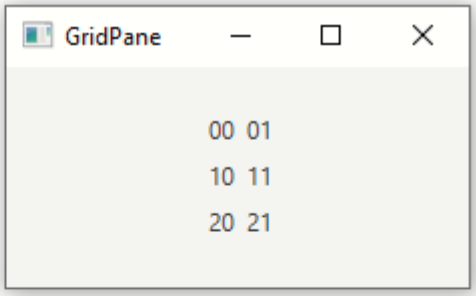


❖ **GridPane**

AGridPane arranges nodes in a grid (matrix) formation. The nodes are placed in the specified column and row indices.

Example: JavaFX program to demonstrate **GridPane**

```
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;
import javafx.application.Application;
import javafx.scene.layout.GridPane;

public class ShowGridPane2 extends Application {
        @Override
        public void start(Stage primaryStage) {
                // Create a pane and set its properties
                GridPane pane = new GridPane();
                pane.setAlignment(Pos.CENTER);
                pane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
            pane.setHgap(5.5);
            pane.setVgap(5.5);
      // Place nodes in the pane
      pane.add(new Label("00"), 0, 0);
      pane.add(new Label("01"), 1, 0);
      pane.add(new Label("10"), 0, 1);
      pane.add(new Label("11"), 1, 1);
      pane.add(new Label("20"), 0, 2);
      pane.add(new Label("21"), 1, 2);

      // Create a scene and place it in the stage
      Scene scene = new Scene(pane);
      primaryStage.setTitle("GridPane"); // Set the stage title
      primaryStage.setScene(scene); // Place the scene in the stage
      primaryStage.show(); // Display the stage
    }
    public static void main(String[] args) {
        launch(args);
      }
  }
```

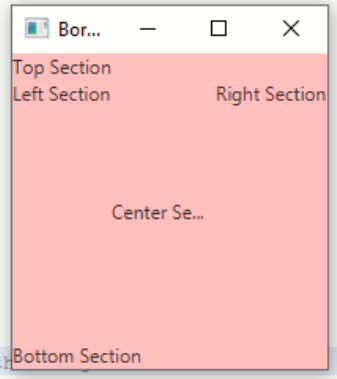


❖ **BorderPane**

A BorderPane can place nodes in five regions: top, bottom, left, right, and centre, using the
setTop(node),
setBottom(node),
setLeft(node),
setRight(node), and
setCenter(node) methods.

Example: JavaFX program to use **BorderPane** to display nodes.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
public class ShowBorderPane extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a border pane
        BorderPane pane = new BorderPane();
pane.setStyle("-fx-background-color:pink");
        // Place nodes in the pane
      pane.setTop(new Label("Top Section"));
      pane.setRight(new Label("Right Section"));
      pane.setBottom(new Label("Bottom Section"));
      pane.setLeft(new Label("Left Section"));
      pane.setCenter(new Label("Center Section"));

      // Create a scene and place it in the stage
      Scene scene = new Scene(pane,200,200);
      primaryStage.setTitle("BorderPane"); // Set th
      primaryStage.setScene(scene); // Place the scene in the stage
      primaryStage.show(); // Display the stage
    }
    public static void main(String[] args) {
        launch(args);  }
  }
```

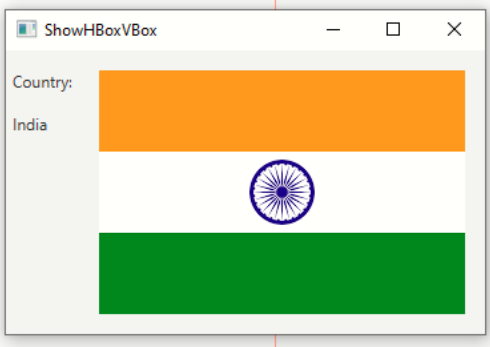


❖ HBox and VBox

An HBox lays out its children in a single horizontal row.
A VBox lays out its children in a single vertical column.
VBox has the similar data Memebers and member functions as HBox.

Example: JavaFX program to use **HBox** and **VBox.**

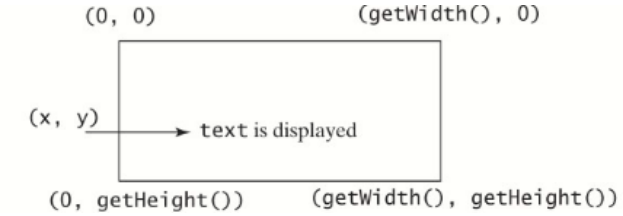
```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

public class ShowHboxVbox extends Application {
        @Override
        public void start(Stage primaryStage) {
        // Create a border pane
        HBox pane = new HBox();

        VBox vBox = new VBox(15);
        vBox.setPadding(new Insets(15, 5, 5, 5));
        vBox.getChildren().add(new Label("Country:"));
        vBox.getChildren().add(new Label("India"));

        HBox hBox = new HBox(15);
        hBox.setPadding(new Insets(15, 15, 15, 15));
        hBox.getChildren().add(vBox);
        ImageView imageView = new ImageView(new Image("flag.png"));
        hBox.getChildren().add(imageView);

        // Place nodes(VBox,HBox) in the root pane(HBox)
        pane.getChildren().addAll(vBox,hBox);
        // Create a scene and place it in the stage
       Scene scene = new Scene(pane);
       primaryStage.setTitle("ShowHBoxVBox");
       primaryStage.setScene(scene);
       primaryStage.show();
    }
        public static void main(String[] args) {
        launch(args);  }
  }
```



7.7 Shapes

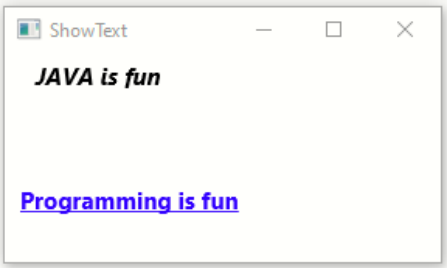
JavaFX provides many shape classes for drawing texts, lines, circles, rectangles,ellipses, arcs, polygons, and polylines.
The Shape class is the root of all shape classes (Text, Line, Rectangle, Circle, Ellipse, Arc, Polygon)

❖ Text

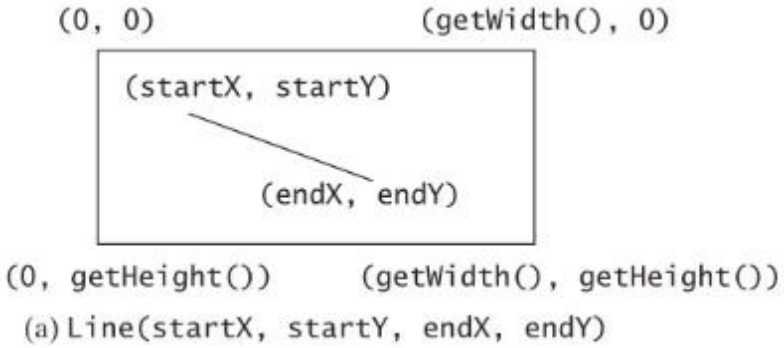


Example: JavaFX program to display **Text**.

```
import javafx.scene.paint.Color;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.FontPosture;
public class ShowText extends Application {
    @Override
    public void start(Stage primaryStage) {
    // Create a pane to hold the texts
      Pane pane = new Pane();
      pane.setPadding(new Insets(5, 5, 5, 5));
      Text text1 = new Text(20, 20, "JAVA is fun");
      text1.setFont(Font.font("Courier", FontWeight.BOLD, FontPosture.ITALIC, 15));
      pane.getChildren().add(text1);

      Text text2 = new Text(10, 100, "Programming is fun\n");
      text2.setFill(Color.BLUE);
      text2.setUnderline(true);
      text2.setFont(Font.font("Courier", FontWeight.BOLD, 15));
      pane.getChildren().add(text2);

      // Create a scene and place it in the stage
      Scene scene = new Scene(pane);
      primaryStage.setTitle("ShowText"); // Set the stage title
      primaryStage.setScene(scene); // Place the scene in the stage
      primaryStage.show(); // Display the stage
    }
```



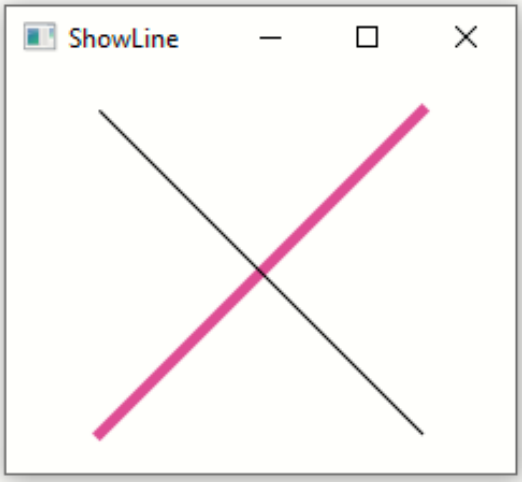
❖ **Line**



(a) Line(startX, startY, endX, endY)

Example: JavaFX program to display **Lines**.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Line;

public class ShowLine extends Application {
    @Override
public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
    Line line1 = new Line();
    line1.setStartX(10);
    line1.setStartY(10);
    line1.setEndX(160);
    line1.setEndY(160);
    line1.setStrokeWidth(5);
    line1.setStroke(Color.PURPLE);
    line1.setOpacity(0.7);
    line1.setRotate(90);

    Line line2 = new Line(10,10,160,160);
    pane.getChildren().add(line1);
    pane.getChildren().add(line2);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowLine");
    primaryStage.setScene(scene);
    primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);  }
  }
```



❖ **Rectangle**

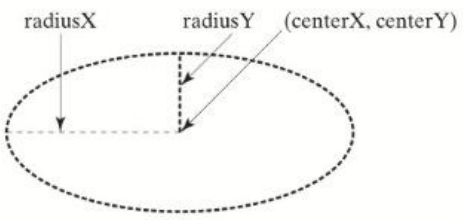
A rectangle is defined by the parameters
x, y, width, height, arcWidth, and arcHeight.

Example: Write a JavaFX program to display **Rectangles**

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.shape.Rectangle;

public class RectangleClass extends Application {
  @Override
  public void start(Stage primaryStage) {
    // Create rectangles
    Rectangle r1 = new Rectangle(25, 10, 60, 30);
    Rectangle r2 = new Rectangle(25, 50, 60, 30);
    r2.setArcWidth(15);
    r2.setArcHeight(25);

    // Create a group and add nodes to the group
    Group group = new Group();
    group.getChildren().addAll(r1, r2);

    // Create a scene and place it in the stage
    Scene scene = new Scene(group, 150, 100);
    primaryStage.setTitle("Rectangle");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
   public static void main(String[] args) {
    launch(args);
  }
}
```

❖ **Circle and Ellipse**

A circle is defined by its parameterscenterX,centerY, and radius.
An ellipse is defined by its parameters centerX,centerY,radiusX, and radius

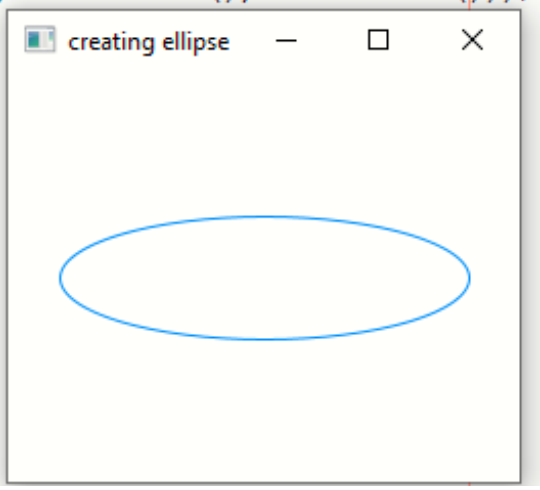


(a) Ellipse(centerX, centerY, radiusX, radiusY)

(b) Multiple ellipses are displayed.

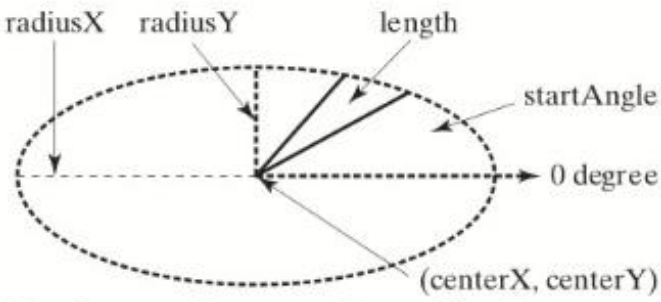
Example: JavaFX program to display an **Ellipse.**

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Ellipse;

public class ShowEllipse extends Application {
          @Override // Override the start method in the Application class
          public void start(Stage stage) {
           // set title for the stage
         stage.setTitle("creating ellipse");
         // create a ellipse
         Ellipse ellipse = new Ellipse(150.0f, 70.0f, 100.0f, 30.f);
         ellipse.setStroke(Color.color(Math.random(), Math.random(), Math.random()));
         ellipse.setFill(Color.WHITE);
         // create a Group
         StackPane pane = new StackPane(ellipse);
         // create a scene
         Scene scene = new Scene(pane,250,200);
         // set the scene
         stage.setScene(scene);
         stage.show();
         }
          public static void main(String[] args) {
            launch(args);
          }
}
```

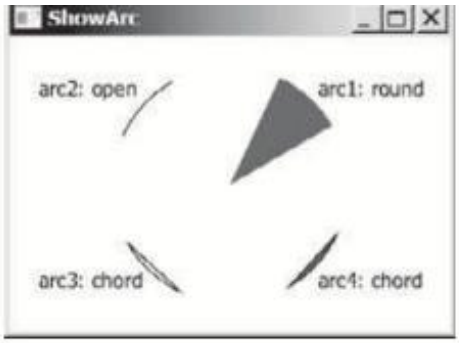


❖ Arc

An arc is conceived as part of an ellipse, defined by the parameters centerX,centerY, radiusX, radiusY,startAngle,length, and an arc type (ArcType.OPEN,ArcType.CHORD, or ArcType.ROUND).



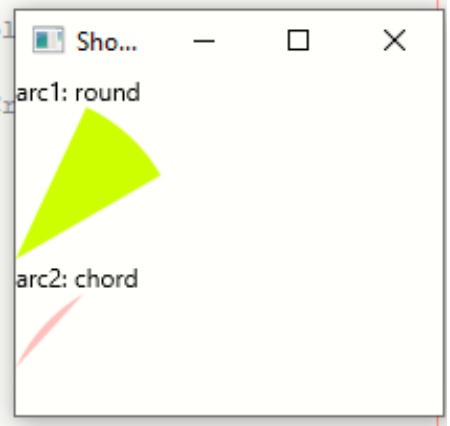
(a) Arc(centerX, centerY, radiusX, radiusY, startAngle, length)



(b) Multiple ellipses are displayed

Example: JavaFX program to display **Arcs.**

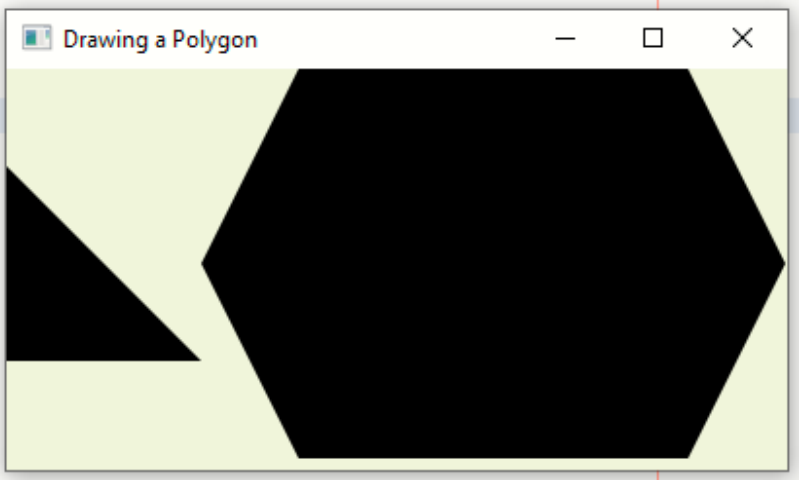
```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcType;
import javafx.scene.text.Text;

public class ShowArc2 extends Application{
    @Override // Override the start method in the Appl
  public void start(javafx.stage.Stage primaryStage) {
    Arc arc1 = new Arc(150, 100, 80, 80, 30, 35); // Cr
    arc1.setFill(Color.YELLOW); // Set fill color
    arc1.setType(ArcType.ROUND); // Set arc type

    Arc arc2 = new Arc(150, 100, 80, 80, 30 + 90, 35);
    arc2.setType(ArcType.CHORD);
    arc2.setFill(Color.PINK);

    // Create a group and add nodes to the group
    VBox group = new VBox();
    group.getChildren().addAll(new Text("arc1: round"), arc1,
                               new Text("arc2: chord"), arc2);

    // Create a scene and place it in the stage
    Scene scene = new Scene(new BorderPane(group), 300, 200);
    primaryStage.setTitle("ShowArc"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```



❖ Polygons

Example: JavaFX program to display **Polygons – Triangle & Hexagon**

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

public class ShowPolygon extends Application {
    @Override
    public void start(Stage stage) {
      Polygon hexagon = new Polygon();
      Polygon triangle = new Polygon();
      //Adding coordinates to the hexagon
        hexagon.getPoints().addAll(new Double[]{
        200.0, 50.0,
        400.0, 50.0,
        450.0, 150.0,
        400.0, 250.0,
        200.0, 250.0,
        150.0, 150.0,
        });
        triangle.getPoints().setAll(
            200.0, 200.0,
            300.0, 300.0,
            200.0, 300.0 );
    GridPane root = new GridPane();
     root.add(triangle,0,0);
     root.add(hexagon,1,0);
     root.setStyle("-fx-background-color:beige");

      Scene scene = new Scene(root);
      stage.setTitle("Drawing a Polygon");
      stage.setScene(scene);
      stage.show();
   }
   public static void main(String[] args) {
    launch(args);
  }
  }
```



7.8 Events and Events sources

You can write code to process events such as a button click, mouse movement, and keystrokes.

Types of Events

In general, the events are mainly classified into the following two types.

1. Foreground Events

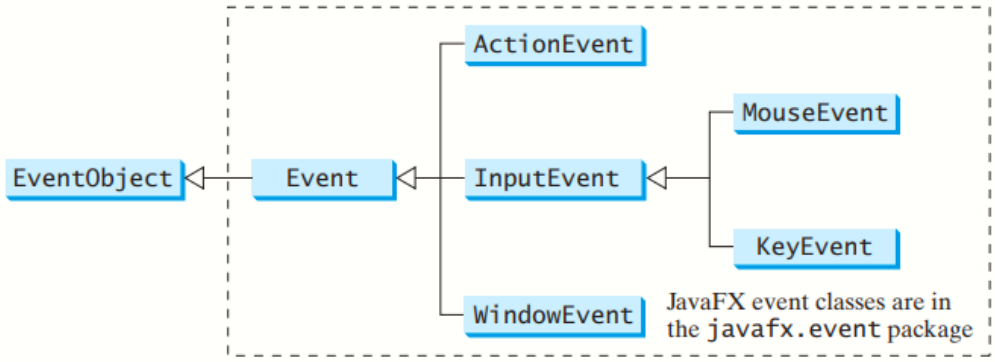
Foreground events are mainly occurred due to the direct interaction of the user with the GUI of the application. Such as clicking the button, pressing a key, selecting an item from the list, scrolling the page, etc.

2. Background Events

Background events doesn't require the user's interaction with the application. These events are mainly occurred to the operating system interrupts, failure, operation completion, etc.

Processing Events in JavaFX

- In JavaFX, events are basically used to notify the application about the actions taken by the user.
- JavaFX provides the class **javafx.event.Event** which contains all the subclasses representing the types of Events that can be generated in JavaFX.
- Any event is an instance of the class **Event** or any of its subclasses.
- There are various events in JavaFX i.e. MouseEvent, KeyEvent, ScrollEvent, DragEvent, etc. We can also define our own event by inheriting the class **javafx.event.Event**.



- The properties of an event is described in the following table.
- The instance of EventType class is further classified into various type of events for example KeyEvent class contains KEY_PRESSED, KEY_RELEASED, and KEY_TYPED types.

Registering Handlers and Handling Events

- JavaFX facilitates us to use the Event Handlers to handle the events generated by Keyboard Actions, Mouse Actions, and many more source nodes.
- To register a handler, use the addEventHandler() method. This method takes the event type and the handler as arguments.
- **Example**

```
// Define an event handler for handling events
EventHandler handler = new EventHandler(<InputEvent>() {
    public void handle(InputEvent event) {
        //code
  }
/ Register the same handler for two different nodes
```

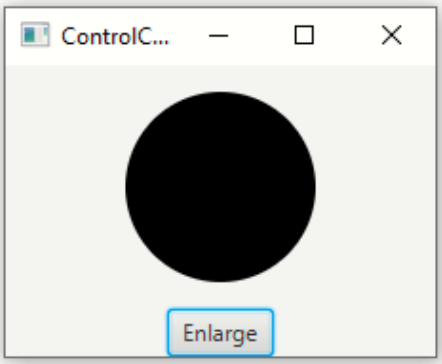
```
myNode1.addEventHandler(MouseEvent.MOUSE_DRAGGED, handler);
myNode2.addEventHandler(MouseEvent.MOUSE_DRAGGED, handler);
```

Inner Classes

- An inner class, or nested class, is a class defined within the scope of another class.
- Inner classes are useful for defining handler classes.
- Inner classes combine dependent classes into the primary class.
- Inner classes can be private and mostly are used to access private data member of an outer class.

Example

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.scene.layout.BorderPane;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class ControlCircle extends Application {
    private final Circle circle = new Circle(50);
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Hold two buttons in an HBox
    HBox hBox = new HBox();
    hBox.setSpacing(10);
    hBox.setAlignment(Pos.CENTER);
    Button btEnlarge = new Button("Enlarge");
    hBox.getChildren().add(btEnlarge);
    // Create and register the handler
    btEnlarge.setOnAction(new EnlargeHandler());
    BorderPane borderPane = new BorderPane();
    borderPane.setCenter(circle);
    borderPane.setBottom(hBox);
    BorderPane.setAlignment(hBox, Pos.CENTER);
    // Create a scene and place it in the stage
    Scene scene = new Scene(borderPane, 200, 150);
    primaryStage.setTitle("ControlCircle"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
  class EnlargeHandler implements EventHandler<ActionEvent> {
    @Override // Override the handle method
    public void handle(ActionEvent e) {

     circle.setRadius(circle.getRadius() + 2);
    }
  }
    public static void main(String[] args) {
    launch(args);
  }
}
```



Anonymous Inner Class Handlers

An anonymous inner class is an inner class without a name. It combines defining an inner class and creating an instance of the class into one step.

```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new EnlargeHandler());
}

class EnlargeHandler
    implements EventHandler<ActionEvent> {
  public void handle(ActionEvent e) {
    circlePane.enlarge();
  }
}
```

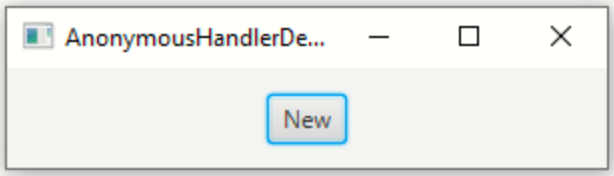
(a) Inner class `EnlargeListener`

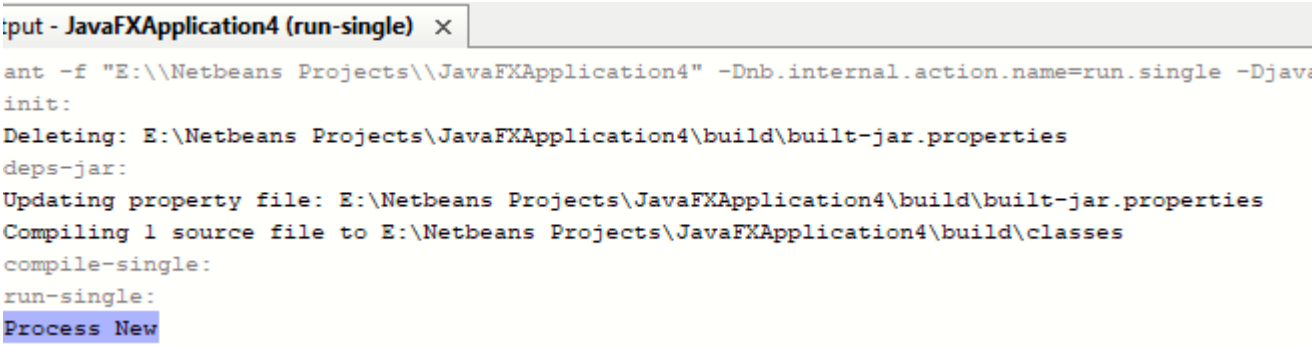
```
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new class EnlargeHandler
      implements EventHandler<ActionEvent>() {
      public void handle(ActionEvent e) {
        circlePane.enlarge();
      }
    });
}
```

(b) Anonymous inner class

Example: A JavaFX program to demonstrate anonymous inner class.

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;

public class Anonymous2 extends Application{
    @Override // Override the start method in the Application class
 public void start(javafx.stage.Stage primaryStage) {
 // Hold two buttons in an HBox
 StackPane p = new StackPane();
 p.setAlignment(Pos.CENTER);
 Button btNew = new Button("New");
 Button btOpen = new Button("Open");
 p.getChildren().addAll(btNew);
 // Create and register the handler
 btNew.setOnAction(new EventHandler<ActionEvent>() {
 @Override // Override the handle method
 public void handle(ActionEvent e) {
 System.out.println("Process New");
 }
 });
 // Create a scene and place it in the stage
 Scene scene = new Scene(p, 300, 50);
 primaryStage.setTitle("AnonymousHandlerDemo"); // Set title
 primaryStage.setScene(scene); // Place the scene in the stage
 primaryStage.show(); // Display the stage
 }
 public static void main(String[] args) {
         launch(args);
     }
 }
```





Mouse and Key events

❖ A MouseEvent is fired whenever a mouse button is pressed, released, clicked, moved, or dragged on a node or a scene.

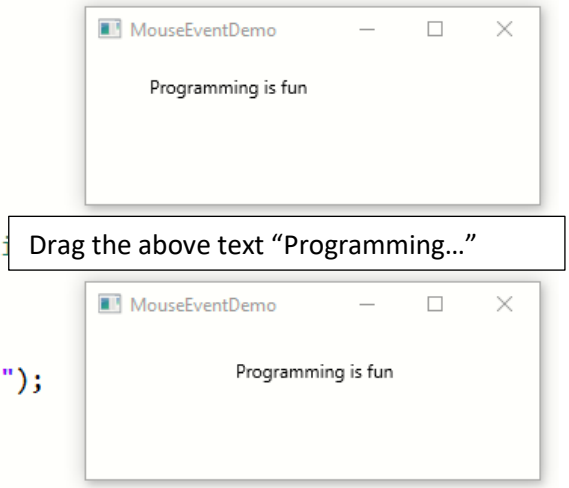
javafx.scene.input.MouseEvent	
+getButton(): MouseButton	Indicates which mouse button has been clicked.
+getClickCount(): int	Returns the number of mouse clicks associated with this event.
+getX(): double	Returns the *x*-coordinate of the mouse point in the event source node.
+getY(): double	Returns the *y*-coordinate of the mouse point in the event source node.
+getSceneX(): double	Returns the *x*-coordinate of the mouse point in the scene.
+getSceneY(): double	Returns the *y*-coordinate of the mouse point in the scene.
+getScreenX(): double	Returns the *x*-coordinate of the mouse point in the screen.
+getScreenY(): double	Returns the *y*-coordinate of the mouse point in the screen.
+isAltDown(): boolean	Returns true if the `Alt` key is pressed on this event.
+isControlDown(): boolean	Returns true if the `Control` key is pressed on this event.
+isMetaDown(): boolean	Returns true if the mouse `Meta` button is pressed on this event.
+isShiftDown(): boolean	Returns true if the `Shift` key is pressed on this event.

❖ A KeyEvent is fired whenever a key is pressed, released, or typed on a node or a scene.
Every key event has an associated code that is returned by the getCode() method in KeyEvent. The key codes are constants defined in KeyCode.

javafx.scene.input.KeyEvent	
+getCharacter(): String	Returns the character associated with the key in this event.
+getCode(): KeyCode	Returns the key code associated with the key in this event.
+getText(): String	Returns a string describing the key code.
+isAltDown(): boolean	Returns true if the `Alt` key is pressed on this event.
+isControlDown(): boolean	Returns true if the `Control` key is pressed on this event.
+isMetaDown(): boolean	Returns true if the mouse `Meta` button is pressed on this event.
+isShiftDown(): boolean	Returns true if the `Shift` key is pressed on this event.

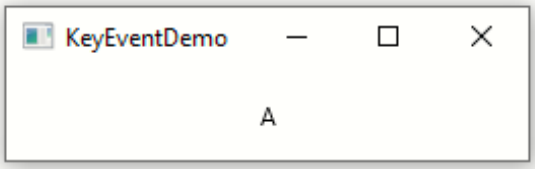
Example: Write a JavaFX program to demonstrate **mouse event** to drag text.

```java
package Chapter13;

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

public class MouseEventDemo extends Application {
  @Override // Override the start method in the Appli
  public void start(Stage primaryStage) {
    // Create a pane and set its properties
    Pane pane = new Pane();
    Text text = new Text(20, 20, "Programming is fun");
    pane.getChildren().addAll(text);
    text.setOnMouseDragged(e -> {
      text.setX(e.getX());
      text.setY(e.getY());
    });

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 300, 100);
    primaryStage.setTitle("MouseEventDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
```



Example: Write a JavaFX program to demonstrate **key event** to display a key, pressed from keyboard.

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Text;
import javafx.stage.Stage;
public class KeyEventDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane and set its properties
    StackPane pane = new StackPane();
    Text text = new Text(20, 20, "A");
    pane.getChildren().add(text);
    text.setOnKeyPressed(e -> {
          text.setText(e.getText());
    });
    // Create a scene and place the pane in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("KeyEventDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
    text.requestFocus(); // text is focused to receive key input
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```



Listeners for Observable objects

You can add a listener to process a value change in an observable object.

An instance of Observable is known as an **observable object**, which contains the **addListener (InvalidationListener listener)** method for adding a listener. The listener class must implement the InvalidationListener interface to override the **invalidated(Observable o)** method for handling the value change.

Once the value is changed in the Observable object, the listener is notified by invoking its invalidated(Observable o) method. Every binding property is an instance of Observable.

Following gives an example of observing and handling a change in a DoubleProperty object balance.

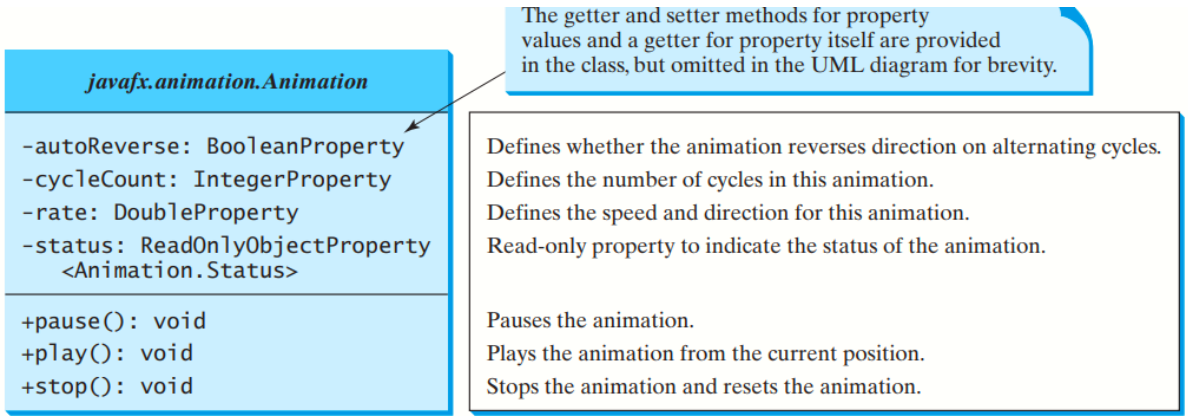
```
import javafx.beans.Observable;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.DoubleProperty;

  public class ObservableObj {
  public static void main(String[] args) {
  DoubleProperty balance = new SimpleDoubleProperty();
  balance.addListener((Observable ov) -> {
      System.out.println("The new value is " +
              balance.doubleValue());
  });
 balance.set(4.5);
 }
 }
```

```
The new value is 4.5
```

7.9 Animation

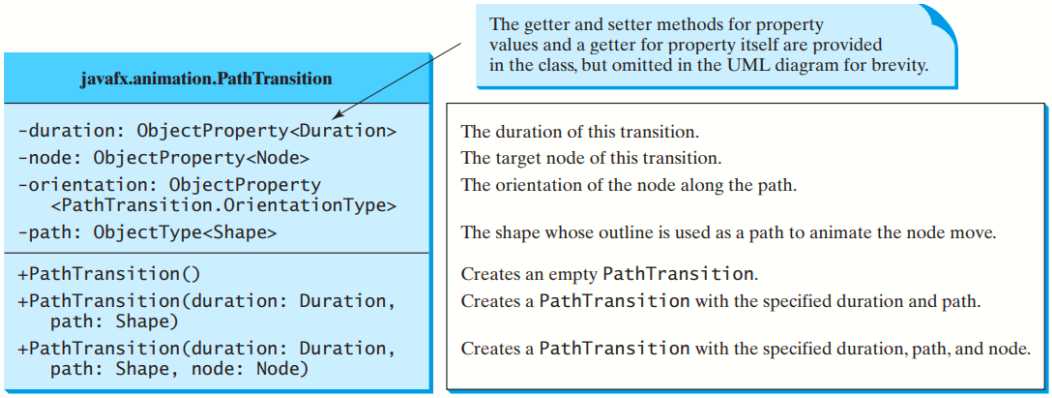
JavaFX provides the Animation class with the core functionality for all animations.

The abstract Animation class is the root class for JavaFX animations.



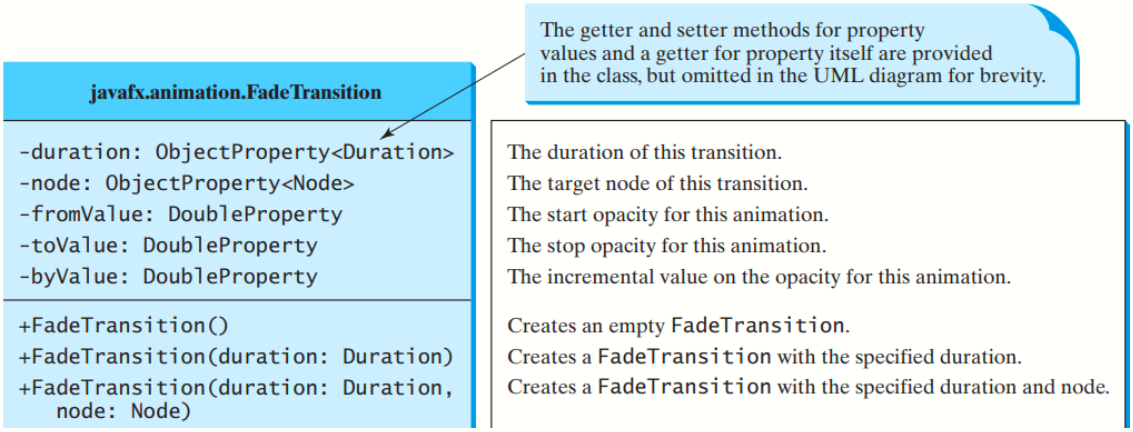
❖ PathTransition

The PathTransition class animates the the moves of a node along a path from one end to the other over a given time. PathTransition is a subtype of Animation



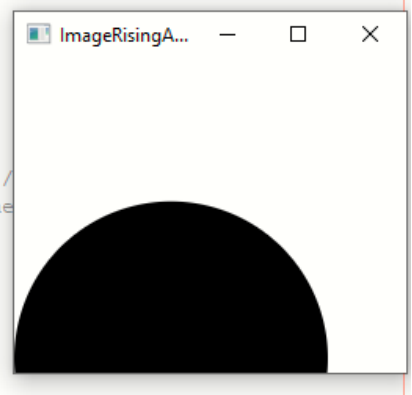
❖ FadeTransition

The FadeTransition class animates the change of the opacity in a node over a given time. FadeTransition is a subtype of Animation.



Example: JavaFX animation program to demonstrate **path transition** to rise the circle.

```
import javafx.animation.PathTransition;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.stage.Stage;
import javafx.util.Duration;
public class PathTransitionAnimation extends Application {
@Override // Override the start method in the Application class
public void start(Stage primaryStage) {
// Create a pane
Pane pane = new Pane();
Circle c = new Circle(100);
pane.getChildren().add(c);

// Create a path transition
PathTransition pt = new PathTransition(Duration.millis(10000),
new Line(100, 200, 100, 100), c);
pt.setCycleCount(5);
pt.play(); // Start animation

// Create a scene and place it in the stage
Scene scene = new Scene(pane, 250, 200);
primaryStage.setTitle("ImageRisingAnimation");
primaryStage.setScene(scene); // Place the scene
primaryStage.show(); // Display the stage
}
public static void main(String[] args) {
        launch(args);
    }
}
```



Example: A JavaFX animation program to demonstrate **fade transition**.

```
import javafx.animation.FadeTransition;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
import javafx.util.Duration;

public class FadeTransitionA                    ion {
 @Override // Override the s                    ation class
 public void start(Stage pri
 // Place an ellipse to the
 StackPane pane = new StackP
 Circle c = new Circle(50);
 pane.getChildren().add(c);
 // Apply a fade transition
 FadeTransition ft =
 new FadeTransition(Duration.millis(3000), c);
 ft.setFromValue(1.0);
 ft.setToValue(0.1);
 ft.setCycleCount(Timeline.INDEFINITE);
 ft.setAutoReverse(true);
 ft.play(); // Start animation
 // Control animation
 c.setOnMousePressed(e -> ft.pause());
 c.setOnMouseReleased(e -> ft.play());
 // Create a scene and place it in the stage
 Scene scene = new Scene(pane, 200, 150);
 primaryStage.setTitle("FadeTransitionDemo"); // Set the stage title
 primaryStage.setScene(scene); // Place the scene in the stage
 primaryStage.show(); // Display the stage
 }
 public static void main(String[] args) {
            launch(args);
          }
 }
```