

Unit-7

JAVA FX

Outline

- ✓ What is JavaFX?
- ✓ Architecture of JavaFX API
- ✓ JavaFX Application Structure
- ✓ Lifecycle of JavaFX Application
- ✓ 2D Shape
- ✓ JavaFX - Colors
- ✓ JavaFX – Image
- ✓ Layout Panes
- ✓ JavaFX – Events
- ✓ Property Binding
- ✓ Animation

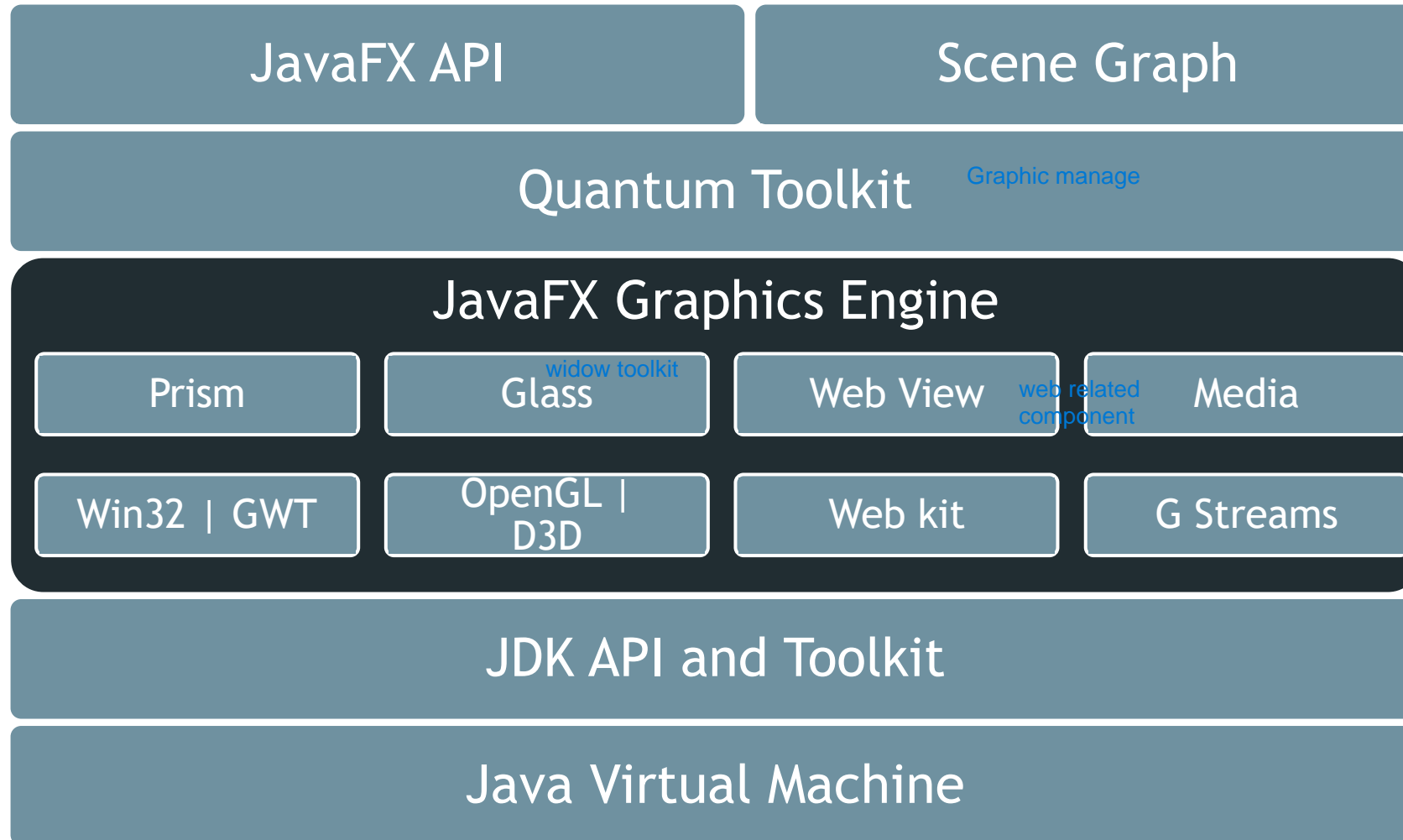
What is JavaFX?

- ▶ *JavaFX* is a Java library used to build Rich Internet Applications (RIA) and Desktop Applications.
- ▶ The applications developed using JavaFX can run on various devices such as Desktop Computers, Mobile Phones, TVs, Tablets, etc.
- ▶ To develop GUI Applications using Java programming language, the programmers rely on libraries such as Advanced Windowing Toolkit (AWT) and Swing. After the advent of JavaFX, these Java programmers can now develop GUI applications effectively with rich content.
- ▶ Why we need JavaFX
 - ▶ To develop Client Side Applications with rich features, the programmers used to depend on various libraries to add features such as Media, UI controls, Web, 2D and 3D, etc.
 - ▶ JavaFX provides a rich set of graphics and media API's and it leverages the modern Graphical Processing Unit through hardware accelerated graphics.
 - ▶ One can use JavaFX with JVM based technologies such as Java, Groovy and JRuby. If developers opt for JavaFX, there is no need to learn additional technologies.

Features of JavaFX

- Written in Java
- Scene Builder
- Swing Interoperability
- Built-in UI controls
- CSS like Styling
- Canvas and Printing API
- Rich set of API's
- Graphics pipeline
- FXML

Architecture of JavaFX API



Architecture of JavaFX API (Cont.)

► Scene Graph

- A Scene Graph is the starting point of the construction of the GUI Application. It holds the (GUI) application primitives that are termed as nodes.
- A node is a visual/graphical object and it may include
 - Geometrical (Graphical) objects
 - UI controls
 - Containers
 - Media elements

► Prism

- Prism is a high performance hardware-accelerated graphical pipeline that is used to render the graphics in JavaFX. It can render both 2-D and 3-D graphics.

► GWT (Glass Windowing Toolkit)

- GWT provides services to manage Windows, Timers, Surfaces and Event Queues.
- GWT connects the JavaFX Platform to the Native Operating System.

▶ Quantum Toolkit

- ▶ It is an abstraction over the low-level components of Prism, Glass, Media Engine, and Web Engine. It ties Prism and GWT together and makes them available to JavaFX.

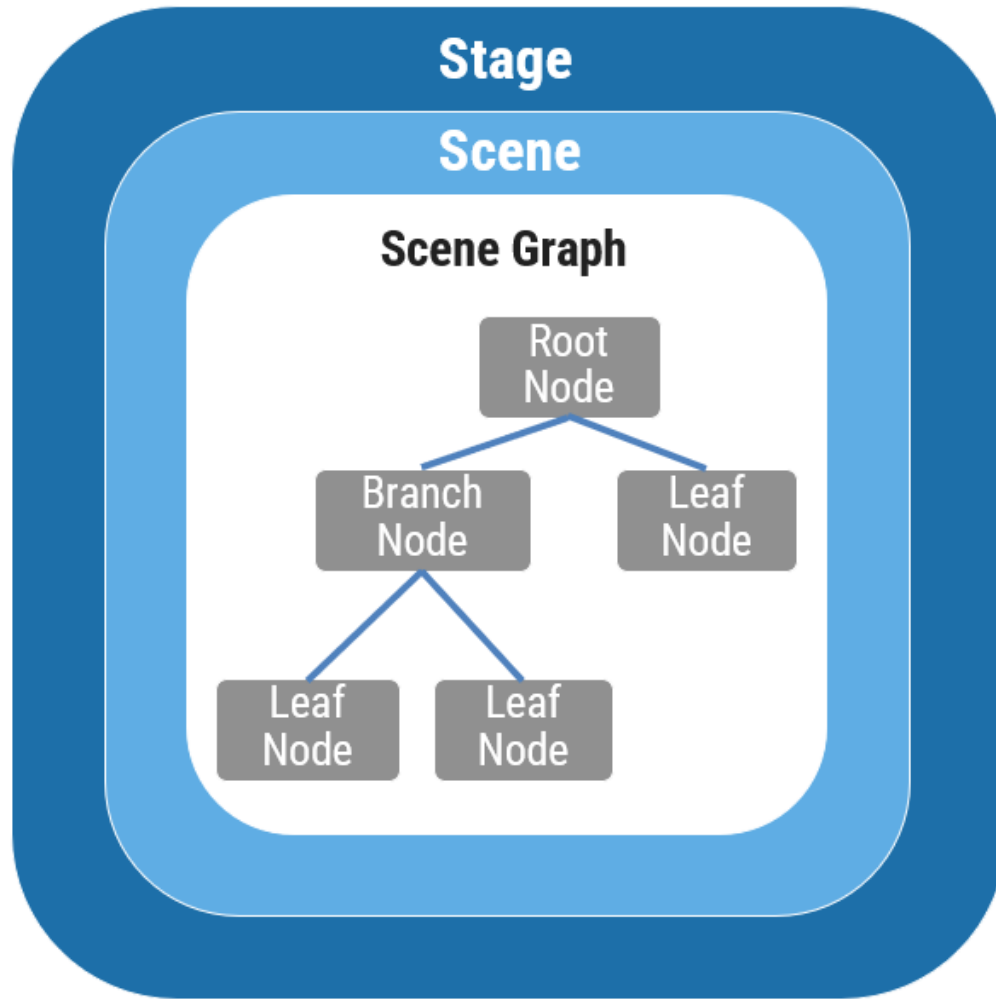
▶ WebView

- ▶ WebView is the component of JavaFX which is used to process HTML content. It uses a technology called Web Kit, which is an internal open-source web browser engine. This component supports different web technologies like HTML5, CSS, JavaScript, DOM and SVG.

▶ Media Engine

- ▶ The JavaFX media engine is based on an open-source engine known as a Streamer. This media engine supports the playback of video and audio content.

JavaFX Application Structure



Stage

- ▶ A stage (a window) contains all the objects of a JavaFX application.
- ▶ It is represented by **Stage** class of the package **javafx.stage**.
- ▶ The primary stage is created by the platform itself. The created stage object is passed as an argument to the **start()** method of the **Application** class.
- ▶ A stage has two parameters determining its position namely Width and Height.
- ▶ There are five types of stages available
 - ▶ Decorated // `stageObj.initStyle(StageStyle.DECORATED);`
 - ▶ Undecorated // `stageObj.initStyle(StageStyle.UNDECORATED);`
 - ▶ Transparent// `stageObj.initStyle(StageStyle.TRANSPARENT);`
 - ▶ Unified // `stageObj.initStyle(StageStyle.UNIFIED);`
 - ▶ Utility // `stageObj.initStyle(StageStyle.UTILITY);`
- ▶ You have to call the **show()** method to display the contents of a stage.

Scene

- A scene represents the physical contents of a JavaFX application. It contains all the contents of a scene graph.
- The class `Scene` of the package `javafx.scene` represents the scene object. At an instance, the scene object is added to only one stage.
- You can create a scene by instantiating the `Scene` Class.
- You can opt for the size of the scene by passing its dimensions (height and width) along with the root node to its constructor.

Scene Graph and Nodes

- A scene graph is a tree-like data structure (hierarchical) representing the contents of a scene. In contrast, a node is a visual/graphical object of a scene graph. A node may include,
 - **Geometrical** (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon, etc.
 - **UI Controls** – Button, Checkbox, Choice Box, Text Area, etc.
 - **Containers** (Layout Panes) – Border Pane, Grid Pane, Flow Pane, etc.
 - **Media elements** – Audio, Video and Image Objects.

- The **Node** class of the package **javafx.scene** represents a node in JavaFX, this class is the super class of all the nodes.

Root Node – First Scene Graph is known as Root node. It's mandatory to pass root node to the scene graph.

Branch Node/Parent Node – The node with child nodes are known as branch/parent nodes. The abstract class named **Parent** of the package **javafx.scene** is the base class of all the parent nodes, and those parent nodes will be of the following types

Group – A group node is a collective node that contains a list of children nodes.

Region – It is the base class of all the JavaFX Node based UI Controls, such as Chart, Pane and Control.

WebView – This node manages the web engine and displays its contents.

Leaf Node – The node without child nodes is known as the leaf node. For example, Rectangle, Ellipse, Box, ImageView, MediaView are examples of leaf nodes.

Steps to create JavaFX application

1. Extend the Application class of `javafx.application` package in your class.
2. Override `start` method of Application class.
3. Prepare a scene graph with the required nodes.
4. Prepare a Scene with the required dimensions and add the scene graph (root node of the scene graph) to it.
5. Prepare a stage and add the scene to the stage and display the contents of the stage.

3. Prepare a Scene graph

- Since the root node is the first node, you need to create a root node and it can be chosen from the *Group*, *Region* or *WebView*.

- Group

A Group node is represented by the class named **Group** which belongs to the package **javafx.scene**, you can create a Group node by instantiating this class as shown below.

```
Group root = new Group();  
Group root = new Group(NodeObject);
```

- Region

It is the Base class of all the JavaFX Node-based UI Controls, such as –

- **Chart** – This class is the base class of all the charts and it belongs to the package **javafx.scene.chart** which embeds charts in application.
- **Pane** – A Pane is the base class of all the layout panes such as AnchorPane, BorderPane, DialogPane, etc. This class belong to a package that is called as – **javafx.scene.layout** which inserts predefined layouts in your application.
- **Control** – It is the base class of the User Interface controls such as Accordion, ButtonBar, ChoiceBox, ComboBoxBase, HTML editor, etc. This class belongs to the package **javafx.scene.control**.

- WebView

This node manages the web engine and displays its contents.

4. Preparing the Scene

- A JavaFX scene is represented by the **Scene** class of the package **javafx.scene**.

```
Scene scene = new Scene(root,width,height);
```

- While instantiating, it is mandatory to pass the root object to the constructor of the **Scene** class whereas width and height of the scene are optional parameters to the constructor.

5. Preparing the Stage

- Stage is the container of any JavaFX application and it provides a window for the application. It is represented by the **Stage** class of the package **javafx.stage**.
- An object of this class is passed as a parameter of the **start()** method of the Application class.
- Using this object, various operations on the stage can be performed like
 - *Set the title* for the stage using the method setTitle().
`primaryStage.setTitle("Sample application");`
 - *Attach the scene* object to the stage using the setScene() method.
`primaryStage.setScene(scene);`
 - *Display the contents* of the scene using the show() method as shown below.
`primaryStage.show();`

Lifecycle of JavaFX Application

- The JavaFX **Application** class has three life cycle methods.
 - init()** – An empty method which can be overridden, stage or scene cannot be created in this method.
 - start()** – The entry point method where the JavaFX graphics code is to be written.
 - stop()** – An empty method which can be overridden, here the logic to stop the application is written.
- It also provides a static method named **launch()** to launch JavaFX application. This method is called from static content only mainly in main method.
- Whenever a JavaFX application is launched, the following actions will be carried out (in the same order).
 - An instance of the application class is created.
 - **init()** method is called.
 - **start()** method is called.
 - The launcher waits for the application to finish and calls the **stop()** method.

Basic Example of JavaFX

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

```
public class MyFirstGUI extends Application{
    public void start(Stage primaryStage) throws Exception{
        Group root = new Group();
        Scene s = new Scene(root,600,400);
        primaryStage.setTitle("My First User Interface");
        s.setFill(Color.ORANGE);
        primaryStage.setScene(s);
        primaryStage.show();
    }

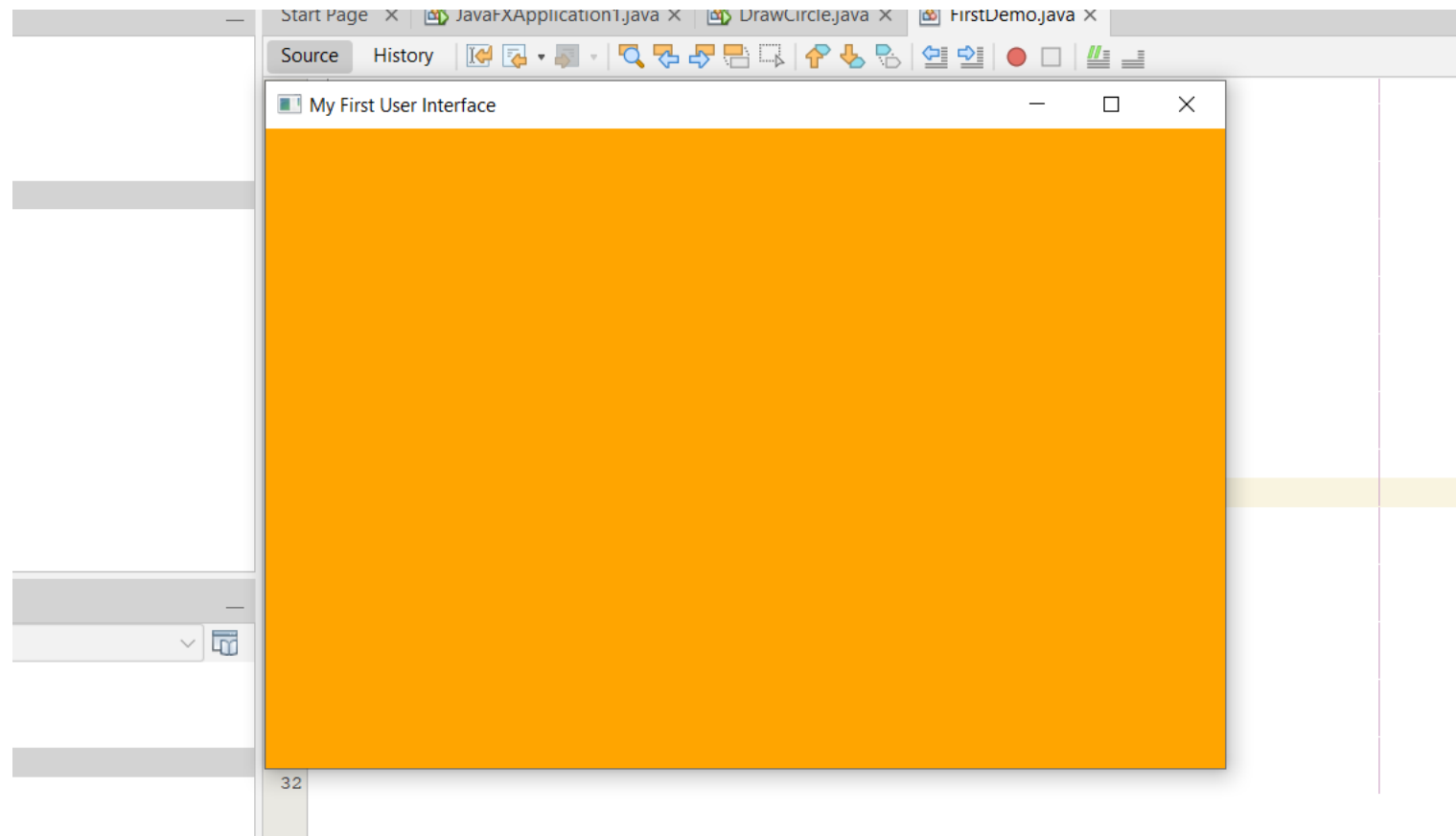
    public static void main(String[] args) {
        launch(args);
    }
}
```

Create Scene Graph
using Group, Region
or WebView

Create Scene by adding
Group (root) to it along
with its width and
height

Set the scene to the stage object
(primaryStage) which is passed as an
argument to start() using setScene()
method

Output



Color class

- In JavaFX, we can fill the shapes with the colors.
- Javafx.scene.paint package provides various classes to apply colors to application.
- We have the flexibility to create our own color using the several methods and pass that as a Paint object into the setFill() method.
- Using these classes, you can apply colors in the following patterns
 - Uniform – color is applied uniformly throughout node.
 - Image Pattern – fills the region of the node with an image pattern.
 - Gradient – the color applied to the node varies from one point to the other. It has two kinds of gradients namely Linear Gradient and Radial Gradient.
- Instance of **Color** class can be created by providing Red, Green, Blue and Opacity value ranging from 0 to 1 in double.
 - `Color color = new Color(double red, double green, double blue, double opacity);`
- `Color color = new Color(0.0,0.3,0.2,1.0);`
- Instance of **Color** class can be created using following methods also
 - `Color c = Color.rgb(0,0,255); //passing RGB values`
 - `Color c = Color.hsb(270,1.0,1.0); //passing HSB values`
 - `Color c = Color.web("0x0000FF",1.0); //passing hex code`

Applying Color to the Nodes

- ▶ `setFill(Color)` method is used to apply color to nodes such as Shape, Text, etc.
- ▶ `setStroke(Color)` method is used to apply strokes to the nodes.

```
//Setting color to the text  
Color color = new Color.BEIGE  
text.setFill(color);
```

```
//Setting color to the stroke  
Color color = new Color.DARKSLATEBLUE  
circle.setStroke(color);
```

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.

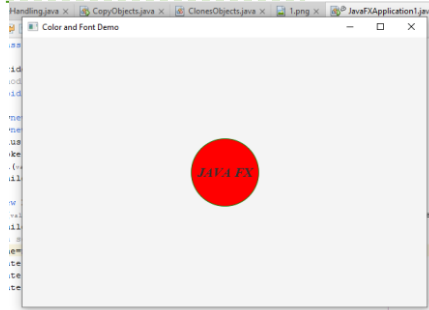
Commonly Used Methods:

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.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.shape.Circle;
import javafx.scene.text.*;
import javafx.scene.paint.Color;
```

```
public class JavaFXApplication1 extends Application {

    @Override
    //method of Application class
    public void start(Stage primaryStage) throws Exception {
        //Create a pan to hold circle(root node)
        Pane pane=new StackPane();
        Circle c1=new Circle();
        c1.setRadius(50);
        c1.setStroke(Color.GREEN);
        c1.setFill(Color.RED);
        pane.getChildren().add(c1);
```



```
Label l=new Label("JAVA FX");
l.setFont(Font.font("Times New
Roman",FontWeight.BOLD,FontPosture.ITALIC,20));
pane.getChildren().add(l);
```

```
//create a scene and place it in the stage
Scene scene=new Scene(pane,600,400);
primaryState.setTitle("Color and Font Demo");
primaryState.setScene(scene);
primaryState.show();
}
```

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

```
}
```

JavaFX - Image

- ▶ 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.
- ▶ Class Image of `javafx.scene.image` package is used to load an image
- ▶ Any of the following argument is required to the constructor of the class

- ▶ An InputStream object of the image to be loaded or

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

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

- ▶ A string variable holding the URL for the image.

```
Image image = new Image("http://sample.com/res/flower.png");
```

- After loading image in Image object, view is set to load the image using `ImageView` class

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


Example

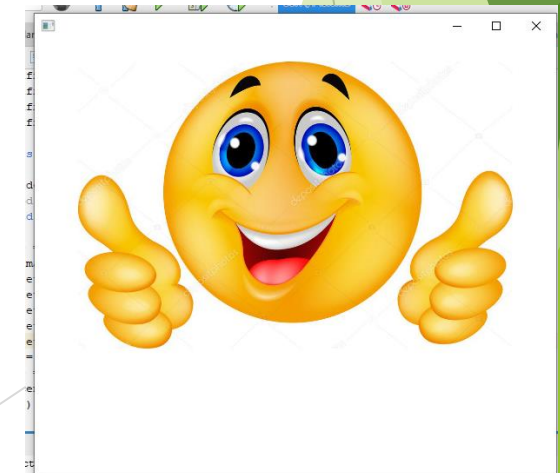
```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
```

```
public class ImageDemo extends Application {

    @Override
    //method of Application class
    public void start(Stage stage) throws Exception {
```

```
Image image = new Image("C:\\Users\\HP\\Desktop\\Javafiles\\img1.jpg");
ImageView imageView = new ImageView(image);
```

```
imageView.setX(50);
imageView.setY(25);
imageView.setFitHeight(455);
imageView.setFitWidth(500);
imageView.setPreserveRatio(true);
Group root = new Group(imageView);
Scene scene = new Scene(root, 600, 500);
stage.setScene(scene);
stage.show();
}
public static void main(String[] args)
{ launch(args);}
}
```



Inner Class

- 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.

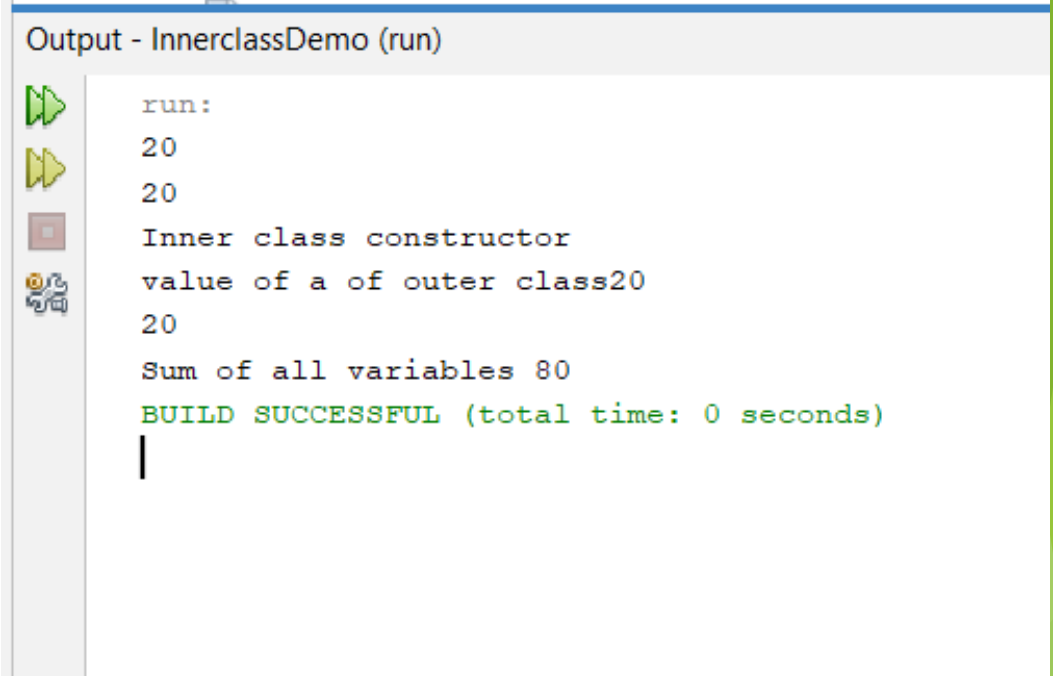
Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing an interface or extending class. The java compiler decides its name.
Local Inner Class	A class was created within the method.
Static Nested Class	A static class was created within the class.

Example of Member nested class and static nested class

```
class outer
{
    static int a=20;
    int b=50;
    private int c=10;
    public static void display()
    {
        System.out.println(a);
    }
    void printA()
    {
        System.out.println("value of a using object of inner
class OBJECT"+a);
    }
    //Member inner class
    class InnerExample
    {
        public void dosum()
        {
            System.out.println("Sum of all variables "+(a+b+c));
        }
    }
}
```

```
//static inner class
//can access outer class static variable
//can not access non-static variables and methods of
outer class
static class inner
{
    inner()
    {
        System.out.println("Inner class constructor");
        System.out.println("value of a of outer class"+a);
        display();
    }
}
}
```

```
public class InnerclassDemo {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        // TODO code application logic here  
        System.out.println(outer.a);  
        outer.display();  
  
        //Creation of inner class object  
        outer.inner ob=new outer.inner();  
  
        outer o1=new outer();  
        outer.InnerExample i1=o1.new InnerExample();  
        i1.dosum();  
    }  
}
```



The screenshot shows a window titled "Output - InnerclassDemo (run)". On the left side of the window, there is a vertical toolbar with icons for running (a green play button), stepping through (a yellow play button), stopping (a red square), and debugging (a magnifying glass over a bug). The main area of the window displays the following text:

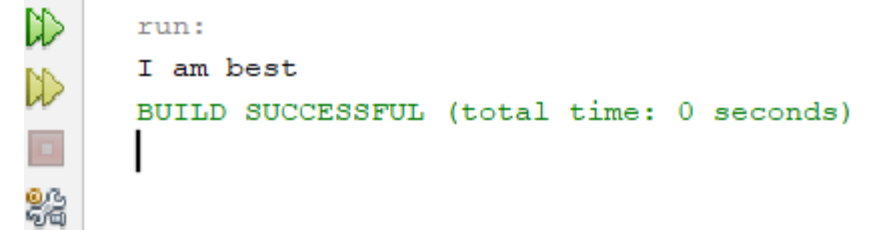
```
run:  
20  
20  
Inner class constructor  
value of a of outer class20  
20  
Sum of all variables 80  
BUILD SUCCESSFUL (total time: 0 seconds)  
|
```

Anonymous Inner Class

```
class A
{
    public void show()
    {
        System.out.println("I am good");
    }
}

public class AnonymousDemo {
    public static void main(String[] args) {
        A obj=new A()
        {
            public void show()
            {
                System.out.println("I am best");
            }
        };
        obj.show();
    }
}
```

Output - InnerclassDemo (run)



```
run:
I am best
BUILD SUCCESSFUL (total time: 0 seconds)
```

An Event

- 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`

Adding an Event Handler

- Event handler is the implementation of the EventHandler interface. The handle() method of the interface contains the logic which is executed when the event is triggered.
- In JavaFX every event has –
 - Target** – The node on which an event occurred. A target can be a window, scene, and a node.
 - Source** – The source from which the event is generated will be the source of the event. In the above scenario, mouse is the source of the event.
 - Type** – Type of the occurred event; in case of mouse event – mouse pressed, mouse released are the type of events.
- To register the EventHandler, **addEventHandler()** is used. In this method, two arguments are passed. One is **event type** and the other is EventHandler object.
- The syntax of **addEventHandler()** is given below.

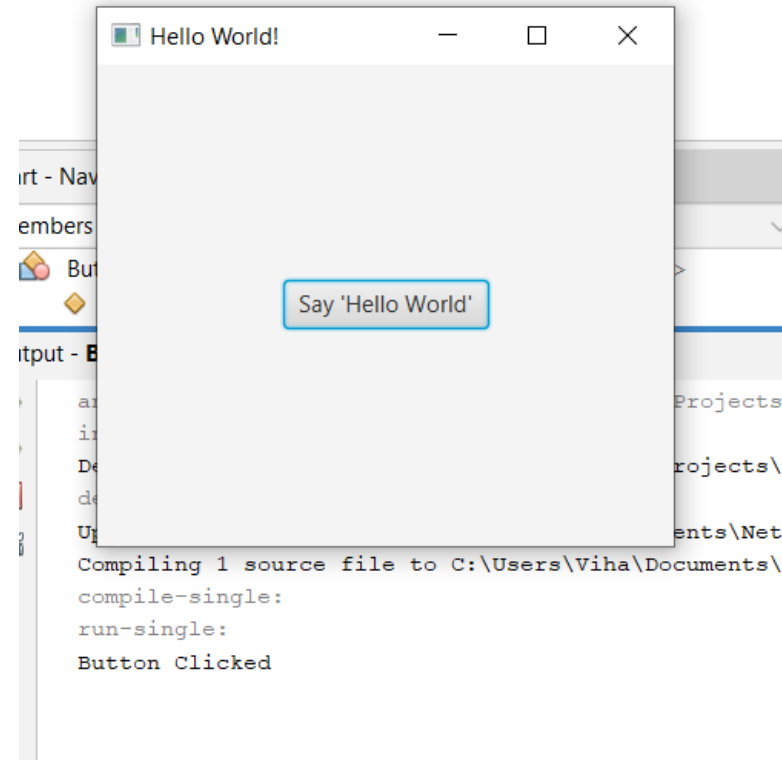
```
node.addEventHandler(<EventType>, new EventHandler<Event-
Type>()
{
    public void handle(<EventType> e)
    {
        //handling code
    }
});
```

Handling Event

```
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 ButtonClickDemo extends Application implements EventHandler<ActionEvent> {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(this);
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    @Override
    public void handle(ActionEvent event)
    {
        System.out.println("Button Clicked");
    }

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



Handling Event with the help of Inner Class

```
import javafx.application.Application;import javafx.event.ActionEvent;import javafx.event.EventHandler;
import javafx.scene.Scene;import javafx.scene.control.*;import javafx.scene.layout.*;
import javafx.scene.shape.*;import javafx.geometry.Pos;import javafx.stage.Stage;
```

```
public class ButtonClickDemo extends Application{
    private final Circle circle = new Circle(50);
    @Override
    public void start(Stage primaryStage) {

        HBox hbox=new HBox();
        hbox.setSpacing(10);
        hbox.setAlignment(Pos.CENTER);
        Button b1=new Button("Enlarge");
        hbox.getChildren().add(b1);
        b1.setOnAction(new EnlargeHandler());
        BorderPane bpane=new BorderPane();
        bpane.setCenter(circle);
        bpane.setBottom(hbox);
        bpane.setAlignment(hbox,Pos.CENTER);
        Scene scene=new Scene(bpane,200,150);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

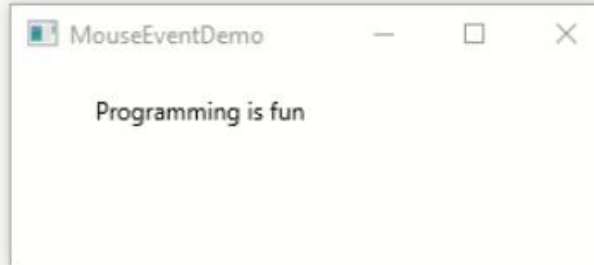
```
class EnlargeHandler implements EventHandler<ActionEvent>
{
    @Override
    public void handle(ActionEvent event)
    {
        circle.setRadius(circle.getRadius()+2);
    }
}

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

Mouse Event

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

```
1 package Chapter13;
2
3 import javafx.application.Application;
4 import javafx.scene.Scene;
5 import javafx.scene.layout.Pane;
6 import javafx.scene.text.Text;
7 import javafx.stage.Stage;
8
9 public class MouseEventDemo extends Application {
10     @Override // Override the start method in the Application class
11     public void start(Stage primaryStage) {
12         // Create a pane and set its properties
13         Pane pane = new Pane();
14         Text text = new Text(20, 20, "Programming is fun");
15         pane.getChildren().addAll(text);
16         text.setOnMouseDragged(e -> {
17             text.setX(e.getX());
18             text.setY(e.getY());
19         });
20
21         // Create a scene and place it in the stage
22         Scene scene = new Scene(pane, 300, 100);
23         primaryStage.setTitle("MouseEventDemo"); // Set the stage title
24         primaryStage.setScene(scene); // Place the scene in the stage
25         primaryStage.show(); // Display the stage
26     }
```



Drag the above text "Programming..."



Lambda Expression :
Anonymous Function
with No name, No
Modifier and No
Return Type

Key Event

```
import javafx.application.Application; import javafx.event.EventHandler;  
import javafx.scene.Scene; import javafx.scene.text.*; import javafx.scene.layout.*;  
import javafx.stage.Stage;
```

```
public class ButtonClickDemo extends Application{  
    //private final Circle circle = new Circle(50);  
    @Override  
    public void start(Stage primaryStage) {  
        StackPane pane = new StackPane();  
        Text text = new Text();  
        pane.getChildren().add(text);  
  
        Scene scene=new Scene(pane,200,150);  
        scene.setOnKeyPressed(e->{  
            text.setText(e.getCode().toString());  
        });  
        primaryStage.setTitle("Hello World!");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
}
```

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



Animation



2D Shape

- ▶ 2D shape is a geometrical figure that can be drawn on the XY plane like Line, Rectangle, Circle, etc.
- ▶ Using the JavaFX library, you can draw –
 - ▶ Predefined shapes - Line, Rectangle, Circle, Ellipse, Polygon, Polyline, Cubic Curve, Quad Curve, Arc.
 - ▶ 2D shape by parsing SVG path.
- ▶ Each of the above mentioned 2D shape is represented by a class which belongs to the package `javafx.scene.shape`. The class named Shape is the base class of all the 2-Dimensional shapes in JavaFX.

Classes for Shape (javafx.scene.shape)

Shape	Class	Example
Line	Line	<pre>Line line = new Line(); line.setStartX(100.0); line.setStartY(150.0); line.setEndX(500.0); line.setEndY(150.0);</pre>
Rectangle & Rounded Rectangle	Rectangle	<pre>Rectangle rectangle = new Rectangle(); rectangle.setX(150.0); rectangle.setY(75.0); rectangle.setWidth(300.0); rectangle.setHeight(150.0); rectangle.setArcWidth(30.0); rectangle.setArcHeight(20.0);</pre>
Circle	Circle	<pre>Circle circle = new Circle(); circle.setCenterX(300.0); circle.setCenterY(135.0); circle.setRadius(100.0);</pre>

(Cont.)

Shape	Class	Example
Ellipse	Ellipse	<pre>Ellipse ellipse = new Ellipse(); ellipse.setCenterX(300.0); ellipse.setCenterY(150.0); ellipse.setRadiusX(150.0); ellipse.setRadiusY(75.0);</pre>

Example (Adding 2-D Shapes)

```
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 MyFirstGUI extends Application {
    public void start(Stage primaryStage) throws Exception {
        Ellipse ellipse = new Ellipse();
        ellipse.setCenterX(300);
        ellipse.setCenterY(150);
        ellipse.setRadiusX(150);
        ellipse.setRadiusY(75);
        ellipse.setStroke(Color.BLACK);
        ellipse.setFill(Color.WHITE);
        Group root = new Group(ellipse);
        Scene s = new Scene(root, 600, 400);
        primaryStage.setScene(s);
        primaryStage.show();
    }
    public static void main(String[] args) {
        Launch(args);
    }
}
```

Create object of Ellipse class and set its properties.

Pass object of ellipse as argument to Group constructor

Layout Panes

- ▶ After constructing all the required nodes in a scene, we will generally arrange them in order.
- ▶ This arrangement of the components within the container is called the Layout of the container.
- ▶ JavaFX provides several predefined layouts such as HBox, VBox, Border Pane, Stack Pane, Text Flow, Anchor Pane, Title Pane, Grid Pane, Flow Panel, etc.
- ▶ Each of the above mentioned layout is represented by a class and all these classes belongs to the package `javafx.layout`. The class named **Pane** is the base class of all the layouts in JavaFX.

<i>Class</i>	<i>Description</i>
Pane	Base class for layout panes. It contains the <code>getChildren()</code> 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.

Thank you...!!!!