SNS COLLEGE OF ENGINEERING

Coimbatore-107

An Autonomous Institution

COURSE NAME : 23CSB201 & Object Oriented Programming

II YEAR/ III SEMESTER

UNIT – V JAVAFX EVENT HANDLING, CONTROLS & COMPONENTS

Topic: JavaFX Events and Controls

Dr.P.Poonkodi

Assistant Professor(SG)

Department of Computer Science and Technology

Introduction

- **JavaFX** is a tool in Java to **build graphical user interfaces (GUIs)**.
- It helps create **windows**, **buttons**, **text boxes**, and **other visual parts** of a program.
- Think of JavaFX like building blocks for apps: you add buttons, text fields, etc., and respond when the user clicks or types!
- JavaFX is a set of graphics and media package
- Enable developer to design, create , test, debug and deploy rich client application
- It is a library used to develop desktop application
- Run on multiple platform
- Intended to replace swing
- Doesn’t depend on operating system

JavaFX Features

- JavaFX is a rich client framework for building desktop and Internet-based applications. It is part of the Java platform and offers modern GUI capabilities

Features	Description
Rich Set of UI Controls	•JavaFX provides many ready-to-use UI components like Button, Label, TableView, TreeView, ComboBox, etc. •It supports complex controls like charts (PieChart, BarChart) and tables (TableView).
CSS Styling	•You can style JavaFX applications using **CSS** (Cascading Style Sheets). •This separates design from logic and makes it easier to customize the UI.
FXML Support	•JavaFX uses **FXML**, an XML-based language, to define the UI structure separately from the Java code. •It promotes the **MVC (Model-View-Controller)** design pattern. •Designers can work on the FXML file while developers code the logic.

Cond.,

Feature	Benefit
UI Controls	Build rich user interfaces easily
CSS Styling	Customize appearance without coding
FXML	Separate UI design and logic
3D Graphics	Create immersive visual experiences
Animations and Effects	Enhance user interaction
WebView	Embed web content
Media Support	Play audio and video
Cross-Platform	Write once, run anywhere
Modular Design	Lightweight and customizable deployments

Controls in JavaFX

- **Controls** are the **ready-made elements** you can use to build your app's interface
- Example

Control	Purpose
`Button`	A clickable button.
`Label`	Shows text (not editable).
`TextField`	Allows typing a single line.
`CheckBox`	Box that can be checked/unchecked.
`RadioButton`	Choose only one option.
`Slider`	Select a value by sliding.
`ListView`	Display a list of items.

Example

- **Button myButton = new Button("Click Me");**
- You can **add** these controls to layouts like VBox, HBox, or StackPane to arrange them on the screen

Events in JavaFX

- An **event** happens when the **user interacts** with your program
- Example

Action	Event Name
User clicks a button	`ActionEvent`
User moves the mouse	`MouseEvent`
User types on the keyboard	`KeyEvent`

How to Handle Events

- You **tell** JavaFX **what to do** when an event happens
- Example: Button Click Event

```
Button myButton = new Button("Click Me");

myButton.setOnAction(event -> {
    System.out.println("Button was clicked!");
});
```

Full Example

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class SimpleJavaFXApp extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Create a button
        Button btn = new Button("Say Hello");
```

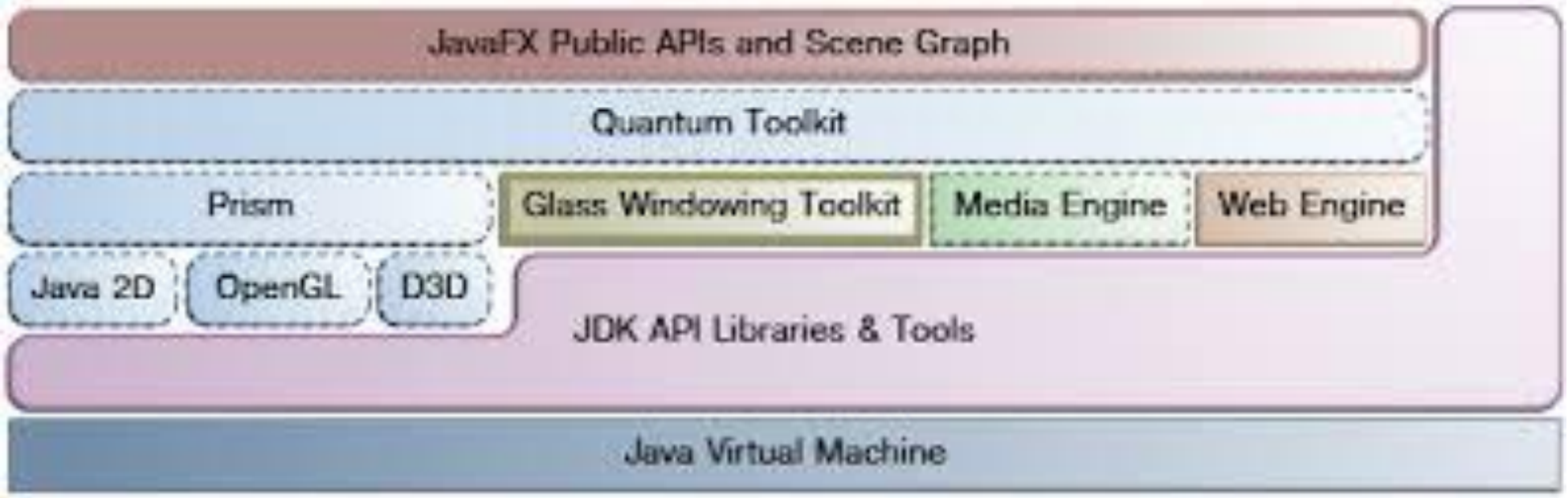
Full Example

```
// Set an event when the button is clicked
 btn.setOnAction(e -> System.out.println("Hello, JavaFX!"));
 // Add the button to a layout
 StackPane root = new StackPane();
 root.getChildren().add(btn);
 // Set up the scene
 Scene scene = new Scene(root, 300, 200);
 // Set up the stage (window)
 primaryStage.setTitle("My First JavaFX App");
 primaryStage.setScene(scene);
 primaryStage.show();
    }
```

Full Example

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

JavaFX Architecture



Lifecycle methods in JavaFX

i) void init()

The init( ) method is called when the application begins execution. It is used to perform various initializations

ii) abstract void start(Stage primarystage)

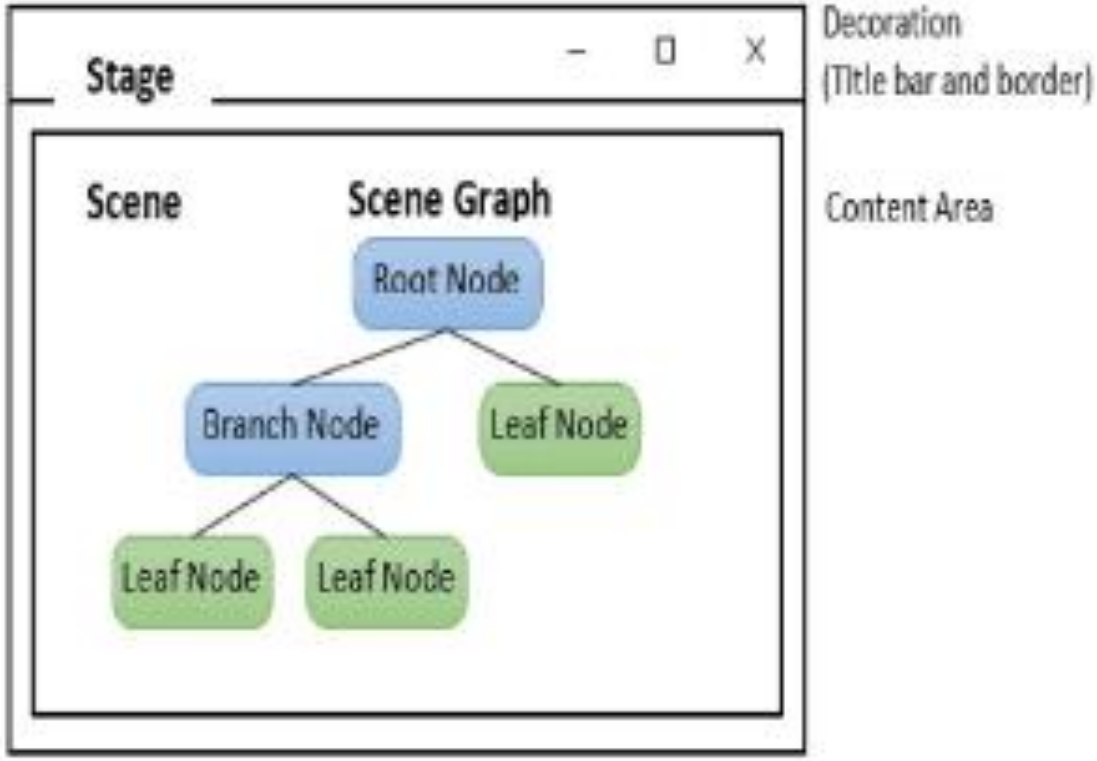
The start( ) method is called after init( ) that can be used to construct and set the scene.

iii) void stop()

stop() method is called when the application is terminated

JavaFX Application Architecture

- Divided into three main components known as stage, scene and nodes



Event Basics

- An **Event** occurs when
  - A user **clicks** a button
  - A user **presses** a key
  - A user **moves** the mouse
  - A window **closes** or **opens**
- Java provides special **classes** to represent these events, such as:
  - ActionEvent
  - MouseEvent
  - KeyEvent
  - WindowEvent

Handling **Key** and **Mouse** Events

- handle **keyboard** and **mouse** events in **Java GUI programs** (like Swing or JavaFX) by:
  - **Listening** for the user's actions
  - **Responding** with code when a key is pressed or the mouse is clicked/moved

Handling Key Events (Keyboard)

- You use a **KeyListener** to listen for **keyboard events** like:
  - **Key Pressed** (key goes down)
  - **Key Released** (key goes up)
  - **Key Typed** (key pressed and released)

Handling Key Events (Keyboard)

```
import javax.swing.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
public class KeyEventDemo extends JFrame implements KeyListener {
    JLabel label;
    int x = 100, y = 100;
    KeyEventDemo() {
        label = new JLabel("Move Me!");
        label.setBounds(x, y, 100, 30);
        add(label);
        addKeyListener(this);
```

Handling Key Events (Keyboard)

```
        setSize(400, 400);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void keyPressed(KeyEvent e) {
        int code = e.getKeyCode();
        if (code == KeyEvent.VK_UP) {
            y -= 10;
        } else if (code == KeyEvent.VK_DOWN) {
            y += 10;
        }
```

Handling Key Events (Keyboard)

```
        else if (code == KeyEvent.VK_LEFT) {
            x -= 10;
        } else if (code == KeyEvent.VK_RIGHT) {
            x += 10;
        }  label.setLocation(x, y);
    }
    public void keyReleased(KeyEvent e) {}
    public void keyTyped(KeyEvent e) {}
    public static void main(String[] args) {
        new KeyEventDemo();
    }
}
```

Handling Mouse Events

- Use a **MouseListener** or **MouseMotionListener** for **mouse events** like:
  - Mouse **Clicked**
  - Mouse **Pressed**
  - Mouse **Released**
  - Mouse **Entered** (into a component)
  - Mouse **Exited** (out of a component)
  - Mouse **Moved**
  - Mouse **Dragged**

Handling Mouse Events

```
import javax.swing.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
public class MouseEventDemo extends JFrame implements MouseListener {
    JLabel label;
    MouseEventDemo() {
        label = new JLabel("Click anywhere!");
        label.setBounds(100, 100, 150, 30);
        add(label);
        addMouseListener(this);
        setSize(400, 400);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
```

Handling Mouse Events

```
    public void mouseClicked(MouseEvent e) {
        label.setText("Mouse Clicked at (" + e.getX() + ", " + e.getY() + ")");
    }

    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}

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

References

- Java : the complete Reference (Eleventh Edition), Herbert Schildt, 2018.


Thank You!