Unit 4

JavaFX

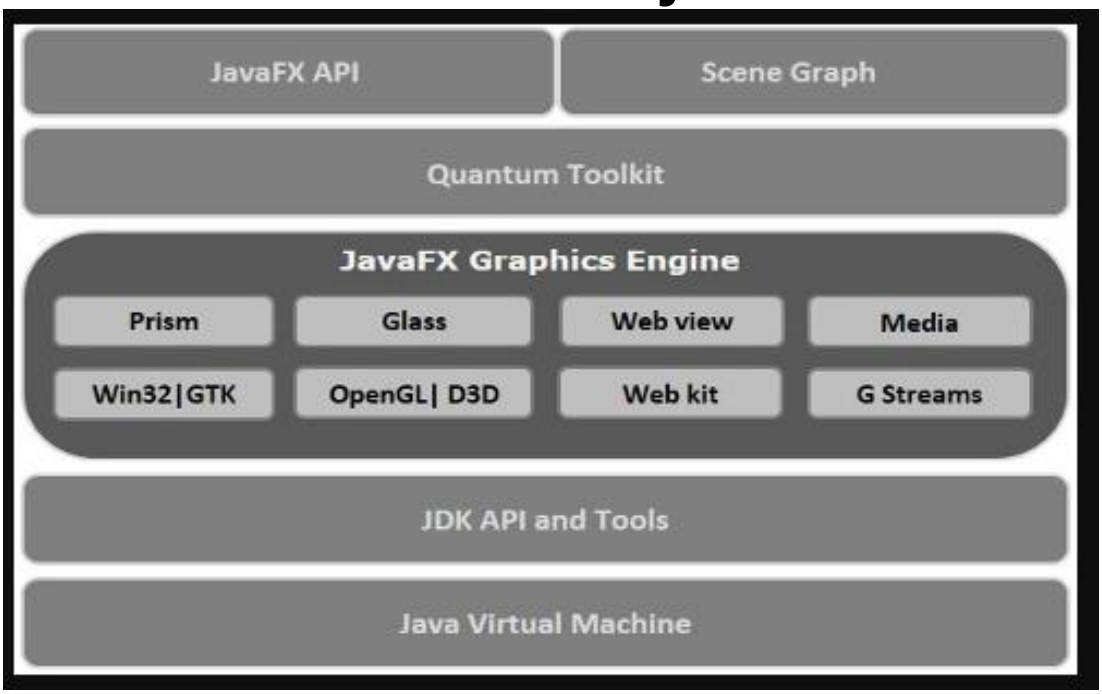
- JavaFX is a Java library and a GUI toolkit designed to develop and facilitate Rich Internet applications, web applications, and desktop applications.
- The most significant perk of using JavaFX is that the applications written using this library can run on multiple operating systems like Windows, Linux, iOS, Android, and several platforms like Desktops, Web, Mobile Phones, TVs, Tablets, etc.

Features of JavaFX

- **Java Library** – JavaFX is a Java library, which allows the user to gain the support of all the Java characteristics such as multithreading, generics, lambda expressions, and many more. The user can also use any of the Java editors or IDE's of their choice, such as Eclipse, NetBeans, to write, compile, run, debug, and package their JavaFX application.
- **Platform Independent** – The rich internet applications made using JavaFX are platform-independent. The JavaFX library is open for all the scripting languages that can be administered on a JVM, which comprise – Java, Groovy, Scala, and JRuby.
- **FXML** – JavaFX emphasizes an HTML-like declarative markup language known as FXML. FXML is based on extensible markup language (XML). The sole objective of this markup language is to specify a user interface (UI). In FXML, the programming can be done to accommodate the user with an improved GUI.

- **CSS Styling** – Just like websites use CSS for styling, JavaFX also provides the feature to integrate the application with CSS styling. The users can enhance the styling of their applications and can also improve the outlook of their implementation by having simple knowledge and understanding of CSS styling.
- **High-Performance media engine** – Like the graphics pipeline, JavaFX also possesses a media pipeline that advances stable internet multimedia playback at low latency. This high-performance media engine or media pipeline is based on a multimedia framework known as Gstreamer.
- **Rich set of APIs** – JavaFX library also presents a valuable collection of APIs that helps in developing GUI applications, 2D and 3D graphics, and many more

Architecture of JavaFX



- **JavaFX API** – The topmost layer of JavaFX architecture holds a JavaFX public API that implements all the required classes that are capable of producing a full-featured JavaFX application with rich graphics. The list of all the important packages of this API is as follows.
- **javafx.animation**: It includes classes that are used to combine transition-based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes (collection of nodes makes a scene graph).
- **javafx.css** – It comprises classes that are used to append CSS–like styling to the JavaFX GUI applications.
- **javafx.scene** – This package of JavaFX API implements classes and interfaces to establish the scene graph. In

extension, it also renders sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc. These are the diverse elements that sustain this precious API of JavaFX.

- **javafx.application** – This package includes a collection of classes that are responsible for the life cycle of the JavaFX application.
- **javafx.event** – It includes classes and interfaces that are used to perform and manage JavaFX events.
- **javafx.stage** – This package of JavaFX API accommodates the top-level container classes used for the JavaFX application.

Basic Structure of a JavaFX Program

- Import JavaFX Packages
- import javafx.application.Application;
- import javafx.scene.Scene;
- import javafx.scene.control.Label;
- import javafx.scene.layout.StackPane;
- import javafx.stage.Stage;
- Extend Application Class
- public class HelloJavaFX extends Application {
- Override start(Stage primaryStage) Method
- The start() method is where you set up your GUI components
- public void start(Stage primaryStage) {
- Create UI Components
- In JavaFX, UI elements are called Nodes. Let's create a simple Label
- Label label = new Label("Hello, JavaFX!");
- **Launch the Application**
- **public static void main(String[] args) {**
- **launch(args);**
- **}**

Pane

- In Java, a Pane is a type of container used in JavaFX to organize UI components. It serves as a layout manager for positioning child nodes within a GUI. There are different types of Panes in JavaFX, each with its own layout behavior.
- **Pane** – The base class that allows free placement of child nodes.
- **StackPane** – Stacks child nodes on top of each other.
- **BorderPane** – Divides the layout into five regions: Top, Bottom, Left, Right, and Center.
- **HBox** – Arranges child nodes in a horizontal row.
- **VBox** – Arranges child nodes in a vertical column.
- **GridPane** – Arranges child nodes in a flexible grid structure.
- **AnchorPane** – Allows positioning of child nodes relative to its edges.
- **FlowPane** – Arranges child nodes in a flow (wraps when necessary).

  Example:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;
public class SimplePane extends Application {
   @Override
   public void start(Stage stage) {
      Pane pane = new Pane();
      Scene scene = new Scene(pane, 300, 200);
      stage.setScene(scene);
      stage.setTitle("JavaFX Pane Example");
      stage.show();
   }
   public static void main(String[] args) {
      launch(args);
   }
}
```

JavaFX - UI Controls

- UI Controls are the graphical elements that allow users to interact with an application or a website. They include buttons, menus, sliders, text

fields, checkboxes, radio buttons, and more. In this tutorial, we will explore the different types of UI Controls of JavaFX.

Example:

```
import javafx.application.Application;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class forms extends Application {
    @Override
    public void start(Stage stage) {
        // Create controls
        TextField nameField = new TextField();
        nameField.setPromptText("Your name");
        TextArea noteArea = new TextArea();
        noteArea.setPromptText("Your notes");
        noteArea.setMaxHeight(100);
        ToggleGroup group = new ToggleGroup();
        RadioButton opt1 = new RadioButton("Option 1");
        RadioButton opt2 = new RadioButton("Option 2");
        opt1.setToggleGroup(group);
        opt2.setToggleGroup(group);
        Button submit = new Button("Submit");
        // Layout
        VBox root = new VBox(10, nameField, noteArea, opt1, opt2,
submit);
        // Show window
        stage.setScene(new Scene(root, 250, 250));
        stage.setTitle("Mini Form");
        stage.show();
    }

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

Advantages of JavaFX

- It supports CSS for styling and animating UI elements, which gives more flexibility and control over the look and feel of the application.

- It allows us to use a wide range of media formats, such as images, audio, video, and 3D graphics, which can be integrated seamlessly into the UI.
- Since it is a Java based technology, it also has built-in support for concurrency and multithreading, which enables the application to handle complex tasks without blocking the UI thread.
- JavaFX also supports binding and properties, which simplifies the communication between the UI and the business logic.

JavaFX UI Controls and Shapes

- JavaFX UI Controls are used to interact with the user.

Control	Description
Button	Clickable button
Label	Displays text
TextField	Single-line text input
TextArea	Multi-line text input
CheckBox	Toggle box for selection
RadioButton	Selectable options (one at a time)
ComboBox	Drop-down selection menu
ListView	Displays a list of items
TableView	Displays tabular data

Shape

- In general, a shape is a geometrical figure that can be drawn on the XY plane, these include Line, Rectangle, Circle, etc.
- Predefined shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Polyline, Cubic Curve, Quad Curve, Arc.
- Path elements such as MoveTO Path Element, Line, Horizontal Line, Vertical Line, Cubic Curve, Quadratic Curve, Arc.
- LineA line is a geometrical structure joining two point. The Line class of the package javafx.scene.shape represents a line in the XY plane.
- **Rectangle**

- In general, a rectangle is a four-sided polygon that has two pairs of parallel and concurrent sides with all interior angles as right angles. In JavaFX, a Rectangle is represented by a class named Rectangle. This class belongs to the package
- **Circle** A circle is a line forming a closed loop, every point on which is a fixed distance from a centre point. In JavaFX, a circle is represented by a class named Circle. This class belongs to the package javafx.scene.shape.
- **Example:**

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class CircleAndSquare extends Application {
    @Override
    public void start(Stage stage) {
        Pane pane = new Pane();
        Circle circle = new Circle(100, 100, 50); // Center at (100,100), Radius = 50
        circle.setFill(Color.BLUE); // Fill color
        circle.setStroke(Color.RED); // Border color
        circle.setStrokeWidth(5);
        // Create a Square (Rectangle with equal width and height)
        Rectangle square = new Rectangle(200, 80, 100, 100); // Top-left at (200,80), Size 100x100
        square.setFill(Color.RED);
        square.setStroke(Color.BLACK);
        square.setStrokeWidth(2);
        // Add shapes to the pane
        pane.getChildren().addAll(square,circle);
        // Scene setup
        Scene scene = new Scene(pane, 400, 300);
        stage.setScene(scene);
        stage.setTitle("JavaFX Circle and Square");
        stage.show();
    }

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

```
        launch(args);
    }
}
```

Property Binding in JavaFX

- Property binding in JavaFX is a powerful mechanism that allows you to create relationships between properties so that when one property changes, the other is automatically updated. This is a key feature of JavaFX's observable properties system.
- **Basic Concepts**
- JavaFX properties are part of the JavaFX Beans specification and provide:
- **Change notifications**
- **Invalidation notifications**
- **Binding capabilities**

Types of Binding

1. Unidirectional Binding

Creates a one-way relationship where the target property updates when the source property changes.

```
SimpleStringProperty name = new
SimpleStringProperty("John");
Label nameLabel = new Label();
nameLabel.textProperty().bind(name); // Label updates when
name changes
// Changing the name will update the label
name.set("Jane");
```

2. Bidirectional Binding

Creates a two-way relationship where changes to either property update the other.

```
TextField textField1 = new TextField();
TextField textField2 = new TextField();

textField1.textProperty().bindBidirectional(textField2.textPro
perty());

// Now changing either text field will update the other
textField1.setText("Hello");
// textField2 now also has "Hello"
```

Java FX Colors and Font

- JavaFX allows you to customize the appearance of UI components using:
- Colors (for text, backgrounds, borders)
- Fonts (family, size, weight, style)
- CSS (for advanced styling)
- JavaFX provides the javafx.scene.paint.Color class to define and apply colors.
- import javafx.scene.paint.Color;
- Color red = Color.RED;
- Color lightBlue = Color.LIGHTBLUE;
- Color customColor1 = new Color(0.2, 0.4, 0.8, 1.0); // R, G, B, Alpha
- Color hexColor1 = Color.web("#FF5733"); // Orange-red

- The javafx.scene.text.Font class allows you to customize text appearance.

- Family: "Arial", "Verdana", etc.
- Size: 12, 14, 16, etc.
- Weight: FontWeight.NORMAL, FontWeight.BOLD
- Posture: FontPosture.REGULAR, FontPosture.ITALIC
- Font font1 = Font.font("Arial", 16);
- // Bold font
- Font font2 = Font.font("Verdana", FontWeight.BOLD, 18);
- // Italic font
- Font font3 = Font.font("Times New Roman", FontPosture.ITALIC, 14);
- // Bold + Italic
- Font font4 = Font.font("Courier New", FontWeight.BOLD, FontPosture.ITALIC, 20);

JavaFX Images

- JavaFX provides two main classes for image handling:
- Image: Loads an image file into memory
- ImageView: Displays the loaded image in your UI
- import javafx.scene.image.Image;
- // Load from local file (absolute path)
- Image localImage = new Image("file:/C:/images/photo.jpg");
- // Load from relative path (project folder)
- Image relativeImage = new Image("images/photo.png");
- Events in JavaFX

- Action Events:
- ActionEvent: Button clicks, menu selections
- Change Events:
- ListChangeEvent: Changes in observable lists
- MapChangeEvent: Changes in observable maps
- Drag-and-Drop Events:
- DragEvent: Drag detection, drag over, drop

Layout Panes and Shapes

- This tutorial covers two essential JavaFX topics: Layout Panes for organizing UI components and Shapes for drawing graphics.

1. JavaFX Layout Panes

HBox (Horizontal Box)

Arranges nodes horizontally in a single row.

```
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;

HBox hbox = new HBox(10); // 10px spacing
hbox.getChildren().addAll(
new Button("Button 1"),
new Button("Button 2"),
new Button("Button 3")
);
```

VBox (Vertical Box)

Arranges nodes vertically in a single column.

```
import javafx.scene.layout.VBox;

VBox vbox = new VBox(10); // 10px spacing
vbox.getChildren().addAll(
    new Button("Top"),
    new Button("Middle"),
    new Button("Bottom")
);
```

BorderPane

Divides layout into 5 regions: top, bottom, left, right, center.

```
import javafx.scene.layout.BorderPane;
BorderPane borderPane = new BorderPane();
borderPane.setTop(new Button("Top"));
borderPane.setLeft(new Button("Left"));
borderPane.setCenter(new Button("Center"));
```

```
borderPane.setRight(new Button("Right"));
borderPane.setBottom(new Button("Bottom"));
```

Events and Events Source in JavaFX

- Events represent user interactions or system notifications. JavaFX uses an event-driven programming model where
- Event Source - The object that generates the event (e.g., Button)
- Event Target - The node that receives the event
- Event Handler - Code that executes in response to the event

Event Class	Description	Common Sources
`ActionEvent`	Button clicks, menu selections	`Button`, `MenuItem`
`MouseEvent`	Mouse actions (click, move, drag)	Any `Node`
`KeyEvent`	Keyboard input	`Scene`, text controls
`WindowEvent`	Window state changes	`Stage`
`DragEvent`	Drag-and-drop operations	Any `Node`

Event Registration and Handling in JavaFX

- What are Events?
- Events represent of interest in a JavaFX application:
- User actions (mouse clicks, key presses)
- Window operations (showing, hiding)
- State changes (selection changes, value updates)
- Event Handling Model

Event Registration and Handling in JavaFX

- Event Sources
- Objects that generate events (buttons, text fields, windows)
- Event Objects
- Contain information about the event (type, source, position, etc.)
- Event Handlers
- Code that responds to events (your implementation)
- 3. Event Processing Phases
- JavaFX processes events in three phases:

- JavaFX processes events in three phases:
- Capture Phase (Top-down):
- Events travel from the Stage down to the target node
- Event filters are executed in this phase
- Target Phase:
- Events reach the target node
- Default handlers are executed
- Bubbling Phase (Bottom-up):
- Events travel back up to the Stage
- Normal event handlers are executed
-

inner classes, anonymous inner classes

- In JavaFX, UI components like Button, TextField, etc., fire events when users interact with them. You respond to these events by using event handlers
- In Java, an inner class is a class defined inside another class. It has access to the outer class's fields and methods

Anonymous Inner Class

- An anonymous inner class is a class without a name, defined and instantiated in one line. It's typically used to implement an interface or extend a class for a one-time use — especially in UI event handling.
- In JavaFX, we use it a lot with interfaces like:
- EventHandler<ActionEvent>
- ChangeListener<T>
- Runnable (for threading)
  import javafx.application.Application;
  import javafx.scene.Scene;
  import javafx.scene.control.Button;
  import javafx.scene.layout.StackPane;
  import javafx.stage.Stage;
  Example:

```
public class AnonymousInnerClassExample extends Application {

    @Override
    public void start(Stage stage) {
        Button button = new Button("Click Me");

        // Anonymous inner class to handle click
```

```
        button.setOnAction(new
javafx.event.EventHandler<javafx.event.ActionEvent>() {
            @Override
            public void handle(javafx.event.ActionEvent event) {
                System.out.println("Button clicked!");
            }
        });

        StackPane root = new StackPane(button);
        Scene scene = new Scene(root, 200, 100);
        stage.setScene(scene);
        stage.setTitle("Anonymous Class Example");
        stage.show();
    }

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

Listeners for Observable objects

- In JavaFX, many classes are observable, meaning they can notify listeners when their state and value changes.
- These include:
- **Properties (like StringProperty, DoubleProperty, etc.)**
- **Collections (ObservableList, ObservableMap, ObservableSet)**
- **Bindings (for syncing values)**
- **And even UI components (like TextField.textProperty())**

ChangeListener Example (Value change)

Example:

```
StringProperty name1 = new SimpleStringProperty("One");
StringProperty name2 = new SimpleStringProperty("Two");

// Bind both ways
name1.bindBidirectional(name2);

// Both now change together
name1.set("Updated");
System.out.println(name2.get()); // Updated
```

Mouse and Key Events in JavaFX

- JavaFX provides comprehensive support for handling mouse interactions. Here are the main mouse event types and their handling
- MOUSE_CLICKED - Occurs when mouse is clicked (pressed and released)
- MOUSE_PRESSED - When mouse button is pressed down
- MOUSE_RELEASED - When mouse button is released
- MOUSE_MOVED - When mouse moves without buttons pressed
- MOUSE_DRAGGED - When mouse moves with buttons pressed
- MOUSE_ENTERED - When mouse enters a node's bounds
- MOUSE_EXITED - When mouse exits a node's bounds

Example:

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

public class MouseKeyEventExample extends Application {

    @Override
    public void start(Stage stage) {
        Label label = new Label("Click or press a key");
        StackPane root = new StackPane(label);
        Scene scene = new Scene(root, 300, 150);
        // 🔷 Mouse Event
        scene.setOnMouseClicked(event -> {
            label.setText("Mouse clicked at: " + event.getX() + ", " +
event.getY());
        });
        // 🔷 Key Event
        scene.setOnKeyPressed(event -> {
            label.setText("Key pressed: " + event.getCode());
        });
        stage.setTitle("Mouse & Key Event");
        stage.setScene(scene);
        stage.show();
        // 🔷 Request focus to receive key events
        root.requestFocus();
    }
    public static void main(String[] args) {
        launch(args);
    }
```

}

animation

In JavaFX, animation means making things move, rotate, fade, or grow automatically — like a cartoon!
Like
Button click animation?
Fading text?
Rotating shapes?

Term	Meaning
`TranslateTransition`	Moves a shape
`Duration.seconds(2)`	Takes 2 seconds to complete
`setByX(200)`	Move right by 200 pixels
`setCycleCount(...)`	Repeat the animation
`setAutoReverse(true)`	Move back after moving forward
`play()`	Start the animation

Example:

```
import javafx.animation.TranslateTransition;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;
import javafx.util.Duration;

public class TinyAnimation extends Application {
    public void start(Stage stage) {
        Circle circle = new Circle(50, 0, 20); // x=50, y=100, radius=20

        // Move right by 150px in 1 second
        TranslateTransition move = new
TranslateTransition(Duration.seconds(2), circle);
        move.setByY(100);
```

```
        move.play(); // start animation

        Pane root = new Pane(circle);
        stage.setScene(new Scene(root, 300, 200));
        stage.setTitle("Tiny Animation");
        stage.show();
    }

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