Java Fx

What is JavaFX?

- JavaFX is a Java library used to build Rich Internet Applications.
- The applications written using this library can run consistently across multiple platforms.
- 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 Tool kit and Swing.
- After the advent of JavaFX, these Java programmers can now develop GUI applications effectively with rich content.

Need of JavaFX

- **All-in-One Library**: JavaFX combines various features (media, UI controls, 2D/3D graphics) into a single library.
- **Integration with Java Libraries**: Easily works with existing Java libraries like Swing, providing flexibility to developers.
- **Hardware Acceleration**: Takes advantage of the computer's graphics hardware for smoother and faster visuals.
- **JVM Compatibility**: Works seamlessly with Java, Groovy, and JRuby, so developers can stick to what they know.
- **Rich Graphics and Media**: Provides powerful tools for creating visually appealing interfaces and handling multimedia.
- **Animation and UI Control**: Allows developers to combine graphics animation with user interface controls for interactive apps.
- **No Additional Learning Curve**: If you know Java, Groovy, or JRuby, you're good to go—no need to learn extra technologies.

Features of JavaFX

- Language Compatibility: JavaFX is written in Java and works with Java, Groovy, and JRuby, making it versatile and platform-independent.
- FXML for UI: FXML, similar to HTML, is used for defining user interfaces in a straightforward way.
- Scene Builder: Scene Builder is a drag-and-drop tool that makes it easy to create user interfaces in JavaFX.
- Swing Integration: You can combine JavaFX with Swing, updating existing Swing applications with modern features.
- Built-in UI Controls: JavaFX comes with a variety of ready-to-use user interface controls for building applications.

- Canvas and Printing: JavaFX offers tools like Canvas for drawing and a Printing API for handling print-related tasks.
- Rich APIs: The library provides a comprehensive set of APIs for GUI, graphics, and more. It supports Java features like Generics and Multithreading.
- Integrated Graphics: JavaFX includes built-in classes for creating both 2D and 3D graphics.

JavaFX vs Swing

Swing	JavaFX
1. Swing is standard toolkit for java developers to develop GUI	1, JavaFX provides platform support for creating desktop applications
2. Swing has more complex set of GUI components	2. JavaFX has less components available than swing
3. Swing doesn’t have support of working with CSS and XML	3. JavaFX has support for customizing using CSS and XML
4. Creating 3D applications is hard using Swing	4. Creating 3D applications is easy with JavaFX
5. Swing is used for creating desktop based applications	5. JavaFX is used in various platforms like web development, mobile applications and desktop applications
6. Swing UI library act as legacy	6. JavaFX componenets are built on top of

Writing JavaFX program

Step 1: Extend javafx.application.Application and override start()

- ➔ “start()” method of Application class is overridden to defiene UI components and behavior
- ➔ Object of class javafx.stage.Stage is passed into the start() method

Step 2: Create a button

- ➔ We create a button by creating object of javafx.scene.control.Button
- ➔ Eg. Button myButton = new Button("Click Me!");

Step 3: Create a layout and add button to it

- ➔ JavaFX provides number of layouts like BorderPane, GridPane, etc.
- ➔ It exist on top of scene graph and and can be seen as root node
- ➔ All other nodes (buttons, text, etc.) need to be added to this layout
- ➔ As an example StackPane layout is used

Step 4: Create a Scene

- ➔ A layout should be added to a scene
- ➔ We need to pass the layout object to the scene class constructor
- ➔ Scene remains at higher level in hierarchy of application structure

Step 5: Prepare the stage

- ➔ javafx.stage.Stage provides the methods to set attributes for stage
- ➔ Like setting the title or calling the show() method

Step 6: Create an event for button

- ➔ We call setOnAction() on button and we pass Event Handler class as parameter to method
- ➔ Event Handler is an anonymous class which must define handle() method for handling event

Step 7: Create a main method

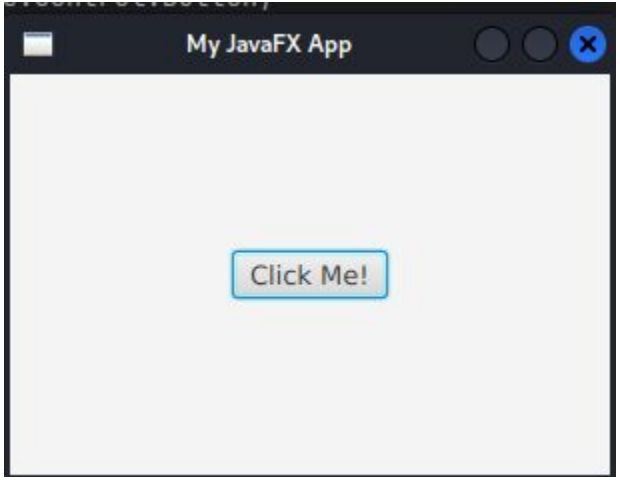
- ➔ At last we need to create a main method
- ➔ We need to call lauch(args) method and can pass command line args to it
- ➔ Launch method launches javafx application

```
1 import javafx.application.Application;
2 import javafx.scene.Scene;
3 import javafx.scene.control.Button;
4 import javafx.scene.layout.StackPane;
5 import javafx.stage.Stage;
```

```java
public class MyJavaFXApp extends Application {
    @Override
    public void start(Stage primaryStage) {

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

```java
@Override
    public void start(Stage primaryStage) {
        // Step 2: Create a button
        Button myButton = new Button("Click Me!");

        // Step 6: Create an event for the button
        myButton.setOnAction(e -> {
            System.out.println("Button Clicked!");
        });

        // Step 3: Create a layout (StackPane) and add the button to it
        StackPane stackPane = new StackPane();
        stackPane.getChildren().add(myButton);

        // Step 4: Create a Scene with the layout as its root node
        Scene scene = new Scene(stackPane, 300, 200);

        // Step 5: Prepare the stage
        primaryStage.setScene(scene);
        primaryStage.setTitle("My JavaFX App"); // Set the title for the window

        // Step 7: Create a main method
        primaryStage.show(); // Display the window
    }
```

Output:




```
/home/kali/.jdks/openjdk-21.0.2/bin/java ...
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Button Clicked!

Process finished with exit code 0
```

JavaFX Layouts

JavaFX provides a variety of layout panes that allow you to organize and arrange UI components within your application. Here's an overview of some common layout panes in JavaFX:

FlowPane:

Arranges its children in a flow, wrapping to the next row or column as needed.

FlowPane flowPane = new FlowPane();

HBox (Horizontal Box):

Arranges its children in a single horizontal row.

HBox hBox = new HBox();

VBox (Vertical Box):

Arranges its children in a single vertical column.

VBox vBox = new VBox();

BorderPane:

Divides the layout into five regions: top, bottom, left, right, and center.

BorderPane borderPane = new BorderPane();

GridPane:

A flexible grid-based layout where you can specify rows and columns to place components.

GridPane gridPane = new GridPane();

FlowPane

The FlowPane is a layout pane in JavaFX that arranges its children in a flow, either horizontally or vertically, wrapping to the next row or column as needed. It's useful for scenarios where you want a dynamic arrangement of components that adapts to the available space.

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

public class FlowPaneExample extends Application {

    @Override
    public void start(Stage primaryStage) {
```

```
    // Creating buttons
    Button button1 = new Button("Button 1");
    Button button2 = new Button("Button 2");
    Button button3 = new Button("Button 3");
    Button button4 = new Button("Button 4");
    Button button5 = new Button("Button 5");

    // Creating a FlowPane and adding buttons to it
    FlowPane flowPane = new FlowPane();
    flowPane.getChildren().addAll(button1, button2, button3, button4, button5);

    Scene scene = new Scene(flowPane, 300, 200);

    // Setting the scene for the stage
    primaryStage.setScene(scene);

    primaryStage.setTitle("FlowPane Example");

    primaryStage.show();
}

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

HBox

The HBox (Horizontal Box) in JavaFX is a layout pane that arranges its children in a single horizontal row. Here is an example demonstrating the usage of HBox:

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

public class HBoxExample extends Application {

    @Override
    public void start(Stage primaryStage) {
```

```
        // Create buttons
        Button button1 = new Button("Button 1");
        Button button2 = new Button("Button 2");
        Button button3 = new Button("Button 3");

        // Create an HBox and add buttons to it
        HBox hBox = new HBox();
        hBox.getChildren().addAll(button1, button2, button3);

        // Create a Scene with the HBox as its root
        Scene scene = new Scene(hBox, 300, 100);

        // Set the scene for the stage
        primaryStage.setScene(scene);

        primaryStage.setTitle("HBox Example");

        primaryStage.show();
    }

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

VBox

The VBox (Vertical Box) in JavaFX is a layout pane that arranges its children in a single vertical column. Here's an example demonstrating the usage of VBox:

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

public class VBoxExample extends Application {

    @Override
    public void start(Stage primaryStage) {
```

```
        // Create buttons
        Button button1 = new Button("Button 1");
        Button button2 = new Button("Button 2");
        Button button3 = new Button("Button 3");

        // Create a VBox and add buttons to it
        VBox vBox = new VBox();
        vBox.getChildren().addAll(button1, button2, button3);

        // Create a Scene with the VBox as its root
        Scene scene = new Scene(vBox, 100, 300);

        // Set the scene for the stage
        primaryStage.setScene(scene);

        primaryStage.setTitle("VBox Example");

        primaryStage.show();
    }

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

BorderPane

The BorderPane in JavaFX is a layout pane that divides the space into five regions: top, bottom, left, right, and center. Here's an example demonstrating the usage of BorderPane:

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

public class BorderPaneExample extends Application {

    @Override
    public void start(Stage primaryStage) {
```

```
// Create buttons for each region
Button topButton = new Button("Top");
Button bottomButton = new Button("Bottom");
Button leftButton = new Button("Left");
Button rightButton = new Button("Right");
Button centerButton = new Button("Center");

// Create a BorderPane and set buttons to its regions
BorderPane borderPane = new BorderPane();
borderPane.setTop(topButton);
borderPane.setBottom(bottomButton);
borderPane.setLeft(leftButton);
borderPane.setRight(rightButton);
borderPane.setCenter(centerButton);

// Create a Scene with the BorderPane as its root
Scene scene = new Scene(borderPane, 300, 200);

// Set the scene for the stage
primaryStage.setScene(scene);

primaryStage.setTitle("BorderPane Example");
```

```
        primaryStage.show();
    }

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

GridPane

The GridPane in JavaFX is a layout pane that allows you to create a flexible grid of rows and columns for arranging UI components. Here's an example demonstrating the usage of GridPane:

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

public class GridPaneExample extends Application {

    @Override
    public void start(Stage primaryStage) {
```

```
// Create buttons
Button button1 = new Button("Button 1");
Button button2 = new Button("Button 2");
Button button3 = new Button("Button 3");
Button button4 = new Button("Button 4");

// Create a GridPane and add buttons to specific positions
GridPane gridPane = new GridPane();
gridPane.add(button1, 0, 0); // column 0, row 0
gridPane.add(button2, 1, 0); // column 1, row 0
gridPane.add(button3, 0, 1); // column 0, row 1
gridPane.add(button4, 1, 1); // column 1, row 1

// Set spacing between cells
gridPane.setHgap(10); // horizontal gap
gridPane.setVgap(10); // vertical gap

// Create a Scene with the GridPane as its root
Scene scene = new Scene(gridPane, 300, 200);
```

```
        // Set the scene for the stage
        primaryStage.setScene(scene);

        primaryStage.setTitle("GridPane Example");

        primaryStage.show();
    }

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

JavaFX UI Controls

JavaFX provides a rich set of UI controls that allow you to create interactive and user-friendly graphical user interfaces. Here's an overview of some commonly used JavaFX UI controls:

Button:

Represents a button that can trigger actions when clicked.

Button button = new Button("Click Me");

Label:

Displays a non-editable text.

Label label = new Label("Hello, JavaFX!");

TextField:

Allows the user to enter and edit single-line text.

TextField textField = new TextField();

TextArea:

Allows the user to enter and edit multi-line text.

TextArea textArea = new TextArea();

Label

The Label class in JavaFX is used to create a non-editable text element within a graphical user interface. It allows developers to display static text or provide informational messages to users.

Construcrors of Label are:

Label label = new Label(); // empty label

Label label = new Label("Hello, JavaFX!"); // label with text

Label label = new Label("Hello, JavaFX!", graphicNode); // label with graphic node like icon

TextField

The TextField class in JavaFX is used to create a text input control that allows users to enter and edit single-line text. It's a common UI element for accepting textual input in various forms such as names, passwords, or search queries. Here are some key features and commonly used constructors for the TextField class:

TextField textField = new TextField();

TextField textField = new TextField("Initial Text")

PasswordField

The PasswordField class in JavaFX is similar to the TextField class, but it is specifically designed for password input. It provides a secure way to handle password entry by displaying the characters as dots or asterisks instead of the actual characters. Here are some key features and commonly used constructors for the PasswordField class:

```
PasswordField passwordField = new PasswordField();
```

Button

The Button class in JavaFX represents a button that users can interact with by clicking. It is one of the most commonly used UI controls for triggering actions in response to user input. Here are some key features and commonly used constructors for the Button class:

```
Button button = new Button();

Button button = new Button("Click Me");

Button button = new Button("Click Me", graphicNode);
```

RadioButton

The RadioButton class in JavaFX is a UI control that represents a radio button. Radio buttons are used when you want users to make a single selection from a set of options. Here are some key features and commonly used constructors for the RadioButton class:

RadioButton radioButton = new RadioButton();

RadioButton radioButton = new RadioButton("Option 1");

RadioButton radioButton = new RadioButton("Option 1", graphicNode);

CheckBox

The CheckBox class in JavaFX is a UI control that represents a checkbox. Checkboxes are used when you want users to make multiple independent selections from a set of options. Here are some key features and commonly used constructors for the CheckBox class:

```
CheckBox checkBox = new CheckBox();

CheckBox checkBox = new CheckBox("Check 1");

CheckBox checkBox = new CheckBox("Enable Feature", graphicNode);
```

Hyperlink

The Hyperlink class in JavaFX is a UI control that represents a hyperlink, allowing users to navigate to a URL or perform some action when clicked. Here are some key features and commonly used constructors for the Hyperlink class:

Hyperlink hyperlink = new Hyperlink();

Hyperlink hyperlink = new Hyperlink("Visit our website");

Hyperlink hyperlink = new Hyperlink("Visit our website", graphicNode);

Menu

In JavaFX, the Menu class is part of a set of classes that are used to create menu bars and menu items in graphical user interfaces. The Menu class represents a menu that can contain menu items or submenus. Here are some key features and commonly used constructors for the Menu class:

Menu menu = new Menu();

Menu menu = new Menu("File");

Menu Class: The Menu class represents a menu that can contain menu items or submenus. It is part of the hierarchical menu structure in JavaFX.

Menu menu = new Menu("File");

MenuBar Class: The MenuBar class represents a horizontal bar of menus. It is typically placed at the top of a window or application and contains instances of the Menu class.

MenuBar menuBar = new MenuBar();

MenuItem Class: The MenuItem class represents an item within a menu. It can be used to perform actions when clicked.

MenuItem menuItem = new MenuItem("Open");

Tooltip

In JavaFX, the Tooltip class is used to create a small pop-up window that provides supplementary information about a GUI component when the user hovers over it. Here are some key features and commonly used constructors for the Tooltip class:

```
Tooltip tooltip = new Tooltip("This is a tooltip");
```

FileChooser

In JavaFX, the FileChooser class is used to create a file dialog that allows users to interact with the file system, choose files or directories, and perform file-related operations. It provides a convenient way for users to select files or specify file paths within a JavaFX application. The FileChooser class provides two types of methods:

- showOpenDialog()
- showSaveDialog()