JAVA FX

In a JavaFX application, the structure of the app window is primarily defined by several key components. Here's an overview of the main parts of a typical JavaFX application window:

Key Components of a JavaFX Application Window

Stage:

- The top-level container representing the window itself. It's similar to a frame in Swing.
- You can think of it as the main window where all the content is displayed.
- It can be configured with properties like title, width, height, and modality.

Scene: A container for all content in a specific stage.

- A stage can contain only one scene at a time, but you can switch scenes during the application's lifecycle.
- The scene can hold various UI components like buttons, labels, and layouts.

Layout Pane:

- A container that arranges the UI components in a specific layout.

Common layout panes include:

- **VBox**: Arranges children in a vertical column.
- **HBox**: Arranges children in a horizontal row.
- **GridPane**: Arranges children in a grid.
- **BorderPane**: Divides the space into five regions (top, bottom, left, right, center).
- **StackPane**: Stacks children on top of each other.
- **UI Controls**: The individual components you use to create the user interface.
- Examples include buttons, labels, text fields, checkboxes, and menus.

Example

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

public class MyJavaFXApp extends Application {
    @Override
```

```
    public void start(Stage primaryStage) {
        // Create UI components
        Label label = new Label("Hello, JavaFX!");
        Button button = new Button("Click Me");

        // Set button action
        button.setOnAction(event -> {
            label.setText("Button Clicked!");
        });

        // Create a layout pane
        VBox vbox = new VBox(10); // 10 is the spacing between elements
        vbox.getChildren().addAll(label, button);

        // Create a scene
        Scene scene = new Scene(vbox, 300, 200);

        // Set the scene on the stage
        primaryStage.setTitle("JavaFX Application");
        primaryStage.setScene(scene);
        primaryStage.show(); // Display the window
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Stage: The primaryStage is created and configured with a title.

Scene: A Scene is created with a VBox layout containing a Label and a Button.

UI Controls: The Label displays a message, and the Button has an action that changes the label's text when clicked.

Layout: The VBox organizes the label and button vertically with a spacing of 10 pixels.

Display text and Image

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

public class TextAndImageExample extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Create a label for the text
        Label label = new Label("Welcome to JavaFX!");

        // Load an image
        Image image = new Image("file:src/resources/image.png"); // Path to your image
        ImageView imageView = new ImageView(image);

        // Set the desired size of the image
        imageView.setFitWidth(200);
        imageView.setFitHeight(150);
        imageView.setPreserveRatio(true); // Preserve the aspect ratio

        // Create a layout pane
        VBox vbox = new VBox(10); // Spacing between elements
        vbox.getChildren().addAll(label, imageView);

        // Create a scene
        Scene scene = new Scene(vbox, 300, 250);
```

```
        // Set the scene on the stage
        primaryStage.setTitle("Display Text and Image");
        primaryStage.setScene(scene);
        primaryStage.show(); // Display the window
    }

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

 Label for Text:

- A Label is created to display the text "Welcome to JavaFX!".

 Loading an Image:

- An Image object is created by providing a file path. Make sure the path is correct, and the image file exists.
- An ImageView is used to display the image.

 Setting Image Properties:

- You can set the width and height of the ImageView using setFitWidth() and setFitHeight().
- setPreserveRatio(true) maintains the aspect ratio of the image.

 VBox Layout:

- A VBox layout pane arranges the label and image vertically, with a spacing of 10 pixels between them.

 Creating a Scene:

- A Scene is created with the layout, and the dimensions are set.

 Stage Setup:

- The stage is titled and the scene is set before displaying the window.

Event Handling

Event handling in JavaFX is a core concept that allows you to create interactive applications. By responding to user actions such as mouse clicks, key presses, and other events, you can control the behavior of your application.

Overview of Event Handling

Event Sources: UI components that generate events (e.g., buttons, text fields).

Event Types: Different types of events, such as ActionEvent, MouseEvent, KeyEvent, etc.

Event Handlers: Methods that define what happens in response to an event.

Basic Steps for Event Handling

Create UI Components: Define the components that will trigger events.

Attach Event Handlers: Use methods to attach event handlers to these components.

Define Event Logic: Implement the logic that will execute when the event occurs.

Mouse Event Handling

In JavaFX, mouse events allow you to respond to various mouse actions, such as clicks, movements, and button presses. You can handle these events by attaching event handlers to your UI components.

Common Mouse Events

Mouse Clicks: Triggered when the mouse button is clicked.

setOnMouseClicked()

Mouse Enter: Triggered when the mouse pointer enters the component area.

setOnMouseEntered()

Mouse Exit: Triggered when the mouse pointer exits the component area.

setOnMouseExited()

Mouse Press: Triggered when a mouse button is pressed down.

setOnMousePressed()

Mouse Release: Triggered when a mouse button is released.

setOnMouseReleased()

Mouse Dragged: Triggered when the mouse is dragged while a button is pressed.

setOnMouseDragged()

Example

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.stage.Stage;

public class MouseEventExample extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Create a button
        Button button = new Button("Hover or Click Me");

        // Create a text to display messages
        Text message = new Text("Mouse Events Demo");

        // Handle mouse entered event
        button.setOnMouseEntered(event -> {
            button.setStyle("-fx-background-color: lightblue;");
            message.setText("Mouse Entered!");
        });

        // Handle mouse exited event
        button.setOnMouseExited(event -> {
            button.setStyle("");
            message.setText("Mouse Exited!");
        });

        // Handle mouse clicked event
        button.setOnMouseClicked(event -> {
```

```
            message.setText("Button Clicked!");
        });

        // Handle mouse pressed event
        button.setOnMousePressed(event -> {
            message.setFill(Color.RED);
            message.setText("Mouse Pressed!");
        });

        // Handle mouse released event
        button.setOnMouseReleased(event -> {
            message.setFill(Color.BLACK);
            message.setText("Mouse Released!");
        });

        // Create a layout pane
        VBox vbox = new VBox(10);
        vbox.getChildren().addAll(button, message);

        // Create a scene
        Scene scene = new Scene(vbox, 300, 200);

        // Set the scene on the stage
        primaryStage.setTitle("Mouse Events Example");
        primaryStage.setScene(scene);
        primaryStage.show(); // Display the window
    }

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

Laying out nodes in scene graph

In JavaFX, laying out nodes in the scene graph involves arranging your UI components (nodes) in a structured way using layout panes. The scene graph is a hierarchical tree structure that represents the visual elements of your application, where each node can be a UI component or a layout container.

Common Layout Panes

1. **VBox**: Stacks nodes vertically.
2. **HBox**: Stacks nodes horizontally.
3. **GridPane**: Arranges nodes in a grid format (rows and columns).
4. **BorderPane**: Divides the area into five regions (top, bottom, left, right, center).
5. **StackPane**: Stacks nodes on top of each other.
6. **FlowPane**: Arranges nodes in a flow, wrapping them as needed.