JAVAFX
BorderPane and GridPane

ABSTRACT
In this document, we will explore two essential layout managers in
JavaFX: BorderPane and GridPane.In JavaFX, a layout manager is a class that arranges the visual components of a JavaFX application. Each layout manager serves a unique purpose in organizing user interface components, detailed explanations, code examples, and references for those layout managers in JavaFX: BorderPane and GridPane. The goal is to understand how to utilize these layout managers effectively in JavaFX applications.

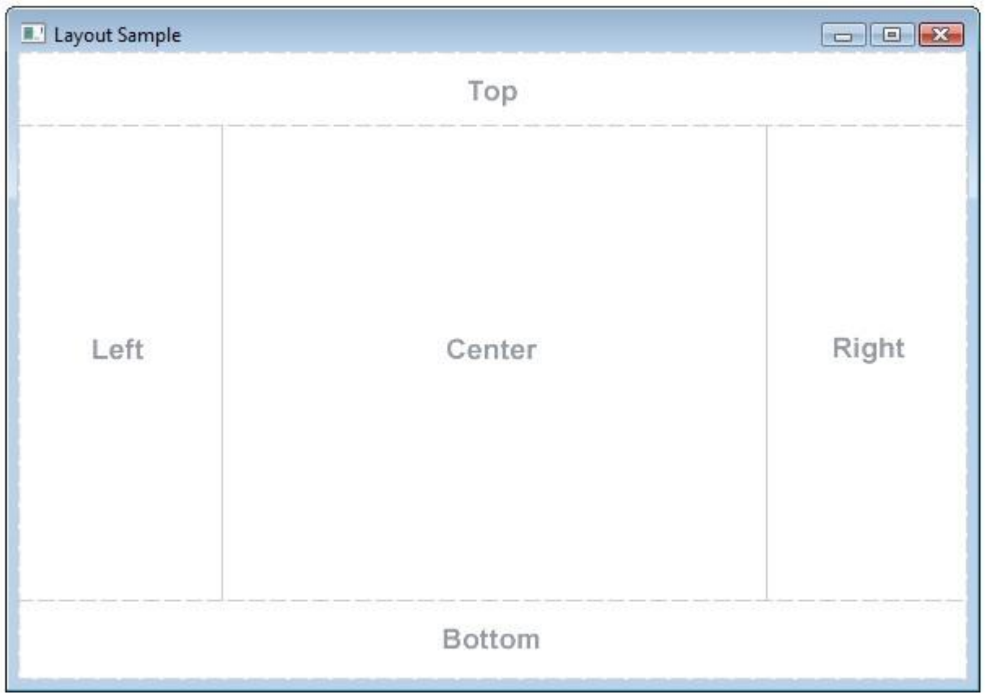
Steve Stylin
Java for Programmers

Introduction:

JavaFX is a versatile suite of graphics and media packages that empowers developers to seamlessly design, create, test, debug, and deploy rich client applications. Its constant performance across a vast range of platforms makes JavaFX a critical tool for delivering superb user experiences. With a spark of creativity, developers can build the most intuitive and engaging Java applications. Key elements in this process include the powerful JavaFX BorderPane and GridPane, which help shape the user interface into something truly remarkable. As a developer, use JavaFX's rich set of UI controls and layout options to craft an engaging and user-friendly application where each element enhances the overall experience, just as spices elevate a meal. Those Ingredients are JavaFX BorderPane and JavaFX GridPane (Oracle, Java Documentation, n.d.).

JavaFX BorderPane

BorderPane is a layout manager that arranges its children in five distinct areas: top, bottom, left, right, and center. This layout, with its straightforward and comfortable usage, is handy for applications that require a structured interface, such as a typical desktop application with a menu bar, toolbars, and a main content area. The simplicity of using BorderPane makes it a comfy choice for developers, ensuring a sense of ease and confidence in their UI design tasks (Oracle, JavaFX BorderPane, n.d.).



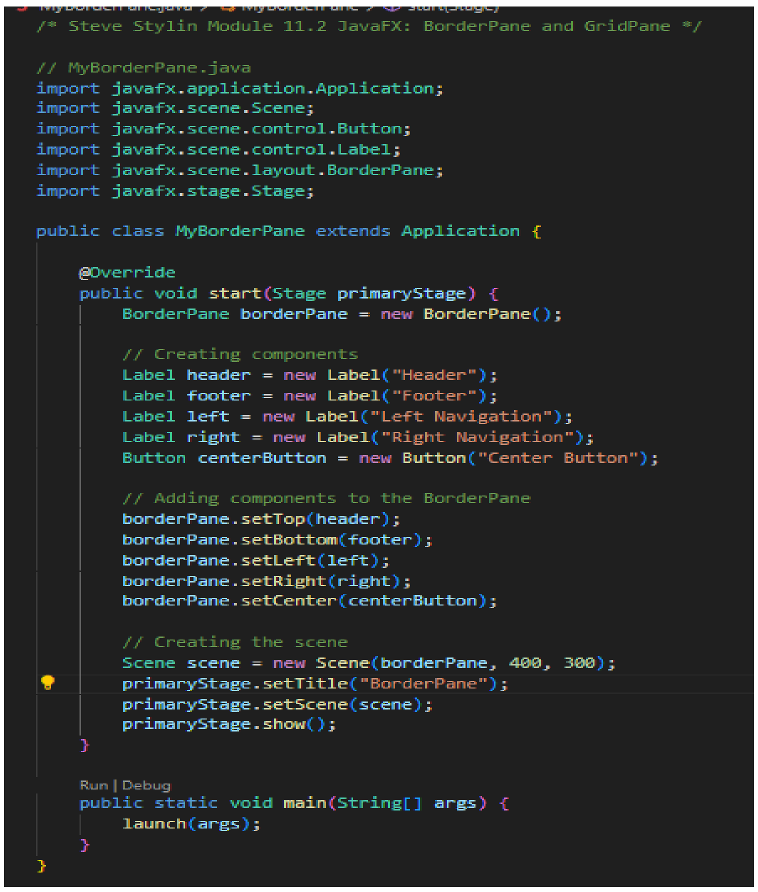
Key Features of BorderPane:

- ✓ Five Regions: BorderPane divides the interface into five carefully defined regions: top, bottom, left, right, and center, allowing for an organized and systematic arrangement of components.
- ✓ Dynamic Flexible Sizing: The center region is designed to expand and adapt to fill any available space, ensuring that it remains the focal point of the layout. Meanwhile, the

other regions can be resized based on their content, providing versatility and a user-friendly experience.

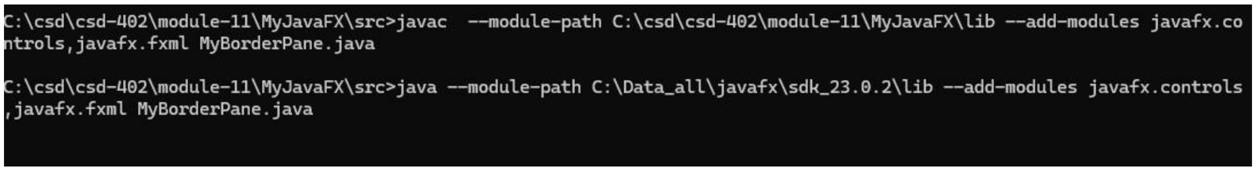
- ✓ User-friendly Organization: BorderPane offers a simple yet effective approach to arranging user interface elements. Its intuitive structure allows developers to position components easily, eliminating the need for complex calculations and enhancing workflow efficiency.

Using a BorderPane in JavaFX helps organize the application interface. You put elements in designated areas for different activities, keeping user preferences in mind. Things make sense with BorderPane; it allows you to assign components neatly to specific regions, creating a harmonious and efficient workspace for your application. The center region is like the heart of the room, with everything else surrounding it, ensuring that each area serves its purpose without cluttering the space.



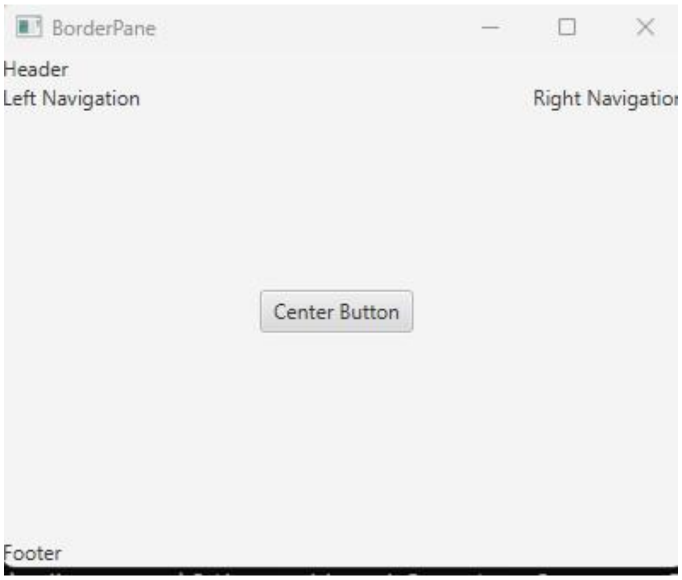
```java
/* Steve Stylin Module 11.2 JavaFX: BorderPane and GridPane */

// MyBorderPane.java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class MyBorderPane extends Application {

    @Override
    public void start(Stage primaryStage) {
        BorderPane borderPane = new BorderPane();

        // Creating components
        Label header = new Label("Header");
        Label footer = new Label("Footer");
        Label left = new Label("Left Navigation");
        Label right = new Label("Right Navigation");
        Button centerButton = new Button("Center Button");

        // Adding components to the BorderPane
        borderPane.setTop(header);
        borderPane.setBottom(footer);
        borderPane.setLeft(left);
        borderPane.setRight(right);
        borderPane.setCenter(centerButton);

        // Creating the scene
        Scene scene = new Scene(borderPane, 400, 300);
        primaryStage.setTitle("BorderPane");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

The start method initializes the layout and adds various components to the respective regions. The application includes a header, footer, side navigation, and a main content area. The Scene is then set on the primary stage, which is displayed to the user.



```
C:\csd\csd-402\module-11\MyJavaFX\src>javac  --module-path C:\csd\csd-402\module-11\MyJavaFX\lib --add-modules javafx.co
ntrols,javafx.fxml MyBorderPane.java

C:\csd\csd-402\module-11\MyJavaFX\src>java --module-path C:\Data_all\javafx\sdk_23.0.2\lib --add-modules javafx.controls
,javafx.fxml MyBorderPane.java
```

The output:



JavaFX GridPane

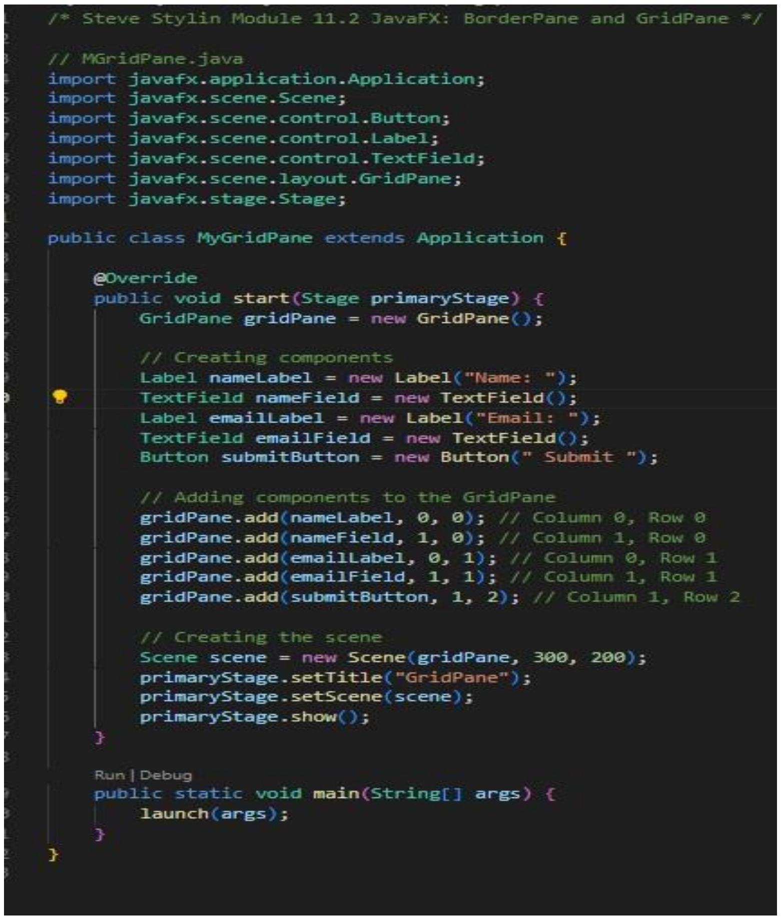
GridPane is a powerful layout manager that arranges its children in a flexible grid of rows and columns. This layout, with its precision and control, is ideal for forms and data entry applications where components need to be aligned in a structured manner. The precision and control offered by GridPane make developers feel empowered and in control of their UI design, making it a preferred choice for many. (Oracle, JavaFX: GridPane. , n.d.).

Key Features of GridPane:

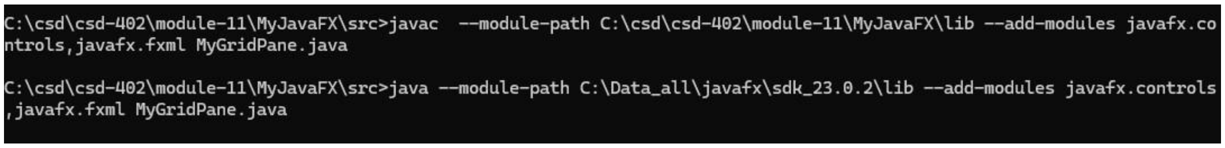
- ✓ Row and Column Management: GridPane meticulously places components within designated rows and columns, enabling users to achieve precise control over their layout and design.
- ✓ Resizable: The grid's ability to automatically adapt to varying screen sizes and resolutions ensures a seamless user experience across devices, maintaining functionality and aesthetics.
- ✓ Alignment Options: Each component within GridPane can be finely aligned within its respective cell, providing enhanced flexibility and creative freedom in arranging and presenting elements.

With GridPane, the user application will be structured with no overlapping or zigzag interface.

In this example below, we are going to create a simple form with labels and text fields. The GridPane is used to create a simple form layout. Each component is added to a specific cell in the grid, allowing for a clean and organized appearance. The Scene is created and displayed on the primary stage.

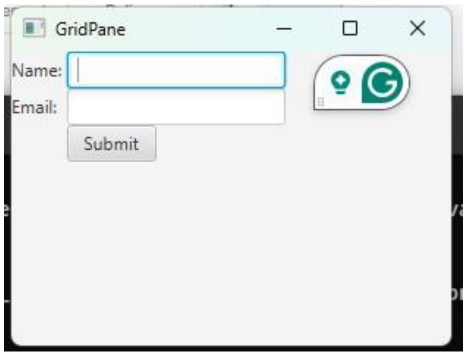


```java
/* Steve Stylin Module 11.2 JavaFX: BorderPane and GridPane */

// MGridPane.java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class MyGridPane extends Application {

    @Override
    public void start(Stage primaryStage) {
        GridPane gridPane = new GridPane();

        // Creating components
        Label nameLabel = new Label("Name: ");
        TextField nameField = new TextField();
        Label emailLabel = new Label("Email: ");
        TextField emailField = new TextField();
        Button submitButton = new Button(" Submit ");

        // Adding components to the GridPane
        gridPane.add(nameLabel, 0, 0); // Column 0, Row 0
        gridPane.add(nameField, 1, 0); // Column 1, Row 0
        gridPane.add(emailLabel, 0, 1); // Column 0, Row 1
        gridPane.add(emailField, 1, 1); // Column 1, Row 1
        gridPane.add(submitButton, 1, 2); // Column 1, Row 2

        // Creating the scene
        Scene scene = new Scene(gridPane, 300, 200);
        primaryStage.setTitle("GridPane");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



```
C:\csd\csd-402\module-11\MyJavaFX\src>javac  --module-path C:\csd\csd-402\module-11\MyJavaFX\lib --add-modules javafx.co
ntrols,javafx.fxml MyGridPane.java

C:\csd\csd-402\module-11\MyJavaFX\src>java --module-path C:\Data_all\javafx\sdk_23.0.2\lib --add-modules javafx.controls
,javafx.fxml MyGridPane.java
```

The output:



Conclusion

BorderPane and GridPane are not just essential layout managers in JavaFX, but they are tools that can dramatically enhance the usability and visual allure of applications. By understanding their unique features and implementation strategies, developers can craft very well-structured and highly responsive user interface designs, making their applications more engaging and effective for users. The impact of these UI design decisions on the user experience is significant and rewarding.

Bibliography

Oracle. (n.d.). *Java Documentation*. Retrieved from JavaFX Overview: https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784

Oracle. (n.d.). *JavaFX BorderPane*. Retrieved from CLass BorderPane: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/BorderPane.html

Oracle. (n.d.). *JavaFX: GridPane.* . Retrieved from https://docs.oracle.com/javase/8/javafx/api/toc.htm