

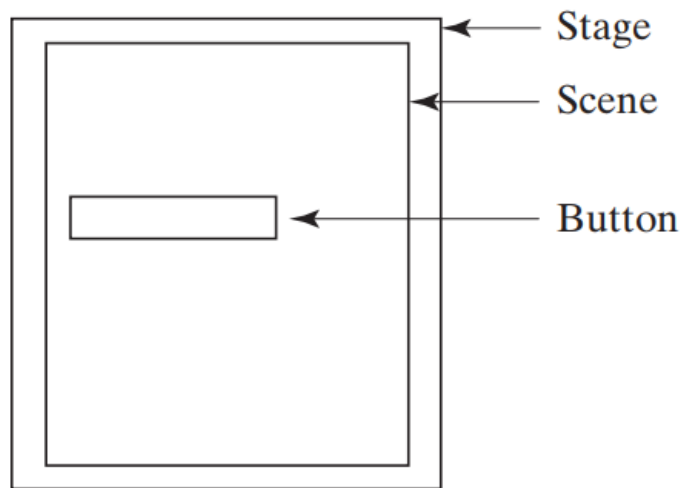
Introduction to GUI

IQRA UNIVERSITY

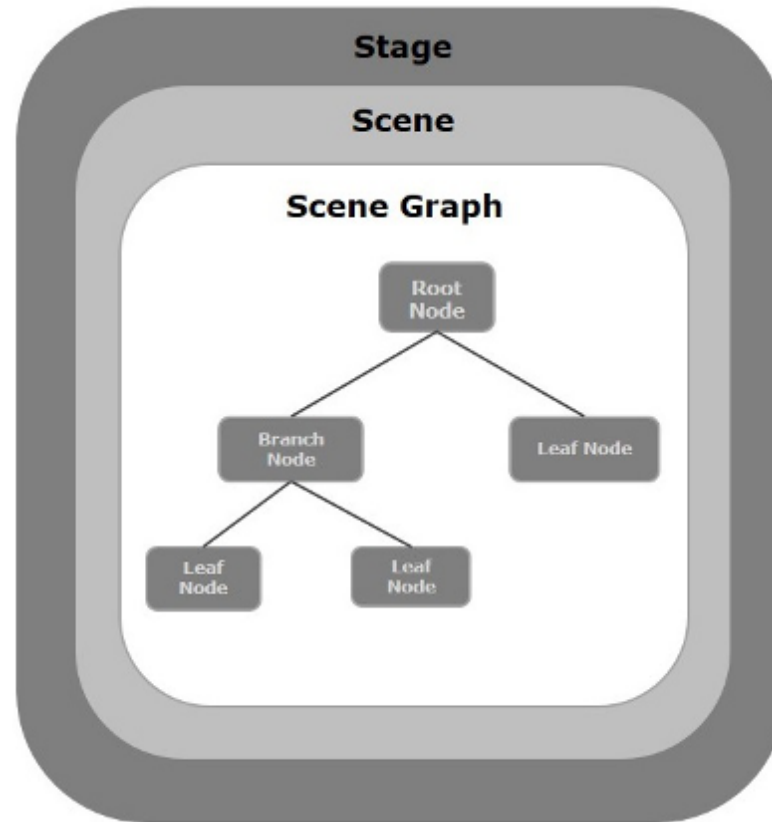
JavaFX

- JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse 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 Swings.
- After the advent of JavaFX, Java programmers can now develop GUI applications effectively with rich content.

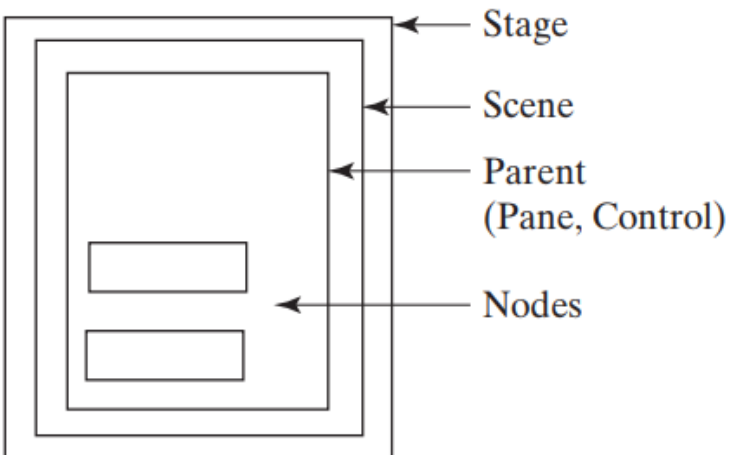
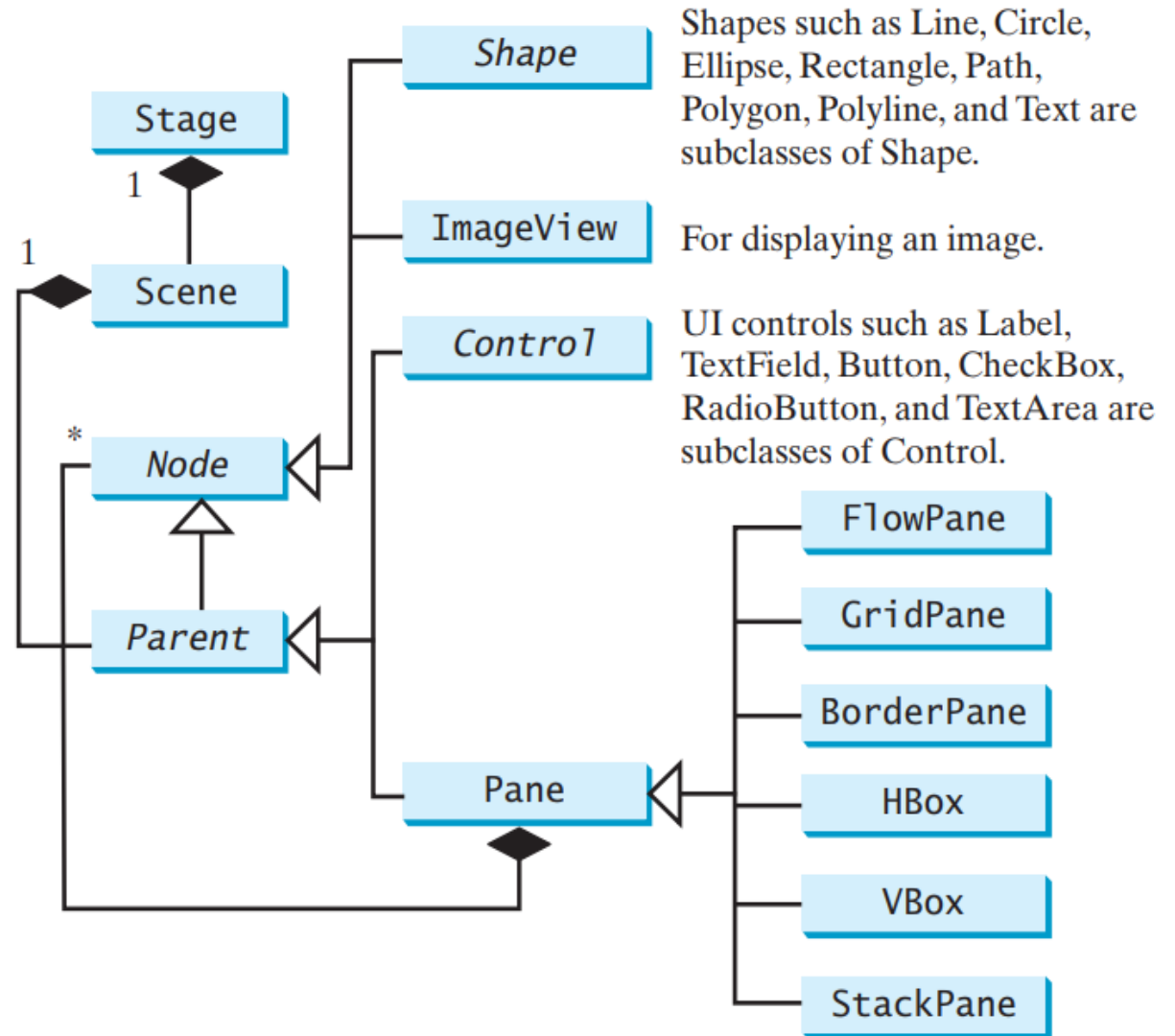
Relationship I



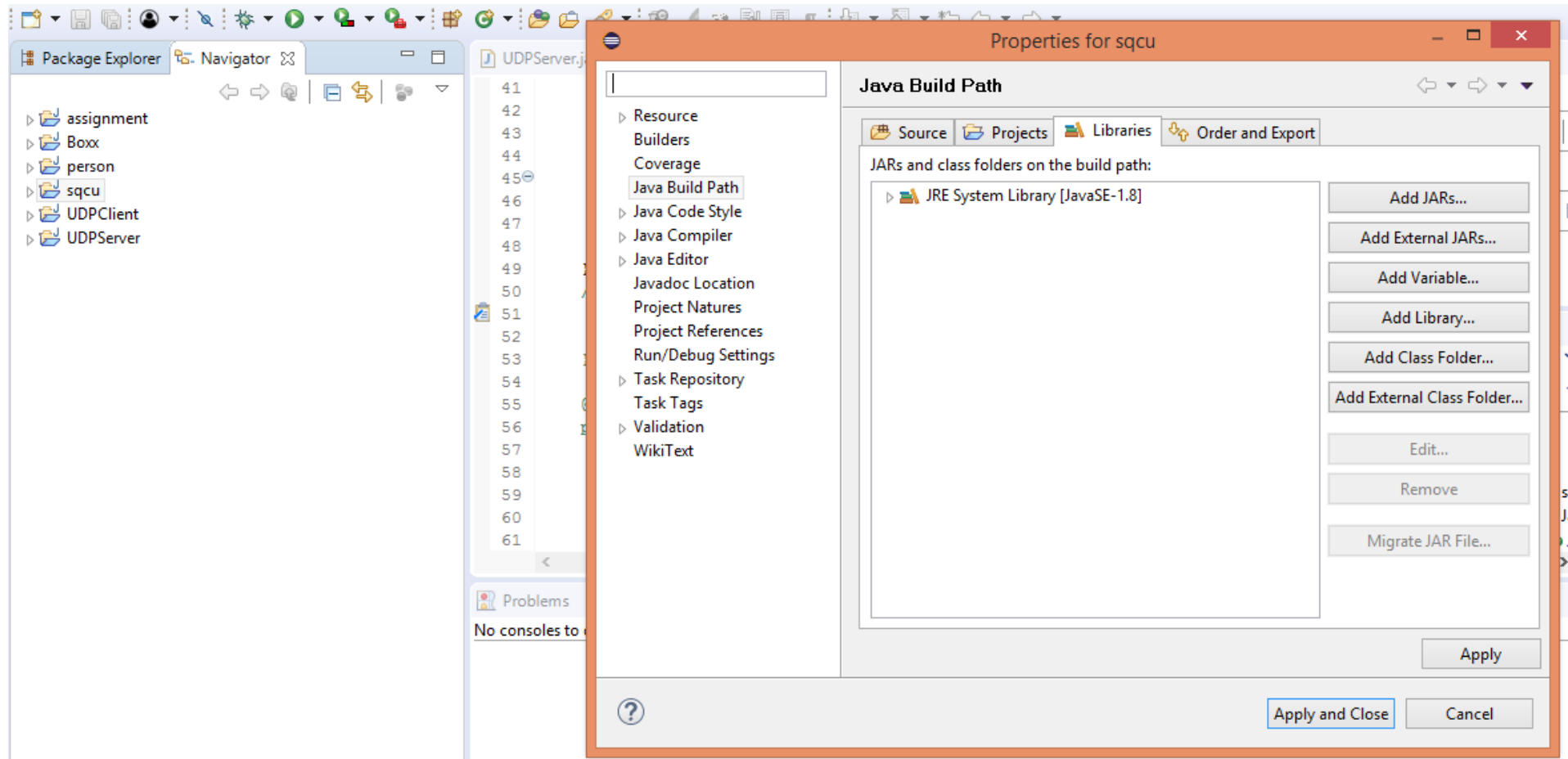
Application Structure



Relationship II



1- Check JRE System library should be 1.8
(by clicking on any java project-> properties->java build
path



2- Add Access Rule

The screenshot shows the Eclipse IDE interface. On the left is the 'Project Explorer' with a tree view containing: Resource, Builders, Coverage, Java Build Path (selected), Java Code Style, Java Compiler, Java Editor, Javadoc Location, Project Natures, Project References, Run/Debug Settings, Task Repository, Task Tags, Validation, and WikiText.

The main area displays the 'Java Build Path' dialog box, with the 'Libraries' tab selected. It shows 'JRE System Library [JavaSE-1.8]' and 'Access rules: 1 rule defined, added to all library ch'. Below this, it lists 'External annotations: (None)', 'Native library location: (None)', and 'resources.jar - C:\eclipse\jre\lib'.

Two 'Add Access Rule' dialog boxes are overlaid on the main window. The one in the foreground has 'Resolution' set to 'Forbidden' and 'Rule Pattern' is empty. The one in the background has 'Resolution' set to 'Accessible' and 'Rule Pattern' set to 'javafx/**'. Both dialog boxes include a help icon (?) and 'OK' and 'Cancel' buttons. The text inside the dialog boxes reads: 'Enter a pattern for the rule.', 'Allowed wildcards are '*', '?' and '*'. Pattern segments are separated by '/'. '*' matches any number of segments. Examples are: 'java/util/*', '**/internal/*', 'org/e*/*'.

At the bottom right, a snippet of the 'Access rules' list is visible, showing 'Accessible: javafx/**' with a checkmark.

Basic Structure of a JavaFX Program

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

public class MyJavaFX extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a scene and place a button in the scene
        Button btOK = new Button("OK");
        Scene scene = new Scene(btOK, 200, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

    /**
     * The main method is only needed for the IDE with limited
     * JavaFX support. Not needed for running from the command line.
     */
    public static void main(String[] args) {
        Application.launch(args);
    }
}
```



Layouts

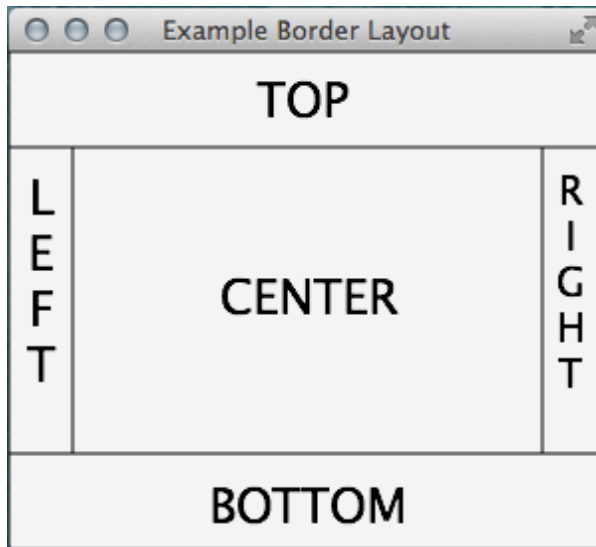
1. BorderPane
2. StackPane
3. GridPane
4. FlowPane
5. HBox
6. VBox
7. TilePane
8. AnchorPane

JavaFX BorderPane

BorderPane arranges the nodes at the left, right, centre, top and bottom of the screen. It is represented by `javafx.scene.layout.BorderPane` class. This class provides various methods like `setRight()`, `setLeft()`, `setCenter()`, `setBottom()` and `setTop()` which are used to set the position for the specified nodes. We need to instantiate BorderPane class to create the BorderPane layout.

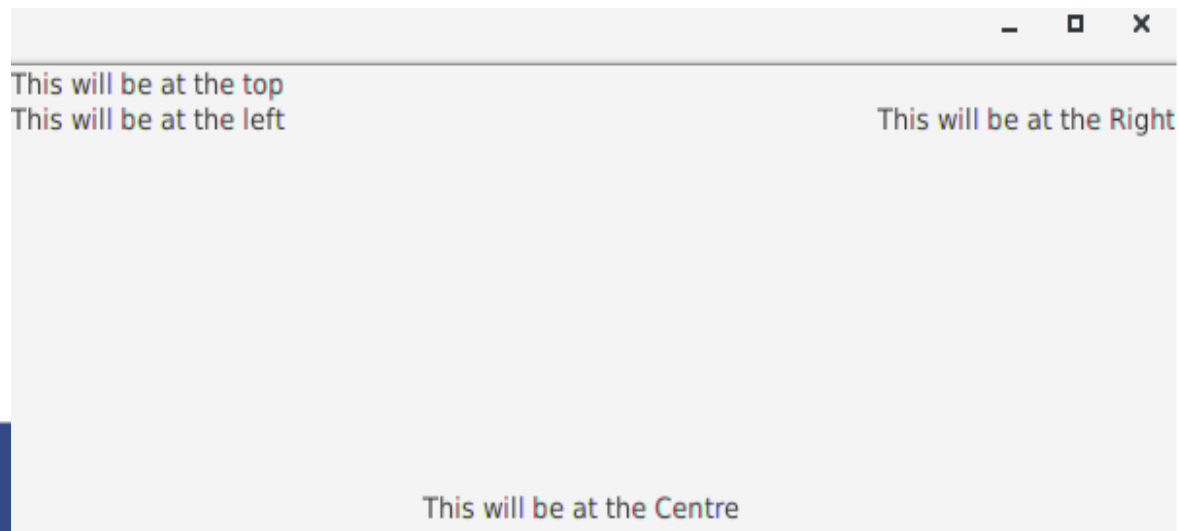
Constructors

1. `BorderPane()` : create the empty layout
2. `BorderPane(Node Center)` : create the layout with the center node
3. `BorderPane(Node Center, Node top, Node right, Node bottom, Node left)` : create the layout with all the nodes



Example of BorderPane

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.*;
import javafx.stage.Stage;
public class Label_Test extends Application {
    public void start(Stage primaryStage) throws Exception {
        BorderPane BPane = new BorderPane();
        BPane.setTop(new Label("This will be at the top"));
        BPane.setLeft(new Label("This will be at the left"));
        BPane.setRight(new Label("This will be at the Right"));
        BPane.setCenter(new Label("This will be at the Centre"));
        BPane.setBottom(new Label("This will be at the bottom"));
        Scene scene = new Scene(BPane,600,400);
        primaryStage.setTitle("Borderpane");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

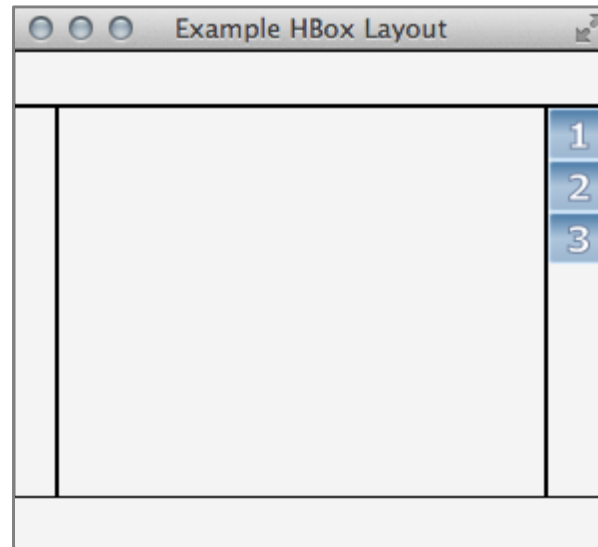


JavaFX StackPane

The StackPane layout pane places all the nodes into a single stack where every new node gets placed on the top of the previous node. It is represented by **`javafx.scene.layout.StackPane`** class. We just need to instantiate this class to implement StackPane layout into our application.

Constructors

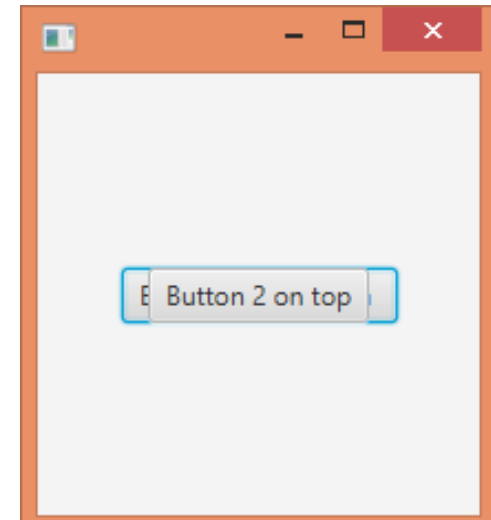
1. `StackPane()`
2. `StackPane(Node/Children)`



StackPane

Example of StackPane

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class Label_Test extends Application {
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("Button 1 on bottom ");
        Button btn2 = new Button("Button 2 on top");
        StackPane root = new StackPane();
        Scene scene = new Scene(root,200,200);
        root.getChildren().addAll(btn1,btn2);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

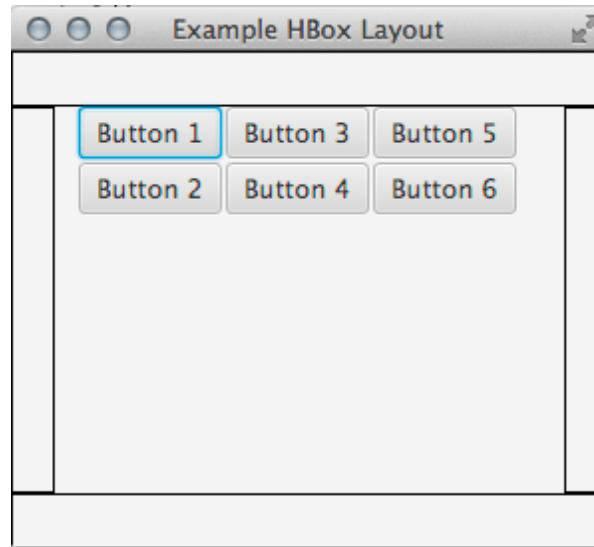


JavaFX GridPane

GridPane Layout pane allows us to add the multiple nodes in multiple rows and columns. It is seen as a flexible grid of rows and columns where nodes can be placed in any cell of the grid. It is represented by **javafx.scene.layout.GridPane** class. We just need to instantiate this class to implement GridPane.

Constructors

Public GridPane(): creates a gridpane with 0 hgap/vgap

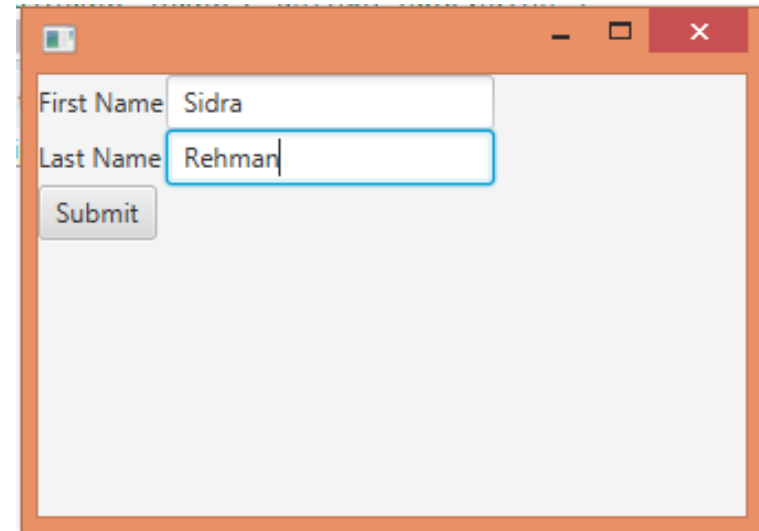


GridPane

Example of GridPane

```
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 Label_Test extends Application {
    public void start(Stage primaryStage) throws Exception {
        Label first_name=new Label("First Name");
        Label last_name=new Label("Last Name");
        TextField tf1=new TextField();
        TextField tf2=new TextField();
        Button Submit=new Button ("Submit");
        GridPane root=new GridPane();
        Scene scene = new Scene(root,400,200);
        root.addRow(0, first_name,tf1);
        root.addRow(1, last_name,tf2);
        root.addRow(2, Submit);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args){
        launch(args);
    }
}
```



JavaFX FlowPane

FlowPane layout pane organizes the nodes in a flow that are wrapped at the flowpane's boundary. The horizontal flowpane arranges the nodes in a row and wrap them according to the flowpane's width. The vertical flowpane arranges the nodes in a column and wrap them according to the flowpane's height. FlowPane layout is represented by **javafx.scene.layout.FlowPane** class. We just need to instantiate this class to create the flowpane layout.

Constructors

There are 8 constructors in the class that are given below.

FlowPane()

FlowPane(Double Hgap, Double Vgap)

FlowPane(Double Hgap, Double Vgap, Node? children)

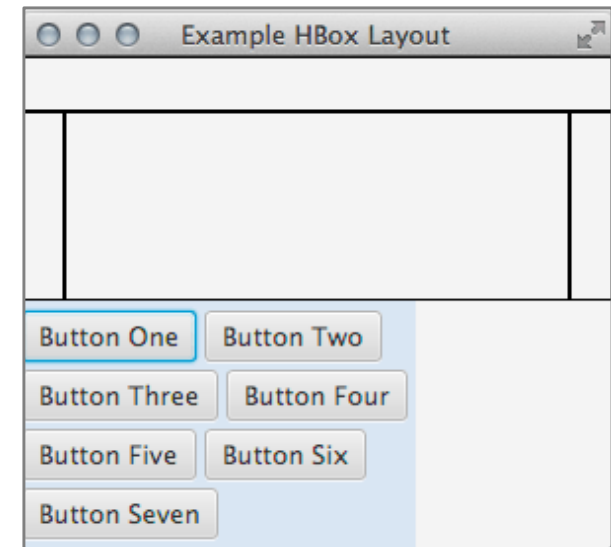
FlowPane(Node... Children)

FlowPane(Orientation orientation)

FlowPane(Orientation orientation, double Hgap, Double Vgap)

FlowPane(Orientation orientation, double Hgap, Double Vgap, Node? children)

FlowPane(Orientation orientation, Node... Children)



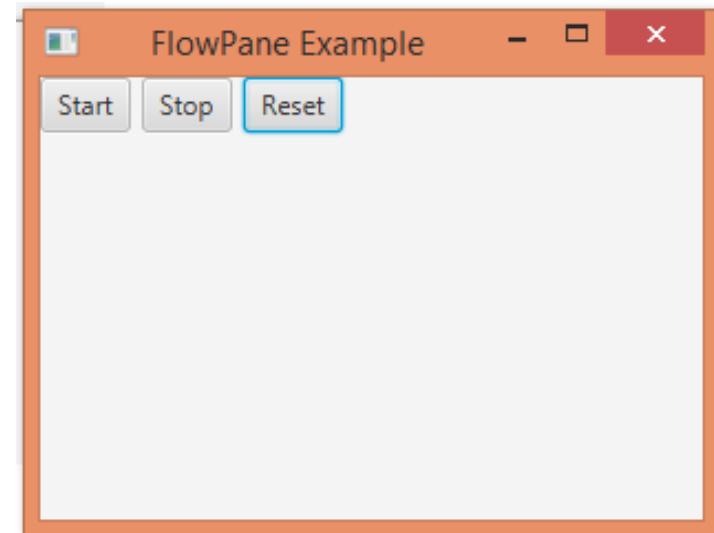
FlowPane

Example of FlowPane

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

public class FlowPaneTest extends Application {
    public void start(Stage primaryStage) {
        primaryStage.setTitle("FlowPane Example");
        FlowPane root = new FlowPane();
        root.setVgap(6);
        root.setHgap(5);
        root.setPrefWrapLength(250);
        root.getChildren().add(new Button("Start"));
        root.getChildren().add(new Button("Stop"));
        root.getChildren().add(new Button("Reset"));
        Scene scene = new Scene(root,300,200);

        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

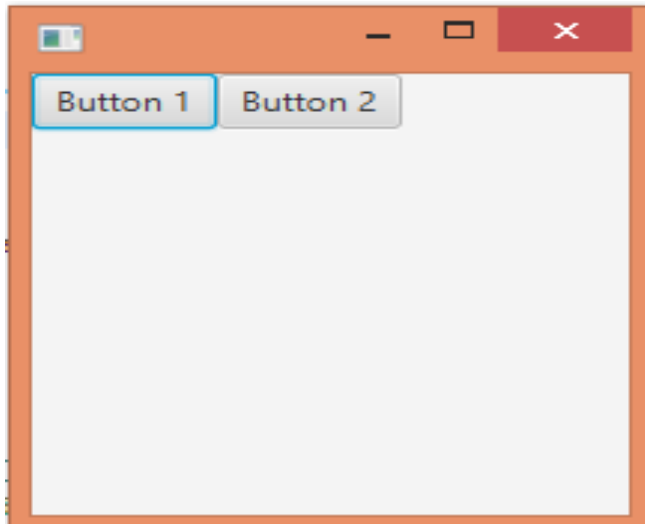


JavaFX HBox

HBox layout pane arranges the nodes in a single row. It is represented by `javafx.scene.layout.HBox` class. We just need to instantiate HBox class in order to create HBox layout.

Constructors

1. `new HBox()` : create HBox layout with 0 spacing
2. `new HBox(Double spacing)` : create HBox layout with a spacing value



HBox

Example

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

public class Label_Test extends Application {
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("Button 1");
        Button btn2 = new Button("Button 2");
        HBox root = new HBox();
        Scene scene = new Scene(root,200,200);
        root.getChildren().addAll(btn1,btn2);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

JavaFX VBox

Instead of arranging the nodes in horizontal row, VBox Layout Pane arranges the nodes in a single vertical column. It is represented by `javafx.scene.layout.VBox` class which provides all the methods to deal with the styling and the distance among the nodes. This class needs to be instantiated in order to implement VBox layout in our application.

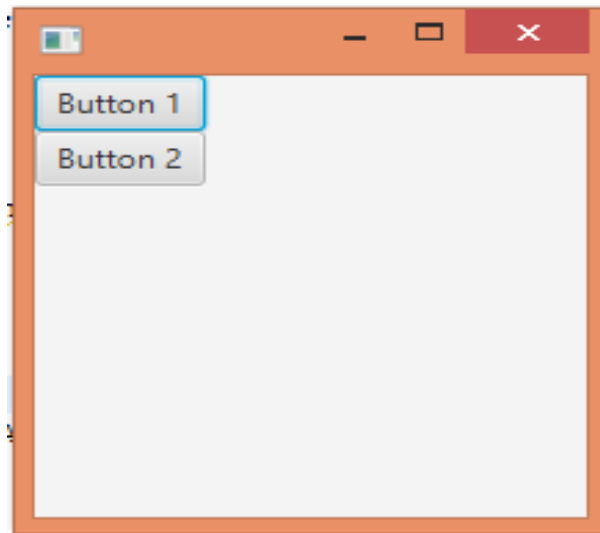
Constructors

`VBox()` : creates layout with 0 spacing

`Vbox(Double spacing)` : creates layout with a spacing value of double type

`Vbox(Double spacing, Node? children)` : creates a layout with the specified spacing among the specified child nodes

`Vbox(Node? children)` : creates a layout with the specified nodes having 0 spacing among them



VBox

Example

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

public class Label_Test extends Application {
    public void start(Stage primaryStage) throws Exception {
        Button btn1 = new Button("Button 1");
        Button btn2 = new Button("Button 2");
        VBox root = new VBox();
        Scene scene = new Scene(root,200,200);
        root.getChildren().addAll(btn1,btn2);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

- 1 [HBox](#) The HBox layout arranges all the nodes in our application in a single horizontal row.
The class named **HBox** of the package **javafx.scene.layout** represents the text horizontal box layout.
- 2 [VBox](#) The VBox layout arranges all the nodes in our application in a single vertical column.
The class named **VBox** of the package **javafx.scene.layout** represents the text Vertical box layout.
- 3 [BorderPane](#) The Border Pane layout arranges the nodes in our application in top, left, right, bottom and center positions.
The class named **BorderPane** of the package **javafx.scene.layout** represents the border pane layout.
- 4 [StackPane](#) The stack pane layout arranges the nodes in our application on top of another just like in a stack. The node added first is placed at the bottom of the stack and the next node is placed on top of it.
The class named **StackPane** of the package **javafx.scene.layout** represents the stack pane layout.
- 5 [AnchorPane](#) The Anchor pane layout anchors the nodes in our application at a particular distance from the pane.
The class named **AnchorPane** of the package **javafx.scene.layout** represents the Anchor Pane layout.
- 6 [TilePane](#) The Tile Pane layout adds all the nodes of our application in the form of uniformly sized tiles.
The class named **TilePane** of the package **javafx.scene.layout** represents the TilePane layout.
- 7 [GridPane](#) The Grid Pane layout arranges the nodes in our application as a grid of rows and columns. This layout comes handy while creating forms using JavaFX.
The class named **GridPane** of the package **javafx.scene.layout** represents the GridPane layout.
- 8 [FlowPane](#) The f bw pane layout wraps all the nodes in a f bw. A horizontal f bw pane wraps the elements of the pane at its height, while a vertical flow pane wraps the elements at its width.
The class named **FlowPane** of the package **javafx.scene.layout** represents the Flow Pane layout.

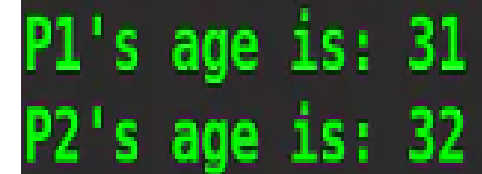
Static keyword ,

- Static keyword in java in Java indicates that a particular member is not an instance, but rather part of a type. The static member will be shared among all instances of the class, so we will only create one instance of it.
- If any member in a class is declared as static, it means that even before the class is initiated, all the static members can be accessed and become active. In contrast to this, non-static members of the same class will cease to exist when there is no object or the object goes out of scope.

When you create an object or instance for a class in Java, each object will have its own copy of the members such as variables and methods.

For example,

```
class Person{
    int age;
}
class Main{
    public static void main(String args[]){
        Person p1 = new Person();
        Person p2 = new Person();
        p1.age = 31;
        p2.age = 32;
        System.out.println("P1\'s age is: " + p1.age);
        System.out.println("P2\'s age is: " + p2.age);
    }
}
```



```
P1's age is: 31
P2's age is: 32
```

When you create an object or instance for a class in Java, each object will have its own copy of the members such as variables and methods.

For example,

```
class Person{
    static int age;
}
class Main{
    public static void main(String args[]){
        Person p1 = new Person();
        Person p2 = new Person();
        p1.age = 31;
        p2.age = 32;
        System.out.println("P1\'s age is: " + p1.age);
        System.out.println("P2\'s age is: " + p2.age);
    }
}
```



```
P1's age is: 32
P2's age is: 32
```



```
class Test{
    int counter;
    Test(){
        counter++;
        System.out.println("Current Value of the Counter is: " + counter);
    }
}

class Main{
    public static void main(String args[]){
        Test t1 = new Test();
        Test t2 = new Test();
        Test t3 = new Test();
    }
}
```

```
Current Value of the Counter is: 1
Current Value of the Counter is: 1
Current Value of the Counter is: 1
```

```
class Test{
    static int counter;
    Test(){
        counter++;
        System.out.println("Current Value of the Counter is: " + counter);
    }
}

class Main{
    public static void main(String args[]){
        Test t1 = new Test();
        Test t2 = new Test();
        Test t3 = new Test();
    }
}
```

```
Current Value of the Counter is: 1
Current Value of the Counter is: 2
Current Value of the Counter is: 3
```

```
class Test{
    int counter;
    public static void increment(){
        counter++;
        System.out.println("Current value of Counter is:"+counter);
    }
}

class Main{
    public static void main(String args[]){
        Test.increment();
    }
}
```

```
Main.java:4: error: non-static variable counter cannot be referenced from a static context
        counter++;
        ^
Main.java:5: error: non-static variable counter cannot be referenced from a static context
        System.out.println("Current value of Counter is: " + counter);
                        ^
2 errors
```

```
class Test{
    static int counter;
    public static void increment(){
        counter++;
        System.out.println("Current value of Counter is: " + counter);
    }
}

class Main{
    public static void main(String args[]){
        Test.increment();
        Test.increment();
        Test.increment();
    }
}
```

```
Current value of Counter is: 1
Current value of Counter is: 2
Current value of Counter is: 3
```

Static Variables	Non-Static Variables
They can access them using class names.	They can be accessed only using objects.
They can access them with static methods as well as non-static methods.	They can be accessed only using non-static methods.
They are allocated memory only once while loading the class.	A memory per object is allocated.
These variables are shared by all the objects or instances of the class.	Each object has its own copy of the non-static variables.
Static variables have global scope.	They have local scope.

Static Methods	Non-Static Methods
These methods can only access static variables of other classes as well as their own class.	They can access both static as well as non-static members.
You can't override static methods.	They can be overridden.
Less memory consumption since they are allocated memory only once when the class is being loaded.	Memories are allocated for each object.
These methods support early or compile-time binding.	They support late, run-time, or dynamic binding.