Chapter 14 JavaFX Basics

Panes, Color and Font Classes

Panes, UI Controls, and Shapes

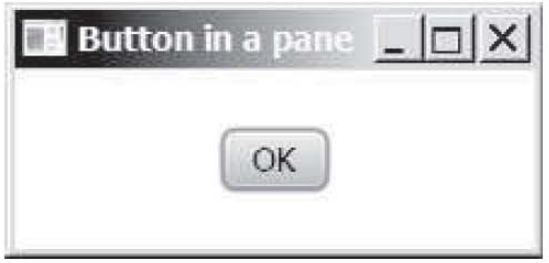
- A Pane is a container classes that is used for automatically laying out the nodes in the desired location and size.
- You place nodes inside a pane and then place the pane into a scene.
- A node is a visual component such as a shape, an image view, a UI control, or a pane.
- A *shape* refers to a text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.
- A *UI control* refers to a label, button, check box, radio button, text field, text area, etc.

Panes, UI Controls, and Shapes

- Scene can contain a Control or a Pane, but not a Shape or an ImageView, while a Pane can contain any subtype of Node.
- You can create a Scene using one of two ways:
- Constructor Scene(Parent, width, height) or,
- Constructor Scene(Parent)
- In the latter constructor the dimension of the scene is automatically decided by JRM.
- Every subclass of Node has a no-arg constructor for creating a default node.

Button Simple Example

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

public class ButtonInPane 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
    StackPane pane = new StackPane();
    pane.getChildren().add(new Button("OK"));
    Scene scene = new Scene(pane, 200, 50);
    primaryStage.setTitle("Button in a pane"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```

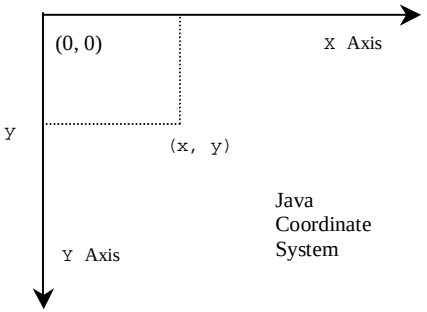


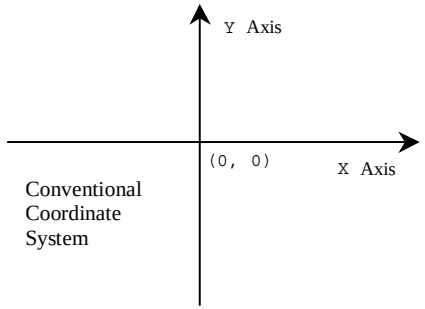
Button Simple Example

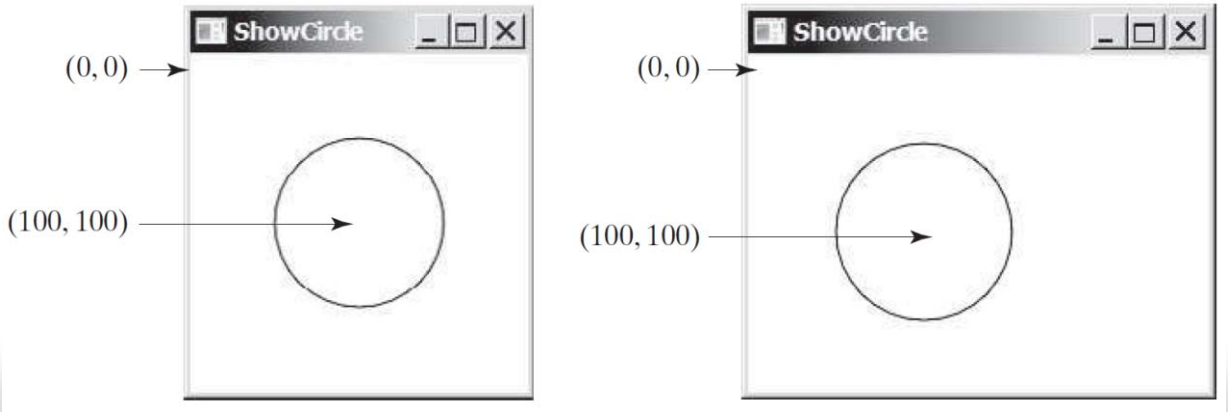
- The getChildren() method returns an instance of javafx.collections.ObservableList of Nodes.
- ObservableList behaves very much like an ArrayList for storing a collection of elements.
- Invoking add(e) adds an element to the list.
- The StackPane places the nodes in the center of the pane on top of each other.
- Here, there is only one node in the pane.
- The StackPane respects a node's default size. So you see the button displayed in its default size.

Another Button Example

```
public class ShowCircle extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.setCenterX(100);
    circle.setCenterY(100);
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);

    // Create a pane to hold the circle
    Pane pane = new Pane();
    pane.getChildren().add(circle);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("ShowCircle"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```







Common Properties and Methods for Nodes

- Nodes share many common properties. The most important among these is the **style** property.
- The style properties in JavaFX are called *JavaFX cascading style sheets (CSS)*.
- Each node has its own style properties.
- In JavaFX, a style property is defined with a prefix –fx-.
- The syntax for setting a style is **styleName:value**.
- Multiple style properties for a node can be set together separated by semicolon (;).

Common Properties and Methods for Nodes

- For example, the following statement
  circle.setStyle**("-fx-stroke: black**; **-fx-fill: red**;");

  sets two JavaFX CSS properties for a circle.
- This statement is equivalent to the following two statements.
  - ✓ circle.setStroke(Color.BLACK);
  - ✓ circle.setFill(Color.RED);
- If an incorrect JavaFX CSS is used, your program will still compile and run, but the style is ignored.

The Color Class

The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.paint.Color

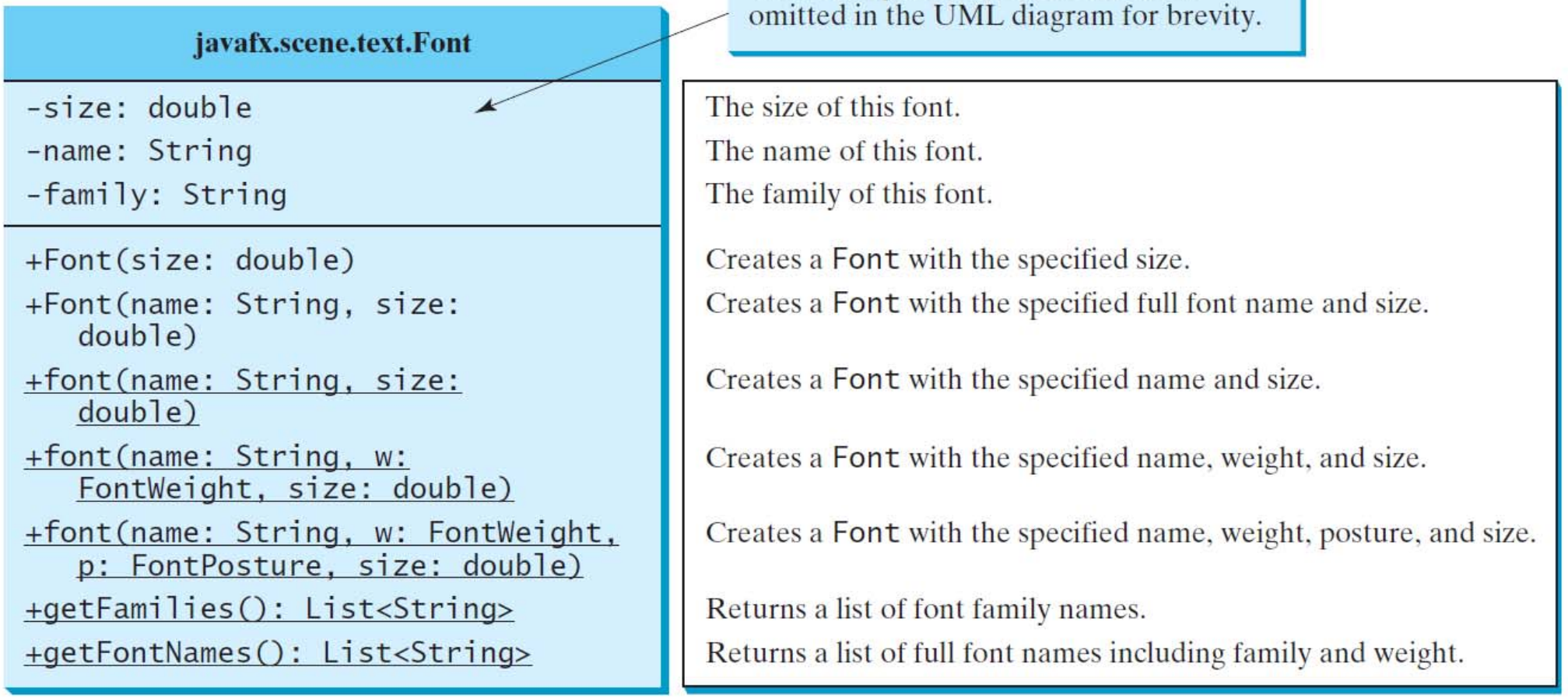
-red: double	The red value of this Color (between 0.0 and 1.0).
-green: double	The green value of this Color (between 0.0 and 1.0).
-blue: double	The blue value of this Color (between 0.0 and 1.0).
-opacity: double	The opacity of this Color (between 0.0 and 1.0).
+Color(r: double, g: double, b: double, opacity: double)	Creates a Color with the specified red, green, blue, and opacity values.
+brighter(): Color	Creates a Color that is a brighter version of this Color.
+darker(): Color	Creates a Color that is a darker version of this Color.
+color(r: double, g: double, b: double): Color	Creates an opaque Color with the specified red, green, and blue values.
+color(r: double, g: double, b: double, opacity: double): Color	Creates a Color with the specified red, green, blue, and opacity values.
+rgb(r: int, g: int, b: int): Color	Creates a Color with the specified red, green, and blue values in the range from 0 to 255.
+rgb(r: int, g: int, b: int, opacity: double): Color	Creates a Color with the specified red, green, and blue values in the range from 0 to 255 and a given opacity.

The Color Class

- JavaFX defines the abstract Paint class for painting a node.
- The javafx.scene.paint.Color is a subclass of Paint.
- The Color class is immutable. Once a Color object is created, its properties cannot be changed.
- The brighter() method returns a new Color with a larger red, green, and blue values.
- The darker() method returns a new Color with a smaller red, green, and blue values.
- Built-in static colors can be used, for example: circle.setFill(**Color.RED**);

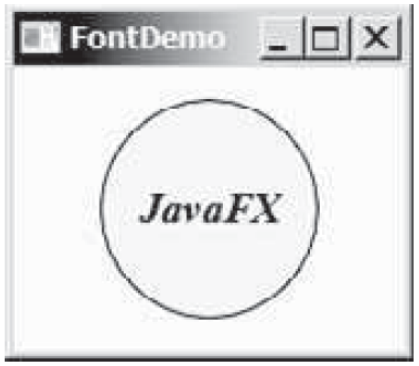
The Font Class

The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.



Font Example

```
public class FontDemo extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the circle
    Pane pane = new StackPane();

    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));
    pane.getChildren().add(circle); // Add circle to the pane

    // Create a label and set its properties
    Label label = new Label("JavaFX");
    label.setFont(Font.font("Times New Roman",
      FontWeight.BOLD, FontPosture.ITALIC, 20));
    pane.getChildren().add(label);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("FontDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```



The Font Class

- A **StackPane** places the nodes in the center and nodes are placed on top of each other.
- As you resize the window, the circle and label are displayed in the center of the window, because the circle and label are placed in the stack pane.
- Stack pane automatically places nodes in the center of the pane.
- A **Font** object is immutable. Once a **Font** object is created, its properties cannot be changed.

Layout Panes

JavaFX provides many types of panes for organizing nodes in a container.

Class	*Description*
`Pane`	Base class for layout panes. It contains the `getChildren()` method for returning a list of nodes in the pane.
`StackPane`	Places the nodes on top of each other in the center of the pane.
`FlowPane`	Places the nodes row-by-row horizontally or column-by-column vertically.
`GridPane`	Places the nodes in the cells in a two-dimensional grid.
`BorderPane`	Places the nodes in the top, right, bottom, left, and center regions.
`HBox`	Places the nodes in a single row.
`VBox`	Places the nodes in a single column.

Layout Panes

- **Pane** is the base class for all specialized panes.
- Each pane contains a list for holding nodes in the pane.
- This list is an instance of **ObservableList**, which can be obtained using pane's **getChildren**() method.
- You can use the **add**(node) method to add an element to the list, or
- use **addAll**(node1, node2, ...) to add a variable number of nodes to the pane.
- A node can be added to only one pane and only one time. Adding a node to a pane multiple times or to different panes will cause a runtime error.

FlowPane

- Arranges the nodes in the pane horizontally from left to right or vertically from top to bottom in the order in which they were added.
- When one row or one column is filled, a new row or column is started.
- Data fields **alignment**, **orientation**, **hgap**, and **vgap** are Object properties.
- Each Object property in JavaFX has a getter method (e.g., getHgap()) that returns its value, a setter method (e.g., sethGap(double)) for setting a value, and a getter method that returns the property itself (e.g., hGapProperty()).

FlowPane

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.FlowPane

```
-alignment: ObjectProperty<Pos>
-orientation:
   ObjectProperty<Orientation>
-hgap: DoubleProperty
-vgap: DoubleProperty
```

```
+FlowPane()
+FlowPane(hgap: double, vgap:
   double)
+FlowPane(orientation:
   ObjectProperty<Orientation>)
+FlowPane(orientation:
   ObjectProperty<Orientation>,
   hgap: double, vgap: double
```

The overall alignment of the content in this pane (default: `Pos.LEFT`).
The orientation in this pane (default: `Orientation.HORIZONTAL`).

The horizontal gap between the nodes (default: 0).
The vertical gap between the nodes (default: 0).

Creates a default `FlowPane`.
Creates a `FlowPane` with a specified horizontal and vertical gap.

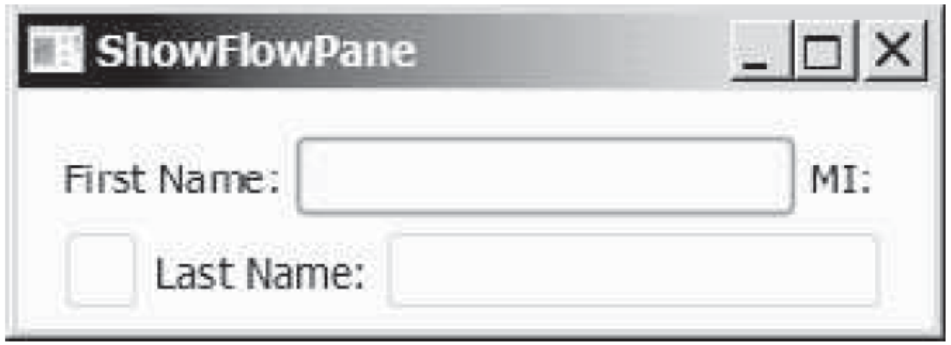
Creates a `FlowPane` with a specified orientation.

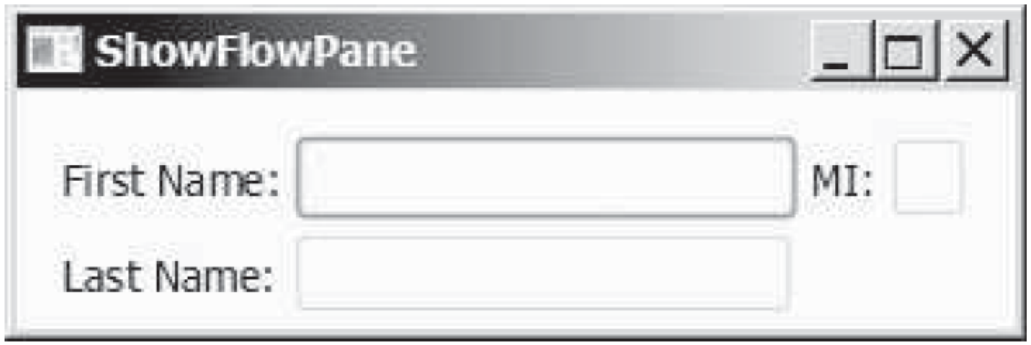
Creates a `FlowPane` with a specified orientation, horizontal gap and vertical gap.

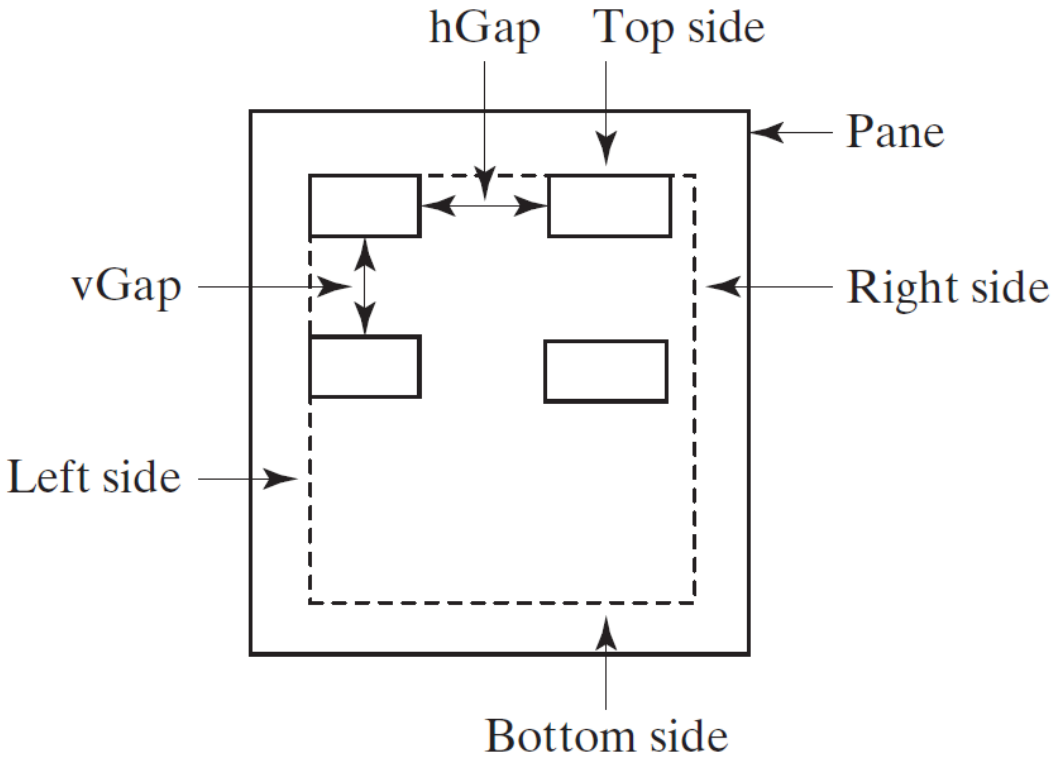
FlowPane Example

```
public class ShowFlowPane extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane and set its properties
    FlowPane pane = new FlowPane();
    pane.setPadding(new Insets(11, 12, 13, 14));
    pane.setHgap(5);
    pane.setVgap(5);

    // Place nodes in the pane
    pane.getChildren().addAll(new Label("First Name:"),
      new TextField(), new Label("MI:"));
    TextField tfMi = new TextField();
    tfMi.setPrefColumnCount(1);
    pane.getChildren().addAll(tfMi, new Label("Last Name:"),
      new TextField());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 250);
    primaryStage.setTitle("ShowFlowPane"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```

FlowPane Example







GridPane

- A GridPane arranges nodes in a grid (matrix) formation.
- The nodes are placed in the specified column and row indices.
- The column and row indexes start from 0.
- Not every cell in the grid needs to be filled.
- To remove a node from a GridPane, use pane.getChildren().remove(node).
- To remove all nodes, use pane.getChildren().removeAll().

GridPane

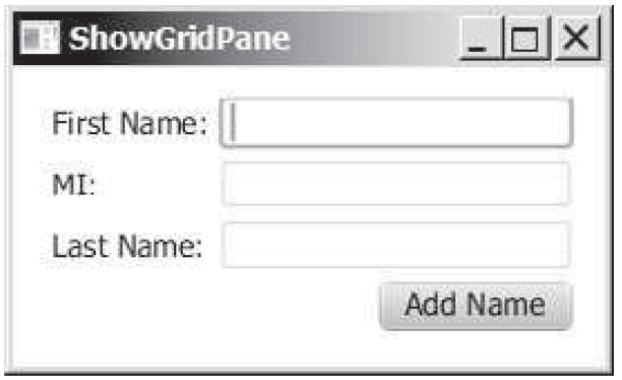
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.GridPane

-alignment: ObjectProperty<Pos>	The overall alignment of the content in this pane (default: Pos.LEFT).
-gridLinesVisible: BooleanProperty	Is the grid line visible? (default: false)
-hgap: DoubleProperty	The horizontal gap between the nodes (default: 0).
-vgap: DoubleProperty	The vertical gap between the nodes (default: 0).
+GridPane()	Creates a GridPane.
+add(child: Node, columnIndex: int, rowIndex: int): void	Adds a node to the specified column and row.
+addColumn(columnIndex: int, children: Node...): void	Adds multiple nodes to the specified column.
+addRow(rowIndex: int, children: Node...): void	Adds multiple nodes to the specified row.
+getColumnIndex(child: Node): int	Returns the column index for the specified node.
+setColumnIndex(child: Node, columnIndex: int): void	Sets a node to a new column. This method repositions the node.
+getRowIndex(child:Node): int	Returns the row index for the specified node.
+setRowIndex(child: Node, rowIndex: int): void	Sets a node to a new row. This method repositions the node.
+setHalighnment(child: Node, value: HPos): void	Sets the horizontal alignment for the child in the cell.
+setValighnment(child: Node, value: VPos): void	Sets the vertical alignment for the child in the cell.

GridPane

```
public class ShowGridPane extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane and set its properties
    GridPane pane = new GridPane();
    pane.setAlignment(Pos.CENTER);
    pane.setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
    pane.setHgap(5.5);
    pane.setVgap(5.5);

    // Place nodes in the pane
    pane.add(new Label("First Name:"), 0, 0);
    pane.add(new TextField(), 1, 0);
    pane.add(new Label("MI:"), 0, 1);
    pane.add(new TextField(), 1, 1);
    pane.add(new Label("Last Name:"), 0, 2);
    pane.add(new TextField(), 1, 2);
    Button btAdd = new Button("Add Name");
    pane.add(btAdd, 1, 3);
    GridPane.setHalignment(btAdd, HPos.RIGHT);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowGridPane"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```



BorderPane

- A BorderPane can place nodes in five regions: top, bottom, left, right, and center, using the setTop(node), setBottom(node), setLeft(node), setRight(node), and setCenter(node) methods.
- Note that a pane is a node. So a pane can be added into another pane.
- To remove a node from the top region, invoke **setTop**(null).
- If a region is not occupied, no space will be allocated for this region.

BorderPane

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.BorderPane

```
-top: ObjectProperty<Node>
-right: ObjectProperty<Node>
-bottom: ObjectProperty<Node>
-left: ObjectProperty<Node>
-center: ObjectProperty<Node>
```

```
+BorderPane()
+setAlignment(child: Node, pos: Pos)
```

The node placed in the top region (default: `null`).
The node placed in the right region (default: `null`).
The node placed in the bottom region (default: `null`).
The node placed in the left region (default: `null`).
The node placed in the center region (default: `null`).

Creates a `BorderPane`.
Sets the alignment of the node in the `BorderPane`.

BorderPane

```
public class ShowBorderPane extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a border pane
    BorderPane pane = new BorderPane();

    // Place nodes in the pane
    pane.setTop(new CustomPane("Top"));
    pane.setRight(new CustomPane("Right"));
    pane.setBottom(new CustomPane("Bottom"));
    pane.setLeft(new CustomPane("Left"));
    pane.setCenter(new CustomPane("Center"));

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowBorderPane"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}

// Define a custom pane to hold a label in the center of the pane
class CustomPane extends StackPane {
  public CustomPane(String title) {
    getChildren().add(new Label(title));
    setStyle("-fx-border-color: red");
    setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
  }
}
```