Chapter 14 JavaFX Basics

Dr. Asem Kitana
Dr. Abdallah Karakra

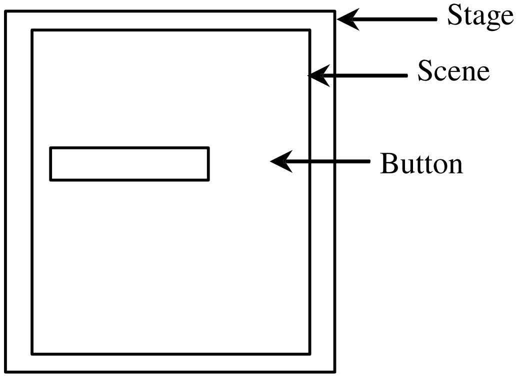
JavaFX vs **Swing** and **AWT**

- When Java was introduced, the **GUI** classes were bundled in a library known as the ***Abstract Windows Toolkit (AWT)***.
  - **AWT** is fine for developing simple graphical user interfaces, *but not for developing comprehensive* **GUI**.
  - In addition, **AWT** is prone to platform-specific bugs.
- The **AWT** components were replaced by a more robust, versatile, and flexible library known as ***Swing***.
  - **Swing** components depend less on the target platform and use less of the native **GUI** resource.
- With the release of Java 8, **Swing** is replaced by a completely new **GUI** platform known as ***JavaFX***.

Basic Structure of JavaFX

- Application: JavaFX programs all start not as some "regular" class like we've been doing, but as an extension of the abstract Application class in JavaFX, javafx.application.Application

- Override the start(Stage) method: The **start** method normally places UI controls in a scene and displays the scene in a stage



- Stage, Scene, and Nodes

Basic Structure of JavaFX

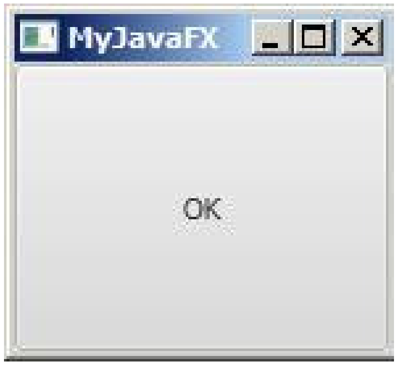
```
public class MyProgram
{
    // Body of class
}
```

Becomes:

```
import javafx.application.Application;
…
public class MyProgram extends Application
{
    // Body of class
}
```

The abstract **javafx.application.Application** *class defines the essential framework for writing JavaFX programs.*

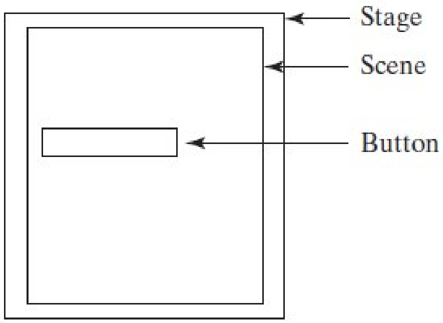
Our First JavaFX Program

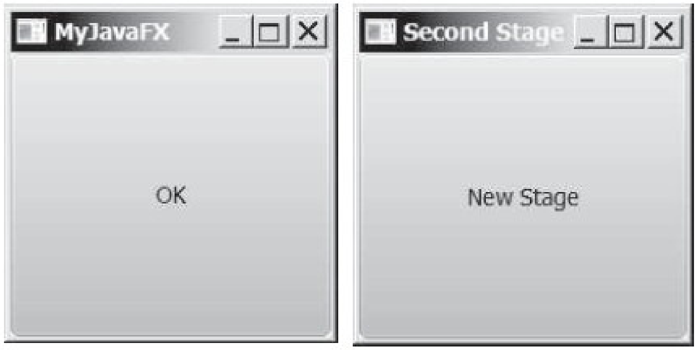


```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class MyJavaFX extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a button and place it in the scene
    Button btOK = new Button("OK"); //Create a button
    Scene scene = new Scene(btOK, 200, 250); //Create a Scene
    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) {
    launch(args);
  }
}
```

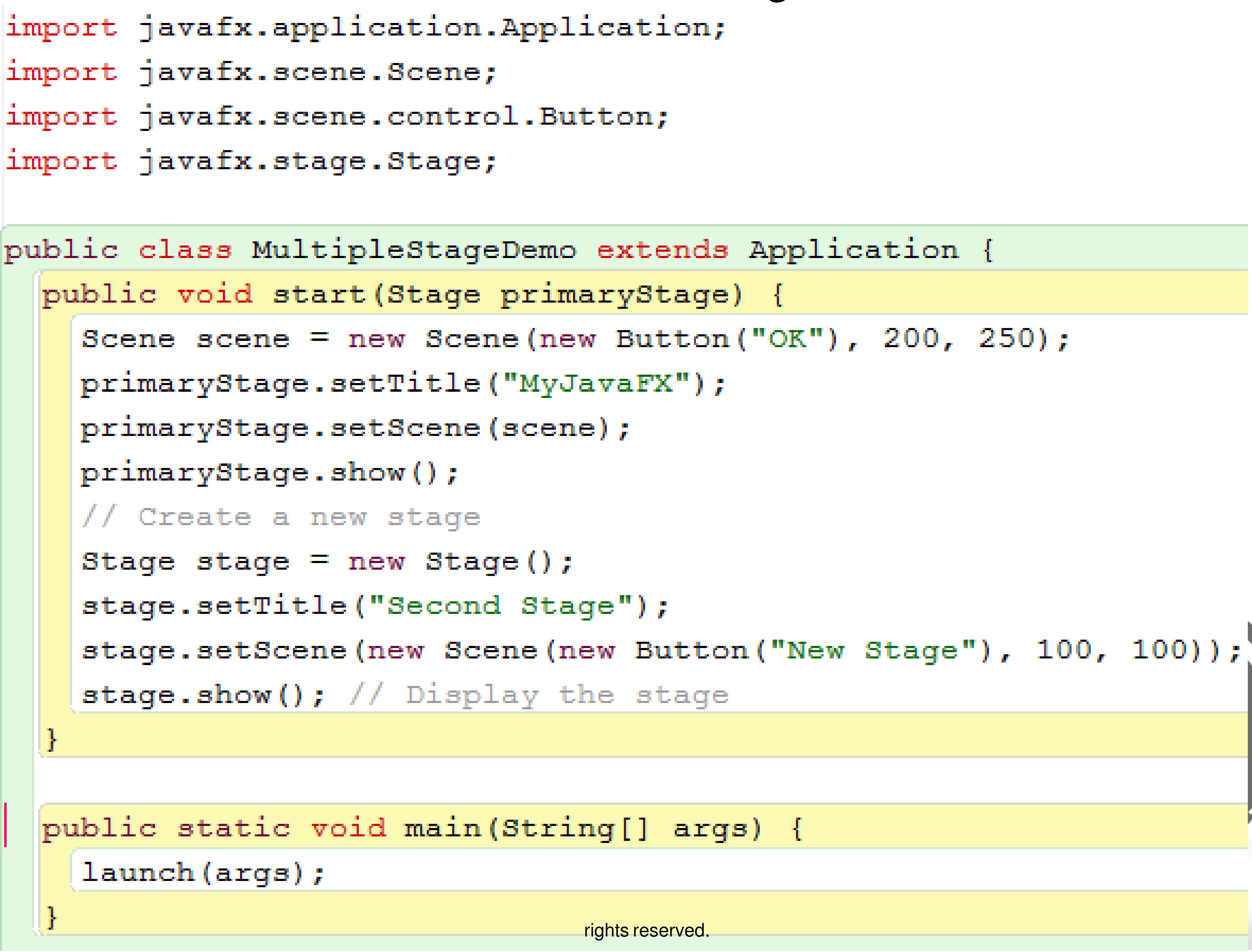
Our First JavaFX Program

- In JavaFX, the stage is the window our code runs in
- Since every GUI application, by definition, involves a window with the UI, we get the primaryStage by default when the application launches.
- Our applications are not limited to a single stage
- The code to setup this two-stage UI is on the next slide





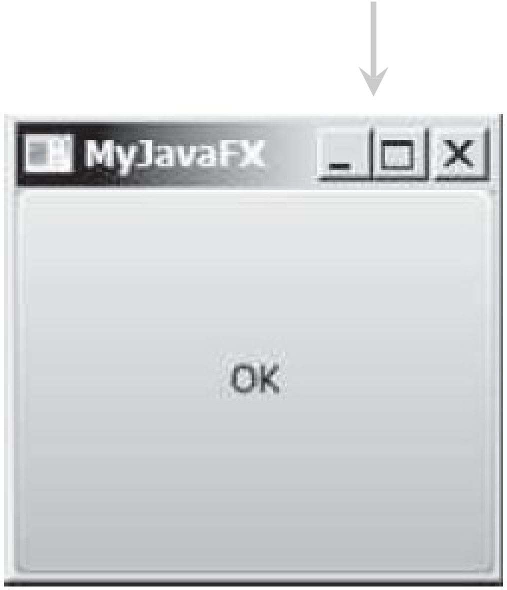
Our First JavaFX Program



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

public class MultipleStageDemo extends Application {
  public void start(Stage primaryStage) {
    Scene scene = new Scene(new Button("OK"), 200, 250);
    primaryStage.setTitle("MyJavaFX");
    primaryStage.setScene(scene);
    primaryStage.show();
    // Create a new stage
    Stage stage = new Stage();
    stage.setTitle("Second Stage");
    stage.setScene(new Scene(new Button("New Stage"), 100, 100));
    stage.show(); // Display the stage
  }

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

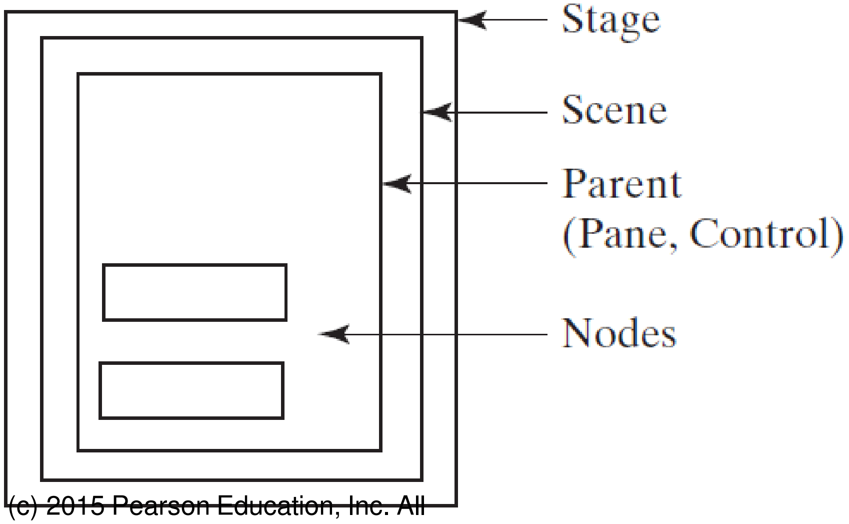
Our First JavaFX Program

- ❑ By default, stages (windows) are resizeable.
- ❑ Note that we have minimize and maximize buttons
- ❑ If we want our stage to be of fixed size (i.e., not resizeable), we can set that property with `stage.setResizeable(false)`



Panes, UI Controls, and Shapes

- In the previous examples, we put the button directly on the scene, which centered the button and made it occupy the entire window.
- Rarely is this what we really want to do
- One approach is to specify the size and location of each UI element (like the buttons)
- A better solution is to put the UI elements (known as *nodes*) into *containers* called *panes*, and then add the panes to the scene.



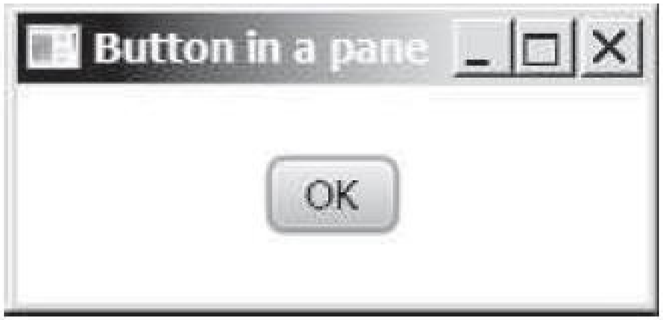
Panes, UI Controls, and Shapes

node is a visual component such as a shape, an image view, a UI control, or a pane.

shape refers to a text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.

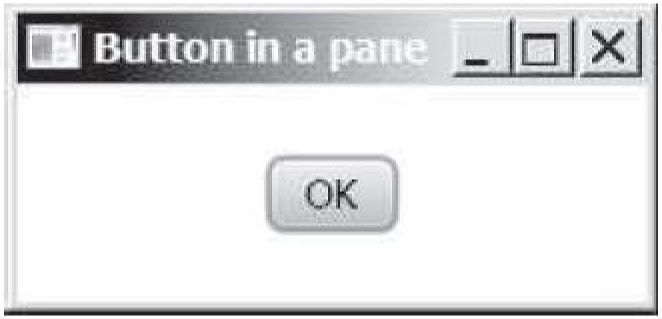
UI control refers to a label, button, check box, radio button, text field, text area, etc.

Panes, UI Controls, and Shapes



- The following slide shows the code to create this version of the same UI, with a single button inside a pane (so that the button doesn’t occupy the whole stage).
- It uses a StackPane (which we’ll discuss later).
- In order to add something to a pane, we need to access the list of things IN the pane, much like an ArrayList.
- The new item we’ll add will be a new child of the pane, so we’re adding it to the list of the pane’s children

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

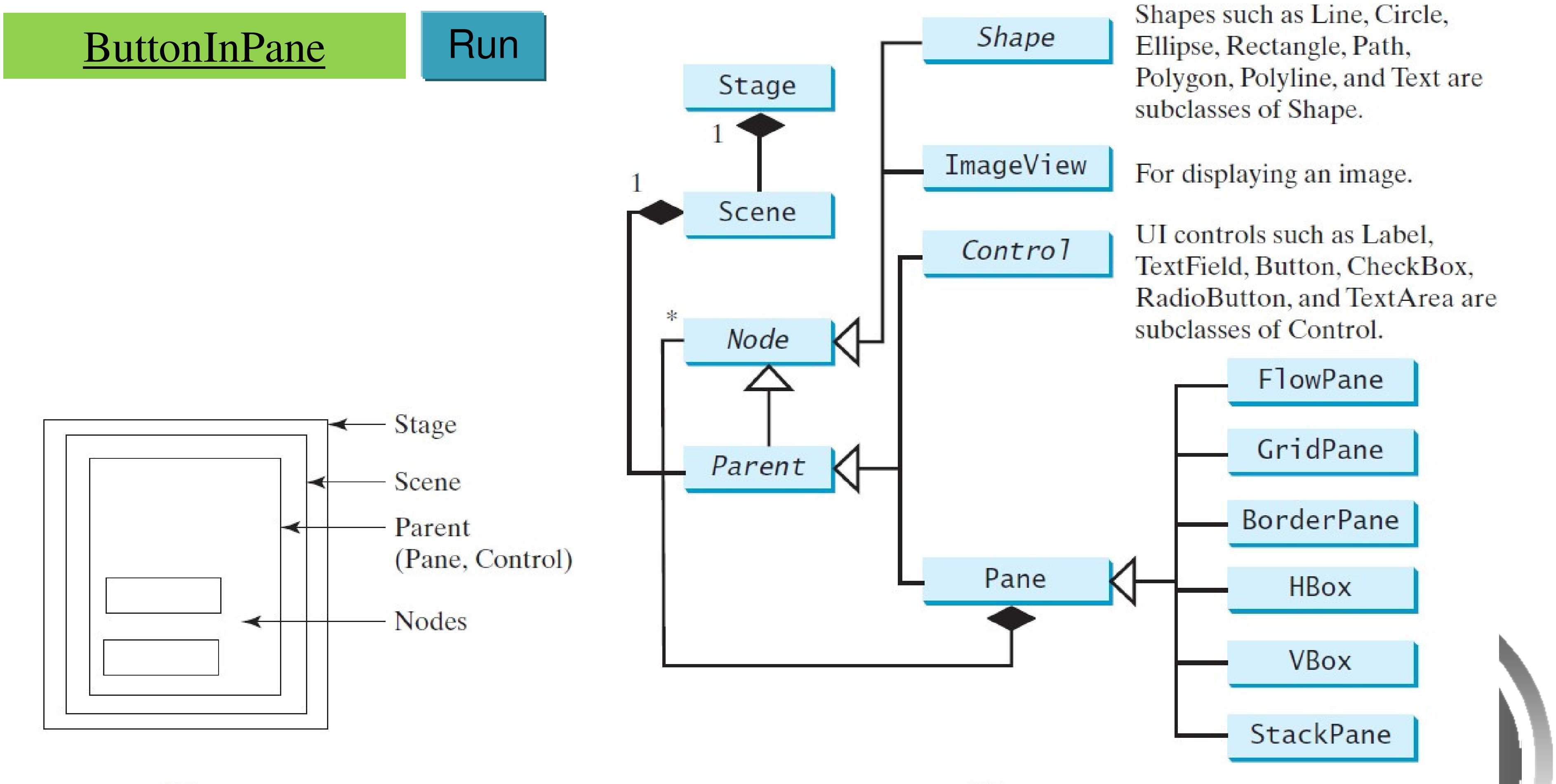


```
public class ButtonInPane extends Application {
  public void start(Stage primaryStage) {
    StackPane pane = new StackPane();
    pane.getChildren().add(new Button("OK"));
    Scene scene = new Scene(pane, 200, 50);
    primaryStage.setTitle("Button in a pane");
    primaryStage.setScene(scene);
    primaryStage.show();
  }

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

Panes, UI Controls, and Shapes

ButtonInPane

Run



Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.

For displaying an image.

UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.

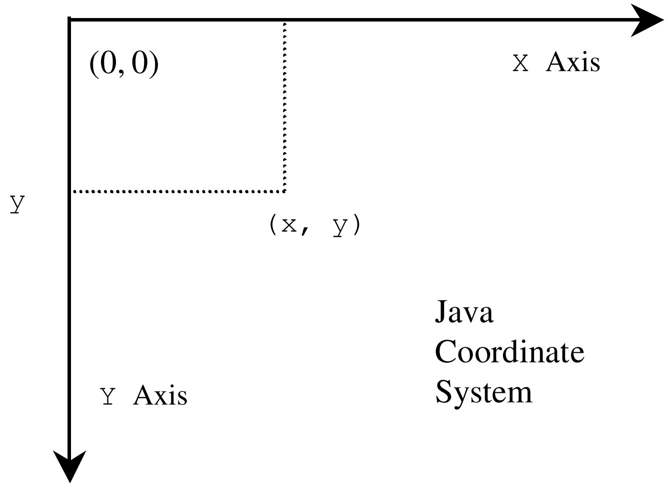
(a)

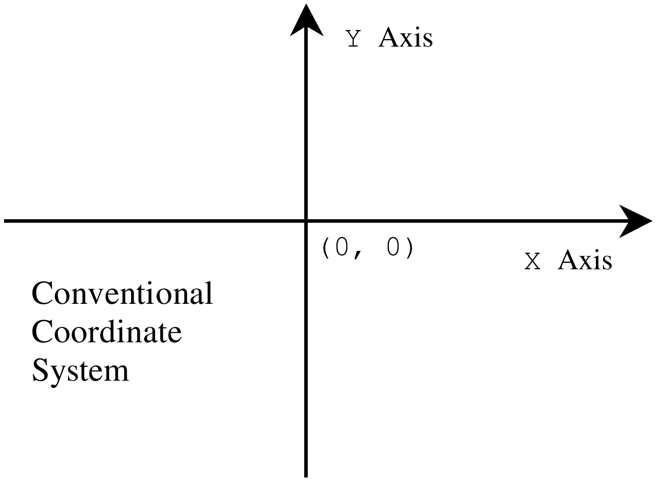
(b)

To create a Scene use one of the following:
Scene(Parent, width, height) or **Scene(Parent)**.

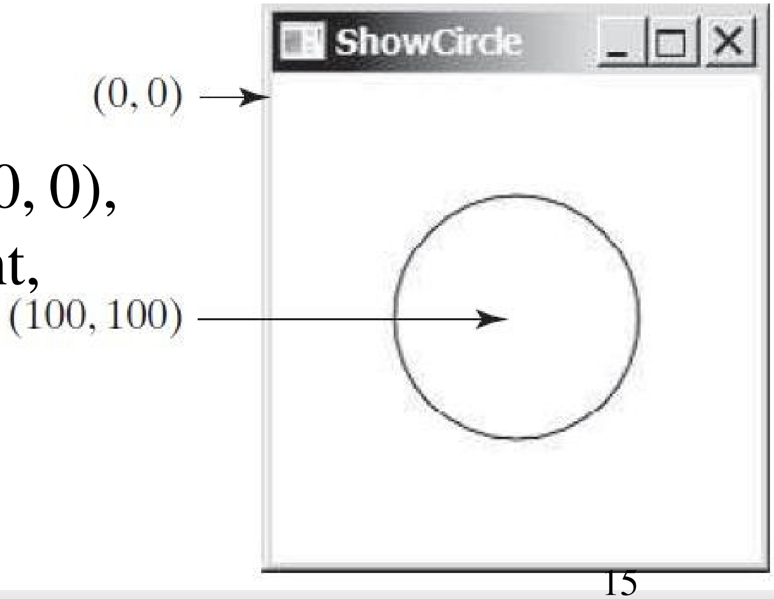
Display a Shape

This example displays a circle in the center of the pane.

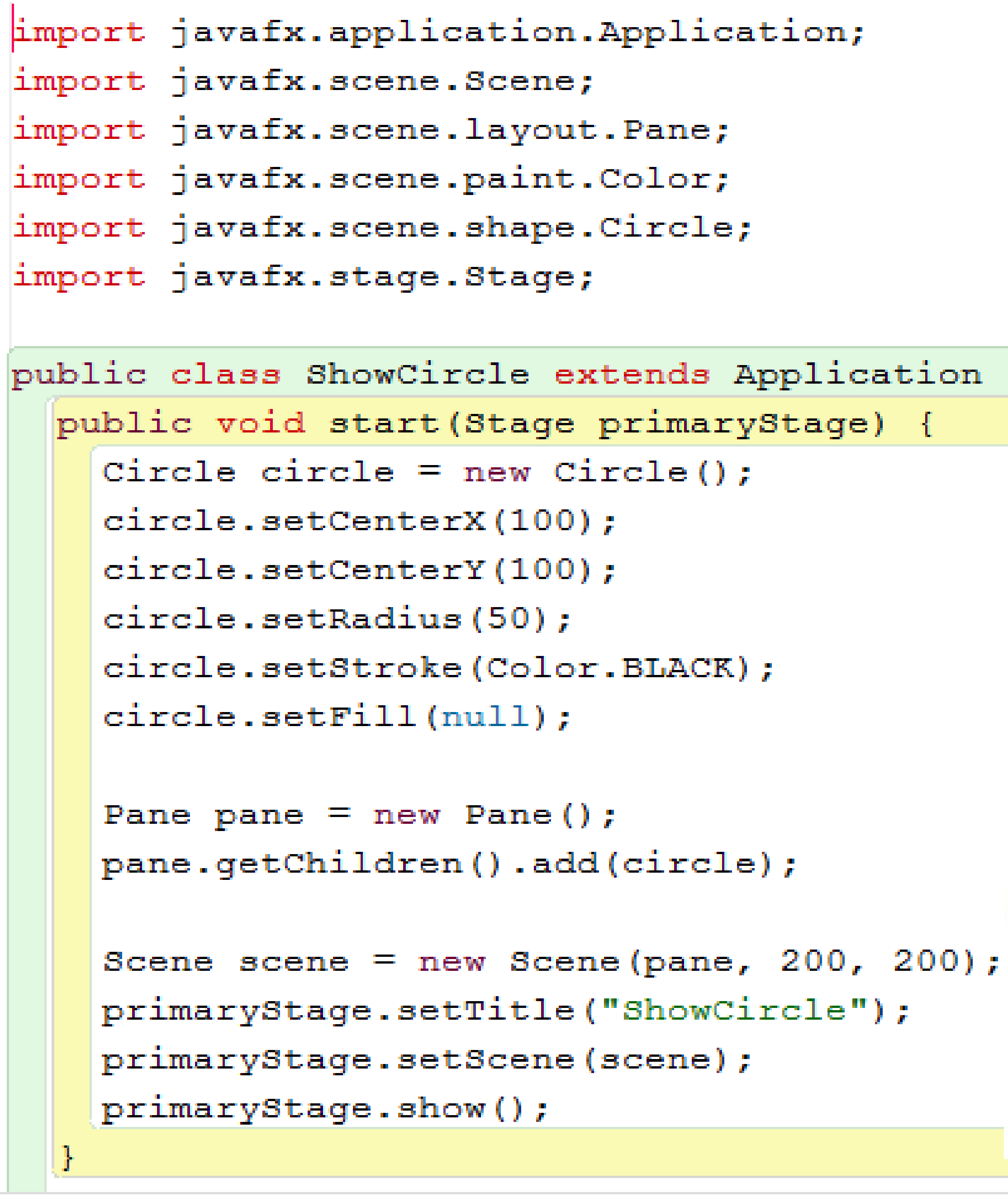




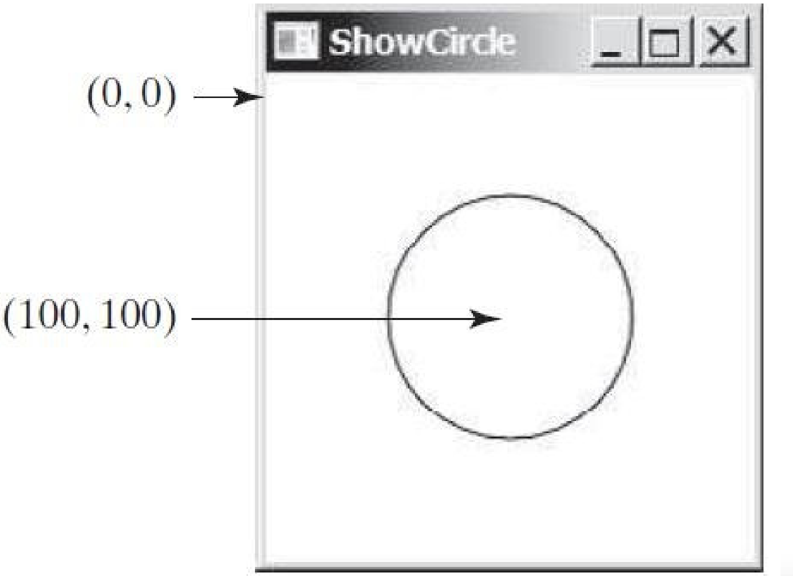
The top-left corner of a scene is *always* (0, 0), and the (positive) X-axis goes to the right, and the (positive) Y-axis goes down.



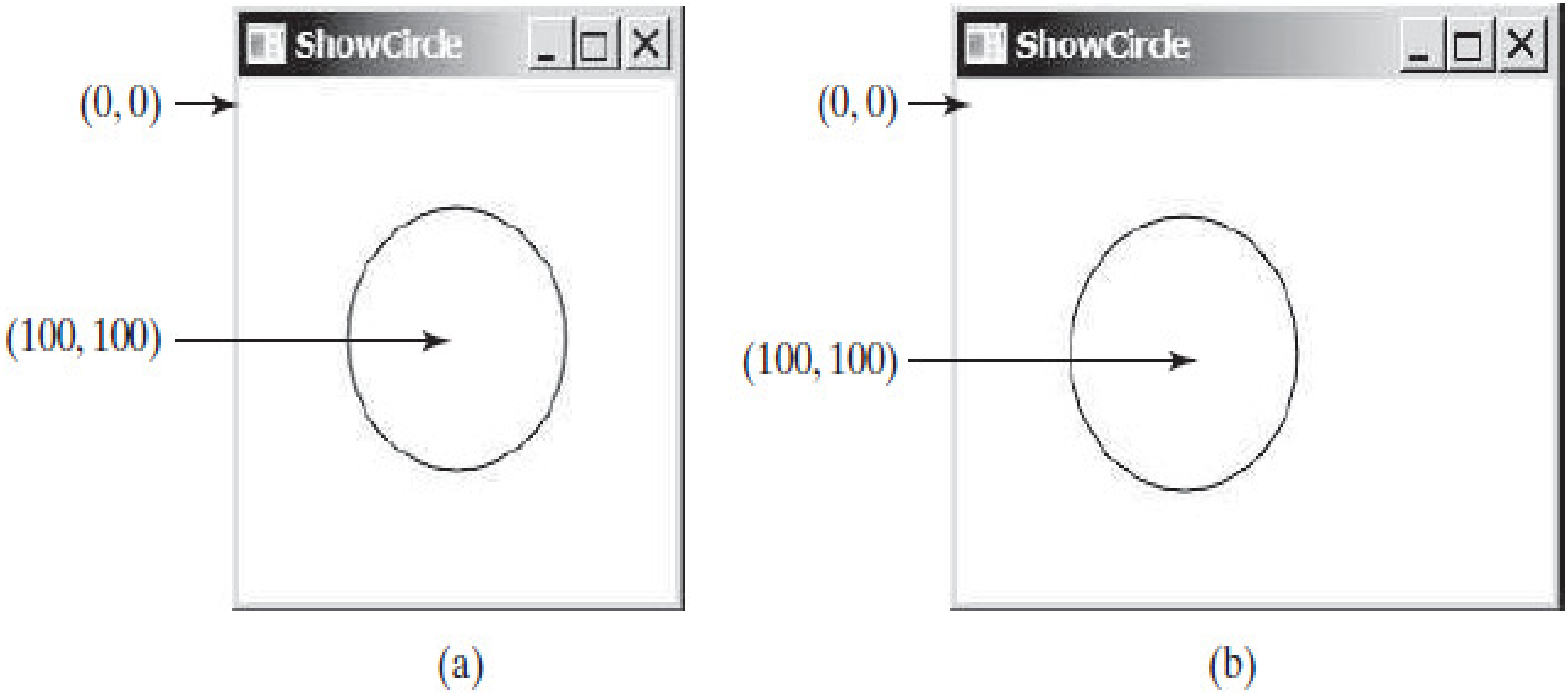
Display a Shape



```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class ShowCircle extends Application {
  public void start(Stage primaryStage) {
    Circle circle = new Circle();
    circle.setCenterX(100);
    circle.setCenterY(100);
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(null);

    Pane pane = new Pane();
    pane.getChildren().add(circle);

    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("ShowCircle");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
```



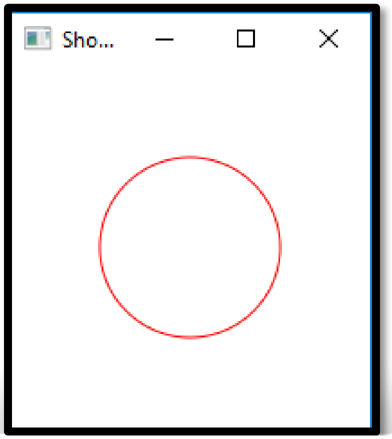
Display a Shape

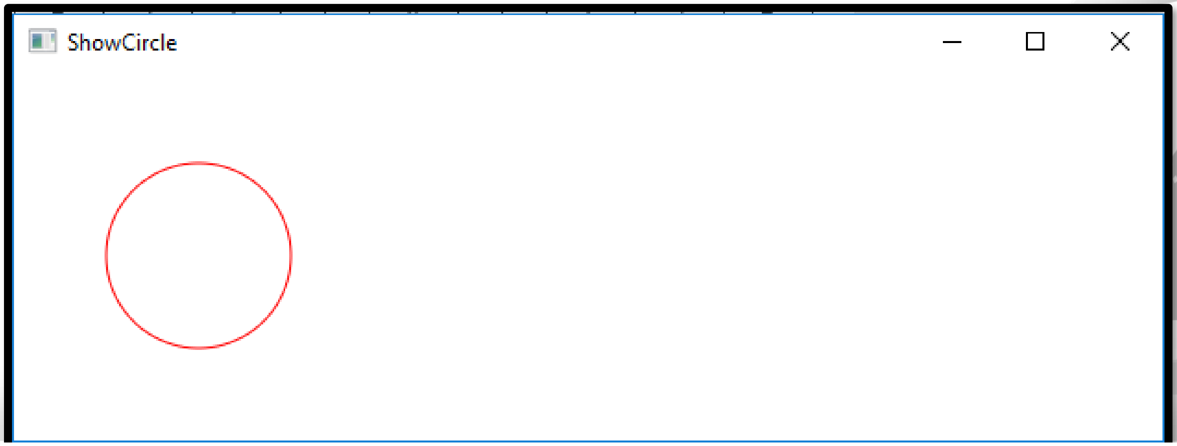


(a) A circle is displayed in the center of the scene. (b) The circle is not centered after the window is resized.

Property Binding

- In the previous example, the (x, y) location of the center of the circle was static – it will always be located at (100, 100).
- What if we want it to be centered in the pane, such that if we re-size the window, the circle will move to stay centered?
- In order to do so, the circle's center has to be *bound* to the pane's height and width, such that a change to the height or width will force a change to the x or y value of the circle's center.
- This is what *property binding* is all about

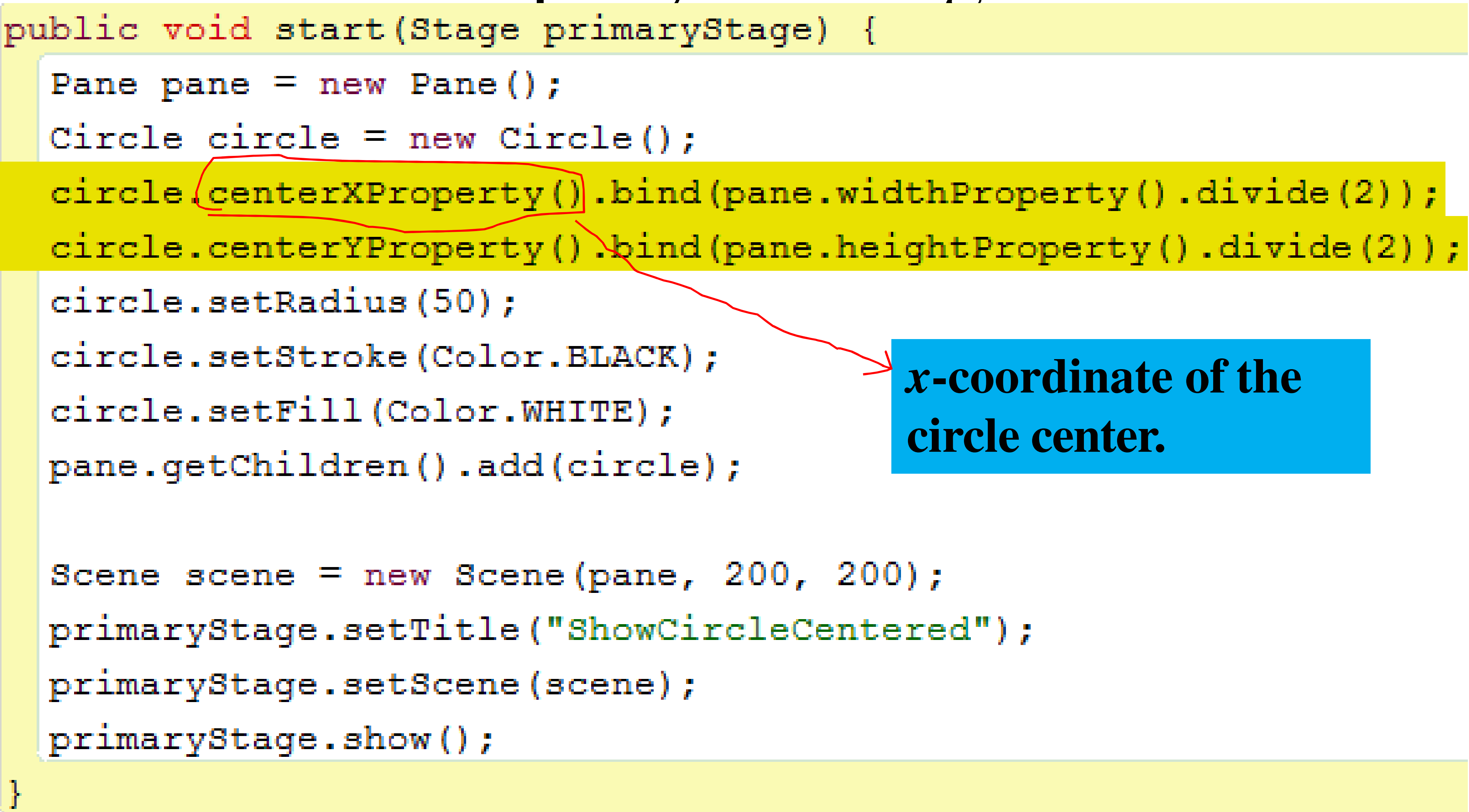




Property Binding

- The target object (called the binding object or *binding property*.) gets bound to the source object (the bindable object)
- When there’s a change to the source, it gets automatically sent to the target.
- The following Code shows how to bind the circle’s X and Y center values to the pane’s width (for clarity, I put just the Start method)

Property Binding

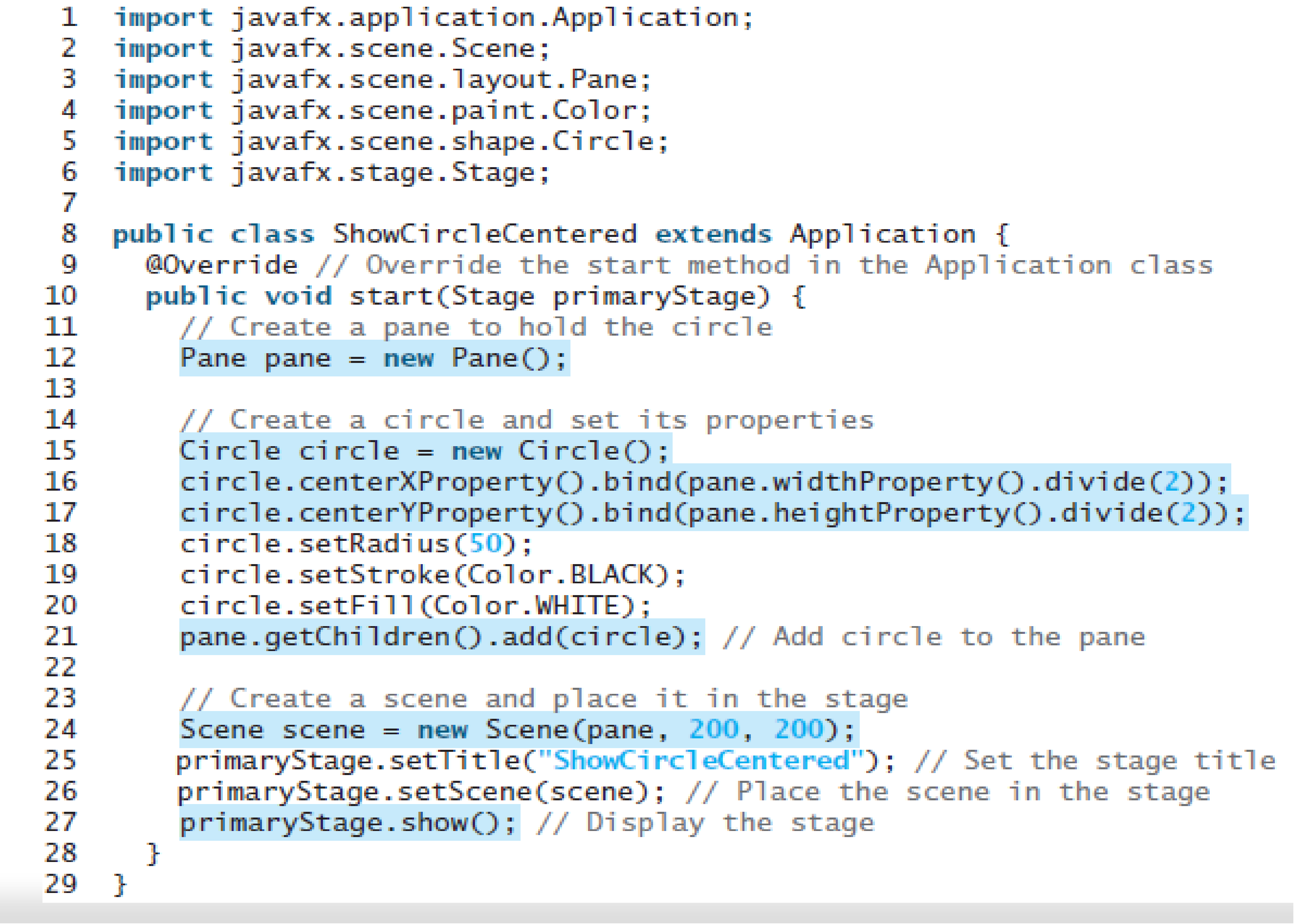


```
public void start(Stage primaryStage) {
    Pane pane = new Pane();
    Circle circle = new Circle();
    circle.centerXProperty().bind(pane.widthProperty().divide(2));
    circle.centerYProperty().bind(pane.heightProperty().divide(2));
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);
    pane.getChildren().add(circle);

    Scene scene = new Scene(pane, 200, 200);
    primaryStage.setTitle("ShowCircleCentered");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

***x*-coordinate of the circle center.**

The target listens for changes in the source and updates itself when the source changes.

the binding syntax is: **target.bind(source);**



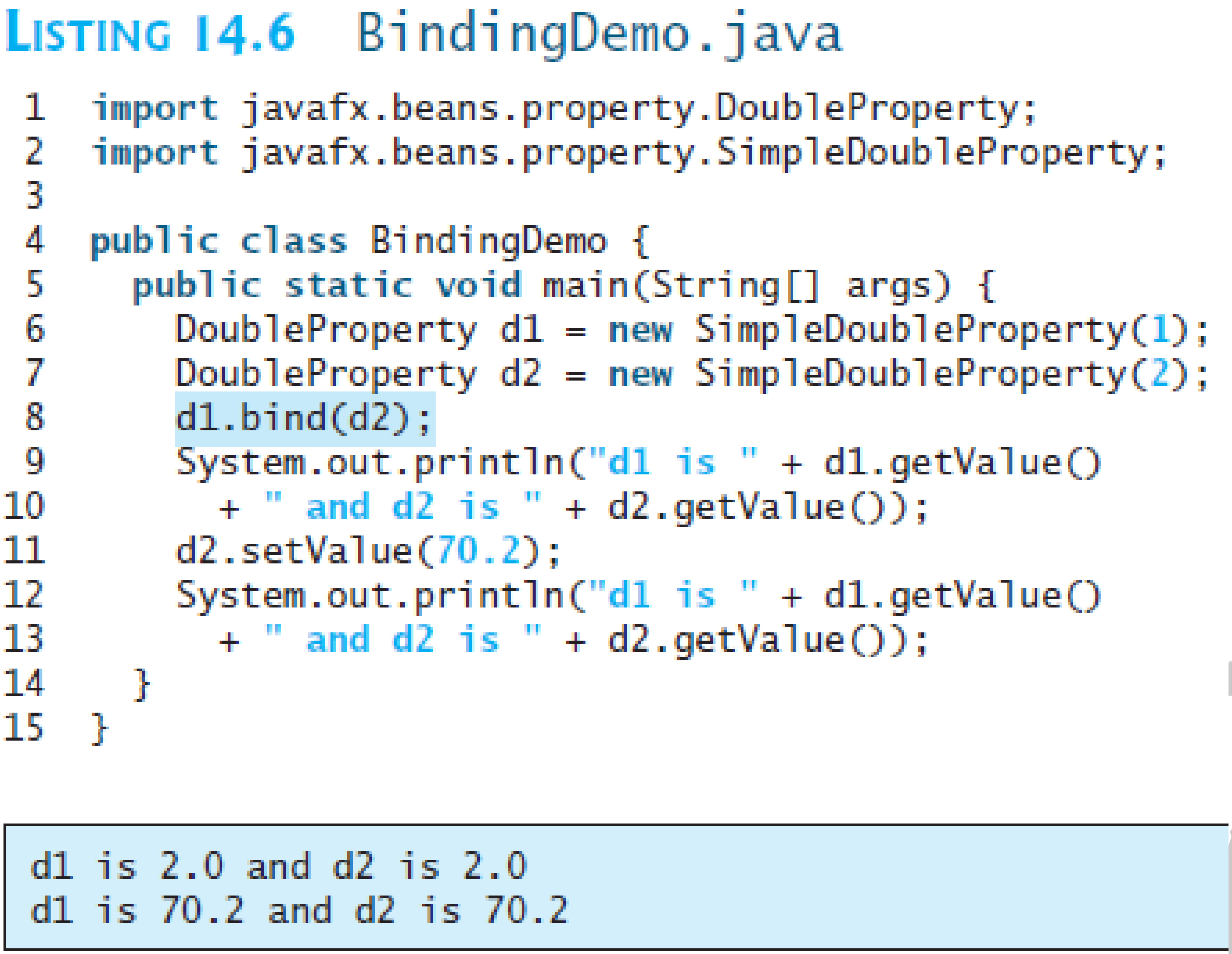
```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class ShowCircleCentered 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 Pane();

    // Create a circle and set its properties
    Circle circle = new Circle();
    circle.centerXProperty().bind(pane.widthProperty().divide(2));
    circle.centerYProperty().bind(pane.heightProperty().divide(2));
    circle.setRadius(50);
    circle.setStroke(Color.BLACK);
    circle.setFill(Color.WHITE);
    pane.getChildren().add(circle); // Add circle to the pane

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

Property Binding

- **JavaFX** defines binding properties for primitive types and strings:

Type	Binding Property Type
double	DoubleProperty
float	FloatProperty
long	LongProperty
int	IntegerProperty
boolean	BooleanProperty
String	StringProperty

Property Binding



LISTING 14.6 BindingDemo.java

```java
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class BindingDemo {
  public static void main(String[] args) {
    DoubleProperty d1 = new SimpleDoubleProperty(1);
    DoubleProperty d2 = new SimpleDoubleProperty(2);
    d1.bind(d2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
    d2.setValue(70.2);
    System.out.println("d1 is " + d1.getValue()
      + " and d2 is " + d2.getValue());
  }
}
```

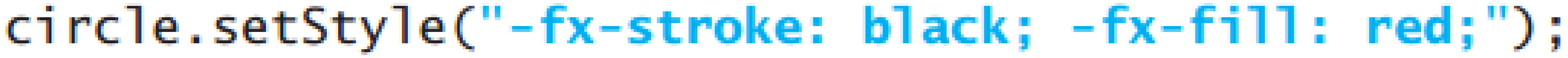
```
d1 is 2.0 and d2 is 2.0
d1 is 70.2 and d2 is 70.2
```

Property Binding

- The target listens for changes in the source and updates itself when the source changes
- Remember, the binding syntax is `target.bind(source);`
- Realize that with a setter, we specify a value; with binding, we specify the *property itself*, rather than the *value* of the property
- That's why we have to use the special methods `.add`, `.subtract`, `.multiply`, and `.divide`, rather than the numeric operators; the methods return *property objects*, rather than numeric values

Common Properties and Methods for Nodes

- The abstract **Node** class defines many properties and methods that are common to all nodes.
  - **style**: set a **JavaFX CSS** style



```
circle.setStyle("-fx-stroke: black; -fx-fill: red;");
```

This statement is equivalent to the following two statements:
circle.setStroke(Color.BLACK);
circle.setFill(Color.RED);

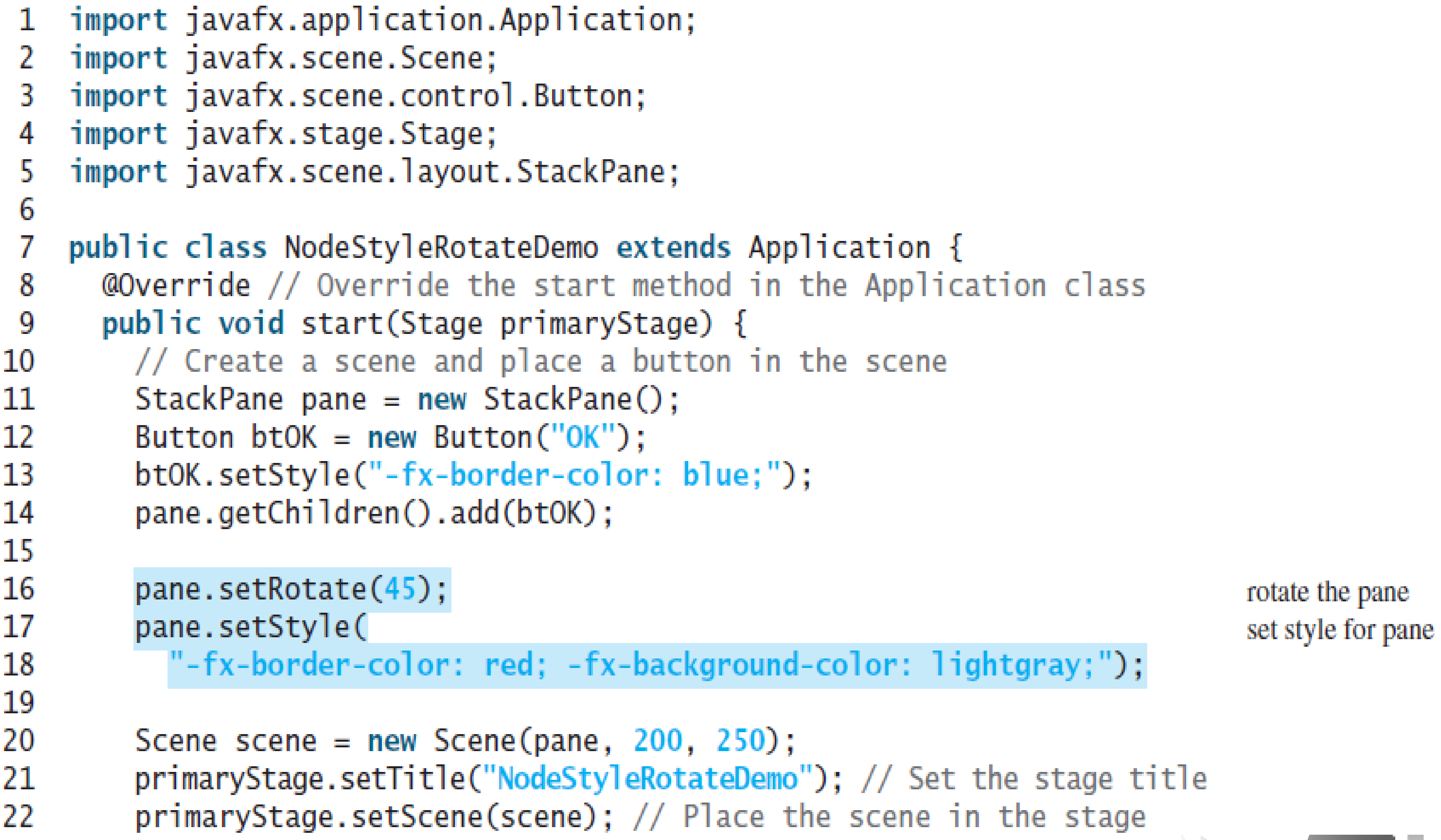
The **rotate** property

The **rotate** property enables you to specify an angle in degrees for rotating the node from its center.

If the degree is positive, the rotation is performed clockwise; otherwise, it is performed counterclockwise.

For example, the following code rotates a button 80 degrees.

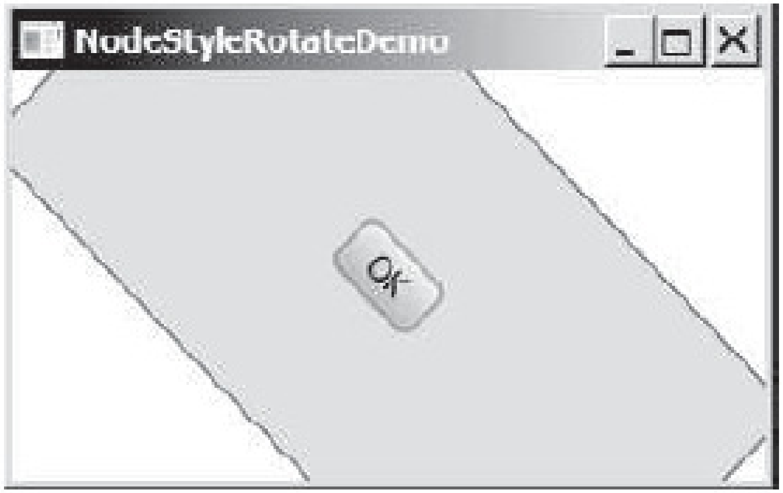
```
button.setRotate(80);
```



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

public class NodeStyleRotateDemo 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();
    Button btOK = new Button("OK");
    btOK.setStyle("-fx-border-color: blue;");
    pane.getChildren().add(btOK);

    pane.setRotate(45);
    pane.setStyle(
      "-fx-border-color: red; -fx-background-color: lightgray;");

    Scene scene = new Scene(pane, 200, 250);
    primaryStage.setTitle("NodeStyleRotateDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show();
  }
}
```

rotate the pane

set style for pane



A pane's style is set and it is rotated 45 degrees.

The border color of this pane is RED

The border color of the button is Blue

contains method

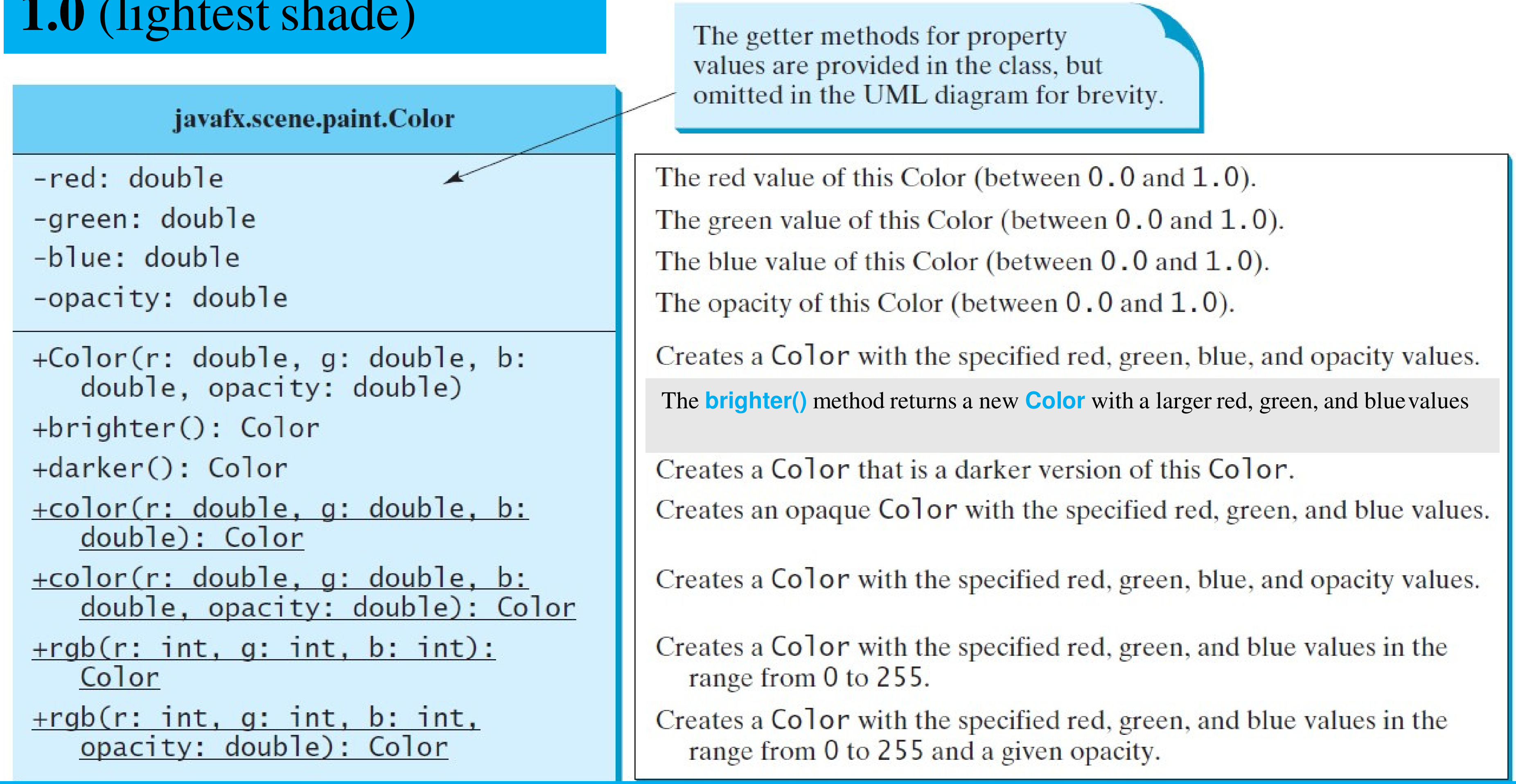
In the **Node** class, there is a method called contains:

contains(double x, double y)

This method returns true if the given point (*x*, *y*) is contained within the boundary of a node.

The Color Class

from **0.0** (darkest shade) to **1.0** (lightest shade)



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`	The **brighter()** method returns a new **Color** with a larger red, green, and blue values
`+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 **opacity** value defines the transparency of a color within the range from **0.0** (completely transparent) to **1.0** (completely opaque).

The Color Class

A color instance can be constructed using the following constructor:

```
public Color(double r, double g, double b, double opacity);
```

in which **r**, **g**, and **b** specify a color by its red, green, and blue components with values in the range from **0.0** (darkest shade) to **1.0** (lightest shade). The **opacity** value defines the transparency of a color within the range from **0.0** (completely transparent) to **1.0** (completely opaque). This is known as the RGBA model, where RGBA stands for red, green, blue, and alpha. The alpha value indicates the opacity. For example:

```
Color color = new Color(0.25, 0.14, 0.333, 0.51);
```

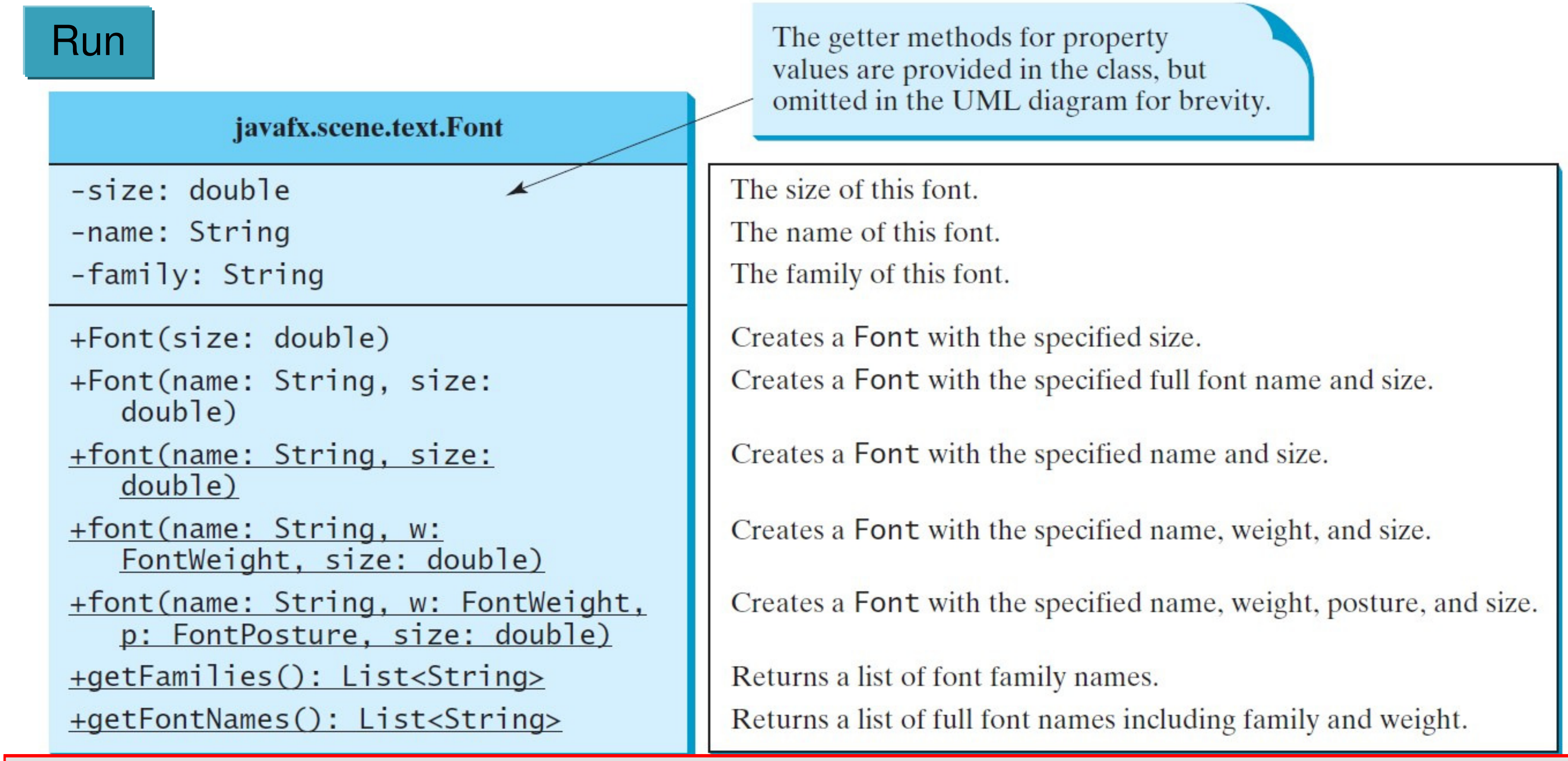
Alternatively, you can use one of the many standard colors such as **BLACK**, **BLUE**,**BROWN**, **CYAN**, **DARKGRAY**, **GOLD**, **GRAY**, **GREEN** defined as constants in the **Color** class.

```
circle.setFill(Color.RED);
```

FontDemo

Run

The Font Class



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

javafx.scene.text.Font

-size: double	The size of this font.
-name: String	The name of this font.
-family: String	The family of this font.
+Font(size: double)	Creates a Font with the specified size.
+Font(name: String, size: double)	Creates a Font with the specified full font name and size.
+font(name: String, size: double)	Creates a Font with the specified name and size.
+font(name: String, w: FontWeight, size: double)	Creates a Font with the specified name, weight, and size.
+font(name: String, w: FontWeight, p: FontPosture, size: double)	Creates a Font with the specified name, weight, posture, and size.
+getFamilies(): List<String>	Returns a list of font family names.
+getFontNames(): List<String>	Returns a list of full font names including family and weight.

You can get a listing of all of the font family names installed on the computer with .getFamilies()

FontPosture, however, comes in exactly two flavors: REGULAR and ITALIC

```
Font font1 = new Font("SansSerif", 16);
Font font2 = Font.font("Times New Roman", FontWeight.BOLD,FontPosture.ITALIC, 12);
```

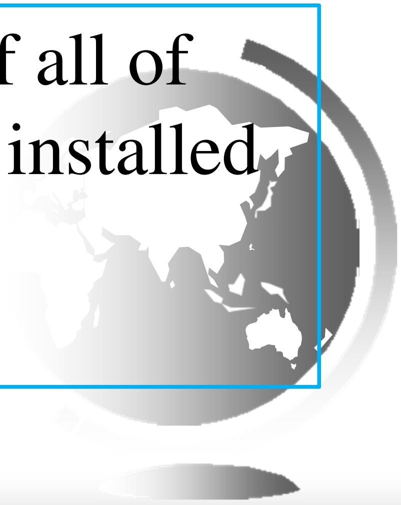
The Font Class

- We typically use just “NORMAL”or “BOLD” for the FontWeights:

THIN
EXTRA_LIGHT
LIGHT
NORMAL
MEDIUM
SEMI_BOLD
BOLD
EXTRA_BOLD
BLACK

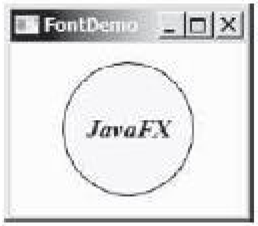
FontPosture, however, comes in exactly two flavors: REGULAR and ITALIC

You can get a listing of all of the font family names installed on the computer with .getFamilies()



```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.scene.text.*;
import javafx.scene.control.*;
import javafx.stage.Stage;
```

```
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
  }
}
```



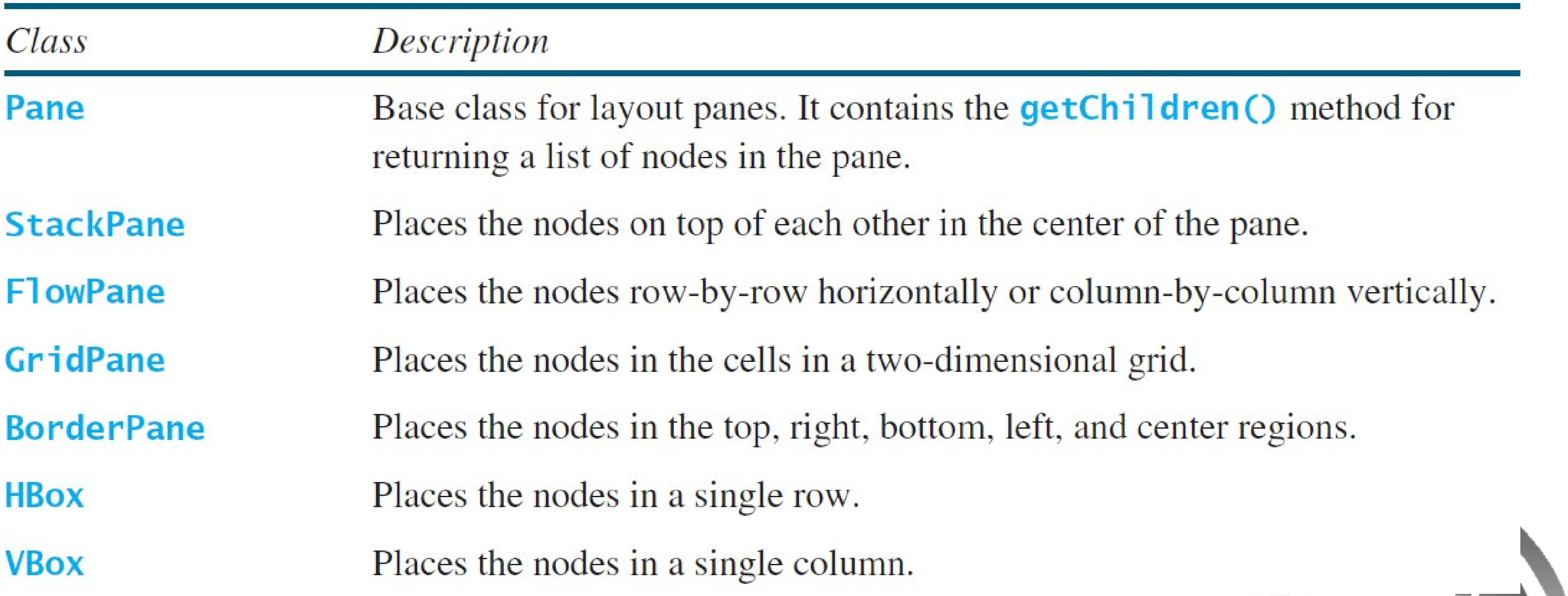
A label is on top of a circle displayed in the center of the scene.

pane.getChildren().addAll(circle, label);

A **StackPane** places the nodes in the center and nodes are placed on top of each other.

Layout Panes

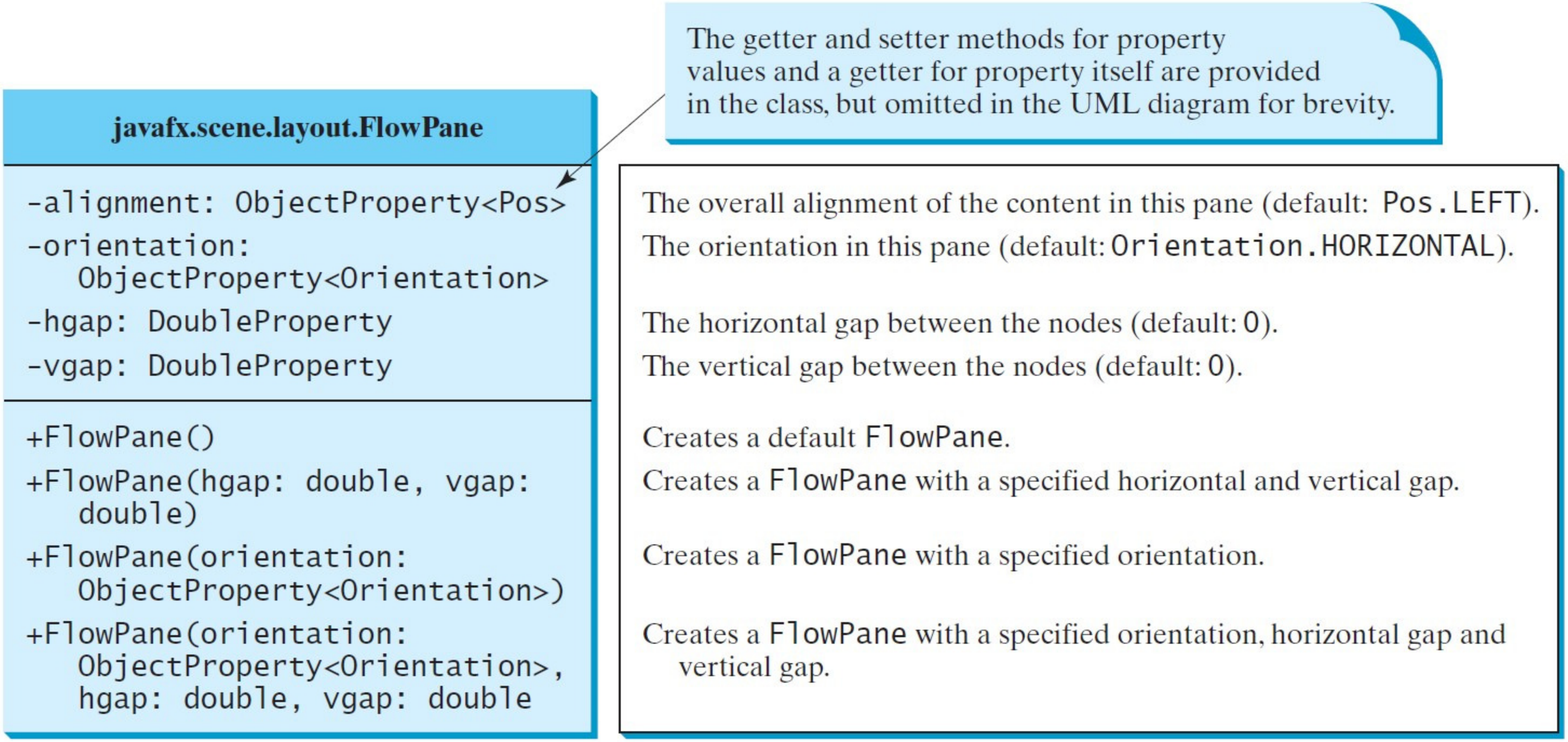
JavaFX provides many types of panes for organizing nodes in a container.



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.

Pane: Base class for layout panes. Use its `getChildren()` method to return the list of nodes on the pane (or add to that list)
Provides no particular layout capabilities – it's a "blank canvas" typically used to draw shapes on

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.

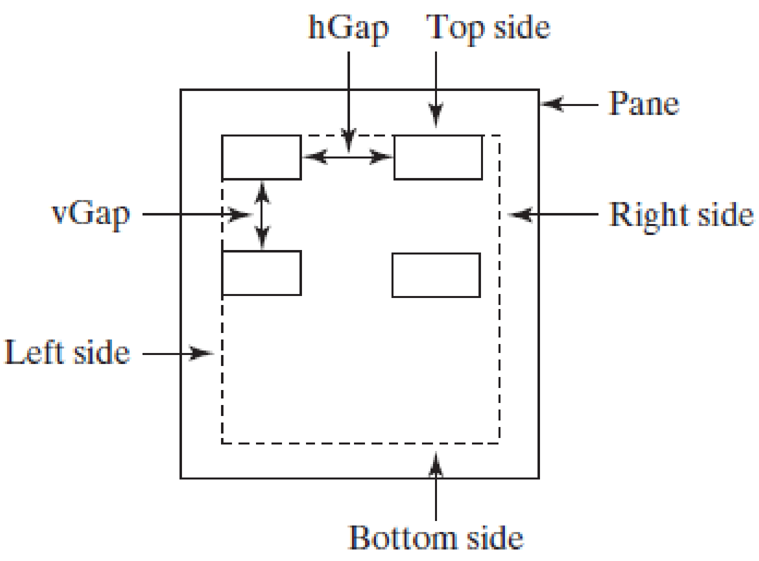
Creates a `FlowPane` with a specified orientation.

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

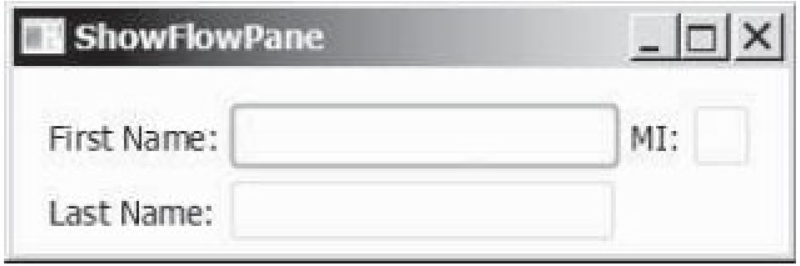
FlowPane arranges the nodes in the pane horizontally from left to right or
vertically from top to bottom in the order in which they were added.
When one row or one column is filled, a new row or column is started.

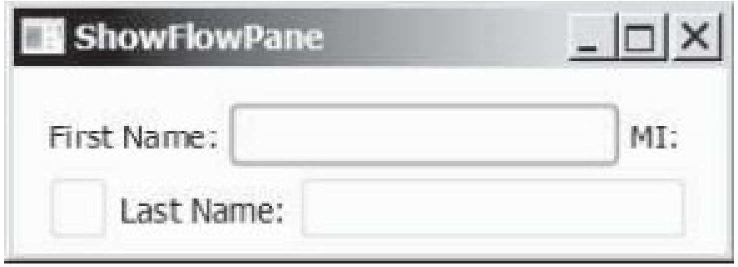
FlowPane

- You can specify the way the nodes are placed horizontally or vertically using one of two constants: **Orientation.HORIZONTAL** or **Orientation.VERTICAL**. You can also specify the gap between the nodes in pixels.



You can specify hGap and vGap between the nodes in a FlowLPane.





To add nodes : add(node) or **addAll(node1, node2, ...)** method. **To remove nodes:remove(node)** or use the **removeAll()** method to remove all nodes from the pane

```java
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

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
  }
}
```

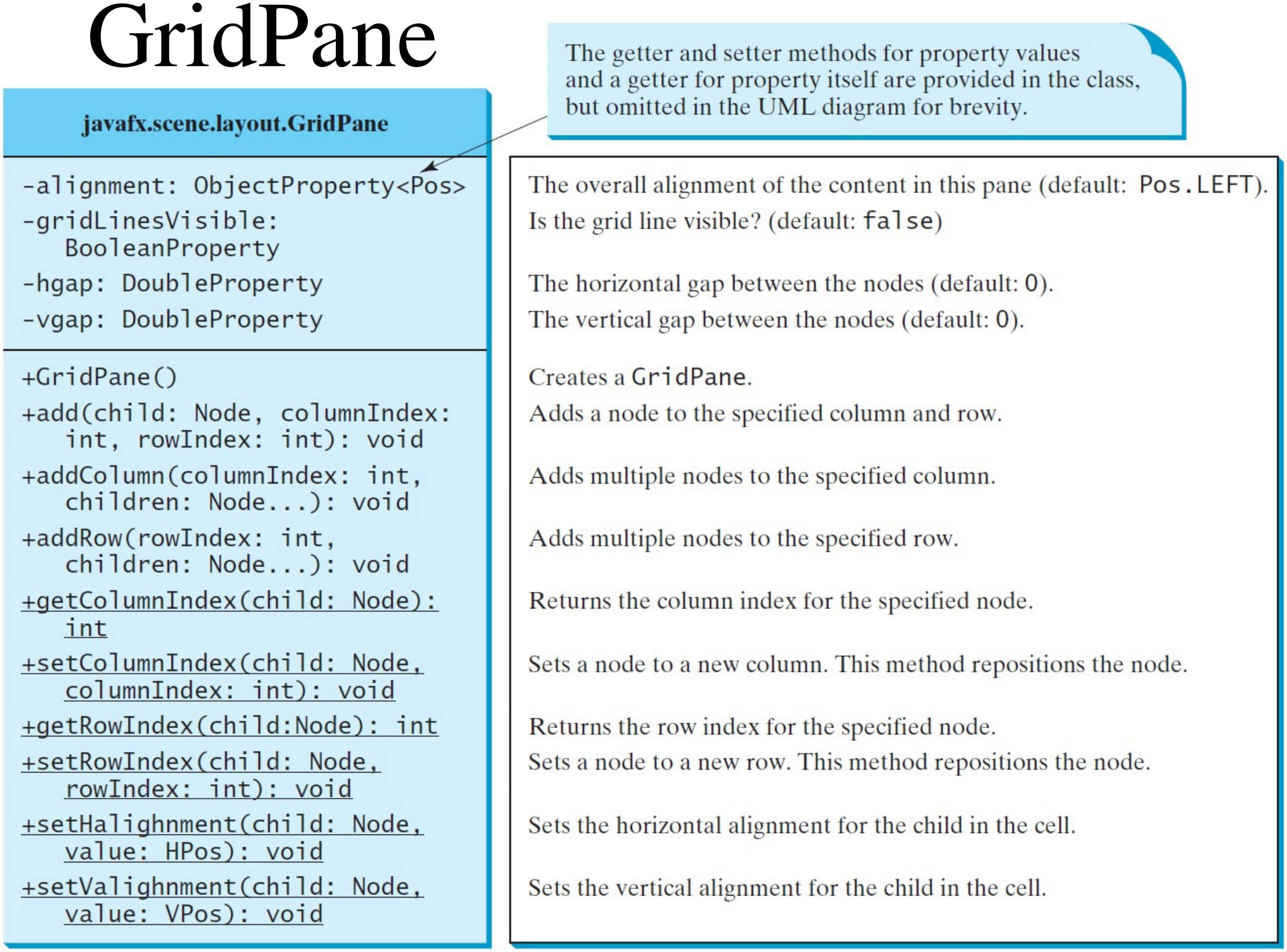
padding property

padding property with an **Insets:**

An **Insets** object specifies the size of the border of a pane. The constructor **Insets(11, 12, 13, 14)** creates an **Insets** with the border sizes for top (11), right (12), bottom (13), and left (14) in pixels.

You can also use the constructor **Insets(value)** to create an **Insets** with the same value for all four sides.

GridPane

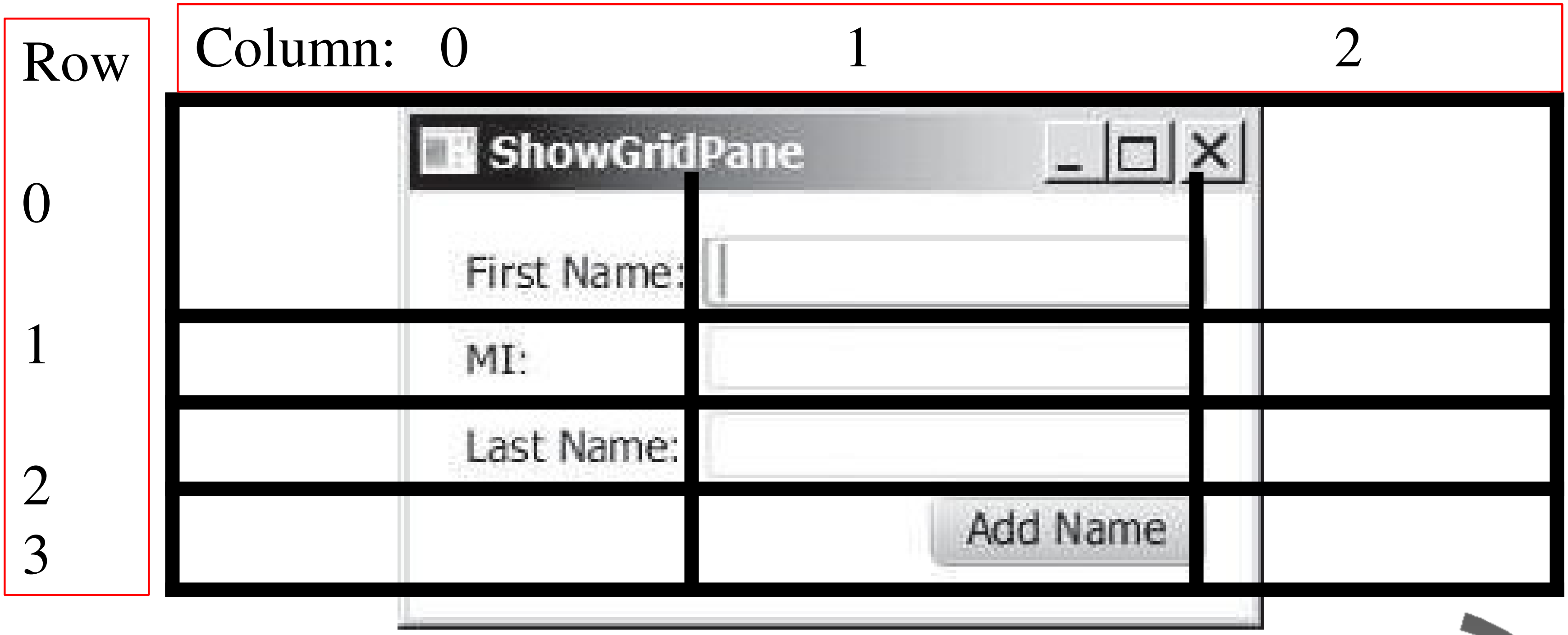


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

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.

A **GridPane** arranges nodes in a grid (matrix) formation. The nodes are placed in the specified column and row indices.

GridPane

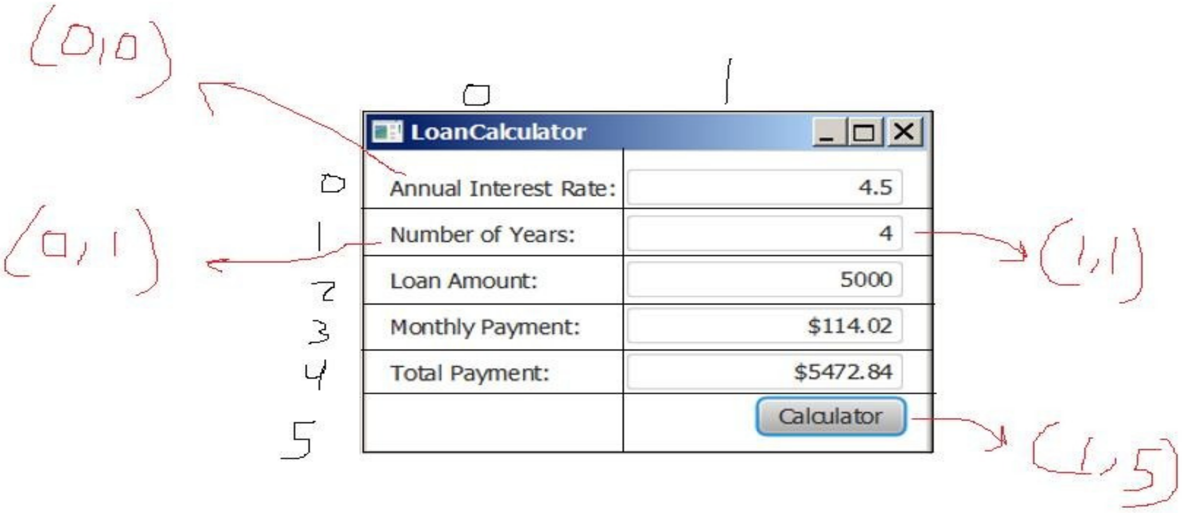


```
pane.add(new Label("First Name:"), 0, 0);
```

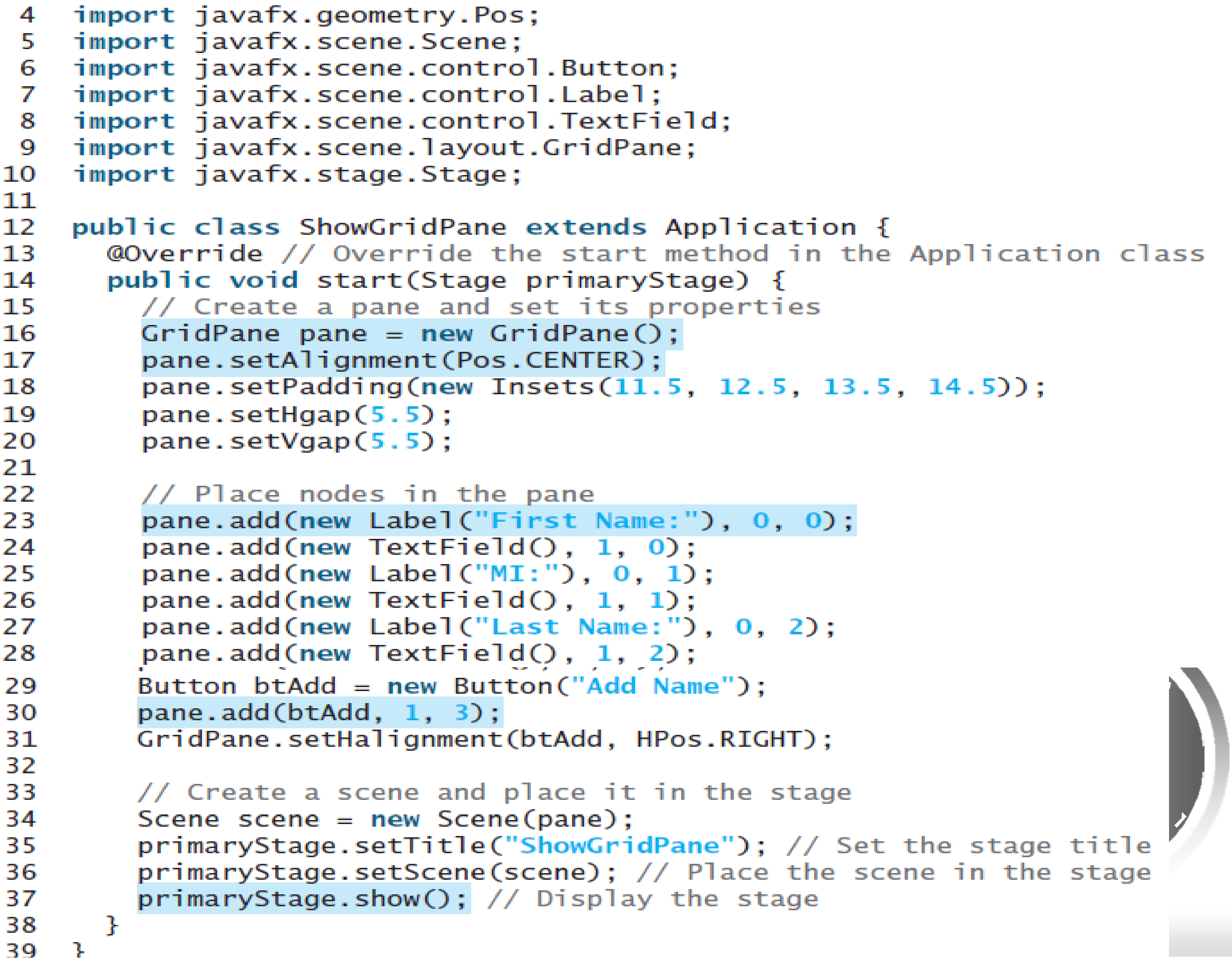
column

row

GridPane

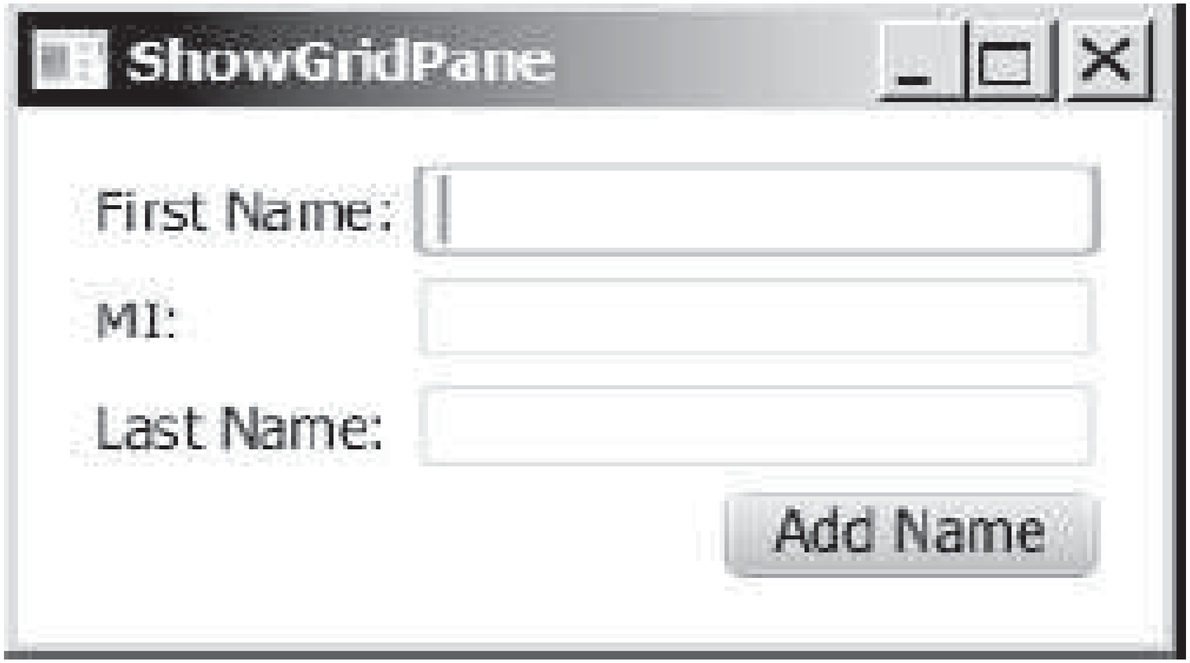


Annual Interest Rate:	4.5
Number of Years:	4
Loan Amount:	5000
Monthly Payment:	$114.02
Total Payment:	$5472.84
	Calculator



```
import javafx.geometry.Pos;
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 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
  }
}
```

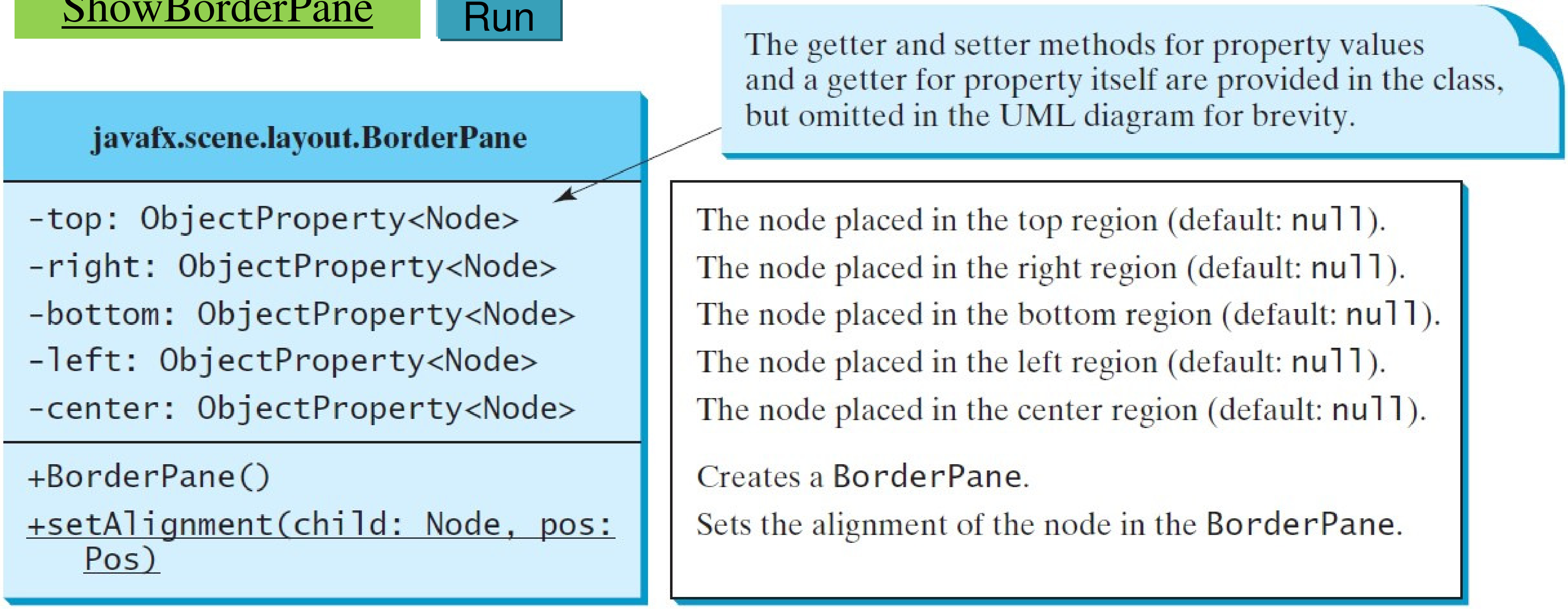
GridPane



BorderPane

ShowBorderPane

Run



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`).

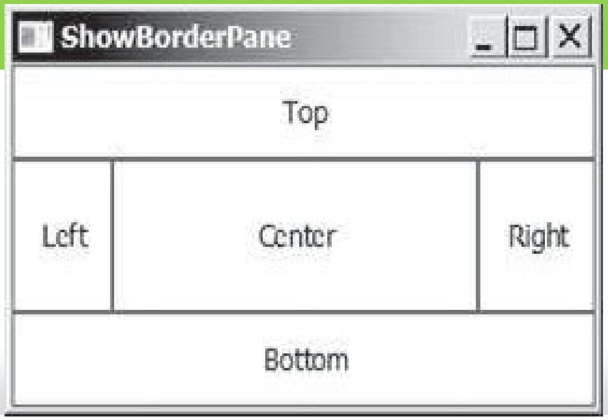
The node placed in the left region (default: `null`).

The node placed in the center region (default: `null`).

Creates a `BorderPane`.

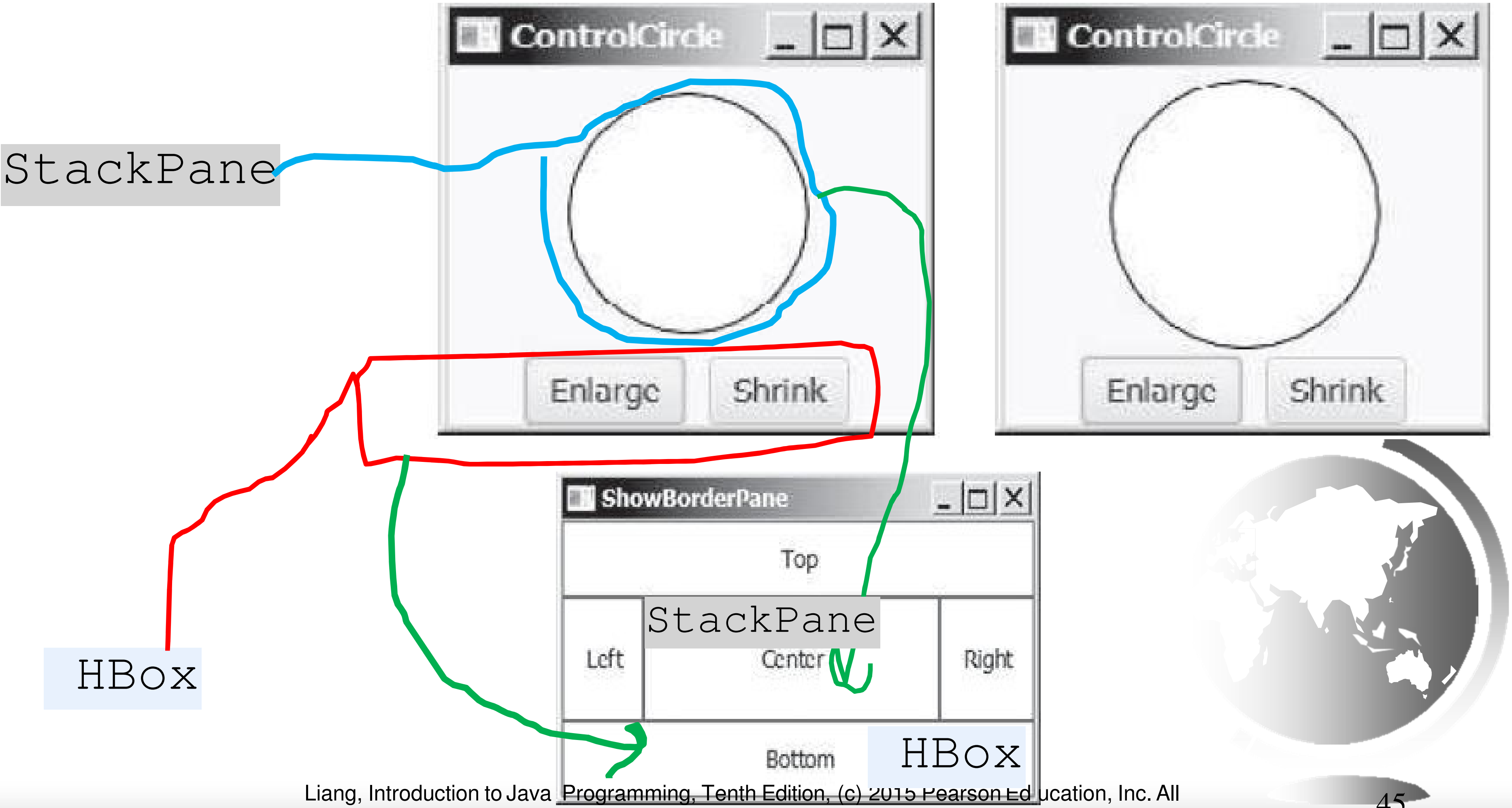
Sets the alignment of the node in the `BorderPane`.

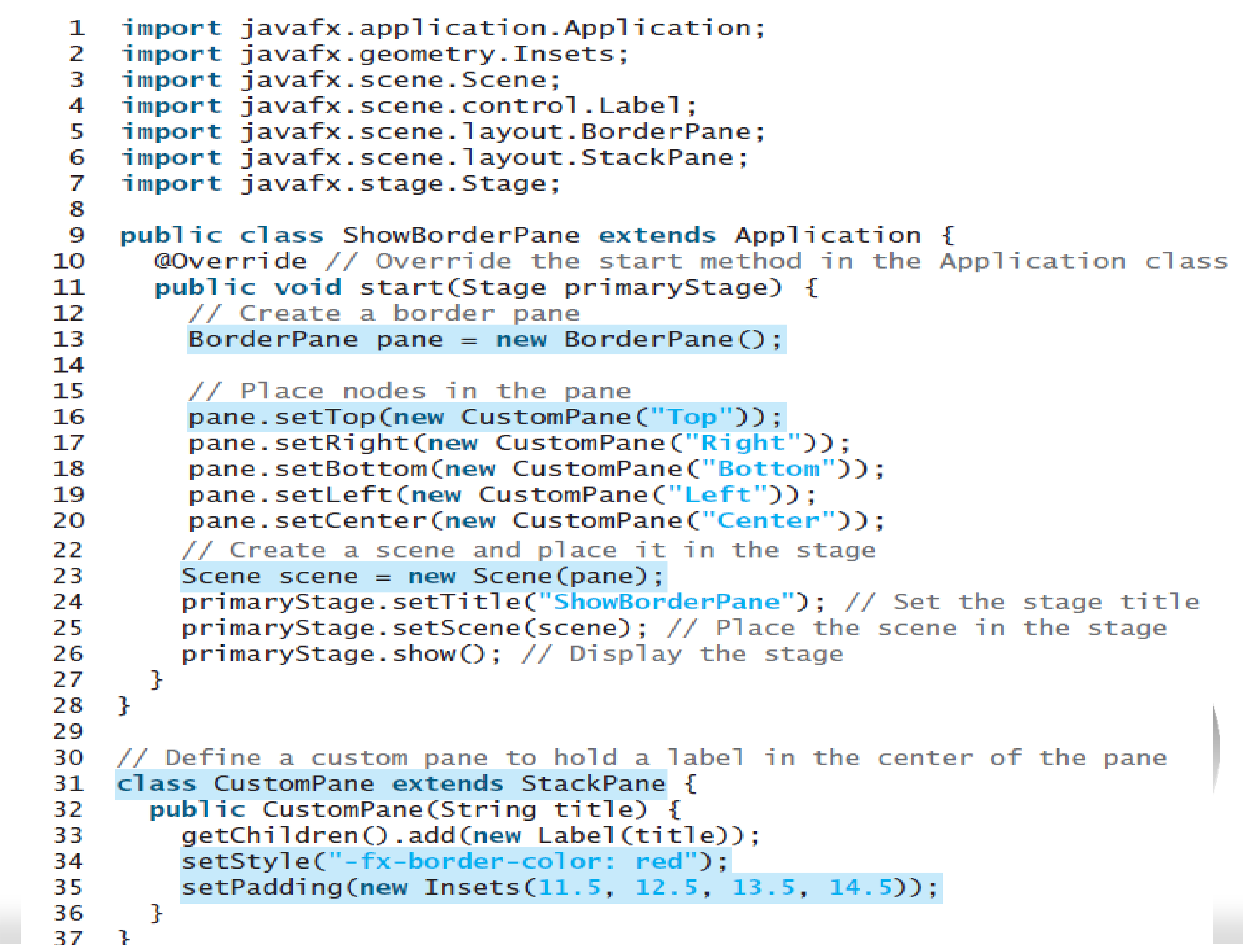
A **BorderPane** can place nodes in five regions: top, bottom, left, right, and center, using the **setTop(node)**, **setBottom(node)**, **setLeft(node)**, **setRight(node)**, and **setCenter(node)** methods.



BorderPane

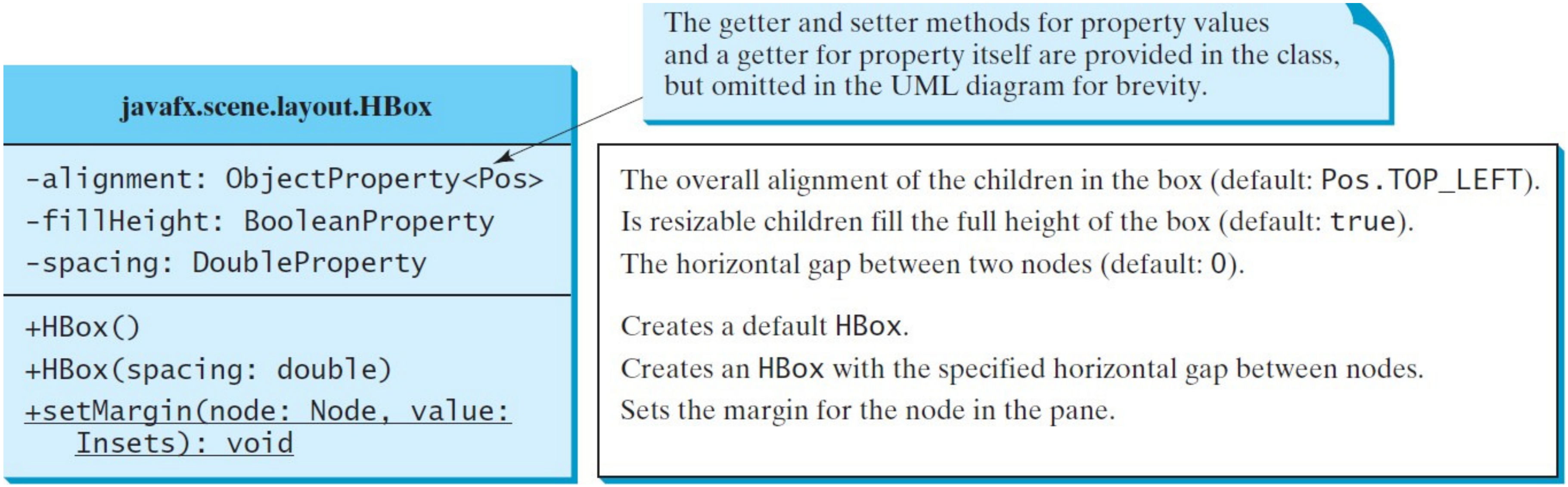
A **BorderPane** can place nodes in five regions: top, bottom, left, right, and center, using the **setTop(node)**, **setBottom(node)**, **setLeft(node)**, **setRight(node)**, and **setCenter(node)** methods.





```java
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

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));
  }
}
```

HBox



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.HBox

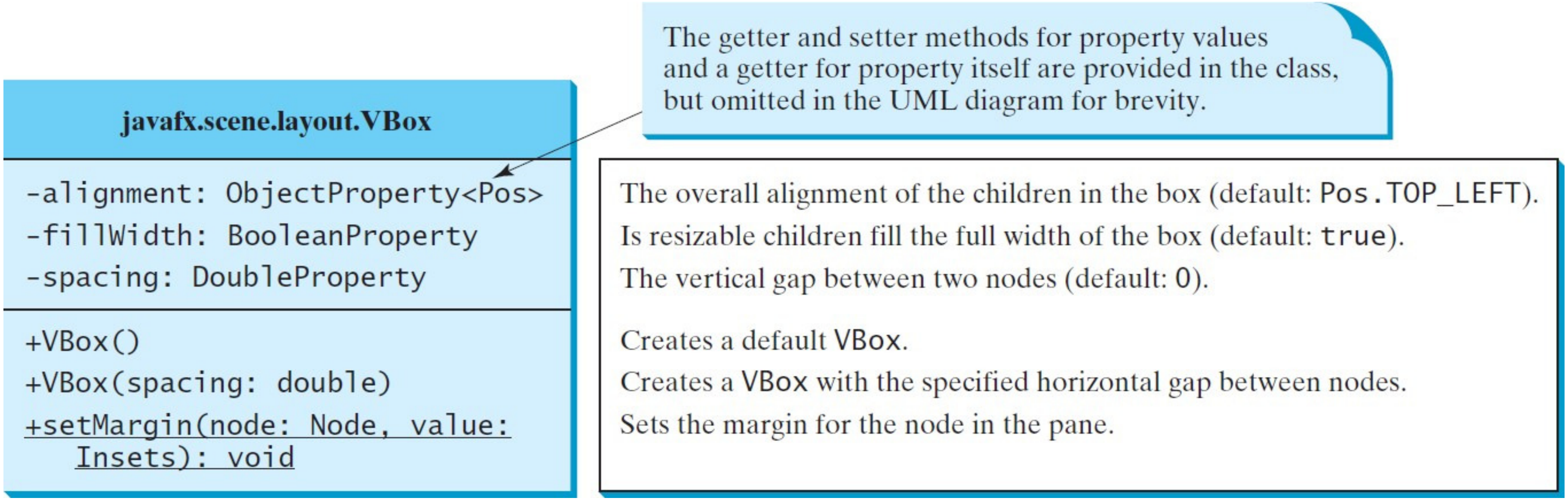
```
-alignment: ObjectProperty<Pos>
-fillHeight: BooleanProperty
-spacing: DoubleProperty

+HBox()
+HBox(spacing: double)
+setMargin(node: Node, value:
   Insets): void
```

The overall alignment of the children in the box (default: Pos.TOP_LEFT).
Is resizable children fill the full height of the box (default: true).
The horizontal gap between two nodes (default: 0).

Creates a default HBox.
Creates an HBox with the specified horizontal gap between nodes.
Sets the margin for the node in the pane.

- ❑ The HBox is a pane that handles placement of multiple nodes for us automatically.
- ❑ As we add nodes to the HBox, they are automatically added in a row (horizontally)

VBox



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.VBox

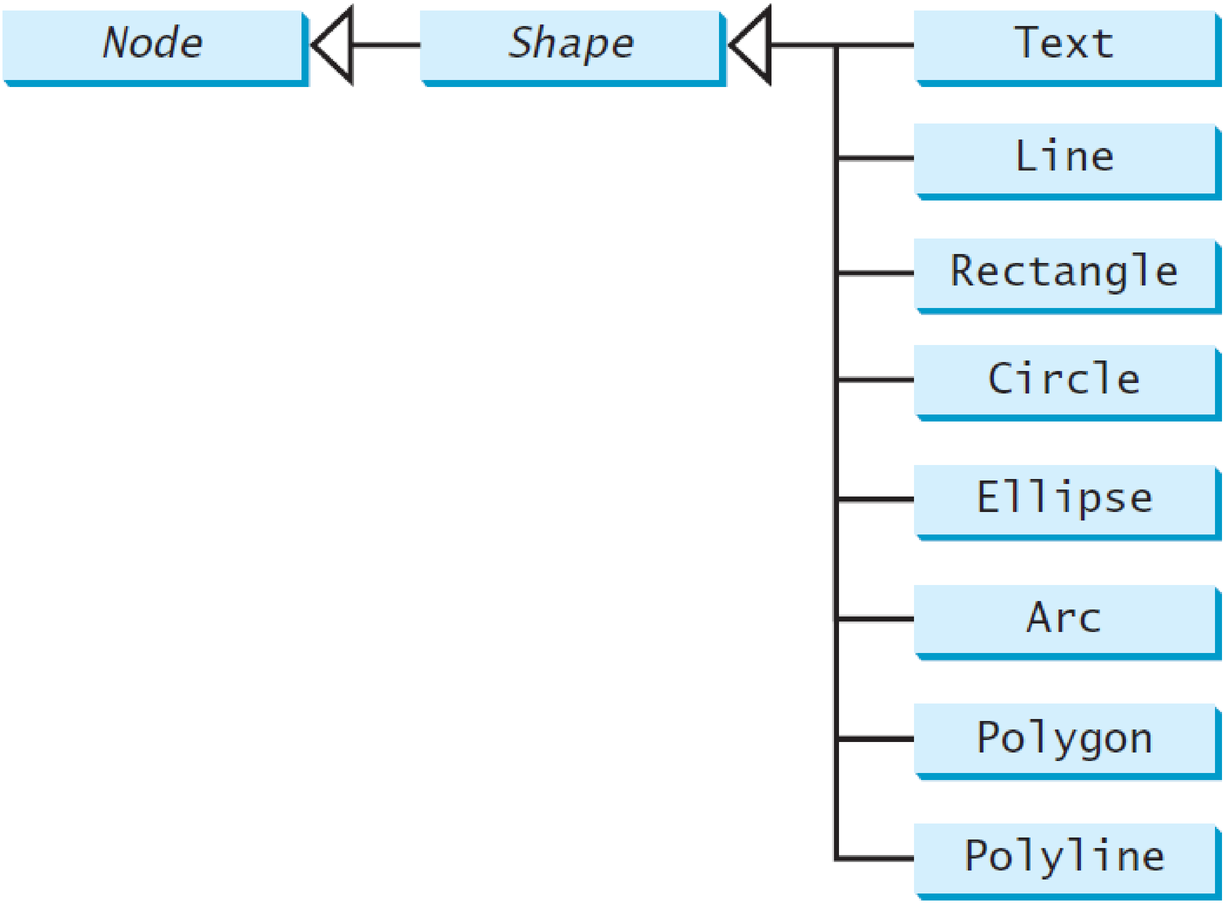
-alignment: ObjectProperty<Pos>	The overall alignment of the children in the box (default: Pos.TOP_LEFT).
-fillWidth: BooleanProperty	Is resizable children fill the full width of the box (default: true).
-spacing: DoubleProperty	The vertical gap between two nodes (default: 0).
+VBox()	Creates a default VBox.
+VBox(spacing: double)	Creates a VBox with the specified horizontal gap between nodes.
+setMargin(node: Node, value: Insets): void	Sets the margin for the node in the pane.

ShowHBoxVBox

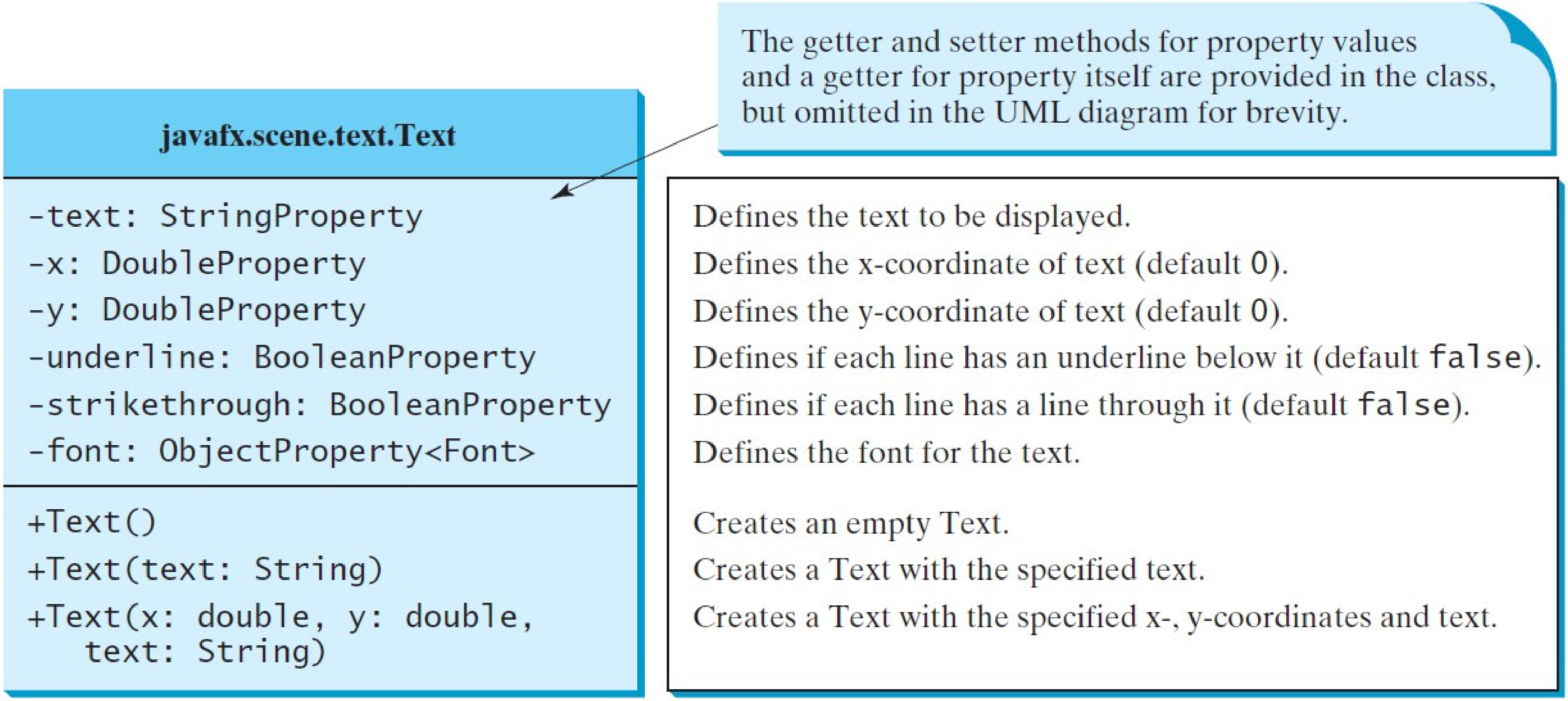
Run

Shapes

JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.



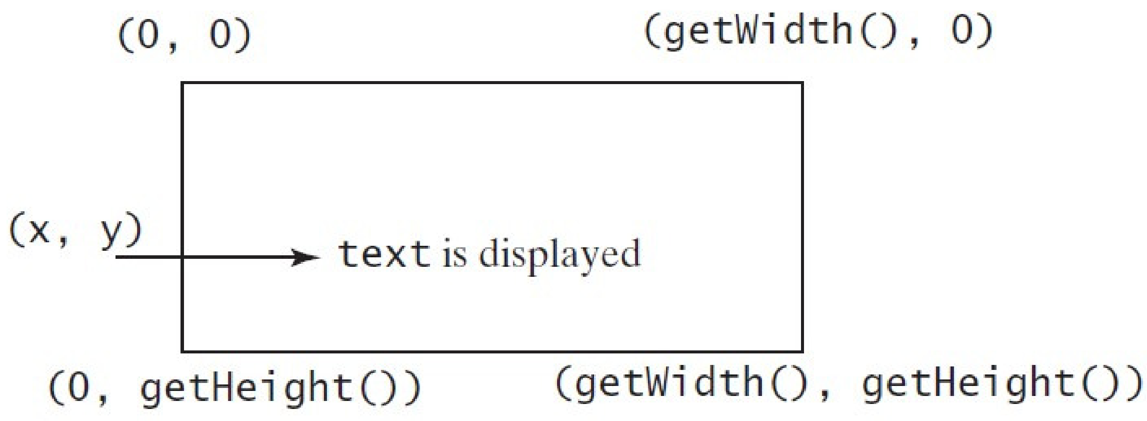
Text



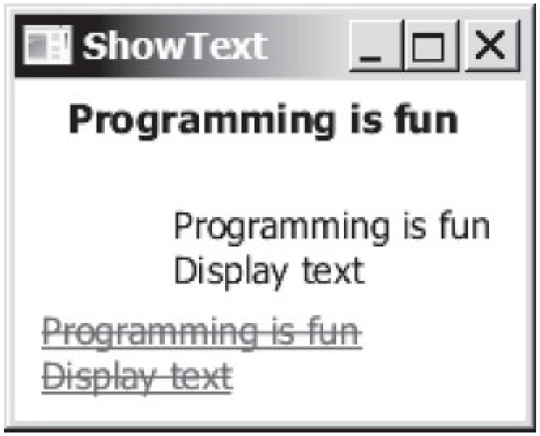
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.text.Text	
-text: StringProperty	Defines the text to be displayed.
-x: DoubleProperty	Defines the x-coordinate of text (default 0).
-y: DoubleProperty	Defines the y-coordinate of text (default 0).
-underline: BooleanProperty	Defines if each line has an underline below it (default false).
-strikethrough: BooleanProperty	Defines if each line has a line through it (default false).
-font: ObjectProperty<Font>	Defines the font for the text.
+Text()	Creates an empty Text.
+Text(text: String)	Creates a Text with the specified text.
+Text(x: double, y: double, text: String)	Creates a Text with the specified x-, y-coordinates and text.

Text Example



(a) Text(x, y, text)



Programming is fun

Programming is fun
Display text

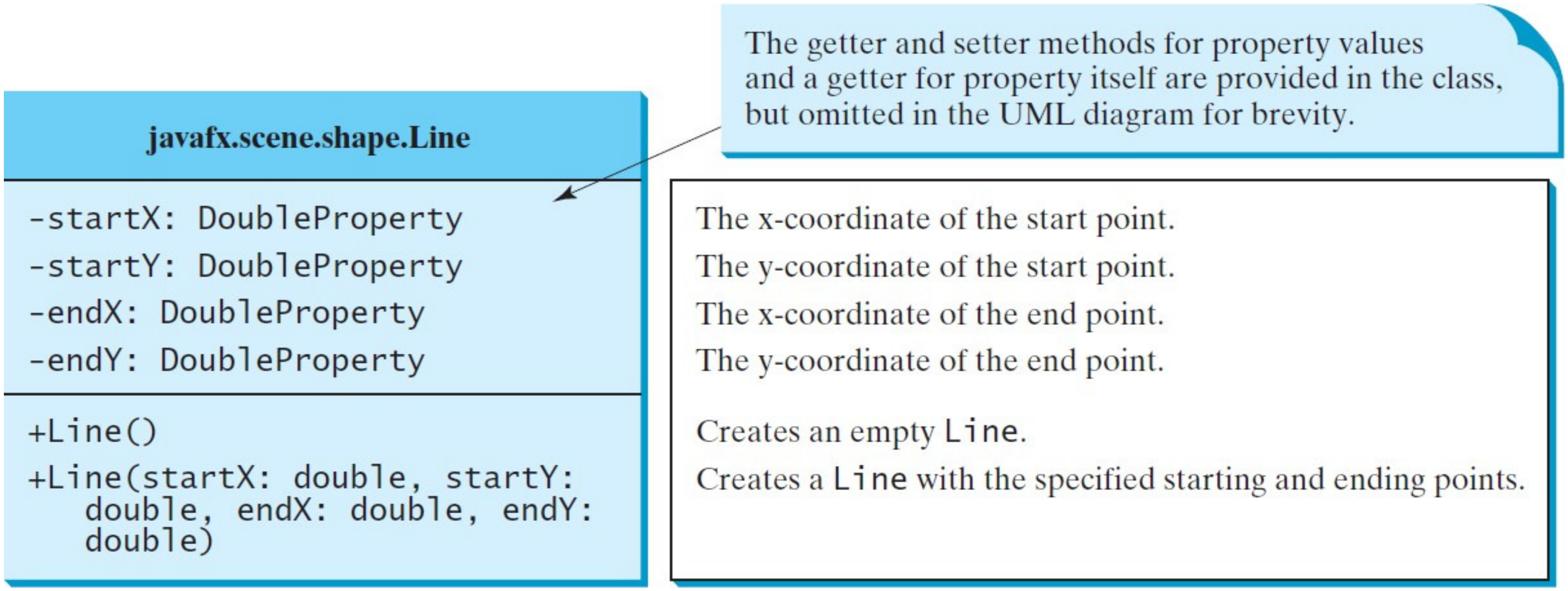
~~Programming is fun~~
~~Display text~~

(b) *Three* Text *objects are displayed*

ShowText

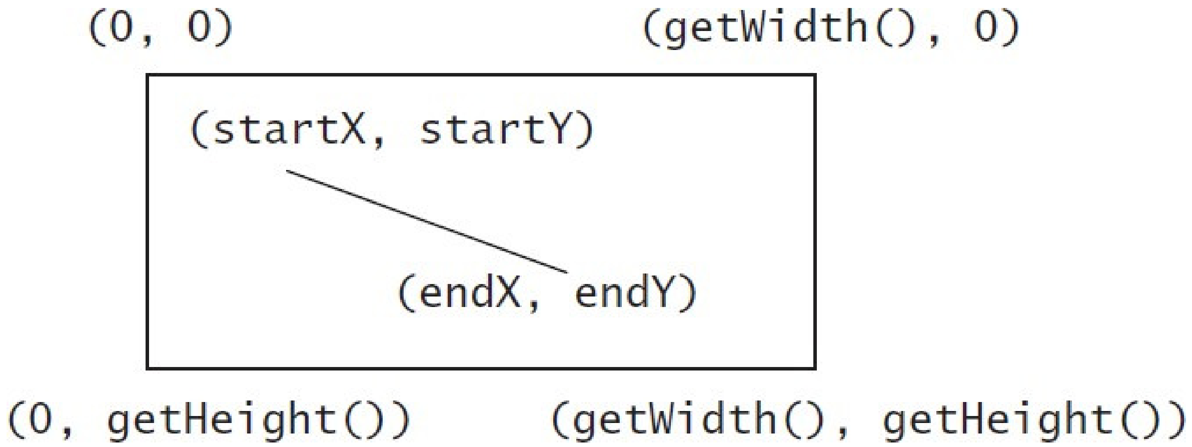
Run

Line



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.

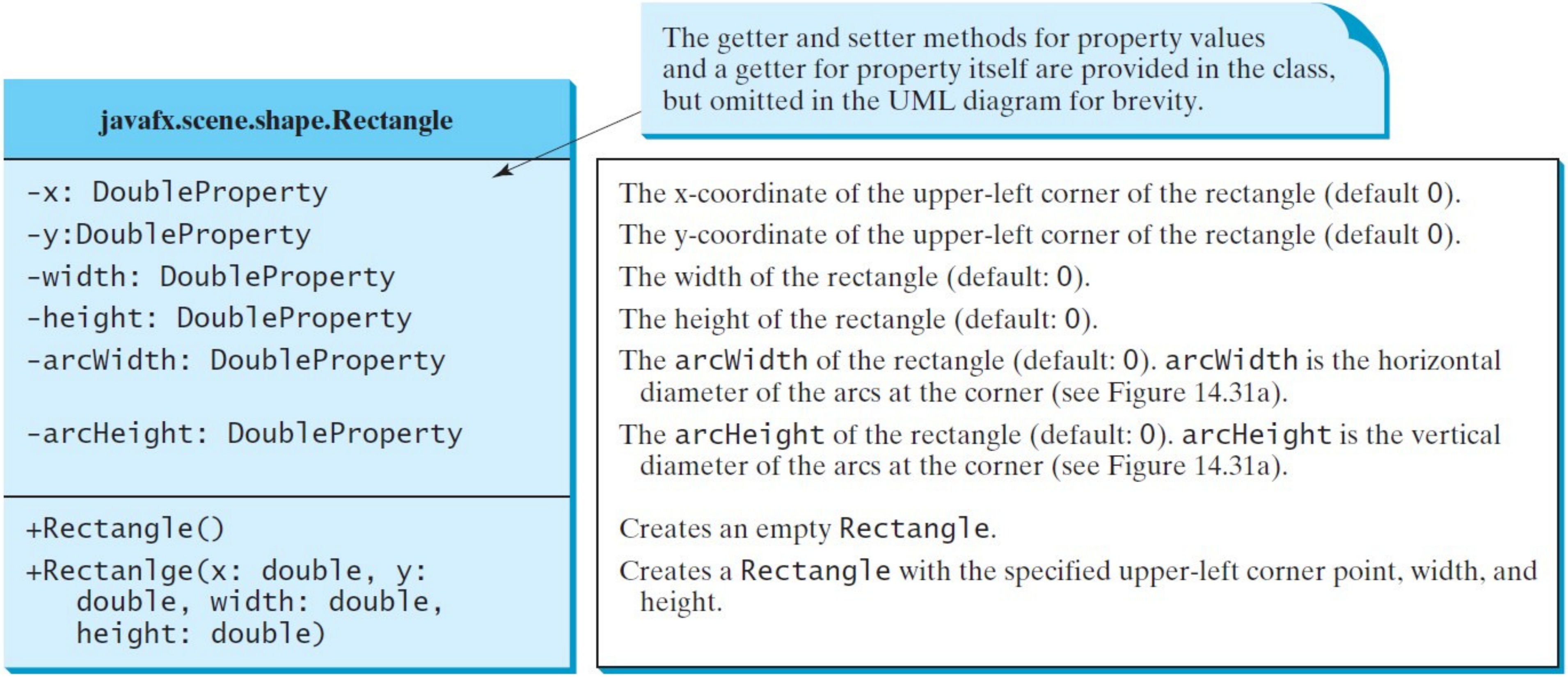
javafx.scene.shape.Line	
`-startX: DoubleProperty`	The x-coordinate of the start point.
`-startY: DoubleProperty`	The y-coordinate of the start point.
`-endX: DoubleProperty`	The x-coordinate of the end point.
`-endY: DoubleProperty`	The y-coordinate of the end point.
`+Line()`	Creates an empty `Line`.
`+Line(startX: double, startY: double, endX: double, endY: double)`	Creates a `Line` with the specified starting and ending points.



ShowLine

Run

Rectangle



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.shape.Rectangle	
-x: DoubleProperty	The x-coordinate of the upper-left corner of the rectangle (default 0).
-y:DoubleProperty	The y-coordinate of the upper-left corner of the rectangle (default 0).
-width: DoubleProperty	The width of the rectangle (default: 0).
-height: DoubleProperty	The height of the rectangle (default: 0).
-arcWidth: DoubleProperty	The arcWidth of the rectangle (default: 0). arcWidth is the horizontal diameter of the arcs at the corner (see Figure 14.31a).
-arcHeight: DoubleProperty	The arcHeight of the rectangle (default: 0). arcHeight is the vertical diameter of the arcs at the corner (see Figure 14.31a).
+Rectangle()	Creates an empty Rectangle.
+Rectanlge(x: double, y: double, width: double, height: double)	Creates a Rectangle with the specified upper-left corner point, width, and height.

Rectangle

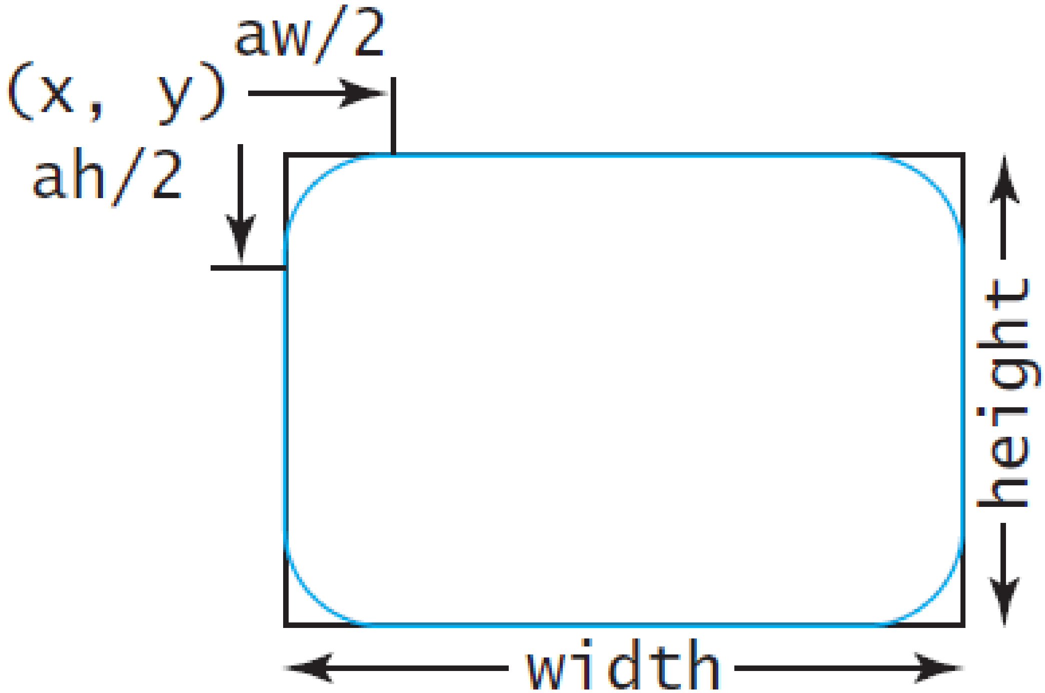
A rectangle is defined by the parameters **x**, **y**, **width**, **height**, **arcWidth**, and **arcHeight**.

The rectangle’s upper-left corner point is at (**x, y**)

and parameter **aw** (**arcWidth**) is the horizontal diameter of the arcs at the corner,

and **ah** (**arcHeight**) is the vertical diameter of the arcs at the corner.

Rectangle Example

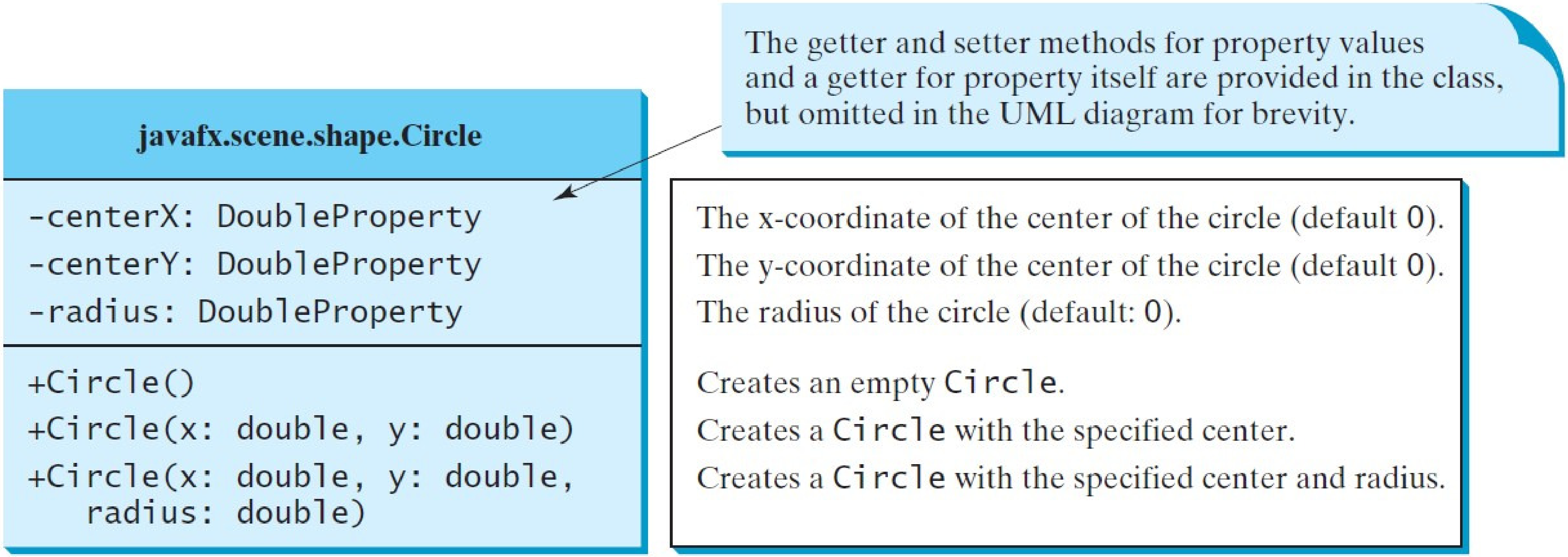


(a) Rectangle(x, y, w, h)

Rectangle Example

Check the example of Rectangle in JavaFX in Ritaj

Circle



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.shape.Circle

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radius: DoubleProperty

+Circle()
+Circle(x: double, y: double)
+Circle(x: double, y: double,
   radius: double)
```

The x-coordinate of the center of the circle (default 0).

The y-coordinate of the center of the circle (default 0).

The radius of the circle (default: 0).

Creates an empty Circle.

Creates a Circle with the specified center.

Creates a Circle with the specified center and radius.