Keenan Knaur
Adjunct Lecturer

California State University, Los Angeles
Computer Science Department

JavaFX Basics

CS2012: Introduction to Programming II

JavaFX Introduction

- incorporates modern GUI technologies to enable the development of Rich Internet Applications (RIA)
  - web apps designed to deliver the same features and functions normally associated with desktop applications.
- JavaFX apps can run seemlessly on a desktop or from a Web browser
- provides multi-touch support for touch-enabled devices such as tablets and smartphones
- has built in 2D and 3D animation support, video and audo playback, runs as a stand-alone application or from a browser.

JavaFX First Examples

JavaFX Application Basic Structure

- Every JavaFX program is defined in a class which extends `javafx.application.Application`
- Just extend the `Application` class and you can start making GUI programs.
- Extending the `Application` class will require you to implement the `start()` method.

LISTING 14.1 MyJavaFX.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");
11     Scene scene = new Scene(btOK, 200, 250);
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title
13     primaryStage.setScene(scene); // Place the scene in the stage
14     primaryStage.show(); // Display the stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {
22     Application.launch(args);
23   }
24 }
```

MyJavaFX.java

- line 8: the Driver class of your program needs to override the `start()` method of `Application`.
  - `public void start(Stage primaryStage).`
  - This method should be used to kickstart your JavaFX application.
  - Your start method should have very little code in it, just enough to create instances of other classes.

- JavaFX applications are built in layers:
  - line 10: creates a button object.
  - line 11: creates a Scene and adds the button to it.
  - lines 12 ~ 14: add the Scene to the Stage and display the window.

MyJavaFX.java

- This program displays a window with a simple button on it.

- The start() method becomes the new main method of a JavaFX application.

- NOTE: line 22 calls the `Application.launch(args)` method.
  - This is a static method in the Application class.
  - Some IDEs (like Eclipse) require that you still have a `main()` method in your program and lines 21-23 will make your program the most compatible with a wide variety of platforms.

MyJavaFX.java

- A Stage object is a window
  - ***primary stage***: a Stage object automatically created by the JVM when the application is launched.

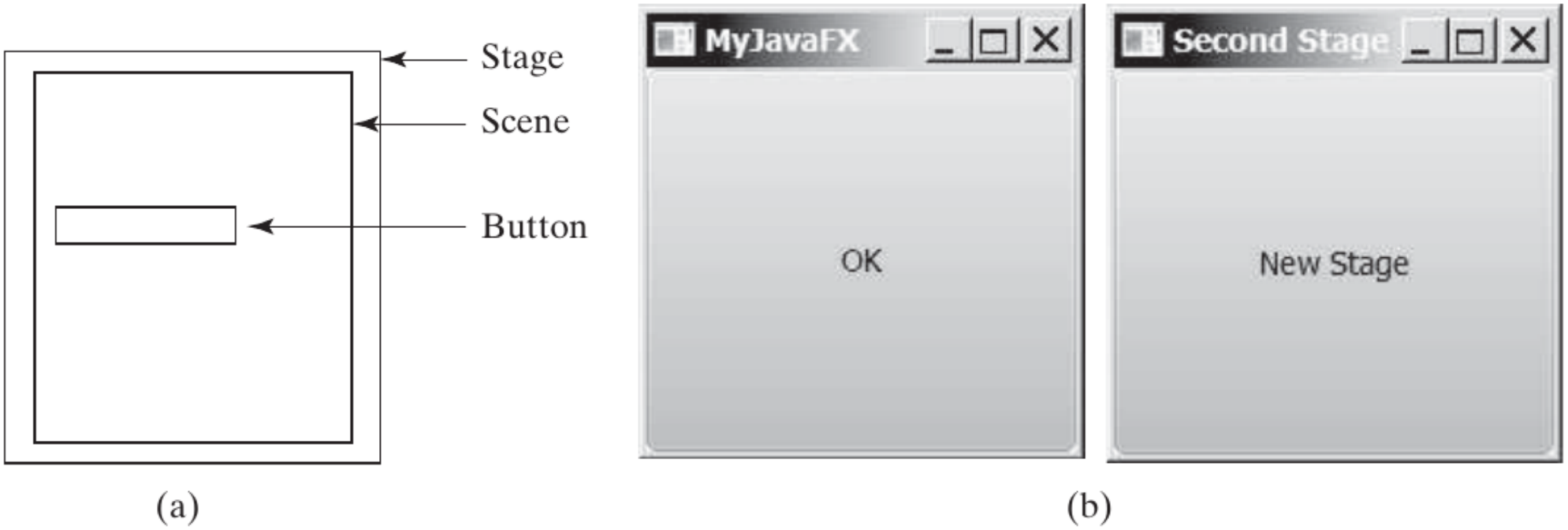


FIGURE 14.2 (a) Stage is a window for displaying a scene that contains nodes. (b) Multiple stages can be displayed in a JavaFX program.

MultipleStageDemo.java

LISTING 14.2 MultipleStageDemo.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MultipleStageDemo extends Application {
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {
9      // Create a scene and place a button in the scene
10     Scene scene = new Scene(new Button("OK"), 200, 250);
11     primaryStage.setTitle("MyJavaFX"); // Set the stage title
12     primaryStage.setScene(scene); // Place the scene in the stage
13     primaryStage.show(); // Display the stage
14
15     Stage stage = new Stage(); // Create a new stage
16     stage.setTitle("Second Stage"); // Set the stage title
17     // Set a scene with a button in the stage
18     stage.setScene(new Scene(new Button("New Stage"), 100, 100));
19     stage.show(); // Display the stage
20   }
21 }
```

JavaFX Nodes

The Node Class

- Panes, UI controls, and Shapes, are all subtypes of the **Node** class.
- ***node***: visual components such as shapes, images, UI controls, or even a pane.
- ***pane***: a container class which helps to automatically layout a node in a desired location and size.
  - nodes are placed inside of a pane and then the pane is placed into a scene
- ***shape***: text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.
- ***UI control***: a label, button, checkbox, radio button, text field, text area, etc.

JavaFX Class Heirarchy

- The relationship between a `Stage`, `Scene`, `Node`, `Control`, and `Pane` is shown on the next slide.
- A `Scene` can contain a `Control` or `Pane`, but not a `Shape` or `ImageView`
- A `Pane` can contain any subtype of Node

- A Scene can be created with the following constructors:
  - `Scene(Parent, width, height)`
  - `Scene(Parent)` (dimensions are automatically decided)

- Every subclass of Node has a no-arg constructor for creating a default node.

JavaFX Class Heirarchy

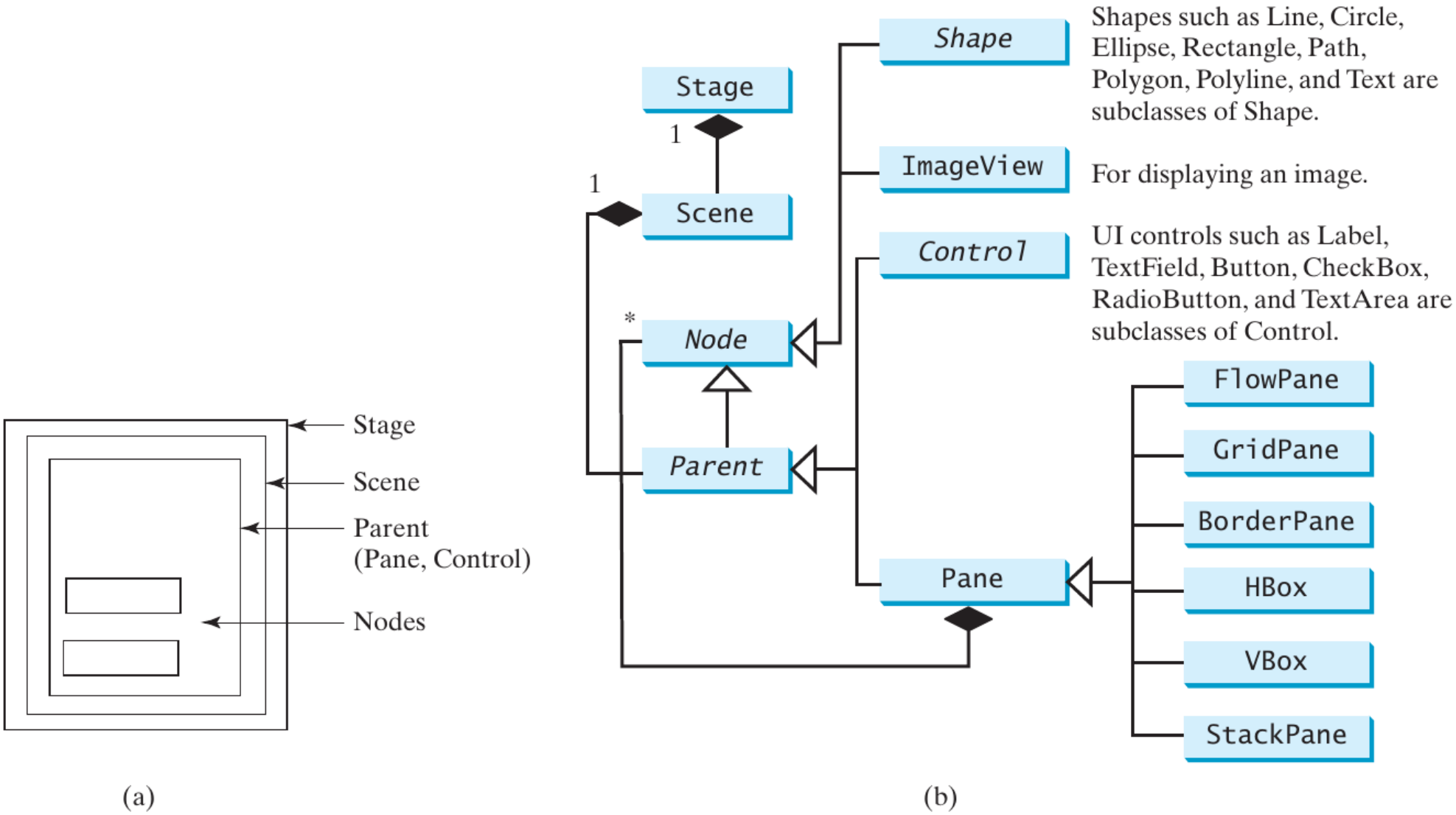


FIGURE 14.3 (a) Panes are used to hold nodes. (b) Nodes can be shapes, image views, UI controls, and panes.

ButtonInPane.java

LISTING 14.3 ButtonInPane.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5  import javafx.scene.layout.StackPane;
6
7  public class ButtonInPane extends Application {
8    @Override // Override the start method in the Application class
9    public void start(Stage primaryStage) {
10     // Create a scene and place a button in the scene
11     StackPane pane = new StackPane();
12     pane.getChildren().add(new Button("OK"));
13     Scene scene = new Scene(pane, 200, 50);
14     primaryStage.setTitle("Button in a pane"); // Set the stage title
15     primaryStage.setScene(scene); // Place the scene in the stage
16     primaryStage.show(); // Display the stage
17   }
18 }
```

ButtonInPane.java

- line 11: program creates a StackPane
  - `StackPane` places nodes in the center of the pane on top of each other.
  - `StackPane` respects a node's preferred size (it does not change the size of the components to fit the window)

- line 12: adds a button as a child of the pane
  - `getChildren()` returns an instance of ObservableList
  - `ObservableList` is a lot like an ArrayList for storing collections of elements
  - `add(e)` adds an element to the list

LISTING 14.4 ShowCircle.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.stage.Stage;
7
8  public class ShowCircle extends Application {
9    @Override // Override the start method in the Application class
10   public void start(Stage primaryStage) {
11     // Create a circle and set its properties
12     Circle circle = new Circle();
13     circle.setCenterX(100);
14     circle.setCenterY(100);
15     circle.setRadius(50);
16     circle.setStroke(Color.BLACK);
17     circle.setFill(Color.WHITE);
18
19     // Create a pane to hold the circle
20     Pane pane = new Pane();
21     pane.getChildren().add(circle);
22
23     // Create a scene and place it in the stage
24     Scene scene = new Scene(pane, 200, 200);
25     primaryStage.setTitle("ShowCircle"); // Set the stage title
26     primaryStage.setScene(scene); // Place the scene in the stage
27     primaryStage.show(); // Display the stage
28   }
29 }
```

ShowInCircle.java

- line 12: creates a `Circle`
- lines 13-14: set the center to be (100, 100)
  - also center of `Scene`, since size of `Scene` is 200 x 200
  - all measurements are in ***pixels***
- line 16: stroke color set to black (line color)
- line 17: fill color set to white (color to fill the shape)
  - color could be **null**
- line 20: creates a `Pane`, puts circle in pane
- line 24: `Pane` is placed in the `Scene`
- line 26: `Scene` is set in the Stage
- NOTE: circle is centered as long as you do not resize the window. We will see how to fix this next lecture.

JavaFX Coordinate System

- the upper left corner of a Pane or Scene is used as (0,0) in the Java coordinate system.
- does not use the conventional coordinate system where (0,0) is the center of the Pane or Scene

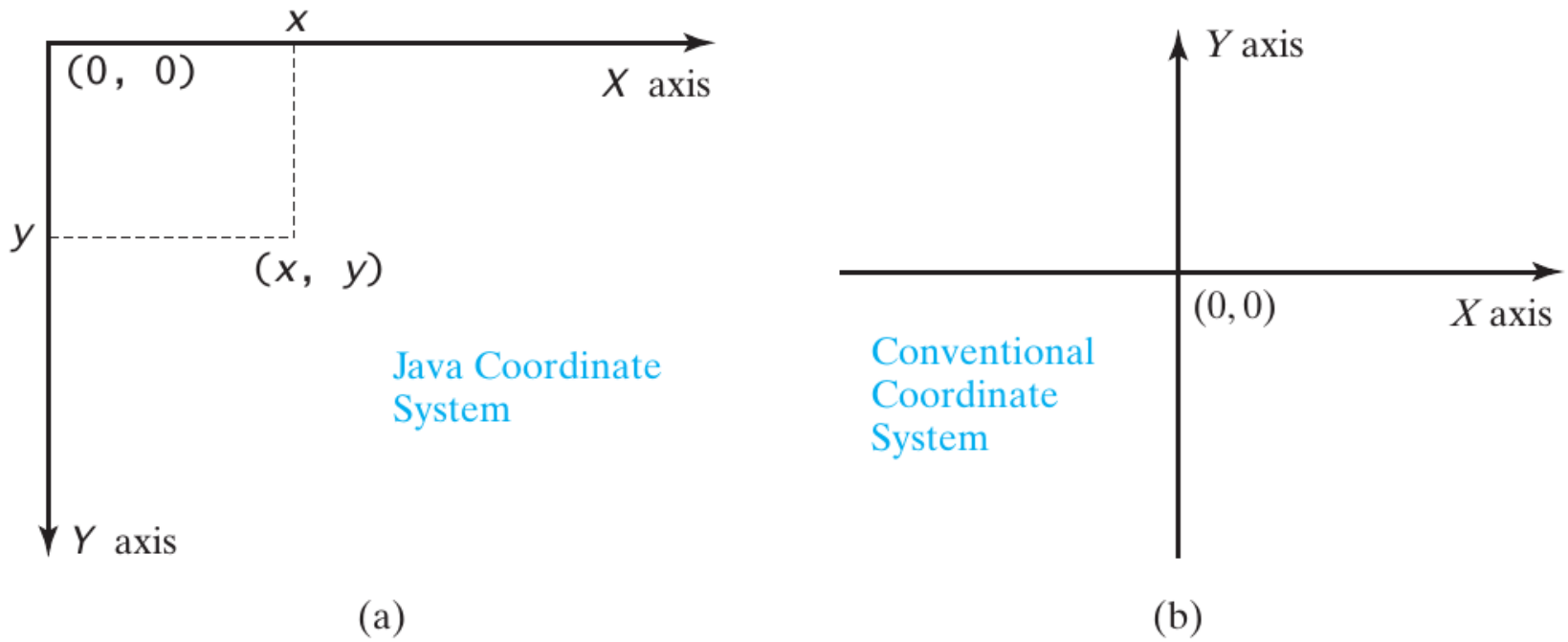


FIGURE 14.6 The Java coordinate system is measured in pixels, with **(0, 0)** at its upper-left corner.

Layout Panes

Layout Panes

- JavaFX provides different layout panes to automatically contain and organize nodes.

TABLE 14.1 Panes for Containing and Organizing Nodes

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.

FlowPane

- Arranges the nodes in the pane horizontally from left to right, or vertically from top to bottom in the order that the nodes were added to the pane.

- If a row or column is filled, and new row or column is started.

- Constants:
  - `Orientation.HORIZONTAL` (place the nodes horizontally)
  - `Orientation.VERTICAL` (place the nodes vertically)

- `FlowPane` has data fields `alignment`, `orientation`, `hgap`, and `vgap`.

FlowPane

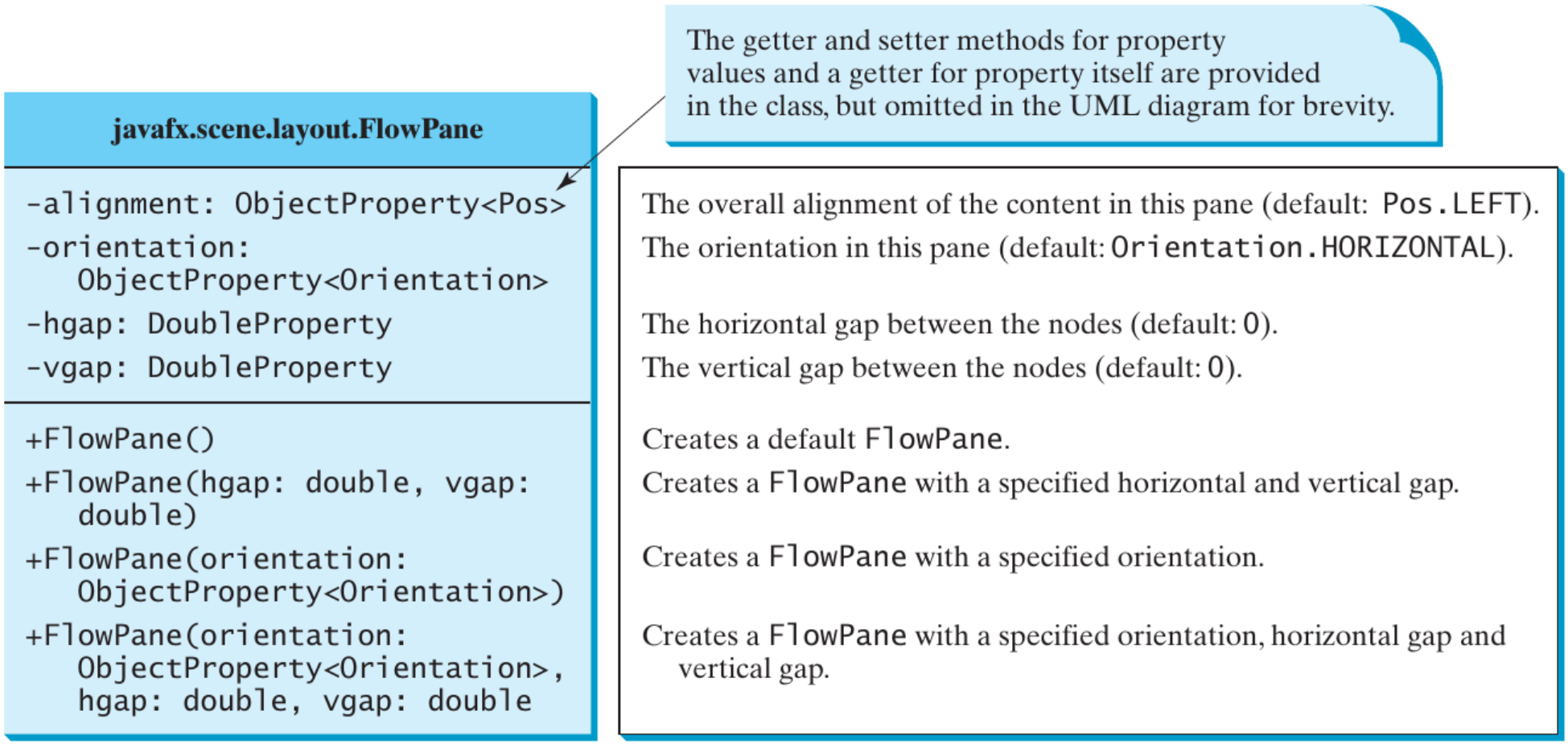


FIGURE 14.15 `FlowPane` lays out nodes row by row horizontally or column by column vertically.

FlowPane

- See ShowFlowPane.java

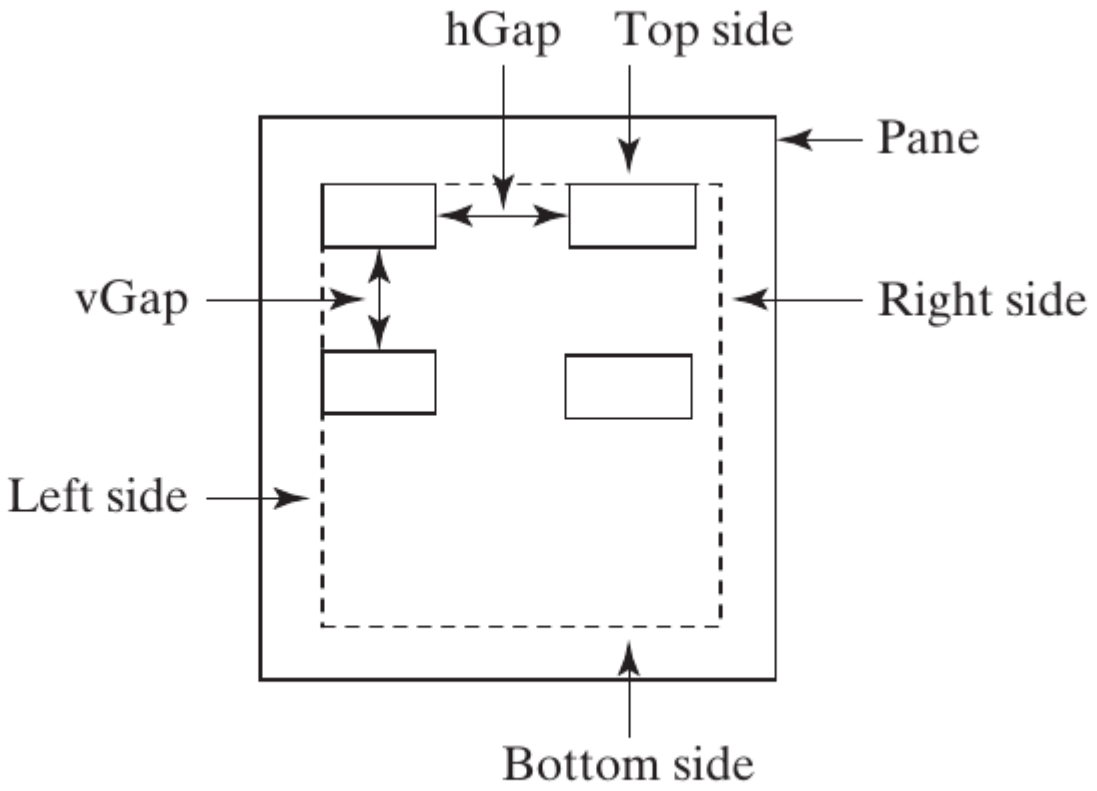


FIGURE 14.17 You can specify `hGap` and `vGap` between the nodes in a `FlowLPane`.

GridPane

- Arranges nodes in a grid (matrix) formation, nodes are placed in the specified row and indices.

- See: ShowGridPane.java

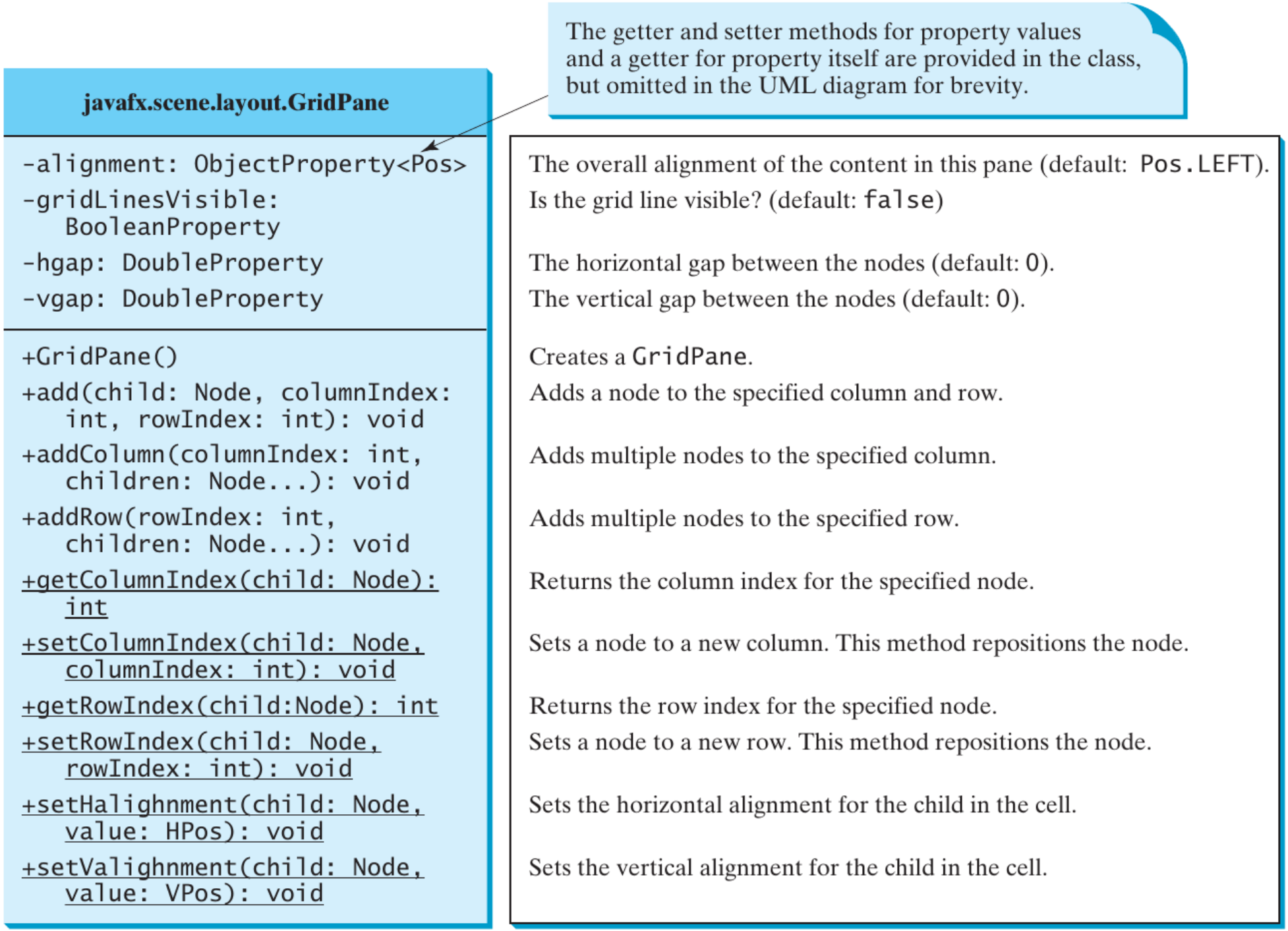


FIGURE 14.18 `GridPane` lays out nodes in the specified cell in a grid.

BorderPane

- Can place nodes in five regions:
  - top `setTop(node)`
  - bottom `setBottom(node)`
  - left `setLeft(node)`
  - right `setRight(node)`
  - center `setCenter(node)`

- See Code: ShowBorderPane.java

BorderPane

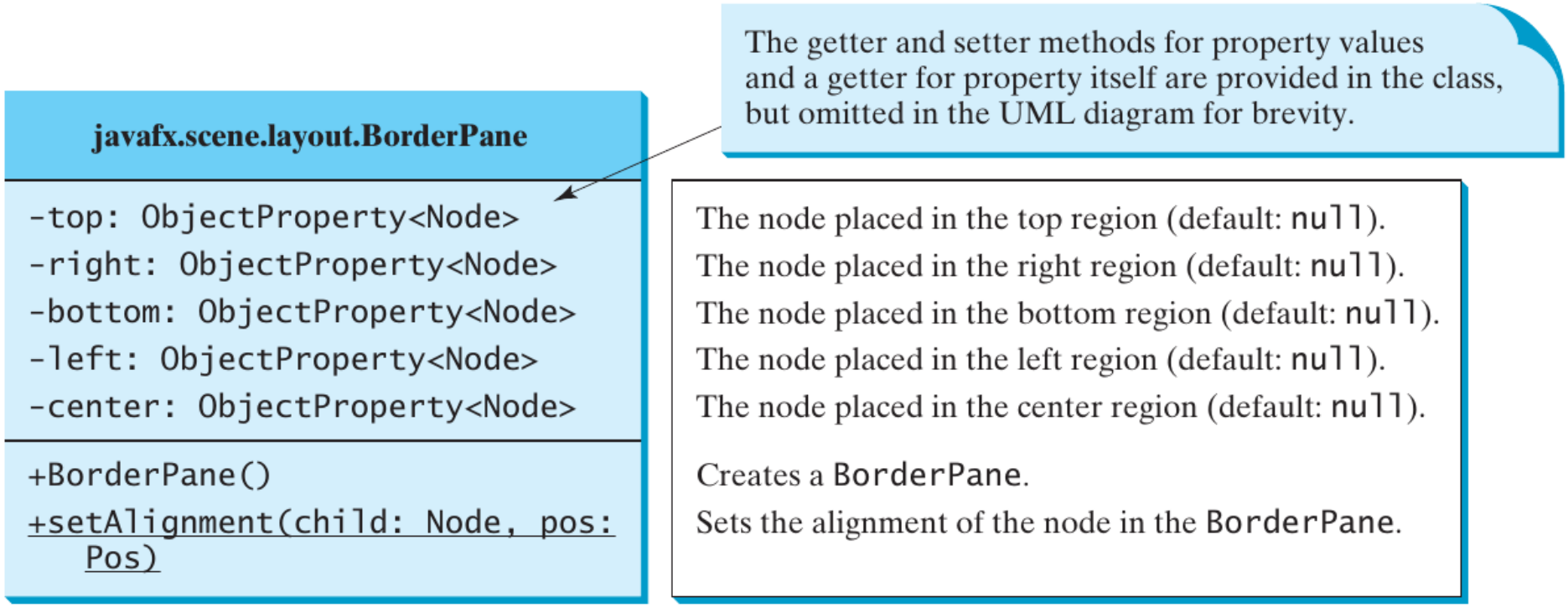


FIGURE 14.20 BorderPane places the nodes in top, bottom, left, right, and center regions.

Hbox and VBox

- **HBox** lays out its children in a single row.
- **VBox** lays out its children in a single vertical column

- See Code: `ShowHBoxVBox.java`

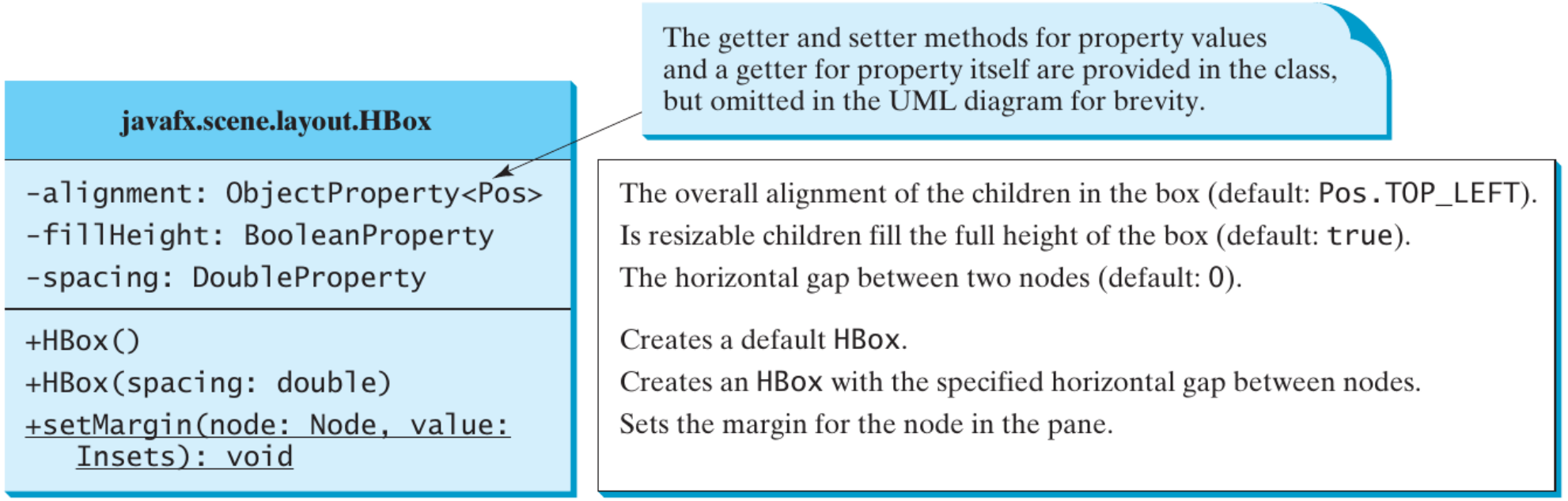


FIGURE 14.22 HBox places the nodes in one row.

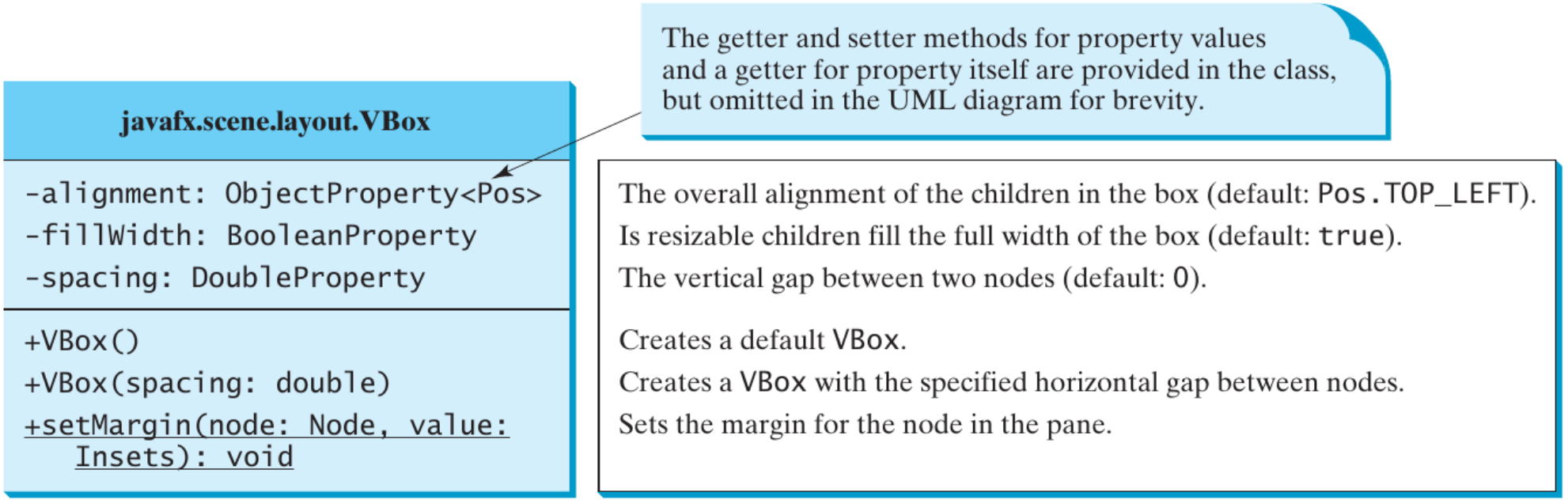


FIGURE 14.23 VBox places the nodes in one column.

The Color Class

The Color Class

- used to create colors
- The abstract Paint class is used for painting a Node
- javafx.scene.paint.Color is a concrete subclass of Paint used to encapsulate colors

The Color Class

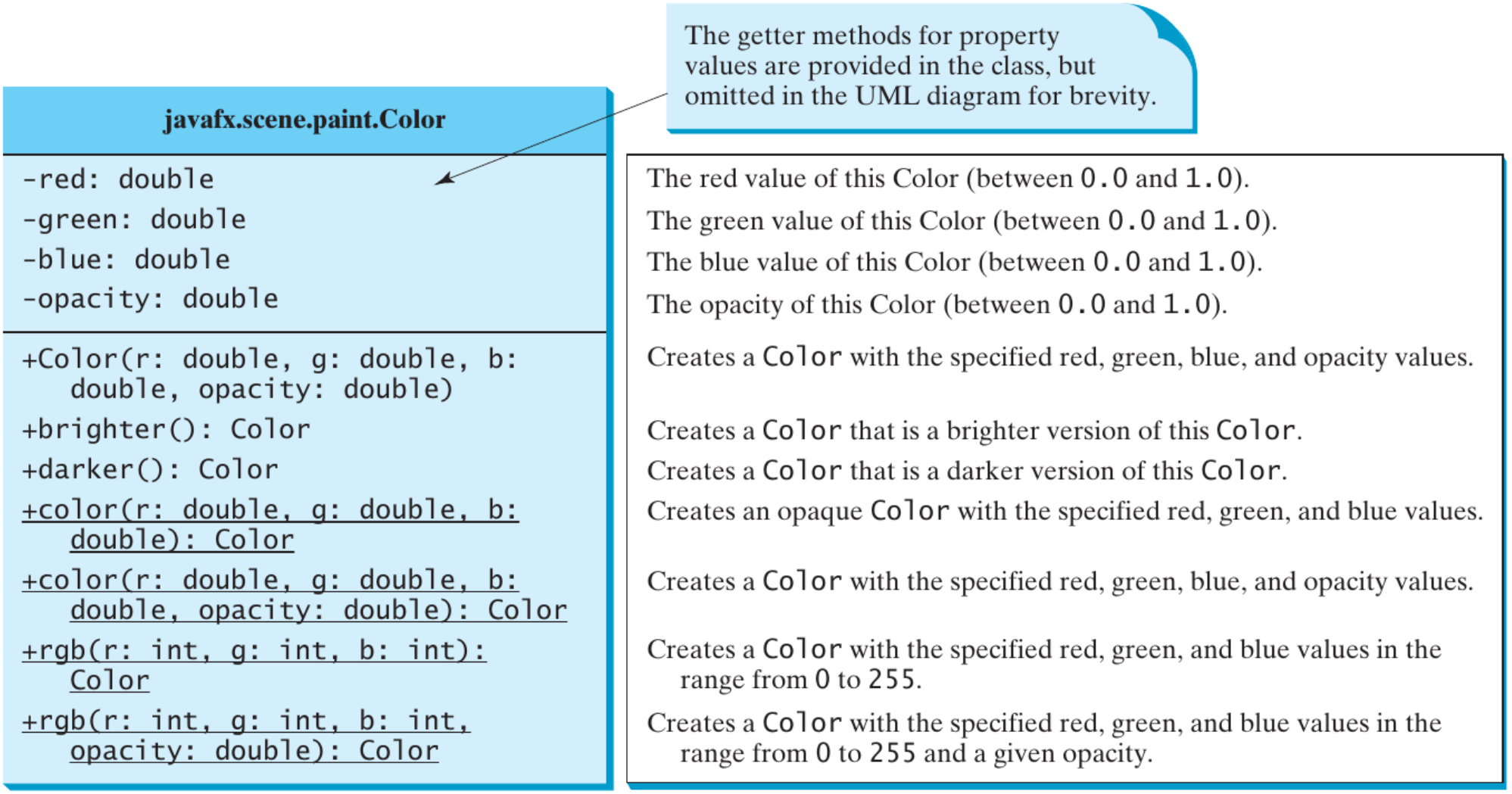


FIGURE 14.9 `Color` encapsulates information about colors.

The Color Class

- Creating a color:
  - `public Color(double r, double g, double b, double opacity);`
  - r, g, b are the red, green, blue components of a color, value range from 0.0 (darkest shade) to 1.0 (lightest shade)
  - opacity is the color transparency between 0.0 (completely transparent) to 1.0 (completely opaque)
  - This follows the RGBA (red, green, blue, alpha) color model.

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

The Color Class

- `Color` is immutable
  - any methods which "alter" the color actually create a new object and return that object.
- You can also create a color using some of the static methods of the `Color` class.
- You can also use some of the standard colors which are defined as constants in the `Color` class.
  - See the API for `Color`
  - Example: `circle.setFill(Color.RED);`

The Font Class

- used to create and set fonts and change their properties.
- `Font` has a name, weight, posture, and size.
- You can see a list of available font family names by calling getFamilies().
- FontPosture is for italics or no italics:
  - `FontPosture.ITALIC`
  - `FontPosture.REGULAR`

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

- See: FontDemo.java

The Font Class

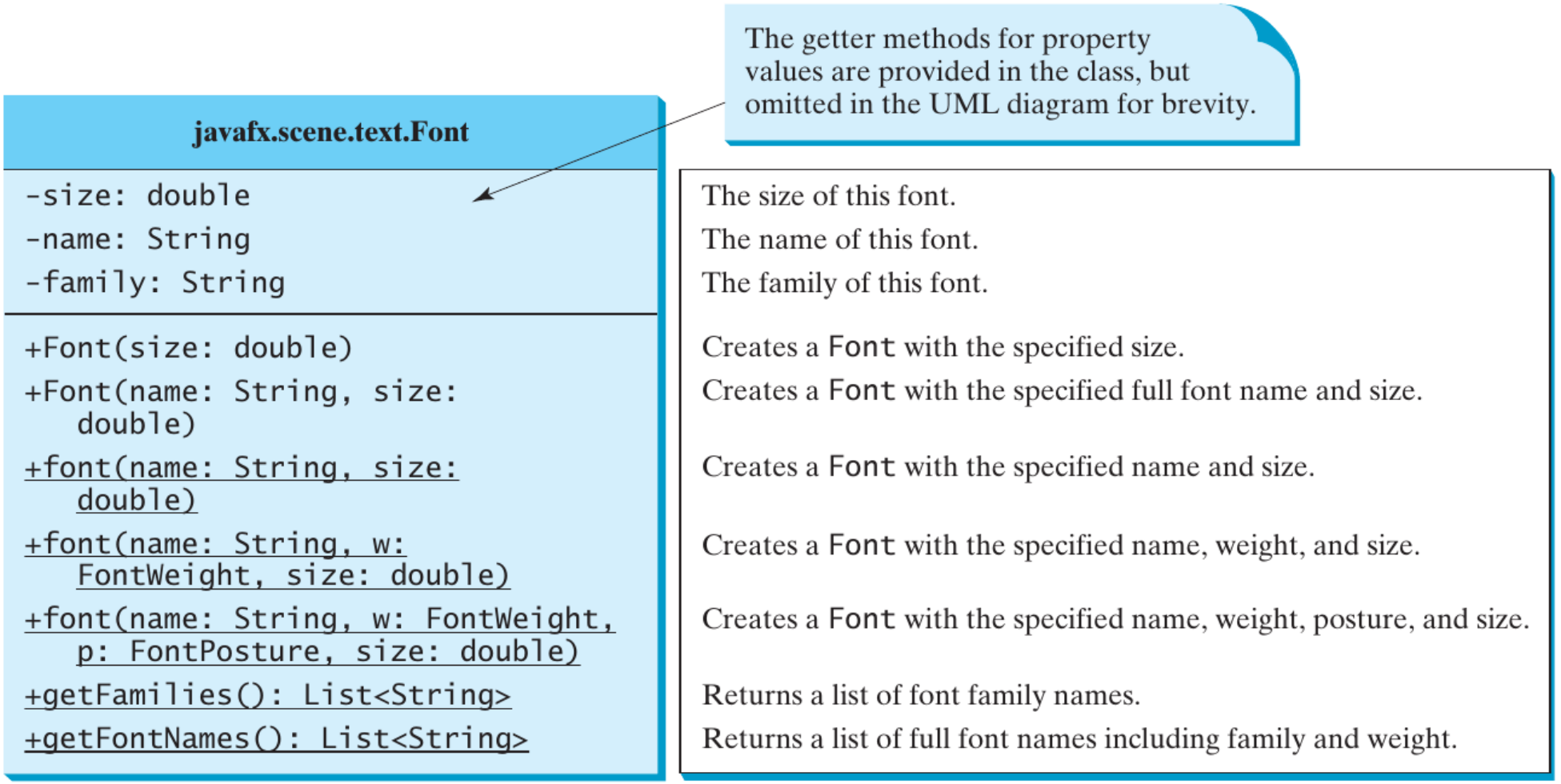


FIGURE 14.10 `Font` encapsulates information about fonts.

The Image and ImageView Classes

The Image and ImageView Classes

- `javafx.scene.image.Image` represents an image specified by a URL or filename.

```
new Image("image/us.gif")
new Image("http://somewebsite.com/us.gif")
```

- `javafx.scene.image.ImageView` is a node for displaying an image and can be created from an Image object

```
Image image = new Image("image/us.gif");
ImageView imageView = new ImageView(image);
```

- You can also make an `ImageView` directly from a file or URL:
  - `ImageView imageView = new ImageView("image/us.gif");`

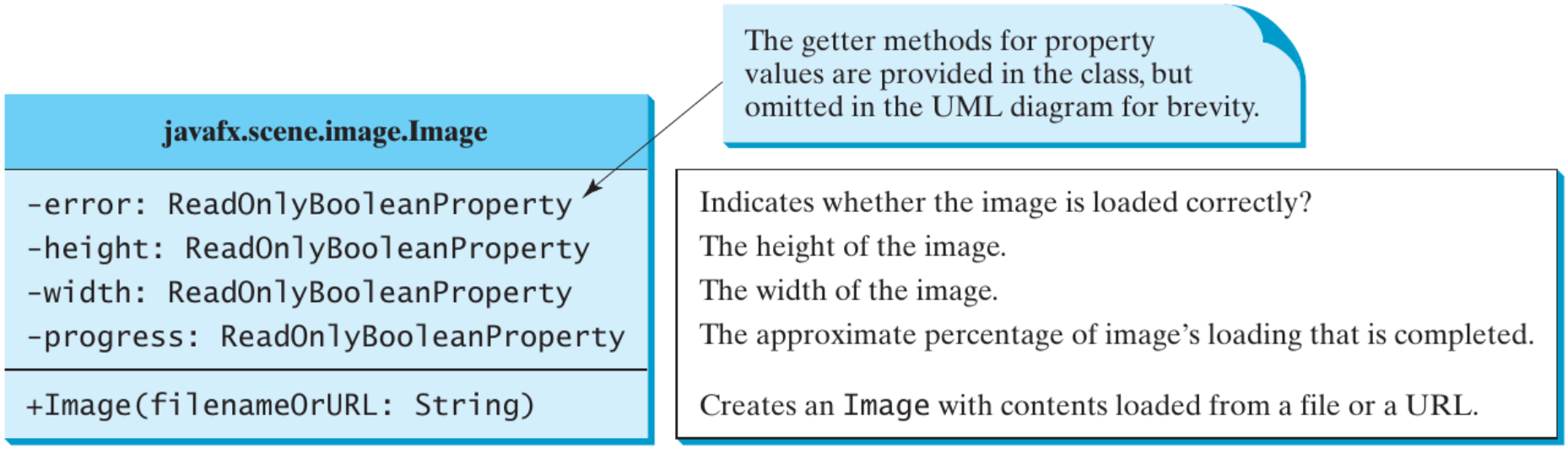


FIGURE 14.12 `Image` encapsulates information about images.

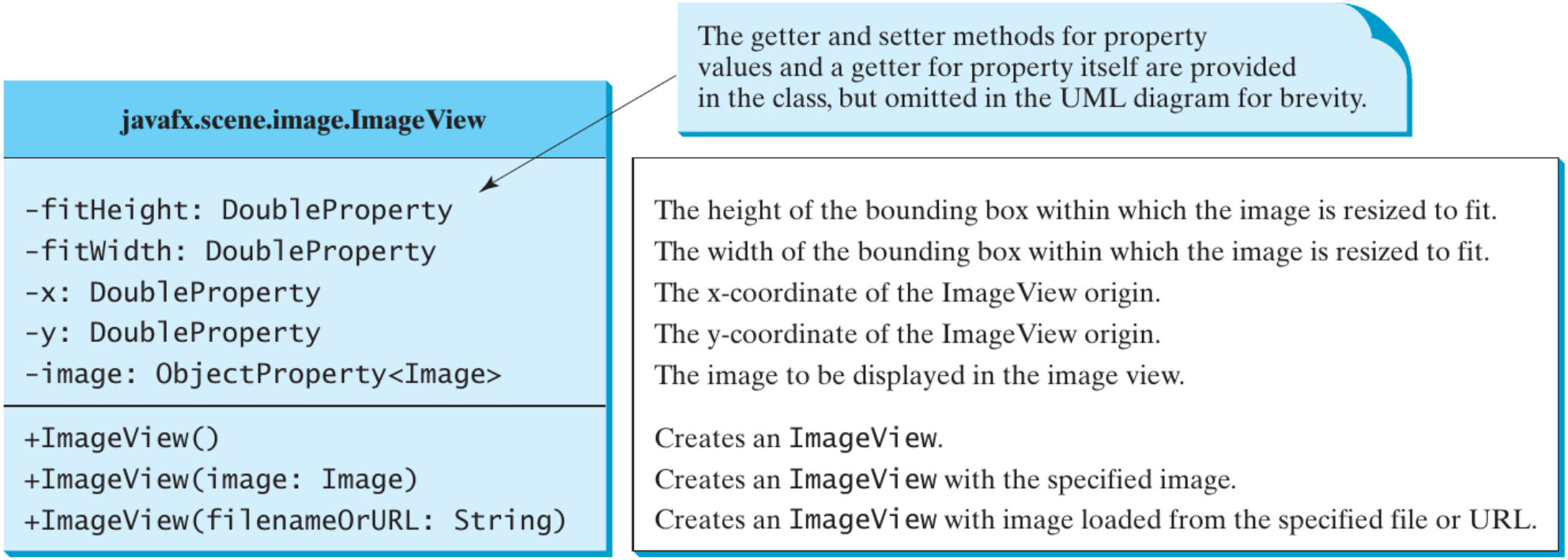


FIGURE 14.13 `ImageView` is a node for displaying an image.

- See Code: `ShowImage.java`

- All resources (images, sounds, videos, etc) should be placed in organized folders in your project structure.

Shapes

Shapes

- JavaFX provides many ways to draw text, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.

- Shape is an abstract base class that defines the common properties of all shapes.

The Text Class

- Text class defines a node that displays a string at a starting point (x, y)
- Usually placed in a pane.
  - upper-left corner of a pane is (0, 0)
  - bottom-right point is (`pane.getWidth()`, `pane.getHeight()`)
- Use `\n` to break a string over multiple lines.
- See Code: ShowText.java

The Text Class

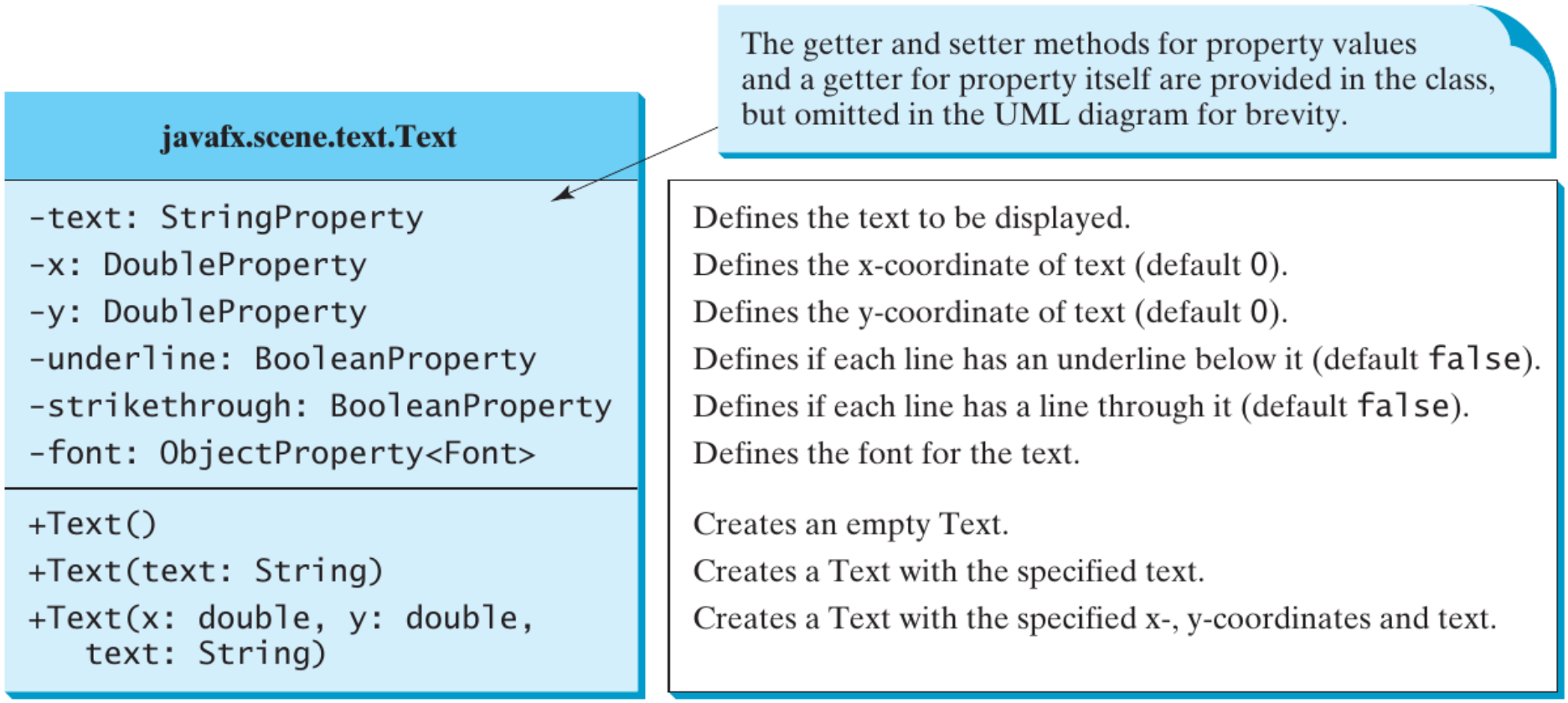
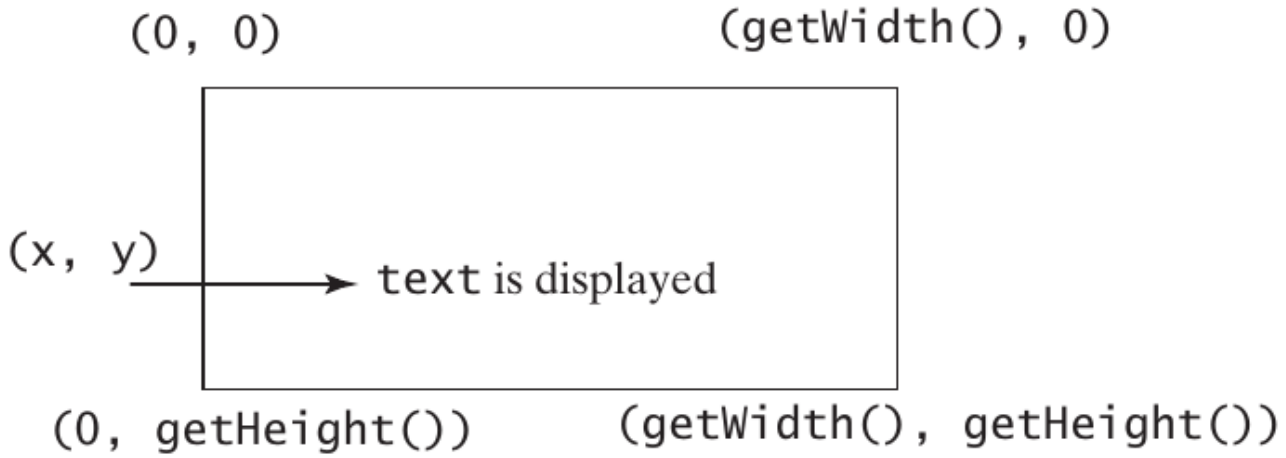


FIGURE 14.26 `Text` defines a node for displaying a text.

The Text Class



(a) Text(x, y, text)

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

FIGURE 14.27 A `Text` object is created to display a text.

The Line Class

- A line connects two points with four parameters startX, startY, endX, and endY

- See Code: ShowLine.java

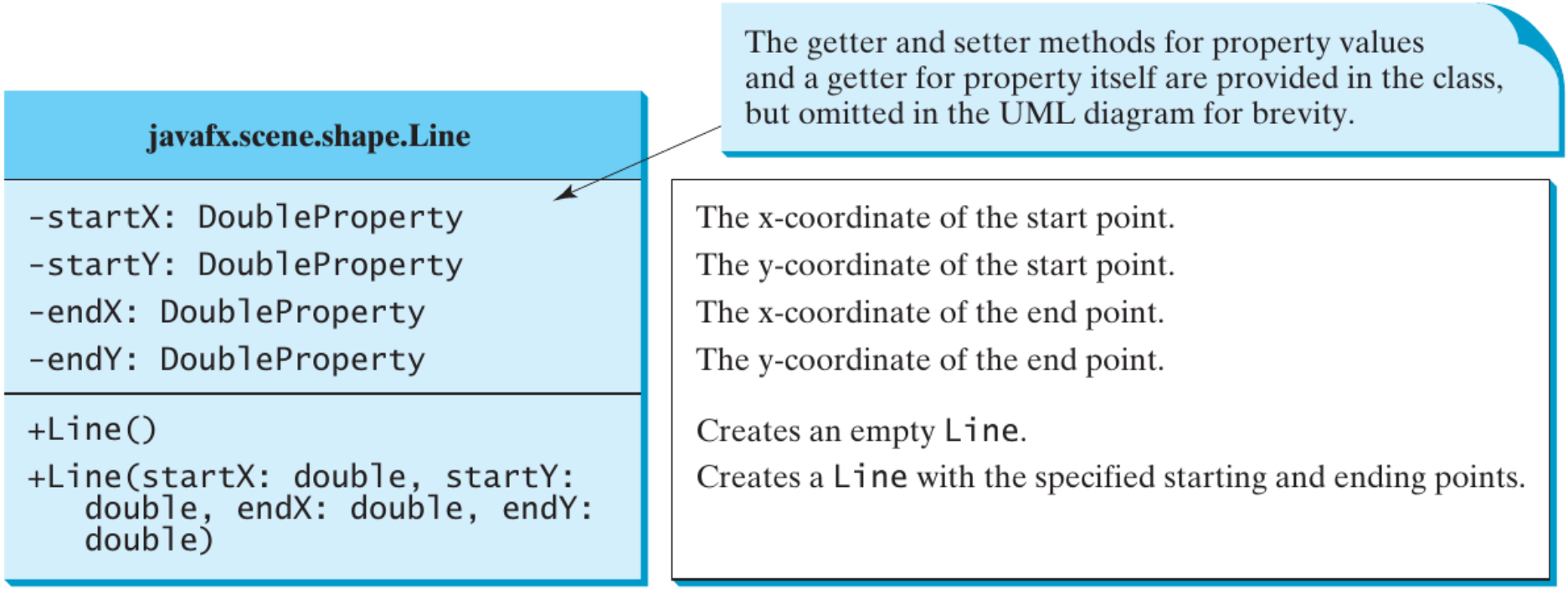
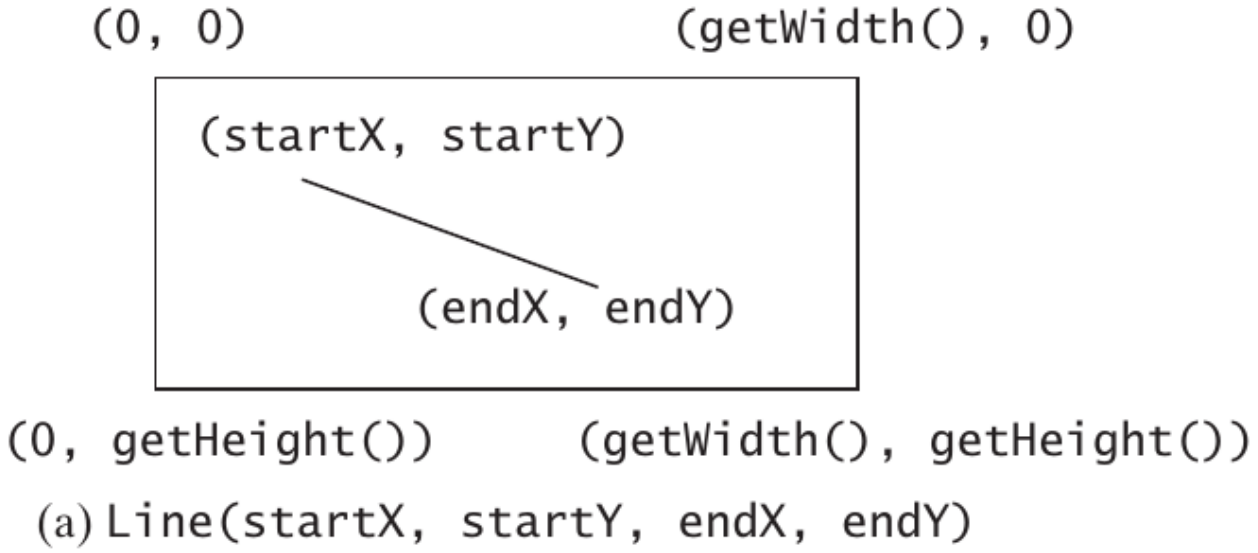
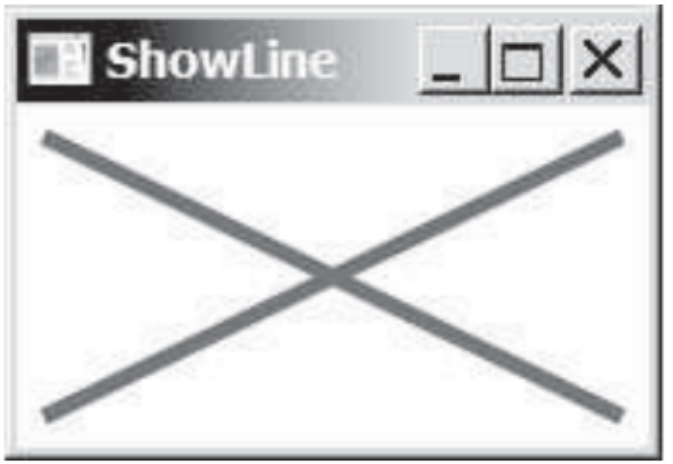


FIGURE 14.28 The `Line` class defines a line.

The Line Class



(a) `Line(startX, startY, endX, endY)`



(b) Two lines are displayed across the pane.

FIGURE 14.29 A `Line` object is created to display a line.

The Rectangle Class

- defined by parameters x, y, width, height, arcWidth, and arcHeight
- upper-left corner is point at (x, y) and parameter aw (arcWidth) the horizontal diameter of the arcs at the corner, and ah (arcHeight) the vertical diameter of the arcs at the corner
- See Code: ShowRectangle.java

The Rectangle Class

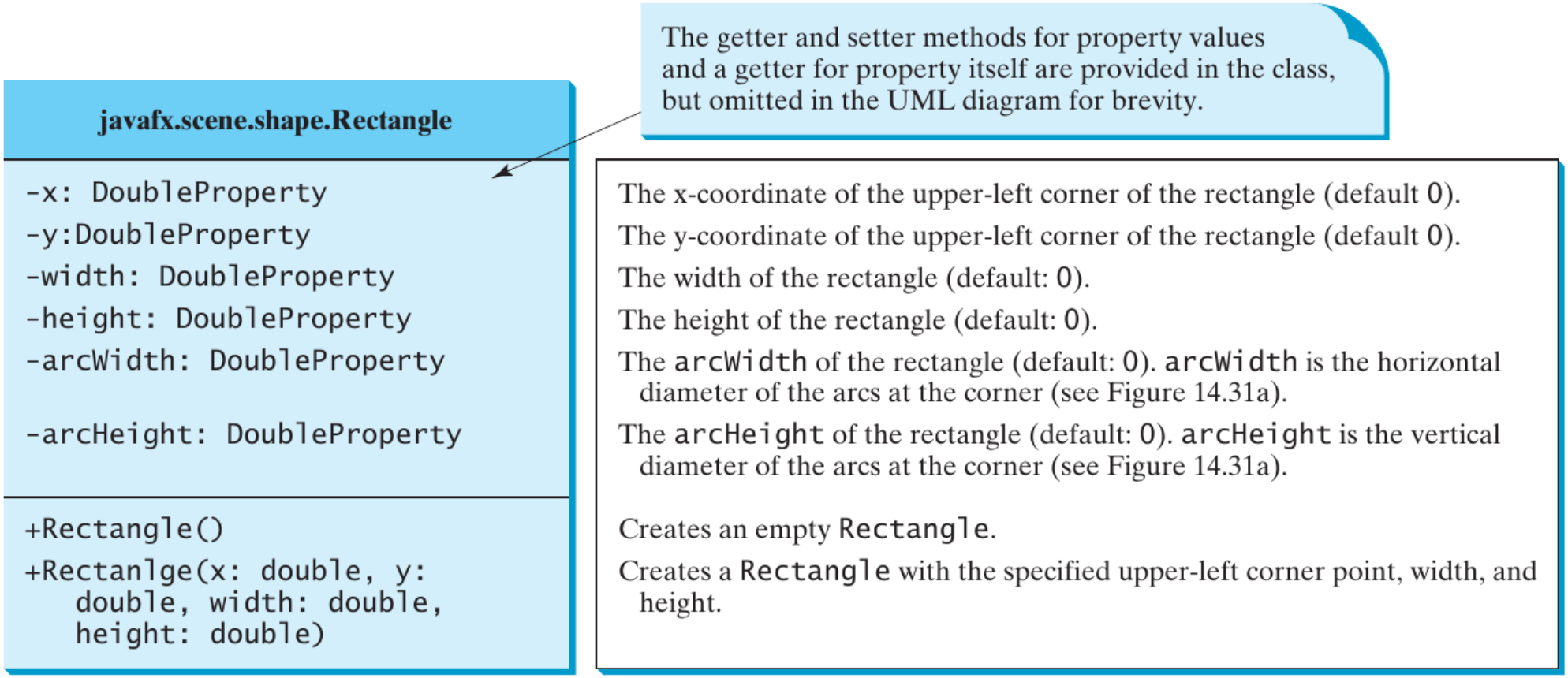
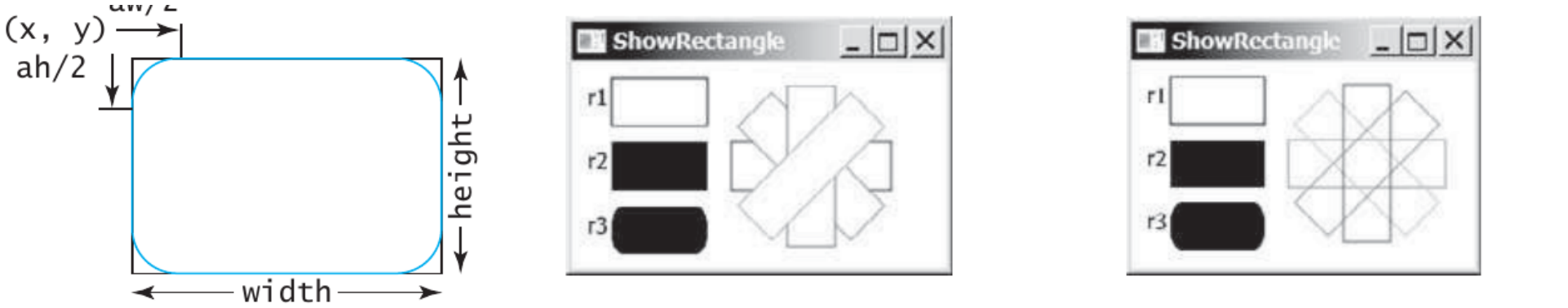


FIGURE 14.30 The `Rectangle` class defines a rectangle.

The Rectangle Class



(a) `Rectangle(x, y, w, h)` (b) Multiple rectangles are displayed (c) Transparent rectangles are displayed

FIGURE 14.31 A `Rectangle` object is created to display a rectangle.

The Circle and Ellipse Classes

- Circles are defined by centerX, centerY, and radius
- An Ellipse is defined by centerX, centerY, radiusX, and radiusY
- See Code: ShowEllipse.java and ShowCircle.java

The Circle and Ellipse Classes

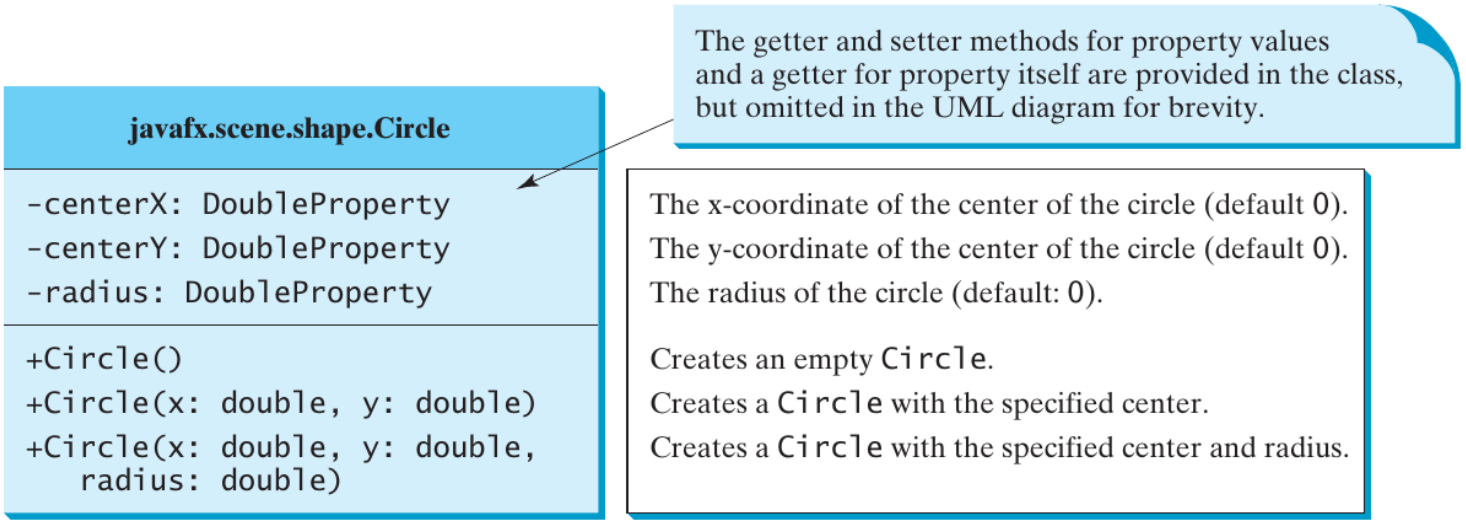


FIGURE 14.32 The `Circle` class defines circles.

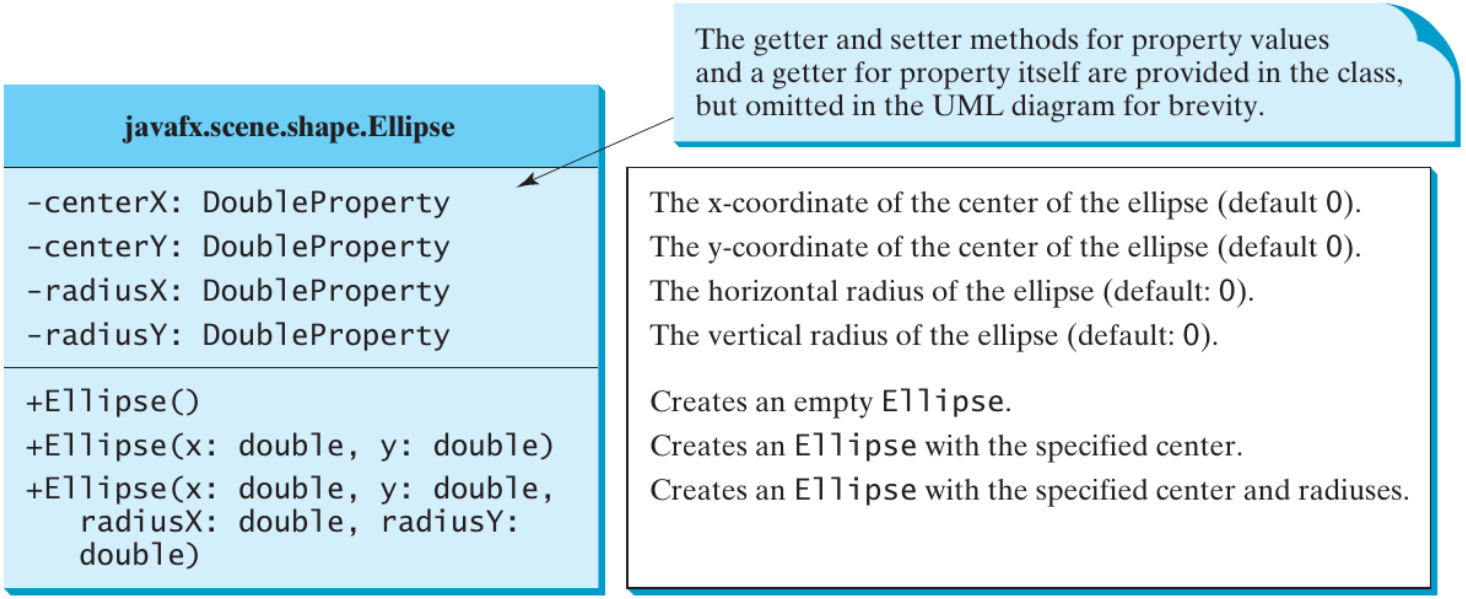
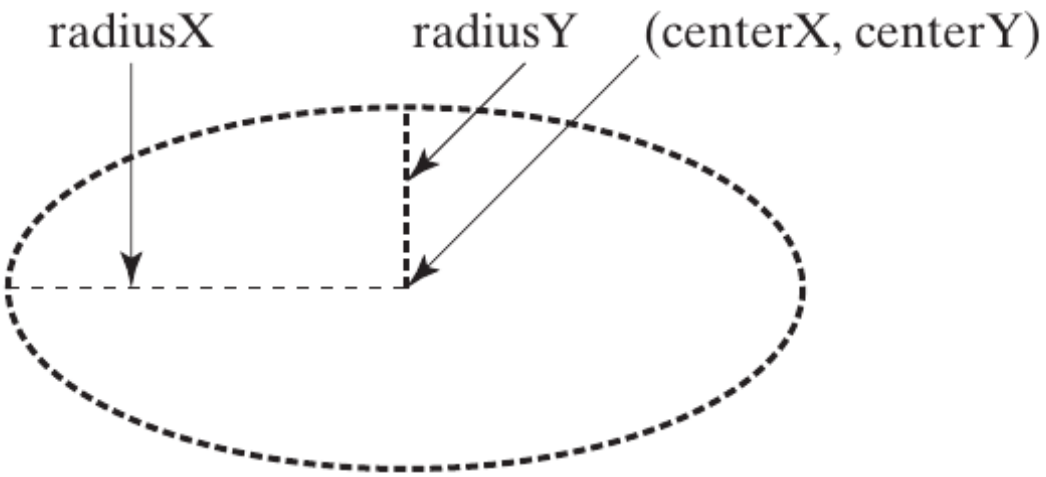


FIGURE 14.33 The `Ellipse` class defines ellipses.

The Circle and Ellipse Classes



(a) `Ellipse(centerX, centerY, radiusX, radiusY)`

(b) Multiple ellipses are displayed.

FIGURE 14.34 An `Ellipse` object is created to display an ellipse.

The Arc Class

- Part of an ellipse
- defined by centerX, centerY, radiusX, radiusY, startAngle, length, and an arc type (ArcType.OPEN, ArchType.CHORD, or ArcType.ROUND)
- Angles are measured in degrees and follow the normal math conventions (0 is in the easterly direction, positive angles go counterclockwise from the easterly direction)
- See Code: ShowArc.java

The Arc Class

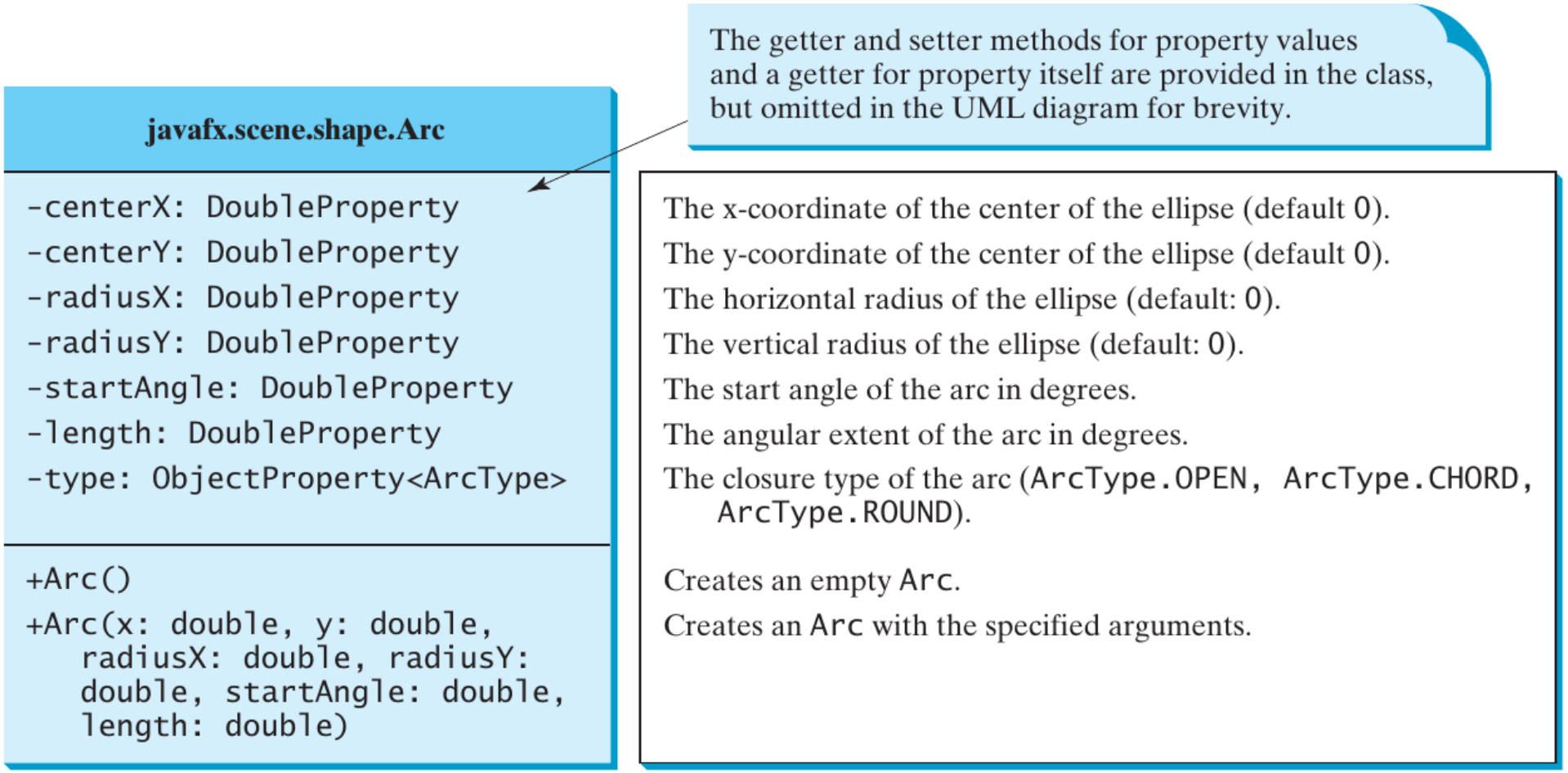
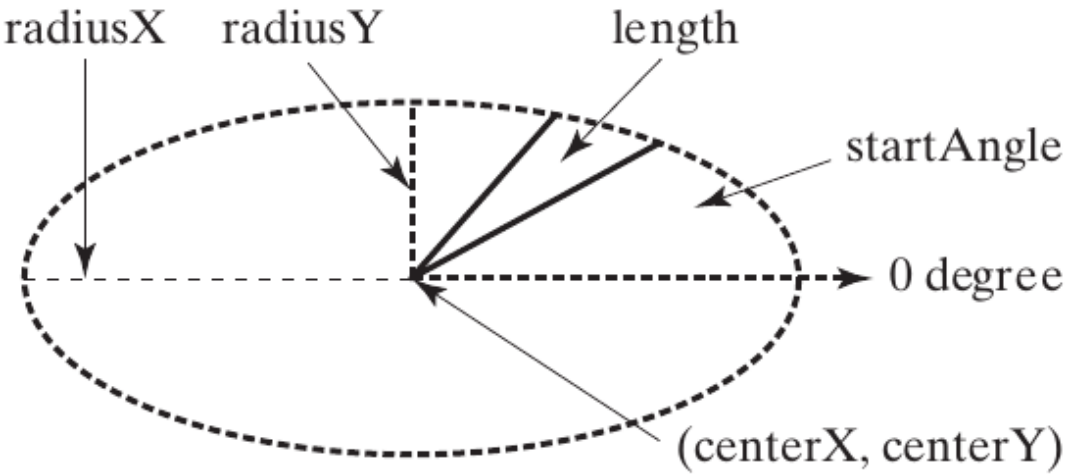
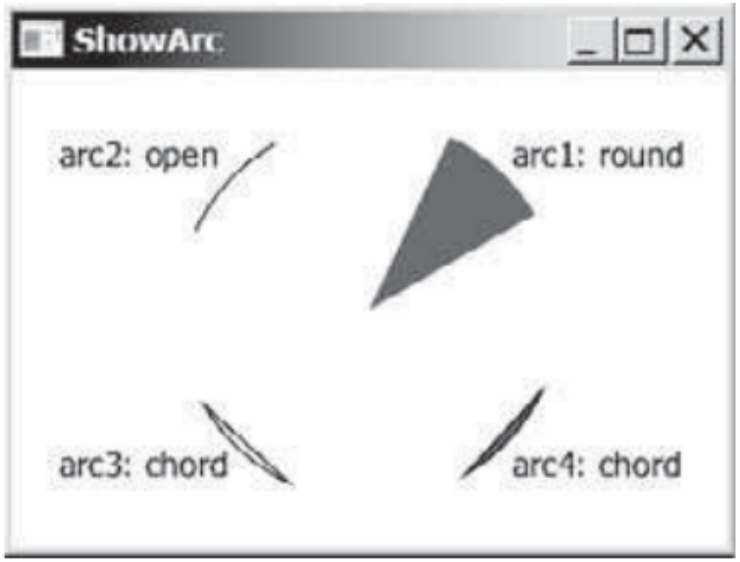


FIGURE 14.35 The `Arc` class defines an arc.

The Arc Class



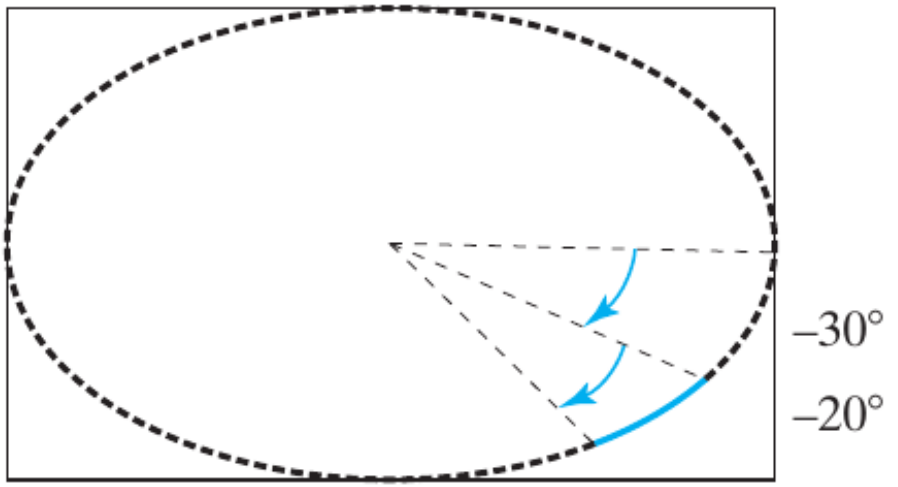
(a) `Arc(centerX, centerY, radiusX, radiusY, startAngle, length)`



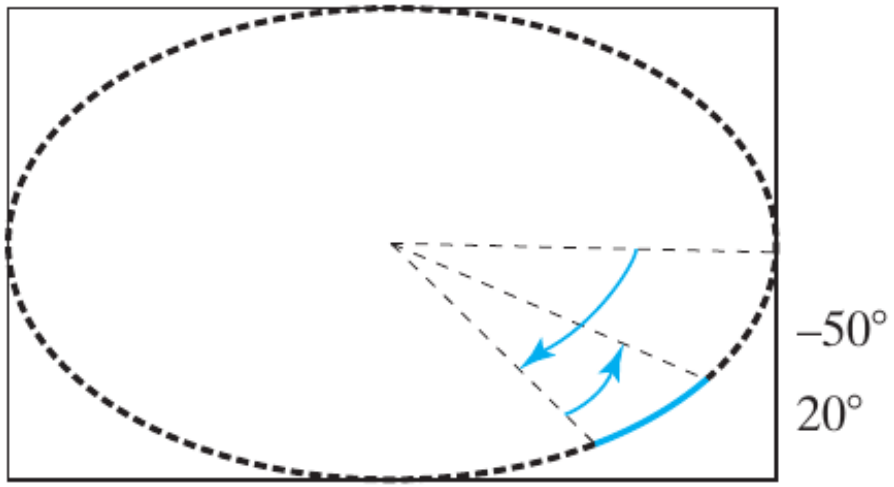
(b) Multiple ellipses are displayed

FIGURE 14.36 An `Arc` object is created to display an arc.

The Arc Class



(a) Negative starting angle –30° and negative spanning angle –20°

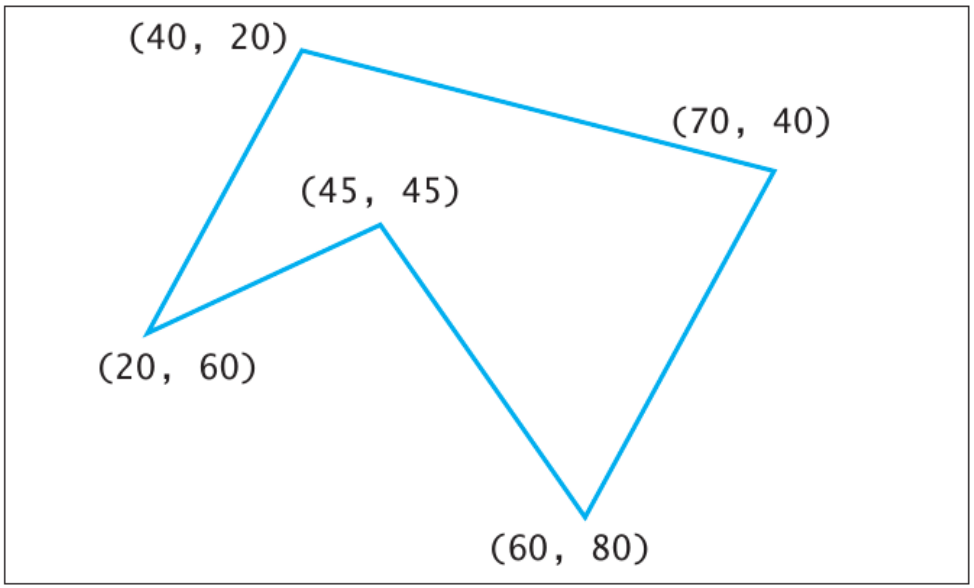


(b) Negative starting angle –50° and positive spanning angle 20°

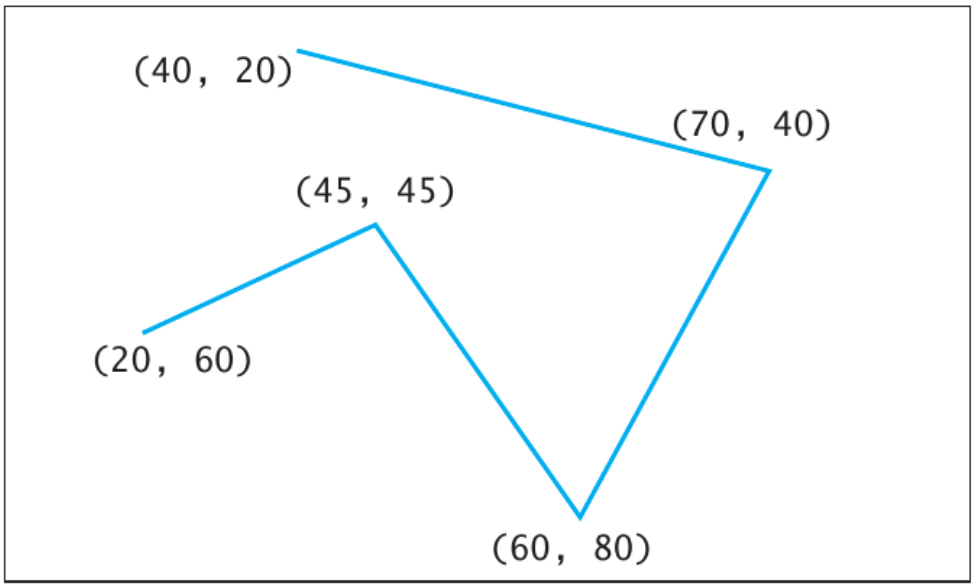
FIGURE 14.37 Angles may be negative.

The Polygon and Polyline Classes

- Polygon defines a polygon that connects a sequence of points.
- Polyline is similar to Polygon but the shape is not automatically closed.
- See Code: ShowPolygon.java



(a) Polygon



(b) Polyline

FIGURE 14.38 `Polygon` is closed and `Polyline` is not closed.

The Polygon and Polyline Classes

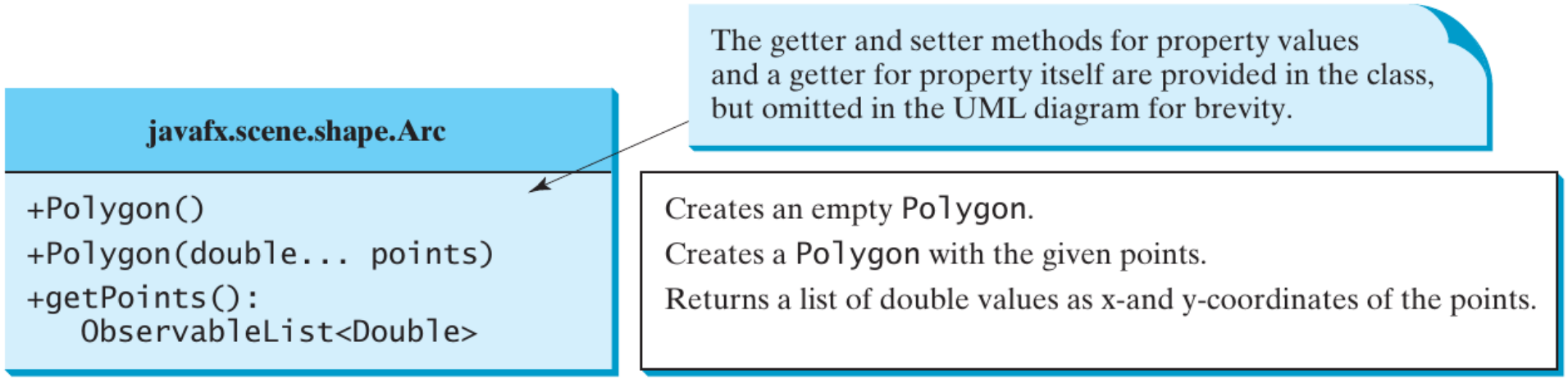


FIGURE 14.39 `Polygon` defines a polygon.

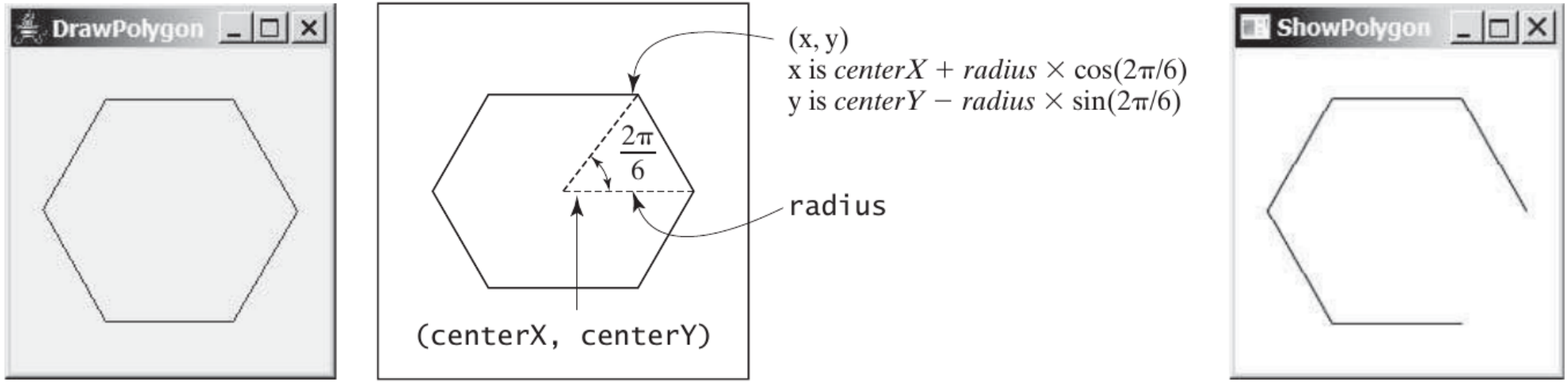


FIGURE 14.40 (a) A `Polygon` is displayed. (b) A `Polyline` is displayed.

Case Study: The ClockPane Class

The ClockPane Class

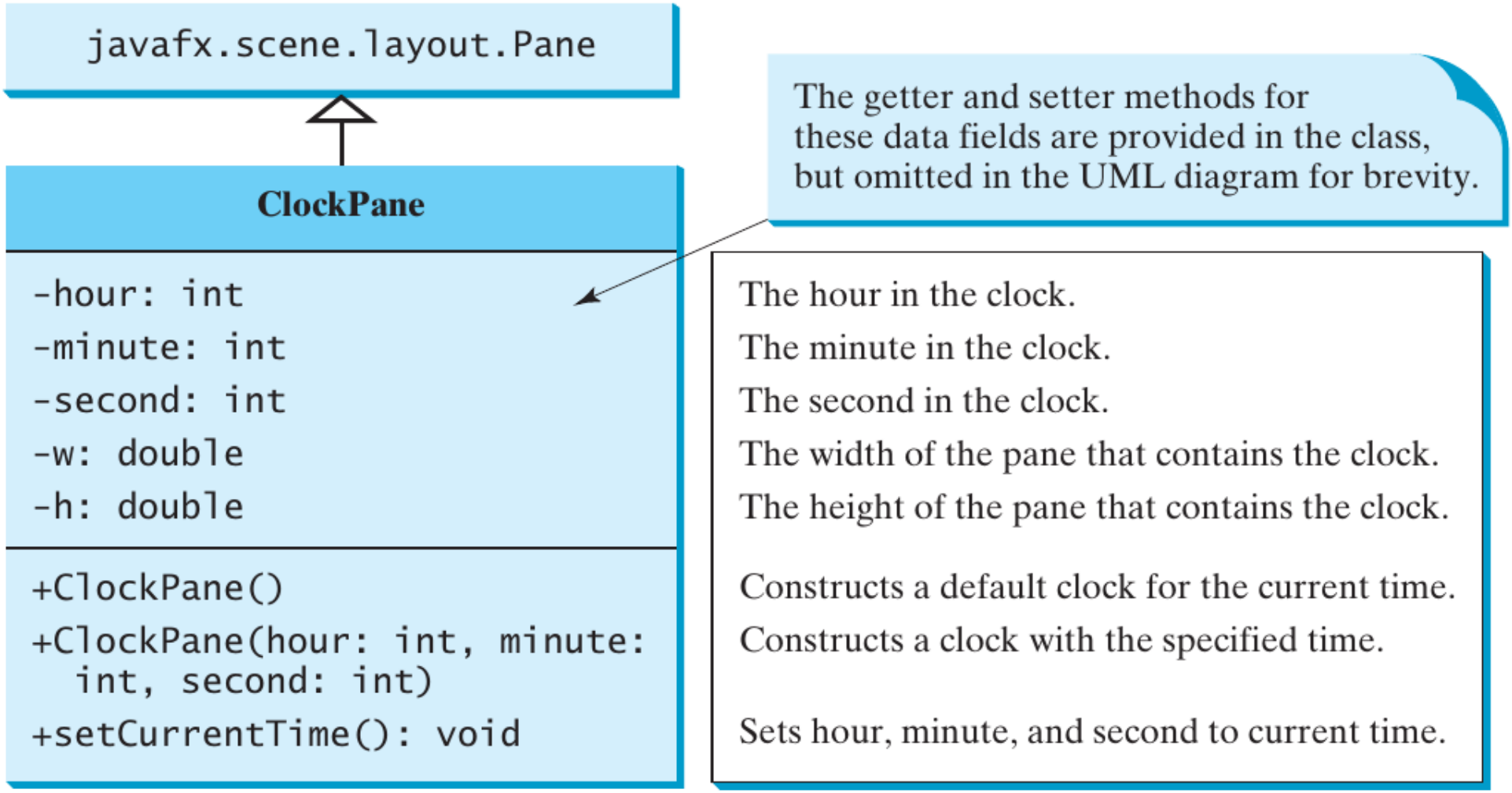


FIGURE 14.41 ClockPane displays an analog clock.

The ClockPane Class

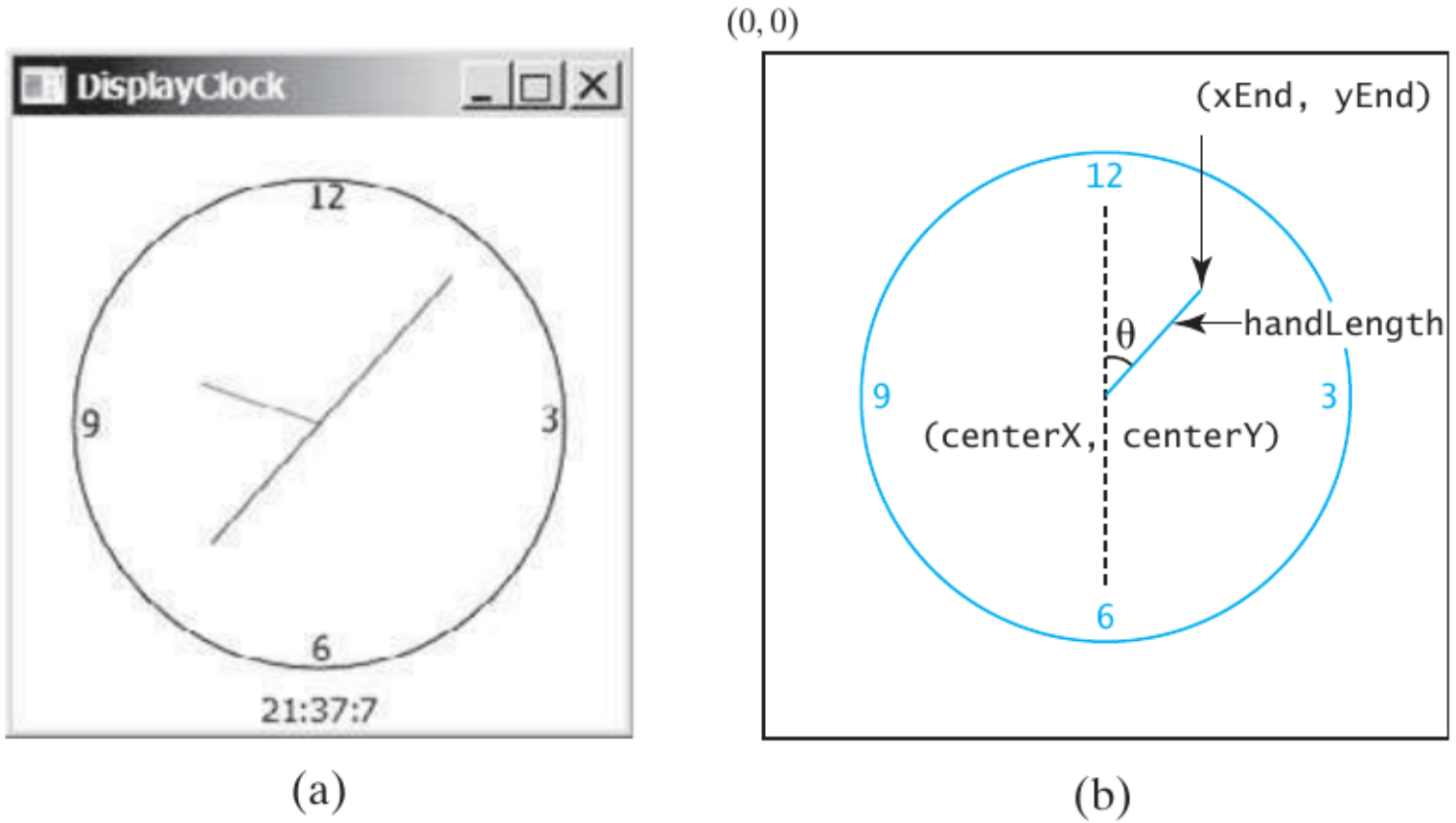


FIGURE 14.42 (a) The `DisplayClock` program displays a clock that shows the current time. (b) The endpoint of a clock hand can be determined, given the spanning angle, the hand length, and the center point.

References

- Liang, Chapter 14: JavaFX Basics