Chapter 14 JavaFX Basics

Motivations

JavaFX is a new framework for developing Java GUI programs. The JavaFX API is an excellent example of how the object-oriented principle is applied. This chapter serves two purposes. First, it presents the basics of JavaFX programming. Second, it uses JavaFX to demonstrate OOP. Specifically, this chapter introduces the framework of JavaFX and discusses JavaFX GUI components and their relationships.

Objectives

To distinguish between JavaFX, Swing, and AWT (§14.2).

To write a simple JavaFX program and understand the relationship among stages, scenes, and nodes (§14.3).

To create user interfaces using panes, UI controls, and shapes (§14.4).

To use binding properties to synchronize property values (§14.5).

To use the common properties **style** and **rotate** for nodes (§14.6).

To create colors using the **Color** class (§14.7).

To create fonts using the **Font** class (§14.8).

To create images using the **Image** class and to create image views using the **ImageView** class (§14.9).

To layout nodes using **Pane**, **StackPane**, **FlowPane**, **GridPane**, **BorderPane**, **HBox**, and **VBox** (§14.10).

To display text using the **Text** class and create shapes using **Line**, **Circle**, **Rectangle**, **Ellipse**, **Arc**, **Polygon**, and **Polyline** (§14.11).

To develop the reusable GUI components **ClockPane** for displaying an analog clock (§14.12).

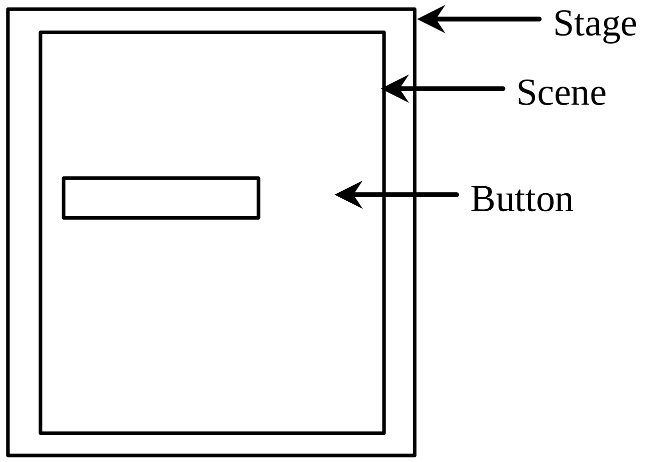
JavaFX vs Swing and AWT

AWT and Swing are replaced by the JavaFX platform for developing rich Internet applications.

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 projects. In addition, AWT is prone to platform-specific bugs. The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as *Swing components*. Swing components are painted directly on canvases using Java code. 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

- The Application class
- Override the start(Stage) method
- Stage, Scene, and Nodes



The main method is only needed for the IDE with limited JavaFX support. Not needed for running from the command line.
start() places GUI in a scene and display the scene in a stage.
starts primaryStage

MyJavaFX Run

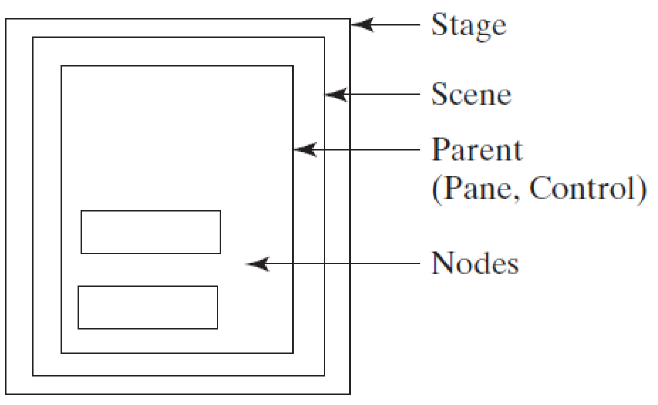
MultipleStageDemo Run

Panes, UI Controls, and Shapes

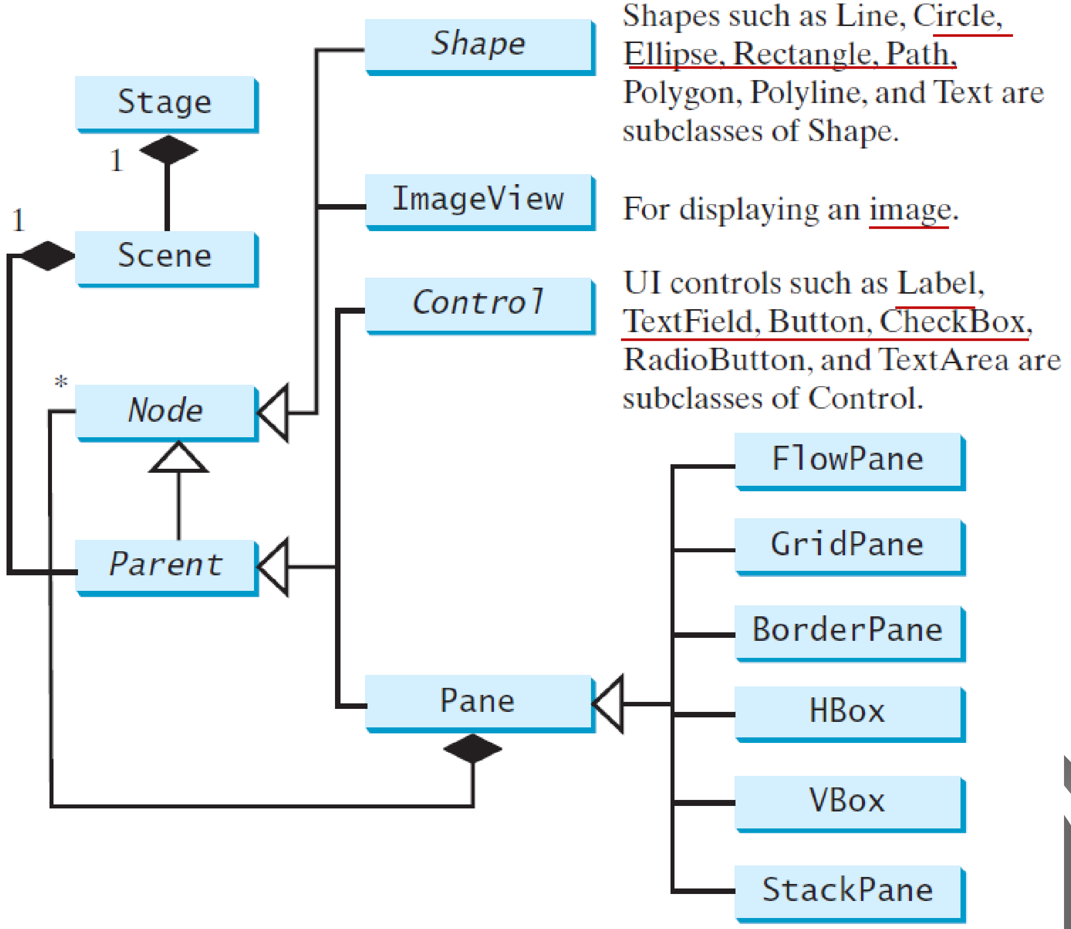
Pane is a container class

Place node in a Pane and put the Pane into a Scene

A Scene can contain a Control or a Pane, but not a Shape or ImageView



(a)



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.

(b)

ButtonInPane

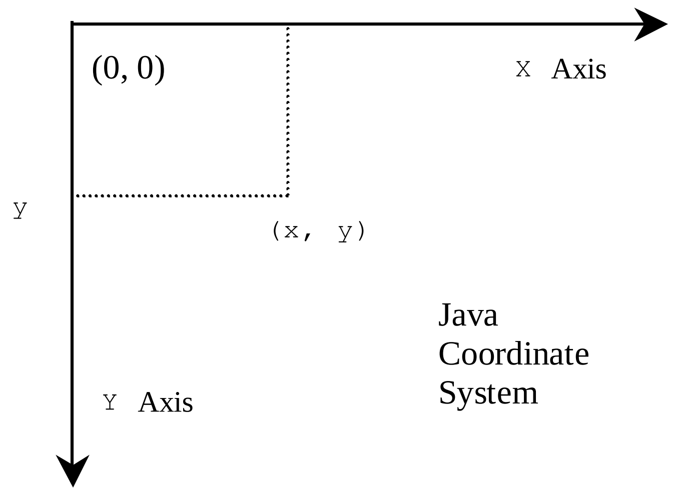
Run

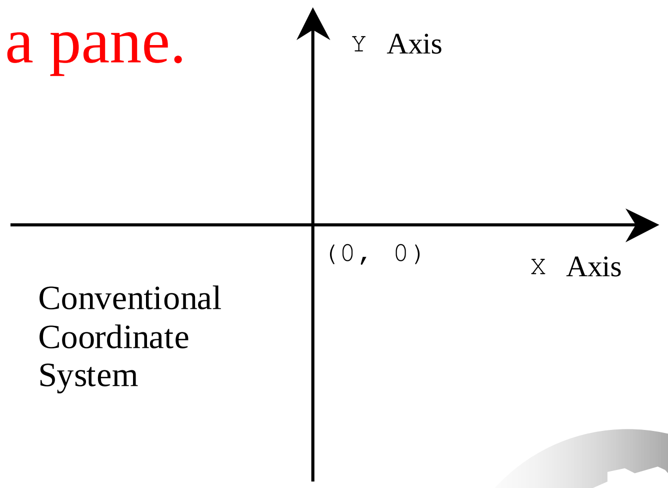
Display a Shape

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

pane.getChildren() returns an instance of ObesrvableList, which can be added with objects.

You must put circle in a pane.





ShowCircle

Run

Binding Properties

JavaFX introduces a new concept called *binding property* that enables a *target object* to be bound to a *source object*. If the value in the source object changes, the target property is also changed automatically. The target object is simply called a *binding object* or a *binding property*.

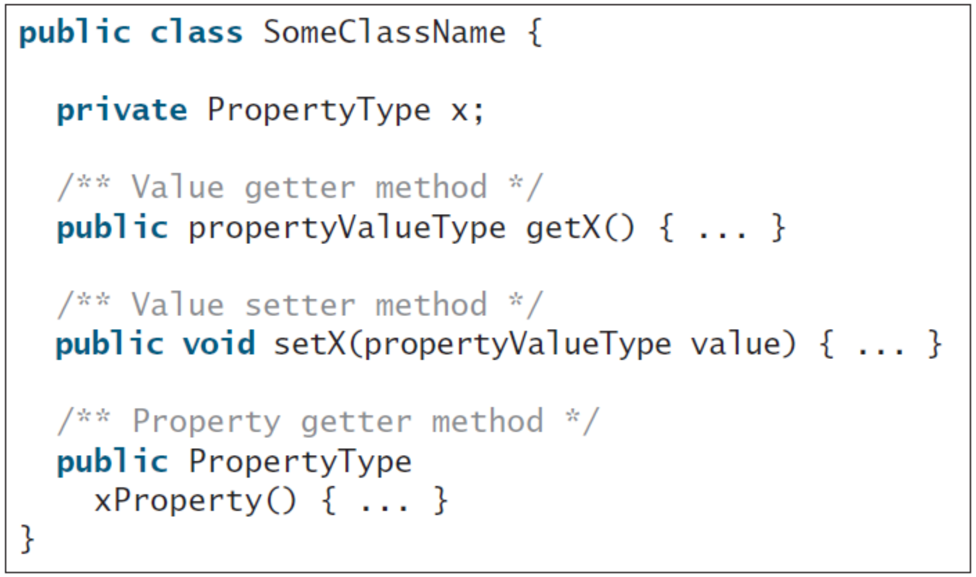
```
circle.centerXProperty().bind(pane.widthProperty().divide(2));
// binds centerX to width / 2

circle.centerYProperty().bind(pane.heightProperty().divide(2));
// binds centerY to height / 2
```

ShowCircleCentered

Run

Binding Property: getter, setter, and property getter

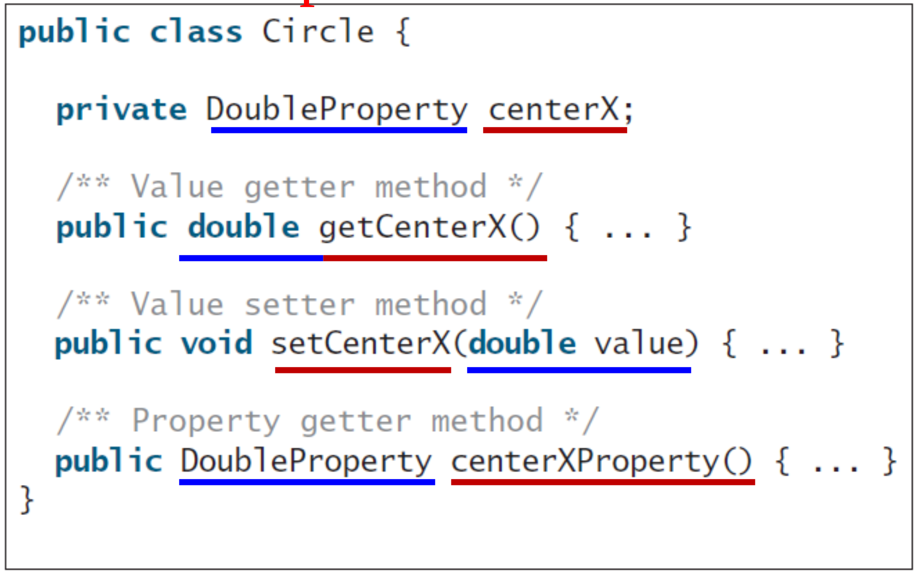
Convention



```
public class SomeClassName {

  private PropertyType x;

  /** Value getter method */
  public propertyValueType getX() { ... }

  /** Value setter method */
  public void setX(propertyValueType value) { ... }

  /** Property getter method */
  public PropertyType
    xProperty() { ... }
}
```

(a) x is a binding property

Example



```
public class Circle {

  private DoubleProperty centerX;

  /** Value getter method */
  public double getCenterX() { ... }

  /** Value setter method */
  public void setCenterX(double value) { ... }

  /** Property getter method */
  public DoubleProperty centerXProperty() { ... }
}
```

(b) centerX is binding property

centerX has a double value, represented in DoubleProperty object

The get and set methods are applied to centerX, …
getCenterX() returns a double value
setCenterX(double) sets centerX to a double value

centerXProperty() returns a DoubleProperty object

Uni/Bidirectional Binding

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

public class BidirectionalBindingDemo {
 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());
     d1.setValue(50.1);
   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());
   }
}
```

target.bind(source);

d1 ← d2

Replace bind() by bindBidirectional() will have bi-directional binding

```
d1 is 2.0 and d2 is 2.0
d1 is 50.1 and d2 is 2.0 (no change)
d1 is 70.2 and d2 is 70.2 (d1 ← d2)
```

Replace bind() by bindBidirectional() will have bi-directional binding

BidirctionalBindingDemo

Run

Common Properties and Methods for Nodes

- style: set a JavaFX CSS style
- rotate: Rotate a node for 45 degrees

CSS: cascading style sheets → style for HTML elements

styleName: value separated by";"

"-fx-border-color: red; -fx-background-color: lightgray;"

Incorrect style is ignored.

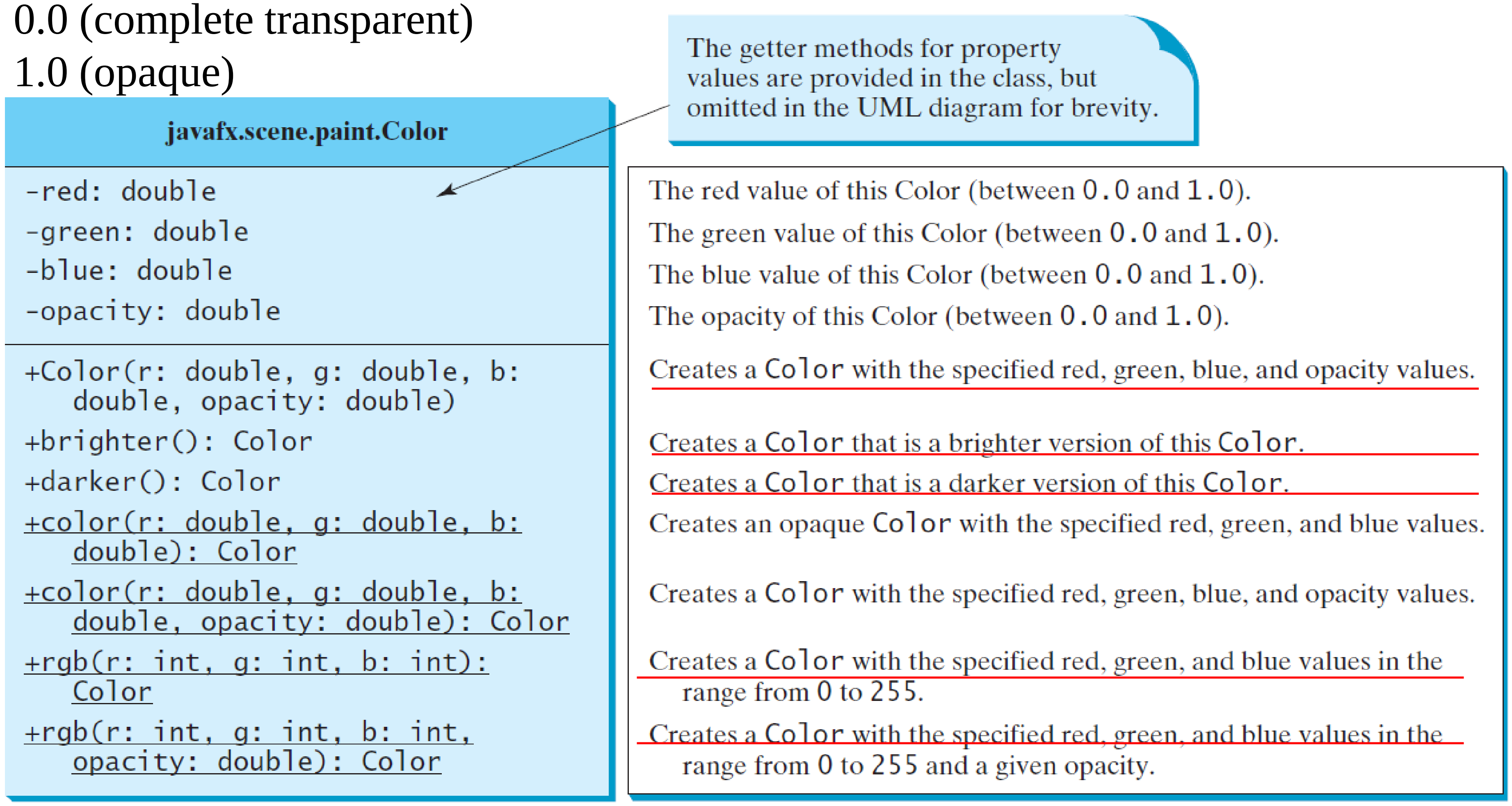
NodeStyleRotateDemo Run

The Color Class

opacity: transparency or translucence

0.0 (complete transparent)

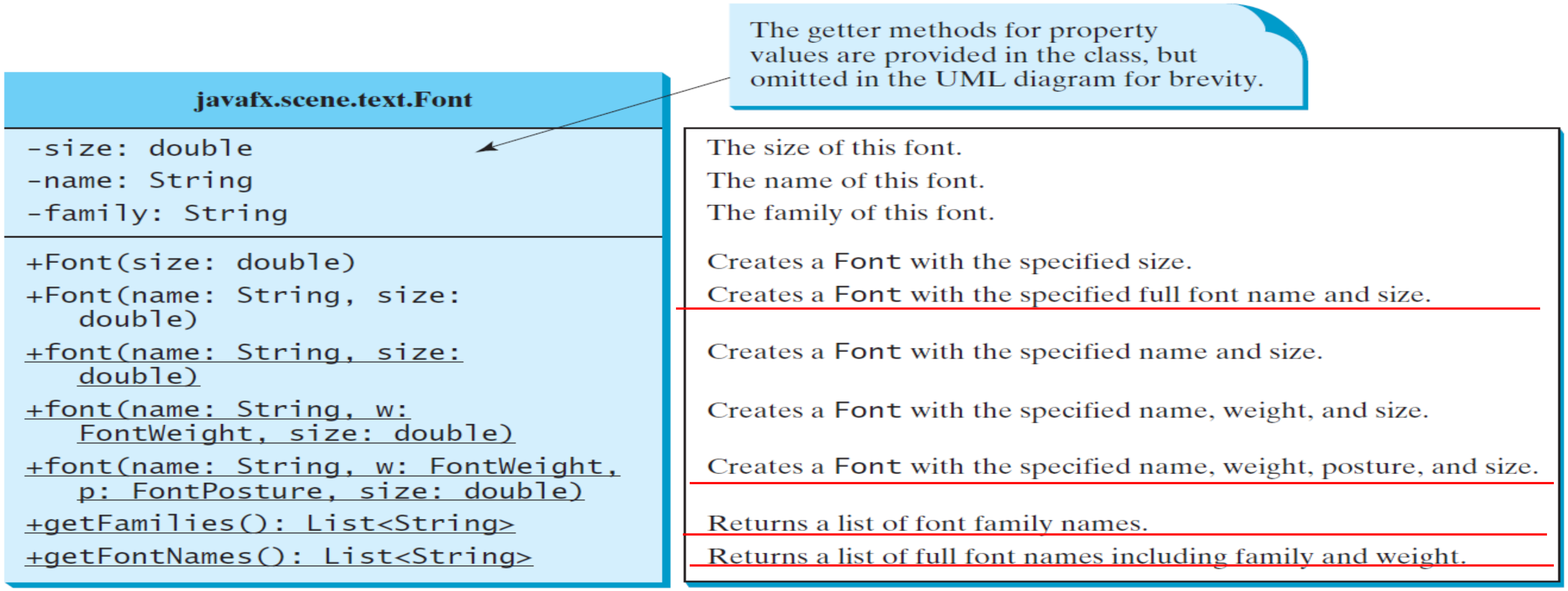
1.0 (opaque)



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 Font Class



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

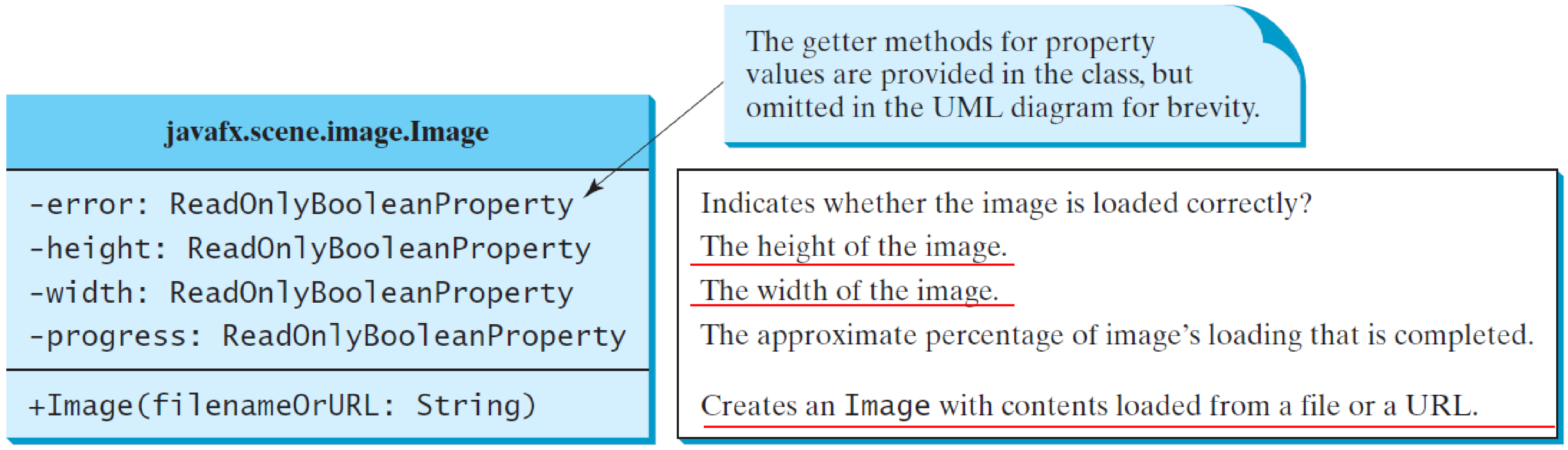
javafx.scene.text.Font

`-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.

```
// name, weight, posture, size
Font.font("Times New Roman", FontWeight.BOLD, FontPosture.ITALIC, 20)

//create a label with text “FavaFX”
Label label = new Label("JavaFX");
```

FontDemo Run

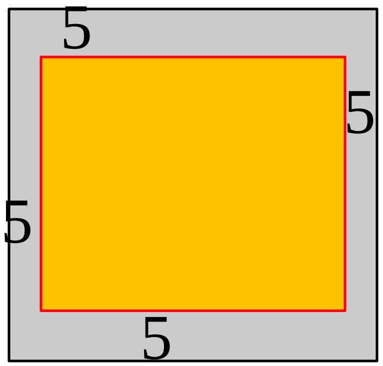
The Image Class



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

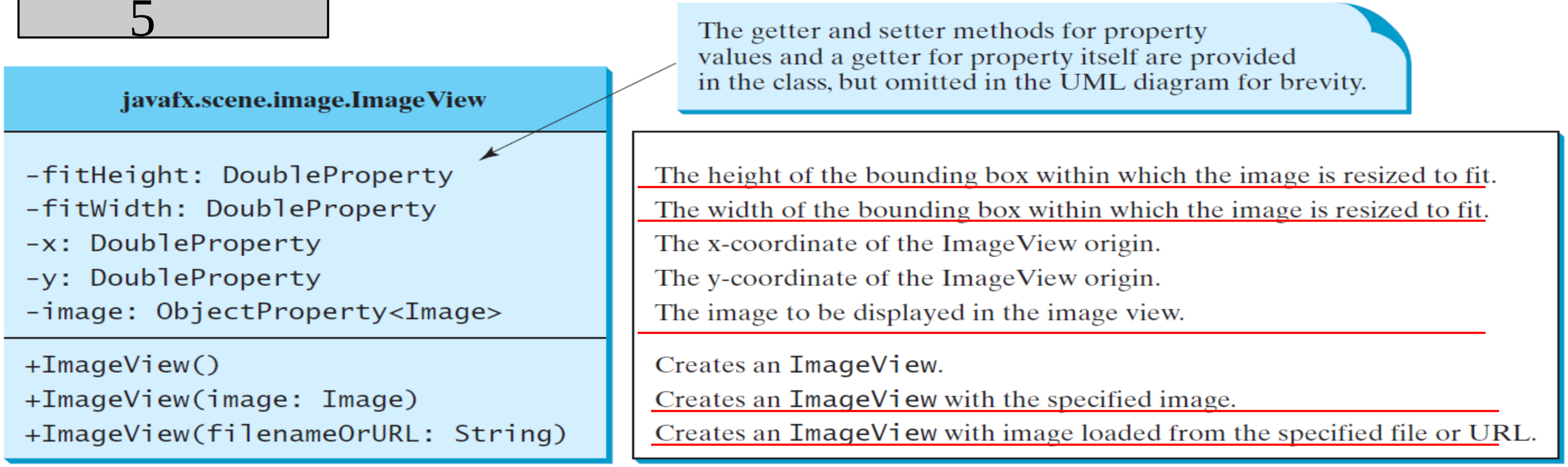
javafx.scene.image.Image	
`-error: ReadOnlyBooleanProperty`	Indicates whether the image is loaded correctly?
`-height: ReadOnlyBooleanProperty`	The height of the image.
`-width: ReadOnlyBooleanProperty`	The width of the image.
`-progress: ReadOnlyBooleanProperty`	The approximate percentage of image's loading that is completed.
`+Image(filenameOrURL: String)`	Creates an `Image` with contents loaded from a file or a URL.

Image **image** = new Image("image/us.gif");
Image **image** = new Image(“http://www. ....");



The ImageView Class

pane.setPadding(new Insets(5, 5, 5, 5));



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.image.ImageView	
`-fitHeight: DoubleProperty`	The height of the bounding box within which the image is resized to fit.
`-fitWidth: DoubleProperty`	The width of the bounding box within which the image is resized to fit.
`-x: DoubleProperty`	The x-coordinate of the ImageView origin.
`-y: DoubleProperty`	The y-coordinate of the ImageView origin.
`-image: ObjectProperty<Image>`	The image to be displayed in the image view.
`+ImageView()`	Creates an `ImageView`.
`+ImageView(image: Image)`	Creates an `ImageView` with the specified image.
`+ImageView(filenameOrURL: String)`	Creates an `ImageView` with image loaded from the specified file or URL.

HBox(10); // is a pane places all objects horizontally.
pane.setPadding(new Insets(5, 5, 5, 5));

pane.getChildren().add(new ImageView(**image**));
pane.getChildren().add(imageView2);
pane.getChildren().add(imageView3);

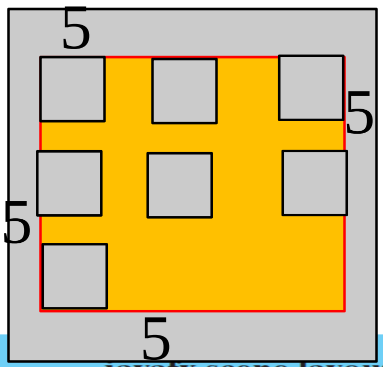
ShowImage

Run

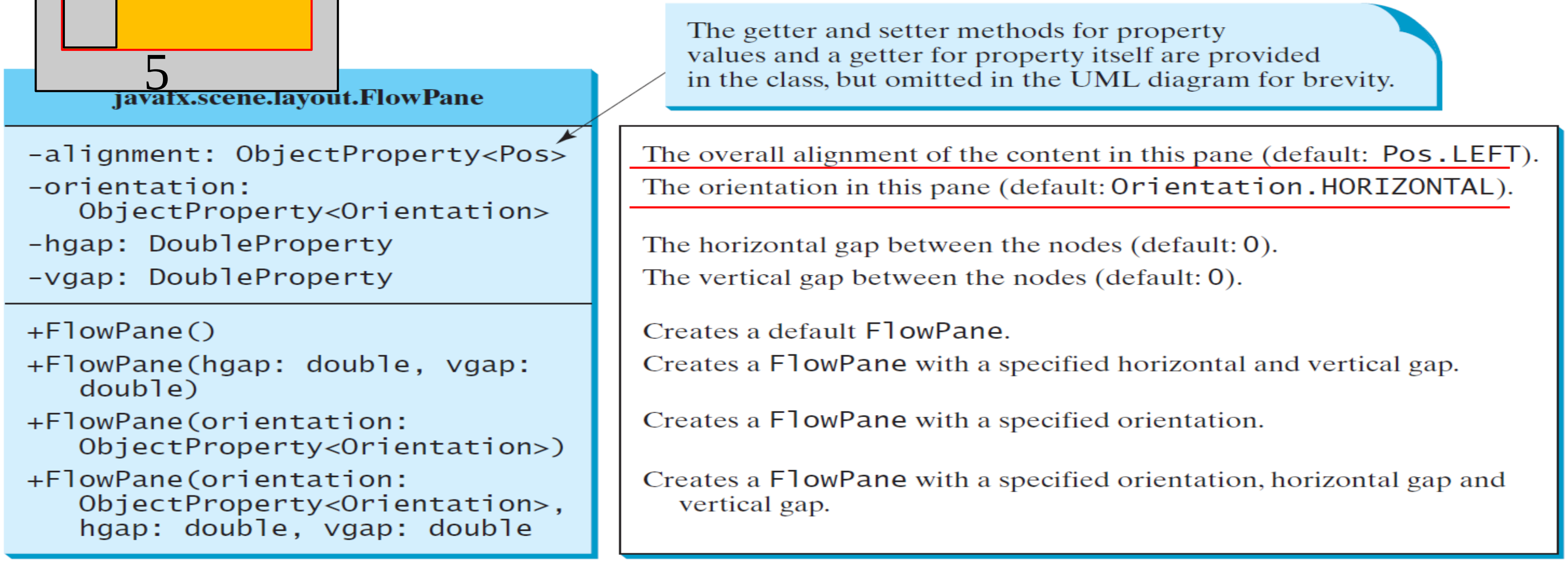
Layout Panes

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.



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.

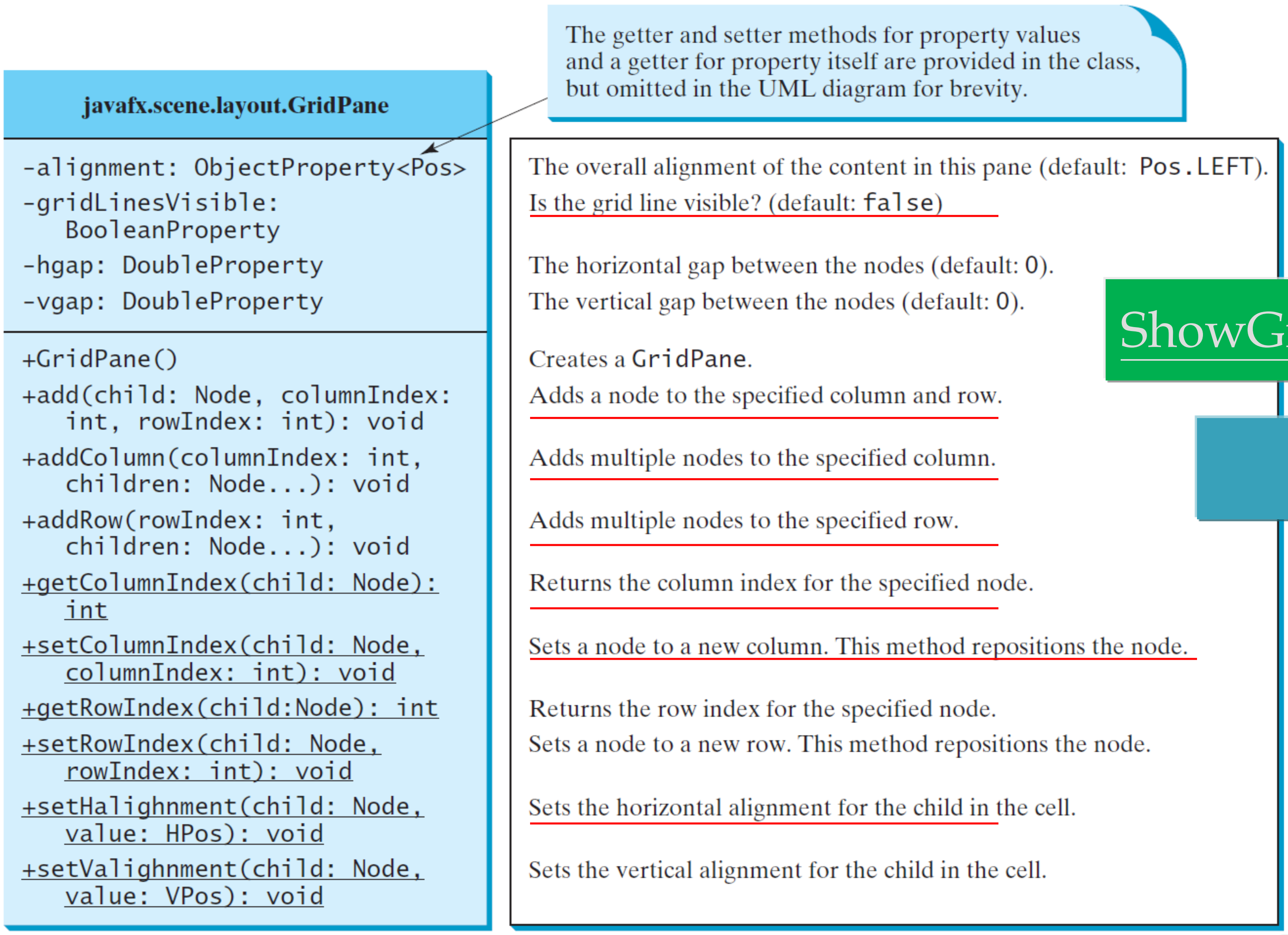
javafx.scene.layout.FlowPane

-alignment: ObjectProperty<Pos>	The overall alignment of the content in this pane (default: Pos.LEFT).
-orientation: ObjectProperty<Orientation>	The orientation in this pane (default: Orientation.HORIZONTAL).
-hgap: DoubleProperty	The horizontal gap between the nodes (default: 0).
-vgap: DoubleProperty	The vertical gap between the nodes (default: 0).
+FlowPane()	Creates a default FlowPane.
+FlowPane(hgap: double, vgap: double)	Creates a FlowPane with a specified horizontal and vertical gap.
+FlowPane(orientation: ObjectProperty<Orientation>)	Creates a FlowPane with a specified orientation.
+FlowPane(orientation: ObjectProperty<Orientation>, hgap: double, vgap: double	Creates a FlowPane with a specified orientation, horizontal gap and vertical gap.

```
Label("First Name:") // create a label
TextField() // create a textfield
addAll(new Label("First Name:"), new TextField(), new Label("MI:"));
tfMi.setPrefColumnCount(1);   // set preferred column count for text
```

ShowFlowPane Run

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.

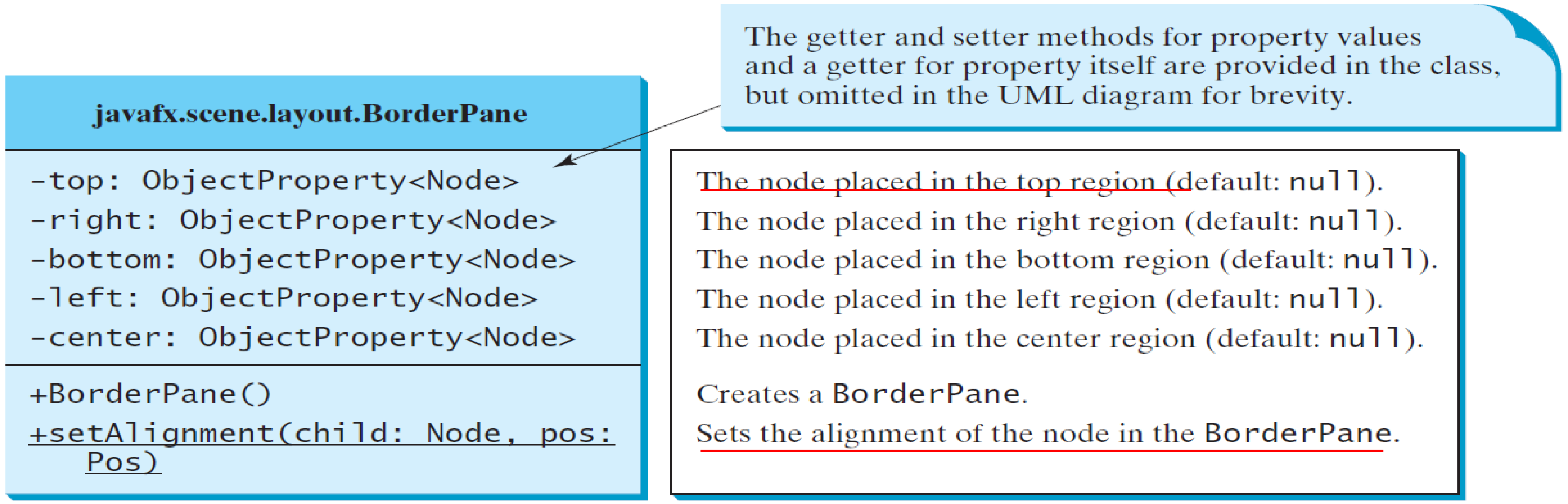
javafx.scene.layout.GridPane

-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.

ShowGridPane

Run

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.

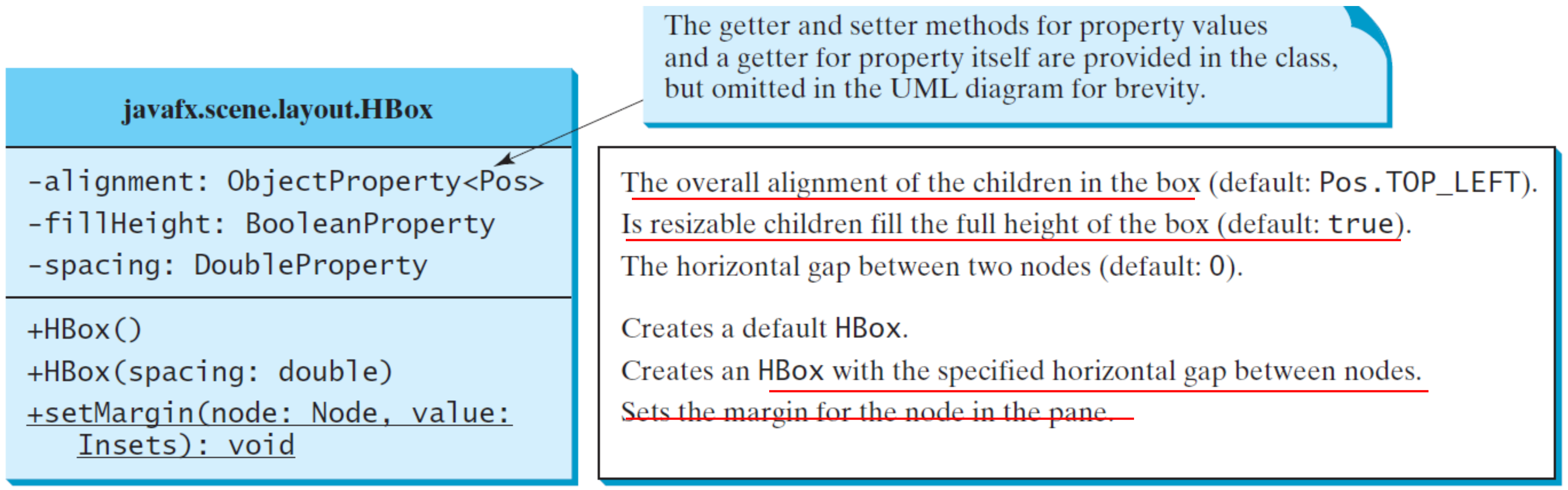
javafx.scene.layout.BorderPane	
-top: ObjectProperty<Node>	The node placed in the top region (default: null).
-right: ObjectProperty<Node>	The node placed in the right region (default: null).
-bottom: ObjectProperty<Node>	The node placed in the bottom region (default: null).
-left: ObjectProperty<Node>	The node placed in the left region (default: null).
-center: ObjectProperty<Node>	The node placed in the center region (default: null).
+BorderPane()	Creates a BorderPane.
+setAlignment(child: Node, pos: Pos)	Sets the alignment of the node in the BorderPane.

```
class CustomPane extends StackPane { /* stackPan is extended */
        public CustomPane(String title) {
                getChildren().add(new Label(title));  /* put label on stack pane */
                setStyle("-fx-border-color: red");
                setPadding(new Insets(11.5, 12.5, 13.5, 14.5)); } }
```

ShowBorderPane

Run

HBox



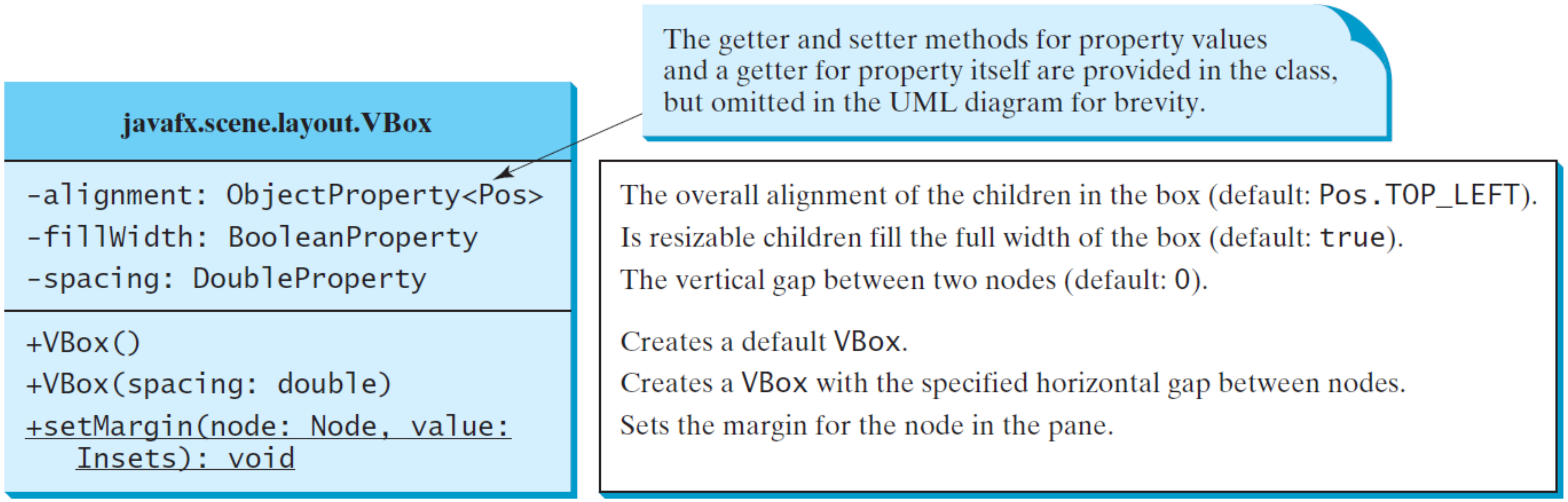
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.layout.HBox

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

A single horizontal row.

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.

A single vertical row.

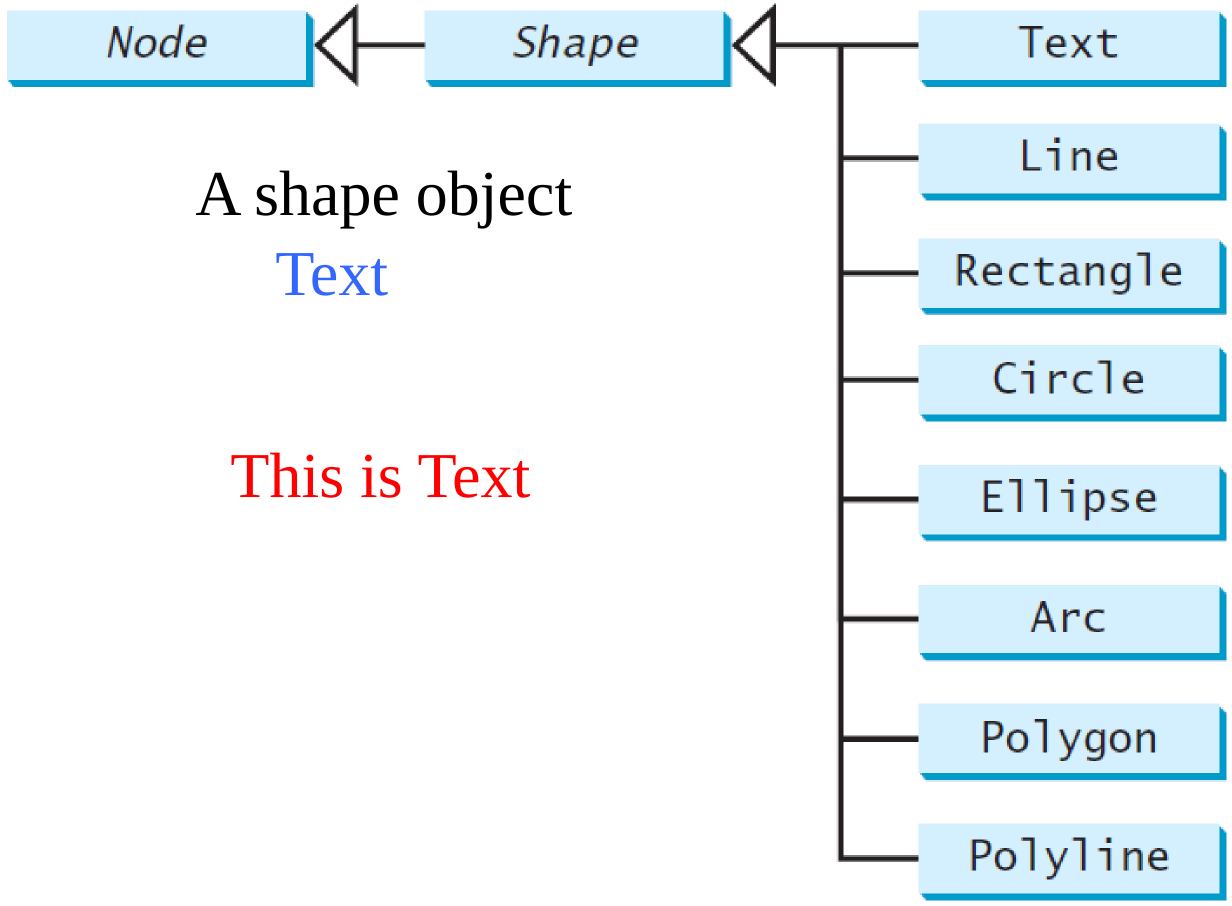
```
Label[] courses = …

for (Label course: courses) {
        VBox.setMargin(course, new Insets(0, 0, 0, 15));
        vBox.getChildren().add(course);
 }
```

ShowHBoxVBox

Run

Shapes

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



A shape object

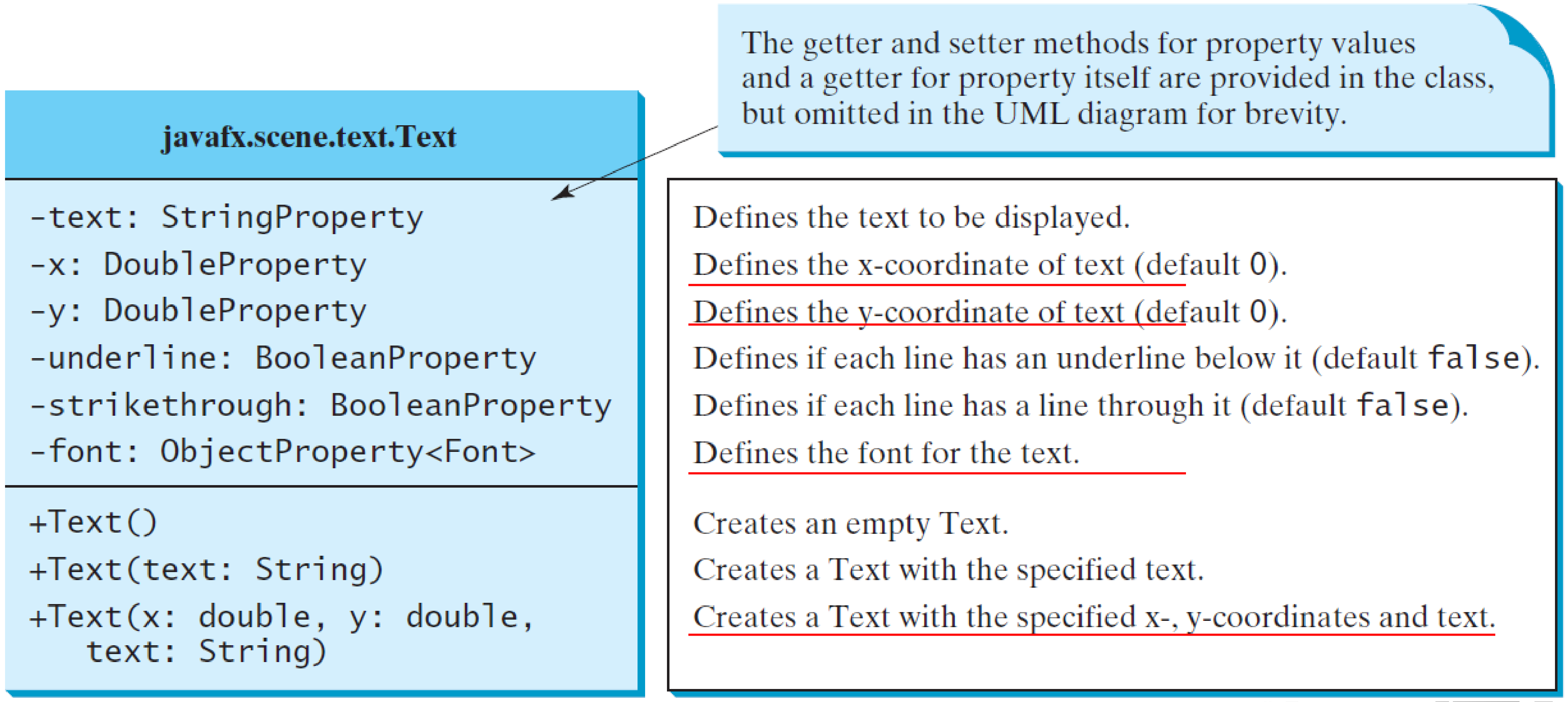
Label

Text

CSCI 1301

This is Text

Text



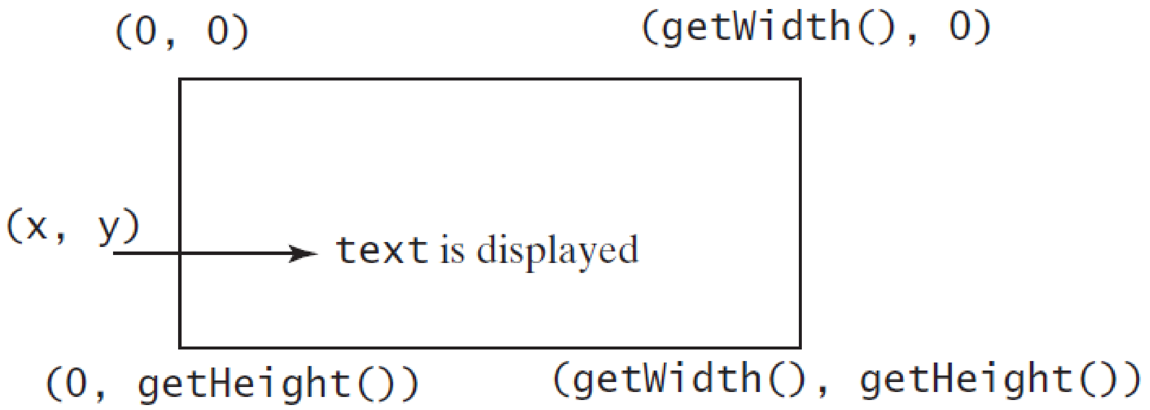
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.text.Text

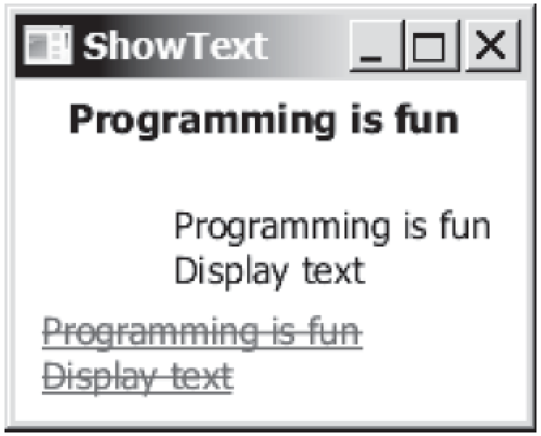
`-text: StringProperty`	Defines the text to be displayed.
`-x: DoubleProperty`	Defines the x-coordinate of text (default `0`).
`-y: DoubleProperty`	Defines the y-coordinate of text (default `0`).
`-underline: BooleanProperty`	Defines if each line has an underline below it (default `false`).
`-strikethrough: 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

Text(20, 20, "Programming is fun")



(a) Text(x, y, text)



Programming is fun

Programming is fun
Display text

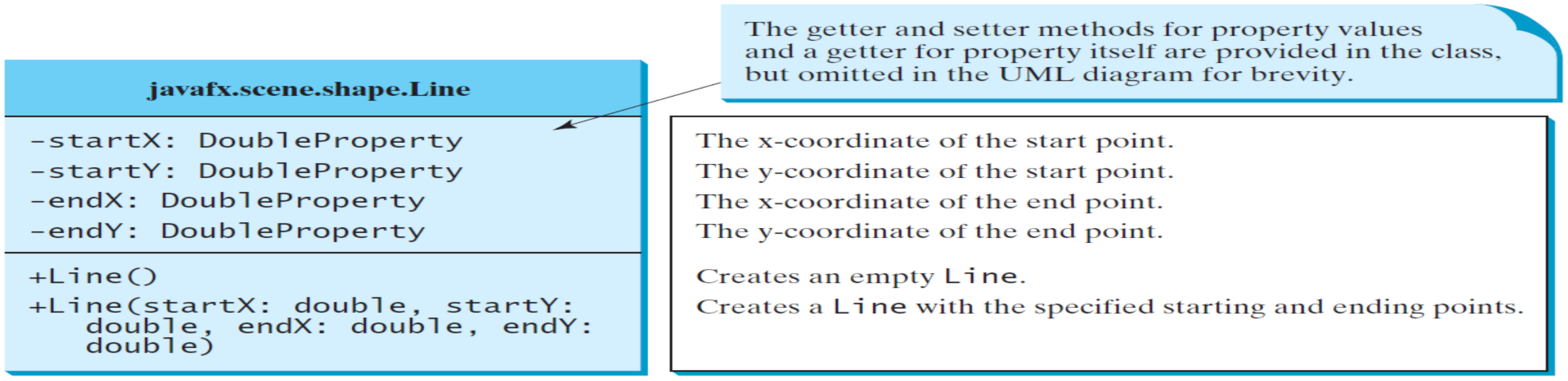
~~Programming is fun~~
~~Display text~~

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

ShowText

Run

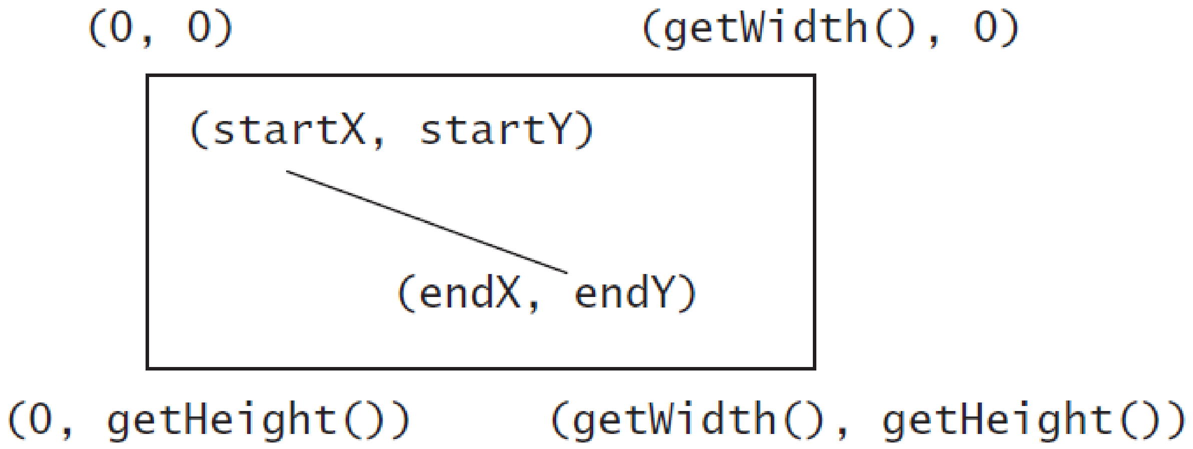
Line



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.

```
line1.setStrokeWidth(5); /* line width */
line1.setStroke(Color.GREEN);
```



ShowLine

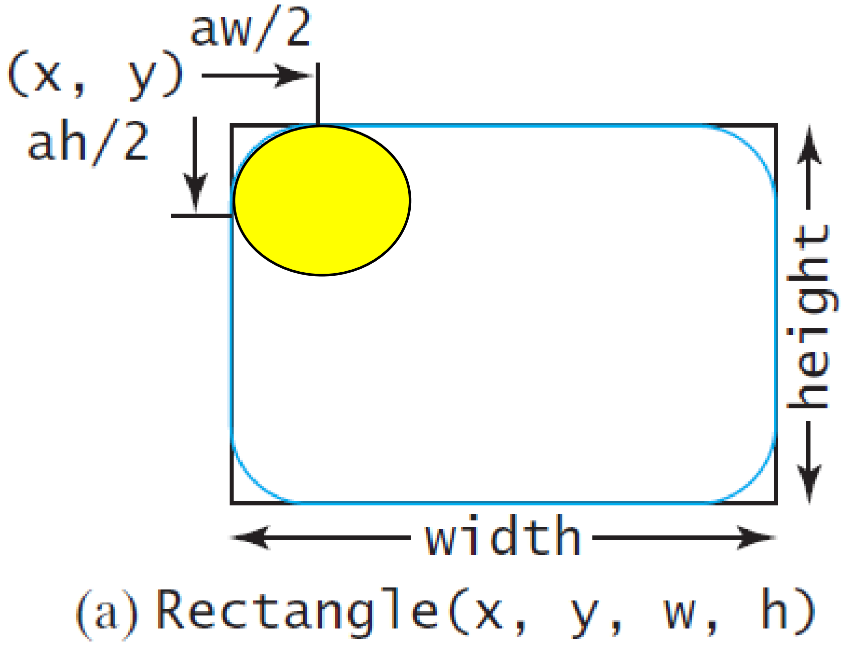
Run

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.

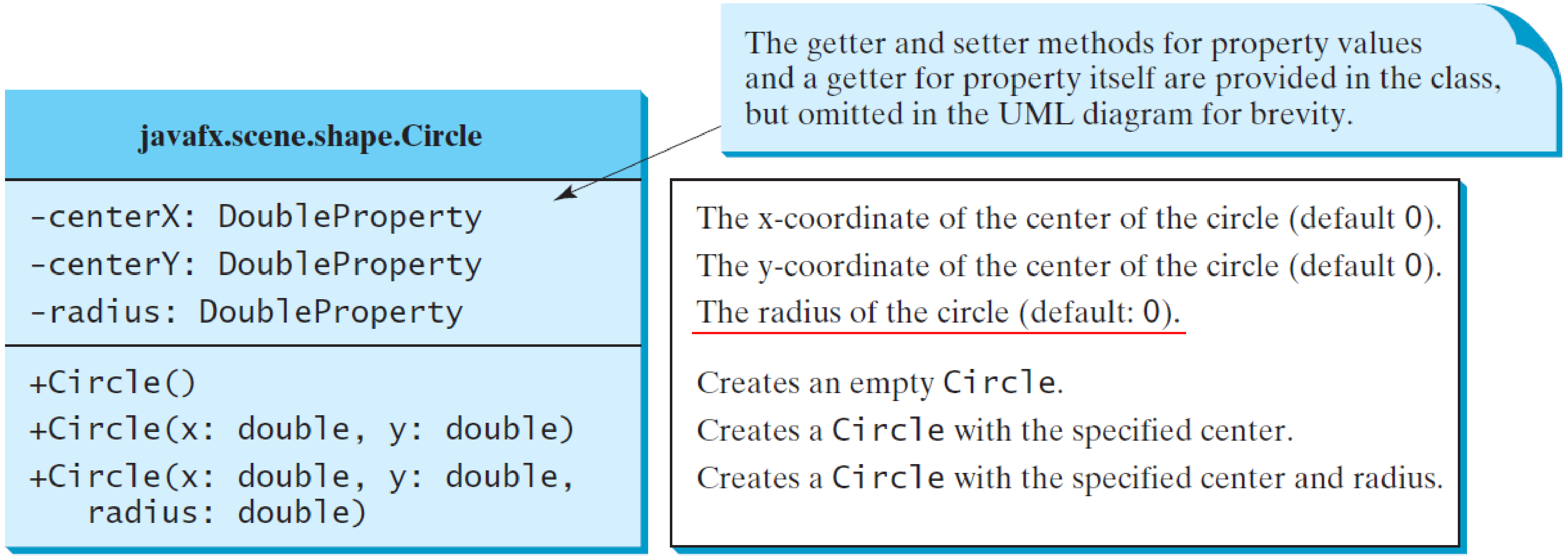
Rectangle Example



```
r3.setArcWidth(15);
r3.setArcHeight(25);

Group group = new Group(); // grouping
    group.getChildren().addAll(
new Text(10, 27, "r1"), r1,
new Text(10, 67, "r2"), r2,
new Text(10, 107, "r3"), r3);

r.setStroke(Color.color(Math.random(),
Math.random(), Math.random()));
```

ShowRectangle Run

Circle

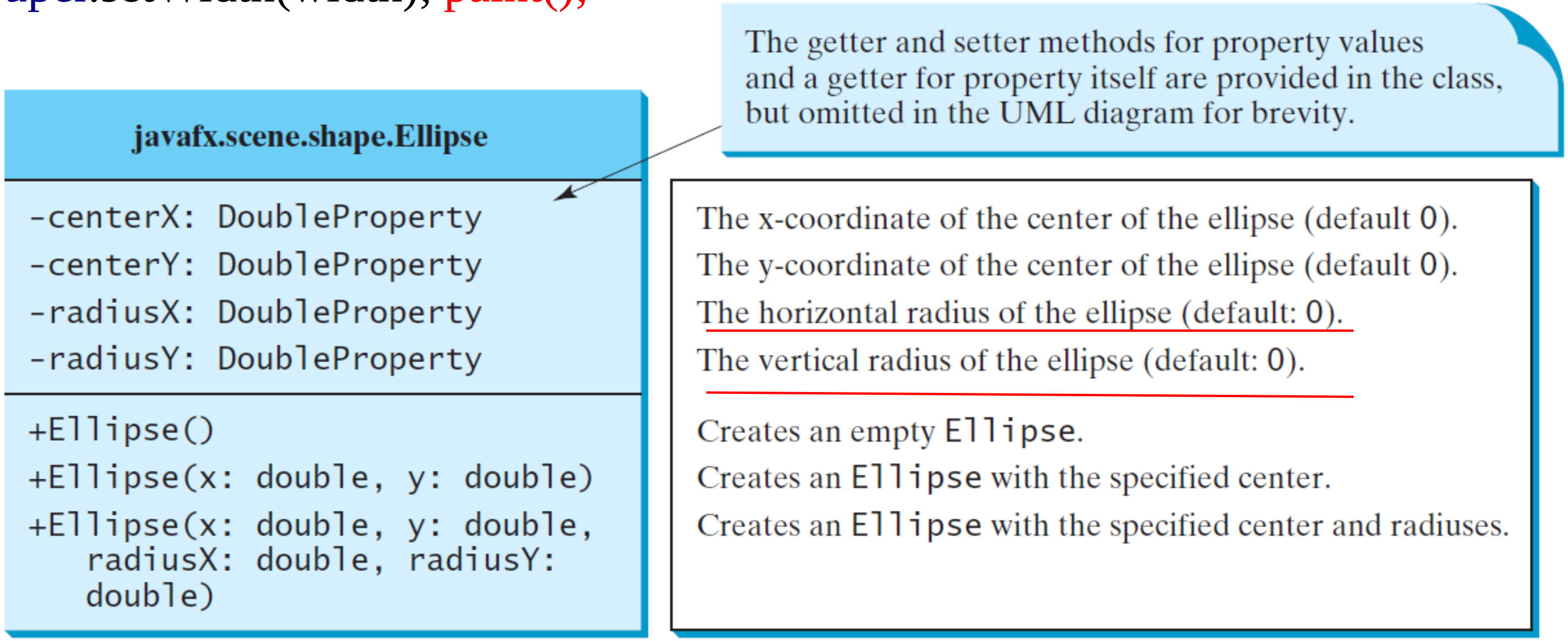


The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.Circle	
`-centerX: DoubleProperty`	The x-coordinate of the center of the circle (default 0).
`-centerY: DoubleProperty`	The y-coordinate of the center of the circle (default 0).
`-radius: DoubleProperty`	The radius of the circle (default: 0).
`+Circle()`	Creates an empty `Circle`.
`+Circle(x: double, y: double)`	Creates a `Circle` with the specified center.
`+Circle(x: double, y: double, radius: double)`	Creates a `Circle` with the specified center and radius.

Ellipse

```
@Override
public void setWidth(double width)
{ super.setWidth(width); paint();
}
```



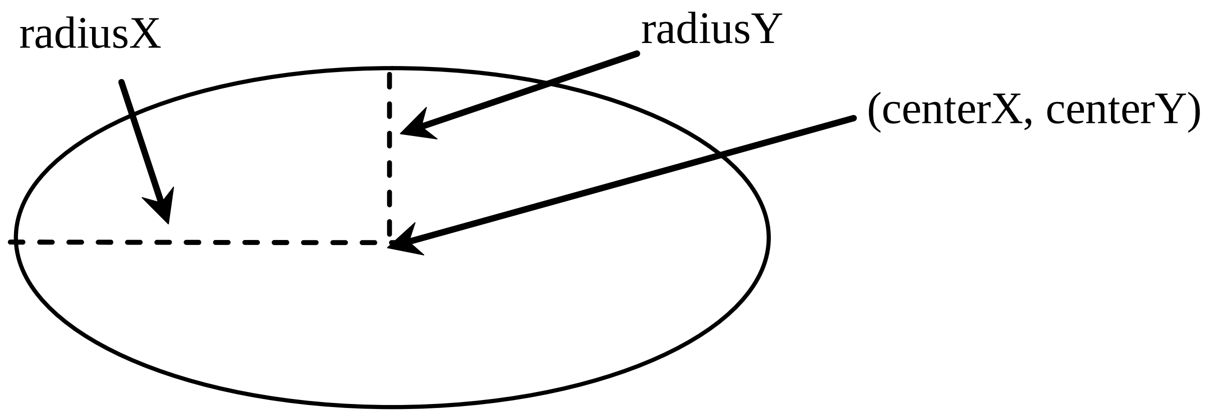
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.Ellipse

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty

+Ellipse()
+Ellipse(x: double, y: double)
+Ellipse(x: double, y: double,
   radiusX: double, radiusY:
   double)
```

The x-coordinate of the center of the ellipse (default 0).
The y-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).

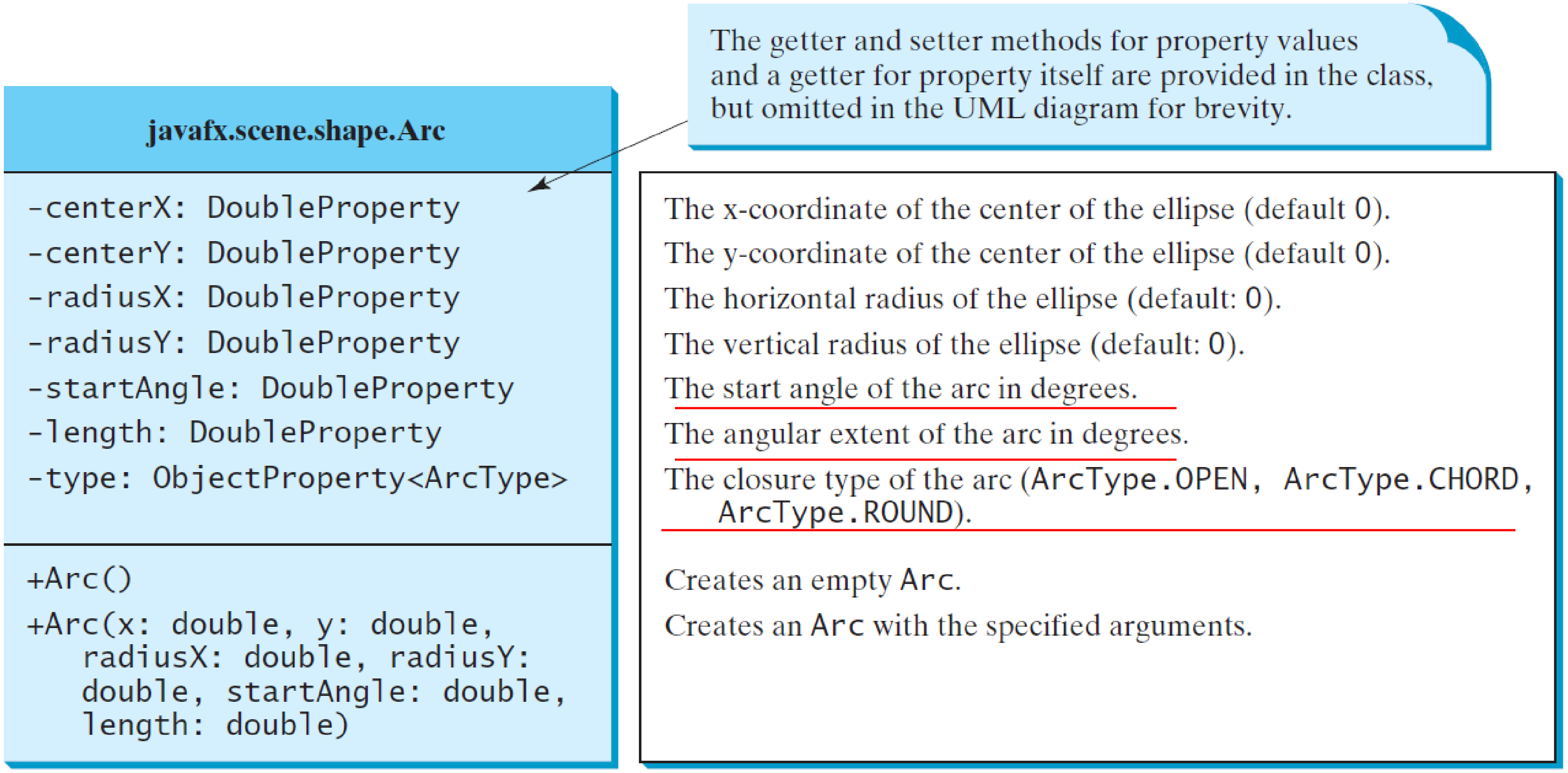
Creates an empty Ellipse.
Creates an Ellipse with the specified center.
Creates an Ellipse with the specified center and radiuses.



ShowEllipse

Run

Arc

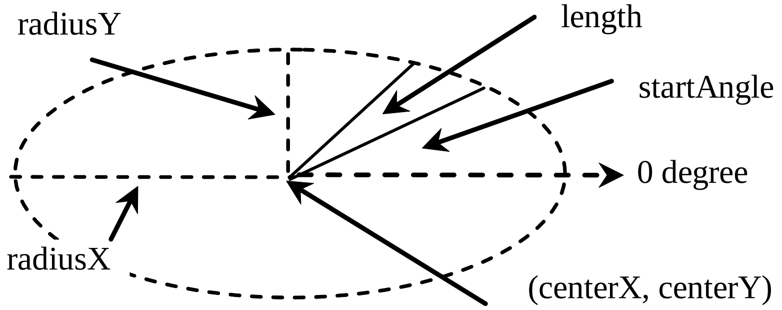


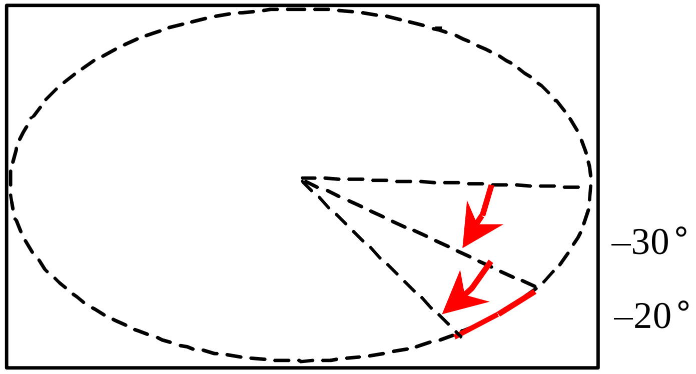
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.shape.Arc	
`-centerX: DoubleProperty`	The x-coordinate of the center of the ellipse (default 0).
`-centerY: DoubleProperty`	The y-coordinate of the center of the ellipse (default 0).
`-radiusX: DoubleProperty`	The horizontal radius of the ellipse (default: 0).
`-radiusY: DoubleProperty`	The vertical radius of the ellipse (default: 0).
`-startAngle: DoubleProperty`	The start angle of the arc in degrees.
`-length: DoubleProperty`	The angular extent of the arc in degrees.
`-type: ObjectProperty<ArcType>`	The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).
`+Arc()`	Creates an empty `Arc`.
`+Arc(x: double, y: double, radiusX: double, radiusY: double, startAngle: double, length: double)`	Creates an `Arc` with the specified arguments.

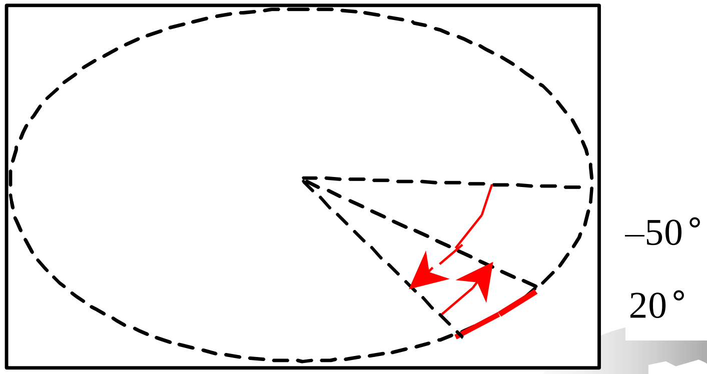
Arc Examples

```
arc1.setType(ArcType.ROUND); // Set arc type
arc2.setType(ArcType.OPEN);
arc3.setType(ArcType.CHORD);
```





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

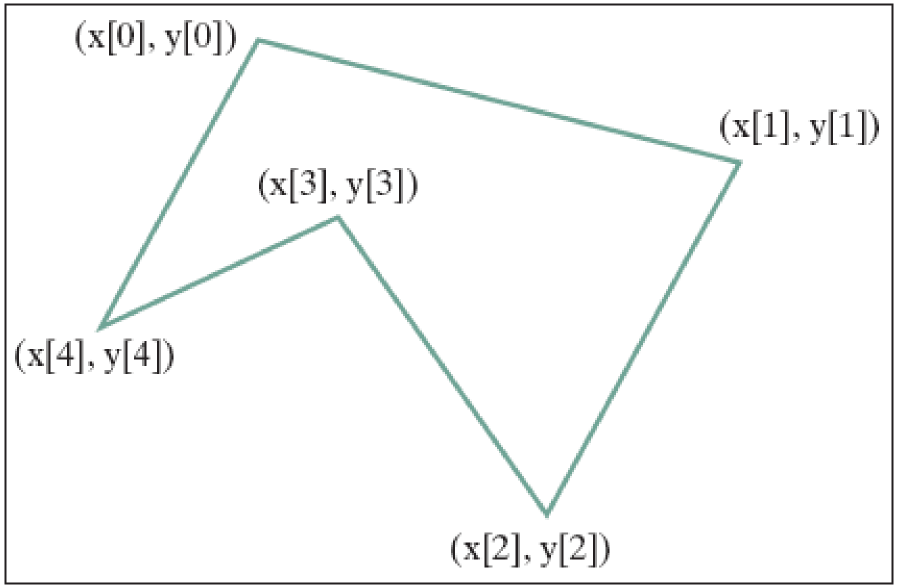


(b) Negative starting angle –50° and positive spanning angle 20°

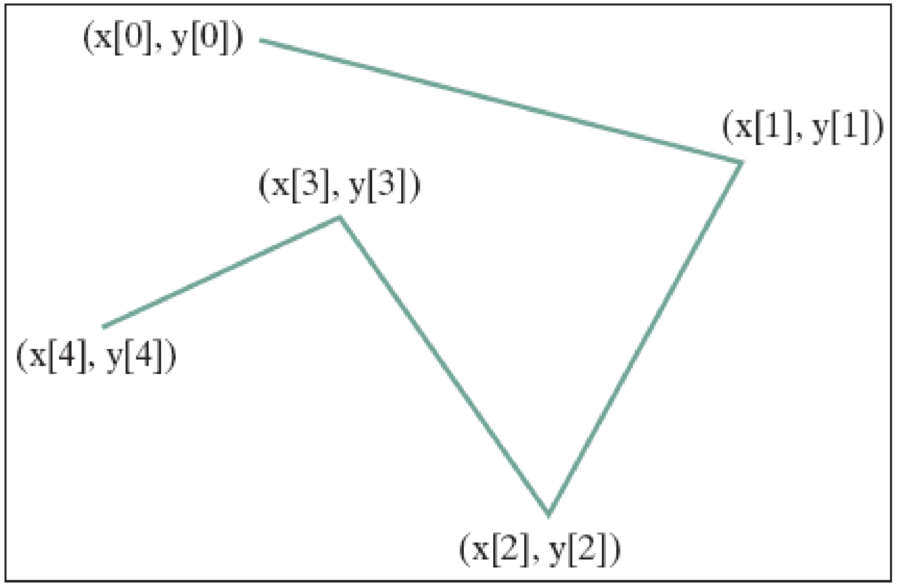
ShowArc

Run

Polygon and Polyline

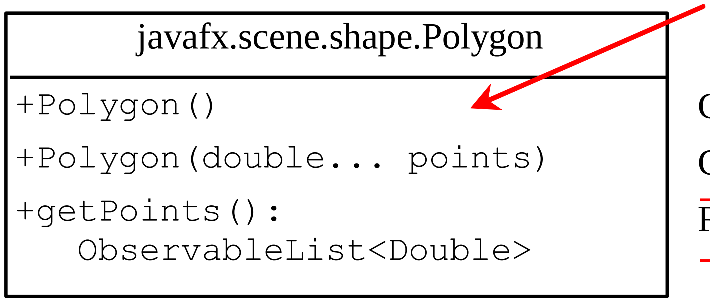


(a) Polygon



(b) Polyline

Polygon



```
javafx.scene.shape.Polygon
----------------------------------
+Polygon()
+Polygon(double... points)
+getPoints():
   ObservableList<Double>
```

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.

Creates an empty polygon.

Creates a polygon with the given points.

Returns a list of double values as x- and y-coordinates of the points.

ObservableList<Double> list = polygon.getPoints();

Returns a list object, which can be added with X1, Y1, X2, Y2, … (all Double values)

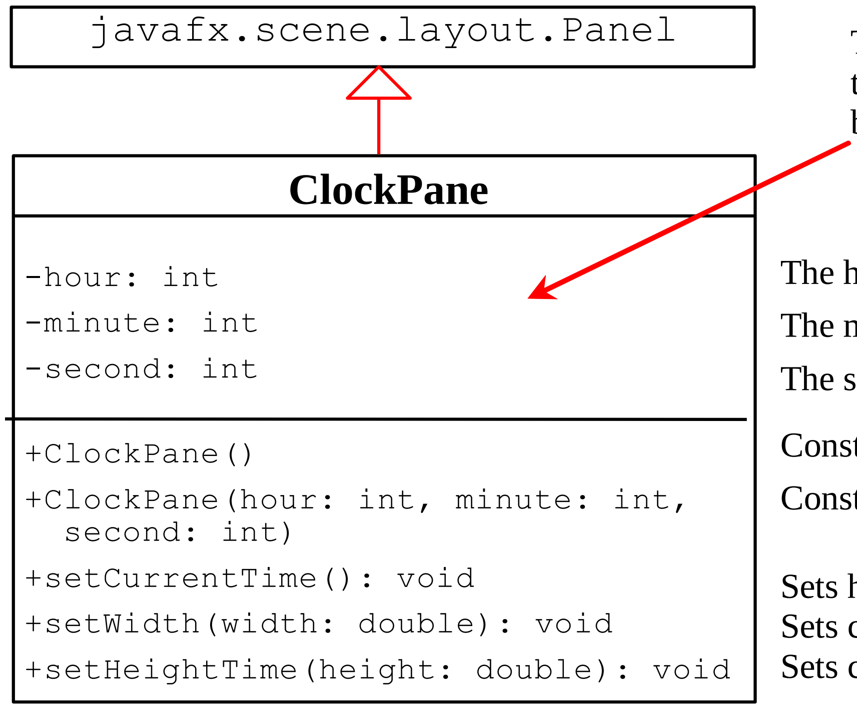
```
// Add points to the polygon list
   for (int i = 0; i < 6; i++) {
     list.add(centerX + radius * Math.cos(2 * i * Math.PI / 6));
     list.add(centerY - radius * Math.sin(2 * i * Math.PI / 6));
```

ShowPolygon

Run

Case Study: The ClockPane Class

This case study develops a class that displays a clock on a pane.



javafx.scene.layout.Panel
ClockPane
-hour: int
-minute: int
-second: int
+ClockPane()
+ClockPane(hour: int, minute: int, second: int)
+setCurrentTime(): void
+setWidth(width: double): void
+setHeightTime(height: double): void

The `getter` and `setter` methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The hour in the clock.

The minute in the clock.

The second in the clock.

Constructs a default clock for the current time.

Constructs a clock with the specified time.

Sets hour, minute, and second for current time.
Sets clock pane's width and repaint the clock,
Sets clock pane's height and repaint the clock,

Use the ClockPane Class

```
getChildren().clear(); // Clear the clock
getChildren().addAll(circle, t1, t2, t3, t4, sLine, mLine, hLine); // Add all drawings
```

DisplayClock

Run