Chapter 14 JavaFX Basics

Motivations

- JavaFX is a new framework for developing Java GUI programs.
- 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.

JavaFX vs Swing and AWT

- Swing and AWT 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.

JavaFX vs Swing and AWT

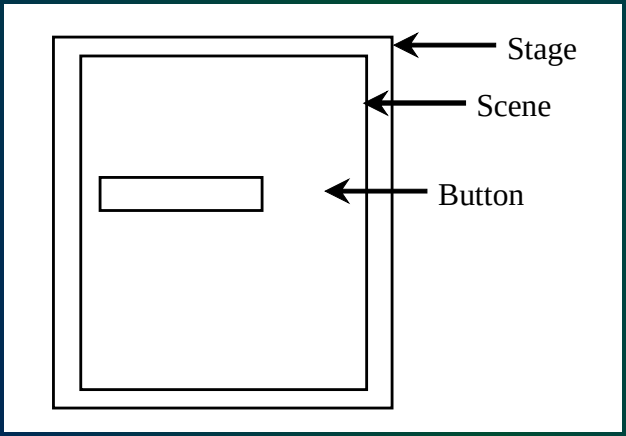
- The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as *Swing components*.
- Swing components are painted directly on canvases using Java code.
- Swing components depend less on the target platform and use less of the native GUI resource.
- With the release of Java 8, Swing is replaced by a completely new GUI platform known as *JavaFX*.

JavaFX Basics

- The abstract javafx.application.Application class defines the essential framework for writing JavaFX programs.
- Every JavaFX program is defined in a class that extends javafx.application.Application

Basic Structure of JavaFX

- A basic JavaFX program contains a class that extends javafx.application.Application class.
- This class needs to override the start(Stage) method to define the structure and contents of the GUI (Stage, Scene, and Nodes).
- Call the launch method from main (if needed). The launch method is a static method defined in the Application class for launching a stand-alone JavaFX application.
- Stage is a window for displaying a scene that contains nodes.
- Multiple stages can be displayed in a JavaFX program.



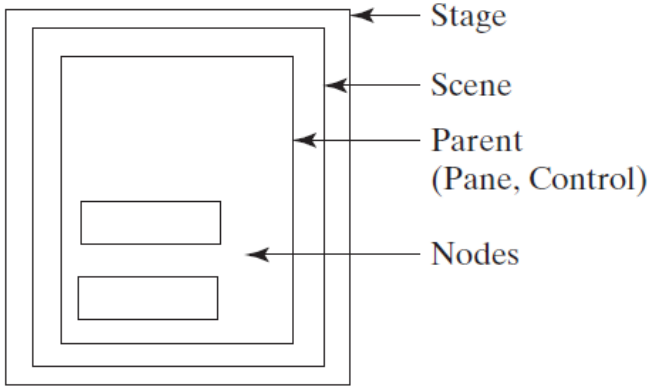
Basic Structure of JavaFX

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

public class JavaFXApplication1 extends Application {

    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

}
```

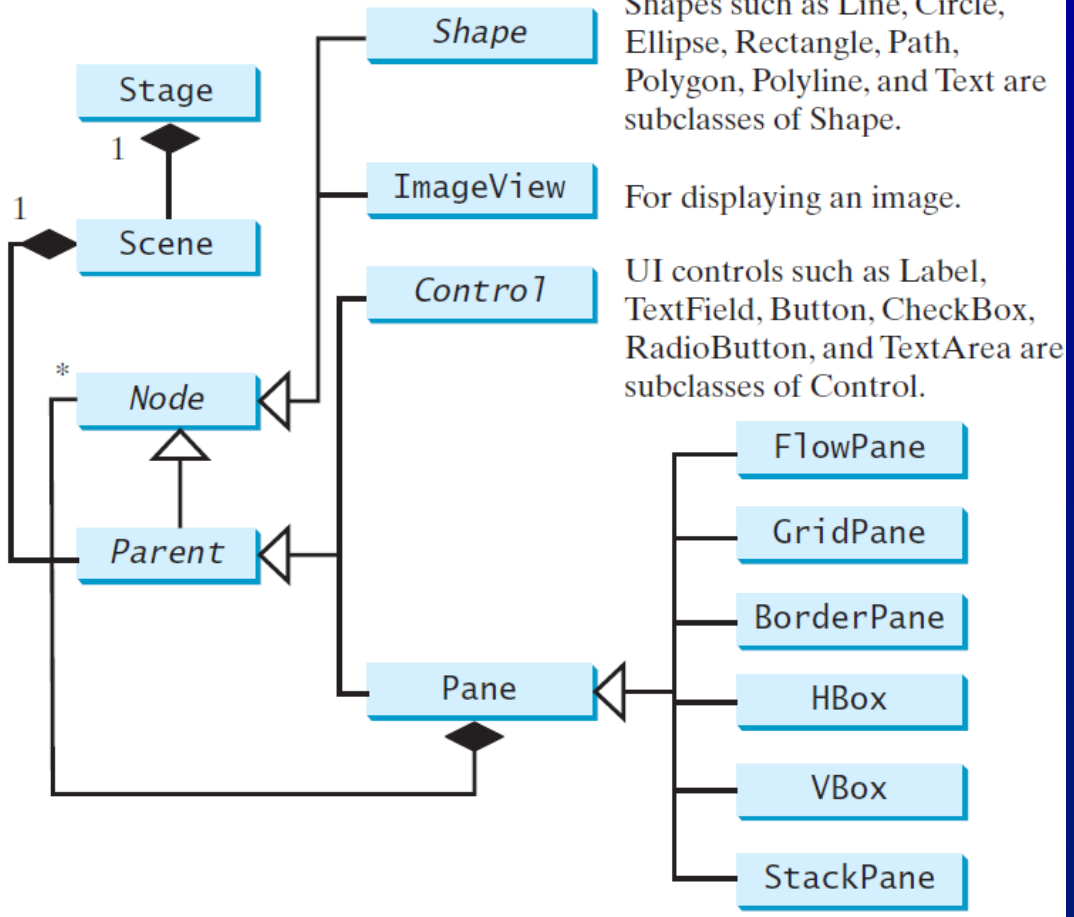
Panes, UI Controls, and Shapes

- *Panes, UI controls, and shapes are subtypes of* **Node**.
- A *node* is a visual component such as a shape, an image view, a UI control, or a pane.
- A *shape* refers to a text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.
- A *UI control* refers to a label, button, check box, radio button, text field, text area, etc.
- A scene can be displayed in a stage

Panes, UI Controls, and Shapes



(a)



(b)

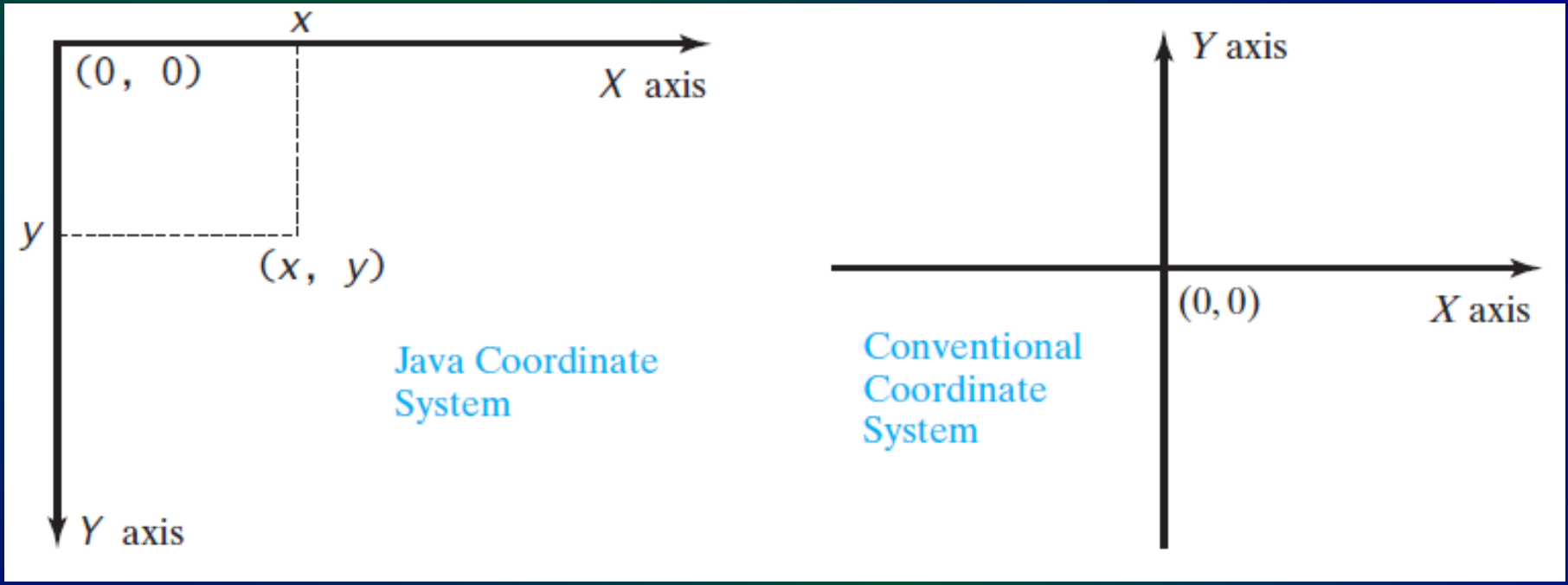
Panes, UI Controls, and Shapes

```
1
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Button;
5  import javafx.scene.layout.StackPane;
6  import javafx.stage.Stage;
7
8  public class HelloWorld extends Application {
9    @Override // Override the start method in the Application class
10   public void start(Stage primaryStage) {
11     // Create a scene and place a button in the scene
12     StackPane pane = new StackPane();
13     pane.getChildren().add(new Button("Hello World"));
14     Scene scene = new Scene(pane, 200, 50);
15     primaryStage.setTitle("Button in a pane"); // Set the stage title
16     primaryStage.setScene(scene); // Place the scene in the stage
17     primaryStage.show(); // Display the stage
18   }
19
20   /**
21    * The main method is only needed for the IDE with limited
22    * JavaFX support. Not needed for running from the command line.
23    */
24   public static void main(String[] args) {
25     launch(args);
26   }
27 }
```

Display a Shape

Java Coordinate System

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



Display a Shape

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

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

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

  /**
   * 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);
  }
}
```

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*.

Binding Properties

```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
  }

  /**
   * 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);
  }
}
```

Binding Property:
getter, setter, and property getter

- By convention, each binding property (e.g., centerX) in a JavaFX class (e.g., Circle) has a getter (e.g., getCenterX()) and setter (e.g., setCenterX(double)) method for returning and setting the property's value. It also has a getter method for returning the property itself.
- The naming convention for this method is the property name followed by the word Property.
- For example, the property getter method for centerX is centerXProperty(). We call the getCenterX() method as the value getter method, the setCenterX(double) method as the value setter method, and centerXProperty() as the property getter method.
- Note that getCenterX() returns a double value and centerXProperty() returns an object of the DoubleProperty type.

Binding Property:
getter, setter, and property getter

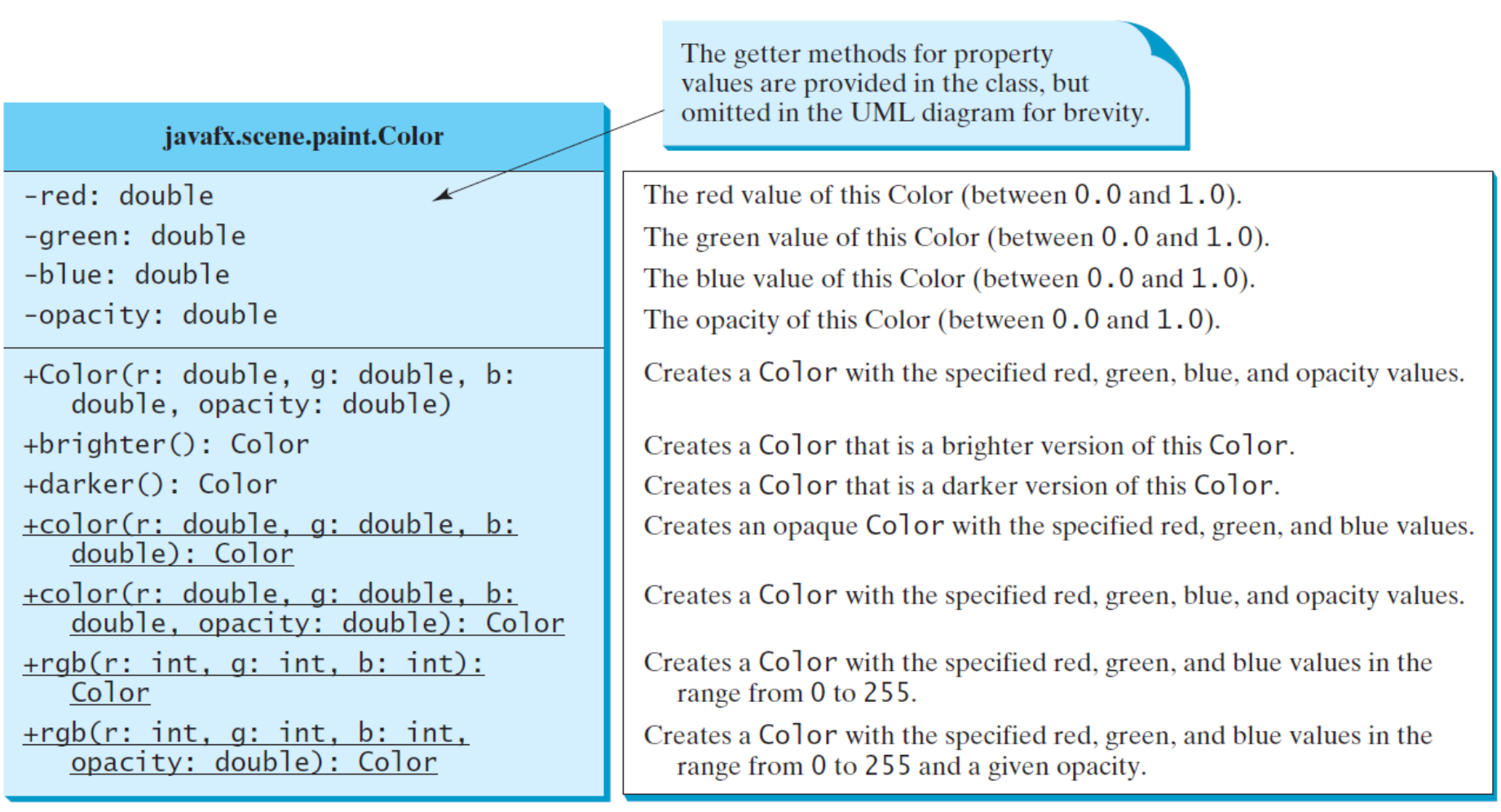
```
public class SomeClassName {

  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

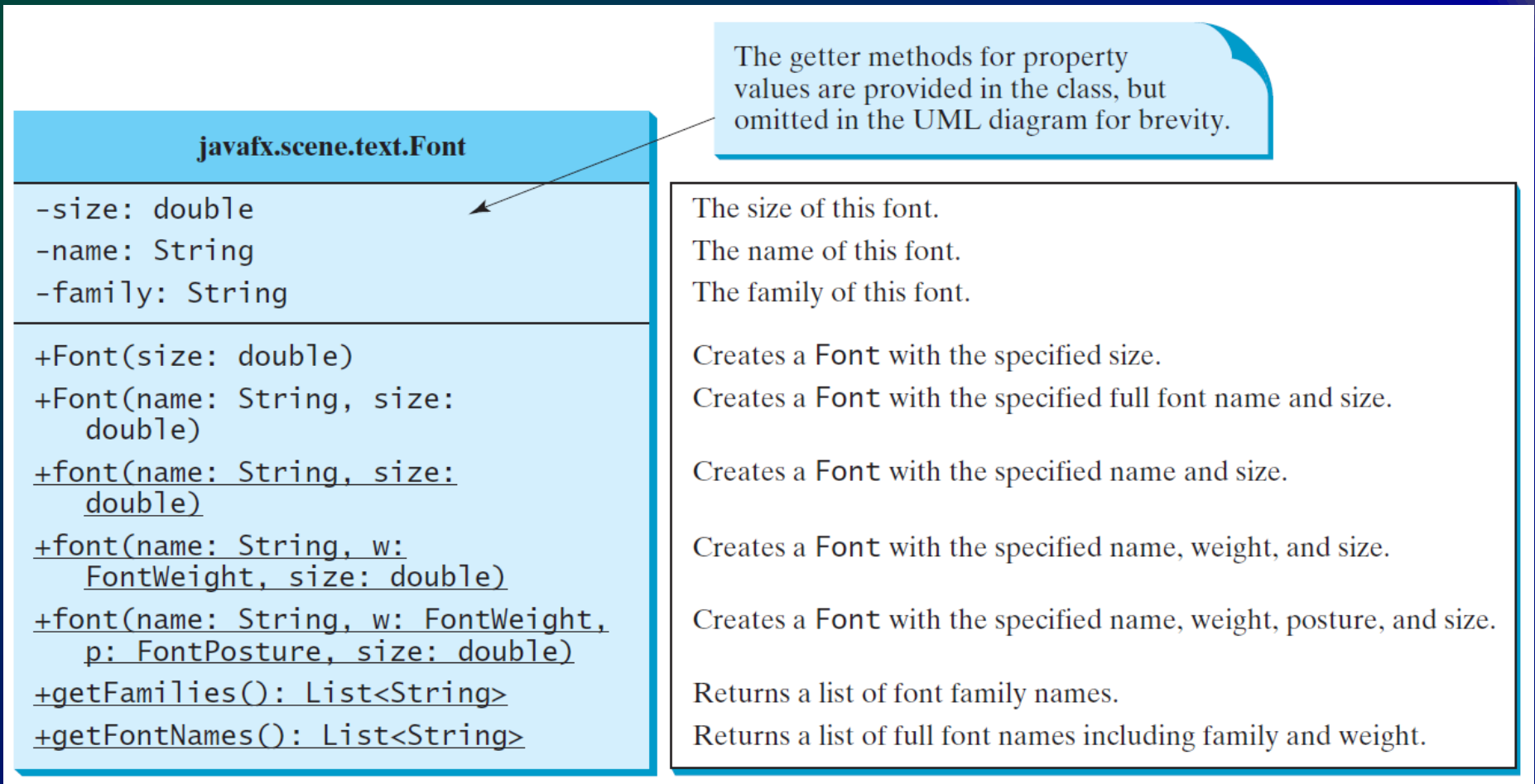
```
public class Circle {

  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

The Color Class



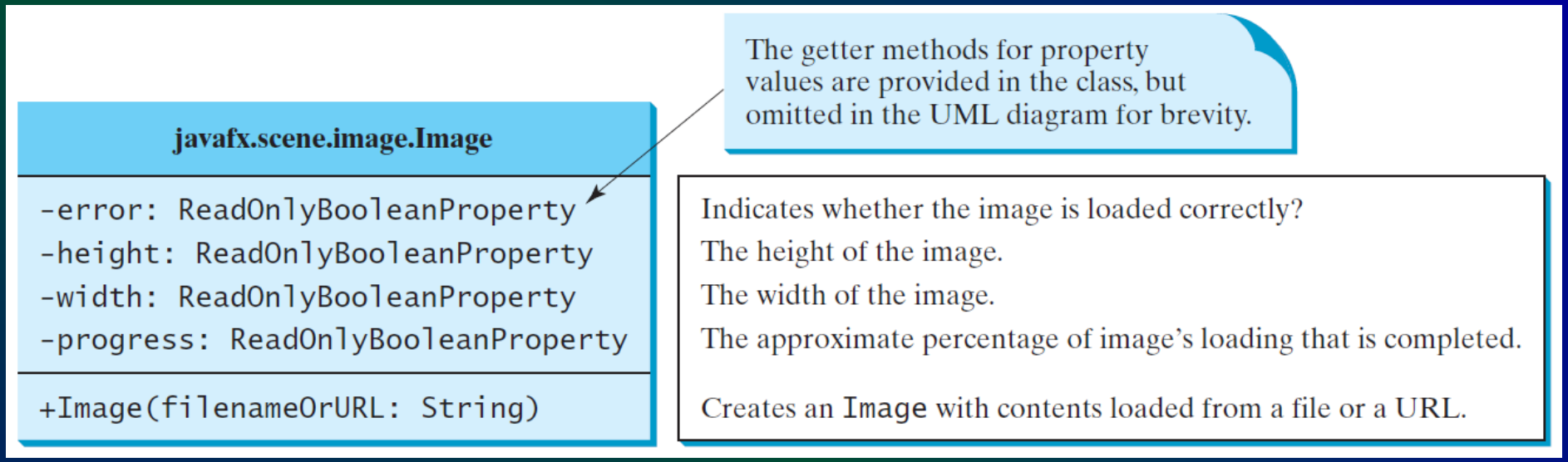
The Font Class



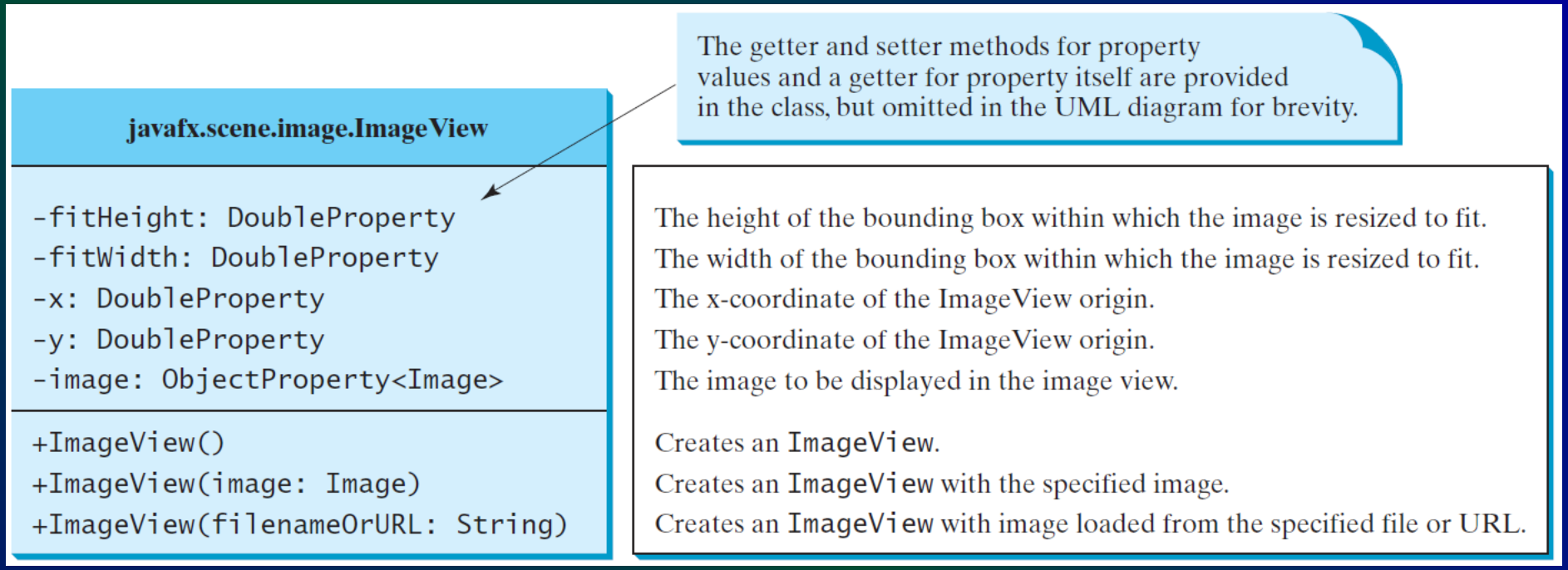
The Font Class
Demo

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.*;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.scene.text.*;
7  import javafx.scene.control.*;
8  import javafx.stage.Stage;
9
10 public class FontDemo extends Application {
11   @Override // Override the start method in the Application class
12   public void start(Stage primaryStage) {
13     // Create a pane to hold the circle
14     Pane pane = new StackPane();
15
16     // Create a circle and set its properties
17     Circle circle = new Circle();
18     circle.setRadius(50);
19     circle.setStroke(Color.BLACK);
20     circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));
21     pane.getChildren().add(circle); // Add circle to the pane
22
23     // Create a label and set its properties
24     Label label = new Label("JavaFX");
25     label.setFont(Font.font("Times New Roman",
26       FontWeight.BOLD, FontPosture.ITALIC, 20));
27     pane.getChildren().add(label);
28
29     // Create a scene and place it in the stage
30     Scene scene = new Scene(pane);
31     primaryStage.setTitle("FontDemo"); // Set the stage title
32     primaryStage.setScene(scene); // Place the scene in the stage
33     primaryStage.show(); // Display the stage
34   }
35
36   /**
37    * The main method is only needed for the IDE with limited
38    * JavaFX support. Not needed for running from the command line.
39    */
40   public static void main(String[] args) {
41     launch(args);
42   }
43 }
```

The Image Class



The ImageView Class



Image/ImageView Example

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

public class ShowImage extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the image views
    Pane pane = new HBox(10);
    pane.setPadding(new Insets(5, 5, 5, 5));
    Image image = new Image("image/us.gif");
    pane.getChildren().add(new ImageView(image));

    ImageView imageView2 = new ImageView(image);
    imageView2.setFitHeight(100);
    imageView2.setFitWidth(100);
    pane.getChildren().add(imageView2);

    ImageView imageView3 = new ImageView(image);
    imageView3.setRotate(90);
    pane.getChildren().add(imageView3);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowImage"); // 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);
  }
}
```

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>
-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.

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>
-gridLinesVisible:
   BooleanProperty
-hgap: DoubleProperty
-vgap: DoubleProperty
```

```
+GridPane()
+add(child: Node, columnIndex:
   int, rowIndex: int): void
+addColumn(columnIndex: int,
   children: Node...): void
+addRow(rowIndex: int,
   children: Node...): void
+getColumnIndex(child: Node):
   int
+setColumnIndex(child: Node,
   columnIndex: int): void
+getRowIndex(child:Node): int
+setRowIndex(child: Node,
   rowIndex: int): void
+setHalighnment(child: Node,
   value: HPos): void
+setValighnment(child: Node,
   value: VPos): void
```

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

Is the grid line visible? (default: `false`)

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

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

Creates a `GridPane`.

Adds a node to the specified column and row.

Adds multiple nodes to the specified column.

Adds multiple nodes to the specified row.

Returns the column index for the specified node.

Sets a node to a new column. This method repositions the node.

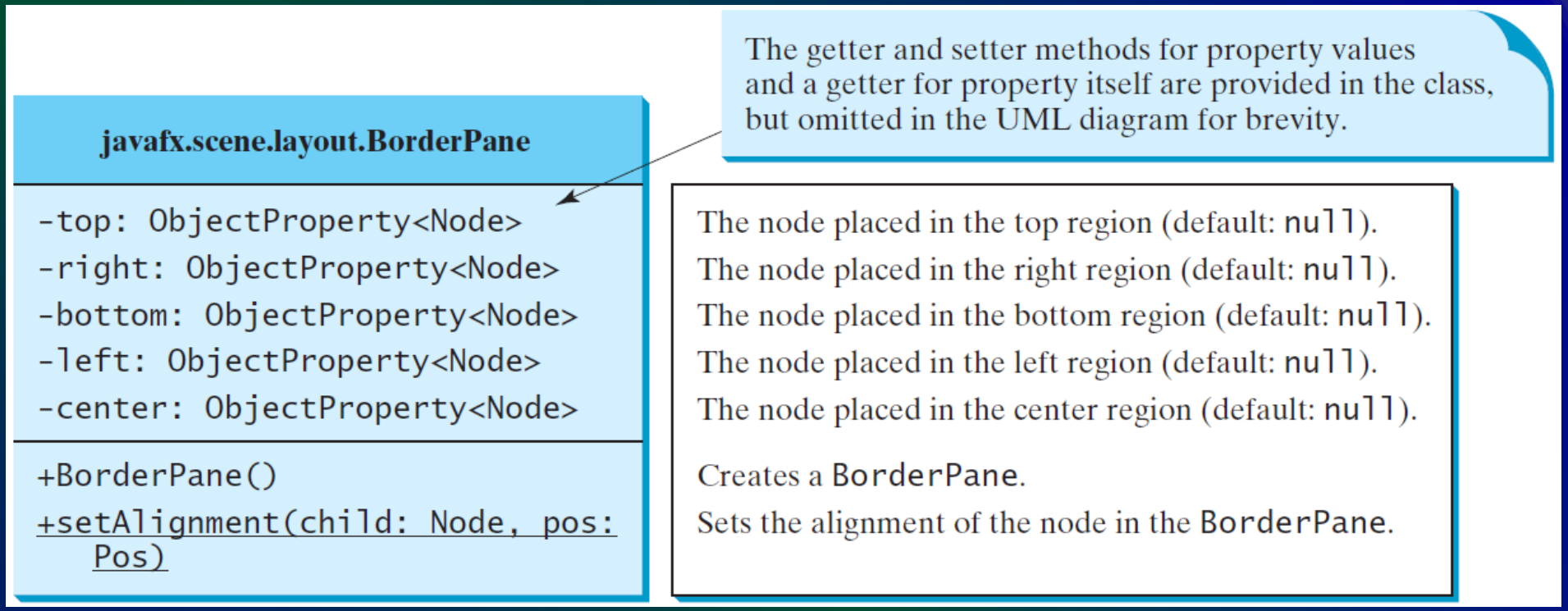
Returns the row index for the specified node.

Sets a node to a new row. This method repositions the node.

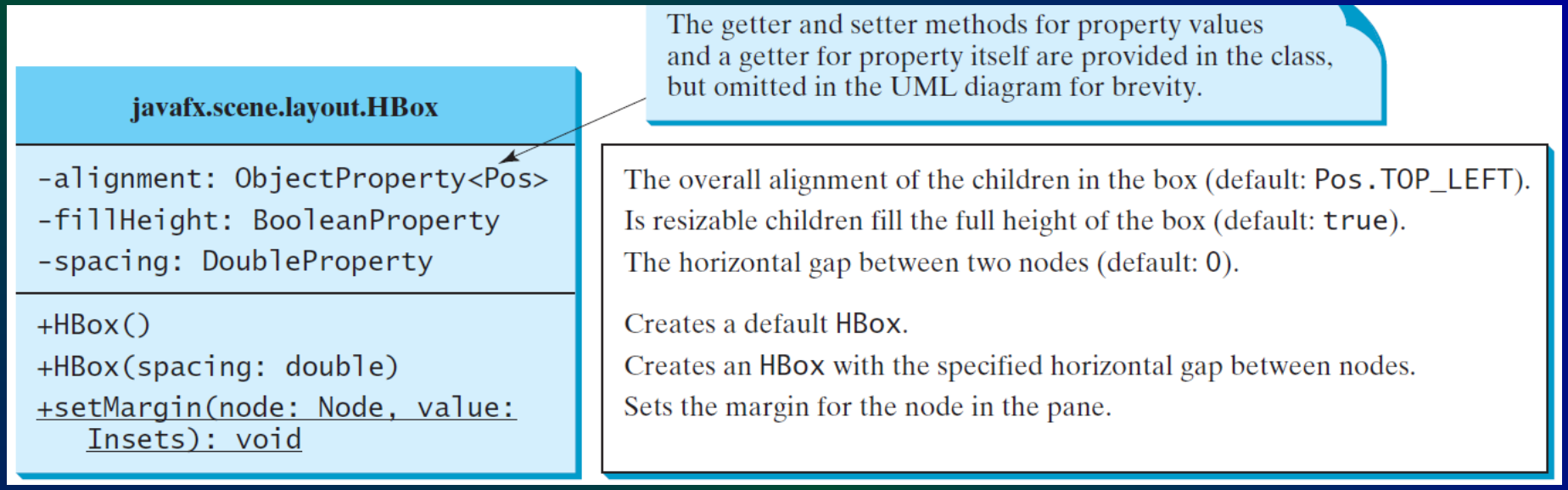
Sets the horizontal alignment for the child in the cell.

Sets the vertical alignment for the child in the cell.

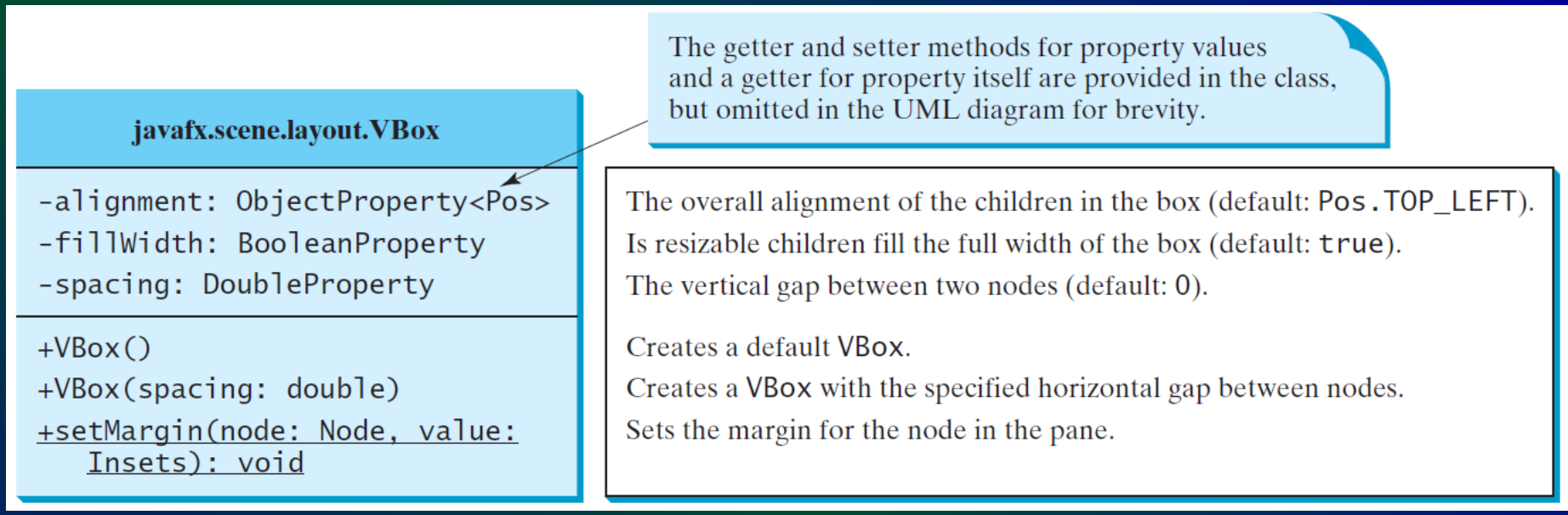
BorderPane


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 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.
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.

HBox

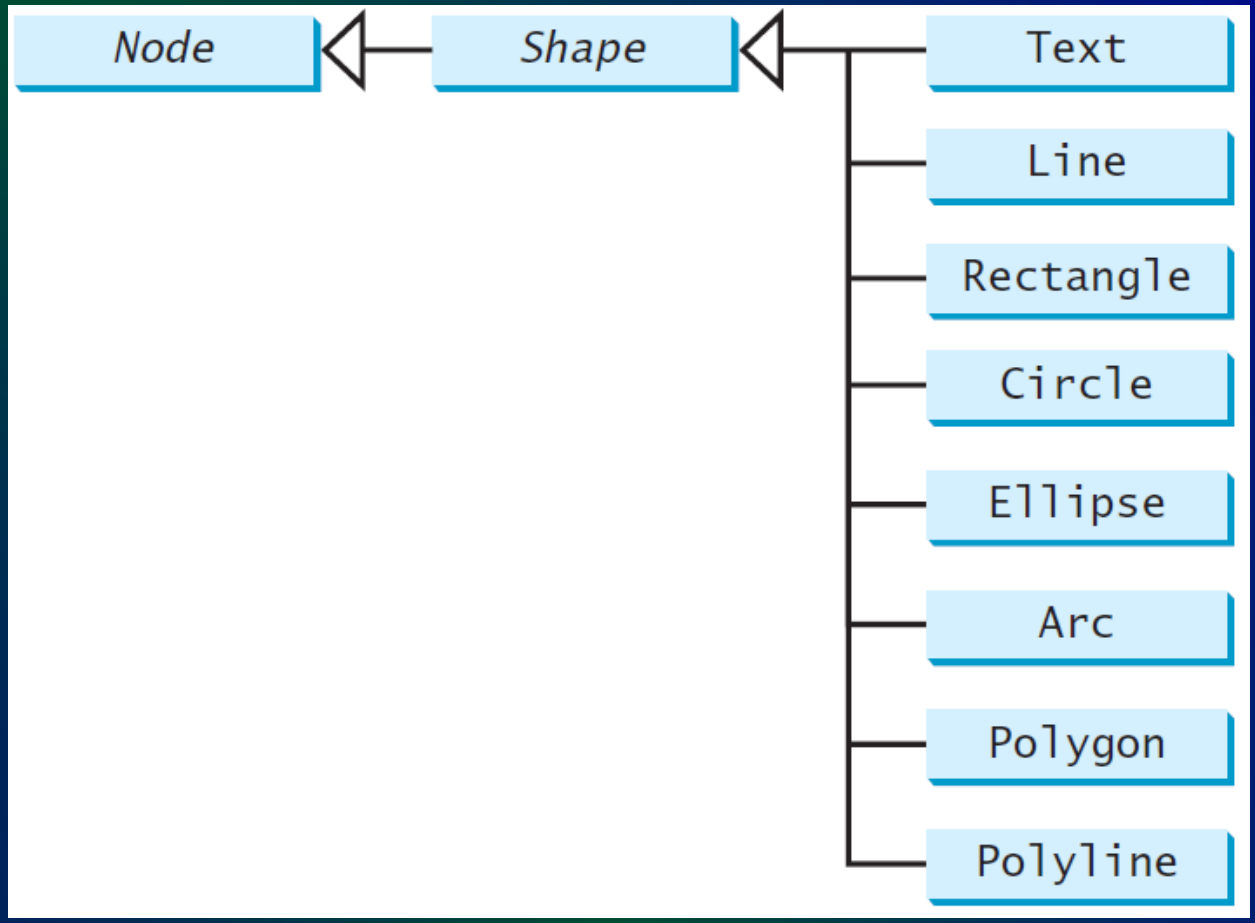

javafx.scene.layout.HBox
-alignment: ObjectProperty<Pos>
-fillHeight: BooleanProperty
-spacing: DoubleProperty
+HBox()
+HBox(spacing: double)
+setMargin(node: Node, value: Insets): void
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
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.

VBox


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

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
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
```

```
+Text()
+Text(text: String)
+Text(x: double, y: double,
   text: String)
```

Defines the text to be displayed.

Defines the x-coordinate of text (default 0).

Defines the y-coordinate of text (default 0).

Defines if each line has an underline below it (default `false`).

Defines if each line has a line through it (default `false`).

Defines the font for the text.

Creates an empty Text.

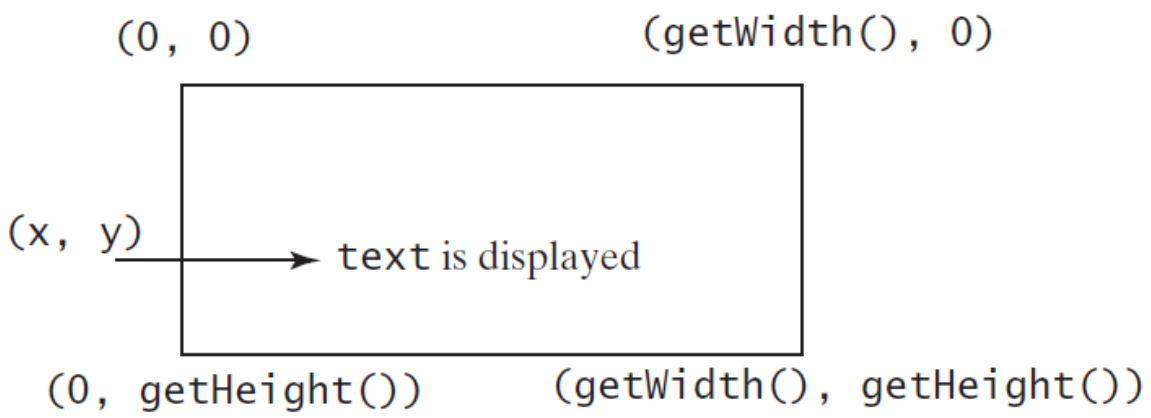
Creates a Text with the specified text.

Creates a Text with the specified x-, y-coordinates and text.

Text Example

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.FontPosture;

public class ShowText extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane to hold the texts
    Pane pane = new Pane();
    pane.setPadding(new Insets(5, 5, 5, 5));
    Text text1 = new Text(20, 20, "Programming is fun");
    text1.setFont(Font.font("Courier", FontWeight.BOLD,
      FontPosture.ITALIC, 15));
    pane.getChildren().add(text1);

    Text text2 = new Text(60, 60, "Programming is fun\nDisplay text");
    pane.getChildren().add(text2);

    Text text3 = new Text(10, 100, "Programming is fun\nDisplay text");
    text3.setFill(Color.RED);
    text3.setUnderline(true);
    text3.setStrikethrough(true);
    pane.getChildren().add(text3);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane);
    primaryStage.setTitle("ShowText"); // 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);
  }
}
```

Text Example



(a) Text(x, y, text)

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

Line

javafx.scene.shape.Line

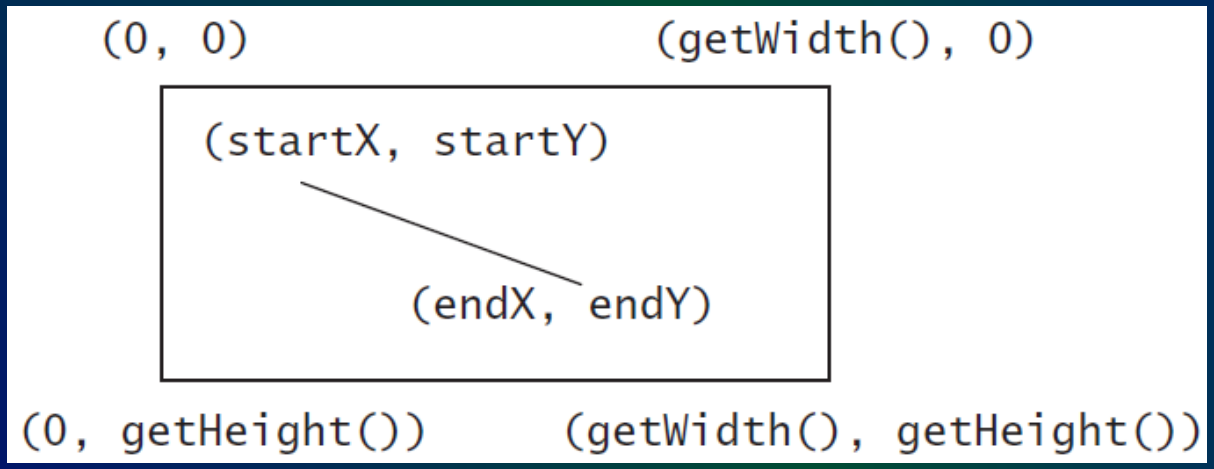
-startX: DoubleProperty
-startY: DoubleProperty
-endX: DoubleProperty
-endY: DoubleProperty

+Line()
+Line(startX: double, startY: double, endX: double, endY: double)

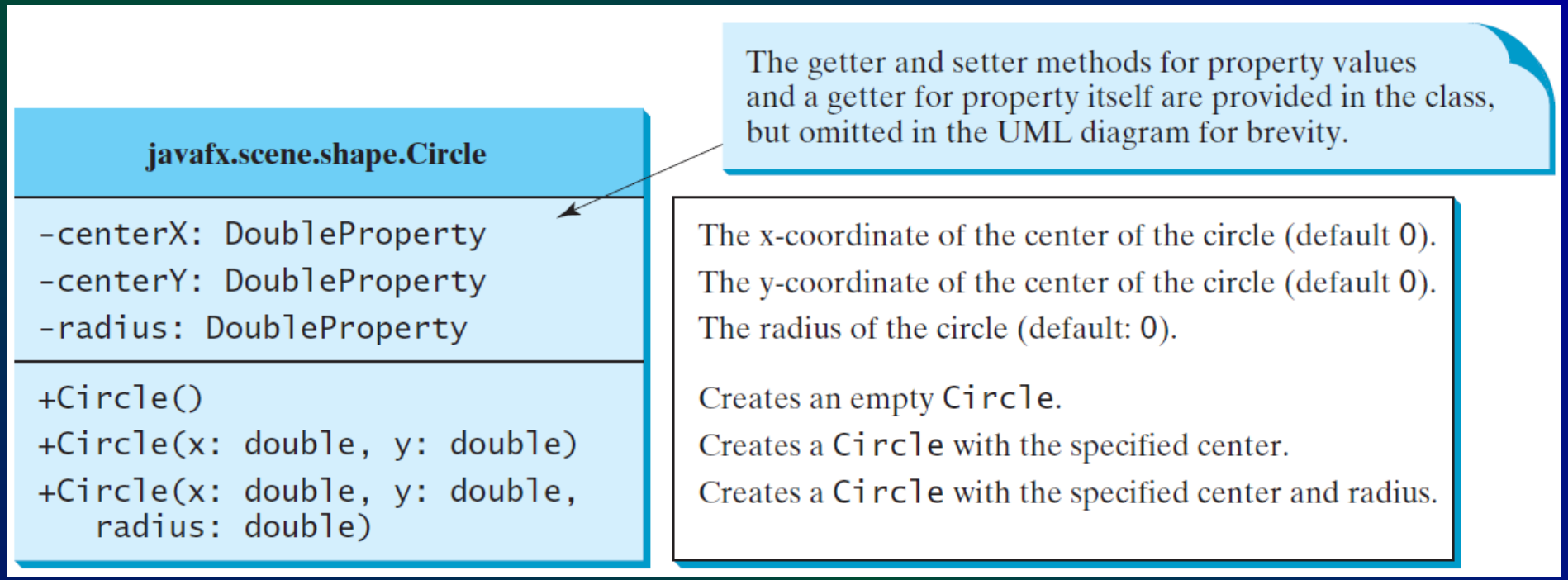
The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The x-coordinate of the start point.
The y-coordinate of the start point.
The x-coordinate of the end point.
The y-coordinate of the end point.

Creates an empty `Line`.
Creates a `Line` with the specified starting and ending points.



Circle


javafx.scene.shape.Circle
-centerX: DoubleProperty
-centerY: DoubleProperty
-radius: DoubleProperty
+Circle()
+Circle(x: double, y: double)
+Circle(x: double, y: double, radius: 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.
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.

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
-y:DoubleProperty
-width: DoubleProperty
-height: DoubleProperty
-arcWidth: DoubleProperty

-arcHeight: DoubleProperty
```

```
+Rectangle()
+Rectanlge(x: double, y:
   double, width: double,
   height: double)
```

The x-coordinate of the upper-left corner of the rectangle (default 0).

The y-coordinate of the upper-left corner of the rectangle (default 0).

The width of the rectangle (default: 0).

The height of the rectangle (default: 0).

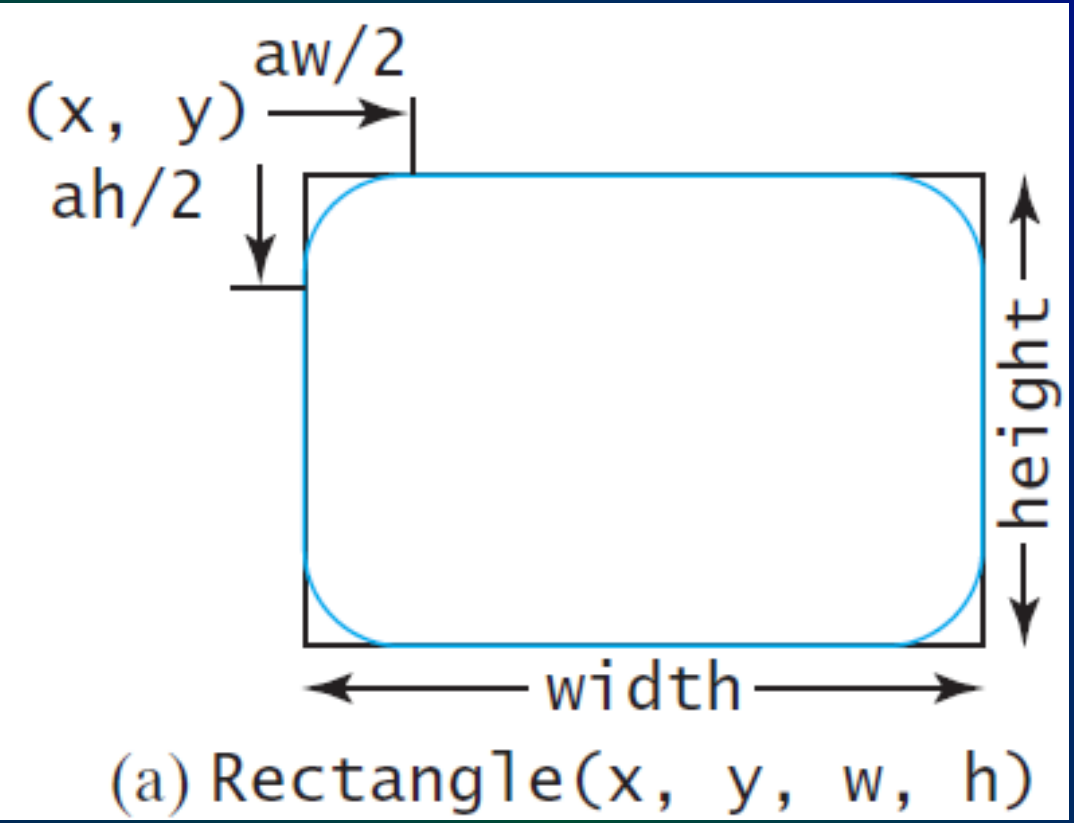
The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

The `arcHeight` of the rectangle (default: 0). `arcHeight` is the vertical diameter of the arcs at the corner (see Figure 14.31a).

Creates an empty `Rectangle`.

Creates a `Rectangle` with the specified upper-left corner point, width, and height.

Rectangle Example



Rectangle Example

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

public class ShowRectangle extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane
    Pane pane = new Pane();

    // Create rectangles and add to pane
    Rectangle r1 = new Rectangle(25, 10, 60, 30);
    r1.setStroke(Color.BLACK);
    r1.setFill(Color.WHITE);
    pane.getChildren().add(new Text(10, 27, "r1"));
    pane.getChildren().add(r1);

    Rectangle r2 = new Rectangle(25, 50, 60, 30);
    pane.getChildren().add(new Text(10, 67, "r2"));
    pane.getChildren().add(r2);

    Rectangle r3 = new Rectangle(25, 90, 60, 30);
    r3.setArcWidth(25);
    r3.setArcHeight(25);
    pane.getChildren().add(new Text(10, 107, "r3"));
    pane.getChildren().add(r3);

    for (int i = 0; i < 4; i++) {
      Rectangle r = new Rectangle(100, 50, 100, 30);
      r.setRotate(i * 360 / 8);
      r.setStroke(Color.color(Math.random(), Math.random(),
          Math.random()));
      r.setFill(Color.WHITE);
      pane.getChildren().add(r);
    }

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 250, 150);
    primaryStage.setTitle("ShowRectangle"); // 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);
  }
}
```

Ellipse

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.

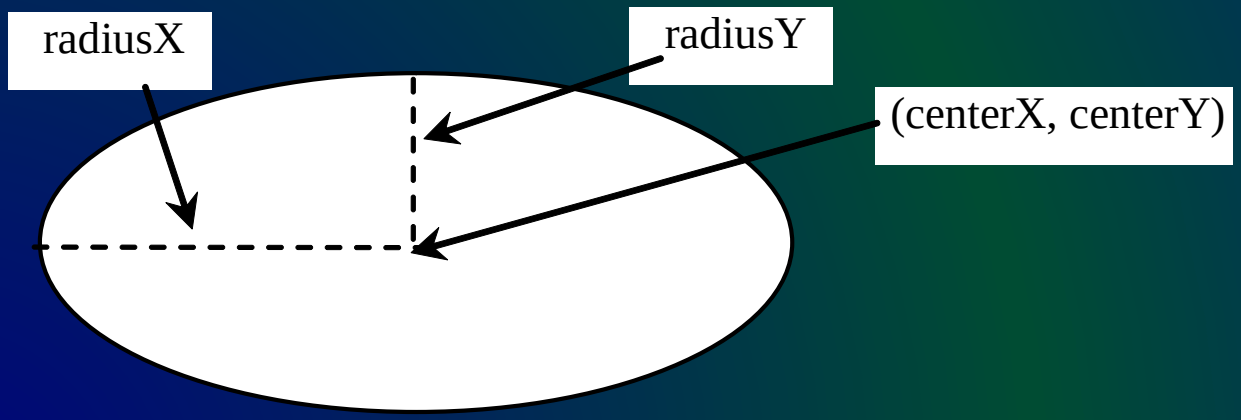
javafx.scene.shape.Ellipse

```
-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).

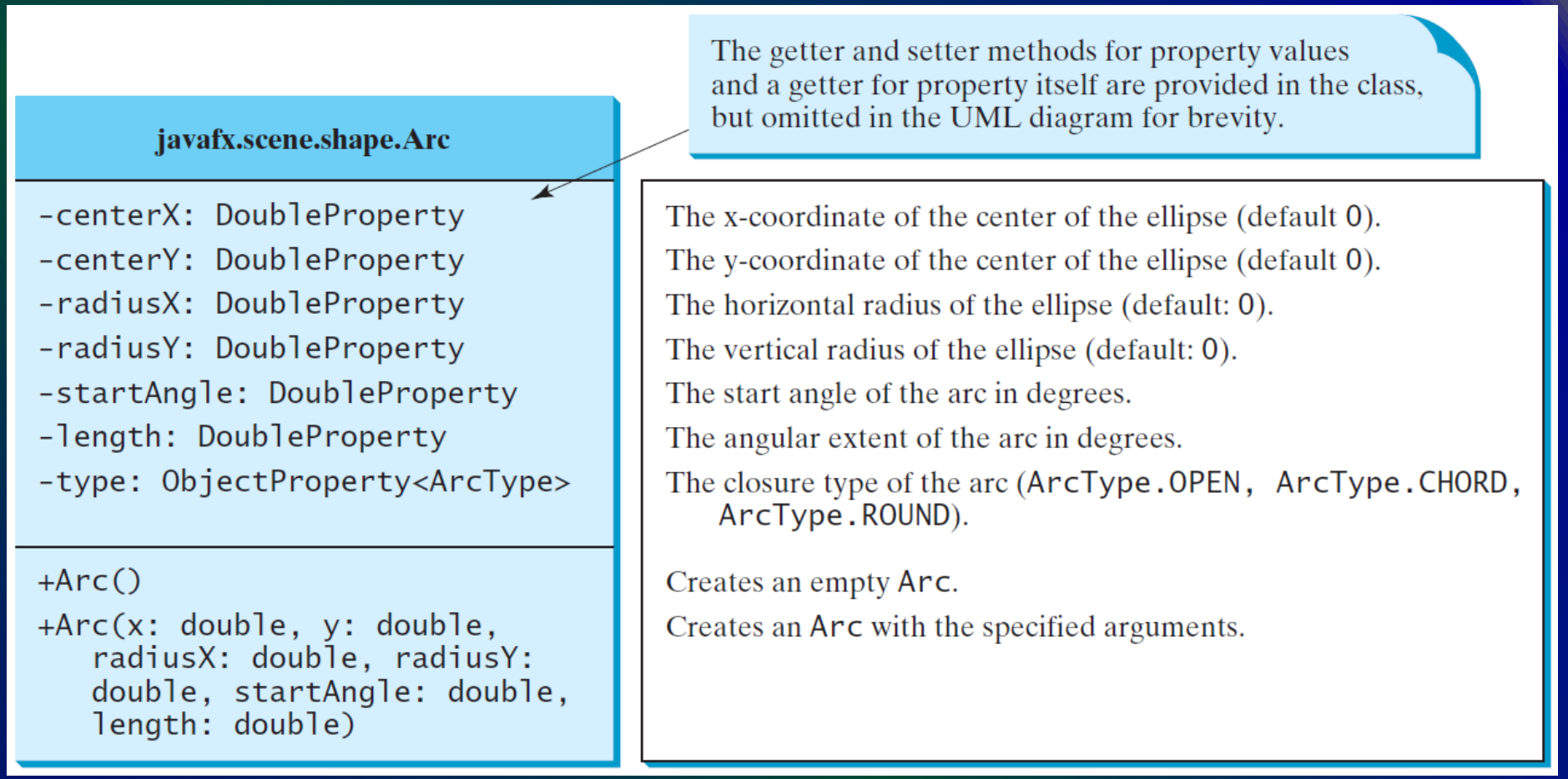
Creates an empty `Ellipse`.
Creates an `Ellipse` with the specified center.
Creates an `Ellipse` with the specified center and radiuses.



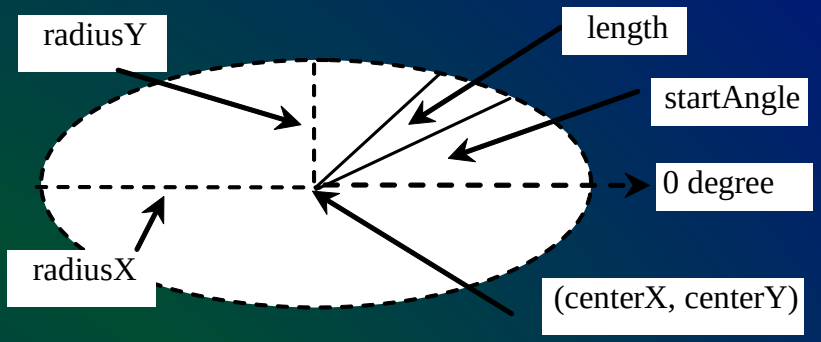
Ellipse Example

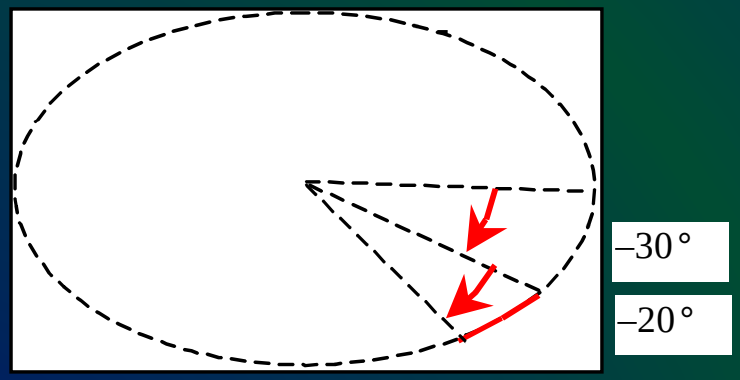
```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Ellipse;

public class ShowEllipse extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane
    Pane pane = new Pane();

    double xRadius = 100;
    double yRadius = 50;

    for (int i = 0; i < 16; i++) {
      // Create an ellipse and add it to pane
      Ellipse e1 = new Ellipse(150, 100, xRadius, yRadius);
      e1.setStroke(Color.color(Math.random(), Math.random(),
        Math.random()));
      e1.setFill(Color.WHITE);
      e1.setRotate(i * 180 / 16);
      pane.getChildren().add(e1);
    }

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 300, 200);
    primaryStage.setTitle("ShowEllipse"); // 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);
  }
}
```

Arc

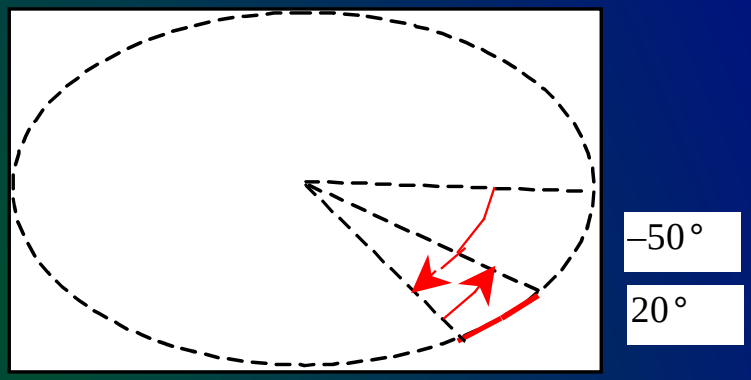

javafx.scene.shape.Arc
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
+Arc()
+Arc(x: double, y: double, radiusX: double, radiusY: double, startAngle: double, length: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

Arc Examples





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

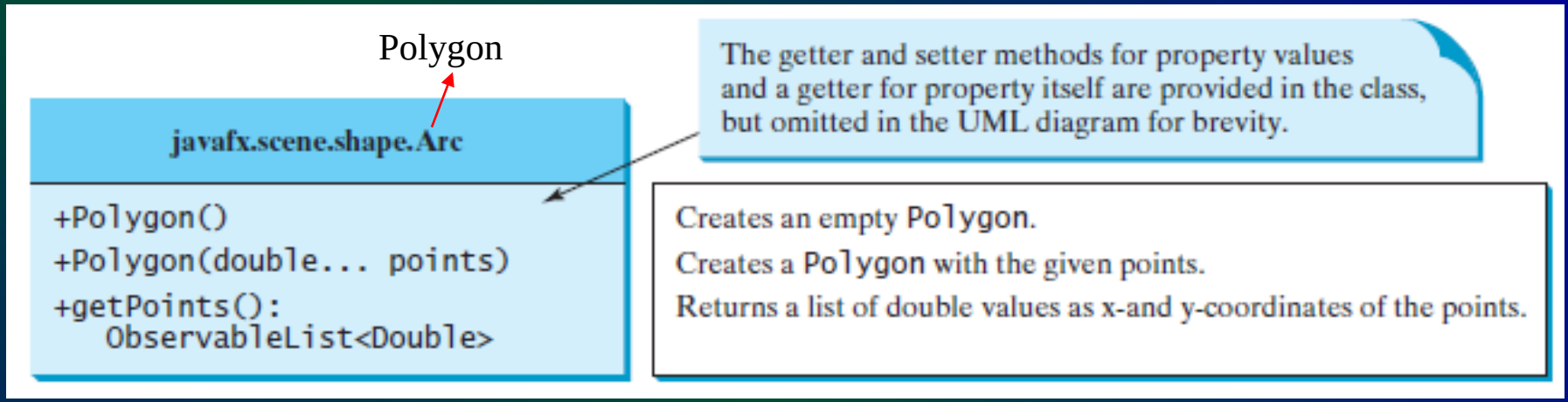


(b) Negative starting angle –50° and positive spanning angle 20°

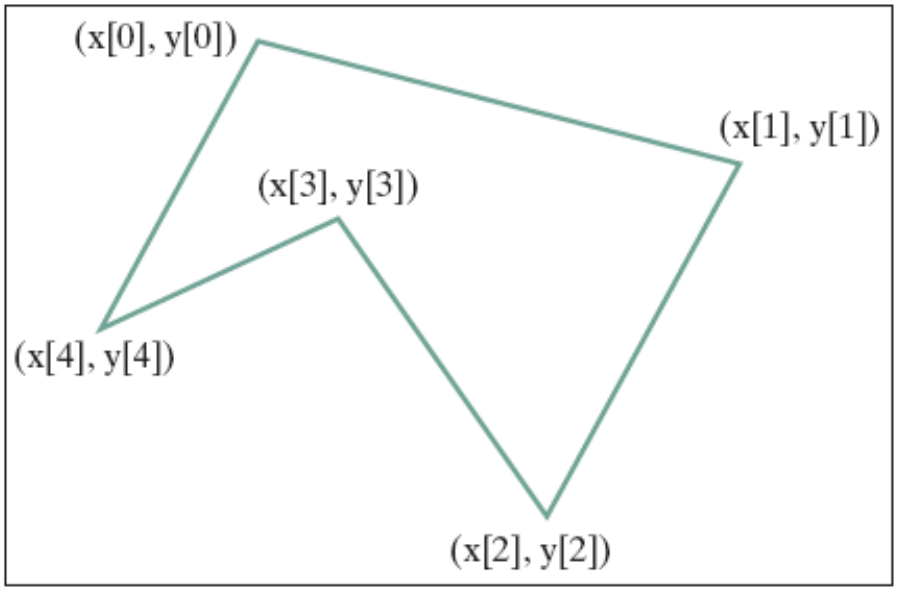
Arc Example

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.shape.Arc;
import javafx.scene.shape.ArcType;
import javafx.scene.text.Text;

public class ShowArc extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a pane
    Pane pane = new Pane();

    Arc arc1 = new Arc(150, 100, 80, 80, 30, 35); // Create an arc
    arc1.setFill(Color.RED); // Set fill color
    arc1.setType(ArcType.ROUND); // Set arc type
    pane.getChildren().add(new Text(210, 40, "arc1: round"));
    pane.getChildren().add(arc1); // Add arc to pane

    Arc arc2 = new Arc(150, 100, 80, 80, 30 + 90, 35);
    arc2.setFill(Color.WHITE);
    arc2.setType(ArcType.OPEN);
    arc2.setStroke(Color.BLACK);
    pane.getChildren().add(new Text(20, 40, "arc2: open"));
    pane.getChildren().add(arc2);

    Arc arc3 = new Arc(150, 100, 80, 80, 30 + 180, 35);
    arc3.setFill(Color.WHITE);
    arc3.setType(ArcType.CHORD);
    arc3.setStroke(Color.BLACK);
    pane.getChildren().add(new Text(20, 170, "arc3: chord"));
    pane.getChildren().add(arc3);

    Arc arc4 = new Arc(150, 100, 80, 80, 30 + 270, 35);
    arc4.setFill(Color.GREEN);
    arc4.setType(ArcType.CHORD);
    arc4.setStroke(Color.BLACK);
    pane.getChildren().add(new Text(210, 170, "arc4: chord"));
    pane.getChildren().add(arc4);

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 300, 200);
    primaryStage.setTitle("ShowArc"); // 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);
  }
}
```

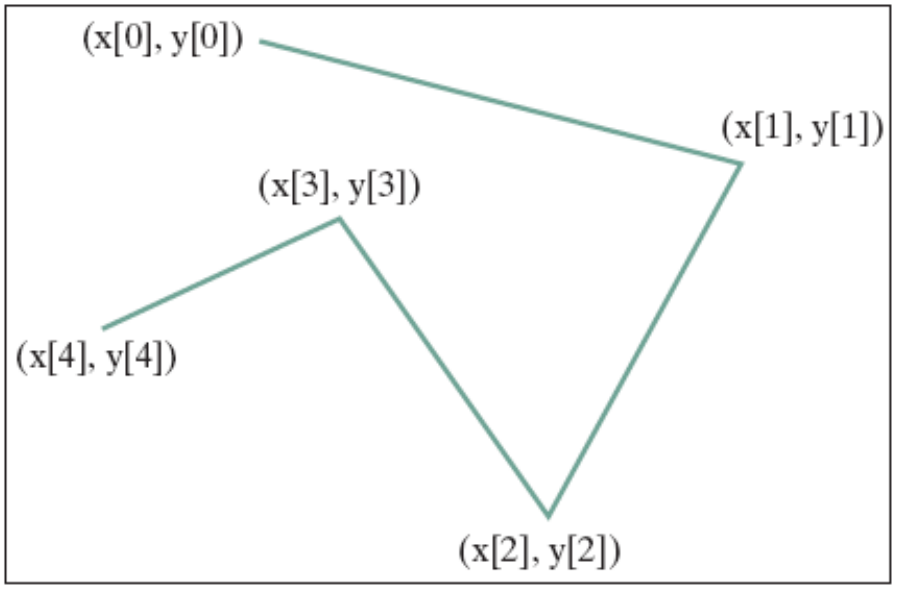
Polygon


Polygon
javafx.scene.shape.Arc
+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.

Polygon and Polyline



(a) Polygon



(b) Polyline

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



public class ShowPolygon extends Application {
  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) throws InterruptedException {
    // Create a pane, a polygon, and place polygon to pane
    Pane pane = new Pane();
    Polygon polygon = new Polygon();
    pane.getChildren().add(polygon);
    polygon.setFill(Color.WHITE);
    polygon.setStroke(Color.BLACK);
    ObservableList<Double> list = polygon.getPoints();

    final double WIDTH = 200, HEIGHT = 200;
    double centerX = WIDTH / 2, centerY = HEIGHT / 2;
    double radius = Math.min(WIDTH, HEIGHT) * 0.4;

    // 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));
    }

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, WIDTH, HEIGHT);
    primaryStage.setTitle("ShowPolygon"); // 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);
  }
}
```

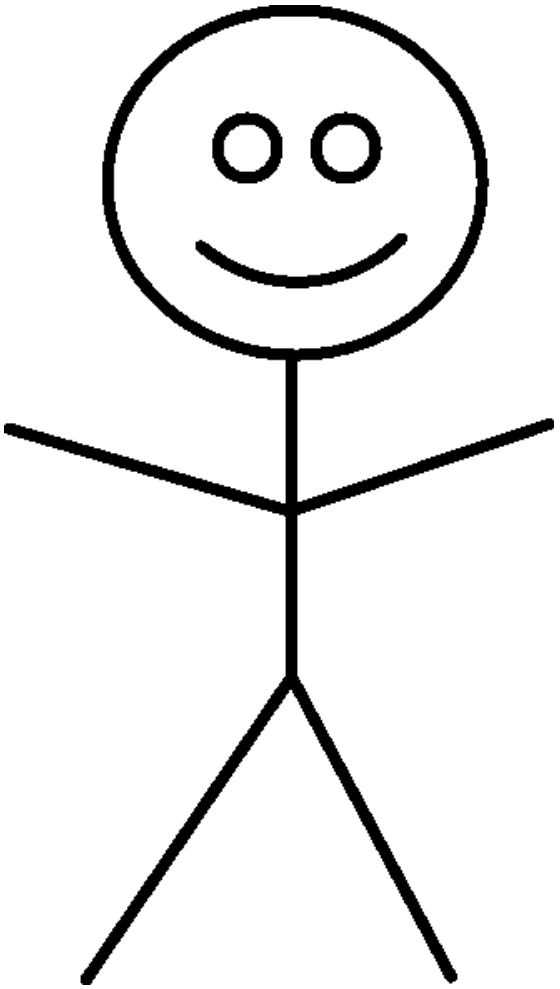
Show Polygon

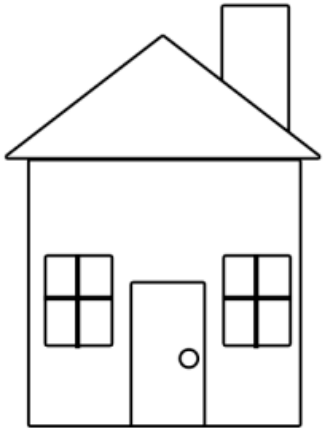
Add vs. addAll

```
Polygon polygon = new Polygon();
polygon.getPoints().add(0.0);
polygon.getPoints().add(0.0);
polygon.getPoints().add(20.0);
polygon.getPoints().add(10.0);
polygon.getPoints().add(10.0);
polygon.getPoints().add(20.0);

Polygon polygon = new Polygon();
polygon.getPoints().addAll(new Double[]{
    0.0, 0.0,
    20.0, 10.0,
    10.0, 20.0 });
```

Programming Challenge

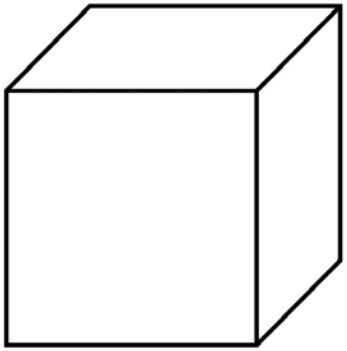
- Draw the following simple figures using JavaFX shapes.

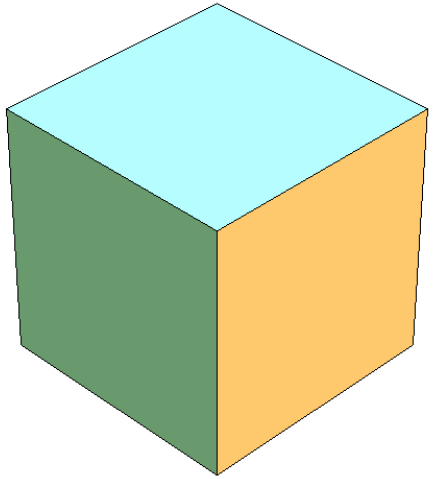


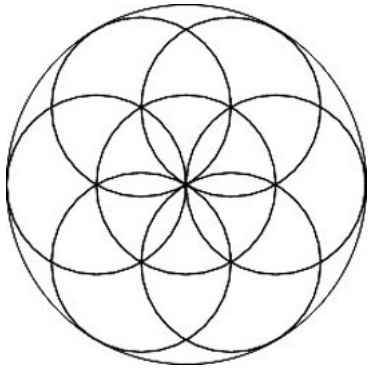


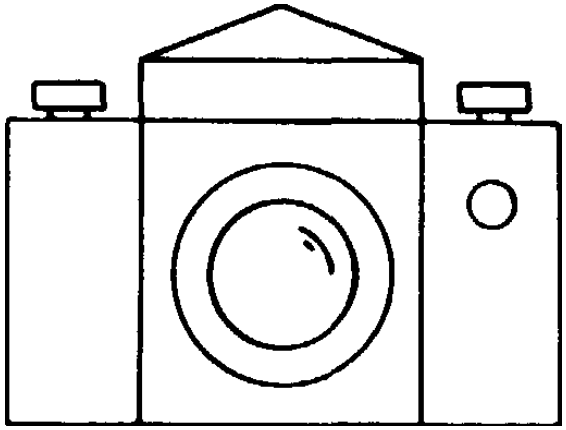
Programming Challenge

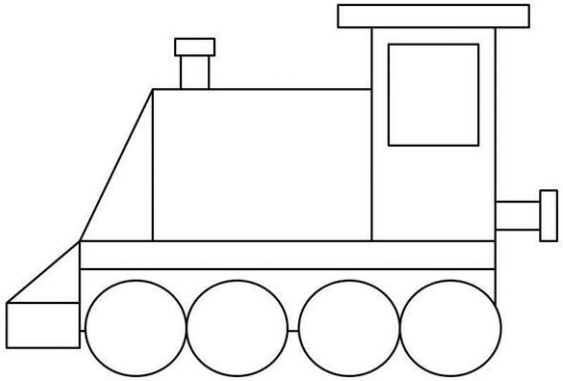
- Draw the following not so simple figures using JavaFX shapes.

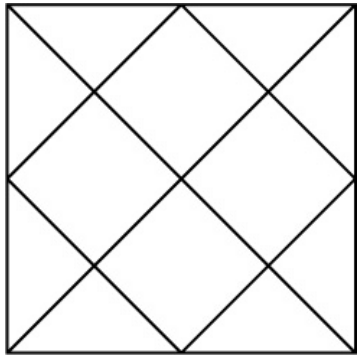












Programming Challenge

- Draw the following scene using JavaFX shapes. Your drawing does not have to be exactly like this one, but capture as much as possible from this picture in your JavaFX scene.