Chapter 14 JavaFX Basics

Using JavaFX in Eclipse

Download an appropriate JavaFX runtime (https://gluonhq.com/products/javafx/)for your operating system and unzip it to a desired location. Put the contents of this zip file somewhere on your system, for example in "C:\Users\selim\Documents\javafx-sdk-11.0.2\".

METHOD I:

Install the JavaFx plug-in for eclipse.

- With Eclipse open, click Help > Install New Software…
- Click the Add… button.
- Enter a name, such as JavaFX.
- Enter the location as follows: http://download.eclipse.org/efxclipse/updates-released/1.2.0/site/
- Check both options "e(fx)clipse – install" and "e(fx)clipse – single components" as shown. Accept other defaults. Click the Next button.
- These are the items that will be installed. Click Next again.
- Accept terms and click Finish.
- You'll be prompted to reboot Eclipse. The plug-in is now installed and is ready to use.

File -> Properties -> Java Build Path -> Libraries tab -> Add external jars
select and include all files in the directory that has unzipped files (for example "C:\Users\selim\Documents\javafx-sdk-11.0.2\lib")

Now, to use JavaFx...

- Navigate to File > New > Project
- In the New Project wizard, select JavaFx > JavaFX Project
- Enter a name for the project, ie, JavaFxDemo, etc. Click Finish.
- Go to the project name > src > application. Click once on application to highlight. Create classes here (right-mouse click, New > Class)

Using JavaFX in Eclipse

METHOD II:

- File -> Properties -> Java Build Path -> Libraries tab -> Add external jars
- select and include all files in the directory that has unzipped files (for example "C:\Users\selim\Documents\javafx-sdk-11.0.2\lib")
- Windows -> Preferences -> Java -> Installed JREs -> Edit
- Put

--module-path "C:\Users\selim\Documents\javafx-sdk-11.0.2\lib" --add-modules javafx.controls,javafx.fxml,javafx.graphics,javafx.media

- in Default VM Arguments. Note that your JavaFX library location can be different depending on the location that you unzipped.

METHOD III:

From CMD Prompt:

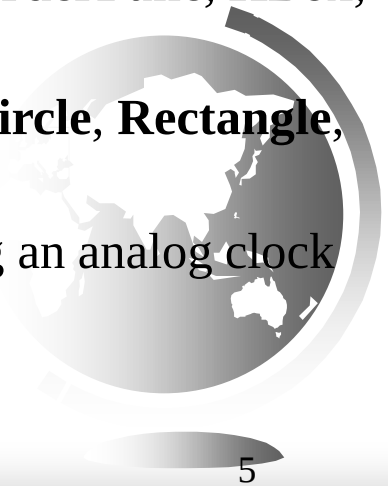
java --module-path "C:\Users\selim\Documents\javafx-sdk-11.0.2\lib" --add-modules javafx.controls BounceBallControl

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

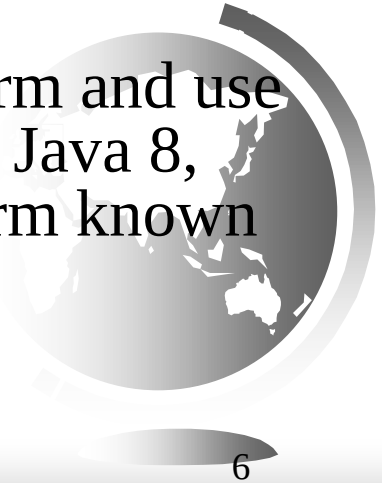
- To distinguish between JavaFX, Swing, and AWT (§14.2).
- To write a simple JavaFX program and understand the relationship among stages, scenes, and nodes (§14.3).
- To create user interfaces using panes, UI controls, and shapes (§14.4).
- To use binding properties to synchronize property values (§14.5).
- 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

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

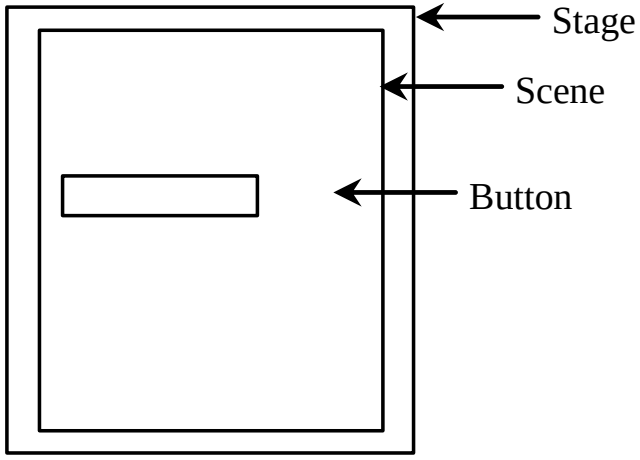
- When Java was introduced, the GUI classes were bundled in a library known as the *Abstract Windows Toolkit (AWT)*.
- AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects. In addition, AWT is prone to platform-specific bugs.
- The AWT user-interface 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*.

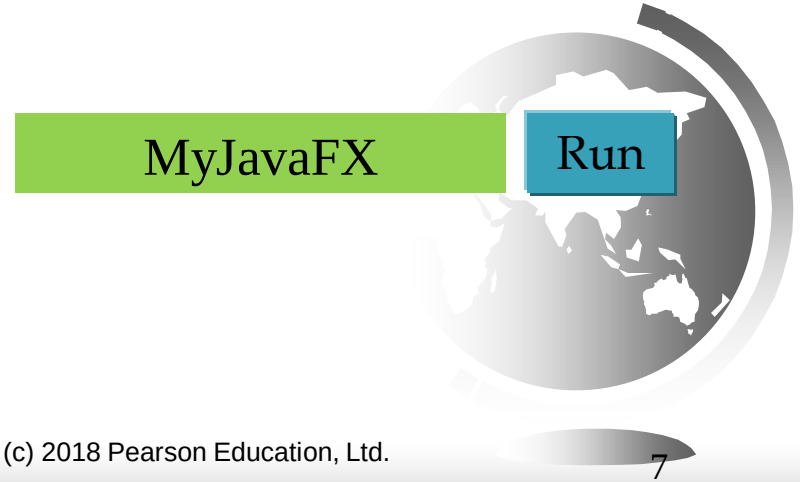


Basic Structure of JavaFX

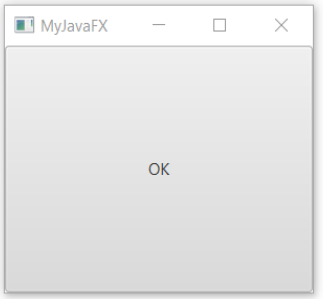
- Application
- Override the start(Stage) method
- Stage, Scene, and Nodes

▪A JavaFX GUI Program extends from javafx.application.Application (just like a Java Swing GUI program extends from javax.swing.JFrame).
▪JavaFX names the **Stage** and **Scene** classes using the analogy from the theater.
▪You may think of stage as the platform to support scenes, and nodes as actors to perform in the scenes.





Basic Structure of JavaFX



- The **launch** method (line 22) is a static method defined in the **Application** class for launching a stand-alone JavaFX application.
- The **main** method (lines 21–23) is not needed if you run the program from the command line. It may be needed to launch a JavaFX program from an IDE with a limited JavaFX support. When you run a JavaFX application without a main method, JVM automatically invokes the **launch** method to run the application.
- The main class overrides the **start** method defined in **javafx.application.Application** (line 8). After a JavaFX application is launched, the JVM constructs an instance of the class using its **no-arg** constructor and invokes its **start** method.
- The **start** method normally places UI controls in a scene and displays the scene in a stage.
- Line 10 creates a **Button** object and places it in a **Scene** object (line 11). A **Scene** object can be created using the constructor **Scene(node, width, height)**. This constructor specifies the width and height of the scene and places the node in the scene.

```
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 button and place it 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     launch(args);
23   }
24 }
```

Basic Structure of JavaFX

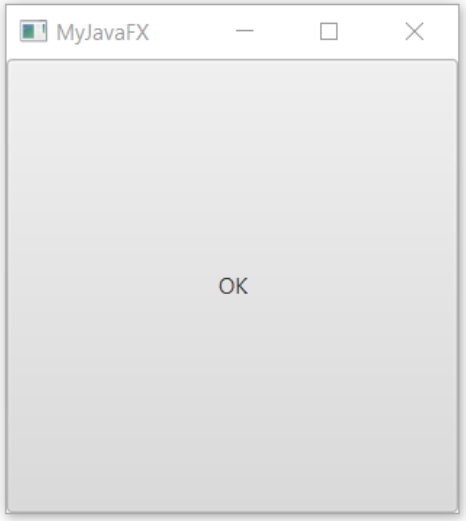
- A **Stage** object is a window. A **Stage** object called *primary stage* is automatically created by the JVM when the application is launched.
- Line 13 sets the scene to the primary stage and line 14 displays the primary stage.
- JavaFX names the **Stage** and **Scene** classes using the analogy from the theater.
- You may think of stage as the platform to support scenes, and nodes as actors to perform in the scenes.
- You can create additional stages if needed. The JavaFX program in Listing 14.2 displays two stages, as shown in Figure 14.2b.

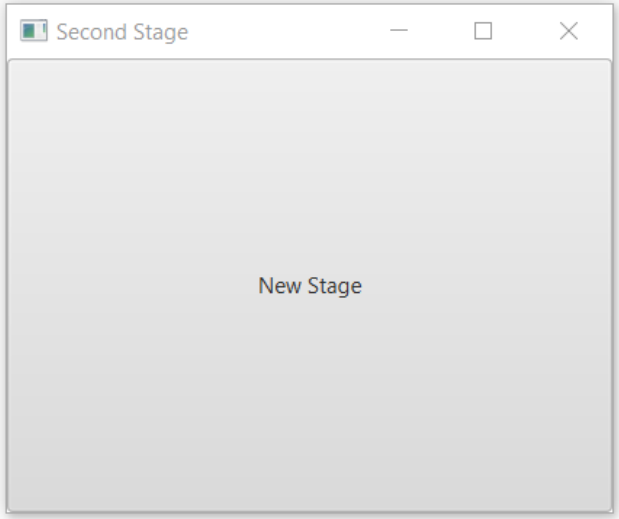
MultipleStageDemo

Run

MultipleStageDemo

```
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"), 200, 250));
19     stage.show(); // Display the stage
20   }
21
22   /**
23    * The main method is only needed for the IDE with limited
24    * JavaFX support. Not needed for running from the command line.
25    */
26   public static void main(String[] args) {
27     launch(args);
28   }
29 }
```





How It Works

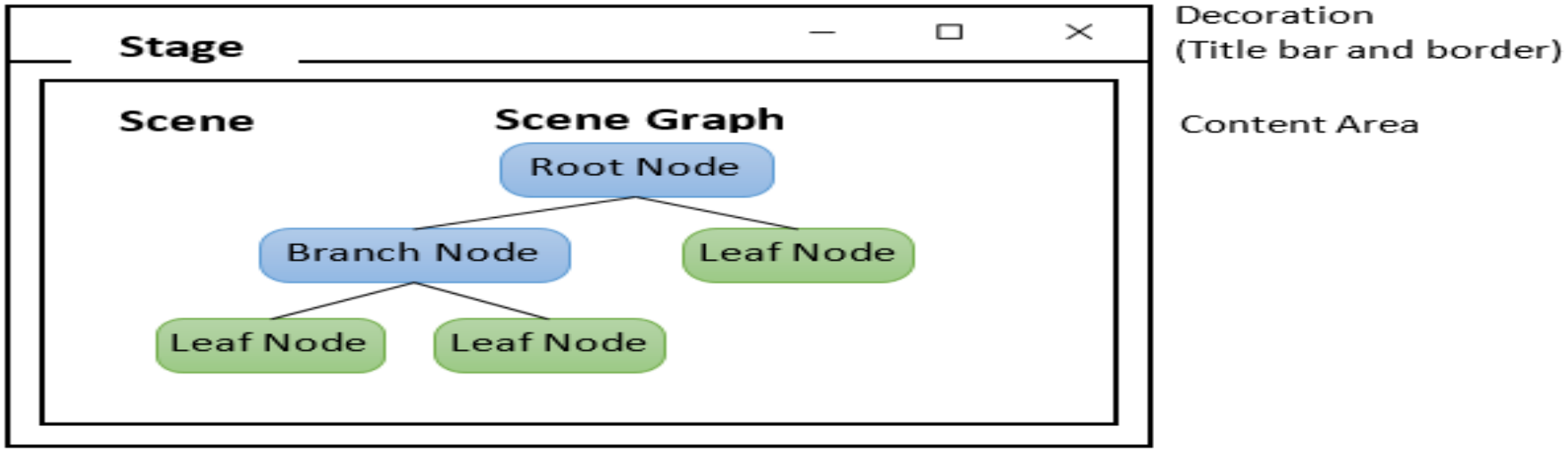
1.JavaFX uses the metaphor of a theater to model the graphics application. A *stage* (defined by the `javafx.stage.Stage` class) represents the top-level container (window). The individual controls (or components) are contained in a *scene* (defined by the `javafx.scene.Scene` class). An application can have more than one scenes, but only one of the scenes can be displayed on the stage at any given time. The contents of a scene is represented in a hierarchical *scene graph of nodes* (defined by `javafx.scene.Node`).

2.To construct the UI:

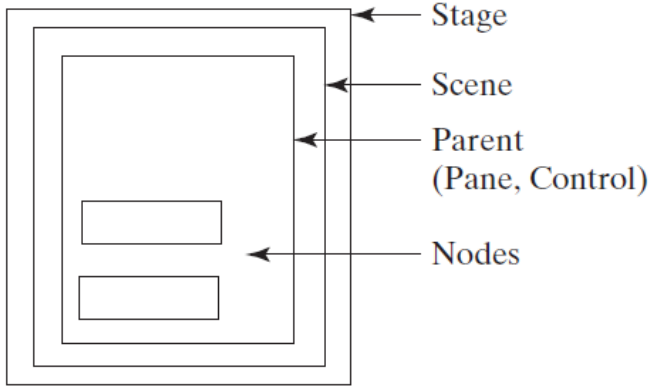
1.Prepare a scene graph.

2.Construct a scene, with the root node of the scene graph.

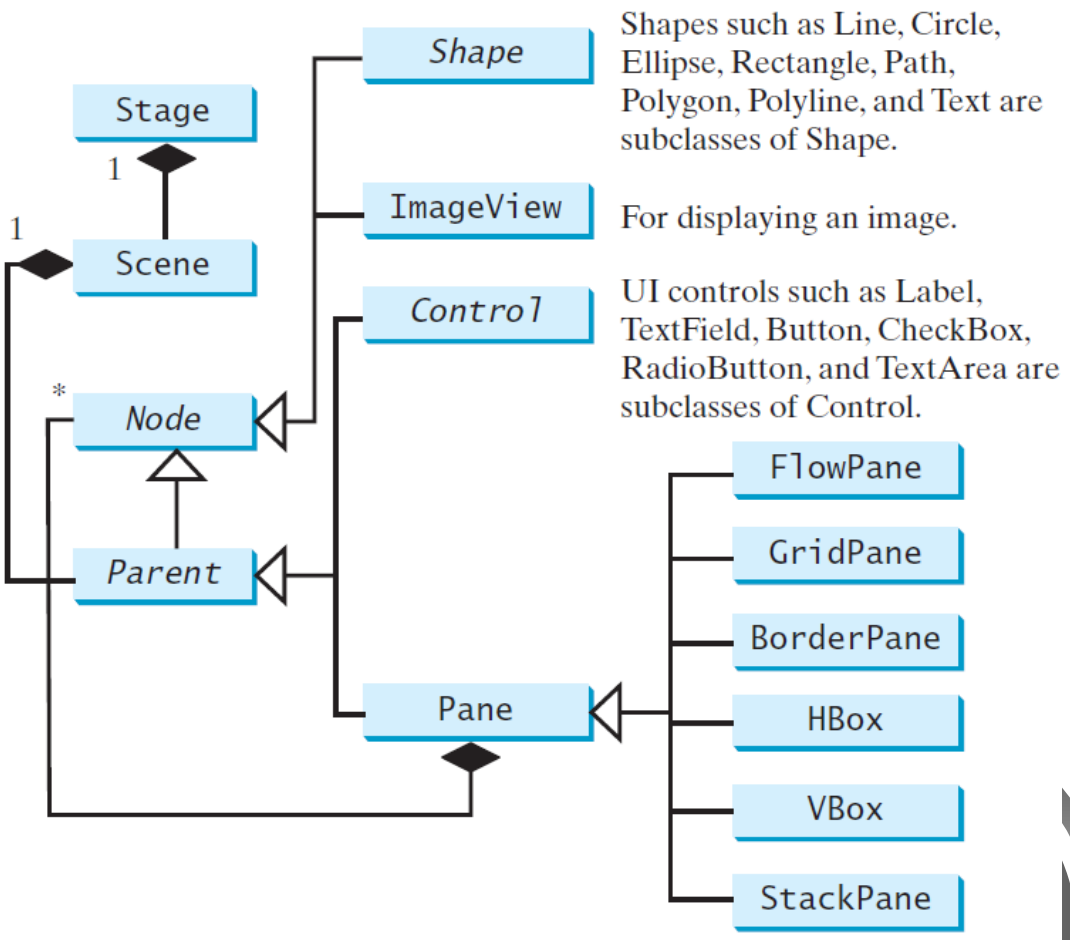
3.Setup the stage with the constructed scene.



Panes, UI Controls, and Shapes



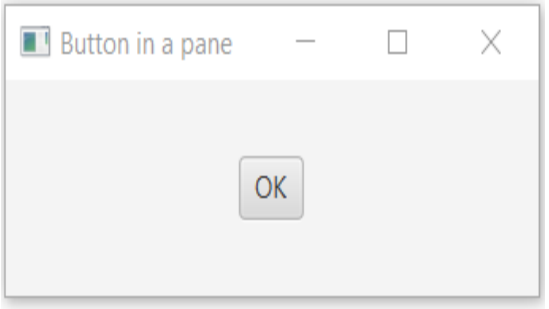
(a)



(b)

ButtonInPane Run

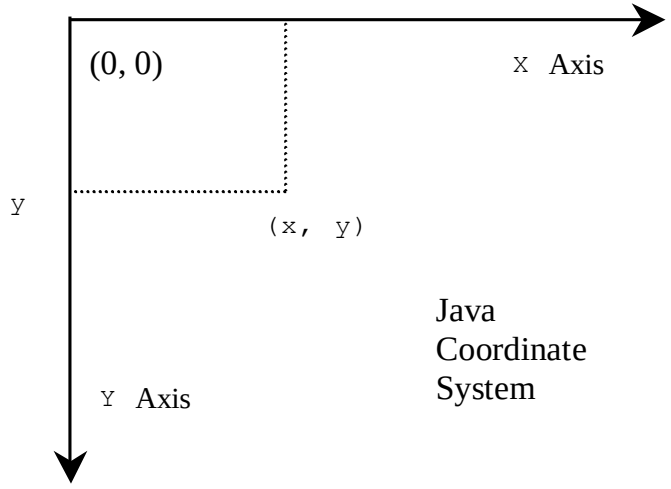
ButtonInPane



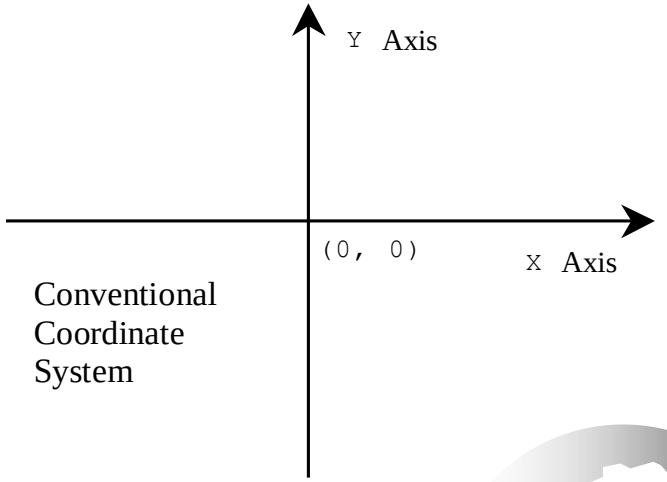

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.scene.layout.StackPane;
5  import javafx.stage.Stage;
6
7  public class ButtonInPane extends Application {
8    @Override // Override the start method in the Application 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
19   /**
20    * The main method is only needed for the IDE with limited
21    * JavaFX support. Not needed for running from the command line.
22    */
23   public static void main(String[] args) {
24     launch(args);
25   }
26 }
```

Display a Shape

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



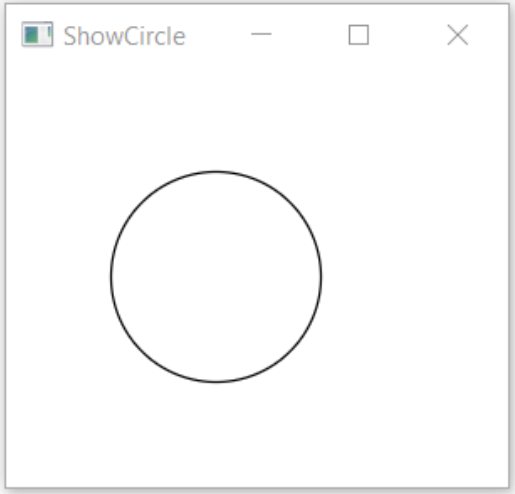
Java Coordinate System



Conventional Coordinate System

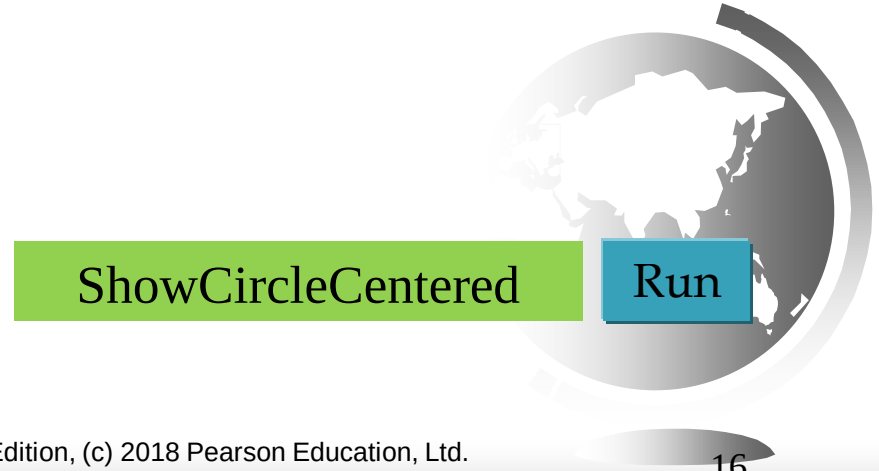
ShowCircle

```
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); // Set circle stroke color
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
30   /**
31    * The main method is only needed for the IDE with limited
32    * JavaFX support. Not needed for running from the command line.
33    */
34   public static void main(String[] args) {
35     launch(args);
36   }
37 }
```



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 Property: getter, setter, and property getter

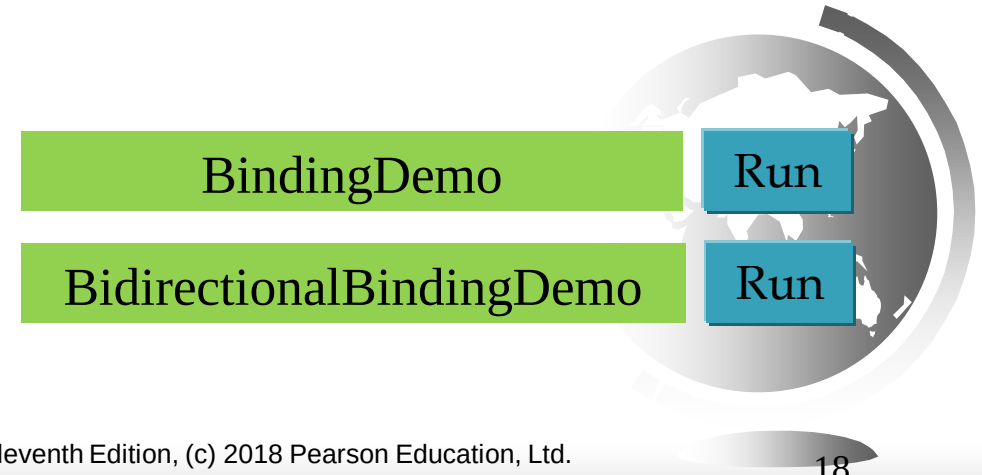
```
public class SomeClassName {

  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

Uni/Bidirectional Binding



Common Properties and Methods for Nodes

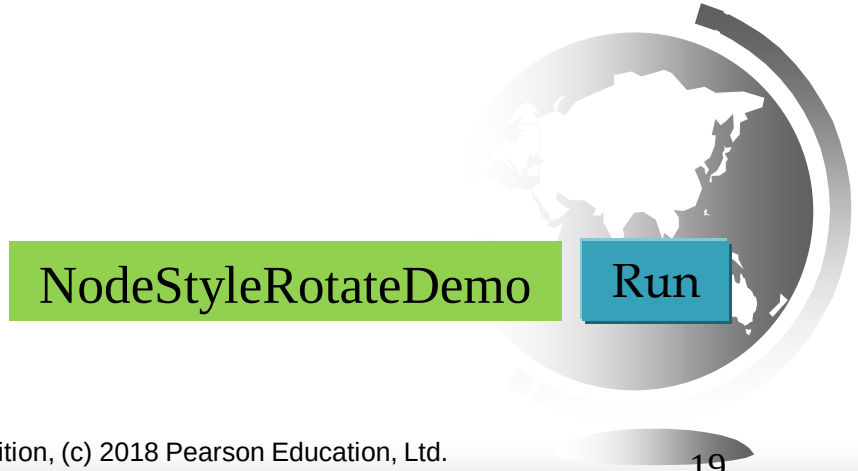
- JavaFX style properties are similar to cascading style sheets (CSS) used to specify the styles for HTML elements in a Web page. Therefore, the style properties in JavaFX are called JavaFX CSS.
- In JavaFX, a style property is defined with a prefix –fx–. Each node has its own style properties.
- You can find these properties at docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html.
- The syntax for setting a style is styleName:value.
- Multiple style properties for a node can be set together separated by semicolon (;). For example, the following statement:

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

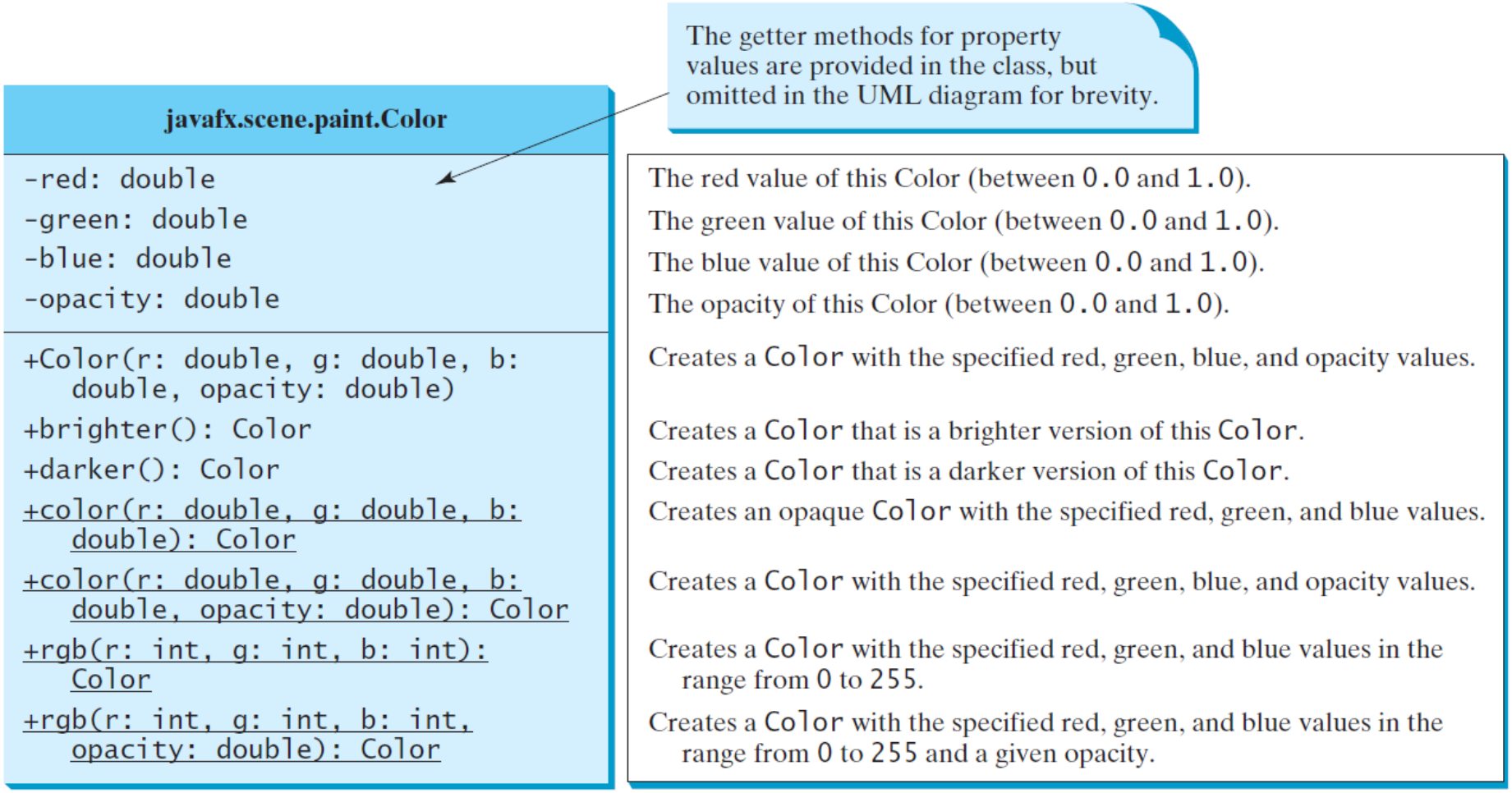
- This statement is equivalent to the following two statements:

```
circle.setStroke(Color.BLACK);

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

- rotate: Rotate a node

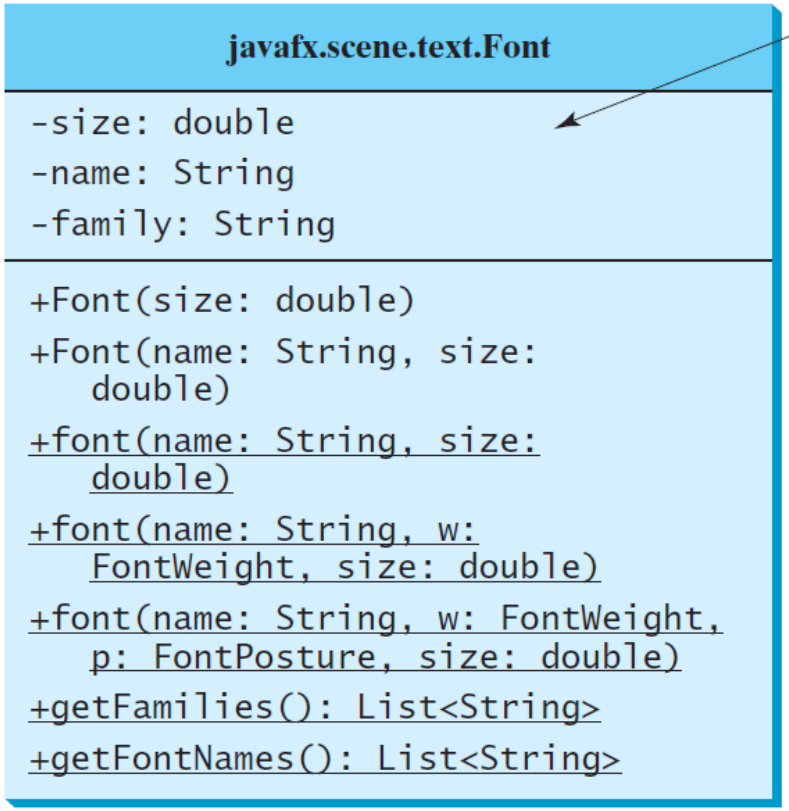


The Color Class



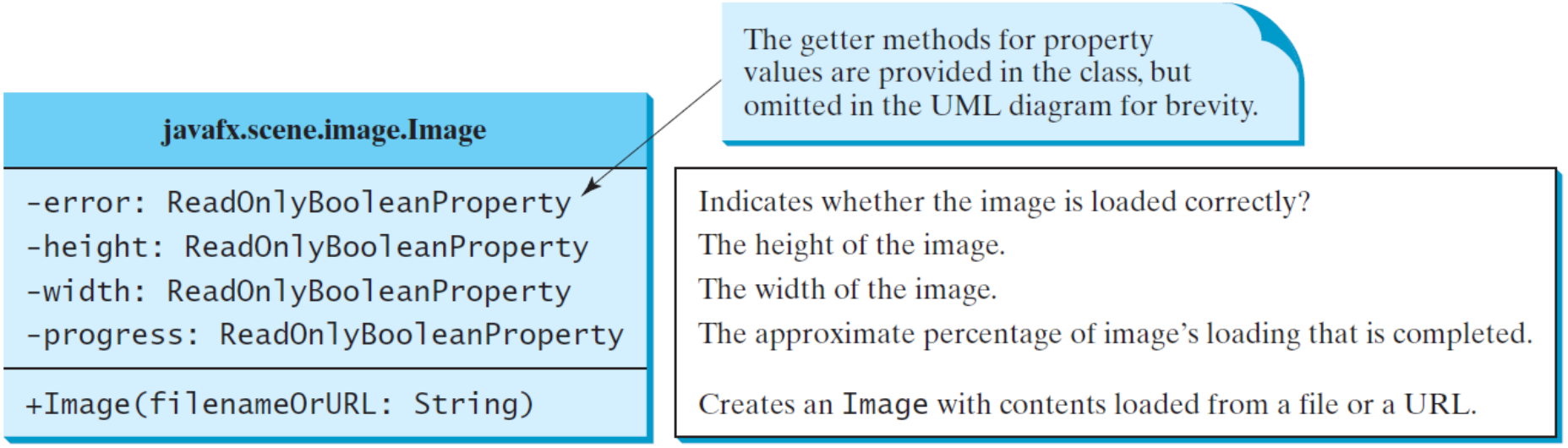
The Font Class

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



FontDemo Run

The Image Class



The ImageView Class

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
-fitWidth: DoubleProperty
-x: DoubleProperty
-y: DoubleProperty
-image: ObjectProperty<Image>
```

```
+ImageView()
+ImageView(image: Image)
+ImageView(filenameOrURL: String)
```

The height of the bounding box within which the image is resized to fit.
The width of the bounding box within which the image is resized to fit.
The x-coordinate of the ImageView origin.
The y-coordinate of the ImageView origin.
The image to be displayed in the image view.

Creates an `ImageView`.
Creates an `ImageView` with the specified image.
Creates an `ImageView` with image loaded from the specified file or URL.

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>
-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.

MultipleStageDemo 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>
-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.

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

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

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

Creates a `BorderPane`.

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

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>
-fillHeight: BooleanProperty
-spacing: DoubleProperty
```

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

The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).

Is resizable children fill the full height of the box (default: `true`).

The horizontal gap between two nodes (default: `0`).

Creates a default `HBox`.

Creates an `HBox` with the specified horizontal gap between nodes.

Sets the margin for the node in the pane.

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>
-fillWidth: BooleanProperty
-spacing: DoubleProperty
```

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

The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).

Is resizable children fill the full width of the box (default: `true`).

The vertical gap between two nodes (default: `0`).

Creates a default `VBox`.

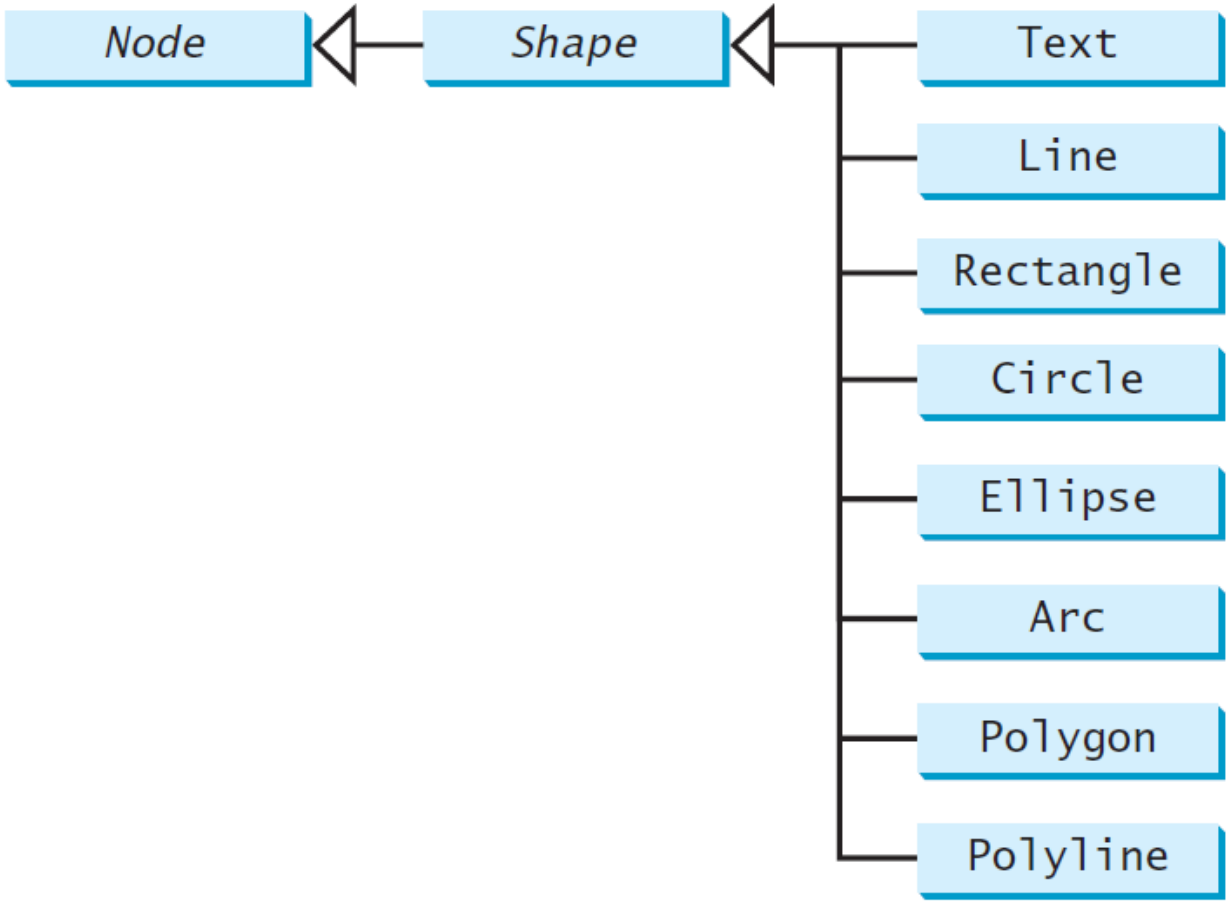
Creates a `VBox` with the specified horizontal gap between nodes.

Sets the margin for the node in the pane.

ShowHBoxVBox Run

Shapes

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



Text

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

javafx.scene.text.Text

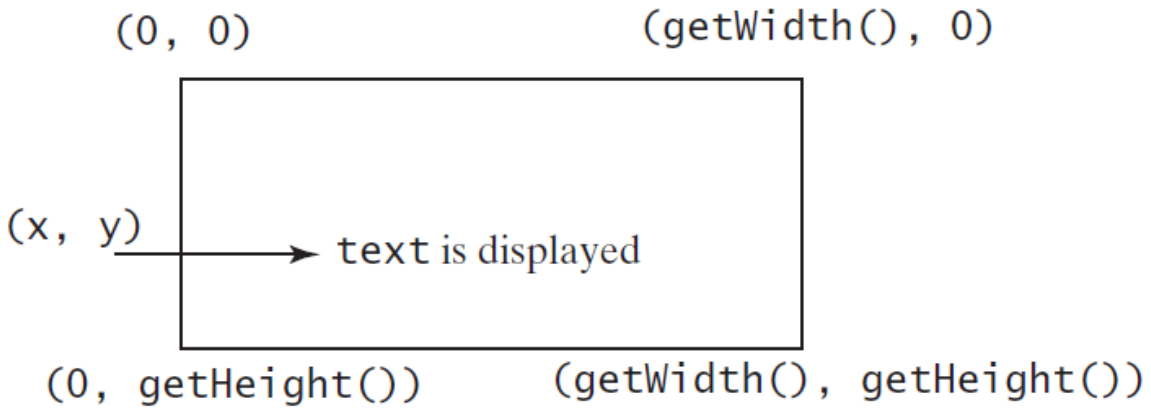
```
-text: StringProperty
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
```

```
+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



(a) Text(x, y, text)

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

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.

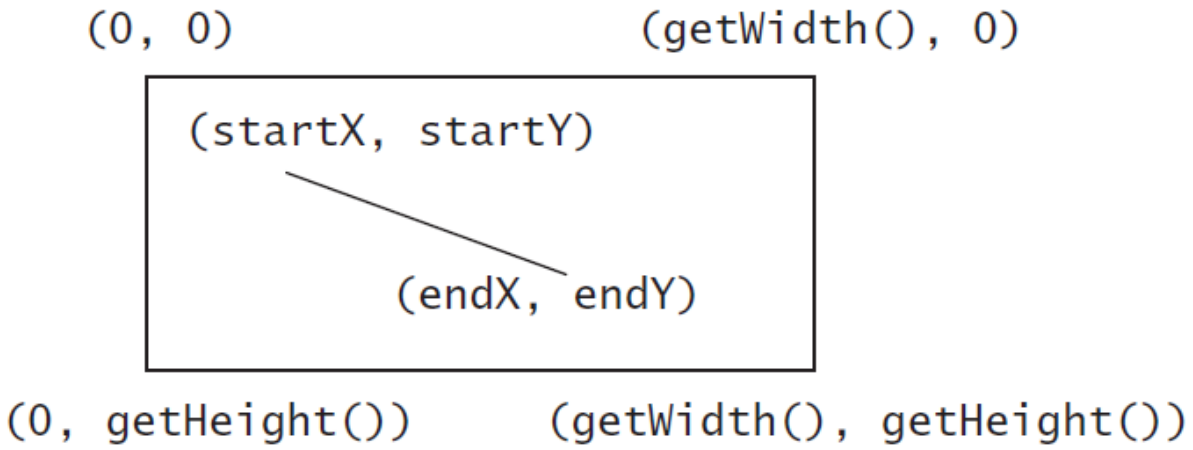
javafx.scene.shape.Line

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

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

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.



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
-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).

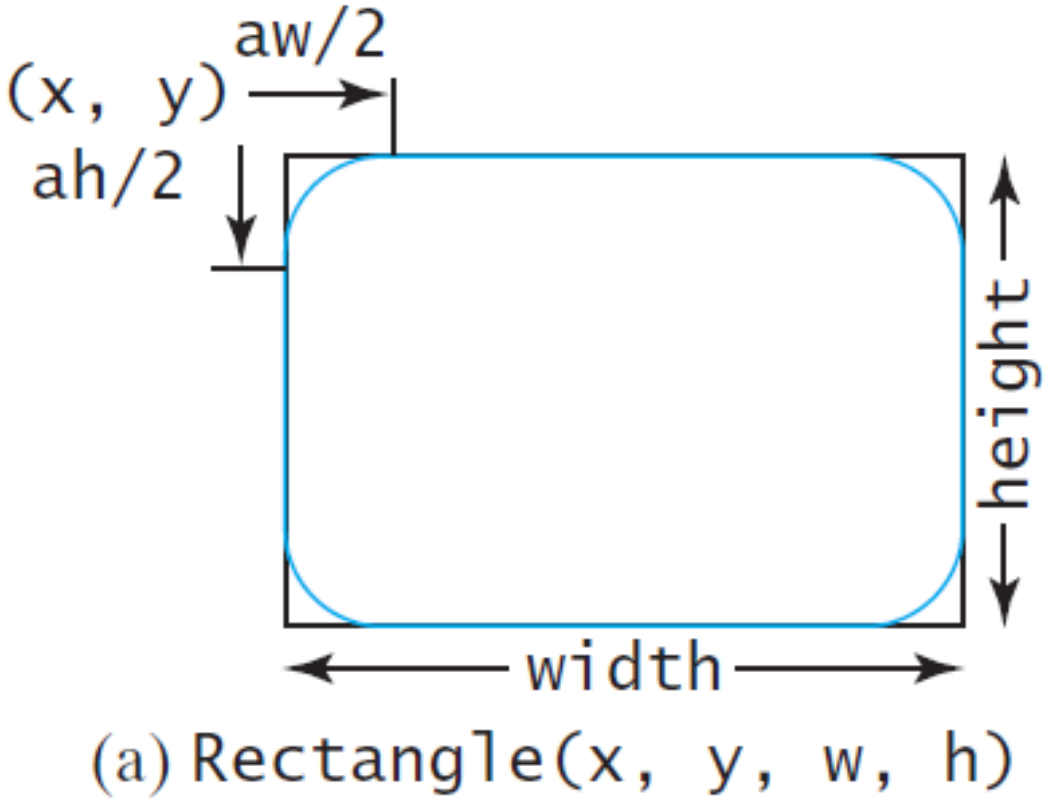
The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

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.

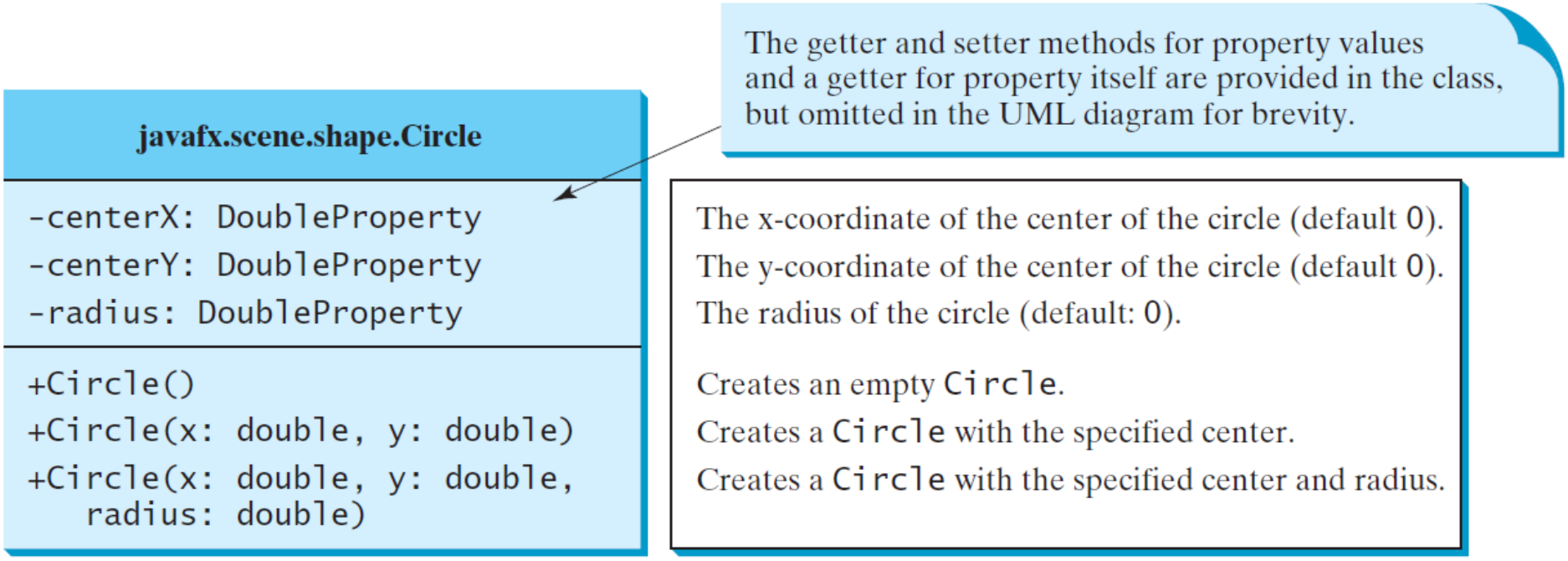
Rectangle Example



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

ShowRectangle Run

Circle



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.

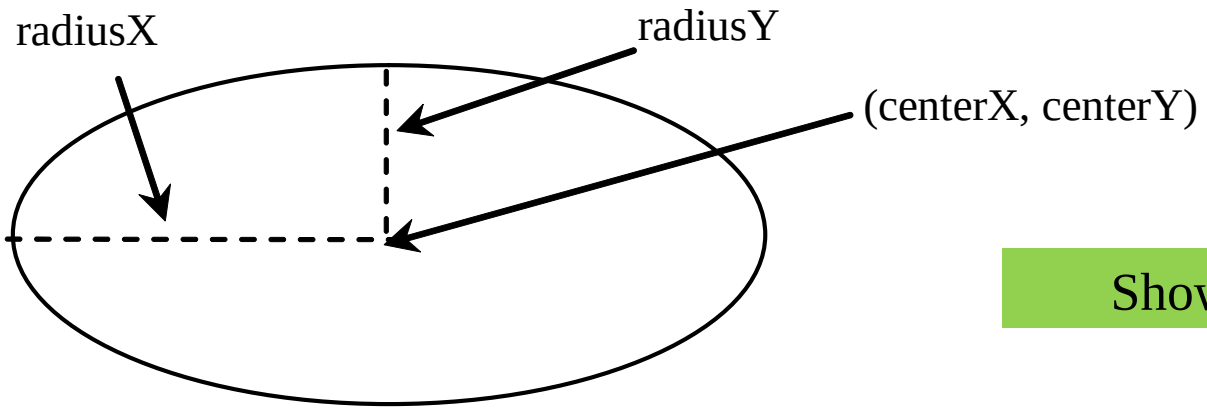
javafx.scene.shape.Ellipse

```
-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.



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
-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 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.

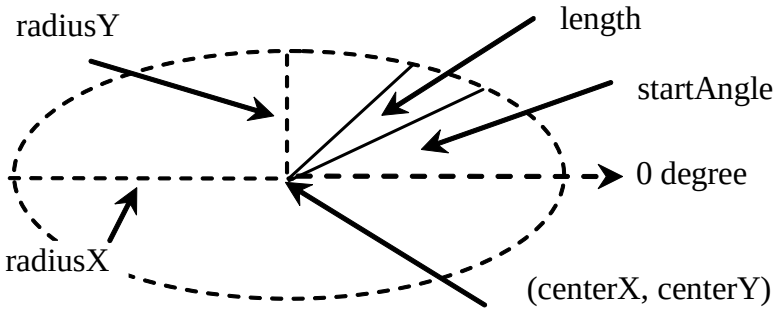
The angular extent of the arc in degrees.

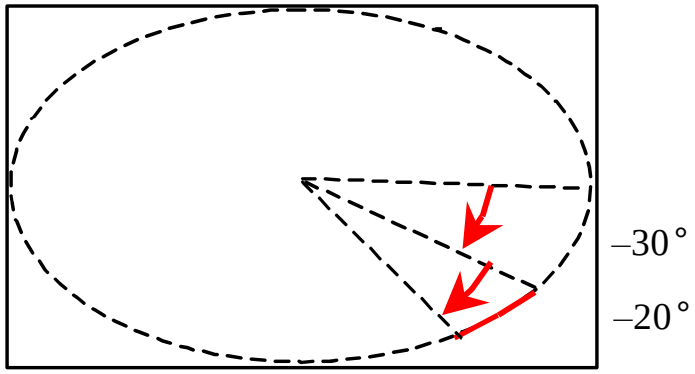
The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

Creates an empty `Arc`.

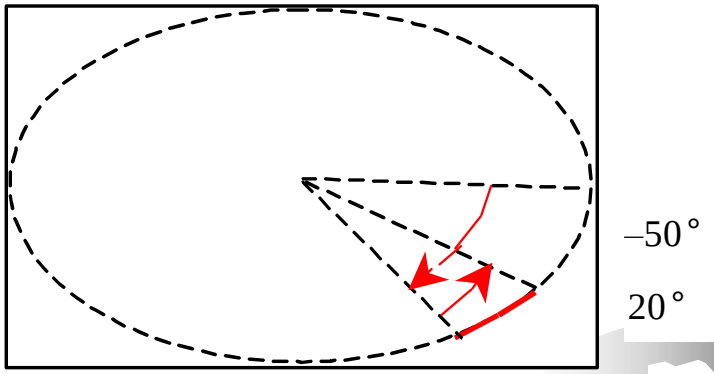
Creates an `Arc` with the specified arguments.

Arc Examples





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

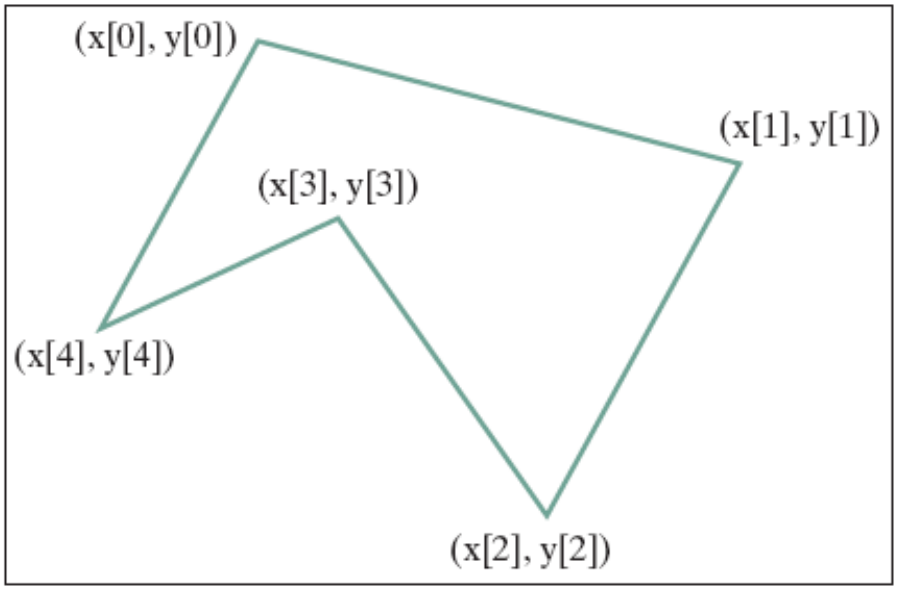


(b) Negative starting angle –50° and positive spanning angle 20°

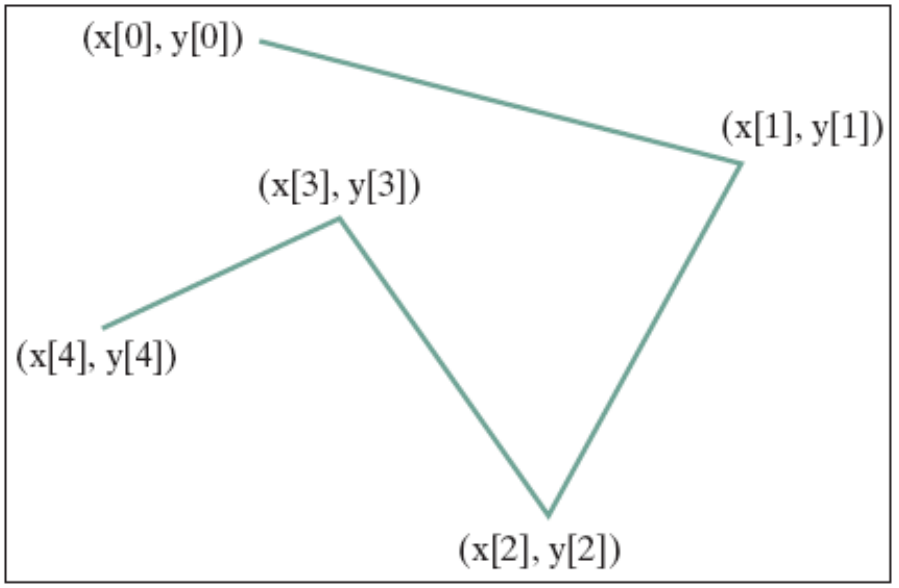
ShowArc

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.

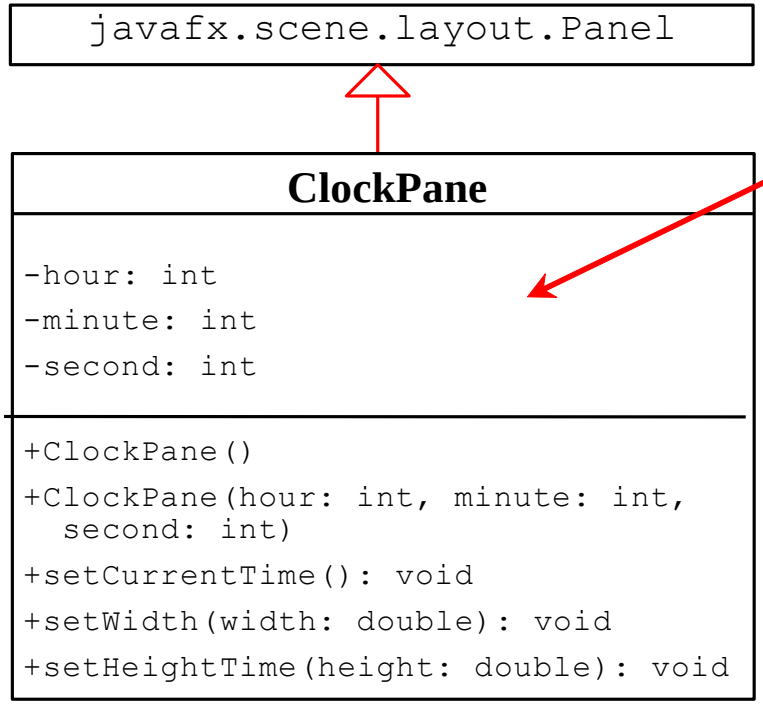
Creates an empty polygon.

Creates a polygon with the given points.

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

Case Study: The ClockPane Class

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



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,

ClockPane

Use the ClockPane Class