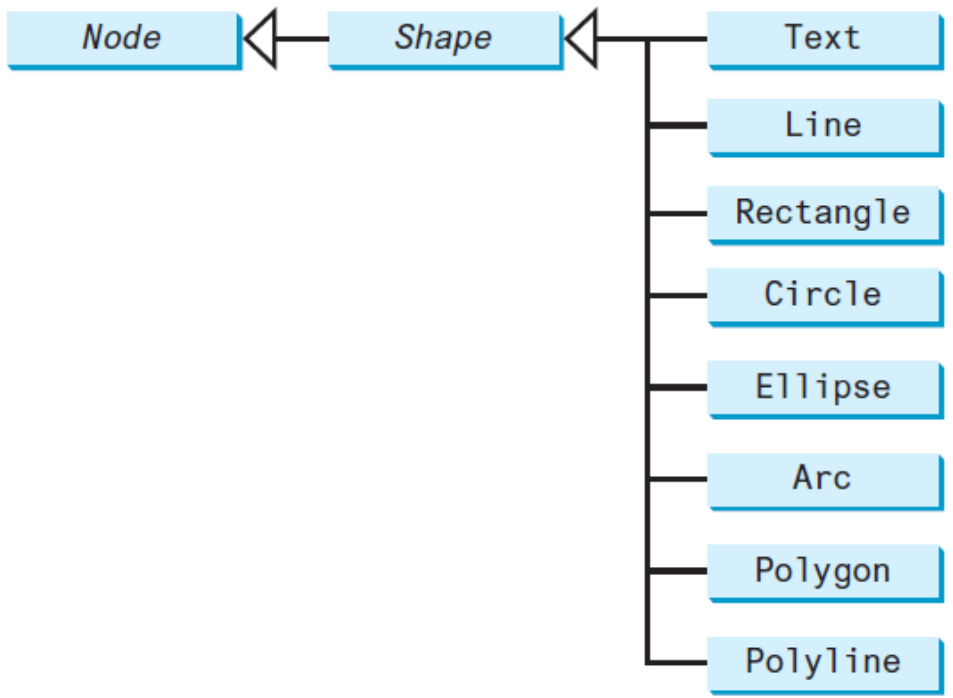
Shapes

Week 6

Shapes

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

The `Shape` class is the abstract base class that defines the common properties for all shapes. Among them are the `fill`, `stroke`, and `strokeWidth` properties. The `fill` property specifies a color that fills the interior of a shape. The `stroke` property specifies a color that is used to draw the outline of a shape. The `strokeWidth` property specifies the width of the outline of a shape. This section introduces the classes `Text`, `Line`, `Rectangle`, `Circle`, `Ellipse`, `Arc`, `Polygon`, and `Polyline` for drawing texts and simple shapes. All these are subclasses of `Shape`, as shown in Figure 14.25.



A shape is a node. The `Shape` class is the root of all shape classes.

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

javafx.scene.text.Text

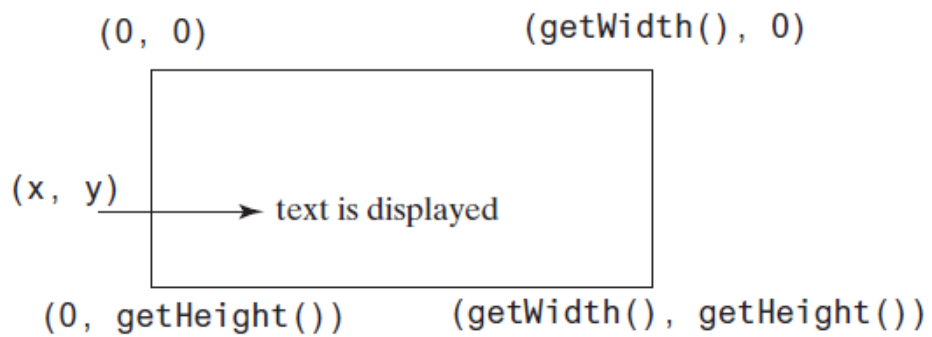
```
-text: StringProperty
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
```

```
+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` defines a node for displaying a text.



(a) `Text`(*x*, *y*, text)

(b) Three `Text` objects are displayed

A Text object is created to display a text.

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

javafx.scene.shape.Line

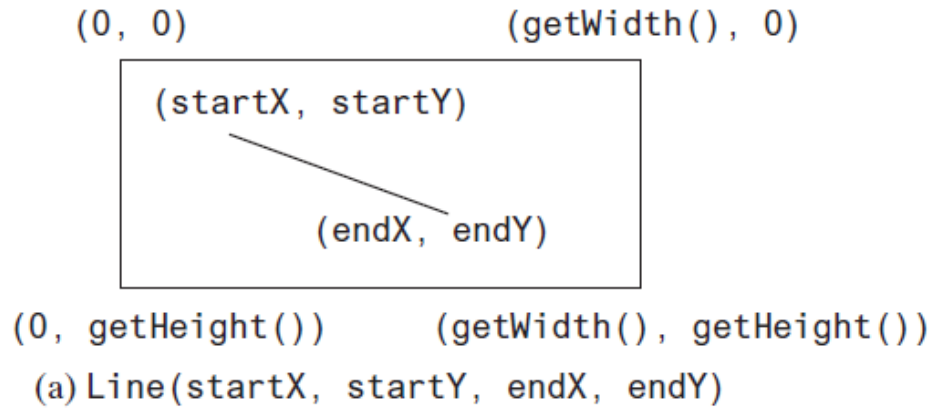
```
-startX: DoubleProperty
-startY: DoubleProperty
-endX: DoubleProperty
-endY: DoubleProperty

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

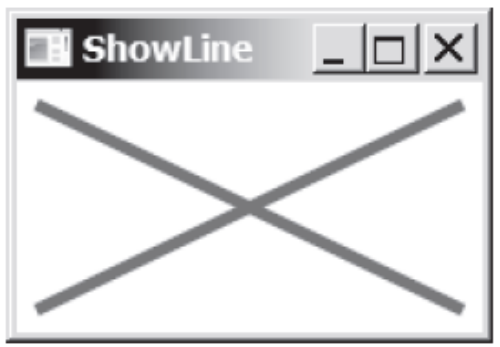
The *x*-coordinate of the start point.
The *y*-coordinate of the start point.
The *x*-coordinate of the end point.
The *y*-coordinate of the end point.

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

The `Line` class defines a line.



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



(b) Two lines are displayed across the pane.

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

The getter and setter me thods 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()
+Rectangle(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).

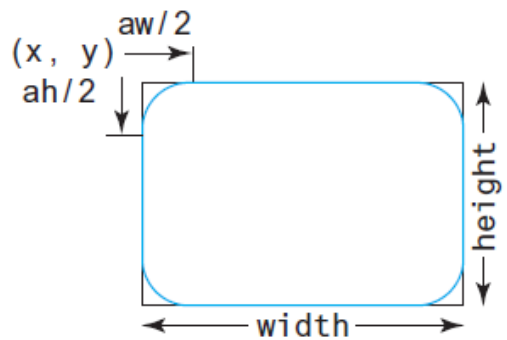
The `arcWidth` of the rectangle (default: 0). arcWidth is the horizontal diameter of the arcs at the corner (see Figure 14.31a).

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

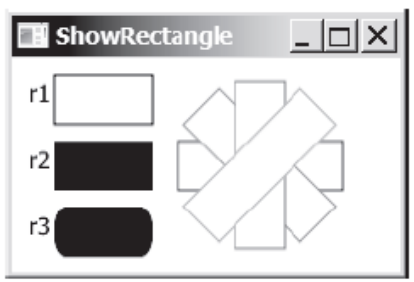
Creates an empty `Rectangle`.

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

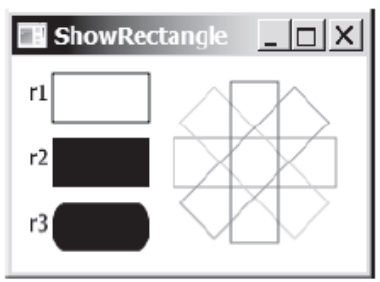
`Rectangle` defines a rectangle.



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

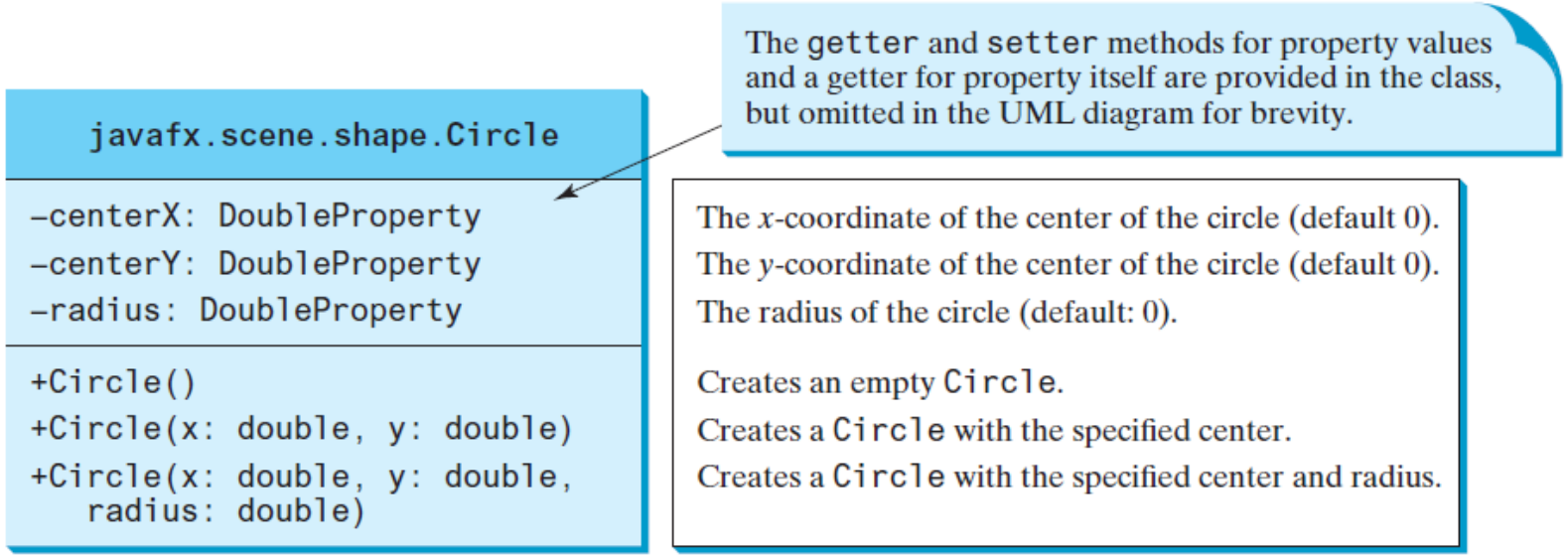


(b) Multiple rectangles are displayed.

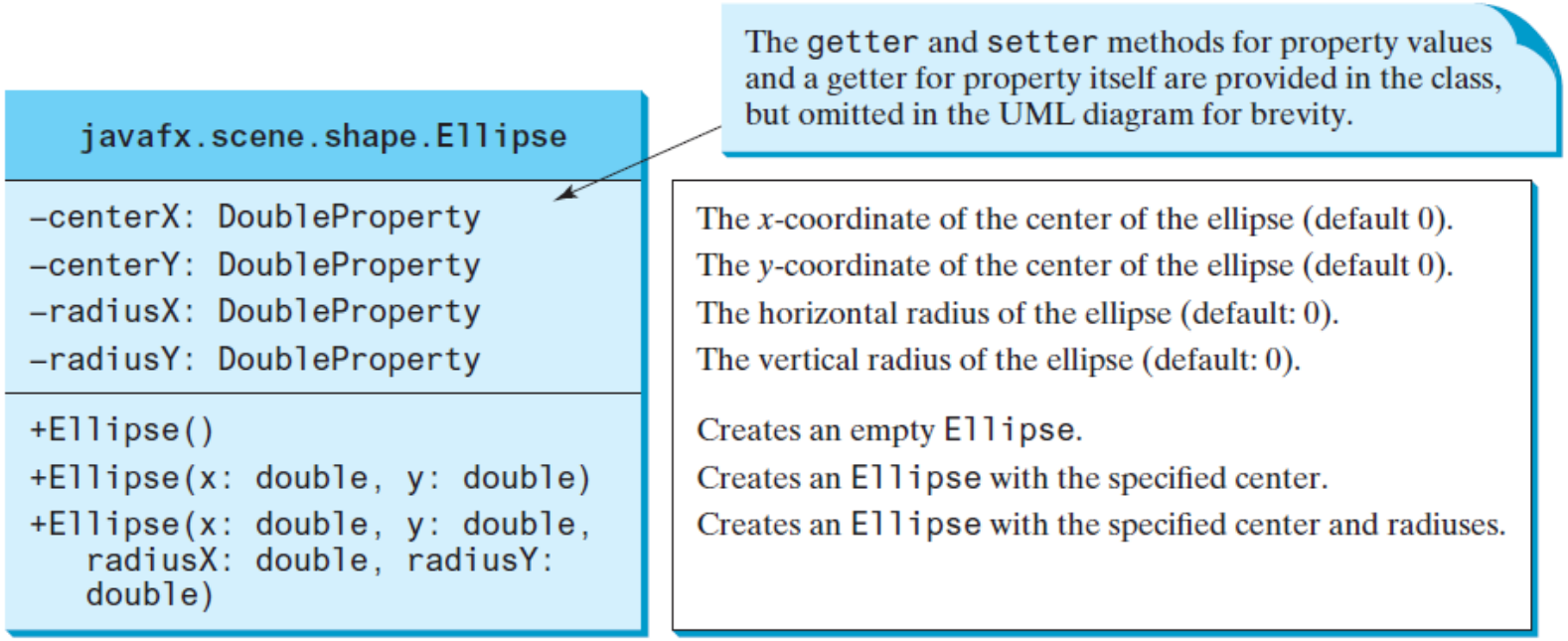


(c) Transparent rectangles are displayed.

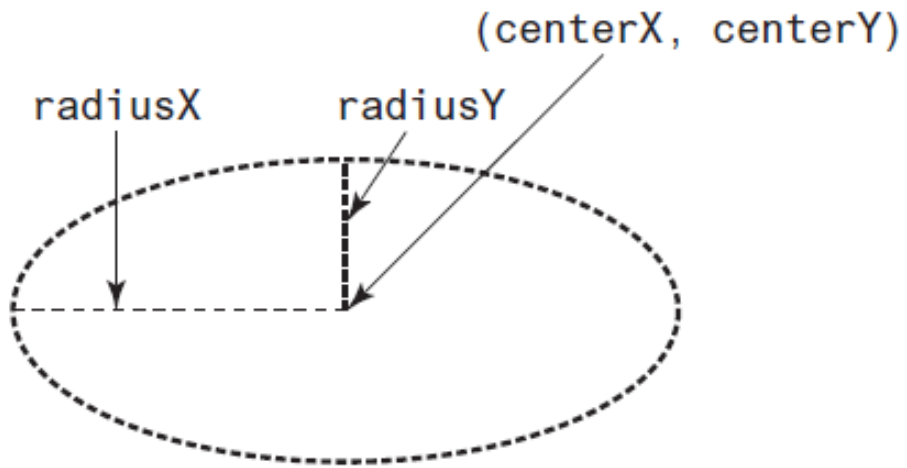
A `Rectangle` object is created to display a rectangle.



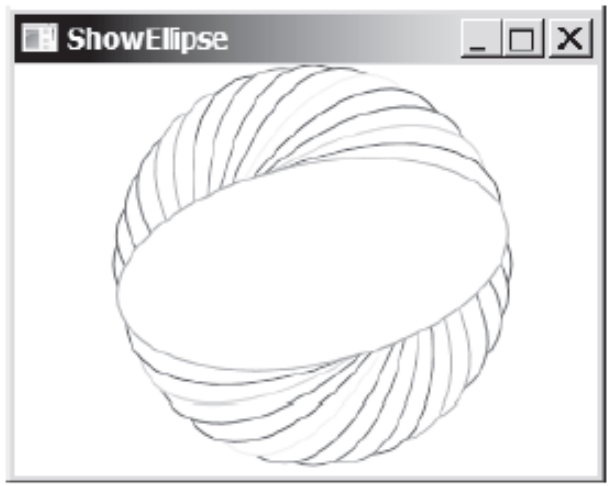
The `Circle` class defines circles.



The `Ellipse` class defines ellipses.



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



(b) Multiple ellipses are displayed.

An `Ellipse` object is created to display an 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.Arc

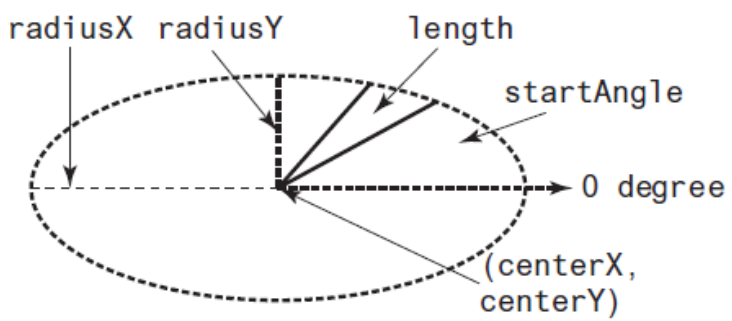
```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
```

```
+Arc()
+Arc(x: double, y: double,
   radiusX: double, radiusY:
   double, startAngle: double,
   length: double)
```

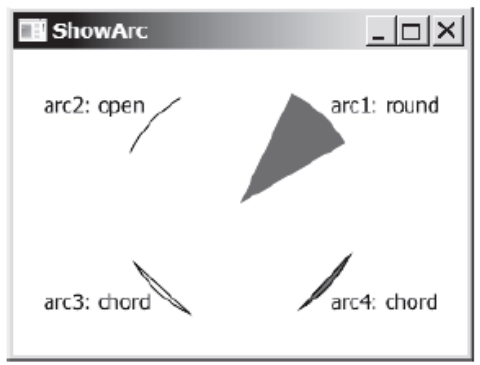
The *x*-coordinate of the center of the ellipse (default 0).
The *y*-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).
The start angle of the arc in degrees.
The angular extent of the arc in degrees.
The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

Creates an empty `Arc`.
Creates an `Arc` with the specified arguments.

The `Arc` class defines an arc.

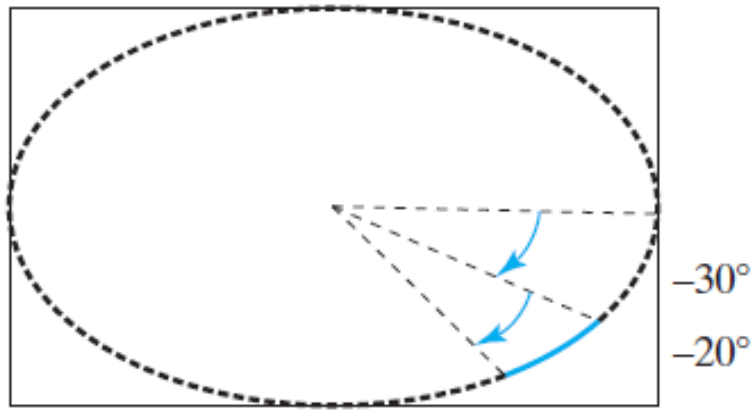


(a) `Arc(centerX, centerY, radiusX, radiusY, startAngle, length)`

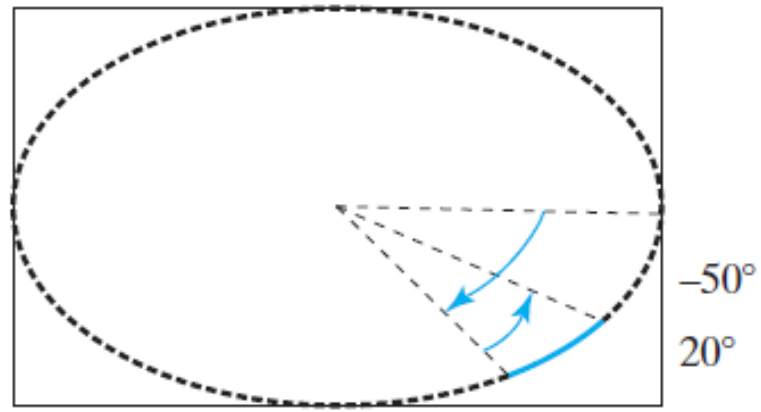


(b) Multiple ellipses are displayed.

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

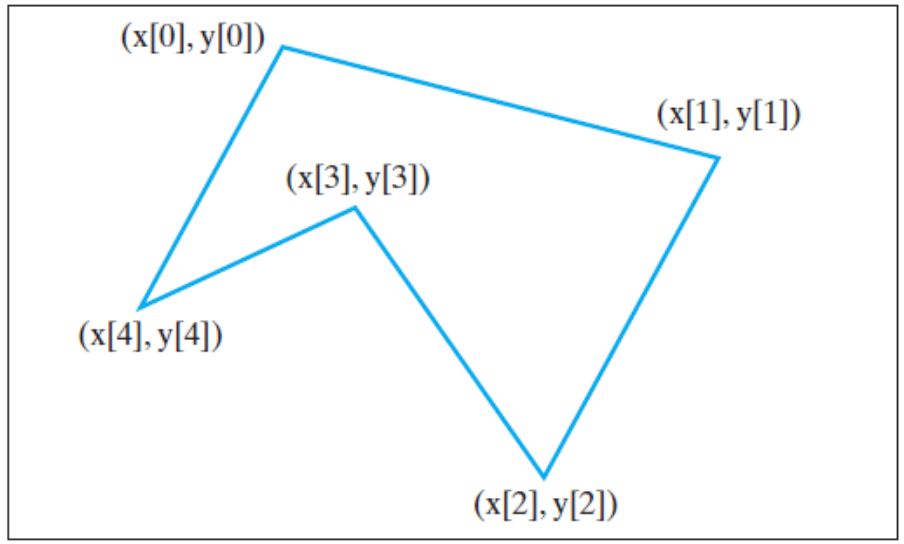


(a) Negative starting angle –30° and negative spanning angle –20°

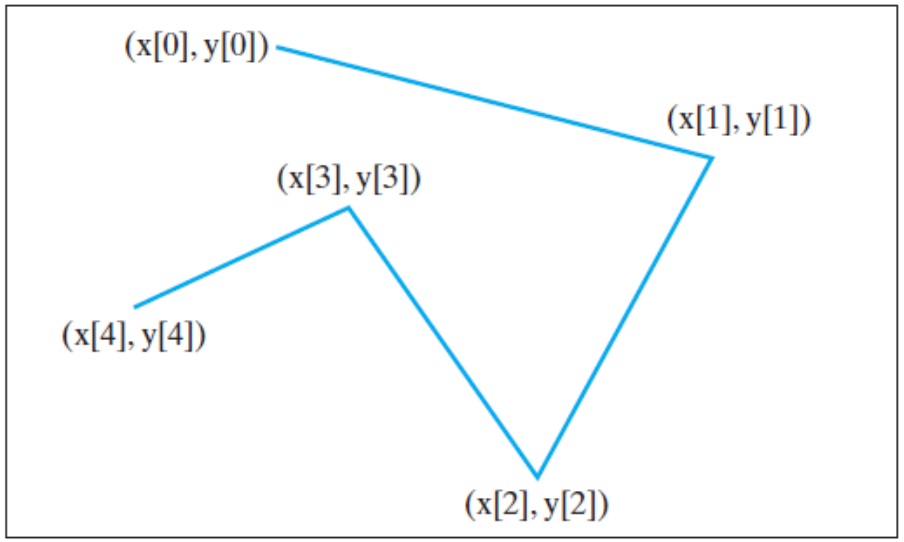


(b) Negative starting angle –50° and positive spanning angle 20°

Angles may be negative.



(a) Polygon

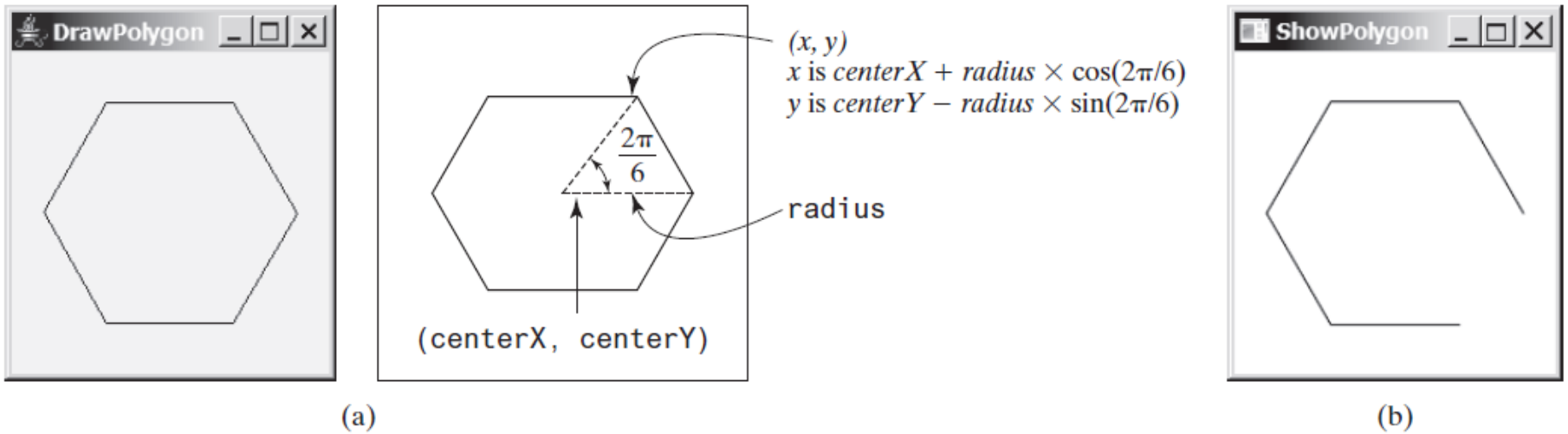


(b) Polyline

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

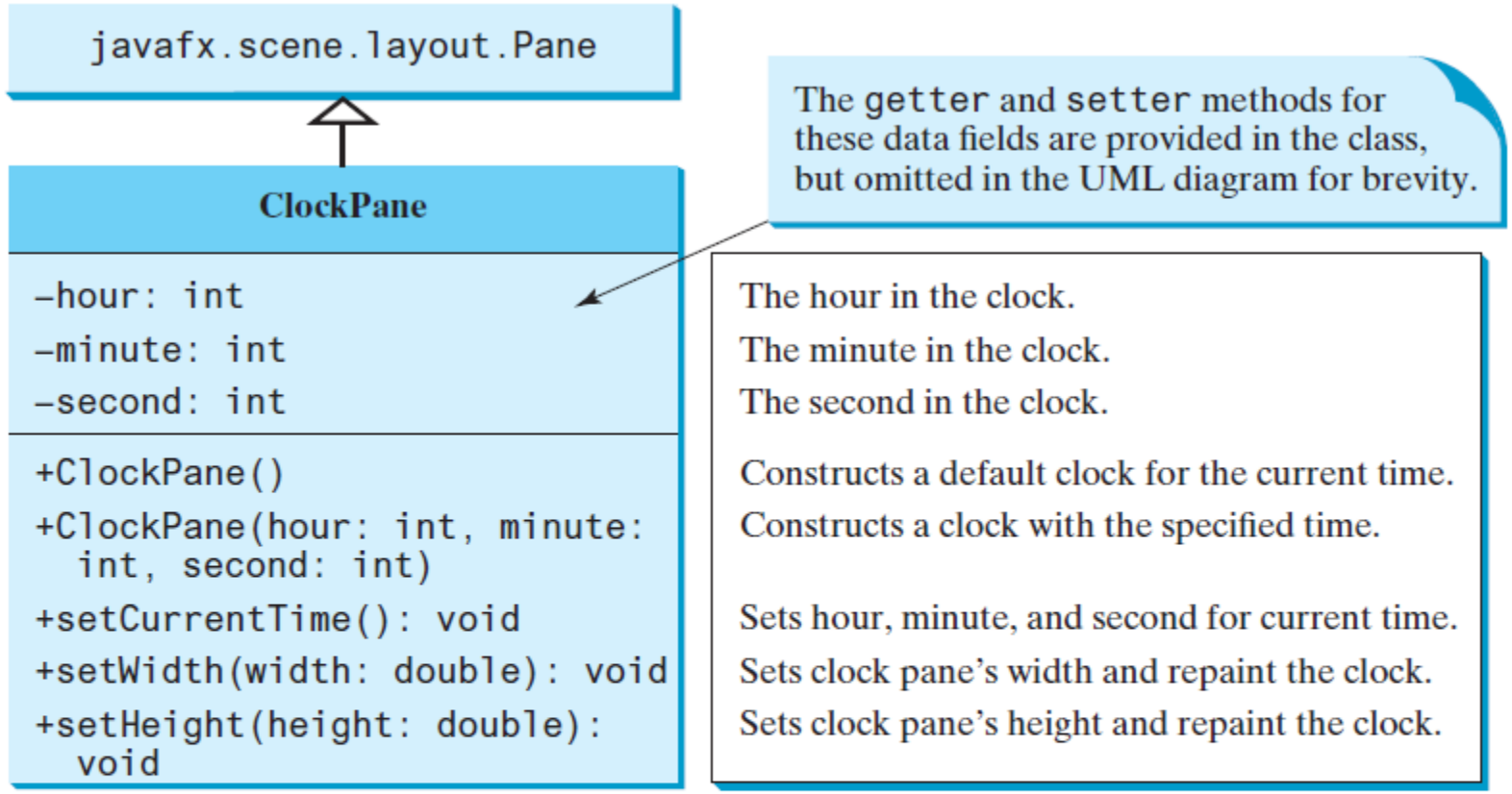
javafx.scene.shape.Polygon	
+Polygon()	Creates an empty `Polygon`.
+Polygon(double... points)	Creates a `Polygon` with the given points.
+getPoints(): ObservableList<Double>	Returns a list of double values as *x*- and *y*-coordinates of the points.

The `Polygon` class defines a polygon.



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

If you replace `Polygon` by `Polyline` (line 23), the program displays a polyline as shown in Figure 14.40b. The `Polyline` class is used in the same way as `Polygon`, except that the starting and ending points are not connected in `Polyline`.



`ClockPane` displays an analog clock.

Assume `ClockPane` is available; we write a test program in Listing 14.20 to display an analog clock and use a label to display the hour, minute, and second, as shown in Figure 14.42.

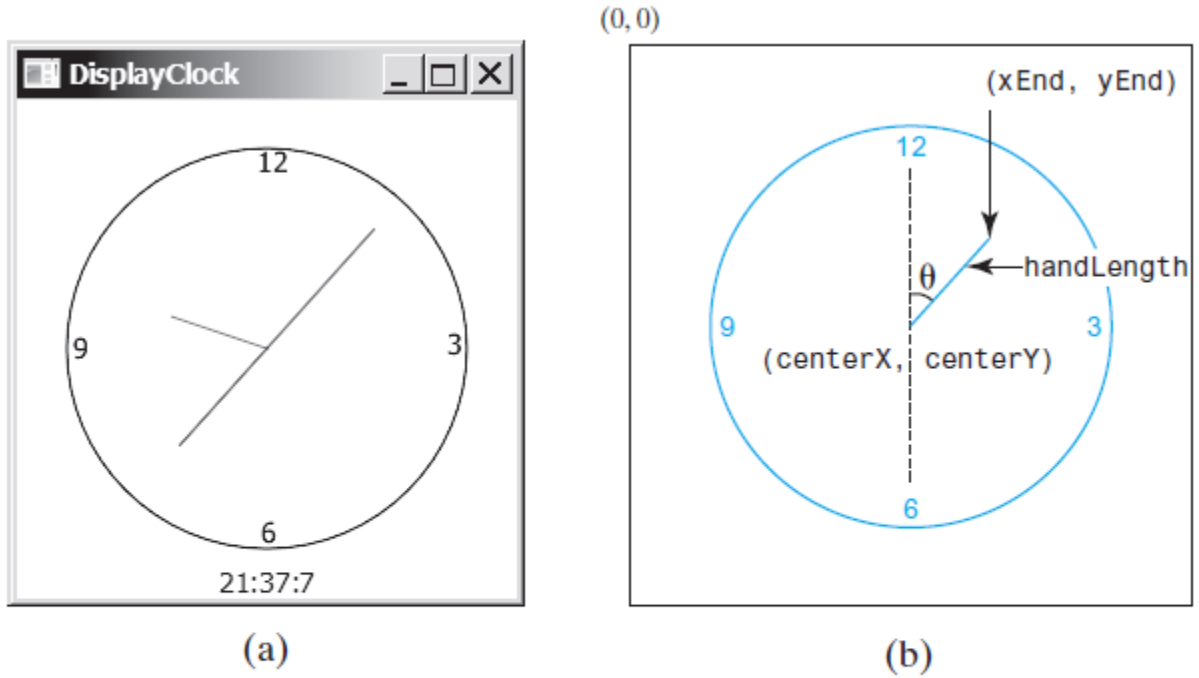


FIGURE 14.42 (a) The `DisplayClock` program displays a clock that shows the current time. (b) The endpoint of a clock hand can be determined, given the spanning angle, the hand length, and the center point.

DisplayClock.java

```
1  import javafx.application.Application;
2  import javafx.geometry.Pos;
3  import javafx.stage.Stage;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Label;
6  import javafx.scene.layout.BorderPane;
7
8  public class DisplayClock extends Application {
9    @Override // Override the start method in the Application class
10   public void start(Stage primaryStage) {
11     // Create a clock and a label
12     ClockPane clock = new ClockPane();                          create a clock
13     String timeString = clock.getHour() + ":" + clock.getMinute()
14       + ":" + clock.getSecond();
15     Label lblCurrentTime = new Label(timeString);               create a label
16
17     // Place clock and label in border pane
18     BorderPane pane = new BorderPane();
19     pane.setCenter(clock);                                      add a clock
20     pane.setBottom(lblCurrentTime);                             add a label
21     BorderPane.setAlignment(lblCurrentTime, Pos.TOP_CENTER);
22
23     // Create a scene and place it in the stage
24     Scene scene = new Scene(pane, 250, 250);
25     primaryStage.setTitle("DisplayClock"); // Set the stage title
26     primaryStage.setScene(scene); // Place the scene in the stage
27     primaryStage.show(); // Display the stage
28   }
29 }
```