8.16 (Optional) GUI and Graphics Case Study: Using Objects with Graphics

Recall from Section 3.6 that one of our goals in this case study is to use object-oriented programming concepts to create an app that draws a variety of shapes. Eventually, we'll want to store the shapes the user draws in a collection. Each shape will know its own location, size and color. Then, the app will be able to recreate the drawing by iterating through the collection and displaying each shape. Also, by using a collection, you'll easily be able to add an *undo* capability to the drawing app—each time the user presses an **Undo** `Button`, you can simply remove the last shape in the collection and then iterate through the collection and redraw the shapes that remain. To do this, we'll create a set of shape classes that store information about each shape. We'll make these classes "smart" by allowing objects of these classes to *draw themselves* when provided with a JavaFX `GraphicsContext` object. In this section's `DrawRandomLines` example, we'll begin with a class named `MyLine` that encapsulates the information for drawing a line. For this section's example, we'll store the `MyLine` objects in an array, but you could also use an `ArrayList`<MyLine> (Section 7.16).

Class `MyLine`

Figure 8.17 declares class `MyLine`, which imports classes `GraphicsContext` and `Color` (lines 3–4). Lines 7–10 declare instance variables for the line's starting and ending coordinates, and line 11 declares the instance variable that stores the line's stroke color. The constructor at lines 14–22 takes five parameters, one for each instance variable that it initializes. Method `draw` at lines 25–28 receives a `GraphicsContext` object and uses it to draw a line in the proper color between the start and end points.

```
1   // Fig. 8.17: MyLine.java
2   // MyLine class represents a line.
3   import javafx.scene.canvas.GraphicsContext;
4   import javafx.scene.paint.Color;
5
6   public class MyLine {
7      private double x1; // x-coordinate of first e
8      private double y1; // y-coordinate of first e
9      private double x2; // x-coordinate of second
10     private double y2; // y-coordinate of second
11     private Color strokeColor; // color of this l
12
13     // constructor with input values
14     public MyLine(
15        double x1, double y1, double x2, double y2
16
17        this.x1 = x1;
18        this.y1 = y1;
19        this.x2 = x2;
20        this.y2 = y2;
21        this.strokeColor = strokeColor;
22     }
23
```

```
24      // draw the line in the specified color
25      public void draw(GraphicsContext gc) {
26          gc.setStroke(strokeColor);
27          gc.strokeLine(x1, y1, x2, y2);
28      }
29  }
```

Fig. 8.17

`MyLine` class represents a line.

Creating the Draw Random Lines App's GUI

The app's GUI is defined in `DrawRandomLines.fxml`. We reused the `DrawLines.fxml` GUI shown in Fig. 4.17, but made the following change to `DrawRandomLines.fxml`:

- We specified `DrawRandomLinesController` as the app's **Controller class** in the Scene Builder **Document** window's **Controller** section.

Class `DrawRandomLinesController`

Again, we do not show the code for `DrawRandomLines.java`, because the only changes from earlier examples are the name of the FXML file to load (`DrawRandomLines.fxml`) and the text displayed in the stage's title bar (`Draw Random Lines`).

In method `drawLinesButtonPressed` (Fig. 8.18), line 21 creates the `MyLine` array `lines` to store 100 lines to draw. Lines 23 and 24 store the `Canvas`'s `width` and `height`, which we use in lines 29–32 when choosing random coordinates. Lines 27–40 create a new `MyLine` for every array element. Lines 29–32 generate random coordinates for each line's endpoints, and lines 35–36 generate a random color. Line 39 creates a new `MyLine` object with the randomly generated values and stores it in the array. Finally, lines 43–47 clear the `Canvas` then iterate through the `MyLine` objects in array `lines` using an enhanced `for` statement. Each iteration calls the current `MyLine`'s `draw` method and passes it the `Canvas`'s `GraphicsContext`. The `MyLine` object actually draws itself.

```
1  // Fig. 8.18: DrawRandomLinesController.java
2  // Drawing random lines using MyLine objects.
3  import java.security.SecureRandom;
4  import javafx.event.ActionEvent;
5  import javafx.fxml.FXML;
6  import javafx.scene.canvas.Canvas;
7  import javafx.scene.canvas.GraphicsContext;
8  import javafx.scene.paint.Color;
9  import javafx.scene.shape.ArcType;
10
11 public class DrawRandomLinesController {
```

```
12      private static final SecureRandom randomNumbe
       13      @FXML private Canvas canvas;
                       14
          15      // draws random lines
                  16      @FXML
17      void drawLinesButtonPressed(ActionEvent event
18         // get the GraphicsContext, which is used
19         GraphicsContext gc = canvas.getGraphicsCon
                       20
21         MyLine[] lines = new MyLine[100]; // store
                       22
23         final nt width = (int) canvas.getWidth();
24         final int height = (int) canvas.getHeight(
                       25
           26         // create lines
27         for (int count = 0; count < lines.length;
   28            // generate random coordinates
29            double x1 = randomNumbers.nextInt(width
30            double y1 = randomNumbers.nextInt(heigh
31            double x2 = randomNumbers.nextInt(width
32            double y2 = randomNumbers.nextInt(heigh
                       33
     34            // generate a random color
35            Color color = Color.rgb(randomNumbers.n
36               randomNumbers.nextInt(256), randomNu
                       37
  38            // add a new MyLine to the array
39            lines[count] = new MyLine(x1, y1, x2, y
                   40         }
                       41
42         // clear the Canvas then draw the lines
43         gc.clearRect(0, 0, canvas.getWidth(), canv
                       44
      45         for (MyLine line : lines) {
           46            line.draw(gc);
                  47         }
                    48      }
                      49   }
```

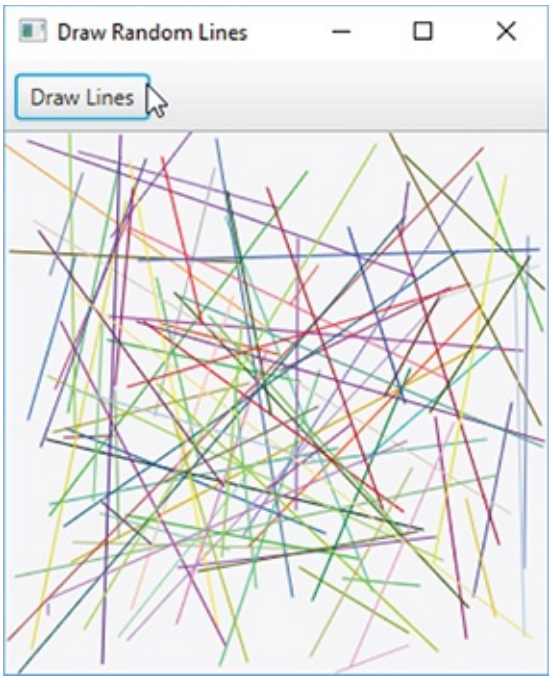

Fig. 8.18

Drawing random lines using `MyLine` objects.

Description

GUI and Graphics Case Study Exercise

1. **8.1** Enhance the program in Figs. 8.17–8.18 to draw the lines in random thicknesses. You can set the line thickness with `GraphicsContext` method `setLineWidth`.
2. **8.2** Enhance the program in Figs. 8.17–8.18 to randomly draw rectangles and ovals. Create classes `MyRectangle` and `MyOval`. Both classes should include *x1*, *y1*, *x2*, *y2* coordinates, a stroke color, a fill color and a

`boolean` flag to determine whether the shape is filled. Declare a constructor in each class with arguments for initializing all the instance variables. To help draw rectangles and ovals, each class should provide methods `getUpperLeftX`, `getUpperLeftY`, `getWidth` and `getHeight` that calculate the upper-left *x*-coordinate, upper-left *y*-coordinate, width and height, respectively, based on the *x1*, *y1*, *x2*, *y2* coordinates. The upper-left *x*-coordinate is the smaller of the two *x*-coordinate values, the upper-left *y*-coordinate is the smaller of the two *y*-coordinate values, the width is the absolute value of the difference between the *x*-coordinate values, and the height is the absolute value of the difference between the *y*-coordinate values. (Declaring classes `MyRectangle` and `MyOval` as described here will make it easier for you to support drawing with the mouse in Exercise 13.9 after you learn about mouse event handling in Section 13.3 .)

The app's GUI should provide separate `Button`s that draw 100 random `MyRectangle`s and 100 random `MyOval`s, respectively. Each `Button` should have a corresponding event handler in the app's controller.

In addition, modify classes `MyLine`, `MyRectangle` and `MyOval` to include the following capabilities that you'll use in Section 10.14 :

1. A constructor with no arguments that sets the shape's coordinates to `0`, the stroke color and—for classes `MyRectangle` and `MyOval` only—the `filled` flag to `false` and the fill color to `Color.BLACK`.
2. *Set* methods for the instance variables in each class. The methods that *set* a coordinate value should verify that the argument is greater than or equal to zero before setting the coordinate—if it's not, they should set the coordinate to zero.
3. *Get* methods for the instance variables in each class. Method `draw` should reference the coordinates by the *get* methods rather than access them directly.