6.13 (Optional) GUI and Graphics Case Study: Colors and Filled Shapes

Class `GraphicsContext` provides many more capabilities than drawing lines, rectangles and ovals. The next two features we introduce are colors and filled shapes. Adding color enriches the drawings a user sees on the computer screen. For example, IDEs generally "syntax color" Java code to help you distinguish different code elements, like keywords and comments. Shapes can be filled with solid colors. As we show in Chapter 22, shapes also can be filled with images or with so-called gradients that transition gradually between colors. In this section, we'll create the `DrawSmiley` app that uses solid colors and filled shapes to draw a smiley face.

Solid colors displayed on computer screens are defined by their *red*, *green*, and *blue* components (called **RGB values**) that each have an integer value from 0 to 255, inclusive. The higher the value of a component color, the richer that color's shade will be. Class `Color` (package `javafx.scene.paint`) represents colors using their RGB values. For convenience, class `Color` contains dozens of predefined `static` `Color` objects that each can be accessed via the class name and a dot (`.`), as in `Color.RED`. For a complete list of the predefined colors see class `Color`'s

documentation at

```
https://docs.oracle.com/javase/8/javafx/api/javafx/sc
```

Class `Color` also provides methods that enable you to create custom `Color`s, though we do not use them in this example.

Creating the Draw Smiley App's GUI

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

- We specified `DrawSmileyController` as the app's **Controller class** in the Scene Builder **Document** window's **Controller** section.
- We set the `Button`'s **Text** property to `Draw Smiley` (in the **Inspector**'s **Properties** section) and set the `Button`'s **On Action** event handler to `drawSmileyButton-Pressed` (in the **Inspector**'s **Code** section).

Class `DrawSmileyController`

Again, we do not show the code for `DrawSmiley.java`, because the only changes from earlier examples are the name of the FXML file to load (`DrawSmiley.fxml`) and the text

displayed in the stage's title bar (`Draw Smiley`). Class `DrawSmileyController` (Fig. 6.11) performs the drawing. `GraphicsContext` methods `fillRect` and `fillOval` draw filled rectangles and ovals, respectively. These have the same parameters as `strokeRect` and `strokeOval` (Section 5.11); the first two are the coordinates for the shape's *upper-left corner*, while the next two determine the *width* and *height*—recall that for an oval these specify the bounding rectangle in which the oval is displayed.

```
 1 // Fig. 6.11: DrawSmileyController.java
 2 // Drawing a smiley face using colors and filled s
 3 import javafx.event.ActionEvent;
 4 import javafx.fxml.FXML;
 5 import javafx.scene.canvas.Canvas;
 6 import javafx.scene.canvas.GraphicsContext;
 7 import javafx.scene.paint.Color;
 8
 9 public class DrawSmileyController {
10    @FXML private Canvas canvas;
11
12    // draws a smiley face
13    @FXML
14    void drawSmileyButtonPressed(ActionEvent event)
15       // get the GraphicsContext, which is used to
16       GraphicsContext gc = canvas.getGraphicsConte
17
18       // draw the face
19       gc.setFill(Color.YELLOW);
20       gc.fillOval(10, 10, 280, 280);
21       gc.strokeOval(10, 10, 280, 280);
22
23       // draw the eyes
24       gc.setFill(Color.BLACK);
25       gc.fillOval(75, 85, 40, 40);
26       gc.fillOval(185, 85, 40, 40);
```

```
27
28        // draw the mouth
29        gc.fillOval(50, 130, 200, 120);
30
31        // "touch up" the mouth into a smile
32        gc.setFill(Color.YELLOW);
33        gc.fillRect(50, 130, 200, 60);
34        gc.fillOval(50, 140, 200, 90);
35     }
36 }
```

Fig. 6.11

Drawing a smiley face using colors and filled shapes.

Description

Method `drawSmileyButtonPressed` draws the smiley

face when the user presses the **Draw Smiley** `Button`. Line 19 uses `GraphicsContext` method `setFill` to set the current fill color to the predefined `Color.YELLOW`—all filled shapes will use this fill until you change it with a subsequent call to `setFill`.

Line 20 draws a filled circle with diameter 280 to represent the face—when the width and height arguments are identical, method `fillOval` draws a circle. Line 21 uses `strokeOval` with the same arguments as `fillOval` to outline the face with a black line for better contrast with the window's background. Note that the `GraphicsContext` separately maintains the fill color and stroke color—as you've seen, the default stroke color is black. You can change the stroke color with `GraphicsContext` method `setStroke`. Next, line 24 sets the fill color to `Color.BLACK`, and lines 25–26 draw two filled circles for the eyes.

Line 29 draws the mouth as an oval, but this is not quite what we want. To create a happy face, we'll touch up the mouth. Line 32 sets the fill color to `Color.YELLOW`, so any shapes we draw will blend in with the face. Line 33 draws a rectangle that's half the mouth's height. This erases the top half of the mouth, leaving just the bottom half. To create a better smile, line 34 draws another oval to slightly cover the upper portion of the mouth.

GUI and Graphics Case

Study Exercises

6.1 Using method `fillOval`, draw a bull's-eye that alternates between two random colors, as in Fig. 6.12. To generate a random color, use the following method call in which `red`, `green` and `blue` are random values in the range 0–255

```
Color.rgb(red, green, blue)
```

Fig. 6.12

A bulls-eye with two alternating, random colors.

Description

6.2 Create a program that draws 10 random filled shapes in random colors, positions and sizes (Fig. 6.13). The event

handler for the **Draw Shapes** `Button` should contain a loop that iterates 10 times. In each iteration, the loop should use a random number to determine whether to draw a filled rectangle or an oval, create a random color and choose the shape's coordinates and dimensions at random. The coordinates should be chosen based on the `Canvas`'s width and height. Lengths of sides should be limited to half the width or height of the window.

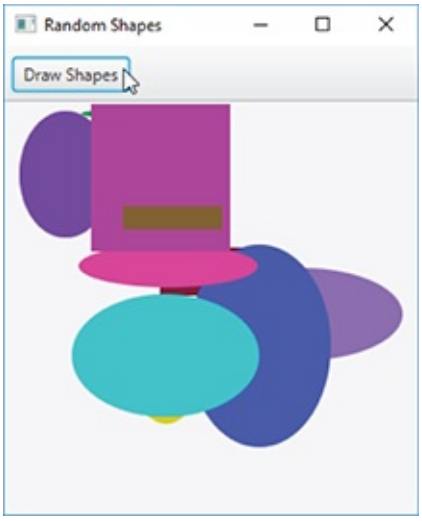

Fig. 6.13

Randomly generated shapes.

Description

6.3 Enhance Exercise 6.2 to randomly choose bordered or filled shapes.

6.4 Enhance the **Draw Smiley** app by using white rectangles to draw teeth and a pink oval to make a tongue. Use the

`Color` objects `Color.WHITE` and `Color.PINK`.

6.5 Enhance the **Draw Smiley** app by drawing the eyes as white circles containing pupils drawn as brown circles (`Color.BROWN`).

6.6 Enhance the **Draw Smiley** app to make the eyes blink each time the user clicks a `Button`.