**Section 13.1 Introduction**

• Java’s coordinate system (p. [556](http://proquest.safaribooksonline.com/9780133813036/ch13_html#page_556)) is a scheme for identifying every point (p. [567](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_567)) on the screen.

• A coordinate pair (p. [556](http://proquest.safaribooksonline.com/9780133813036/ch13_html#page_556)) has an *x*-coordinate (horizontal) and a *y*-coordinate (vertical).

• Coordinates are used to indicate where graphics should be displayed on a screen.

• Coordinate units are measured in pixels (p. [556](http://proquest.safaribooksonline.com/9780133813036/ch13_html#page_556)). A pixel is a display monitor’s smallest unit of resolution.

#### Section 13.2 Graphics Contexts and Graphics Objects

• A Java graphics context (p. [558](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec1_html#page_558)) enables drawing on the screen.

• Class Graphics (p. [558](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec1_html#page_558)) contains methods for drawing strings, lines, rectangles and other shapes. Methods are also included for font manipulation and color manipulation.

• A Graphics object manages a graphics context and draws pixels on the screen that represent text and other graphical objects, e.g., lines, ellipses, rectangles and other polygons (p. [558](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec1_html#page_558)).

• Class Graphics is an abstract class. Each Java implementation has a Graphics subclass that provides drawing capabilities. This implementation is hidden from us by class Graphics, which supplies the interface that enables us to use graphics in a platform-independent manner.

• Method paintComponent can be used to draw graphics in any JComponent component.

• Method paintComponent receives a Graphics object that is passed to the method by the system when a lightweight Swing component needs to be repainted.

• When an application executes, the application container calls method paintComponent. For paintComponent to be called again, an event must occur.

• When a JComponent is displayed, its paintComponentmethod is called.

• Calling method repaint (p. [559](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec2_html#page_559)) on a component updates the graphics drawn on that component.

#### Section 13.3 Color Control

• Class Color (p. [559](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec2_html#page_559)) declares methods and constants for manipulating colors in a Java program.

• Every color is created from a red, a green and a blue component. Together these components are called RGB values (p. [560](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec3_html#page_560)). The RGB components specify the amount of red, green and blue in a color, respectively. The larger the value, the greater the amount of that particular color.

• Color methods getRed, getGreen and getBlue (p.[560](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec3_html#page_560)) return int values from 0 to 255 representing the amount of red, green and blue, respectively.

• Graphics method getColor (p. [560](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec3_html#page_560)) returns a Colorobject with the current drawing color.

• Graphics method setColor (p. [560](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec3_html#page_560)) sets the current drawing color.

• Graphics method fillRect (p. [560](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec3_html#page_560)) draws a rectangle filled by the Graphics object’s current color.

• Graphics method drawString (p. [562](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec1_html#page_562)) draws a Stringin the current color.

• The JColorChooser GUI component (p. [563](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec1_html#page_563)) enables application users to select colors.

• JColorChooser static method showDialog (p. [564](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec1_html#page_564)) displays a modal JColorChooser dialog.

#### Section 13.4 Manipulating Fonts

• Class Font (p. [566](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec1_html#page_566)) contains methods and constants for manipulating fonts.

• Class Font’s constructor takes three arguments—the font name (p. [567](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_567)), font style and font size.

• A Font’s font style can be Font.PLAIN, Font.ITALIC orFont.BOLD (each is a static field of class Font). Font styles can be used in combination (e.g., Font.ITALIC + Font.BOLD).

• The font size is measured in points. A point is 1/72 of an inch.

• Graphics method setFont (p. [567](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_567)) sets the drawing font in which text will be displayed.

• Font method getSize (p. [567](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_567)) returns the font size in points.

• Font method getName (p. [567](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_567)) returns the current font name as a string.

• Font method getStyle (p. [569](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_569)) returns an integer value representing the current Font’s style.

• Font method getFamily (p. [569](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_569)) returns the name of the font family to which the current font belongs. The name of the font family is platform specific.

• Class FontMetrics (p. [569](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_569)) contains methods for obtaining font information.

• Font metrics (p. [569](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec4_html#page_569)) include height, descent and leading.

#### Section 13.5 Drawing Lines, Rectangles and Ovals

• Graphics methods fillRoundRect (p. [573](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_573)) anddrawRoundRect (p. [573](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_573)) draw rectangles with rounded corners.

• Graphics methods draw3DRect (p. [575](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_575)) andfill3DRect (p. [575](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_575)) draw three-dimensional rectangles.

• Graphics methods drawOval (p. [575](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_575)) and fillOval(p. [575](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_575)) draw ovals.

#### Section 13.6 Drawing Arcs

• An arc (p. [575](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_575)) is drawn as a portion of an oval.

• Arcs sweep from a starting angle by the number of degrees specified by their arc angle (p. [575](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_575)).

• Graphics methods drawArc (p. [575](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_575)) and fillArc (p.[575](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec5_html#page_575)) are used for drawing arcs.

#### Section 13.7 Drawing Polygons and Polylines

• Class Polygon contains methods for creating polygons.

• Polygons are closed multisided shapes composed of straight-line segments.

• Polylines (p. [578](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec7_html#page_578)) are sequences of connected points.

• Graphics method drawPolyline (p. [580](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec7_html#page_580)) displays a series of connected lines.

• Graphics methods drawPolygon (p. [580](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec7_html#page_580)) andfillPolygon (p. [581](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec7_html#page_581)) are used to draw polygons.

• Polygon method addPoint (p. [581](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec7_html#page_581)) adds pairs of x- and y-coordinates to the Polygon.

#### Section 13.8 Java 2D API

• The Java 2D API (p. [581](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec7_html#page_581)) provides advanced two-dimensional graphics capabilities.

• Class Graphics2D (p. [581](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec7_html#page_581))—a subclass of Graphics—is used for drawing with the Java 2D API.

• The Java 2D API’s classes for drawing shapes includeLine2D.Double, Rectangle2D.Double,RoundRectangle2D.Double, Arc2D.Double andEllipse2D.Double (p. [581](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec7_html#page_581)).

• Class GradientPaint (p. [584](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec4_html#page_584)) helps draw a shape in gradually changing colors—called a gradient (p. [584](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec4_html#page_584)).

• Graphics2D method fill (p. [584](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec4_html#page_584)) draws a filled object of any type that implements interface Shape (p. [584](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec4_html#page_584)).

• Class BasicStroke (p. [584](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec4_html#page_584)) helps specify the drawing characteristics of lines.

• Graphics2D method draw (p. [584](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec4_html#page_584)) is used to draw aShape object.

• Classes GradientPaint (p. [584](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec4_html#page_584)) and TexturePaint (p.[585](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec6_html#page_585)) help specify the characteristics for filling shapes with colors or patterns.

• A general path (p. [586](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec8_html#page_586)) is a shape constructed from straight lines and complex curves and is represented with an object of class GeneralPath (p. [586](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec8_html#page_586)).

• GeneralPath method moveTo (p. [587](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec9_html#page_587)) specifies the first point in a general path.

• GeneralPath method lineTo (p. [588](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec9_html#page_588)) draws a line to the next point in the path. Each new call to lineTodraws a line from the previous point to the current point.

• GeneralPath method closePath (p. [588](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec9_html#page_588)) draws a line from the last point to the point specified in the last call to moveTo. This completes the general path.

• Graphics2D method translate (p. [588](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec9_html#page_588)) is used to move the drawing origin to a new location.

• Graphics2D method rotate (p. [588](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec9_html#page_588)) is used to rotate the next displayed shape.

### Self-Review Exercises

[**13.1**](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec12_html#ch13ans1) Fill in the blanks in each of the following statements:

a) In Java 2D, method \_\_\_\_\_\_\_\_\_ of class \_\_\_\_\_\_\_\_\_\_ sets the characteristics of a stroke used to draw a shape.

b) Class \_\_\_\_\_\_\_\_\_\_ helps specify the fill for a shape such that the fill gradually changes from one color to another.

c) The \_\_\_\_\_\_\_\_\_ method of class Graphics draws a line between two points.

d) RGB is short for \_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_.

e) Font sizes are measured in units called \_\_\_\_\_\_\_\_\_.

f) Class \_\_\_\_\_\_\_\_\_ helps specify the fill for a shape using a pattern drawn in a BufferedImage.

[**13.2**](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec12_html#ch13ans2) State whether each of the following is true or false. Iffalse, explain why.

a) The first two arguments of Graphics methoddrawOval specify the center coordinate of the oval.

b) In the Java coordinate system, x-coordinates increase from left to right and y-coordinates from top to bottom.

c) Graphics method fillPolygon draws a filled polygon in the current color.

d) Graphics method drawArc allows negative angles.

e) Graphics method getSize returns the size of the current font in centimeters.

f) Pixel coordinate (0, 0) is located at the exact center of the monitor.

[**13.3**](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec12_html#ch13ans3) Find the error(s) in each of the following and explain how to correct them. Assume that g is a Graphics object.

a) g.setFont("SansSerif");

b) g.erase(x, y, w, h); // clear rectangle at (x, y)

c) Font f = new Font("Serif", Font.BOLDITALIC,12);

d) g.setColor(255, 255, 0); // change color to yellow

### Answers to Self-Review Exercises

[**13.1**](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec11_html#ch13que1)

a) setStroke, Graphics2D.

b) GradientPaint.

c) drawLine.

d) red, green, blue.

e) points.

f) TexturePaint.

[**13.2**](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec11_html#ch13que2)

a) False. The first two arguments specify the upper-left corner of the bounding rectangle.

b) True.

c) True.

d) True.

e) False. Font sizes are measured in points.

f) False. The coordinate (0,0) corresponds to the upper-left corner of a GUI component on which drawing occurs.

[**13.3**](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec11_html#ch13que3)

a) The setFont method takes a Font object as an argument—not a String.

b) The Graphics class does not have an erase method. The clearRect method should be used.

c) Font.BOLDITALIC is not a valid font style. To get a bold italic font, use Font.BOLD + Font.ITALIC.

d) Method setColor takes a Color object as an argument, not three integers.

### Exercises

**13.4** Fill in the blanks in each of the following statements:

a) Class \_\_\_\_\_\_\_\_\_ of the Java 2D API is used to draw ovals.

b) Methods draw and fill of class Graphics2D require an object of type \_\_\_\_\_\_\_\_\_ as their argument.

c) The three constants that specify font style are \_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_.

d) Graphics2D method \_\_\_\_\_\_\_\_\_ sets the painting color for Java 2D shapes.

**13.5** State whether each of the following is true or false. Iffalse, explain why.

a) Graphics method drawPolygon automatically connects the endpoints of the polygon.

b) Graphics method drawLine draws a line between two points.

c) Graphics method fillArc uses degrees to specify the angle.

d) In the Java coordinate system, values on the y-axis increase from left to right.

e) Graphics inherits directly from class Object.

f) Graphics is an abstract class.

g) The Font class inherits directly from class Graphics.

**13.6 (Concentric Circles Using Method** ***drawArc*)** Write an application that draws a series of eight concentric circles. The circles should be separated by 10 pixels. Use Graphicsmethod drawArc.

**13.7 (Concentric Circles Using Class** ***Ellipse2D.Double*)**Modify your solution to [Exercise 13.6](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que6) to draw the ovals by using class Ellipse2D.Double and method draw of classGraphics2D.

**13.8 (Random Lines Using Class** ***Line2D.Double*)** Modify your solution to [Exercise 13.7](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que7) to draw random lines in random colors and random thicknesses. Use classLine2D.Double and method draw of class Graphics2D to draw the lines.

**13.9 (Random Triangles)** Write an application that displays randomly generated triangles in different colors. Each triangle should be filled with a different color. Use classGeneralPath and method fill of class Graphics2D to draw the triangles.

**13.10 (Random Characters)** Write an application that randomly draws characters in different fonts, sizes and colors.

**13.11 (Grid Using Method** ***drawLine*)** Write an application that draws an 8-by-8 grid. Use Graphics method drawLine.

**13.12 (Grid Using Class** ***Line2D.Double*)** Modify your solution to [Exercise 13.11](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que11) to draw the grid using instances of class Line2D.Double and method draw of class Graphics2D.

**13.13 (Grid Using Method** ***drawRect*)** Write an application that draws a 10-by-10 grid. Use the Graphics methoddrawRect.

**13.14 (Grid Using Class** ***Rectangle2D.Double*)** Modify your solution to [Exercise 13.13](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que13) to draw the grid by using class Rectangle2D.Double and method draw of classGraphics2D.

**13.15 (Drawing Tetrahedrons)** Write an application that draws a tetrahedron (a three-dimensional shape with four triangular faces). Use class GeneralPath and method draw of class Graphics2D.

**13.16 (Drawing Cubes)** Write an application that draws a cube. Use class GeneralPath and method draw of classGraphics2D.

**13.17 (Circles Using Class** ***Ellipse2D.Double*)** Write an application that asks the user to input the radius of a circle as a floating-point number and draws the circle, as well as the values of the circle’s diameter, circumference and area. Use the value 3.14159 for π. [Note: You may also use the predefined constant Math.PI for the value of π. This constant is more precise than the value 3.14159. Class Math is declared in the java.lang package, so you need not importit.] Use the following formulas (r is the radius):

diameter*=* 2r  
circumference*=* 2πr  
area *=* πr2

The user should also be prompted for a set of coordinates in addition to the radius. Then draw the circle and display its diameter, circumference and area, using anEllipse2D.Double object to represent the circle and methoddraw of class Graphics2D to display it.

**13.18 (Screen Saver)** Write an application that simulates a screen saver. The application should randomly draw lines using method drawLine of class Graphics. After drawing 100 lines, the application should clear itself and start drawing lines again. To allow the program to draw continuously, place a call to repaint as the last line in method paintComponent. Do you notice any problems with this on your system?

**13.19 (Screen Saver Using** ***Timer*)** Package javax.swingcontains a class called Timer that is capable of calling method actionPerformed of interface ActionListener at a fixed time interval (specified in milliseconds). Modify your solution to [Exercise 13.18](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que18) to remove the call to repaintfrom method paintComponent. Declare your class to implement ActionListener. (The actionPerformed method should simply call repaint.) Declare an instance variable of type Timer called timer in your class. In the constructor for your class, write the following statements:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p593pro01a)

timer = new Timer(1000, this);  
timer.start();

This creates an instance of class Timer that will call thisobject’s actionPerformed method every 1000 milliseconds (i.e., every second).

**13.20 (Screen Saver for a Random Number of Lines)**Modify your solution to [Exercise 13.19](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que19) to enable the user to enter the number of random lines that should be drawn before the application clears itself and starts drawing lines again. Use a JTextField to obtain the value. The user should be able to type a new number into the JTextField at any time during the program’s execution. Use an inner class to perform event handling for the JTextField.

**13.21 (Screen Saver with Shapes)** Modify your solution to [Exercise 13.20](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que20) such that it uses random-number generation to choose different shapes to display. Use methods of class Graphics.

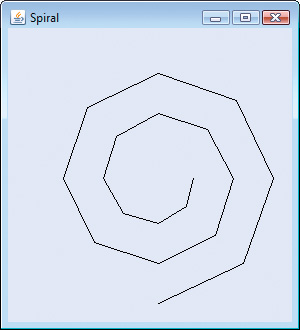
**13.22 (Screen Saver Using the Java 2D API)** Modify your solution to [Exercise 13.21](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que21) to use classes and drawing capabilities of the Java 2D API. Draw shapes like rectangles and ellipses, with randomly generated gradients. Use classGradientPaint to generate the gradient.

**13.23 (Turtle Graphics)** Modify your solution to [Exercise 7.21](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que21)—Turtle Graphics—to add a graphical user interface using JTextFields and JButtons. Draw lines rather than asterisks (\*). When the turtle graphics program specifies a move, translate the number of positions into a number of pixels on the screen by multiplying the number of positions by 10 (or any value you choose). Implement the drawing with Java 2D API features.

**13.24 (Knight’s Tour)** Produce a graphical version of the Knight’s Tour problem ([Exercise 7.22](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que22), [Exercise 7.23](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que23) and[Exercise 7.26](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que26)). As each move is made, the appropriate cell of the chessboard should be updated with the proper move number. If the result of the program is a full tour or a closed tour, the program should display an appropriate message. If you like, use class Timer (see [Exercise 13.19](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que19)) to help animate the Knight’s Tour.

**13.25 (Tortoise and Hare)** Produce a graphical version of the Tortoise and Hare simulation ([Exercise 7.28](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que28)). Simulate the mountain by drawing an arc that extends from the bottom-left corner of the window to the top-right corner. The tortoise and the hare should race up the mountain. Implement the graphical output to actually print the tortoise and the hare on the arc for every move. [Hint: Extend the length of the race from 70 to 300 to allow yourself a larger graphics area.]

**13.26 (Drawing Spirals)** Write an application that usesGraphics method drawPolyline to draw a spiral similar to the one shown in [Fig. 13.33](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13fig33).



**Fig. 13.33** | Spiral drawn using method drawPolyline.

**13.27 (Pie Chart)** Write a program that inputs four numbers and graphs them as a pie chart. Use classArc2D.Double and method fill of class Graphics2D to perform the drawing. Draw each piece of the pie in a separate color.

**13.28 (Selecting Shapes)** Write an application that allows the user to select a shape from a JComboBox and draws it 20 times with random locations and dimensions in methodpaintComponent. The first item in the JComboBox should be the default shape that is displayed the first timepaintComponent is called.

**13.29 (Random Colors)** Modify [Exercise 13.28](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que28) to draw each of the 20 randomly sized shapes in a randomly selected color. Use all 13 predefined Color objects in an array ofColors.

**13.30 (*JColorChooser*** **Dialog)** Modify [Exercise 13.28](http://proquest.safaribooksonline.com/9780133813036/ch13lev1sec13_html#ch13que28) to allow the user to select the color in which shapes should be drawn from a JColorChooser dialog.

**(Optional) GUI and Graphics Case Study Exercise: Adding Java 2D**

**13.31** Java 2D introduces many new capabilities for creating unique and impressive graphics. We’ll add a small subset of these features to the drawing application you created in[Exercise 12.17](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec62_html#ch12que17). In this version, you’ll enable the user to specify gradients for filling shapes and to change stroke characteristics for drawing lines and outlines of shapes. The user will be able to choose which colors compose the gradient and set the width and dash length of the stroke.

First, you must update the MyShape hierarchy to support Java 2D functionality. Make the following changes in classMyShape:

a) Change abstract method draw’s parameter type from Graphics to Graphics2D.

b) Change all variables of type Color to type Paint to enable support for gradients. [*Note:* Recall that classColor implements interface Paint.]

c) Add an instance variable of type Stroke in classMyShape and a Stroke parameter in the constructor to initialize the new instance variable. The default stroke should be an instance of class BasicStroke.

Classes MyLine, MyBoundedShape, MyOval andMyRectangle should each add a Stroke parameter to their constructors. In the draw methods, each shape should set the Paint and the Stroke before drawing or filling a shape. Since Graphics2D is a subclass of Graphics, we can continue to use Graphics methods drawLine, drawOval, fillOval, and so on to draw the shapes. When these methods are called, they’ll draw the appropriate shape using the specifiedPaint and Stroke settings.

Next, you’ll update the DrawPanel to handle the Java 2D features. Change all Color variables to Paint variables. Declare an instance variable currentStroke of type Strokeand provide a *set* method for it. Update the calls to the individual shape constructors to include the Paint andStroke arguments. In method paintComponent, cast theGraphics reference to type Graphics2D and use theGraphics2D reference in each call to MyShape method draw.

Next, make the new Java 2D features accessible from the GUI. Create a JPanel of GUI components for setting the Java 2D options. Add these components at the top of theDrawFrame below the panel that currently contains the standard shape controls (see [Fig. 13.34](http://proquest.safaribooksonline.com/9780133813036/ch13lev2sec18_html#ch13fig34)). These GUI components should include:

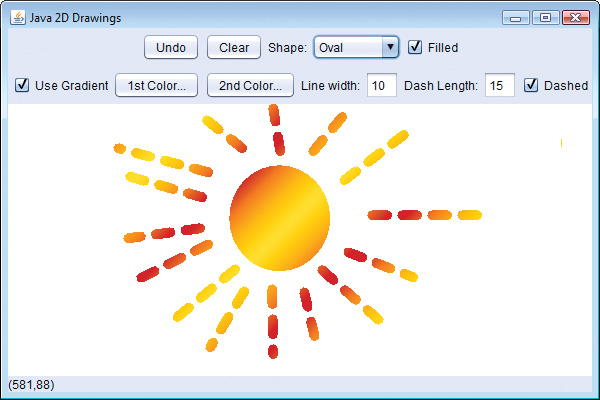
a) A checkbox to specify whether to paint using a gradient.

b) Two JButtons that each show a JColorChooserdialog to allow the user to choose the first and second color in the gradient. (These will replace theJComboBox used for choosing the color in [Exercise 12.17](http://proquest.safaribooksonline.com/9780133813036/ch12lev2sec62_html#ch12que17).)

c) A text field for entering the Stroke width.

d) A text field for entering the Stroke dash length.

e) A checkbox for selecting whether to draw a dashed or solid line.



**Fig. 13.34** | Drawing with Java 2D.

If the user selects to draw with a gradient, set the Painton the DrawPanel to be a gradient of the two colors chosen by the user. The expression

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0595pro01a)

new GradientPaint(0, 0, color1, 50, 50, color2, true))

creates a GradientPaint that cycles diagonally from the upper-left to the bottom-right every 50 pixels. Variablescolor1 and color2 represent the colors chosen by the user. If the user does not select to use a gradient, then simply set the Paint on the DrawPanel to be the first Color chosen by the user.

For strokes, if the user chooses a solid line, then create theStroke with the expression

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0595pro02a)

new BasicStroke(width,BasicStroke.CAP\_ROUND,BasicStroke.JOIN\_ROUND)

where variable width is the width specified by the user in the line-width text field. If the user chooses a dashed line, then create the Stroke with the expression

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0595pro03a)

new BasicStroke(width,BasicStroke.CAP\_ROUND,BasicStroke.JOIN\_ROUND,  
   10, dashes, 0)

where width again is the width in the line-width field, anddashes is an array with one element whose value is the length specified in the dash-length field. The Panel andStroke objects should be passed to the shape object’s constructor when the shape is created in DrawPanel.

### Making a Difference

**13.32****(Large-Type Displays for People with Low Vision)** The accessibility of computers and the Internet to all people, regardless of disabilities, is becoming more important as these tools play increasing roles in our personal and business lives. According to a recent estimate by the World Health Organization ([www.who.int/mediacentre/factsheets/fs282/en/](http://www.who.int/mediacentre/factsheets/fs282/en/)), 246 million people worldwide have low vision. To learn more about low vision, check out the GUI-based low-vision simulation atwww.webaim.org/simulations/lowvision.php. People with low vision might prefer to choose a font and/or a larger font size when reading electronic documents and web pages. Java has five built-in “logical” fonts that are guaranteed to be available in any Java implementation, including Serif, Sans-serif and Monospaced. Write a GUI application that provides a JTextArea in which the user can type text. Allow the user to select Serif, Sans-serif or Monospaced from aJComboBox. Provide a **Bold** JCheckBox, which, if checked, makes the text bold. Include **Increase Font Size** and **DecreaseFont Size** JButtons that allow the user to scale the size of the font up or down, respectively, by one point at a time. Start with a font size of 18 points. For the purposes of this exercise, set the font size on the JComboBox, JButtons andJCheckBox to 20 points so that a person with low vision will be able to read the text on them.