12

Graphics and Java 2D™

One picture is worth ten thousand words.
—Chinese proverb

Treat nature in terms of the cylinder, the sphere, the cone, all in perspective.
—Paul Cézanne

Colors, like features, follow the changes of the emotions.
—Pablo Picasso

Nothing ever becomes real till it is experienced—even a proverb is no proverb to you till your life has illustrated it.
—John Keats

OBJECTIVES

In this chapter you will learn:

- To understand graphics contexts and graphics objects.
- To manipulate colors.
- To manipulate fonts.
- To use methods of class `Graphics` to draw lines, rectangles, rectangles with rounded corners, three-dimensional rectangles, ovals, arcs and polygons.
- To use methods of class `Graphics2D` from the `Java 2D` API to draw lines, rectangles, rectangles with rounded corners, ellipses, arcs and general paths.
- To specify `Paint` and `Stroke` characteristics of shapes displayed with `Graphics2D`.

Self-Review Exercises

12.1 Fill in the blanks in each of the following statements:

a) In Java 2D, method _____ of class _____ sets the characteristics of a line used to draw a shape.

ANS: `setStroke`, `Graphics2D`

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

ANS: `GradientPaint`

c) The _____ method of class `Graphics` draws a line between two points.

ANS: `drawLine`

d) RGB is short for _____, _____ and _____.

ANS: red, green, blue

e) Font sizes are measured in units called _____.

ANS: points

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

ANS: `TexturePaint`

12.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The first two arguments of `Graphics` method `drawOval` specify the center coordinate of the oval.

ANS: False. The first two arguments specify the upper-left corner of the bounding rectangle.

b) In the Java coordinate system, *x*-values increase from left to right.

ANS: True.

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

ANS: True.

d) `Graphics` method `drawArc` allows negative angles.

ANS: True.

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

ANS: False. Font sizes are measured in points.

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

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

12.3 Find the error(s) in each of the following and explain how to correct the error(s). Assume that `g` is a `Graphics` object.

a)
```
g.setFont( "SansSerif" );
```
ANS: The `setFont` method takes a `Font` object as an argument—not a `String`.

b)
```
g.erase( x, y, w, h );    // clear rectangle at (x, y)
```
ANS: The `Graphics` class does not have an `erase` method. The `clearRect` method should be used.

c)
```
Font f = new Font( "Serif", Font.BOLDITALIC, 12 );
```
ANS: `Font.BOLDITALIC` is not a valid font style. To get a bold italic font, use `Font.BOLD + Font.ITALIC`.

d)
```
g.setColor( 255, 255, 0 );  // change color to yellow
```
ANS: Method `setColor` takes a `Color` object as an argument, not three integers.

Exercises

12.4 Fill in the blanks in each of the following statements:

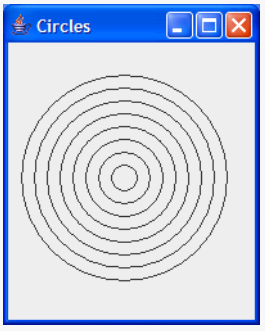
a) Class _____ of the Java 2D API is used to draw ovals.
ANS: `Ellipse2D`.
b) Methods `draw` and `fill` of class `Graphics2D` require an object of type _____ as their argument.
ANS: `Shape`.
c) The three constants that specify font style are _____, _____ and _____.
ANS: `Font.PLAIN`, `Font.BOLD` and `Font.ITALIC`.
d) `Graphics2D` method _____ sets the painting color for Java 2D shapes.
ANS: `setColor`.

12.5 State whether each of the following is *true* or *false*. If *false*, explain why.
a) `Graphics` method `drawPolygon` automatically connects the endpoints of the polygon.
ANS: True.
b) `Graphics` method `drawLine` draws a line between two points.
ANS: True.
c) `Graphics` method `fillArc` uses degrees to specify the angle.
ANS: True.
d) In the Java coordinate system, values on the *y*-axis increase from left to right.
ANS: False. In the Java coordinate system, values on the *y*-axis increase from top to bottom.
e) `Graphics` inherits directly from class `Object`.
ANS: True.
f) `Graphics` is an `abstract` class.
ANS: True.
g) The `Font` class inherits directly from class `Graphics`.
ANS: False. Class `Font` inherits directly from class `Object`.

12.6 *(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 `Graphics` method `drawArc`.
ANS:

```
1  // Exercise 12.6 Solution: CirclesJPanel.java
2  // This program draws concentric circles
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class CirclesJPanel extends JPanel
7  {
8     // draw eight Circles separated by 10 pixels
9     public void paintComponent( Graphics g )
10    {
11       super.paintComponent( g );
12
13       // create 8 concentric circles
14       for ( int topLeft = 0; topLeft < 80; topLeft += 10 )
15       {
16          int radius = 160 - ( topLeft * 2 );
17          g.drawArc( topLeft + 10, topLeft + 25, radius, radius, 0, 360 );
18       } // end for
19    } // end method paintComponent
20 } // end class CirclesJPanel
```

```
// Exercise 12.6 Solution: Circles.java
// This program draws concentric circles
import javax.swing.JFrame;

public class Circles extends JFrame
{
   public static void main( String args[] )
   {
      // create frame for CirclesJPanel
      JFrame frame = new JFrame( "Circles" );
      frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

      CirclesJPanel circlesJPanel = new CirclesJPanel();
      frame.add( circlesJPanel ); // add circlesJPanel to frame
      frame.setSize( 200, 250 ); // set frame size
      frame.setVisible( true ); // display frame
   } // end main
} // end class Circles
```



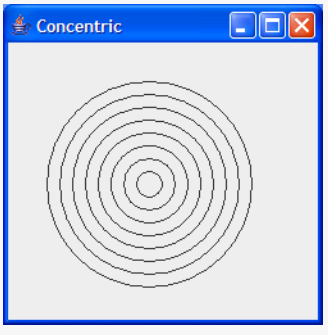
12.7 *(Concentric Circles Using Class `Ellipse2D.Double`)* Modify your solution to Exercise 12.6 to draw the ovals by using class `Ellipse2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
// Exercise 12.7 Solution: CirclesJPanel.java
// This program draws concentric circles using Graphics2D
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import javax.swing.JPanel;

public class CirclesJPanel extends JPanel
{
   // draw eight concentric circles separated by 10 pixels
   public void paintComponent( Graphics g )
```

```
12      {
13         super.paintComponent( g );
14
15         // create 2D by casting g to Graphics 2D
16         Graphics2D g2d = ( Graphics2D ) g;
17
18         for ( int x = 0; x < 80; x += 10 )
19         {
20            int y = 160 - ( x * 2 );
21            g2d.draw( new Ellipse2D.Double( x + 30, x + 30, y, y ) );
22         } // end for
23      } // end method paintComponent
24   } // end class CirclesJPanel
```

```
1   // Exercise 12.7 Solution: Concentric.java
2   // This program draws concentric circles using Graphics2D
3   import javax.swing.JFrame;
4
5   public class Concentric extends JFrame
6   {
7      public static void main( String args[] )
8      {
9         // create frame for CirclesJPanel
10        JFrame frame = new JFrame( "Concentric" );
11        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13        CirclesJPanel circlesJPanel = new CirclesJPanel();
14        frame.add( circlesJPanel ); // add circlesJPanel to frame
15        frame.setSize( 250, 250 ); // set frame size
16        frame.setVisible( true ); // display frame
17     } // end main
18  } // end class Concentric
```



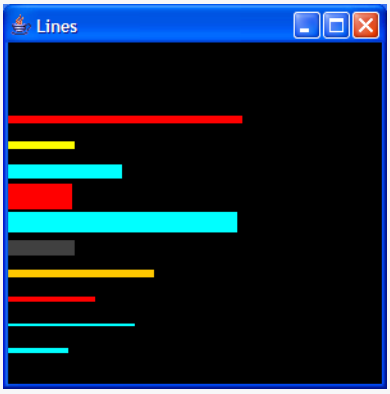
12.8 *(Random Lines Using Class `Line2D.Double`)* Write an application that draws lines of random lengths, colors and thicknesses. Use class `Line2D.Double` and method `draw` of class `Graphics2D` to draw the lines.

ANS:

```
// Exercise 12.8 Solution: LinesJPanel.java
// This program draws lines of different colors
import java.awt.Color;
import java.awt.BasicStroke;
import java.awt.geom.Line2D;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.Random;
import javax.swing.JPanel;

public class LinesJPanel extends JPanel
{
   private Color colors[] = { Color.GREEN, Color.CYAN, Color.YELLOW,
           Color.DARK_GRAY, Color.RED, Color.ORANGE,
           Color.GRAY, Color.PINK, Color.MAGENTA };

   // create 10 lines
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );
      setBackground( Color.BLACK ); // set JPanel background
      Random random = new Random(); // get random number generator

      // create 2D by casting g to Graphics 2D
      Graphics2D g2d = ( Graphics2D ) g;

      for ( int y = 60; y < 250; y += 20 )
      {
         // choose a random color from array
         int color = random.nextInt( 9 );
         g2d.setColor( colors[ color ] );

         // choose a random thickness from 1-20
         int thickness = 1 + random.nextInt( 20 );
         g2d.setStroke( new BasicStroke( thickness ) );

         // choose a random length and draw line
         int x1 = 1 + random.nextInt( 199 );
         g2d.draw( new Line2D.Double( 1, y, x1, y ) );
      } // end for
   } // end method paintComponent
} // end class LinesJPanel
```

```
// Exercise 12.8 Solution: Lines2.java
// This program draws lines of different colors
```

```
3  import java.awt.Color;
4  import javax.swing.JFrame;
5
6  public class Lines2
7  {
8     public static void main( String args[] )
9     {
10       // create frame for LinesJPanel
11       JFrame frame = new JFrame( "Lines" );
12       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14       LinesJPanel linesJPanel = new LinesJPanel();
15       frame.add( linesJPanel ); // add linesJPanel to frame
16       frame.setSize( 300, 300 ); // set frame size
17       frame.setVisible( true ); // display frame
18    } // end main
19 } // end class Lines2
```



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

ANS:

```
1  // Exercise 12.9 Solution: TrianglesJPanel.java
2  // Displays randomly generated triangles in different colors.
3  import java.awt.Color;
4  import java.awt.geom.GeneralPath;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import java.util.Random;
8  import javax.swing.JPanel;
9
10 public class TrianglesJPanel extends JPanel
11 {
```

```
   // draw ten triangles
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );
      setBackground( Color.BLACK ); // set JPanel background color
      Graphics2D g2d = ( Graphics2D ) g; // cast graphics object
      Random random = new Random(); // get random number generator

      // create a triangle from three random points
      for ( int i = 0; i < 10; i++ )
      {
         // create the object which will be the triangle
         GeneralPath triangle = new GeneralPath();

         // use method moveTo to start the triangle
         int x = random.nextInt( 375 ) + 25;
         int y = random.nextInt( 375 ) + 25;
         triangle.moveTo( x, y );

         // draw a line to the second point
         x = random.nextInt( 375 ) + 25;
         y = random.nextInt( 375 )+ 25;
         triangle.lineTo( x, y );

         // draw a line to the third point
         x = random.nextInt( 375 ) + 25;
         y = random.nextInt( 375 ) + 25;
         triangle.lineTo( x, y );

         triangle.closePath(); // draw a line back to the initial point

         // choose a random color
         g2d.setColor( new Color( random.nextInt( 256 ),
            random.nextInt( 256 ), random.nextInt( 256 ) ) );

         g2d.fill( triangle ); // color the interior of the triangle
      } // end for
   } // end method paintComponent
} // end class TrianglesJPanel
```

```
// Exercise 12.9 Solution: Triangles.java
// Displays randomly generated triangles in different colors.
import javax.swing.JFrame;

public class Triangles
{
   public static void main( String args[] )
   {
      // create frame for TrianglesJPanel
      JFrame frame = new JFrame( "Drawing Triangles" );
```

```
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        TrianglesJPanel trianglesJPanel = new TrianglesJPanel();
        frame.add( trianglesJPanel ); // add trianglesJPanel to frame
        frame.setSize( 400, 400 ); // set frame size
        frame.setVisible( true ); // display frame
     } // end main
  } // end class Triangles
```

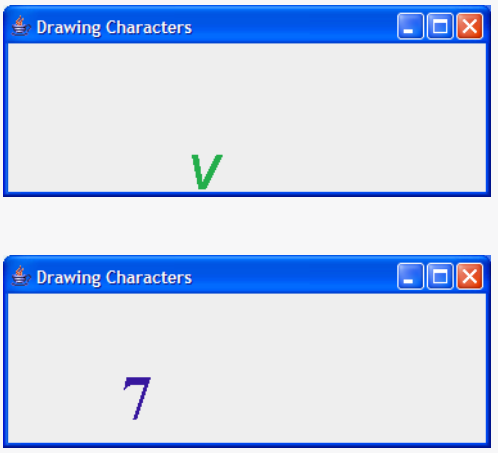
12.10 *(Random Characters)* Write an application that randomly draws characters in different font sizes and colors.

ANS:

```
// Exercise 12.10 Solution: CharactersJPanel.java
// Program randomly draws characters
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.util.Random;
import javax.swing.JPanel;

public class CharactersJPanel extends JPanel
{
   private final int DELAY = 999999;

   // draw characters
   public void paintComponent( Graphics g )
   {
      String fontOptions[] =
```

```
17          { "Serif", "Monospaced", "SansSerif", "Dialog", "DialogInput" };
18       int styleOptions[] =
19          { Font.PLAIN, Font.ITALIC, Font.BOLD, Font.ITALIC + Font.BOLD };
20
21       super.paintComponent( g ); // call superclass's paintComponent
22       Random random = new Random(); // get random number generator
23
24       // randomly choose font name, position and style
25       int fontSize = ( int ) ( 10 + random.nextFloat() * 63 );
26       String fontName = fontOptions[ ( int ) ( random.nextFloat() * 5 ) ];
27       int fontStyle = styleOptions[ ( int ) ( random.nextFloat() * 4 ) ];
28       int x = ( int ) ( random.nextFloat() * 380 );
29       int y = ( int ) ( 50 + random.nextFloat() * 95 );
30       char letters[] = { 'V', 'O', 'L', 'S', '8', '7' };
31
32       // create font from random data
33       Font font = new Font( fontName, fontStyle, fontSize );
34
35       // draw character with random color
36       g.setColor( new Color( ( float ) random.nextFloat(),
37          ( float ) random.nextFloat(), ( float ) random.nextFloat() ) );
38       g.setFont( font );
39       g.drawString( String.valueOf(
40          letters[ ( int ) ( random.nextFloat() * 6 ) ] ), x, y );
41
42       // adding delay is optional the body of the for loop is empty
43       for ( int h = 1; h < DELAY; h++ ) ;
44
45       repaint(); // repaint panel
46    } // end method paintComponent
47 } // end class CharactersJPanel
```

```
1  // Exercise 12.10 Solution: Draw.java
2  // Program randomly draws characters
3  import javax.swing.JFrame;
4
5  public class Draw
6  {
7     public static void main( String args[] )
8     {
9        // create frame for CharactersJPanel
10       JFrame frame = new JFrame( "Drawing Characters" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       CharactersJPanel charactersJPanel = new CharactersJPanel();
14       frame.add( charactersJPanel ); // add charactersJPanel to frame
15       frame.setSize( 380, 150 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Draw
```



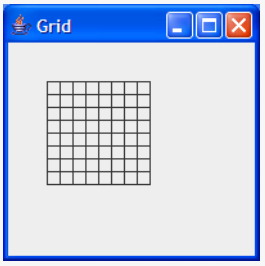
12.11 *(Grid Using Method `drawLine`)* Write an application that draws an 8-by-8 grid. Use `Graphics` method `drawLine`.

ANS:

```
1  // Exercise 12.11 Solution: GridJPanel.java
2  // This program draws an 8 x 8 grid
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class GridJPanel extends JPanel
7  {
8     // draw a grid using the drawLine method
9     public void paintComponent( Graphics g )
10    {
11       super.paintComponent( g );
12
13       int y = 30, x1 = 30;
14
15       for ( int row = 0; row <= 8; row++, y += 10 )
16          g.drawLine( 30, y, 110, y );
17
18       for ( int column = 0; column <= 8; column++, x1 += 10 )
19          g.drawLine( x1, 30, x1, 110 );
20    } // end method paintComponent
21 } // end class GridJPanel
```

```
1  // Exercise 12.11 Solution: Grid1.java
2  // This program draws an 8 x 8 grid
```

```
3  import javax.swing.JFrame;
4
5  public class Grid1
6  {
7     public static void main( String args[] )
8     {
9        // create frame for GridJPanel
10       JFrame frame = new JFrame( "Grid" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       GridJPanel gridJPanel = new GridJPanel();
14       frame.add( gridJPanel ); // add gridJPanel to frame
15       frame.setSize( 200, 200 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Grid1
```



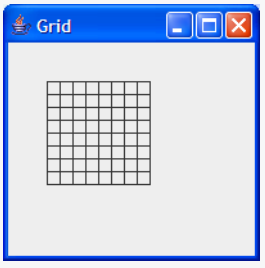
12.12 *(Grid Using Class `Line2D.Double`)* Modify your solution to Exercise 12.11 to draw the grid using instances of class `Line2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
1  // Exercise 12.12 Solution: GridJPanel.java
2  // This program draws an 8 x 8 grid
3  import java.awt.geom.Line2D;
4  import java.awt.Graphics;
5  import java.awt.Graphics2D;
6  import javax.swing.JPanel;
7
8  public class GridJPanel extends JPanel
9  {
10    // draw an 8x8 grid
11    public void paintComponent( Graphics g )
12    {
13       super.paintComponent( g );
14
15       int y = 30, x1 = 30;
16       Graphics2D g2d = ( Graphics2D ) g;
17
18       for ( int row = 0; row <= 8; row++, y += 10 )
```

```
19           g2d.draw( new Line2D.Double( 30, y, 110, y ) );
20
21        for ( int column = 0; column <= 8; column++, x1 += 10 )
22           g2d.draw( new Line2D.Double(  x1, 30, x1, 110 ) );
23     } // end method paintComponent
24  } // end class GridJPanel
```

```
1  // Exercise 12.12 Solution: Grid2.java
2  // This program draws an 8 x 8 grid
3  import javax.swing.JFrame;
4
5  public class Grid2
6  {
7     public static void main( String args[] )
8     {
9        // create frame for GridJPanel
10       JFrame frame = new JFrame( "Grid" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       GridJPanel gridJPanel = new GridJPanel();
14       frame.add( gridJPanel ); // add gridJPanel to frame
15       frame.setSize( 200, 200 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Grid2
```



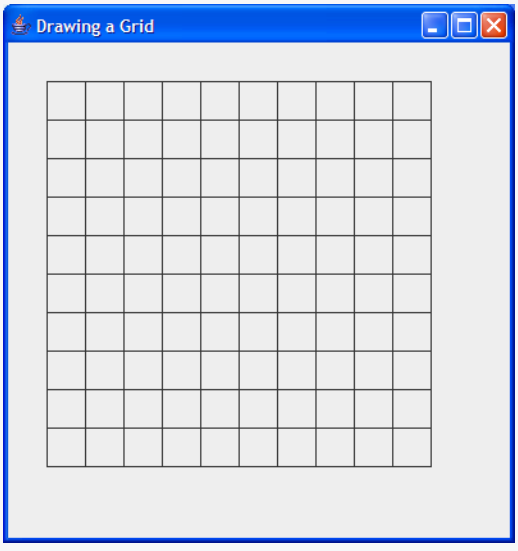
12.13 *(Grid Using Method `drawRect`)* Write an application that draws a 10-by-10 grid. Use the `Graphics` method `drawRect`.

ANS:

```
1  // Exercise 12.13 Solution: GridJPanel.java
2  // Program draws a 10x10 grid using drawRect().
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class GridJPanel extends JPanel
```

```
{
   // draw grid
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g ); // call superclass's paintComponent

      for ( int x = 30; x <= 300; x += 30 )
         for ( int y = 30; y <= 300; y += 30 )
            g.drawRect( x, y, 30, 30 );
   } // end method printComponent
} // end class GridJPanel
```

```
// Exercise 12.13 Solution: Grid.java
// Program draws a 10x10 grid.
import javax.swing.JFrame;

public class Grid
{
   public static void main( String args[] )
   {
      // create frame for GridJPanel
      JFrame frame = new JFrame( "Drawing a Grid" );
      frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

      GridJPanel gridJPanel = new GridJPanel();
      frame.add( gridJPanel ); // add gridJPanel to frame
      frame.setSize( 400, 420 ); // set frame size
      frame.setVisible( true ); // display frame
   } // end main
} // end class Grid
```

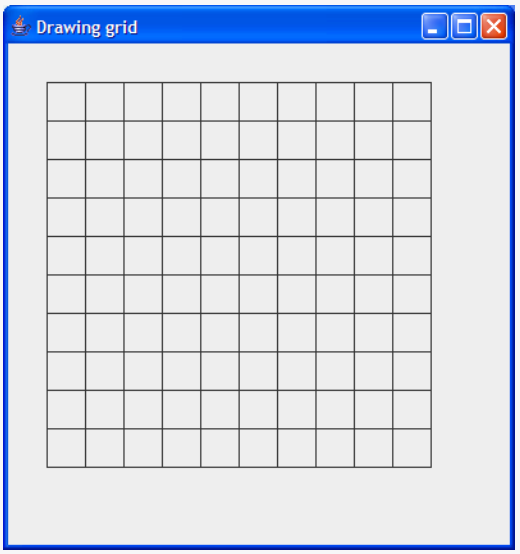


12.14 *(Grid Using Class `Rectangle2D.Double`)* Modify your solution to Exercise 12.13 to draw the grid by using class `Rectangle2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
1  // Exercise 12.14 Solution: Grid2JPanel.java
2  // Program draws 10x10 grid using draw().
3  import java.awt.geom.Rectangle2D;
4  import java.awt.Graphics;
5  import java.awt.Graphics2D;
6  import javax.swing.JPanel;
7
8  public class Grid2JPanel extends JPanel
9  {
10    // draw grid
11    public void paintComponent( Graphics g )
12    {
13       super.paintComponent( g );
14
15       Graphics2D g2d = ( Graphics2D ) g;
16
17       for ( int x = 30; x <= 300; x += 30 )
18          for ( int y = 30; y <= 300; y += 30 )
19             g2d.draw( new Rectangle2D.Double( x, y, 30, 30 ) );
20    } // end method paintComponent
21 } // end class Grid2JPanel
```

```
1  // Exercise 12.14 Solution: Grid2.java
2  // Program draws 10x10 grid using draw().
3  import javax.swing.JFrame;
4
5  public class Grid2
6  {
7     public static void main( String args[] )
8     {
9        // create frame for Grid2JPanel
10       JFrame frame = new JFrame( "Drawing grid" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       Grid2JPanel grid2JPanel = new Grid2JPanel();
14       frame.add( grid2JPanel ); // add grid2JPanel to frame
15       frame.setSize( 400, 425 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Grid2
```



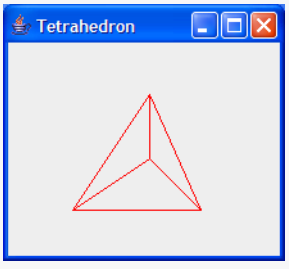
12.15 *(Drawing Tetrahedrons)* Write an application that draws a tetrahedron (a three-dimensional shape with four triangular faces). Use class `GeneralPath` and method `draw` of class `Graphics2D`.

ANS:

```
1  // Exercise 12.15 Solution: TetrahedronJPanel.java
2  // Program draws a tetrahedron.
3  import java.awt.Color;
4  import java.awt.geom.GeneralPath;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import javax.swing.JPanel;
8
9  public class TetrahedronJPanel extends JPanel
10 {
11    // draw tetrahedron
12    public void paintComponent( Graphics g )
13    {
14       super.paintComponent( g );
15
16       int baseX[] = { 110, 150, 50, 110 };
17       int baseY[] = { 90, 130, 130, 90 };
18       int x = 110, y = 40;
19
20       Graphics2D g2d = ( Graphics2D ) g;
21       g2d.setColor( Color.red );
22
23       GeneralPath tetrahedron = new GeneralPath();
24       tetrahedron.moveTo( baseX[ 0 ], baseY[ 0 ] );
```

```
25
26       // create path for the tetrahedron
27       for ( int i = 1; i < 4; i++ )
28       {
29          tetrahedron.lineTo( x, y );
30          tetrahedron.moveTo( baseX[ i - 1 ], baseY[ i - 1 ] );
31          tetrahedron.lineTo( baseX[ i ], baseY[ i ] );
32       } // end for
33
34       tetrahedron.closePath();
35       g2d.draw( tetrahedron );
36    } // end method paintComponent
37 } // end class TetrahedronJPanel
```

```
1  // Exercise 12.15 Solution: Tetrahedron.java
2  // Program draws a tetrahedron.
3  import javax.swing.JFrame;
4
5  public class Tetrahedron
6  {
7     public static void main( String args[] )
8     {
9        // create frame for TetrahedronJPanel
10       JFrame frame = new JFrame( "Tetrahedron" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       TetrahedronJPanel tetrahedronJPanel = new TetrahedronJPanel();
14       frame.add( tetrahedronJPanel ); // add tetrahedronJPanel to frame
15       frame.setSize( 220, 200 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Tetrahedron
```

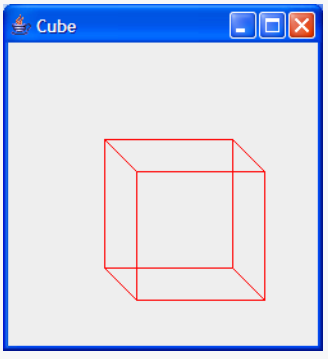


12.16 *(Drawing Cubes)* Write an application that draws a cube. Use class `GeneralPath` and method `draw` of class `Graphics2D`.

ANS:

```
// Exercise 12.16 Solution: CubeJPanel.java
// Program draws a cube.
import java.awt.Color;
import java.awt.geom.GeneralPath;
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JPanel;

public class CubeJPanel extends JPanel
{
   // draw cube
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );

      // one base
      int base1X[] = { 100, 100, 200, 200, 100 };
      int base1Y[] = { 100, 200, 200, 100, 100 };

      // second base
      int base2X[] = { 75, 75, 175, 175, 75 };
      int base2Y[] = { 75, 175, 175 ,75, 75 };

      Graphics2D g2d = ( Graphics2D ) g;
      g2d.setColor( Color.red );

      GeneralPath cube = new GeneralPath();

      // create path for the cube
      for ( int i = 1; i <= 4; i++ )
      {
         // create the first base
         cube.moveTo( base1X[ i - 1 ], base1Y[ i - 1 ] );
         cube.lineTo( base1X[ i ], base1Y[ i ] );

         // create the second base
         cube.moveTo( base2X[ i - 1 ], base2Y[ i - 1 ] );
         cube.lineTo( base2X[ i ], base2Y[ i ] );

         // create the lines between the bases
         cube.moveTo( base1X[ i ], base1Y[ i ] );
         cube.lineTo( base2X[ i ], base2Y[ i ] );
      } // end for

      g2d.draw( cube );
   } // end method paintComponent
} // end class CubeJPanel
```

```
1  // Exercise 12.16 Solution: Cube.java
2  // Program draws a cube.
3  import javax.swing.JFrame;
4
5  public class Cube
6  {
7     public static void main( String args[] )
8     {
9        // create frame for CubeJPanel
10       JFrame frame = new JFrame( "Cube" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       CubeJPanel cubeJPanel = new CubeJPanel();
14       frame.add( cubeJPanel ); // add cubeJPanel to frame
15       frame.setSize( 250, 270 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Cube
```



12.17 *(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 do not need to `import` it.] Use the following formulas (r is the radius):

$diameter = 2r$
$circumference = 2\pi r$
$area = \pi r^2$

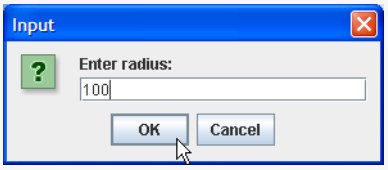
The user should also be prompted for a set of coordinates in addition to the radius. Then draw the circle, and display the circle's diameter, circumference and area, using an `Ellipse2D.Double` object to represent the circle and method `draw` of class `Graphics2D` to display the circle.

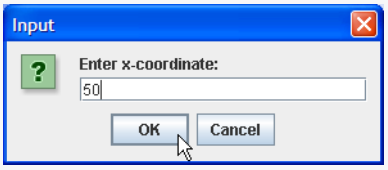
ANS:

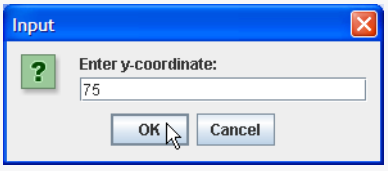
```
1  // Exercise 12.17 Solution: CirclesJPanel.java
2  // Program calculates the area, circumference
3  // and diameter for a circle and draws the circle
4  import java.awt.geom.Ellipse2D;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import javax.swing.JPanel;
8
9  public class CirclesJPanel extends JPanel
10 {
11    private double radius;
12    private int x;
13    private int y;
14
15    // constructor initialize applet by obtaining values from user
16    public CirclesJPanel( double inputRadius, int inputX, int inputY )
17    {
18       radius = inputRadius;
19       x = inputX;
20       y = inputY;
21    } // end CirclesJPanel constructor
22
23    // draw results on applet's background
24    public void paintComponent( Graphics g )
25    {
26       Graphics2D g2d = ( Graphics2D ) g;
27
28       g.drawString( String.format(
29          "Diameter is %f", ( 2 * radius ) ), 25, 30 );
30       g.drawString( String.format(
31          "Area is %f", ( Math.PI * radius * radius ) ), 25, 45 );
32       g.drawString( String.format(
33          "Circumference is %f", ( 2 * Math.PI * radius ) ), 25, 60 );
34
35       g2d.draw( new Ellipse2D.Double( x, y, radius, radius ) );
36    } // end method paintComponent
37 } // end class CirclesJPanel
```

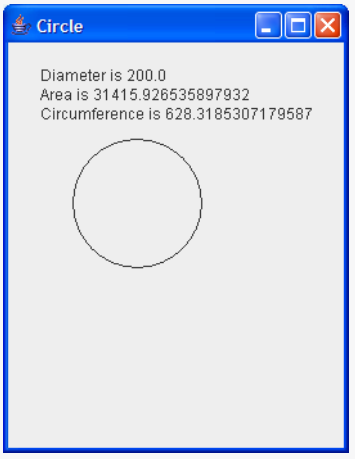
```
1  // Exercise 12.17 Solution: Circle.java
2  // Program calculates the area, circumference
3  // and diameter for a circle and draws the circle
4  import javax.swing.JFrame;
5  import javax.swing.JOptionPane;
6
7  public class Circle
```

```
{
   public static void main( String args[] )
   {
      double inputRadius;
      int inputX;
      int inputY;

      // create frame for CirclesJPanel
      JFrame frame = new JFrame( "Circle" );
      frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

      // read from user, convert to proper type
      inputRadius = Double.parseDouble(
         JOptionPane.showInputDialog( "Enter radius:" ) );
      inputX = Integer.parseInt(
         JOptionPane.showInputDialog( "Enter x-coordinate:" ) );
      inputY = Integer.parseInt(
         JOptionPane.showInputDialog( "Enter y-coordinate:" ) );

      CirclesJPanel circlesJPanel =
         new CirclesJPanel( inputRadius, inputX, inputY );
      frame.add( circlesJPanel ); // add circlesJPanel to frame
      frame.setSize( 275, 350 ); // set frame size
      frame.setVisible( true ); // display frame
   } // end main
} // end class Circle
```









12.18 *(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?

ANS:

```
1  // Exercise 12.18 Solution: Saver1JPanel.java
2  // Program simulates a simple screen saver
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class Saver1JPanel extends JPanel
8  {
9     private final int DELAY = 9999999;
10
11    // draw lines
12    public void paintComponent( Graphics g )
13    {
14       super.paintComponent( g ); // call superclass's paintComponent
15
16       int x, y, x1, y1;
17
18       // draw 100 random lines
19       for ( int i = 0; i < 100; i++ )
20       {
21          x = ( int ) ( Math.random() * 300 );
22          y = ( int ) ( Math.random() * 300 );
23          x1 = ( int ) ( Math.random() * 300 );
24          y1 = ( int ) ( Math.random() * 300 );
25
26          g.setColor( new Color( ( float ) Math.random(),
27             ( float ) Math.random(), ( float ) Math.random() ) );
28          g.drawLine( x, y, x1, y1 );
29
30          // slow the drawing down. the body of the for loop is empty
31          for ( int q = 1; q < DELAY; q++ ) ;
32       } // end outer for
33
34       repaint(); // repaint component
35    } // end method paintComponent
36 } // end class Saver1JPanel
```

```
1  // Exercise 12.18 Solution: Saver1.java
2  // Program simulates a simple screen saver
3  import javax.swing.JFrame;
4
5  public class Saver1
6  {
```

```
7      public static void main( String args[] )
8      {
9         // create frame for Saver1JPanel
10        JFrame frame = new JFrame( "Saver1" );
11        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13        Saver1JPanel saver1JPanel = new Saver1JPanel();
14        frame.add( saver1JPanel ); // add saver1JPanel to frame
15        frame.setSize( 300, 300 ); // set frame size
16        frame.setVisible( true ); // display frame
17     } // end main
18  } // end class Saver1
```

12.19 *(Screen Saver Using `Timer`)* Package `javax.swing` contains 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 12.18 to remove the call to `repaint` from 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:

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

This creates an instance of class `Timer` that will call `this` object's `actionPerformed` method every 1000 milliseconds (i.e., every second).

ANS:

```
1  // Exercise 12.19 Solution: SaverJPanel.java
2  // Program simulates a simple screen saver
3  import java.awt.Color;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import java.awt.Graphics;
7  import java.util.Random;
```

```
import javax.swing.JPanel;
import javax.swing.Timer;

public class SaverJPanel extends JPanel implements ActionListener
{
   private final int DELAY = 9999999;
   private Timer timer;

   // constructor sets window's title bar string and dimensions
   public SaverJPanel()
   {
      timer = new Timer( 1000, this ); // create the timer
      timer.start();
   } // end SaverJPanel constructor

   // draw lines
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );
      Random random = new Random(); // creat random number generator

      int x, y, x1, y1;

      for ( int i = 0; i < 100; i++ )
      {
         x = random.nextInt( 300 );
         y = random.nextInt( 300 );
         x1 = random.nextInt( 300 );
         y1 = random.nextInt( 300 );

         g.setColor( new Color( random.nextFloat(),
            random.nextFloat(), random.nextFloat() ) );
         g.drawLine( x, y, x1, y1 );

         // slow the drawing down. the body of the for loop is empty
         for ( int q = 1; q < DELAY; q++ ) ;
      } // end for
   } // end method paintComponent

   // repaint JPanel
   public void actionPerformed( ActionEvent actionEvent )
   {
      repaint();
   } // end method actionPerformed
} // end class SaverJPanel
```

```
// Exercise 12.19 Solution: Saver2.java
// Program simulates a simple screen saver
import javax.swing.JFrame;

public class Saver2
{
```

```
7      public static void main( String args[] )
8      {
9         // create frame for SaverJPanel
10        JFrame frame = new JFrame( "Saver2" );
11        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13        SaverJPanel saverJPanel = new SaverJPanel();
14        frame.add( saverJPanel ); // add saverJPanel to frame
15        frame.setSize( 300, 300 ); // set frame size
16        frame.setVisible( true ); // display frame
17     } // end main
18  } // end class Saver2
```

12.20 *(Screen Saver for a Random Number of Lines)* Modify your solution to Exercise 12.19 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`.

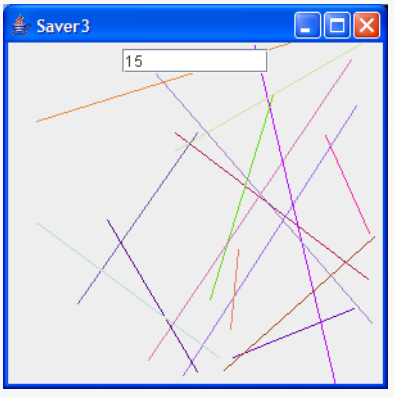
ANS:

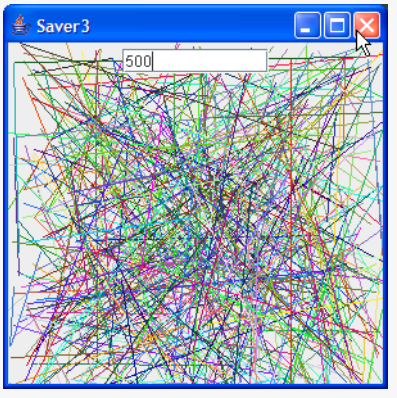
```
1  // Exercise 12.20 Solution: SaverJPanel.java
2  // Program simulates a simple screen saver
3  import java.awt.Color;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import java.awt.FlowLayout;
7  import java.awt.Graphics;
8  import java.util.Random;
9  import javax.swing.JPanel;
10 import javax.swing.JTextField;
11 import javax.swing.Timer;
12
13 public class SaverJPanel extends JPanel implements ActionListener
14 {
```

```
15    private final int DELAY = 9999999;
16    private Timer timer;
17    private int numberLines;
18
19    private JTextField input;
20
21    // constructor sets window's title bar string and dimensions
22    public SaverJPanel()
23    {
24       numberLines = 100; // initialize the number of lines to draw
25
26       timer = new Timer( 1000, this ); // create the timer
27       timer.start();
28
29       input = new JTextField( 10 );
30       input.addActionListener(
31
32          new ActionListener() // anonymous inner class
33          {
34             public void actionPerformed( ActionEvent event )
35             {
36                numberLines = Integer.parseInt( input.getText() );
37             } // end method actionPerformed
38          } // end anonymous inner class
39       ); // end call to addActionListner
40
41       setLayout( new FlowLayout() );
42       add( input );
43    } // end SaverJPanel constructor
44
45    // draw lines
46    public void paintComponent( Graphics g )
47    {
48       super.paintComponent( g );
49       Random random = new Random(); // create random number generator
50       int x, y, x1, y1;
51
52       for ( int i = 0; i < numberLines; i++ )
53       {
54          x = random.nextInt( 300 );
55          y = random.nextInt( 300 );
56          x1 = random.nextInt( 300 );
57          y1 = random.nextInt( 300 );
58
59          g.setColor( new Color( random.nextFloat(),
60             random.nextFloat(), random.nextFloat() ) );
61          g.drawLine( x, y, x1, y1 );
62
63          // slow the drawing down. the body of the for loop is empty
64          for ( int q = 1; q < DELAY; q++ ) ;
65       } // end for
66    } // end method paintComponent
67
68    // repaint JPanel
```

```
69       public void actionPerformed( ActionEvent actionEvent )
70       {
71          repaint();
72       } // end method actionPerformed
73    } // end class SaverJPanel
```

```
1  // Exercise 12.20 Solution: Saver3.java
2  // Program simulates a simple screen saver
3  import javax.swing.JFrame;
4
5  public class Saver3
6  {
7     public static void main( String args[] )
8     {
9        // create frame for SaverJPanel
10       JFrame frame = new JFrame( "Saver3" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       SaverJPanel saverJPanel = new SaverJPanel();
14       frame.add( saverJPanel ); // add saverJPanel to frame
15       frame.setSize( 300, 300 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Saver3
```





12.21 *(Screen Saver with Shapes)* Modify your solution to Exercise 12.19 such that it uses random-number generation to choose different shapes to display. Use methods of class `Graphics`.

ANS:

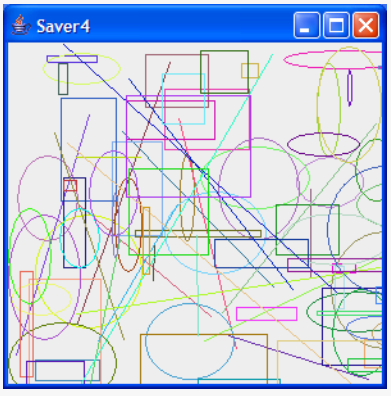
```
1  // Exercise 12.21 Solution: SaverJPanel.java
2  // Program simulates a simple screen saver
```

```
3  import java.awt.Color;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import java.awt.Graphics;
7  import java.util.Random;
8  import javax.swing.JPanel;
9  import javax.swing.Timer;
10
11 public class SaverJPanel extends JPanel implements ActionListener
12 {
13    private final int DELAY = 9999999;
14    private Timer timer;
15    private Random random;
16
17    // constructor sets window's title bar string and dimensions
18    public SaverJPanel()
19    {
20       random = new Random(); // create random number generator
21       timer = new Timer( 1000, this ); // create the timer
22       timer.start();
23    } // end SaverJPanel constructor
24
25    // draw shapes
26    public void paintComponent( Graphics g )
27    {
28       super.paintComponent( g );
29       int shape;
30
31       for ( int i = 0; i < 100; i++ )
32       {
33          shape = random.nextInt( 4 ); // pick a random shape
34
35          // draw different shapes
36          switch( shape )
37          {
38              case 0:
39                 makeLine( g );
40                 break;
41             case 1:
42                 makeRect( g );
43                 break;
44             case 2:
45                 makeOval( g );
46                 break;
47             case 3:
48                 makeRoundRect( g );
49                 break;
50          } // end switch
51
52          // slow the drawing down. the body of the for loop is empty
53          for ( int q = 1; q < DELAY; q++ ) ;
54       } // end for
55    } // end method paintComponent
56
```

```
57      // repaint JPanel
58      public void actionPerformed( ActionEvent actionEvent )
59      {
60         repaint();
61      } // end method actionPerformed
62
63      // draw a random lines
64      private void makeLine( Graphics g )
65      {
66         int x = random.nextInt( 300 );
67         int y = random.nextInt( 300 );
68         int x1 = random.nextInt( 300 );
69         int y1 = random.nextInt( 300 );
70
71         g.setColor( new Color( random.nextFloat(),
72            random.nextFloat(), random.nextFloat() ) );
73         g.drawLine( x, y, x1, y1 );
74      } // end method makeLine
75
76      // draw a random rectangle
77      private void makeRect( Graphics g )
78      {
79         int x = random.nextInt( 300 );
80         int y = random.nextInt( 300 );
81         int width = random.nextInt( 100 );
82         int height = random.nextInt( 100 );
83
84         g.setColor( new Color( random.nextFloat(),
85            random.nextFloat(), random.nextFloat() ) );
86         g.drawRect( x, y, width, height );
87      } // end method makeRect
88
89      // draw a random oval
90      private void makeOval( Graphics g )
91      {
92         int x = random.nextInt( 300 );
93         int y = random.nextInt( 300 );
94         int width = random.nextInt( 100 );
95         int height = random.nextInt( 100 );
96
97         g.setColor( new Color( random.nextFloat(),
98            random.nextFloat(), random.nextFloat() ) );
99         g.drawOval( x, y, width, height );
100     } // end method makeOval
101
102     // draw a random rounded rectangle
103     private void makeRoundRect( Graphics g )
104     {
105        int x = random.nextInt( 300 );
106        int y = random.nextInt( 300 );
107        int width = random.nextInt( 100 );
108        int height = random.nextInt( 100 );
109        int arcWidth = random.nextInt() * width;
110        int arcHeight = random.nextInt() * height;
```

```
111
112        g.setColor( new Color( random.nextFloat(),
113           random.nextFloat(), random.nextFloat() ) );
114        g.drawRoundRect( x, y, width, height, arcWidth, arcHeight );
115     } // end method makeRoundRect
116  } // end class SaverJPanel
```

```
 1  // Exercise 12.21 Solution: Saver4.java
 2  // Program simulates a simple screen saver
 3  import javax.swing.JFrame;
 4
 5  public class Saver4
 6  {
 7     public static void main( String args[] )
 8     {
 9        // create frame for SaverJPanel
10        JFrame frame = new JFrame( "Saver4" );
11        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13        SaverJPanel saverJPanel = new SaverJPanel();
14        frame.add( saverJPanel ); // add saverJPanel to frame
15        frame.setSize( 300, 300 ); // set frame size
16        frame.setVisible( true ); // display frame
17     } // end main
18  } // end class Saver4
```



12.22 *(Screen Saver Using the Java 2D API)* Modify your solution to Exercise 12.21 to use classes and drawing capabilities of the Java 2D API. Draw shapes like rectangles and ellipses with randomly generated gradients. Use class `GradientPaint` to generate the gradient.

ANS:

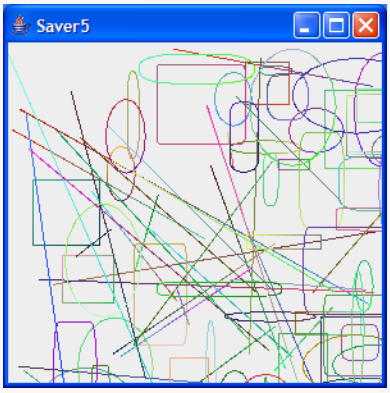
```
1  // Exercise 12.22 Solution: SaverJPanel.java
2  // Program simulates a simple screen saver
3  import java.awt.Color;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import java.awt.geom.Ellipse2D;
7  import java.awt.geom.Line2D;
8  import java.awt.geom.Rectangle2D;
9  import java.awt.geom.RoundRectangle2D;
10 import java.awt.GradientPaint;
11 import java.awt.Graphics;
12 import java.awt.Graphics2D;
13 import java.util.Random;
14 import javax.swing.JPanel;
15 import javax.swing.Timer;
16
17 public class SaverJPanel extends JPanel implements ActionListener
18 {
19    private final int DELAY = 9999999;
20    private Timer timer;
21    private Random random;
22
23    // constructor sets window's title bar string and dimensions
24    public SaverJPanel()
25    {
26       random = new Random(); // create random number generator
27       timer = new Timer( 1000, this ); // create a new timer
28       timer.start();
29    } // end SaverJPanel constructor
30
31    // draw shapes
32    public void paintComponent( Graphics g )
33    {
34       super.paintComponent( g );
35
36       int shape;
37
38       // draw 100 different shapes
39       for ( int i = 0; i < 100; i++ )
40       {
41          shape = random.nextInt( 4 ); // pick a random shape
42
43          switch( shape )
44          {
45             case 0:
46                makeLine( g );
47                break;
48             case 1:
49                makeRect( g );
```

```
50                break;
51             case 2:
52                makeOval( g );
53                break;
54             case 3:
55                makeRoundRect( g );
56                break;
57          } // end switch
58
59          // slow the drawing down. the body of the for loop is empty
60          for ( int q = 1; q < DELAY; q++ ) ;
61       } // end for
62    } // end method paintComponent
63
64    // repaint JPanel
65    public void actionPerformed( ActionEvent actionEvent )
66    {
67       repaint();
68    } // end method actionPerformed
69
70    // draw a random lines
71    private void makeLine( Graphics g )
72    {
73       Graphics2D g2d = ( Graphics2D ) g;
74
75       double x = random.nextDouble() * 300;
76       double y = random.nextDouble() * 300;
77       double x1 = random.nextDouble() * 300;
78       double y1 = random.nextDouble() * 300;
79
80       g2d.setPaint( new GradientPaint( ( int ) x, ( int ) y,
81          new Color( random.nextFloat(), random.nextFloat(),
82          random.nextFloat() ), ( int ) x1, ( int ) y1,
83          new Color( random.nextFloat(), random.nextFloat(),
84          random.nextFloat() ), true ) );
85
86       g2d.draw( new Line2D.Double( x, y, x1, y1 ) );
87    } // end method makeline
88
89    // draw a random rectangle
90    private void makeRect( Graphics g )
91    {
92       Graphics2D g2d = ( Graphics2D ) g;
93
94       double x = random.nextDouble() * 300;
95       double y = random.nextDouble() * 300;
96       double width = random.nextDouble() * 100;
97       double height = random.nextDouble() * 100;
98
99       g2d.setPaint( new GradientPaint( ( int ) x, ( int ) y,
100         new Color( random.nextFloat(), random.nextFloat(),
101         random.nextFloat() ), ( int )( x + width ),
102         ( int )( y + height ),
103         new Color( random.nextFloat(), random.nextFloat(),
104         random.nextFloat() ), true ) );
```

```
105
106        g2d.draw( new Rectangle2D.Double( x, y, width, height ) );
107     } // end method makeRect
108
109     // draw a random oval
110     private void makeOval( Graphics g )
111     {
112        Graphics2D g2d = ( Graphics2D ) g;
113
114        double x = random.nextDouble() * 300;
115        double y = random.nextDouble() * 300;
116        double width = random.nextDouble() * 100;
117        double height = random.nextDouble() * 100;
118
119        g2d.setPaint( new GradientPaint( ( int ) x, ( int ) y,
120           new Color( random.nextFloat(), random.nextFloat(),
121           random.nextFloat() ), ( int )( x + width ),
122           ( int )( y + height ),
123           new Color( random.nextFloat(), random.nextFloat(),
124           random.nextFloat() ), true ) );
125
126        g2d.draw( new Ellipse2D.Double( x, y, width, height ) );
127     } // end method makeOval
128
129     // create a random rounded rectangle
130     private void makeRoundRect( Graphics g )
131     {
132        Graphics2D g2d = ( Graphics2D ) g;
133
134        double x = random.nextDouble() * 300;
135        double y = random.nextDouble() * 300;
136        double width = random.nextDouble() * 100;
137        double height = random.nextDouble() * 100;
138        double arcWidth = random.nextDouble() * width;
139        double arcHeight = random.nextDouble() * height;
140
141        g2d.setPaint( new GradientPaint( ( int ) x, ( int ) y,
142           new Color( random.nextFloat(), random.nextFloat(),
143           random.nextFloat() ), ( int )( x + width ),
144           ( int )( y + height ),
145           new Color( random.nextFloat(), random.nextFloat(),
146           random.nextFloat() ), true ) );
147
148        g2d.draw( new RoundRectangle2D.Double( x, y, width, height,
149           arcWidth, arcHeight ) );
150     } // end method makeRoundRect
151  } // end class SaverJPanel
```

```
1  // Exercise 12.22 Solution: Saver5.java
2  // Program simulates a simple screen saver
3  import javax.swing.JFrame;
```

```
4
5  public class Saver5
6  {
7     public static void main( String args[] )
8     {
9        // create frame for SaverJPanel
10       JFrame frame = new JFrame( "Saver5" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       SaverJPanel saverJPanel = new SaverJPanel();
14       frame.add( saverJPanel ); // add saverJPanel to frame
15       frame.setSize( 300, 300 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Saver5
```



12.23 *(Turtle Graphics)* Modify your solution to Exercise 7.21—*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.

ANS:

```
1  // Exercise 12.23: TurtleJPanel.java
2  // Drawing turtle graphics based on turtle commands.
3  import java.awt.Color;
4  import java.awt.event.ActionListener;
5  import java.awt.event.ActionEvent;
6  import java.awt.FlowLayout;
7  import java.awt.geom.Line2D;
8  import java.awt.Graphics;
9  import java.awt.Graphics2D;
10 import javax.swing.JButton;
11 import javax.swing.JPanel;
```

```
12 import javax.swing.JTextField;
13
14 public class TurtleJPanel extends JPanel
15 {
16    // an array of turtle commands
17    private static final int MAXCOMMANDS = 10;
18    private int commandArray[][] = new int [ MAXCOMMANDS ][ 2 ];
19
20    // state variables
21    private int direction;
22    private int count;
23    private int xPos;
24    private int yPos;
25    private boolean penDown;
26    private boolean print;
27
28    private JTextField input;
29    private JButton downJButton;
30    private JButton upJButton;
31    private JButton moveJButton;
32    private JButton turnRightJButton;
33    private JButton turnLeftJButton;
34    private JButton printJButton;
35    private JButton clearJButton;
36
37    private static final int PEN_UP = 1, PEN_DOWN = 2,
38       TURN_RIGHT = 3, TURN_LEFT = 4, MOVE = 5;
39
40    // constructor sets up GUI components and initialize variables
41    public TurtleJPanel()
42    {
43       upJButton = new JButton( "Pen Up" );
44       upJButton.addActionListener(
45          new ActionListener () // anonymous inner class
46          {
47             public void actionPerformed( ActionEvent event )
48             {
49                commandArray[ count ][ 0 ] = PEN_UP;
50                count++;
51
52                if ( count == MAXCOMMANDS )
53                {
54                   upJButton.setEnabled( false );
55                   downJButton.setEnabled( false );
56                   moveJButton.setEnabled( false );
57                   input.setEnabled( false );
58                   turnRightJButton.setEnabled( false );
59                   turnLeftJButton.setEnabled( false );
60                } // end if
61             } // end method actionPerformed
62          } // end anonymous inner class
63       ); // end call to addActionListener
64
65       downJButton = new JButton( "Pen Down" );
```

```
66          downJButton.addActionListener(
67             new ActionListener() // anonymous inner class
68             {
69                public void actionPerformed( ActionEvent event )
70                {
71                   commandArray[ count ][ 0 ] = PEN_DOWN;
72                   count++;
73
74                   if ( count == MAXCOMMANDS )
75                   {
76                      upJButton.setEnabled( false );
77                      downJButton.setEnabled( false );
78                      moveJButton.setEnabled( false );
79                      input.setEnabled( false );
80                      turnRightJButton.setEnabled( false );
81                      turnLeftJButton.setEnabled( false );
82                   } // end if
83                } // end method actionPerformed
84             } // end anonymous inner class
85          ); // end call to addActionListener
86
87          moveJButton = new JButton( "Move forward" );
88          moveJButton.addActionListener(
89             new ActionListener() // anonymous inner class
90             {
91                public void actionPerformed( ActionEvent event )
92                {
93                   // get the distance to move
94                   int spaces = Integer.parseInt( input.getText() );
95                   input.setText( "" );
96
97                   commandArray[ count ][ 0 ] = MOVE;
98                   commandArray[ count ][ 1 ] = spaces;
99
100                  count++;
101
102                  if ( count == MAXCOMMANDS )
103                  {
104                     upJButton.setEnabled( false );
105                     downJButton.setEnabled( false );
106                     moveJButton.setEnabled( false );
107                     input.setEnabled( false );
108                     turnRightJButton.setEnabled( false );
109                  } // end if
110               } // end method actionPerformed
111            } // end anonymous inner class
112         ); // end call to addActionListener
113
114         input = new JTextField( 4 );
115         input.addActionListener(
116            new ActionListener () // anonymous inner class
117            {
118               public void actionPerformed( ActionEvent event )
119               {
```

```
120                // get the distance to move
121                int spaces = Integer.parseInt( input.getText() );
122                input.setText( "" );
123
124                commandArray[ count ][ 0 ] = MOVE;
125                commandArray[ count ][ 1 ] = spaces;
126
127                count++;
128
129                if ( count == MAXCOMMANDS )
130                {
131                   upJButton.setEnabled( false );
132                   downJButton.setEnabled( false );
133                   moveJButton.setEnabled( false );
134                   input.setEnabled( false );
135                   turnRightJButton.setEnabled( false );
136                   turnLeftJButton.setEnabled( false );
137                } // end if
138             } // end method actionPerformed
139          } // end anonymous inner class
140       ); // end call to addActionListener
141
142       turnRightJButton = new JButton( "Turn right" );
143       turnRightJButton.addActionListener(
144          new ActionListener() // anonymous inner class
145          {
146             public void actionPerformed( ActionEvent event )
147             {
148                commandArray[ count ][ 0 ] = TURN_RIGHT;
149                count++;
150
151                if ( count == MAXCOMMANDS )
152                {
153                   upJButton.setEnabled( false );
154                   downJButton.setEnabled( false );
155                   moveJButton.setEnabled( false );
156                   input.setEnabled( false );
157                   turnRightJButton.setEnabled( false );
158                   turnLeftJButton.setEnabled( false );
159                } // end if
160             } // end method actionPerformed
161          } // end anonymous inner class
162       ); // end call to addActionListener
163
164       turnLeftJButton = new JButton( "Turn left" );
165       turnLeftJButton.addActionListener(
166          new ActionListener() // anonymous inner class
167          {
168             public void actionPerformed( ActionEvent event )
169             {
170                commandArray[ count ][ 0 ] = TURN_LEFT;
171                count++;
172
173                if ( count == MAXCOMMANDS )
```

```
174                {
175                   upJButton.setEnabled( false );
176                   downJButton.setEnabled( false );
177                   moveJButton.setEnabled( false );
178                   input.setEnabled( false );
179                   turnRightJButton.setEnabled( false );
180                   turnLeftJButton.setEnabled( false );
181                } // end if
182             } // end method actionPerformed
183          } // end anonymous inner class
184       ); // end call to addActionListener
185
186       printJButton = new JButton( "Output drawing" );
187       printJButton.addActionListener(
188          new ActionListener () // anonymous inner class
189          {
190             public void actionPerformed( ActionEvent event )
191             {
192                print = true; // run the commands
193                repaint(); // update drawing
194             } // end method actionPerformed
195          } // end anonymous inner class
196       ); // end call to addActionListener
197
198       clearJButton = new JButton( "Clear drawing" );
199       clearJButton.addActionListener(
200          new ActionListener() // anonymous inner class
201          {
202             public void actionPerformed( ActionEvent event )
203             {
204                repaint(); // clear the application
205                clearCommands(); // clear the command array
206
207                upJButton.setEnabled( true );
208                downJButton.setEnabled( true );
209                moveJButton.setEnabled( true );
210                input.setEnabled( true );
211                turnRightJButton.setEnabled( true );
212                turnLeftJButton.setEnabled( true );
213             } // end method actionPerformed
214          } // end anonymous inner class
215       ); // end call to addActionListener
216
217       setLayout( new FlowLayout() );
218       add( upJButton );
219       add( downJButton );
220       add( moveJButton );
221       add( input );
222       add( turnRightJButton );
223       add( turnLeftJButton );
224       add( printJButton );
225       add( clearJButton );
226
227       print = false; // will set to true when printJButton is clicked
```

```
228        clearCommands();
229     } // end constructor TurtleJPanel
230
231     // draw on JPanel
232     public void paintComponent( Graphics g )
233     {
234        super.paintComponent( g );
235
236        Graphics2D g2d = ( Graphics2D ) g;
237
238        if ( print ) // printJButton is clicked, draw turtle graphics
239        {
240           g2d.setColor( Color.RED );
241           executeCommands( g2d );
242        } // end if
243     } // end method paintComponent
244
245     // reset array commandArray
246     public void clearCommands()
247     {
248        count = 0;
249
250        // initialize array commandArray
251        for ( int i = 0; i < MAXCOMMANDS; i++ )
252           for ( int j = 0; j < 2; j++ )
253              commandArray[ i ][ j ] = 0;
254     } // end method clearCommands
255
256     // execute commands stored in commandArray
257     public void executeCommands( Graphics2D g2d )
258     {
259        // reset the state variables
260        direction = 0;
261        xPos = 150;
262        yPos = 150;
263        penDown = false;
264
265        // continue executing commands until reach the end
266        for ( int commandNumber = 0; commandNumber < count;
267           commandNumber++ )
268        {
269           int command = commandArray[ commandNumber ][ 0 ];
270
271           // determine what command was entered  and peform desired action
272           switch ( command )
273           {
274              case PEN_UP:
275                 penDown = false;
276                 break;
277              case PEN_DOWN:
278                 penDown = true;
279                 break;
280              case TURN_RIGHT:
281                 direction = turnRight( direction );
```

```
282                break;
283             case TURN_LEFT:
284                direction = turnLeft( direction );
285                break;
286             case MOVE:
287                int distance = commandArray[ commandNumber ][ 1 ];
288                movePen( g2d, penDown, direction, distance );
289                break;
290          } // end switch
291       } // end for
292
293       print = false; // reset print
294    } // end method executeCommands
295
296    // method to turn turtle to the right
297    public int turnRight( int d )
298    {
299       return --d < 0 ? 3 : d;
300    } // end method turnRight
301
302    // method to turn turtle to the left
303    public int turnLeft( int d )
304    {
305       return ++d > 3 ? 0 : d;
306    } // end method turnLeft
307
308    // method to move the pen
309    public void movePen( Graphics2D g2d, boolean down, int dir, int dist )
310    {
311       // determine which way to move pen
312       switch ( dir )
313       {
314          case 0: // move down
315             if ( down )
316                g2d.draw( new Line2D.Double(
317                   xPos, yPos, xPos, yPos + dist * 10 ) );
318             yPos += dist * 10;
319             break;
320          case 1: // move right
321             if ( down )
322                g2d.draw( new Line2D.Double(
323                   xPos, yPos, xPos + dist * 10, yPos ) );
324             xPos += dist * 10;
325             break;
326          case 2: // move up
327             if ( down )
328                g2d.draw( new Line2D.Double(
329                   xPos, yPos, xPos, yPos - dist * 10 ) );
330             yPos -= dist * 10;
331             break;
332          case 3: // move left
333             if ( down )
334                g2d.draw( new Line2D.Double(
335                   xPos, yPos, xPos - dist * 10, yPos ) );
```

```
            xPos -= dist * 10;
            break;
         } // end switch
      } // end method movePen
   } // end class TurtleJPanel
```

```
1  // Exercise 12.23: TurtleGraphics.java
2  // Drawing turtle graphics based on turtle commands.
3  import javax.swing.JFrame;
4
5  public class TurtleGraphics
6  {
7     public static void main( String args[] )
8     {
9        // create frame for TurtleJPanel
10       JFrame frame = new JFrame( "Turtle Graphics" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       TurtleJPanel turtlePanel = new TurtleJPanel();
14       frame.add( turtlePanel ); // add turtleJPanel to frame
15       frame.setSize( 800, 500 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class TurtleGraphics
```

12.24 *(Knight's Tour)* Produce a graphical version of the Knight's Tour problem (Exercise 7.22, Exercise 7.23 and Exercise 7.26). 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 12.19) to help animate the Knight's Tour.

ANS:

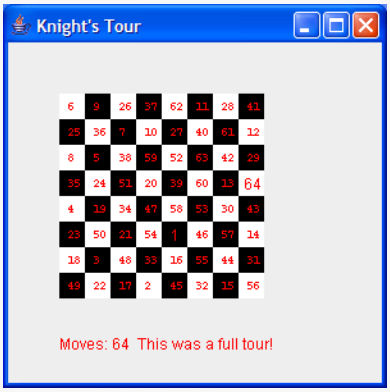
```
1  // Exercise 12.24 Solution: KnightJPanel.java
2  // Knight's Tour - access version runs one tour
3  import java.awt.Color;
4  import java.awt.Font;
5  import java.awt.Graphics;
6  import java.util.Random;
7  import javax.swing.JPanel;
8
9  public class KnightJPanel extends JPanel
10 {
11    private final int DELAY = 8000000;
12    private final int INSET = 40;
13    private boolean done;
14    private boolean closed;
15    private int board[][];
16    private int currentRow;
17    private int currentColumn;
18    private int moveNumber;
19    private int testRow;
20    private int testColumn;
21    private int count;
22    private int minRow;
23    private int minColumn;
24    private int minAccess;
25    private int accessNumber;
26    private int firstMoveRow;
27    private int firstMoveColumn;
28
29    private int access[][] = { { 2, 3, 4, 4, 4, 4, 3, 2 },
30                               { 3, 4, 6, 6, 6, 6, 4, 3 },
31                               { 4, 6, 8, 8, 8, 8, 6, 4 },
32                               { 4, 6, 8, 8, 8, 8, 6, 4 },
33                               { 4, 6, 8, 8, 8, 8, 6, 4 },
34                               { 4, 6, 8, 8, 8, 8, 6, 4 },
35                               { 3, 4, 6, 6, 6, 6, 4, 3 },
36                               { 2, 3, 4, 4, 4, 4, 3, 2 } };
37    private int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
38    private int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
39    private Random random;
40
41    // constructor sets up the play
42    public KnightJPanel()
43    {
44       minAccess = 9;
45       board = new int[ 8 ][ 8 ];
46       random = new Random();
47       currentRow = random.nextInt( 8 );
```

```
48          currentColumn = random.nextInt( 8 );
49          firstMoveRow = currentRow;
50          firstMoveColumn = currentColumn;
51       } // end KnightJPanel constructor
52
53       // returns if spot is within range and empty
54       public boolean validMove( int row, int column )
55       {
56          return ( row >= 0 && row < 8 && column >= 0 && column < 8
57                   && board[ row ][ column ] == 0 );
58       } // end method validMove
59
60       // draw chess board with tour
61       public void paintComponent( Graphics g )
62       {
63          super.paintComponent( g );
64
65          int x1 = 0, y1 = 0;
66
67          // draw black and white patterned chess board
68          for ( int j = 0; j <= 7; j++ )
69          {
70             for ( int k = 0; k <= 7; k++ )
71             {
72                if ( ( k + j ) % 2 == 1 )
73                   g.setColor( Color.BLACK );
74                else
75                   g.setColor( Color.WHITE );
76
77                g.fillRect( x1 + INSET, y1 + INSET, 20, 20 );
78                x1 += 20;
79             } // end for
80
81             y1 += 20;
82             x1 = 0;
83          } // end for
84
85          // place first move on board the red font is used for emphasis
86          board[ currentRow ][ currentColumn ] = ++moveNumber;
87          g.setColor( Color.RED );
88          g.drawString( "1", INSET + 7 + 20 * currentRow,
89             INSET + 15 + 20 * currentColumn );
90
91          while ( !done )
92          {
93             accessNumber = minAccess;
94
95             // cycle through each column of the current row and
96             // determine the move with the minimum access number
97             for ( int moveType = 0; moveType < board.length; moveType++ )
98             {
99                testRow = currentRow + vertical[ moveType ];
100               testColumn = currentColumn + horizontal[ moveType ];
101
```

```
102             if ( validMove( testRow, testColumn ) )
103             {
104                if ( access[ testRow ][ testColumn ] < accessNumber )
105                {
106                   accessNumber = access[ testRow ][ testColumn ];
107                   minRow = testRow;
108                   minColumn = testColumn;
109                } // end if
110
111                --access[ testRow ][ testColumn ];
112             } // end if
113          } // end for
114
115          // determine whether the next move has been found
116          if ( accessNumber == minAccess )
117             done = true;
118          else
119          {
120             currentRow = minRow;
121             currentColumn = minColumn;
122             board[ currentRow ][ currentColumn ] = ++moveNumber;
123
124             Font oldFont = g.getFont();
125             Font newFont = new Font( "Monospaced", Font.BOLD, 9 );
126
127             // slow animation
128             for ( int h = 1; h <= DELAY; h++ )
129                ;     // do nothing
130
131             // draw the new move onto the board
132             if ( moveNumber != 64 )
133             {
134                g.setFont( newFont );
135                g.drawString( String.valueOf(
136                   board[ currentRow ][ currentColumn ] ),
137                   INSET + 7 + 20 * currentRow, INSET + 13 + 20 *
138                   currentColumn );
139             } // end if
140             else // emphasize move 64 (final one in board)
141             {
142                g.setFont( oldFont );
143                g.drawString( String.valueOf(
144                   board[ currentRow ][ currentColumn ] ),
145                   INSET + 4 + 20 * currentRow, INSET + 15 + 20 *
146                   currentColumn );
147             } // end else
148
149             g.setFont( oldFont );
150          } // end else
151       } // end while
152
153       // if this is the final move, determine whether it is possible to
154       // reach the first square and close the knights tour
155       if ( moveNumber == 64 )
```

```
156         {
157            for ( int moveType = 0; moveType < 8; moveType++ )
158            {
159               testRow = currentRow + vertical[ moveType ];
160               testColumn = currentColumn + horizontal[ moveType ];
161
162               if ( testRow == firstMoveRow &&
163                  testColumn == firstMoveColumn )
164               {
165                  closed = true;
166                  break;
167               } // end if
168            } // end for
169         } // end if
170
171         // draw result
172         if ( moveNumber == 64 && closed == true )
173            g.drawString( "Moves: " + moveNumber +
174            "  This was a CLOSED tour!", INSET + 0,
175               INSET + 200 );
176         else if ( moveNumber == 64 )
177            g.drawString( "Moves: " + moveNumber +
178               "  This was a full tour!", INSET + 0,
179               INSET + 200 );
180         else
181            g.drawString( "Moves: " + moveNumber +
182               "  This was not a full tour.", INSET + 0,
183               INSET + 200 );
184      } // end method paintComponent
185   } // end class KnightJPanel
```

```
1  // Exercise 12.24 Solution: Knight.java
2  // Knight's Tour - access version runs one tour
3  import javax.swing.JFrame;
4
5  public class Knight
6  {
7     public static void main( String args[] )
8     {
9        // create frame for KnightJPanel
10       JFrame frame = new JFrame( "Knight's Tour" );
11       frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12
13       KnightJPanel knightJPanel = new KnightJPanel();
14       frame.add( knightJPanel ); // add knightJPanel to frame
15       frame.setSize( 300, 300 ); // set frame size
16       frame.setVisible( true ); // display frame
17    } // end main
18 } // end class Knight
```



12.25 *(Tortoise and Hare)* Produce a graphical version of the *Tortoise and Hare* simulation (Exercise 7.28). Simulate the mountain by drawing an arc that extends from the bottom-left corner of the window to the top-right corner of the window. 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. [*Note:* Extend the length of the race from 70 to 300 to allow yourself a larger graphics area.]

ANS:

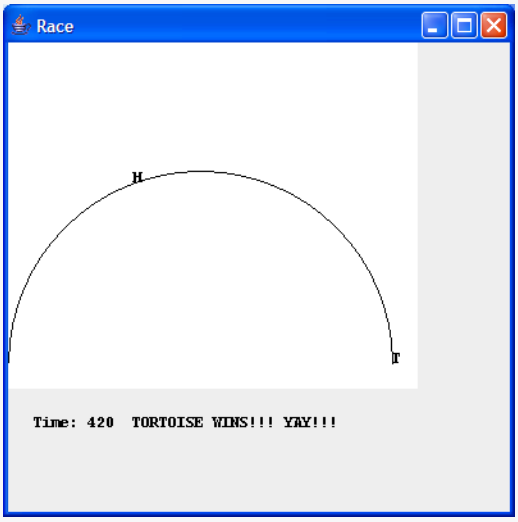
```
1  // Exercise 12.25 Solution: RaceJPanel.java
2  // Program simulates the race between the tortoise and the hare.For
3  // graphical purposes the race has been extended to 300. The mountain is
4  // represented by a semicircle using the equation for a
5  // circle: (x-h)^2 + (y-k)^2 = r^2. A slight adjustment was made in the
6  // equation to help compensate for the different directions of the
7  // coordinate systems.
8  import java.awt.Color;
9  import java.awt.Font;
10 import java.awt.Graphics;
11 import java.util.Random;
12 import javax.swing.JPanel;
13
14 public class RaceJPanel extends JPanel
15 {
16    private final int RACE_END = 300;
17    private int tortoise = 1;
18    private int hare = 1;
19    private int timer = 0;
20    private Font f;
21    private Random random;
22
23    public RaceJPanel()
24    {
25       f = new Font( "Monospaced", Font.BOLD, 12 );
26       random = new Random();
```

```
27      } // end RaceJPanel constructor
28
29      public void paintComponent( Graphics g )
30      {
31         super.paintComponent( g );
32
33         g.setFont( f );
34
35         while ( tortoise != RACE_END && hare != RACE_END )
36         {
37            // slow animation
38            for ( int k = 1; k <= 100000; k++ )
39               ;   // do nothing
40
41            g.setColor( Color.WHITE );
42            g.fillRect( 0, 0, 320, 270 );  // html size
43            g.setColor( Color.BLACK );
44            g.drawArc( 0, 100, 300, 300, 0, 180 );
45
46            moveHare();
47            moveTortoise();
48            printCurrentPositions( g );
49            ++timer;
50         } // end while
51
52         if ( tortoise >= hare )
53            g.drawString( "Time: " + timer + "  TORTOISE WINS!!! YAY!!!",
54               20, 300 );
55         else
56            g.drawString( "Time: " + timer + "  Hare wins. Yuch!", 20, 300 );
57      } // end method paintComponent
58
59      public void printCurrentPositions( Graphics g2 )
60      {
61         int yHare, yTortoise;
62
63         yHare = ( int ) ( 250 - Math.sqrt( 150 * 150 -
64                               Math.pow( hare - 150, 2 ) ) );
65
66         yTortoise = ( int ) ( 250 - Math.sqrt( 150 * 150 -
67                               Math.pow( tortoise - 150, 2 ) ) );
68
69         if ( yHare == yTortoise && hare == tortoise )
70         {
71            g2.drawString( "OUCH!", hare, yHare - 60 );
72            g2.drawString( "H", hare, yHare - 20 ); // make hare jump
73         } // end if
74         else
75            g2.drawString( "H", hare, yHare );
76
77         g2.drawString( "T", tortoise, yTortoise );
78      } // end method printCurrentPositions
79
80      public void moveTortoise()
81      {
```

```
82          int x =  random.nextInt( 10 ) + 1;
83          int tortoiseMoves[] = { 3, 6 };
84
85          if ( x >= 1 && x <= 5 )             // fast plod
86             tortoise += tortoiseMoves[ 0 ];
87          else if ( x == 6 || x == 7 )        // slip
88             tortoise -= tortoiseMoves[ 1 ];
89          else                                // slow plod
90             ++tortoise;
91
92          if ( tortoise < 1 )
93             tortoise = 1;
94          else if ( tortoise > RACE_END )
95             tortoise = RACE_END;
96       } // end method moveTortoise
97
98       public void moveHare()
99       {
100         int y = random.nextInt( 10 ) + 1;
101         int hareMoves[] = { 9, 12, 2 };
102
103         if ( y == 3 || y == 4 )        // big hop
104            hare += hareMoves[ 0 ];
105         else if ( y == 5 )             // big slip
106            hare -= hareMoves[ 1 ];
107         else if ( y >= 6 && y <= 8 )   // small hop
108            ++hare;
109         else if ( y > 8 )              // small slip
110            hare -= hareMoves[ 2 ];
111
112         if ( hare < 1 )
113            hare = 1;
114         else if ( hare > RACE_END )
115            hare = RACE_END;
116      } // end method moveHare
117   } // end class RaceJPanel
```

```
1   // Exercise 12.25 Solution: Race2.java
2   // Program simulates the race between the tortoise and the hare.For
3   // graphical purposes the race has been extended to 300. The mountain is
4   // represented by a semicircle using the equation for a
5   // circle: (x-h)^2 + (y-k)^2 = r^2. A slight adjustment was made in the
6   // equation to help compensate for the different directions of the
7   // coordinate systems.
8   import javax.swing.JFrame;
9
10  public class Race2
11  {
12     public static void main( String args[] )
13     {
14        // create frame for RaceJPanel
15        JFrame frame = new JFrame( "Race" );
```

```
16         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
17
18         RaceJPanel raceJPanel = new RaceJPanel();
19         frame.add( raceJPanel ); // add raceJPanel to frame
20         frame.setSize( 400, 400 ); // set frame size
21         frame.setVisible( true ); // display frame
22      } // end main
23   } // end class Race2
```



12.26 *(Drawing Spirals)* Write an application that uses `Graphics` method `drawPolyline` to draw a spiral similar to the one shown in Fig. 12.33.

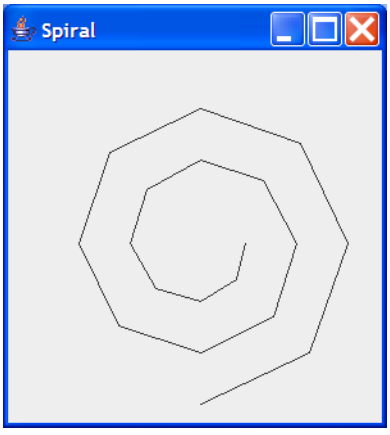
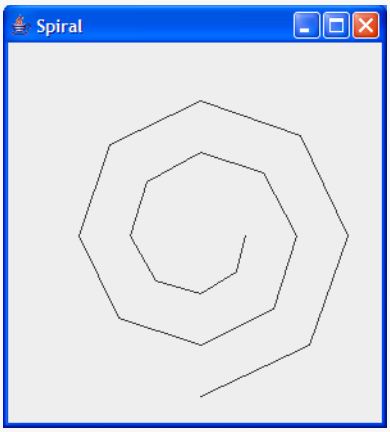


Fig. 12.33 | Spiral drawn using method `drawPolyline`.

ANS:

```
// Exercise 12.26 Solution: SpiralJPanel.java
// Program creates a spiral.
import java.awt.Graphics;
import javax.swing.JPanel;

public class SpiralJPanel extends JPanel
{
   private int[] x = { 150, 235, 265, 228, 150,  79,  55,  86,
                       150, 207, 225, 199, 150, 108,  95, 115,
                       150, 178, 185, 171, 150, 136, 135, 143,
                       150 };
   private int[] y = { 275, 235, 150,  72,  45,  79, 150, 214,
                       235, 207, 150, 101,  85, 108, 150, 185,
                       195, 178, 150, 129, 125, 136, 150, 157,
                       155 };

   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );

      g.drawPolyline( x, y, 19 );
   } // end method paintComponent
} // end class SpiralJPanel
```

```
// Exercise 12.26 Solution: Spiral.java
// Program creates a spiral.
import javax.swing.JFrame;

public class Spiral
{
   public static void main( String args[] )
   {
      // create frame for SpiralJPanel
      JFrame frame = new JFrame( "Spiral" );
      frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

      SpiralJPanel spiralJPanel = new SpiralJPanel();
      frame.add( spiralJPanel ); // add spiralJPanel to frame
      frame.setSize( 300, 330 ); // set frame size
      frame.setVisible( true ); // display frame
   } // end main
} // end class Spiral
```



12.27 *(Pie Chart)* Write a program that inputs four numbers and graphs them as a pie chart. Use class `Arc2D.Double` and method `fill` of class `Graphics2D` to perform the drawing. Draw each piece of the pie in a separate color.

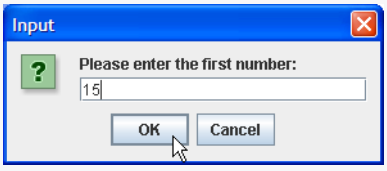
ANS:

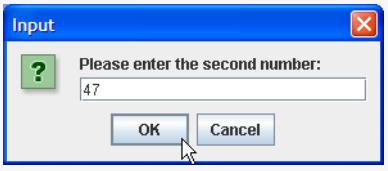
```
1  // Exercise 12.27 Solution: PieChartJPanel.java
2  // Program creates a pie chart.
3  import java.awt.Color;
4  import java.awt.geom.Arc2D;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import javax.swing.JPanel;
8
9  public class PieChartJPanel extends JPanel
10 {
11    private int one;
12    private int two;
13    private int three;
14    private int four;
15    private int sum;
16
17    // constructor gets user input
18    public PieChartJPanel( int inputOne, int inputTwo, int inputThree,
19       int inputFour )
20    {
21       one = inputOne;
22       two = inputTwo;
23       three = inputThree;
24       four = inputFour;
25       sum = one + two + three + four;
26    } // end PieChartJPanel constructor
27
28    // scale the arc by the sum of the values
```

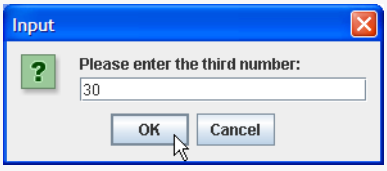
```
29     public void paintComponent( Graphics g )
30     {
31        super.paintComponent( g );
32
33        Graphics2D g2d = ( Graphics2D )g;
34
35        g2d.setColor( Color.RED );
36        g2d.fill( new Arc2D.Double( 50, 50, 200, 200, 0, 360.0 *
37           ( double )one / sum, Arc2D.PIE ) );
38
39        g2d.setColor( Color.BLUE );
40        g2d.fill( new Arc2D.Double( 50, 50, 200, 200, 360.0 *
41           ( double )one / sum, 360.0 * ( double )two / sum, Arc2D.PIE ) );
42
43        g2d.setColor( Color.GREEN );
44        g2d.fill( new Arc2D.Double( 50, 50, 200, 200, 360.0 *
45           ( double )( two + one ) / sum, 360.0 *
46           ( double )three / sum, Arc2D.PIE ) );
47
48        g2d.setColor( Color.YELLOW );
49        g2d.fill( new Arc2D.Double( 50, 50, 200, 200, 360.0 *
50           ( double )( three + two + one ) / sum, 360.0 *
51           ( double )four / sum, Arc2D.PIE ) );
52     } // end method painComponent
53  } // end class PieChartJPanel
```

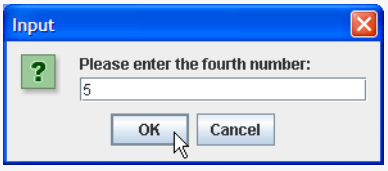
```
1   // Exercise 12.27 Solution: PieChart.java
2   // Program creates a pie chart.
3   import javax.swing.JFrame;
4   import javax.swing.JOptionPane;
5
6   public class PieChart
7   {
8      public static void main( String args[] )
9      {
10        int inputOne;
11        int inputTwo;
12        int inputThree;
13        int inputFour;
14        int inputSum;
15
16        // create frame for PieChartJPanel
17        JFrame frame = new JFrame( "PieChart" );
18        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
19
20        // input the four values
21        inputOne = Integer.parseInt( JOptionPane.showInputDialog(
22           frame, "Please enter the first number:" ) );
23        inputTwo = Integer.parseInt( JOptionPane.showInputDialog(
24           frame, "Please enter the second number:" ) );
```

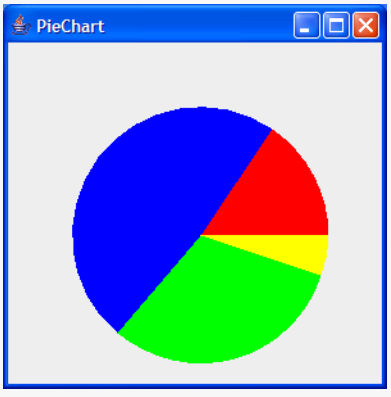
```
25         inputThree = Integer.parseInt( JOptionPane.showInputDialog(
26            frame, "Please enter the third number:" ) );
27         inputFour = Integer.parseInt( JOptionPane.showInputDialog(
28            frame, "Please enter the fourth number:" ) );
29
30         PieChartJPanel pieChartJPanel = new PieChartJPanel( inputOne,
31            inputTwo, inputThree, inputFour );
32         frame.add( pieChartJPanel ); // add pieChartJPanel to frame
33         frame.setSize( 300, 300 ); // set frame size
34         frame.setVisible( true ); // display frame
35      } // end main
36   } // end class PieChart
```











12.28 *(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 method `paintComponent`.

The first item in the `JComboBox` should be the default shape that is displayed the first time `paintComponent` is called.

ANS:

```
1  // Exercise 12.28 Solution: ShapeOption.java
2  // Defines an enum type for the program's shape options.
3
4  public enum ShapeOption
5  {
6     // declare contents of enum type
7     CIRCLE( 1 ),
8     SQUARE( 2 ),
9     OVAL( 3 ),
10    RECTANGLE( 4 );
11
12    private final int value; // current shape
13
14    ShapeOption( int valueOption )
15    {
16       value = valueOption;
17    } // end ShapeOption enum constructor
18
19    public int getValue()
20    {
21       return value;
22    } // end method getValue
23 } // end enum ShapeOption
```

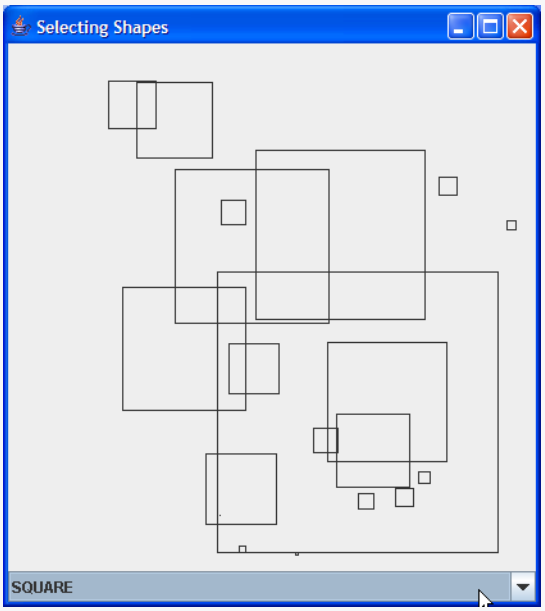
```
1  // Exercise 12.28 Solution: SelectShapeJPanel.java
2  // Draw a shape 20 times in random positions
3  import java.awt.BorderLayout;
4  import java.awt.event.ItemListener;
5  import java.awt.event.ItemEvent;
6  import java.awt.Graphics;
7  import java.util.Random;
8  import javax.swing.JComboBox;
9  import javax.swing.JPanel;
10
11 public class SelectShapeJPanel extends JPanel
12 {
13    private final int SIZE = 400;
14    private ShapeOption shape = ShapeOption.CIRCLE;
15
16    // draw the new shape in random locations 20 times
17    public void paintComponent( Graphics g )
18    {
19       super.paintComponent( g );
20       Random random = new Random(); // get random number generator
21
22       for ( int count = 1; count <= 20; count++ )
```

```
23         {
24            // add 10 and 25 to prevent drawing over edge
25            int x = ( int ) ( random.nextFloat() * SIZE ) + 10;
26            int y = ( int ) ( random.nextFloat() * SIZE ) + 25;
27            int width = ( int ) ( random.nextFloat() * ( SIZE - x ) );
28            int height = ( int ) ( random.nextFloat() * ( SIZE - y ) );
29
30            // used for circle and square, to prevent drawing off the window
31            int diameter = width;
32
33            if ( width > height )
34               diameter = height;
35
36            // draw the appropriate shape
37            switch ( shape )
38            {
39               case CIRCLE:
40                  g.drawOval( x, y, diameter, diameter );
41                  break;
42               case SQUARE:
43                  g.drawRect( x, y, diameter, diameter );
44                  break;
45               case OVAL:
46                  g.drawOval( x, y, width, height );
47                  break;
48               case RECTANGLE:
49                  g.drawRect( x, y, width, height );
50                  break;
51            } // end switch
52         } // end for
53      } // end method paintComponent
54
55      // set new shape
56      public void setShape( ShapeOption preference )
57      {
58         shape = preference;
59      } // end method setShape
60   } // end class SelectShapeJPanel
```

```
1   // Exercise 12.28 Solution: SelectShapeJFrame.java
2   // Draw a shape 20 times in random positions
3   import java.awt.BorderLayout;
4   import java.awt.event.ItemListener;
5   import java.awt.event.ItemEvent;
6   import java.awt.Graphics;
7   import javax.swing.JComboBox;
8   import javax.swing.JFrame;
9
10  public class SelectShapeJFrame extends JFrame
11  {
```

```
   private ShapeOption shapeOptions[] = { ShapeOption.CIRCLE,
      ShapeOption.SQUARE, ShapeOption.OVAL, ShapeOption.RECTANGLE };
   private ShapeOption shape = ShapeOption.CIRCLE;

   private JComboBox choiceJComboBox;
   private SelectShapeJPanel selectShapeJPanel;

   public SelectShapeJFrame()
   {
      super( "Selecting Shapes" );

      selectShapeJPanel = new SelectShapeJPanel();

      choiceJComboBox = new JComboBox( shapeOptions );
      choiceJComboBox.addItemListener(

         new ItemListener() // anonymous inner class
         {
            public void itemStateChanged( ItemEvent e )
            {
               selectShapeJPanel.setShape(
                  ( ShapeOption ) choiceJComboBox.getSelectedItem() );
               repaint();
            } // end method itemStateChanged
         } // end anonymous inner class
      ); // end call to addItemListener

      add( selectShapeJPanel, BorderLayout.CENTER );
      add( choiceJComboBox, BorderLayout.SOUTH );
   } // end SelectShapeJFrame constructor
} // end class SelectShapeJFrame
```

```
// Exercise 12.28 Solution: SelectShape.java
// Draw a shape 20 times in random positions
import javax.swing.JFrame;

public class SelectShape
{
   public static void main( String args[] )
   {
      SelectShapeJFrame selectShapeJFrame = new SelectShapeJFrame();
      selectShapeJFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      selectShapeJFrame.setSize( 420, 470 ); // set frame size
      selectShapeJFrame.setVisible( true ); // display frame
   } // end main
} // end class SelectShape
```



12.29 *(Random Colors)* Modify Exercise 12.28 to draw each of the 20 randomly sized shapes in a randomly selected color. Use all 13 predefined `Color` objects in an array of `Colors`.

ANS:

```
1  // Exercise 12.29 Solution: ShapeOption.java
2  // Defines an enum type for the program's shape options.
3
4  public enum ShapeOption
5  {
6     // declare contents of enum type
7     CIRCLE( 1 ),
8     SQUARE( 2 ),
9     OVAL( 3 ),
10    RECTANGLE( 4 );
11
12    private final int value; // current shape
13
14    ShapeOption( int valueOption )
15    {
16       value = valueOption;
17    } // end ShapeOption enum constructor
18
19    public int getValue()
20    {
21       return value;
22    } // end method getValue
23 } // end enum ShapeOption
```

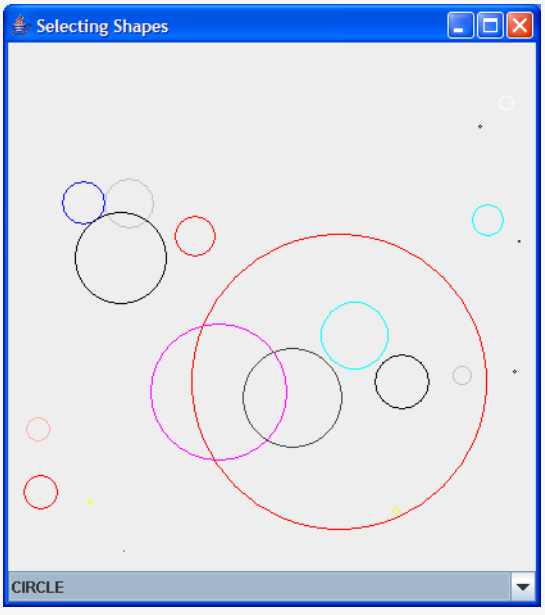
```
// Exercise 12.29 Solution: SelectShapeJPanel.java
// Draw a shape 20 times in random positions
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.Graphics;
import java.util.Random;
import javax.swing.JComboBox;
import javax.swing.JPanel;

public class SelectShapeJPanel extends JPanel
{
   private final int SIZE = 400;
   private ShapeOption shape = ShapeOption.CIRCLE;

   Color colors[] = { Color.GREEN, Color.PINK, Color.BLACK, Color.ORANGE,
      Color.MAGENTA, Color.WHITE, Color.CYAN, Color.GRAY, Color.DARK_GRAY,
      Color.BLUE, Color.YELLOW, Color.RED, Color.LIGHT_GRAY };

   // draw the new shape in random locations 20 times
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );
      Random random = new Random(); // get random number generator

      for ( int count = 1; count <= 20; count++ )
      {
         // add 10 and 25 to prevent drawing over edge
         int x = ( int ) ( random.nextFloat() * SIZE ) + 10;
         int y = ( int ) ( random.nextFloat() * SIZE ) + 25;
         int width = ( int ) ( random.nextFloat() * ( SIZE - x ) );
         int height = ( int ) ( random.nextFloat() * ( SIZE - y ) );

         // used for circle and square, to prevent drawing off the window
         int diameter = width;

         if ( width > height )
            diameter = height;

         // set random color
         int color = ( int ) ( random.nextFloat() * colors.length );
         g.setColor( colors[ color ] );

         // draw the appropriate shape
         switch ( shape )
         {
            case CIRCLE:
               g.drawOval( x, y, diameter, diameter );
               break;
            case SQUARE:
               g.drawRect( x, y, diameter, diameter );
```

```
                break;
             case OVAL:
                g.drawOval( x, y, width, height );
                break;
             case RECTANGLE:
                g.drawRect( x, y, width, height );
                break;
          } // end switch
       } // end for
    } // end method paintComponent

    // set new shape
    public void setShape( ShapeOption preference )
    {
       shape = preference;
    } // end method setShape
} // end class SelectShapeJPanel
```

```
// Exercise 12.29 Solution: SelectShapeJFrame.java
// Draw a shape 20 times in random positions
import java.awt.BorderLayout;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.Graphics;
import javax.swing.JComboBox;
import javax.swing.JFrame;

public class SelectShapeJFrame extends JFrame
{
   private ShapeOption shapeOptions[] = { ShapeOption.CIRCLE,
      ShapeOption.SQUARE, ShapeOption.OVAL, ShapeOption.RECTANGLE };
   private ShapeOption shape = ShapeOption.CIRCLE;

   private JComboBox choiceJComboBox;
   private SelectShapeJPanel selectShapeJPanel;

   public SelectShapeJFrame()
   {
      super( "Selecting Shapes" );

      selectShapeJPanel = new SelectShapeJPanel();

      choiceJComboBox = new JComboBox( shapeOptions );
      choiceJComboBox.addItemListener(

         new ItemListener() // anonymous inner class
         {
            public void itemStateChanged( ItemEvent e )
            {
               selectShapeJPanel.setShape(
```

```
                  ( ShapeOption ) choiceJComboBox.getSelectedItem() );
               repaint();
            } // end method itemStateChanged
         } // end anonymous inner class
      ); // end call to addItemListener

      add( selectShapeJPanel, BorderLayout.CENTER );
      add( choiceJComboBox, BorderLayout.SOUTH );
   } // end SelectShapeJFrame constructor
} // end class SelectShapeJFrame
```

```
// Exercise 12.29 Solution: SelectShape.java
// Draw a shape 20 times in random positions
import javax.swing.JFrame;

public class SelectShape
{
   public static void main( String args[] )
   {
      SelectShapeJFrame selectShapeJFrame = new SelectShapeJFrame();
      selectShapeJFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      selectShapeJFrame.setSize( 420, 470 ); // set frame size
      selectShapeJFrame.setVisible( true ); // display frame
   } // end main
} // end class SelectShape
```



12.30 *(`JColorChooser` Dialog)* Modify Exercise 12.28 to allow the user to select the color in which shapes should be drawn from a `JColorChooser` dialog.

ANS:

```
1  // Exercise 12.30 Solution: ShapeOption.java
2  // Defines an enum type for the program's shape options.
3
4  public enum ShapeOption
5  {
6     // declare contents of enum type
7     CIRCLE( 1 ),
8     SQUARE( 2 ),
9     OVAL( 3 ),
10    RECTANGLE( 4 );
11
12    private final int value; // current shape
13
14    ShapeOption( int valueOption )
15    {
16       value = valueOption;
17    } // end ShapeOption enum constructor
18
19    public int getValue()
20    {
21       return value;
22    } // end method getValue
23 } // end enum ShapeOption
```

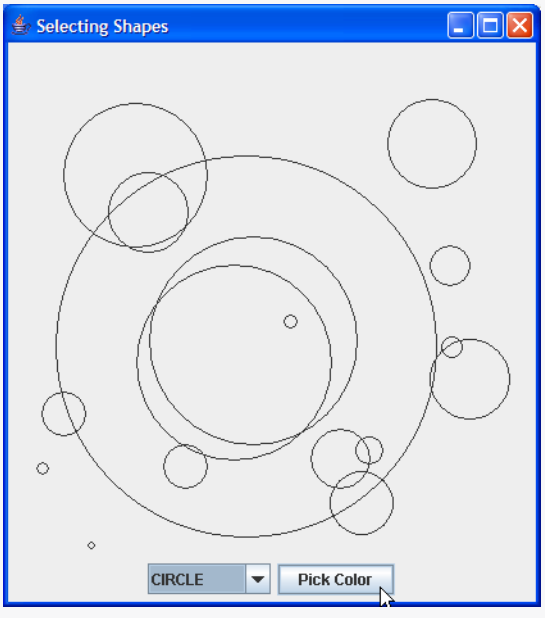
```
1  // Exercise 12.30 Solution: SelectShapeJPanel.java
2  // Draw a shape 20 times in random positions
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.event.ItemListener;
6  import java.awt.event.ItemEvent;
7  import java.awt.Graphics;
8  import java.util.Random;
9  import javax.swing.JColorChooser;
10 import javax.swing.JComboBox;
11 import javax.swing.JPanel;
12
13 public class SelectShapeJPanel extends JPanel
14 {
15    private final int SIZE = 400;
16    private ShapeOption shape = ShapeOption.CIRCLE;
17    private Color color;
18
19    // draw the new shape in random locations 20 times
20    public void paintComponent( Graphics g )
21    {
```

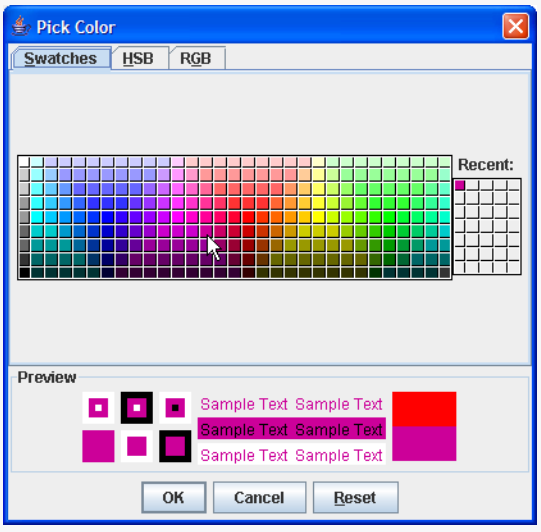
```
22          super.paintComponent( g );
23          Random random = new Random(); // get random number generator
24
25          g.setColor( color );
26
27          for ( int count = 1; count <= 20; count++ )
28          {
29             // add 10 and 25 to prevent drawing over edge
30             int x = ( int ) ( random.nextFloat() * SIZE ) + 10;
31             int y = ( int ) ( random.nextFloat() * SIZE ) + 25;
32             int width = ( int ) ( random.nextFloat() * ( SIZE - x ) );
33             int height = ( int ) ( random.nextFloat() * ( SIZE - y ) );
34
35             // used for circle and square, to prevent drawing off the window
36             int diameter = width;
37
38             if ( width > height )
39                diameter = height;
40
41             // draw the appropriate shape
42             switch ( shape )
43             {
44                case CIRCLE:
45                   g.drawOval( x, y, diameter, diameter );
46                   break;
47                case SQUARE:
48                   g.drawRect( x, y, diameter, diameter );
49                   break;
50                case OVAL:
51                   g.drawOval( x, y, width, height );
52                   break;
53                case RECTANGLE:
54                   g.drawRect( x, y, width, height );
55                   break;
56             } // end switch
57          } // end for
58       } // end method paintComponent
59
60       // set new shape
61       public void setShape( ShapeOption preference )
62       {
63          shape = preference;
64       } // end method setShape
65
66       public Color getColor()
67       {
68          return color;
69       } // end method getColor
70
71       public void setColor( Color newColor )
72       {
73          color = newColor;
74       } // end method setColor
75    } // end class SelectShapeJPanel
```

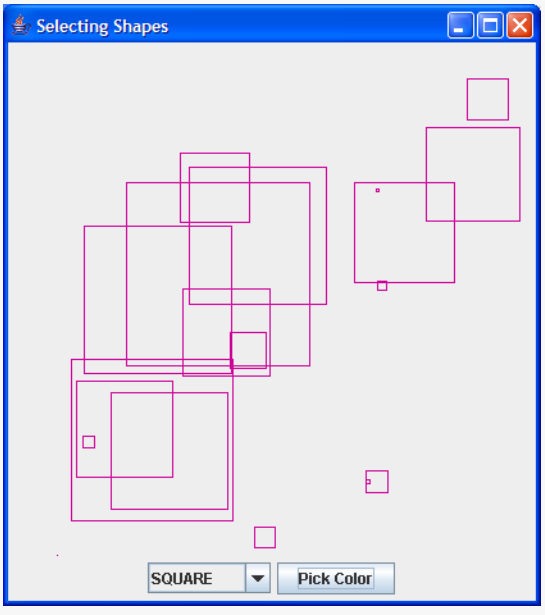
```
// Exercise 12.30 Solution: SelectShapeJFrame.java
// Draw a shape 20 times in random positions
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.Graphics;
import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class SelectShapeJFrame extends JFrame
{
   private ShapeOption shapeOptions[] = { ShapeOption.CIRCLE,
      ShapeOption.SQUARE, ShapeOption.OVAL, ShapeOption.RECTANGLE };
   private ShapeOption shape = ShapeOption.CIRCLE;
   private JButton chooseColorJButton;
   private JComboBox choiceJComboBox;
   private JPanel controlJPanel;
   private SelectShapeJPanel selectShapeJPanel;

   public SelectShapeJFrame()
   {
      super( "Selecting Shapes" );

      controlJPanel = new JPanel();
      selectShapeJPanel = new SelectShapeJPanel();

      choiceJComboBox = new JComboBox( shapeOptions );
      choiceJComboBox.addItemListener(

         new ItemListener() // anonymous inner class
         {
            public void itemStateChanged( ItemEvent e )
            {
               selectShapeJPanel.setShape(
                  ( ShapeOption ) choiceJComboBox.getSelectedItem() );
               selectShapeJPanel.repaint();
            } // end method itemStateChanged
         } // end anonymous inner class
      ); // end call to addItemListener

      // add button that pops up a color dialog
      chooseColorJButton = new JButton( "Pick Color" );
      chooseColorJButton.addActionListener(

         new ActionListener() // anonymous inner class
         {
            public void actionPerformed( ActionEvent e )
            {
```

```
                selectShapeJPanel.setColor( JColorChooser.showDialog( null,
                   "Pick Color", Color.RED ) );
                selectShapeJPanel.repaint();
             } // end method actionPerformed
          } // end anonymous inner class
       ); // end call to addActionListener

       controlJPanel.add( choiceJComboBox );
       controlJPanel.add( chooseColorJButton );

       add( selectShapeJPanel, BorderLayout.CENTER );
       add( controlJPanel, BorderLayout.SOUTH );
    } // end SelectShapeJFrame constructor
} // end class SelectShapeJFrame
```

```
// Exercise 12.30 Solution: SelectShape.java
// Draw a shape 20 times in random positions
import javax.swing.JFrame;

public class SelectShape
{
   public static void main( String args[] )
   {
      SelectShapeJFrame selectShapeJFrame = new SelectShapeJFrame();
      selectShapeJFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      selectShapeJFrame.setSize( 420, 470 ); // set frame size
      selectShapeJFrame.setVisible( true ); // display frame
   } // end main
} // end class SelectShape
```







(Optional) GUI and Graphics Case Study: Adding Java2D

12.31 Java2D introduces many new capabilities for creating unique and impressive graphics. We will add a small subset of these features to the drawing application you created in Exercise 11.18. In this version of the drawing application, you will 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 Java2D functionality. Make the following changes in class `MyShape`:

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 class `Color` implements interface `Paint`.]
c) Add an instance variable of type `Stroke` in class `MyShape` 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` and `MyRect` 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 will draw the appropriate shape using the specified `Paint` and `Stroke` settings.

Next, you will update the `DrawPanel` to handle the Java2D features. Change all `Color` variables to `Paint` variables. Declare an instance variable `currentStroke` of type `Stroke` and provide a *set* method for it. Update the calls to the individual shape constructors to include the `Paint` and `Stroke` arguments. In method `paintComponent`, cast the `Graphics` reference to type `Graphics2D` and use the `Graphics2D` reference in each call to `MyShape` method `draw`.

Next, make the new Java2D features accessible from the GUI. Create a `JPanel` of GUI components for setting the Java2D options. Add these components at the top of the `DrawFrame` below the panel that currently contains the standard shape controls (see Fig. 12.34). These GUI components should include:

a) A check box to specify whether to paint using a gradient
b) Two `JButtons` that each show a `JColorChooser` dialog to allow the user to choose the first and second color in the gradient. (These will replace the `JComboBox` used for choosing the color in Exercise 11.18.)
c) A text field for entering the `Stroke` width
d) A text field for entering the `Stroke` dash length
e) A check box for selecting whether to draw a dashed or solid line

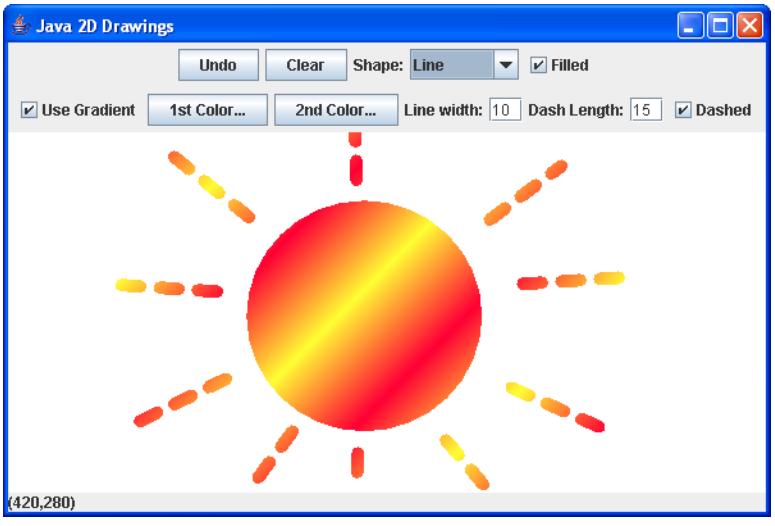


Fig. 12.34 | Drawing with Java2D.

If the user selects to draw with a gradient, set the `Paint` on the `DrawPanel` to be a gradient of the two colors chosen by the user. Use the expression

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

to create a `GradientPaint` that cycles diagonally from the upper-left to the bottom-right every 50 pixels. Variables `color1` and `color2` are the two colors chosen by the user. If the user did 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 the `Stroke` with the expression

```
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

```
new BasicStroke( width, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
   10, dashes, 0 )
```

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

ANS:

```
1  // Exercise 12.32 Solution: DrawPanel.java
2  // JPanel that allows the user to draw shapes with the mouse.
3  import java.awt.BasicStroke;
4  import java.awt.Color;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import java.awt.Paint;
8  import java.awt.Stroke;
9  import java.awt.event.MouseAdapter;
10 import java.awt.event.MouseEvent;
11 import java.awt.event.MouseMotionListener;
12 import javax.swing.JLabel;
13 import javax.swing.JPanel;
14
15 public class DrawPanel extends JPanel
16 {
17    private MyShape shapes[]; // array containing all the shapes
18    private int shapeCount; // statistic on the number of each shape
19
20    private int shapeType; // the type of shape to draw
21    private MyShape currentShape; // the current shape being drawn
22    private Paint currentPaint; // the paint of the shape
23    private Stroke currentStroke; // the stroke of the current shape
24    private boolean filledShape; // whether this shape is filled
25
26    private JLabel statusLabel; // label displaying mouse coordinates
27
28    // constructor
29    public DrawPanel( JLabel status )
```

```
30      {
31         shapes = new MyShape[ 100 ]; // create the array
32         shapeCount = 0; // initially we have no shapes
33
34         shapeType = 0; // initially draw lines
35         currentShape = null; // not drawing anything initially
36         currentPaint = Color.BLACK; // start drawing with black
37         currentStroke = new BasicStroke(); // draw simple thin lines
38         filledShape = false;// not filled by default
39
40         setBackground( Color.WHITE ); // set a white background
41
42         // add the mouse listeners
43         MouseHandler mouseHandler = new MouseHandler();
44         addMouseListener( mouseHandler );
45         addMouseMotionListener( mouseHandler );
46
47         // create a label for the status bar
48         statusLabel = status
49      } // end DrawPanel constructor
50
51      // draw shapes using polymorphism
52      public void paintComponent( Graphics g )
53      {
54         super.paintComponent( g );
55
56         Graphics2D g2d = ( Graphics2D ) g;
57
58         for ( int i = 0; i < shapeCount; i++ )
59            shapes[ i ].draw( g2d );
60
61         if ( currentShape != null )
62            currentShape.draw( g2d );
63      } // end method paintComponent
64
65      // sets the type of shape to draw
66      public void setShapeType( int shapeType )
67      {
68         if ( shapeType < 0 || shapeType > 2 )
69            shapeType = 0;
70
71         this.shapeType = shapeType;
72      } // end method setShapeType
73
74      // sets the drawing paint
75      public void setDrawingPaint( Paint paint )
76      {
77         currentPaint = paint;
78      } // end method setDrawingPaint
79
80      // sets the current drawing stroke
81      public void setDrawingStroke( Stroke stroke )
82      {
83         currentStroke = stroke;
```

```
84      } // end method setDrawingStroke
85
86      // clears the last shape drawn
87      public void clearLastShape()
88      {
89         if ( shapeCount > 0 )
90         {
91            shapeCount--;
92            repaint();
93         } // end if
94      } // end method clearLastShape
95
96      // clears all drawings on this panel
97      public void clearDrawing()
98      {
99         shapeCount = 0;
100        repaint();
101     } // end method clearDrawing
102
103     // sets whether to draw a filled shape
104     public void setFilledShape( boolean isFilled )
105     {
106        filledShape = isFilled;
107     } // end method setFilledShape
108
109     // Handles mouse events for this JPanel
110     private class MouseHandler extends MouseAdapter
111        implements MouseMotionListener
112     {
113        // creates and sets the initial position for the new shape
114        public void mousePressed( MouseEvent e )
115        {
116           if ( currentShape != null )
117              return;
118
119           // create the appropriate shape based on shapeType
120           switch ( shapeType )
121           {
122              case 0:
123                 currentShape = new MyLine( e.getX(), e.getY(), e.getX(),
124                    e.getY(), currentPaint, currentStroke );
125                 break;
126              case 1:
127                 currentShape = new MyOval( e.getX(), e.getY(), e.getX(),
128                  e.getY(), currentPaint, currentStroke, filledShape );
129                 break;
130              case 2:
131                 currentShape = new MyRect( e.getX(), e.getY(), e.getX(),
132                  e.getY(), currentPaint, currentStroke, filledShape );
133                 break;
134           } // end switch
135        } // end method mousePressed
136
137        // fixes the current shape onto the panel
```

```
138        public void mouseReleased( MouseEvent e )
139        {
140           if ( currentShape == null )
141              return;
142
143           // set the second point on the shape
144           currentShape.setX2( e.getX() );
145           currentShape.setY2( e.getY() );
146
147           // only set the shape if there is room in the array
148           if ( shapeCount < shapes.length )
149           {
150              shapes[ shapeCount ] = currentShape;
151              shapeCount++;
152           } // end if
153
154           currentShape = null; // clear the temporary drawing shape
155           repaint();
156        } // end method mouseReleased
157
158        // update the shape to the current mouse position while dragging
159        public void mouseDragged( MouseEvent e )
160        {
161           if ( currentShape != null )
162           {
163              currentShape.setX2( e.getX() );
164              currentShape.setY2( e.getY() );
165              repaint();
166           } // end if
167
168           mouseMoved( e ); // update status bar
169        } // end method mouseDragged
170
171        // updates the status bar to show the current mouse coordinates
172        public void mouseMoved( MouseEvent e )
173        {
174           statusLabel.setText(
175              String.format( "(%d,%d)", e.getX(), e.getY() ) );
176        } // end method mouseMoved
177     } // end class MouseHandler
178  } // end class DrawPanel
```

```
1  // Exercise 12.32 Solution: DrawFrame.java
2  // Program that creates a panel for the user to draw shapes.
3  // Allows the user to choose the shape set various java2d options
4  import java.awt.BasicStroke;
5  import java.awt.BorderLayout;
6  import java.awt.Color;
7  import java.awt.FlowLayout;
8  import java.awt.GradientPaint;
```

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JColorChooser;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class DrawFrame extends JFrame
   implements ItemListener, ActionListener
{
   // Array of possible shapes
   private String shapes[] = { "Line", "Oval", "Rectangle" };

   private DrawPanel drawPanel; // the panel that handles the drawing

   private JButton undoButton; // button to undo the last shape drawn
   private JButton clearButton; // button to clear all shapes
   private JComboBox shapeChoices; // combo box for selecting shapes
   private JCheckBox filledCheckBox; // check box to toggle filled shapes

   private JButton color1Button; // button to choose the first color
   private JButton color2Button; // button to choose the second color
   private JCheckBox gradientCheckBox; // check box to toggle gradients

   private JTextField lineWidthField; // text field for the line width
   private JTextField dashLengthField; // text field for the dash length
   private JCheckBox dashedCheckBox; // check box to toggle dashed lines

   private Color color1; // first color
   private Color color2; // second color only used in the gradient

   // constructor, sets up the components in the frame
   public DrawFrame()
   {
      super( "Java 2D Drawings" ); // set the title by calling super

      // create the top panel that contains all the drawing options
      JPanel topPanel = new JPanel( new GridLayout( 2, 1 ) );

      // add the top panel to the frame
      topPanel.add( createStandardPanel() );
      topPanel.add( createJava2dPanel() );
      add( topPanel, BorderLayout.NORTH );

      // create a label for the status bar
      JLabel statusLabel = new JLabel( "(0,0)" );

```

```
63          // add the status bar at the bottom
64          add( statusLabel, BorderLayout.SOUTH );
65
66          // create the DrawPanel with its status bar label
67          drawPanel = new DrawPanel( statusLabel );
68
69          add( drawPanel ); // add the drawing area to the center
70
71          // set initial drawing settings
72          setDrawingPaint();
73          setDrawingStroke();
74       } // end DrawFrame constructor
75
76       // create the JPanel with components for the standard drawing options
77       private JPanel createStandardPanel()
78       {
79          // create a JPanel
80          JPanel standardOptions = new JPanel( new FlowLayout() );
81
82          // create a button for clearing the last drawing
83          undoButton = new JButton( "Undo" );
84          undoButton.addActionListener( this );
85          standardOptions.add( undoButton );
86
87          // create a button for clearing all drawings
88          clearButton = new JButton( "Clear" );
89          clearButton.addActionListener( this );
90          standardOptions.add( clearButton );
91
92          // create a combo-box for choosing shapes
93          standardOptions.add( new JLabel( "Shape:" ) );
94          shapeChoices = new JComboBox( shapes );
95          shapeChoices.addItemListener( this );
96          standardOptions.add( shapeChoices );
97
98          // create a check-box to determine whether the shape is filled
99          filledCheckBox = new JCheckBox( "Filled" );
100         filledCheckBox.addItemListener( this );
101         standardOptions.add( filledCheckBox );
102
103         return standardOptions;
104      } // end method createStandardPanel
105
106      // create the JPanel with components for the java 2d options
107      private JPanel createJava2dPanel()
108      {
109         // create a JPanel
110         JPanel java2dOptions = new JPanel( new FlowLayout() );
111
112         // create a check-box to determine whether a gradient is used
113         gradientCheckBox = new JCheckBox( "Use Gradient" );
114         gradientCheckBox.addItemListener( this );
115         java2dOptions.add( gradientCheckBox );
116
```

```
117         // create a combo-box for choosing the first color
118         color1Button = new JButton( "1st Color..." );
119         color1Button.addActionListener( this );
120         color1 = Color.BLACK;
121         java2dOptions.add( color1Button );
122
123         // create a combo-box for choosing the second color
124         color2Button = new JButton( "2nd Color..." );
125         color2Button.addActionListener( this );
126         color2 = Color.BLACK;
127         java2dOptions.add( color2Button );
128
129         // create a textfield for entering in line width
130         java2dOptions.add( new JLabel( "Line width:" ) );
131         lineWidthField = new JTextField( "1.0", 2 );
132         lineWidthField.addActionListener( this );
133         java2dOptions.add( lineWidthField );
134
135         // create a textfield for entereing in dash length
136         java2dOptions.add( new JLabel( "Dash Length:" ) );
137         dashLengthField = new JTextField( "10", 2 );
138         dashLengthField.addActionListener( this );
139         java2dOptions.add( dashLengthField );
140
141         // create a check box to determine whether the Stroke is dashed
142         dashedCheckBox = new JCheckBox( "Dashed" );
143         dashedCheckBox.addItemListener( this );
144         java2dOptions.add( dashedCheckBox );
145
146         return java2dOptions;
147      } // end method createJava2dPanel
148
149      // handle selections made to a combo box or check box
150      public void itemStateChanged( ItemEvent e )
151      {
152         if ( e.getSource() == shapeChoices ) // choosing a shape
153            drawPanel.setShapeType( shapeChoices.getSelectedIndex() );
154         else if ( e.getSource() == filledCheckBox ) // filled/unfilled
155            drawPanel.setFilledShape( filledCheckBox.isSelected() );
156         else if ( e.getSource() == gradientCheckBox ) // use/no gradient
157            setDrawingPaint();
158         else if ( e.getSource() == dashedCheckBox ); // dashed/solid
159            setDrawingStroke();
160      } // end method itemStateChanged
161
162      // handle button clicks
163      public void actionPerformed( ActionEvent e )
164      {
165         if ( e.getSource() == undoButton ) // undo last shape
166            drawPanel.clearLastShape();
167         else if ( e.getSource() == clearButton ) // clear all shapes
168            drawPanel.clearDrawing();
169         else if ( e.getSource() == color1Button ) // set 1st color
170         {
```

```
171          Color temp =
172             JColorChooser.showDialog( this, "Choose a color", color1 );
173
174          if ( temp != null ) // check if the user pressed cancel
175             color1 = temp;
176
177          setDrawingPaint();
178       } // end else if
179       else if ( e.getSource() == color2Button ) // set 2nd color
180       {
181          Color temp =
182             JColorChooser.showDialog( this, "Choose a color", color2 );
183
184          if ( temp != null ) // check if the user pressed cancel
185             color2 = temp;
186
187          setDrawingPaint();
188       } // end else if
189       else if ( e.getSource() == lineWidthField ) // set line width
190          setDrawingStroke();
191       else if ( e.getSource() == dashLengthField ) // set dash length
192          setDrawingStroke();
193    } // end method actionPerformed
194
195    // sets the Paint for the drawing panel
196    private void setDrawingPaint()
197    {
198       if ( gradientCheckBox.isSelected() )
199          drawPanel.setDrawingPaint(
200             new GradientPaint( 0, 0, color1, 50, 50, color2, true ) );
201       else // use color1 if there is no gradient
202          drawPanel.setDrawingPaint( color1 );
203    } // end method setDrawingStroke
204
205    // sets the Stroke for the current drawing panel
206    private void setDrawingStroke()
207    {
208       float width = Float.parseFloat( lineWidthField.getText() );
209
210       if ( dashedCheckBox.isSelected() )
211       {
212          float dashes[] =
213             { Float.parseFloat( dashLengthField.getText() ) };
214
215          drawPanel.setDrawingStroke( new BasicStroke(
216             width, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
217             10, dashes, 0 ) );
218       } // end if
219       else
220          drawPanel.setDrawingStroke(new BasicStroke(
221             width, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND ) );
222    } // end method setDrawingStroke
223 } // end class DrawFrame
```

```
// Exercise 12.32 Solution: MyShape.java
// Declaration of class MyShape.
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.Stroke;

public abstract class MyShape
{
   private int x1; // x coordinate of first endpoint
   private int y1; // y coordinate of first endpoint
   private int x2; // x coordinate of second endpoint
   private int y2; // y coordinate of second endpoint
   private Paint myPaint; // color of this shape
   private Stroke myStroke; // stroke used to draw this shape

   // default constructor initializes values with 0
   public MyShape()
   {
      this( 0, 0, 0, 0, Color.BLACK, new BasicStroke() ); // set values
   } // end MyShape no-argument constructor

   // constructor
   public MyShape( int x1, int y1, int x2, int y2,
      Paint paint, Stroke stroke )
   {
      setX1( x1 ); // set x coordinate of first endpoint
      setY1( y1 ); // set y coordinate of first endpoint
      setX2( x2 ); // set x coordinate of second endpoint
      setY2( y2 ); // set y coordinate of second endpoint
      setPaint( paint ); // set the color
      setStroke( stroke );
   } // end MyShape constructor

   // set the x-coordinate of the first point
   public void setX1( int x1 )
   {
     this.x1 = ( x1 >= 0 ? x1 : 0 );
   } // end method setX1

   // get the x-coordinate of the first point
   public int getX1()
   {
     return x1;
   } // end method getX1

   // set the x-coordinate of the second point
   public void setX2( int x2 )
   {
     this.x2 = ( x2 >= 0 ? x2 : 0 );
   } // end method setX2
```

```
53
54      // get the x-coordinate of the second point
55      public int getX2()
56      {
57        return x2;
58      } // end method getX2
59
60      // set the y-coordinate of the first point
61      public void setY1( int y1 )
62      {
63        this.y1 = ( y1 >= 0 ? y1 : 0 );
64      } // end method setY1
65
66      // get the y-coordinate of the first point
67      public int getY1()
68      {
69        return y1;
70      } // end method getY1
71
72      // set the y-coordinate of the second point
73      public void setY2( int y2 )
74      {
75        this.y2 = ( y2 >= 0 ? y2 : 0 );
76      } // end method setY2
77
78      // get the y-coordinate of the second point
79      public int getY2()
80      {
81        return y2;
82      } // end method getY2
83
84      // set the color
85      public void setPaint( Paint paint )
86      {
87         myPaint = paint;
88      } // end method setColor
89
90      // get the color
91      public Paint getPaint()
92      {
93         return myPaint;
94      } // end method getColor
95
96      // sets the stroke used to draw this shape
97      public void setStroke( Stroke stroke )
98      {
99         myStroke = stroke;
100     } // end method setStroke
101
102     // get the stroke used to draw this shape
103     public Stroke getStroke()
104     {
105        return myStroke;
106     } // end method getStroke
```

```
107
108    // abstract draw method
109    public abstract void draw( Graphics2D g2d );
110 } // end class MyShape
```

```
 1  // Exercise 12.32 Solution: MyLine.java
 2  // Declaration of class MyLine.
 3  import java.awt.Paint;
 4  import java.awt.Graphics2D;
 5  import java.awt.Stroke;
 6
 7  public class MyLine extends MyShape
 8  {
 9     // call default superclass constructor
10     public MyLine()
11     {
12        super();
13     } // end MyLine no-argument constructor
14
15     // call superclass constructor passing parameters
16     public MyLine( int x1, int y1, int x2, int y2,
17        Paint paint, Stroke stroke )
18     {
19        super( x1, y1, x2, y2, paint, stroke );
20     } // end MyLine constructor
21
22     // draw line in specified color
23     public void draw( Graphics2D g2d )
24     {
25        g2d.setPaint( getPaint() );
26        g2d.setStroke( getStroke() );
27        g2d.drawLine( getX1(), getY1(), getX2(), getY2() );
28     } // end method draw
29  } // end class MyLine
```

```
 1  // Exercise 12.32 Solution: MyBoundedShape.java
 2  // Declaration of class MyBoundedShape.
 3  import java.awt.Paint;
 4  import java.awt.Stroke;
 5
 6  public abstract class MyBoundedShape extends MyShape
 7  {
 8     private boolean filled; // whether this shape is filled
 9
10     // call default superclass constructor
11     public MyBoundedShape()
12     {
```

```
      super();
      setFilled( false );
   } // end MyBoundedShape no-argument constructor

   // call superclass constructor passing parameters
   public MyBoundedShape( int x1, int y1, int x2, int y2,
      Paint paint, Stroke stroke, boolean filled )
   {
      super( x1, y1, x2, y2, paint, stroke );
      setFilled( filled );
   } // end MyBoundedShape constructor

   // get upper left x coordinate
   public int getUpperLeftX()
   {
      return Math.min( getX1(), getX2() );
   } // end method getUpperLeftX

   // get upper left y coordinate
   public int getUpperLeftY()
   {
      return Math.min( getY1(), getY2() );
   } // end method getUpperLeftY

   // get shape width
   public int getWidth()
   {
      return Math.abs( getX2() - getX1() );
   } // end method getWidth

   // get shape height
   public int getHeight()
   {
      return Math.abs( getY2() - getY1() );
   } // end method getHeight

   // whether this shape is filled
   public boolean isFilled()
   {
      return filled;
   } // end method isFilled

   // set whether this shape is filled
   public void setFilled( boolean isFilled )
   {
      filled = isFilled;
   } // end method setFilled
} // end class MyBoundedShape
```

```
// Exercise 12.32 Solution: MyOval.java
// Declaration of class MyOval.
import java.awt.Graphics2D;
```

```
4  import java.awt.Paint;
5  import java.awt.Stroke;
6
7  public class MyOval extends MyBoundedShape
8  {
9     // call default superclass constructor
10    public MyOval()
11    {
12       super();
13    } // end MyOval no-argument constructor
14
15    // call superclass constructor passing parameters
16    public MyOval( int x1, int y1, int x2, int y2,
17       Paint paint, Stroke stroke, boolean filled )
18    {
19       super( x1, y1, x2, y2, paint, stroke, filled );
20    } // end MyOval constructor
21
22    // draw oval
23    public void draw( Graphics2D g2d )
24    {
25       g2d.setPaint( getPaint() );
26       g2d.setStroke( getStroke() );
27
28       if ( isFilled() )
29          g2d.fillOval( getUpperLeftX(), getUpperLeftY(),
30             getWidth(), getHeight() );
31       else
32          g2d.drawOval( getUpperLeftX(), getUpperLeftY(),
33             getWidth(), getHeight() );
34    } // end method draw
35 } // end class MyOval
```

```
1  // Exercise 12.32 Solution: MyRect.java
2  // Declaration of class MyRect.
3  import java.awt.Graphics2D;
4  import java.awt.Paint;
5  import java.awt.Stroke;
6
7  public class MyRect extends MyBoundedShape
8  {
9     // call default superclass constructor
10    public MyRect()
11    {
12       super();
13    } // end MyRect no-argument constructor
14
15    // call superclass constructor passing parameters
16    public MyRect( int x1, int y1, int x2, int y2,
17       Paint paint, Stroke stroke, boolean filled )
18    {
```

```
19          super( x1, y1, x2, y2, paint, stroke, filled );
20       } // end MyRect constructor
21
22       // draw rectangle
23       public void draw( Graphics2D g2d )
24       {
25          g2d.setPaint( getPaint() );
26          g2d.setStroke( getStroke() );
27
28          if ( isFilled() )
29             g2d.fillRect( getUpperLeftX(), getUpperLeftY(),
30                getWidth(), getHeight() );
31          else
32             g2d.drawRect( getUpperLeftX(), getUpperLeftY(),
33                getWidth(), getHeight() );
34       } // end method draw
35    } // end class MyRect
```

```
1  // Exercise 12.32 Solution: TestDraw.java
2  // Test application to display a DrawFrame
3  import javax.swing.JFrame;
4
5  public class TestDraw
6  {
7     public static void main( String args[] )
8     {
9        DrawFrame application = new DrawFrame();
10       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       application.setSize( 600, 400 );
12       application.setVisible( true );
13    } // end main
14 } // end class TestDraw
```