

1

2

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 understand and be able to manipulate colors.**
- **To understand and be able 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 be able to specify `Paint` and `Stroke` characteristics of shapes displayed with `Graphics2D`.**



- 12.1 Introduction**
- 12.2 Graphics Contexts and Graphics Objects**
- 12.3 Color Control**
- 12.4 Font Control**
- 12.5 Drawing Lines, Rectangles and Ovals**
- 12.6 Drawing Arcs**
- 12.7 Drawing Polygons and Polylines**
- 12.8 Java 2D API**
- 12.9 Wrap-Up**



12.1 Introduction

- **Java contains support for graphics that enable programmers to visually enhance applications**
- **Java contains many more sophisticated drawing capabilities as part of the Java 2D API**
- **Classes**
 - **Color**
 - **Font, FontMetrics**
 - **Graphics2D**
 - **Polygon**
 - **BasicStroke**
 - **GradientPaint, TexturePaint**
 - **Java 2D shape classes**



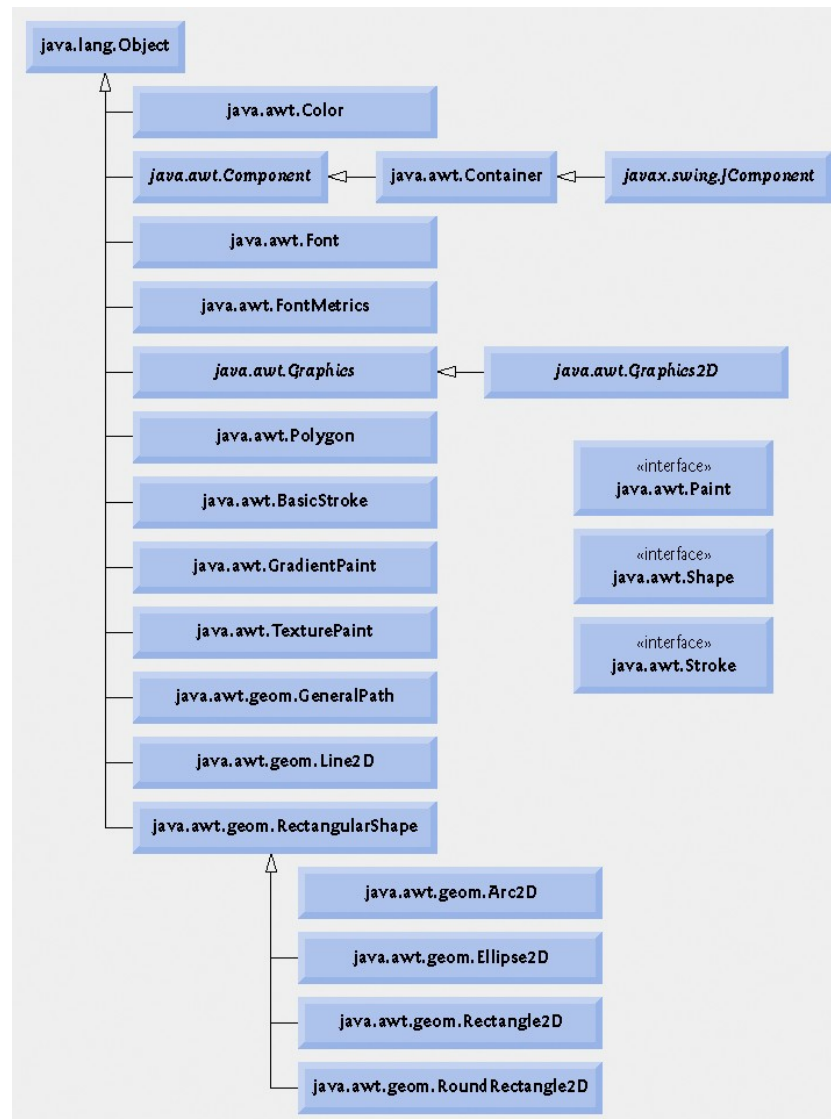


Fig. 12.1 | Classes and interfaces used in this chapter from Java's original graphics capabilities and from the Java 2D API. [Note: Class `Object` appears here because it is the superclass of the Java class hierarchy.]



12.1 Introduction

- **Java coordinate system**
 - Upper-left corner of a GUI component has the coordinates (0, 0)
 - Contains x-coordinate (horizontal coordinate) - horizontal distance moving right from the left of the screen
 - Contains y-coordinate (vertical coordinate) - vertical distance moving down from the top of the screen
- **Coordinate units are measured in pixels. A pixel is a display monitor's smallest unit of resolution.**



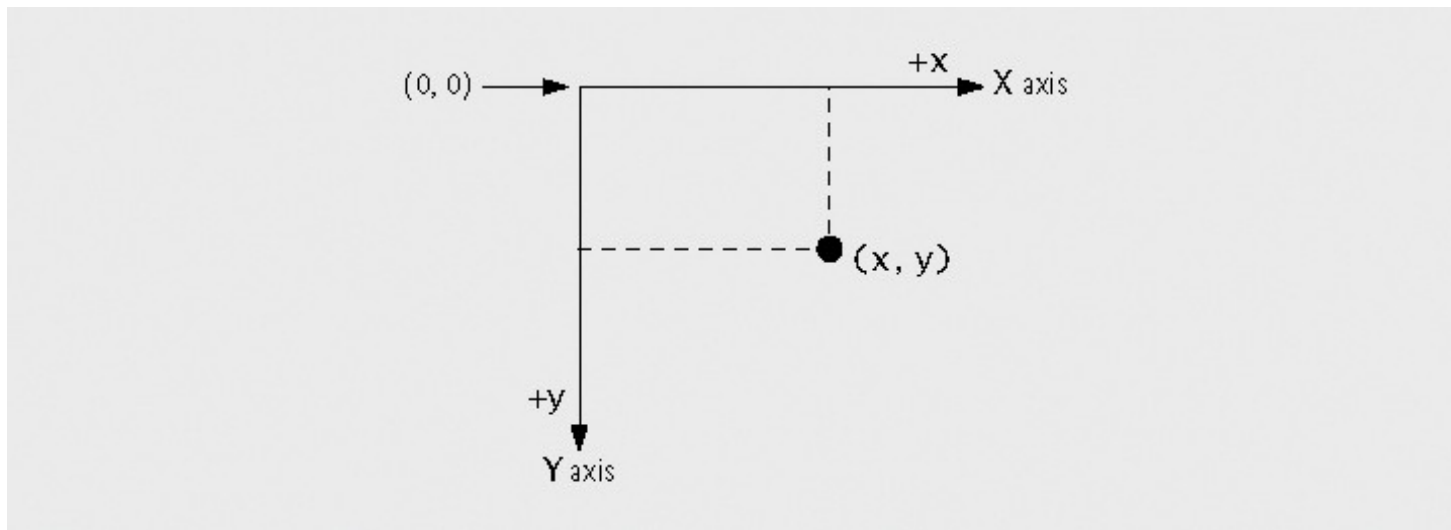


Fig. 12.2 | Java coordinate system. Units are measured in pixels.

Portability Tip 12.1

Different display monitors have different resolutions (i.e., the density of the pixels varies). This can cause graphics to appear to be different sizes on different monitors or on the same monitor with different settings.



12.2 Graphics Contexts and Graphics Objects

- A Java graphics context enables drawing on the screen
- Class **Graphics**
 - Manages a graphics context and draws pixels on the screen
 - An **abstract** class – contributes to Java's portability
- Method **paintComponent**
 - Used to draw graphics
 - Member of class **JComponent**, subclass of **Component**
 - **Graphics** object passed to **paintComponent** by the system when a lightweight Swing component needs to be repainted
 - If programmer needs to have **paintComponent** execute, a call is made to method **repaint**



12.3 Color Control

- **Class Color declares methods and constants for manipulating colors in a Java program**
- **Every color is created from a red, a green and a blue component – RGB values**



Color constant	Color	RGB value
<code>public final static Color RED</code>	red	255, 0, 0
<code>public final static Color GREEN</code>	green	0, 255, 0
<code>public final static Color BLUE</code>	blue	0, 0, 255
<code>public final static Color ORANGE</code>	orange	255, 200, 0
<code>public final static Color PINK</code>	pink	255, 175, 175
<code>public final static Color CYAN</code>	cyan	0, 255, 255
<code>public final static Color MAGENTA</code>	magenta	255, 0, 255
<code>public final static Color YELLOW</code>	yellow	255, 255, 0
<code>public final static Color BLACK</code>	black	0, 0, 0
<code>public final static Color WHITE</code>	white	255, 255, 255
<code>public final static Color GRAY</code>	gray	128, 128, 128
<code>public final static Color LIGHT_GRAY</code>	light gray	192, 192, 192
<code>public final static Color DARK_GRAY</code>	dark gray	64, 64, 64

Fig. 12.3 | Color constants and their RGB values.



Method	Description
<i>Color constructors and methods</i>	
<code>public Color(int r, int g, int b)</code>	Creates a color based on red, green and blue components expressed as integers from 0 to 255.
<code>public Color(float r, float g, float b)</code>	Creates a color based on red, green and blue components expressed as floating-point values from 0.0 to 1.0.
<code>public int getRed()</code>	Returns a value between 0 and 255 representing the red content.
<code>public int getGreen()</code>	Returns a value between 0 and 255 representing the green content.
<code>public int getBlue()</code>	Returns a value between 0 and 255 representing the blue content.
<i>Graphics methods for manipulating Colors</i>	
<code>public Color getColor()</code>	Returns Color object representing current color for the graphics context.
<code>public void setColor(Color c)</code>	Sets the current color for drawing with the graphics context.

Fig. 12.4 | Color methods and color-related Graphics methods.



Outline

ColorJPanel.java

```

1 // Fig. 12.5: ColorJPanel.java
2 // Demonstrating Colors.
3 import java.awt.Graphics;
4 import java.awt.Color;
5 import javax.swing.JPanel;
6
7 public class ColorJPanel extends JPanel
8 {
9     // draw rectangles and Strings in different colors
10    public void paintComponent( Graphics g )
11    {
12        super.paintComponent( g ); // call superclass's paintComponent
13
14        this.setBackground( Color.WHITE );
15
16        // set new drawing color using integers
17        g.setColor( new Color( 255, 0, 0 ) );
18        g.fillRect( 15, 25, 100, 20 );
19        g.drawString( "Current RGB: " + g.getColor(), 130, 40 );
20
21        // set new drawing color using floats
22        g.setColor( new Color( 0.50f, 0.75f, 0.0f ) );
23        g.fillRect( 15, 50, 100, 20 );
24        g.drawString( "Current RGB: " + g.getColor(), 130, 65 );
25
26        // set new drawing color using static Color objects
27        g.setColor( Color.BLUE );
28        g.fillRect( 15, 75, 100, 20 );
29        g.drawString( "Current RGB: " + g.getColor(), 130, 90 );
30    }

```

Method paintComponent paints JPanel 2)

Set current drawing color with method setColor

Draw filled rectangle using current color

Draw text value of current color

Set current drawing color, specify float arguments to Color

Set current drawing color using Color constant



Outline

ColorJPanel.java

(2 of 2)

```
31 // display individual RGB values
32 Color color = Color.MAGENTA;
33 g.setColor( color );
34 g.fillRect( 15, 100, 100, 20 );
35 g.drawString( "RGB values: " + color.getRed() + ", " +
36             color.getGreen() + ", " + color.getBlue(), 130, 115 );
37 } // end method paintComponent
38 } // end class ColorJPanel
```

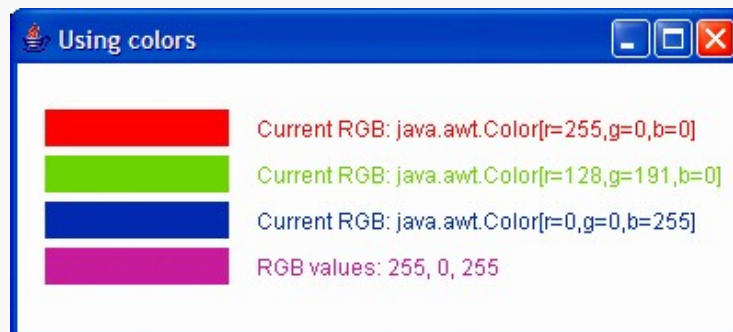
Retrieving RGB values using
methods `getRed`, `getGreen` and
`getBlue`



Outline

ShowColors.java

```
1 // Fig. 12.6: ShowColors.java
2 // Demonstrating Colors.
3 import javax.swing.JFrame;
4
5 public class ShowColors
6 {
7     // execute application
8     public static void main( String args[] )
9     {
10         // create frame for ColorJPanel
11         JFrame frame = new JFrame( "Using colors" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14         ColorJPanel colorJPanel = new ColorJPanel(); // create ColorJPanel
15         frame.add( colorJPanel ); // add colorJPanel to frame
16         frame.setSize( 400, 180 ); // set frame size
17         frame.setVisible( true ); // displayt frame
18     } // end main
19 } // end class ShowColors
```



Look-and-Feel Observation 12.1

Everyone perceives colors differently. Choose your colors carefully to ensure that your application is readable. Try to avoid using many different colors in close proximity.



Software Engineering Observation 12.1

To change the color, you must create a new Color object (or use one of the predeclared Color constants). Like String objects, Color objects are immutable (not modifiable).



12.3 Color Control

- **JColorChooser** GUI component enables application users to select colors
 - Method **showDialog** creates a **JColorChooser** object, attaches it to a dialog box and displays the dialog
 - Modal dialog
 - Allows the user to select a color from a variety of color swatches
 - Tabs – Swatches, HSB and RGB



Outline


ShowColors2JFrame

.java

(1 - 10)

```
1 // Fig. 12.7: ShowColors2JFrame.java
2 // Choosing colors with JColorChooser.
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import javax.swing.JButton;
8 import javax.swing.JFrame;
9 import javax.swing.JColorChooser;
10 import javax.swing.JPanel;
11
12 public class ShowColors2JFrame extends JFrame
13 {
14     private JButton changeColorJButton;
15     private Color color = Color.LIGHT_GRAY;
16     private JPanel colorJPanel;
17
18     // set up GUI
19     public ShowColors2JFrame()
20     {
21         super( "Using JColorChooser" );
22
23         // create JPanel for display color
24         colorJPanel = new JPanel();
25         colorJPanel.setBackground( color );
26
27         // set up changeColorJButton and register its event handler
28         changeColorJButton = new JButton( "Change Color" );
29         changeColorJButton.addActionListener(
30
```

Import JColorChooser class



Outline

```

31 new ActionListener() // anonymous inner class
32 {
33     // display JColorChooser when user clicks button
34     public void actionPerformed((ActionEvent ev)
35     {
36         color = JColorChooser.showDialog(
37             ShowColors2JFrame.this, "Choose a color", color );
38
39         // set default color, if no color is returned
40         color = Color.LIGHT_GRAY;
41
42         // change content pane's background color
43         colorJPanel.setBackground( color );
44     } // end method actionPerformed
45 } // end anonymous inner class
46 ); // end call to addActionListener
47
48
49 add( colorJPanel, BorderLayout.CENTER ); // add colorJPanel
50 add( changeColorJButton, BorderLayout.SOUTH ); // add button
51
52 setSize( 400, 130 ); // set frame size
53 setVisible( true ); // display frame
54 } // end ShowColor2JFrame constructor
55 } // end class ShowColors2JFrame

```

Diagram annotations:

- Display JColorChooser dialog**: Points to the `JColorChooser.showDialog` call in line 36.
- Reference to parent component**: Points to `ShowColors2JFrame.this` in line 36.
- Title bar text**: Points to the string `"Choose a color"` in line 36.
- Initial selected color**: Points to the `color` parameter in line 36.
- Change background color of JPanel**: Points to the `colorJPanel.setBackground(color);` call in line 43.

File information: ShowColors2JFrame.java



Outline

ShowColors2.java

(1 of 2)

```
1 // Fig. 12.8: ShowColors2.java
2 // Choosing colors with JColorChooser.
3 import javax.swing.JFrame;
4
5 public class ShowColors2
6 {
7     // execute application
8     public static void main( String args[] )
9     {
10         ShowColors2JFrame application = new ShowColors2JFrame();
11         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
12     } // end main
13 } // end class ShowColors2
```

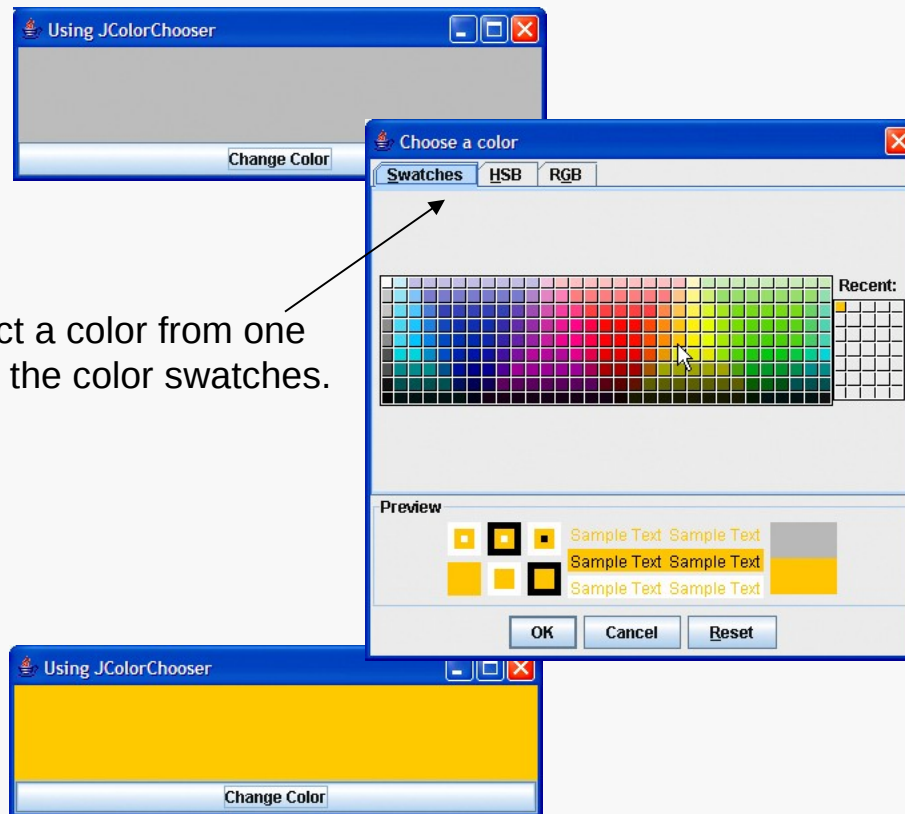


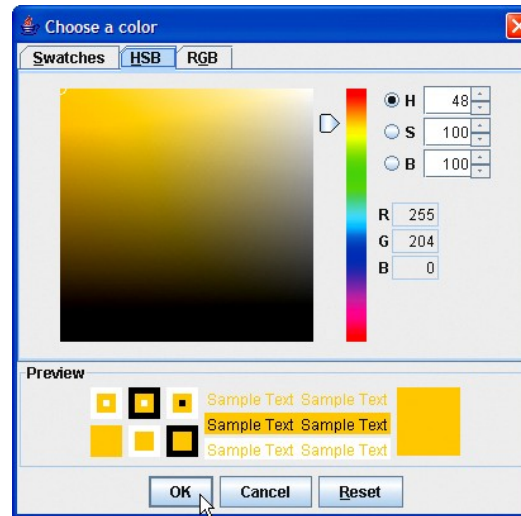
Outline

ShowColors2.java

(2 of 2)

Select a color from one of the color swatches.





Sliders to select
the red, green
and blue color
components

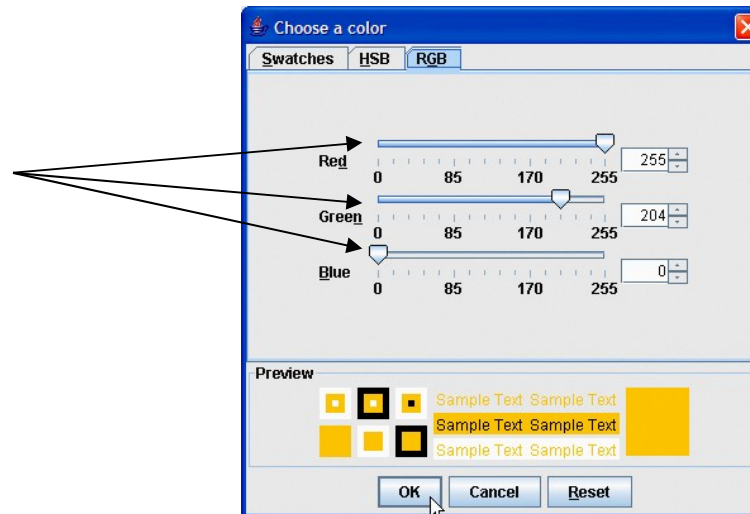


Fig. 12.9 | HSB and RGB tabs of the JColorChooser dialog.

12.4 Font Control

- **Class Font**

- **Constructor takes three arguments—the font name, font style and font size**
 - **Font name** – any font currently supported by the system on which the program is running
 - **Font style** – **Font.PLAIN**, **Font.ITALIC** or **Font.BOLD**. Font styles can be used in combination
 - **Font sizes** – measured in points. A point is 1/72 of an inch.
- **Methods `getName`, `getStyle` and `getSize` retrieve information about Font object**
- **Graphics methods `getFont` and `setFont` retrieve and set the current font, respectively**



Method or constant	Description
Font constants, constructors and methods	
<code>public final static int PLAIN</code>	A constant representing a plain font style.
<code>public final static int BOLD</code>	A constant representing a bold font style.
<code>public final static int ITALIC</code>	A constant representing an italic font style.
<code>public Font(String name, int style, int size)</code>	Creates a Font object with the specified font name, style and size.
<code>public int getStyle()</code>	Returns an integer value indicating the current font style.
<code>public int getSize()</code>	Returns an integer value indicating the current font size.

Fig. 12.10 | Font-related methods and constants.
(Part 1 of 2)



Method or constant	Description
<code>public String getName()</code>	Returns the current font name as a string.
<code>public String getFamily()</code>	Returns the font's family name as a string.
<code>public boolean isPlain()</code>	Returns true if the font is plain, else false.
<code>public boolean isBold()</code>	Returns true if the font is bold, else false.
<code>public boolean isItalic()</code>	Returns true if the font is italic, else false.
<i>Graphics methods for manipulating Fonts</i>	
<code>public Font getFont()</code>	Returns a Font object reference representing the current font.
<code>public void setFont(Font f)</code>	Sets the current font to the font, style and size specified by the Font object reference f.

**Fig. 12.10 | Font-related methods and constants.
(Part 2 of 2)**



Portability Tip 12.2

The number of fonts varies greatly across systems. Java provides five logical font names—`Serif`, `Monospaced`, `SansSerif`, `Dialog` and `DialogInput`—that can be used on all Java platforms. The Java runtime environment (JRE) on each platform maps these logical font names to actual fonts installed on the platform. The actual fonts used may vary by platform.



Outline

FontJPanel.java

(1 of 2)

```

1  // Fig. 12.11: FontJPanel.java
2  // Display strings in different fonts and colors.
3  import java.awt.Font;
4  import java.awt.Color;
5  import java.awt.Graphics;
6  import javax.swing.JPanel;
7
8  public class FontJPanel extends JPanel
9  {
10     // display Strings in different fonts and colors
11     public void paintComponent( Graphics g )
12     {
13         super.paintComponent( g ); // call superclass's paintComponent
14
15         // set font to Serif (Times), bold, 12pt and draw a string
16         g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
17         g.drawString( "Serif 12 point bold.", 20, 50 );
18
19         // set font to Monospaced (Courier), italic, 24pt and draw a string
20         g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
21         g.drawString( "Monospaced 24 point italic.", 20, 70 );
22
23         // set font to SansSerif (Helvetica), plain, 14pt and draw a string
24         g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
25         g.drawString( "SansSerif 14 point plain.", 20, 90 );
26

```

Font name

Font style

Font size

Creating Font objects



```
27 // set font to Serif (Times), bold/italic, 18pt and draw it
28 g.setColor( Color.RED );
29 g.setFont( new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
30 g.drawString( g.getFont().getName() + " " + g.getFont().getSize() +
31     " point bold italic.", 20, 110 );
32 } // end method paintComponent
33 } // end class FontJPanel
```

Combining styles

Outline

FontJPanel.java

(2 of 2)

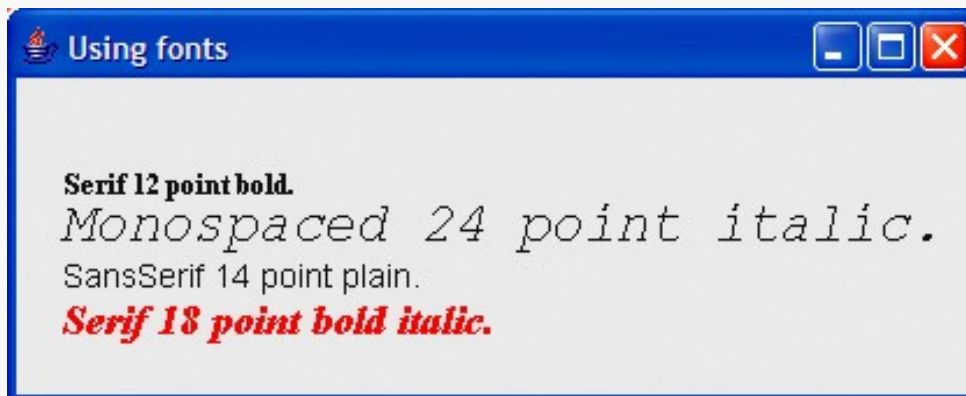
Retrieve font name and size of
Graphics object's current Font



Outline

Fonts.java

```
1 // Fig. 12.12: Fonts.java
2 // Using fonts.
3 import javax.swing.JFrame;
4
5 public class Fonts
6 {
7     // execute application
8     public static void main( String args[] )
9     {
10         // create frame for FontJPanel
11         JFrame frame = new JFrame( "Using fonts" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14         FontJPanel fontJPanel = new FontJPanel(); // create FontJPanel
15         frame.add( fontJPanel ); // add fontJPanel to frame
16         frame.setSize( 420, 170 ); // set frame size
17         frame.setVisible( true ); // display frame
18     } // end main
19 } // end class Fonts
```



Software Engineering Observation 12.2

To change the font, you must create a new Font object. Font objects are immutable—class Font has no *set* methods to change the characteristics of the current font.



Font Metrics

- **Font** class methods
 - **getFamily** – returns name of font family to which the current font belongs
 - **isPlain**, **isBold**, **isItalic** – used to determine font style
- **Font metrics** – precise information about a font
 - **Height**
 - **Descent** – amount a character dips below the baseline
 - **Ascent** – amount a character rises above the baseline
 - **Leading** – the interline spacing
 - **Class FontMetrics** declares several methods for obtaining font metrics



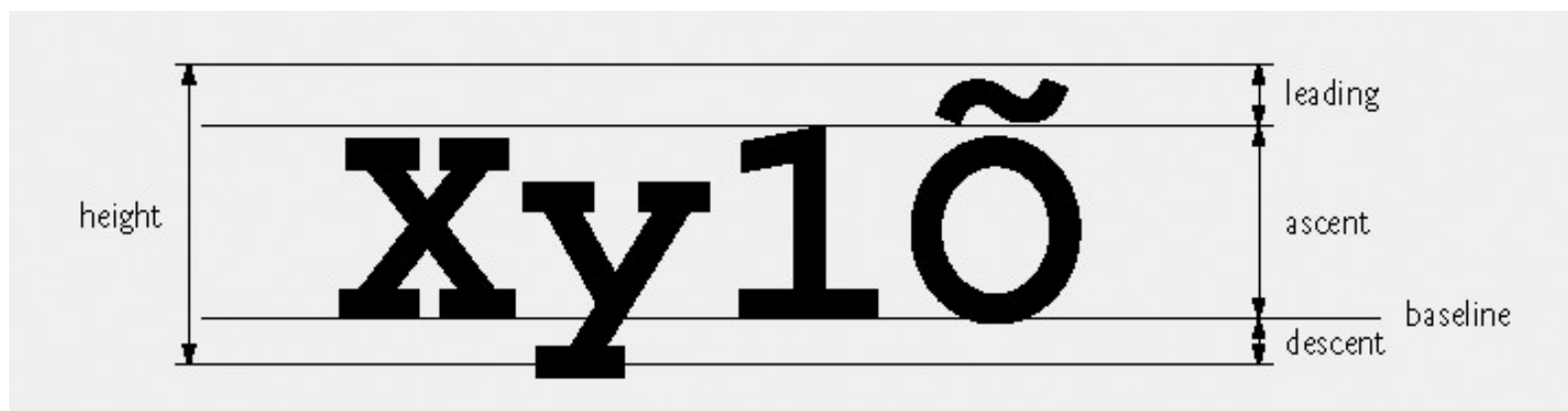


Fig. 12.13 | Font metrics.

Method	Description
<i>FontMetrics methods</i>	
<code>public int getAscent()</code>	Returns the ascent of a font in points.
<code>public int getDescent()</code>	Returns the descent of a font in points.
<code>public int getLeading()</code>	Returns the leading of a font in points.
<code>public int getHeight()</code>	Returns the height of a font in points.
<i>Graphics methods for getting a Font's FontMetrics</i>	
<code>public FontMetrics getFontMetrics()</code>	Returns the FontMetrics object for the current drawing Font.
<code>public FontMetrics getFontMetrics(Font f)</code>	Returns the FontMetrics object for the specified Font argument.

Fig. 12.14 | FontMetrics and Graphics methods for obtaining font metrics.



Outline

MetricsJPanel.java

(1 of 2)

```

1 // Fig. 12.15: MetricsJPanel.java
2 // FontMetrics and Graphics methods useful for obtaining font metrics.
3 import java.awt.Font;
4 import java.awt.FontMetrics;
5 import java.awt.Graphics;
6 import javax.swing.JPanel;
7
8 public class MetricsJPanel extends JPanel
9 {
10     // display font metrics
11     public void paintComponent( Graphics g )
12     {
13         super.paintComponent( g ); // call superclass's paintComponent
14
15         g.setFont( new Font( "SansSerif", Font.BOLD, 12
16         FontMetrics metrics = g.getFontMetrics();
17
18         g.drawString( "Current font: " + g.getFont(), 10, 40 );
19         g.drawString( "Ascent: " + metrics.getAscent(), 10, 55 );
20         g.drawString( "Descent: " + metrics.getDescent(), 10, 70 );
21         g.drawString( "Height: " + metrics.getHeight(), 10, 85 );
22         g.drawString( "Leading: " + metrics.getLeading(), 10, 100 );

```

Retrieve FontMetrics object of current Font

Retrieve font metric values



```
23 Font font = new Font( "Serif", Font.ITALIC, 14 );
24 metrics = g.getFontMetrics( font );
25 g.setFont( font );
26 g.drawString( "Current font: " + font, 10, 130 );
27 g.drawString( "Ascent: " + metrics.getAscent(), 10, 145 );
28 g.drawString( "Descent: " + metrics.getDescent(), 10, 160 );
29 g.drawString( "Height: " + metrics.getHeight(), 10, 175 );
30 g.drawString( "Leading: " + metrics.getLeading(), 10, 190 );
31 } // end method paintComponent
32 } // end class MetricsJPanel
```

Outline

MetricsJPanel.java

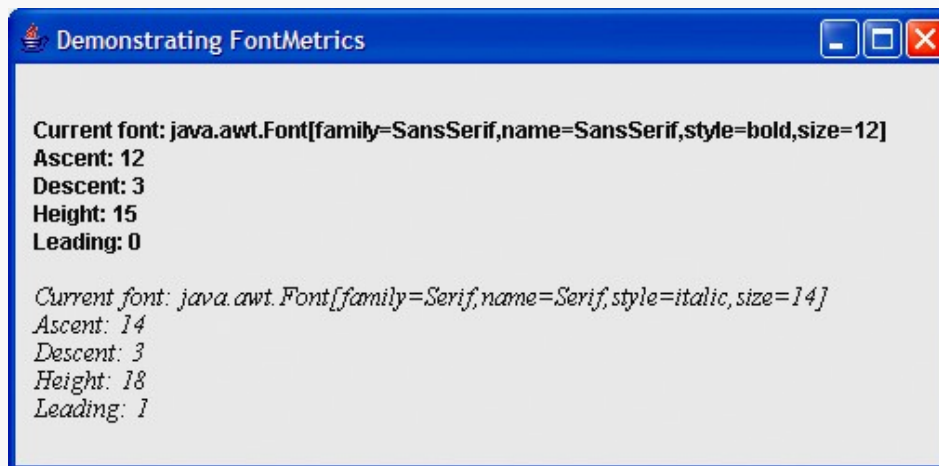
(2 of 2)



Outline

Metrics.java

```
1 // Fig. 12.16: Metrics.java
2 // Displaying font metrics.
3 import javax.swing.JFrame;
4
5 public class Metrics
6 {
7     // execute application
8     public static void main( String args[] )
9     {
10         // create frame for MetricsJPanel
11         JFrame frame = new JFrame( "Demonstrating FontMetrics" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14         MetricsJPanel metricsJPanel = new MetricsJPanel();
15         frame.add( metricsJPanel ); // add metricsJPanel to frame
16         frame.setSize( 510, 250 ); // set frame size
17         frame.setVisible( true ); // display frame
18     } // end main
19 } // end class Metrics
```



12.5 Drawing Lines, Rectangles and Ovals

- **Graphics** methods for drawing lines, rectangles and ovals
 - **fillRoundRect** and **drawRoundRect** – draw rectangles with rounded corners
 - bounding rectangle—the area in which a rounded rectangle or oval will be drawn
 - **draw3DRect** and **fill3DRect** – draw a 3D rectangle that is either raised or lowered
 - **drawOval** and **fillOval** – draw ovals



Method	Description
<code>public void drawLine(int x1, int y1, int x2, int y2)</code>	Draws a line between the point (x1, y1) and the point (x2, y2).
<code>public void drawRect(int x, int y, int width, int height)</code>	Draws a rectangle of the specified width and height. The top-left corner of the rectangle has the coordinates (x, y). Only the outline of the rectangle is drawn using the Graphics object's color—the body of the rectangle is not filled with this color.
<code>public void fillRect(int x, int y, int width, int height)</code>	Draws a filled rectangle with the specified width and height. The top-left corner of the rectangle has the coordinate (x, y). The rectangle is filled with the Graphics object's color.
<code>public void clearRect(int x, int y, int width, int height)</code>	Draws a filled rectangle with the specified width and height in the current background color. The top-left corner of the rectangle has the coordinate (x, y). This method is useful if the programmer wants to remove a portion of an image.
<code>public void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	Draws a rectangle with rounded corners in the current color with the specified width and height. The arcWidth and arcHeight determine the rounding of the corners (see Fig. 12.20). Only the outline of the shape is drawn.

Fig. 12.17 | Graphics methods that draw lines, rectangles and ovals.
(Part 1 of 2)



Method	Description
<code>public void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code>	Draws a filled rectangle with rounded corners in the current color with the specified width and height . The arcWidth and arcHeight determine the rounding of the corners (see Fig. 12.20).
<code>public void draw3DRect(int x, int y, int width, int height, boolean b)</code>	Draws a three-dimensional rectangle in the current color with the specified width and height . The top-left corner of the rectangle has the coordinates (x, y). The rectangle appears raised when b is true and lowered when b is false. Only the outline of the shape is drawn.
<code>public void fill3DRect(int x, int y, int width, int height, boolean b)</code>	Draws a filled three-dimensional rectangle in the current color with the specified width and height . The top-left corner of the rectangle has the coordinates (x, y). The rectangle appears raised when b is true and lowered when b is false.
<code>public void drawOval(int x, int y, int width, int height)</code>	Draws an oval in the current color with the specified width and height . The bounding rectangle's top-left corner is at the coordinates (x, y). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 12.21). Only the outline of the shape is drawn.
<code>public void fillOval(int x, int y, int width, int height)</code>	Draws a filled oval in the current color with the specified width and height . The bounding rectangle's top-left corner is at the coordinates (x, y). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 12.21).

Fig. 12.17 | Graphics methods that draw lines, rectangles and ovals.
(Part 2 of 2)



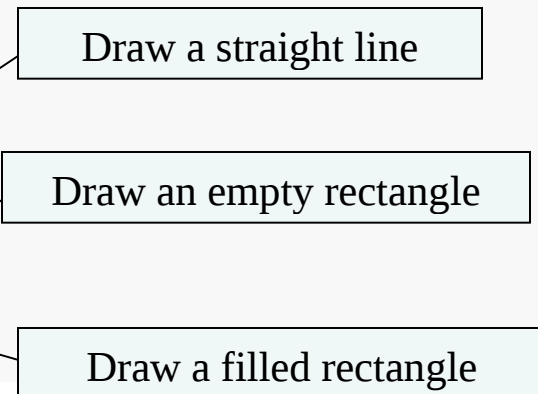
Outline

LinesRectsOvals

JPanel.java

(1 of 2)

```
1 // Fig. 12.18: LinesRectsOvalsJPanel.java
2 // Drawing lines, rectangles and ovals.
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import javax.swing.JPanel;
6
7 public class LinesRectsOvalsJPanel extends JPanel
8 {
9     // display various lines, rectangles and ovals
10    public void paintComponent( Graphics g )
11    {
12        super.paintComponent( g ); // call superclass's paint method
13
14        this.setBackground( Color.WHITE );
15
16        g.setColor( Color.RED );
17        g.drawLine( 5, 30, 380, 30 );
18
19        g.setColor( Color.BLUE );
20        g.drawRect( 5, 40, 90, 55 );
21        g.fillRect( 100, 40, 90, 55 );
22    }
```



Draw a straight line

Draw an empty rectangle

Draw a filled rectangle



```

23 g.setColor( Color.CYAN );
24 g.fillRect( 195, 40, 90, 55, 50, 50 );
25 g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
26
27 g.setColor( Color.YELLOW );
28 g.draw3DRect( 5, 100, 90, 55, true );
29 g.fill3DRect( 100, 100, 90, 55, false );
30
31 g.setColor( Color.MAGENTA );
32 g.drawOval( 195, 100, 90, 55 );
33 g.fillOval( 290, 100, 90, 55 );
34 } // end method paintComponent
35 } // end class LinesRectsOvalsJPanel

```

Draw a filled rectangle with rounded corners

Draw an empty rectangle with rounded corners

Draw an empty rectangle that is raised

Draw a filled rectangle that is lowered

Draw an empty oval

Draw a filled oval

JPanel.java

(2 of 2)



Outline

LinesRectsOvals

.java

(1 of 2)

```
1 // Fig. 12.19: LinesRectsOvals.java
2 // Drawing lines, rectangles and ovals.
3 import java.awt.Color;
4 import javax.swing.JFrame;
5
6 public class LinesRectsOvals
7 {
8     // execute application
9     public static void main( String args[] )
10    {
11        // create frame for LinesRectsOvalsJPanel
12        JFrame frame =
13            new JFrame( "Drawing lines, rectangles and ovals" );
14        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
15    }
```



```

16 LinesRectsOvalsJPanel linesRectsOvalsJPanel =
17     new LinesRectsOvalsJPanel();
18 linesRectsOvalsJPanel.setBackground( Color.WHITE );
19 frame.add( linesRectsOvalsJPanel ); // add panel to frame
20 frame.setSize( 400, 210 ); // set frame size
21 frame.setVisible( true ); // display frame
22 } // end main
23 } // end class LinesRectsOvals

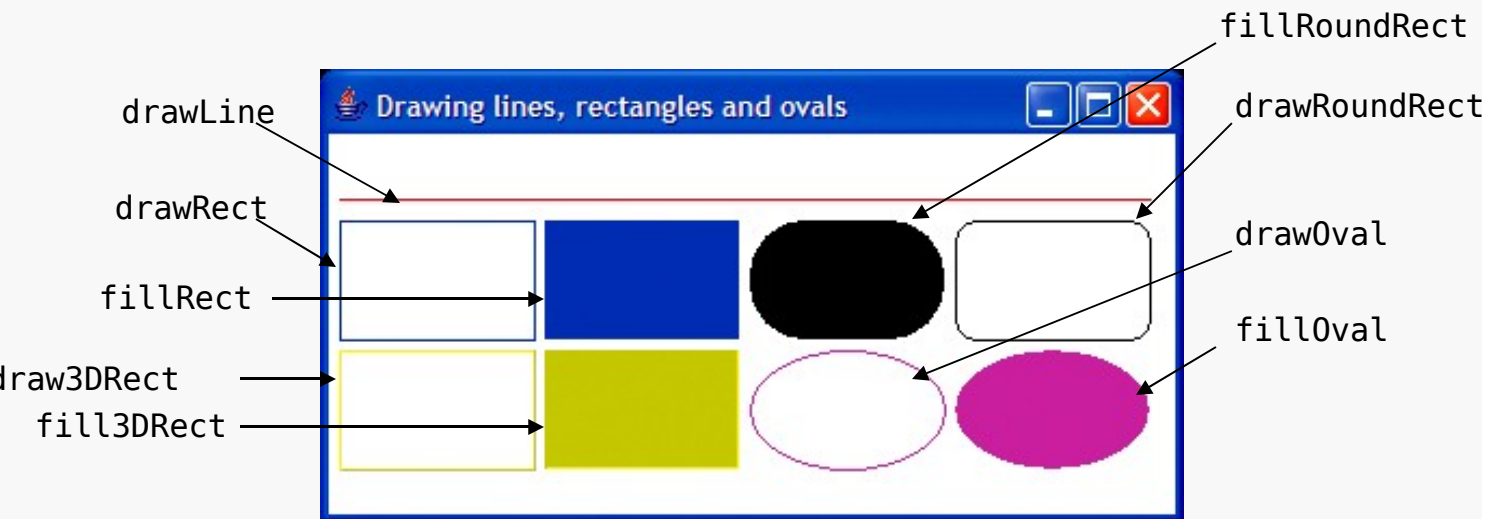
```

Outline

LinesRectsOvals

.java

(1 of 2)



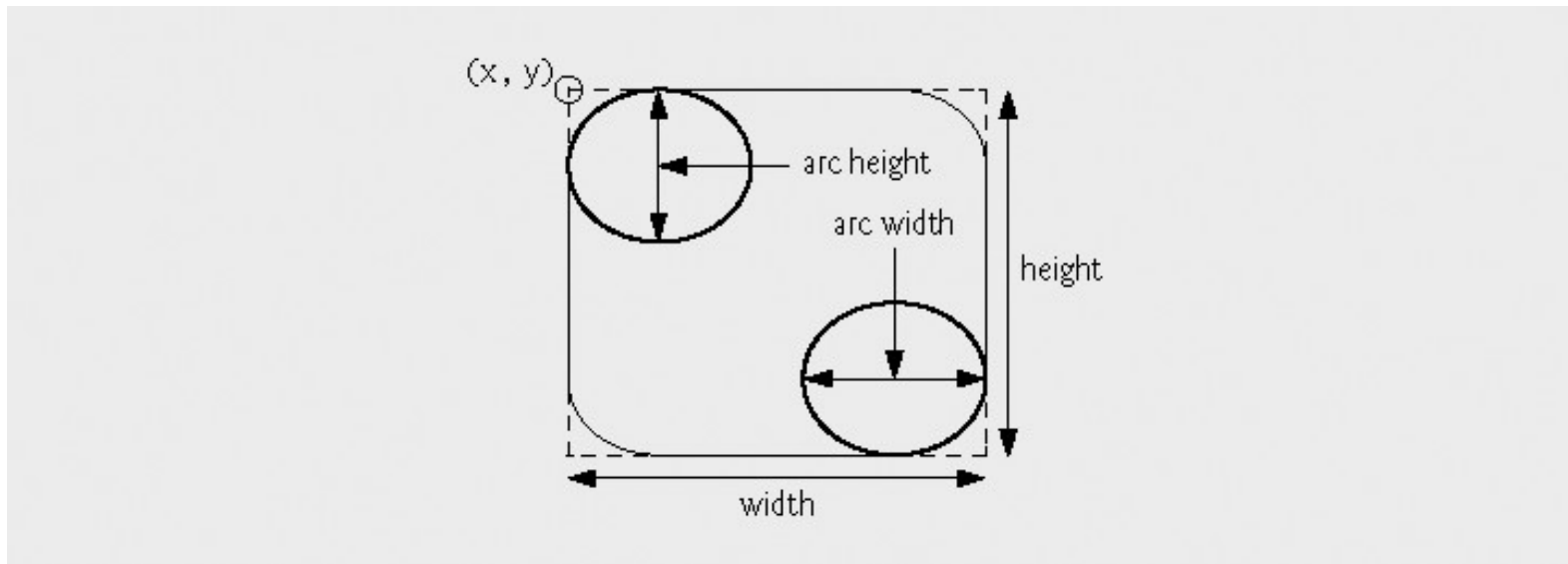


Fig. 12.20 | Arc width and arc height for rounded rectangles.

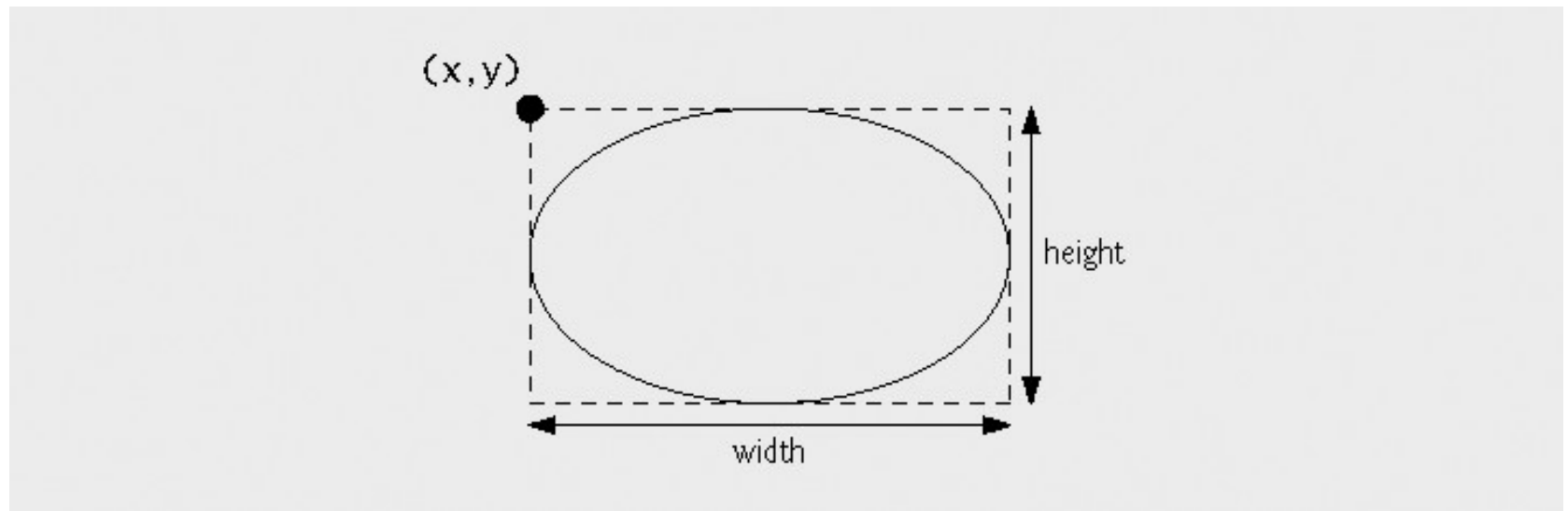


Fig. 12.21 | Oval bounded by a rectangle.

12.6 Drawing Arcs

- An arc is drawn as a portion of an oval
- Arcs sweep (i.e., move along a curve) from a starting angle by the number of degrees specified by their arc angle
 - Counterclockwise sweep measured in positive degrees
 - Clockwise sweep measured in negative degrees
- **Graphics** methods **drawArc** and **fillArc** are used to draw arcs





Fig. 12.22 | Positive and negative arc angles.

Method	Description
<code>public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	Draws an arc relative to the bounding rectangle's top-left x and y coordinates with the specified width and height. The arc segment is drawn starting at startAngle and sweeps arcAngle degrees.
<code>public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code>	Draws a filled arc (i.e., a sector) relative to the bounding rectangle's top-left x and y coordinates with the specified width and height. The arc segment is drawn starting at startAngle and sweeps arcAngle degrees.

Fig. 12.23 | Graphics methods for drawing arcs.



Outline

ArcsJPanel.java

(1 of 2)

1 // Fig. 12.24: ArcsJPanel.java

2 // Drawing arcs.

3 import java.awt.Color;

4 import java.awt.Graphics;

5 import javax.swing.JPanel;

6

7 public class ArcsJPanel extends JPanel

8 {

9 // draw rectangle

10 public void paintComponent(

11 {

12 super.paintComponent(

13

14 // start at 0 and sweep 360 degrees

15 g.setColor(Color.RED);

16 g.drawRect(15, 35, 80, 80);

17 g.setColor(Color.BLACK);

18 g.drawArc(15, 35, 80, 80, 0, 360);

19

20 // start at 0 and sweep 110 degrees

21 g.setColor(Color.RED);

22 g.drawRect(100, 35, 80, 80);

23 g.setColor(Color.BLACK);

24 g.drawArc(100, 35, 80, 80, 0, 110);

25

x- and y-coordinates for upper left
corner of bounding rectangle

Width and height of bounding
rectangle

Starting angle

Sweep angle

Draw empty arcs



Outline

ArcsJPanel.java

(2 of 2)

```
26 // start at 0 and sweep -270 degrees
27 g.setColor( Color.RED );
28 g.drawRect( 185, 35, 80, 80 );
29 g.setColor( Color.BLACK );
30 g.drawArc( 185, 35, 80, 80, 0, -270 );
31
32 // start at 0 and sweep 360 degrees
33 g.fillArc( 15, 120, 80, 40, 0, 360 );
34
35 // start at 270 and sweep -90 degrees
36 g.fillArc( 100, 120, 80, 40, 270, -90 );
37
38 // start at 0 and sweep -270 degrees
39 g.fillArc( 185, 120, 80, 40, 0, -270 );
40 } // end method paintComponent
41 } // end class ArcsJPanel
```

Draw filled arcs

Negative values indicate arc
should sweep clockwise



Outline

DrawArcs.java

(1 of 2)

```
1 // Fig. 12.25: DrawArcs.java
2 // Drawing arcs.
3 import javax.swing.JFrame;
4
5 public class DrawArcs
6 {
7     // execute application
8     public static void main( String args[] )
9     {
10         // create frame for ArcsJPanel
11         JFrame frame = new JFrame( "Drawing Arcs" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13     }
```

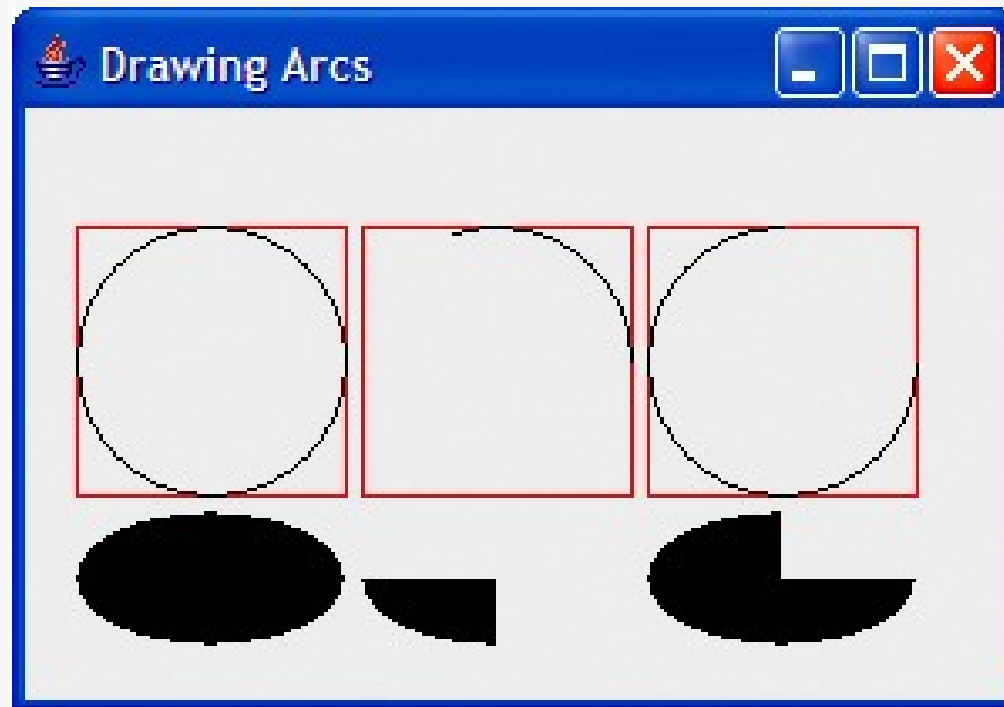


```
14 ArcsJPanel arcsJPanel = new ArcsJPanel(); // create ArcsJPanel
15 frame.add( arcsJPanel ); // add arcsJPanel to frame
16 frame.setSize( 300, 210 ); // set frame size
17 frame.setVisible( true ); // display frame
18 } // end main
19 } // end class DrawArcs
```

Outline

DrawArcs.java

(2 of 2)



12.7 Drawing Polygons and Polylines

- **Polygons**

- Closed multisided shapes composed of straight line segments
- **Graphics** methods **drawPolygon** and **fillPolygon** to display polygons
- Polygons can be represented using class **Polygon** – class contains method **addPoint** to add points to a **Polygon**

- **Polylines**

- Sequences of connected points
- **Graphics** method **drawPolyline** to display polylines



Method	Description
Graphics methods for drawing polygons	
<code>public void drawPolygon(int xPoints[], int yPoints[], int points)</code>	Draws a polygon. The x-coordinate of each point is specified in the <code>xPoints</code> array, and the y-coordinate of each point in the <code>yPoints</code> array. The last argument specifies the number of <code>points</code> . This method draws a closed polygon. If the last point is different from the first, the polygon is closed by a line that connects the last point to the first.
<code>public void drawPolyline(int xPoints[], int yPoints[], int points)</code>	Draws a sequence of connected lines. The x-coordinate of each point is specified in the <code>xPoints</code> array, and the y-coordinate of each point in the <code>yPoints</code> array. The last argument specifies the number of <code>points</code> . If the last point is different from the first, the polyline is not closed.
<code>public void drawPolygon(Polygon p)</code>	Draws the specified polygon.
<code>public void fillPolygon(int xPoints[], int yPoints[], int points)</code>	Draws a filled polygon. The x-coordinate of each point is specified in the <code>xPoints</code> array, and the y-coordinate of each point in the <code>yPoints</code> array. The last argument specifies the number of <code>points</code> . This method draws a closed polygon. If the last point is different from the first, the polygon is closed by a line that connects the last point to the first.

Fig. 12.26 | Graphics methods for polygons and class Polygon methods.
(Part 1 of 2)



Method	Description
public void fillPolygon(Polygon p)	Draws the specified filled polygon. The polygon is closed.
Polygon constructors and methods	
public Polygon()	Constructs a new polygon object. The polygon does not contain any points.
public Polygon(int xValues[], int yValues[], int numberOfPoints)	Constructs a new polygon object. The polygon has numberOfPoints sides, with each point consisting of an x-coordinate from xValues and a y-coordinate from yValues .
public void addPoint(int x, int y)	Adds pairs of x- and y-coordinates to the Polygon .

Fig. 12.26 | Graphics methods for polygons and class Polygon methods.
(Part 2 of 2)



Outline

PolygonsJPanel .java

(1 of 2)

```
1 // Fig. 12.27: PolygonsJPanel.java
2 // Drawing polygons.
3 import java.awt.Graphics;
4 import java.awt.Polygon;
5 import javax.swing.JPanel;
6
7 public class PolygonsJPanel extends JPanel
8 {
9     // draw polygons and polylines
10    public void paintComponent( Graphics g )
11    {
12        super.paintComponent( g ); // call superclass's paintComponent
13
14        // draw polygon with Polygon object
15        int xValues[] = { 20, 40, 50, 30, 20, 15 };
16        int yValues[] = { 50, 50, 60, 80, 80, 60 };
17        Polygon polygon1 = new Polygon( xValues, yValues, 6 );
18        g.drawPolygon( polygon1 );
19
20        // draw polylines with two arrays
21        int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };
22        int yValues2[] = { 100, 100, 110, 110, 130, 110, 100 };
23        g.drawPolyline( xValues2, yValues2, 7 );
24    }
```

Create Polygon object from
sets of x- and y-coordinates

Draw an empty Polygon

Draw polyline from sets of x- and
y-coordinates



Outline

Draw polygon from sets of x- and y-coordinates,
without creating Polygon object

PolygonsJPanel

.java

Add coordinates to Polygon
with method addPoint

(2 of 2)

```
25 // fill polygon with two arrays
26 int xValues3[] = { 120, 140, 150, 190 };
27 int yValues3[] = { 40, 70, 80, 60 };
28 g.fillPolygon( xValues3, yValues3, 4 );
29
30 // draw filled polygon with Polygon object
31 Polygon polygon2 = new Polygon();
32 polygon2.addPoint( 165, 135 );
33 polygon2.addPoint( 175, 150 );
34 polygon2.addPoint( 270, 200 );
35 polygon2.addPoint( 200, 220 );
36 polygon2.addPoint( 130, 180 );
37 g.fillPolygon( polygon2 );
38 } // end method paintComponent
39 } // end class PolygonsJPanel
```



Outline

DrawPolygons.java

(1 of 2)

```
1 // Fig. 12.28: DrawPolygons.java
2 // Drawing polygons.
3 import javax.swing.JFrame;
4
5 public class DrawPolygons
6 {
7     // execute application
8     public static void main( String args[] )
9     {
10         // create frame for PolygonsJPanel
11         JFrame frame = new JFrame( "Drawing Polygons" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
```

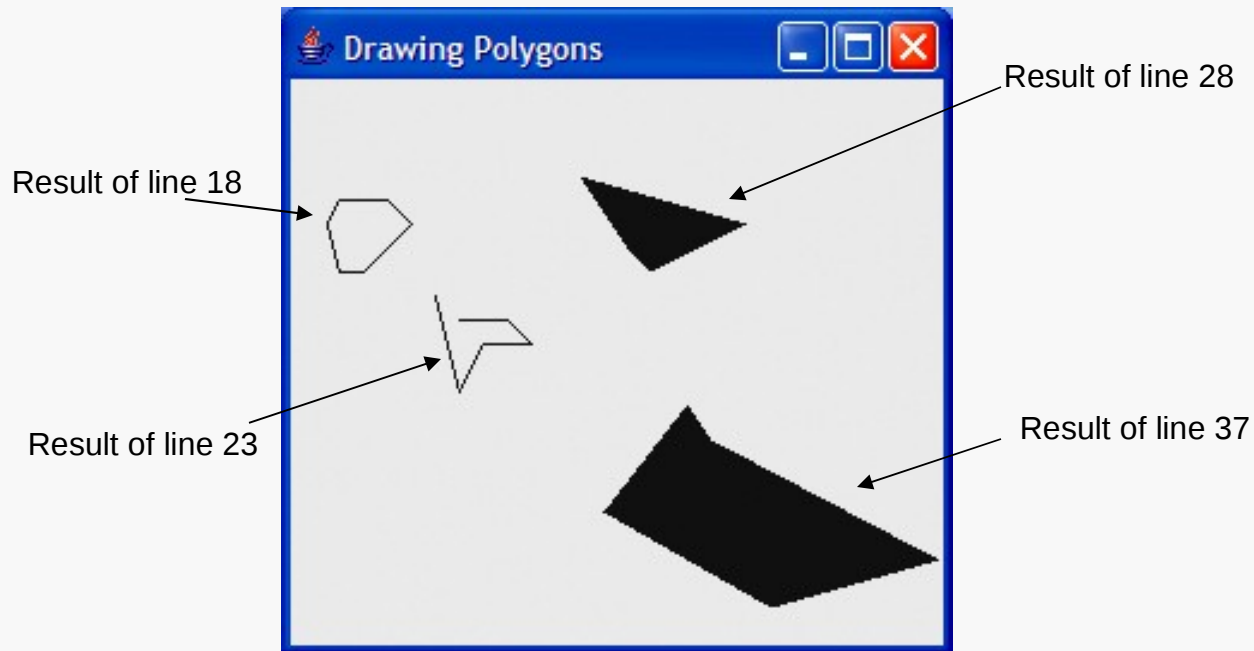


```
14 PolygonsJPanel polygonsJPanel = new PolygonsJPanel();
15 frame.add( polygonsJPanel ); // add polygonsJPanel to frame
16 frame.setSize( 280, 270 ); // set frame size
17 frame.setVisible( true ); // display frame
18 } // end main
19 } // end class DrawPolygons
```

Outline

DrawPolygons.java

(2 of 2)



Common Programming Error 12.1

An `ArrayIndexOutOfBoundsException` is thrown if the number of points specified in the third argument to method `drawPolygon` or method `fillPolygon` is greater than the number of elements in the arrays of coordinates that specify the polygon to display.



12.8 Java 2D API

- **Provides advanced two-dimensional graphics capabilities for detailed and complex graphical manipulations**
- **Features for processing line art, text and images**
- **Accomplished with `Graphics2D` class**



Lines, Rectangles, Round Rectangles, Arcs and Ellipses

- Java 2D shapes specified with double-precision floating-point values – **Line2D.Double**, **Rectangle2D.Double**, **RoundRectangle2D.Double**, **Arc2D.Double**, **Ellipse2D.Double**
- Painting with **Graphics2D** object
 - Method **setPaint** – sets color for **Graphics2D** object when shapes are drawn as **Paint** object
 - Paint object can be a predeclared **Color** object, or an instance of **GradientPaint**, **SystemColor** or **TexturePaint** classes
 - **GradientPaint** used for drawing with a gradient – gradients can be cyclic or acyclic
 - **TexturePaint** used for painting by replicating a stored image



Lines, Rectangles, Round Rectangles, Arcs and Ellipses

- **Graphics2D** method **fill** used to draw a filled **Shape** object – an object that implements interface **Shape**
- **Graphics2D** method **draw** used to draw a **Shape** object
- Setting stroke of a line or border
 - **Graphics2D** method **setStroke** – requires argument that implements interface **Stroke**
 - **BasicStroke** class – can specify width of line, end caps, line joins
- **Arc2D.Double** constants
 - **Arc2D.PIE** – arc should be closed by two lines—one from starting point to center, one from center to ending point
 - **Arc2D.CHORD** – draws a line from the starting point to the ending point
 - **Arc2D.OPEN** – arc should not be closed



Outline

ShapesJPanel.java

(1 of 3)

```
1 // Fig. 12.29: ShapesJPanel.java
2 // Demonstrating some Java 2D shapes.
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import java.awt.BasicStroke;
6 import java.awt.GradientPaint;
7 import java.awt.TexturePaint;
8 import java.awt.Rectangle;
9 import java.awt.Graphics2D;
10 import java.awt.geom.Ellipse2D;
11 import java.awt.geom.Rectangle2D;
12 import java.awt.geom.RoundRectangle2D;
13 import java.awt.geom.Arc2D;
14 import java.awt.geom.Line2D;
15 import java.awt.image.BufferedImage;
16 import javax.swing.JPanel;
17
18 public class ShapesJPanel extends JPanel
19 {
20     // draw shapes with Java 2D API
21     public void paintComponent( Graphics g )
22     {
23         super.paintComponent( g ); // call superclass's paintComponent
24
25         Graphics2D g2d = ( Graphics2D ) g; // cast g to Graphics2D
26     }
```

Java 2D API shape classes

Creating Graphics2D reference



Outline

Panel.java

```

27 // draw 2D ellipse filled with a blue-yellow gradient
28 g2d.setPaint( new GradientPaint( 5, 30, Color.BLUE, 35, 100,
29     Color.YELLOW, true ) );
30 g2d.fill( new Ellipse2D.Double( 5, 30, 65, 100 ) );
31
32 // draw 2D rectangle in red
33 g2d.setPaint( Color.RED );
34 g2d.setStroke( new BasicStroke( 10.0f ) );
35 g2d.draw( new Rectangle2D.Double( 80, 30, 65, 100 ) );
36
37 // draw 2D rounded rectangle with a buffered background
38 BufferedImage buffImage = new BufferedImage( 10, 10,
39     BufferedImage.TYPE_INT_RGB );
40
41 // obtain Graphics2D from buffImage and draw on it
42 Graphics2D gg = buffImage.createGraphics();
43 gg.setColor( Color.YELLOW ); // draw in yellow
44 gg.fillRect( 0, 0, 10, 10 ); // draw a filled rectangle
45 gg.setColor( Color.BLACK ); // draw in black
46 gg.drawRect( 1, 1, 6, 6 ); // draw a rectangle
47 gg.setColor( Color.BLUE ); // draw in blue
48 gg.fillRect( 1, 1, 3, 3 ); // draw a filled rectangle
49 gg.setColor( Color.RED ); // draw in red
50 gg.fillRect( 4, 4, 3, 3 ); // draw a filled rectangle
51

```

Draw ellipse filled using gradient

Set Graphics2D object to draw using

Set width of border to 10 pixels

Create image to be used for
TexturePaint object



Outline

```

52 // paint buffImage onto the JFrame
53 g2d.setPaint( new TexturePaint( buffImage,
54     new Rectangle( 10, 10 ) ) );
55 g2d.fill(
56     new RoundRectangle2D.Double( 155, 30, 75,
57 // draw 2D pie-shaped arc in white
58 g2d.setPaint( Color.WHITE );
59 g2d.setStroke( new BasicStroke( 6.0f ) );
60 g2d.draw(
61     new Arc2D.Double( 240, 30, 75, 100, 0, 270, Arc2D.PIE ) );
62
63
64 // draw 2D lines in green and yellow
65 g2d.setPaint( Color.GREEN );
66 g2d.draw( new Line2D.Double( 395, 30, 320, 150 ) );
67
68 // draw 2D line using stroke
69 float dashes[] = { 10 }; // specify dash pattern
70 g2d.setPaint( Color.YELLOW );
71 g2d.setStroke( new BasicStroke( 4, BasicStroke.CAP_ROUND,
72     BasicStroke.JOIN_ROUND, 10, dashes, 0 ) );
73 g2d.draw( new Line2D.Double( 320, 30, 395, 150 ) );
74 } // end method paintComponent
75 } // end class ShapesJPanel

```

Create TexturePaint object from image

Draw rounded rectangle, filled with repeating image

Draw arc using white border, 6 pixels wide

Draw solid green line

Set stroke to use dashes

Draw dashed yellow line



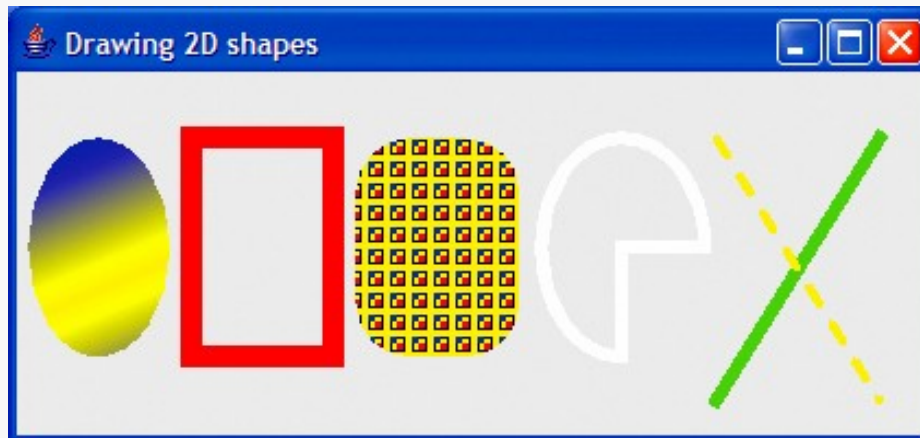
Outline

Shapes.java

```

1 // Fig. 12.30: Shapes.java
2 // Demonstrating some Java 2D shapes.
3 import javax.swing.JFrame;
4
5 public class Shapes
6 {
7     // execute application
8     public static void main( String args[] )
9     {
10         // create frame for ShapesJPanel
11         JFrame frame = new JFrame( "Drawing 2D shapes" );
12         frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13
14         // create ShapesJPanel
15         ShapesJPanel shapesJPanel = new ShapesJPanel();
16
17         frame.add( shapesJPanel ); // add shapesJPanel to frame
18         frame.setSize( 425, 200 ); // set frame size
19         frame.setVisible( true ); // display frame
20     } // end main
21 } // end class Shapes

```



General Paths

- A general path is a shape constructed from straight lines and complex curves
- Class **GeneralPath**
 - Method **moveTo** specifies the first point in a general path
 - Method **lineTo** draws a line to the next point in the path
 - Method **closePath** completes the general path
- **Graphics2D** method **translate** – used to move the drawing origin
- **Graphics2D** method **rotate** – used to rotate the next displayed shape



Outline

Shapes2JPanel.java

(1 of 2)

```
1 // Fig. 12.31: Shapes2JPanel.java
2 // Demonstrating a general path.
3 import java.awt.Color;
4 import java.awt.Graphics;
5 import java.awt.Graphics2D;
6 import java.awt.geom.GeneralPath;
7 import java.util.Random;
8 import javax.swing.JPanel;
9
10 public class Shapes2JPanel extends JPanel
11 {
12     // draw general paths
13     public void paintComponent( Graphics g )
14     {
15         super.paintComponent( g ); // call superclass's paintComponent
16         Random random = new Random(); // get random number generator
17
18         int xPoints[] = { 55, 67, 109, 73, 83, 55, 27, 37 };
19         int yPoints[] = { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };
20
21         Graphics2D g2d = ( Graphics2D ) g;
22         GeneralPath star = new GeneralPath(); // create GeneralPath object
23
24         // set the initial coordinate of the General Path
25         star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );
26     }
27 }
```

Create GeneralPath object

Set starting point of
GeneralPath object



Outline

Shapes2JPanel.java

```

27 // create the star--this does not draw the star
28 for ( int count = 1; count < xPoints.length; count++ )
29     star.lineTo( xPoints[ count ], yPoints[ count ] );
30
31 star.closePath(); // close the shape
32
33 g2d.translate( 200, 200 ); // translate the origin
34
35 // rotate around origin and
36 for ( int count = 1; count < xPoints.length; count++ )
37 {
38     g2d.rotate( Math.PI / 10.0 ); // rotate coordinate system
39
40     // set random drawing color
41     g2d.setColor( new Color( random.nextInt( 256 ),
42                             random.nextInt( 256 ), random.nextInt( 256 ) ) );
43
44     g2d.fill( star ); // draw filled star
45 } // end for
46 } // end method paintComponent
47 } // end class Shapes2JPanel

```

Add lines of general path

Draw line from last point to first point

Rotate approximately 18 degrees

Draw star at current angle
around origin



Outline

Shapes2.java

(1 of 2)

```
1 // Fig. 12.32: Shapes2.java
2 // Demonstrating a general path.
3 import java.awt.Color;
4 import javax.swing.JFrame;
5
6 public class Shapes2
7 {
8     // execute application
9     public static void main( String args[] )
10    {
11        // create frame for Shapes2JPanel
12        JFrame frame = new JFrame( "Drawing 2D Shapes" );
13        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
14    }
```



```
15 Shapes2JPanel shapes2JPanel = new Shapes2JPanel();
16 frame.add( shapes2JPanel ); // add shapes2JPanel to frame
17 frame.setBackground( Color.WHITE ); // set frame background color
18 frame.setSize( 400, 400 ); // set frame size
19 frame.setVisible( true ); // display frame
20 } // end main
21 } // end class Shapes2
```

Outline

Shapes2.java

(2 of 2)

