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

Fig. 12.3 | Color constants and their RGB values.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Color constructors and methods</strong></td>
<td></td>
</tr>
<tr>
<td><code>public Color( int r, int g, int b )</code></td>
<td>Creates a color based on red, green and blue components expressed as integers from 0 to 255.</td>
</tr>
<tr>
<td><code>public Color( float r, float g, float b )</code></td>
<td>Creates a color based on red, green and blue components expressed as floating-point values from 0.0 to 1.0.</td>
</tr>
<tr>
<td><code>public int getRed()</code></td>
<td>Returns a value between 0 and 255 representing the red content.</td>
</tr>
<tr>
<td><code>public int getGreen()</code></td>
<td>Returns a value between 0 and 255 representing the green content.</td>
</tr>
<tr>
<td><code>public int getBlue()</code></td>
<td>Returns a value between 0 and 255 representing the blue content.</td>
</tr>
<tr>
<td><strong>Graphics methods for manipulating Colors</strong></td>
<td></td>
</tr>
<tr>
<td><code>public Color getColor()</code></td>
<td>Returns <code>Color</code> object representing current color for the graphics context.</td>
</tr>
<tr>
<td><code>public void setColor( Color c )</code></td>
<td>Sets the current color for drawing with the graphics context.</td>
</tr>
</tbody>
</table>

**Fig. 12.4** | **Color methods and color-related Graphics methods.**
// Fig. 12.5: ColorJPanel.java
// Demonstrating Colors.
import java.awt.Graphics;
import java.awt.Color;
import javax.swing.JPanel;

public class ColorJPanel extends JPanel {
    // draw rectangles and Strings in different colors
    public void paintComponent(Graphics g) {
        super.paintComponent(g); // call superclass's paintComponent

        this.setBackground(Color.WHITE);

        // set new drawing color using integers
        g.setColor(new Color(255, 0, 0));
        g.fillRect(15, 25, 100, 20);
        g.drawString("Current RGB: " + g.getColor(), 130, 40);

        // set new drawing color using floats
        g.setColor(new Color(0.50f, 0.75f, 0.0f));
        g.fillRect(15, 50, 100, 20);
        g.drawString("Current RGB: " + g.getColor(), 130, 65);

        // set new drawing color using static Color objects
        g.setColor(Color.BLUE);
        g.fillRect(15, 75, 100, 20);
        g.drawString("Current RGB: " + g.getColor(), 130, 90);
    }
}

Method paintComponent paints JPanel
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

© 1992-2007 Pearson Education, Inc. All rights reserved.
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
Fig. 12.6: ShowColors.java

Demonstrating Colors.

import javax.swing.JFrame;

public class ShowColors
{
    // execute application
    public static void main( String args[] )
    {
        // create frame for ColorJPanel
        JFrame frame = new JFrame( "Using colors" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        frame.add( colorJPanel ); // add ColorJPanel to frame
        frame.setSize( 400, 180 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // end class ShowColors

Using colors

Current RGB: java.awt.Color[r=255,g=0,b=0]
Current RGB: java.awt.Color[r=128,g=191,b=0]
Current RGB: java.awt.Color[r=0,g=0,b=255]
RGB values: 255, 0, 255
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
public class ShowColors2JFrame
{
    private JButton changeColorJButton;
    private Color color = Color.LIGHT_GRAY;
    private JPanel colorJPanel;

    // set up GUI
    public ShowColors2JFrame()
    {
       super( "Using JColorChooser" );

       // create JPanel for display color
       colorJPanel = new JPanel();
       colorJPanel.setBackground( color );

       // set up changeColorJButton and register its event handler
       changeColorJButton = new JButton( "Change Color" );
       changeColorJButton.addActionListener(

   ```
new ActionListener() // anonymous inner class
{
    // display JColorChooser when user clicks button
    public void actionPerformed(ActionEvent event)
    {
        color = JColorChooser.showDialog(ShowColors2JFrame.this, "Choose a color", color);
        // set default color, if no color is returned
        if (color == null)
            color = Color.LIGHT_GRAY;
        // change content pane's background color
        colorJPanel.setBackground(color);
    } // end method actionPerformed
} // end anonymous inner class
); // end call to addActionListener

add(colorJPanel, BorderLayout.CENTER); // add colorJPanel
add(changeColorJButton, BorderLayout.SOUTH); // add button

setSize(400, 130); // set frame size
setVisible(true); // display frame

} // end ShowColors2JFrame constructor
} // end class ShowColors2JFrame
public class ShowColors2
{
    // execute application
    public static void main( String args[] )
    {
        ShowColors2JFrame application = new ShowColors2JFrame();
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    } // end main
} // end class ShowColors2
Select a color from one of the color swatches.
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
<table>
<thead>
<tr>
<th>Method or constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Font constants, constructors and methods</strong></td>
<td></td>
</tr>
<tr>
<td>public final static int PLAIN</td>
<td>A constant representing a plain font style.</td>
</tr>
<tr>
<td>public final static int BOLD</td>
<td>A constant representing a bold font style.</td>
</tr>
<tr>
<td>public final static int ITALIC</td>
<td>A constant representing an italic font style.</td>
</tr>
<tr>
<td><strong>public Font( String name, int style, int size )</strong></td>
<td>Creates a Font object with the specified font name, style and size.</td>
</tr>
<tr>
<td>public int getStyle()</td>
<td>Returns an integer value indicating the current font style.</td>
</tr>
<tr>
<td>public int getSize()</td>
<td>Returns an integer value indicating the current font size.</td>
</tr>
<tr>
<td>Method or constant</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>public String getName()</code></td>
<td>Returns the current font name as a string.</td>
</tr>
<tr>
<td><code>public String getFamily()</code></td>
<td>Returns the font’s family name as a string.</td>
</tr>
<tr>
<td><code>public boolean isPlain()</code></td>
<td>Returns <code>true</code> if the font is plain, else <code>false</code>.</td>
</tr>
<tr>
<td><code>public boolean isBold()</code></td>
<td>Returns <code>true</code> if the font is bold, else <code>false</code>.</td>
</tr>
<tr>
<td><code>public boolean isItalic()</code></td>
<td>Returns <code>true</code> if the font is italic, else <code>false</code>.</td>
</tr>
</tbody>
</table>

**Graphics methods for manipulating Fonts**

<table>
<thead>
<tr>
<th>Method or constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Font getFont()</code></td>
<td>Returns a <code>Font</code> object reference representing the current font.</td>
</tr>
<tr>
<td><code>public void setFont(Font f)</code></td>
<td>Sets the current font to the font, style and size specified by the <code>Font</code> object reference <code>f</code>.</td>
</tr>
</tbody>
</table>

**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.
// Fig. 12.11: FontJPanel.java
// Display strings in different fonts and colors.
import java.awt.Font;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class FontJPanel extends JPanel {
    // display Strings in different fonts and colors
    public void paintComponent(Graphics g) {
        super.paintComponent(g); // call superclass's paintComponent

        // set font to Serif (Times), bold, 12pt and draw a string
        g.setFont(new Font("Serif", Font.BOLD, 12));
        g.drawString("Serif 12 point bold.", 20, 50);

        // set font to Monospaced (Courier), italic, 24pt and draw a string
        g.setFont(new Font("Monospaced", Font.ITALIC, 24));
        g.drawString("Monospaced 24 point italic.", 20, 70);

        // set font to SansSerif (Helvetica), plain, 14pt and draw a string
        g.setFont(new Font("SansSerif", Font.PLAIN, 14));
        g.drawString("SansSerif 14 point plain.", 20, 90);
    }
}

Creating and setting Font objects

Font name
Font style
Font size
Combining styles

Retrieve font name and size of Graphics object’s current Font
public class Fonts {
    public static void main( String args[] ) {
        JFrame frame = new JFrame( "Using fonts" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        frame.add( fontJPanel ); // add FontJPanel to frame
        frame.setSize( 420, 170 ); // set frame size
        frame.setVisible( true ); // display frame
    }
}
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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FontMetrics methods</strong></td>
<td></td>
</tr>
<tr>
<td>public int getAscent()</td>
<td>Returns the ascent of a font in points.</td>
</tr>
<tr>
<td>public int getDescent()</td>
<td>Returns the descent of a font in points.</td>
</tr>
<tr>
<td>public int getLeading()</td>
<td>Returns the leading of a font in points.</td>
</tr>
<tr>
<td>public int getHeight()</td>
<td>Returns the height of a font in points.</td>
</tr>
<tr>
<td><strong>Graphics methods for getting a Font's FontMetrics</strong></td>
<td></td>
</tr>
<tr>
<td>public FontMetrics getFontMetrics()</td>
<td>Returns the FontMetrics object for the current drawing Font.</td>
</tr>
<tr>
<td>public FontMetrics getFontMetrics(Font f)</td>
<td>Returns the FontMetrics object for the specified Font argument.</td>
</tr>
</tbody>
</table>

Fig. 12.14 | FontMetrics and Graphics methods for obtaining font metrics.
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import javax.swing.JPanel;

public class MetricsJPanel extends JPanel {

    // display font metrics
    public void paintComponent( Graphics g ) {
        super.paintComponent( g ); // call superclass's paintComponent

        g.setFont( new Font( "SansSerif", Font.BOLD, 12 ) );
        FontMetrics metrics = g.getFontMetrics();

        g.drawString( "Current font: " + g.getFont(), 10, 40 );
        g.drawString( "Ascent: " + metrics.getAscent(), 10, 55 );
        g.drawString( "Descent: " + metrics.getDescent(), 10, 70 );
        g.drawString( "Height: " + metrics.getHeight(), 10, 85 );
        g.drawString( "Leading: " + metrics.getLeading(), 10, 100 );
    }
}

Retrieve Font Metrics object of current Font
Retrieve font metric values
Font font = new Font("Serif", Font.ITALIC, 14);
metrics = g.getFontMetrics(font);
g.setFont(font);
g.drawString("Current font: " + font, 10, 130);
g.drawString("Ascent: " + metrics.getAscent(), 10, 145);
g.drawString("Descent: " + metrics.getDescent(), 10, 160);
g.drawString("Height: " + metrics.getHeight(), 10, 175);
g.drawString("Leading: " + metrics.getLeading(), 10, 190);
} // end method paintComponent
} // end class MetricsJPanel
// Fig. 12.16: Metrics.java
// Displaying font metrics.

import javax.swing.JFrame;

public class Metrics
{
    // execute application
    public static void main( String args[] )
    {
        // create frame for MetricsJPanel
        JFrame frame = new JFrame( "Demonstrating FontMetrics" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        MetricsJPanel metricsJPanel = new MetricsJPanel();
        frame.add( metricsJPanel ); // add metricsJPanel to frame
        frame.setSize( 510, 250 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void drawLine( int x1, int y1, int x2, int y2 )</td>
<td></td>
</tr>
<tr>
<td>Draws a line between the point ((x_1, y_1)) and the point ((x_2, y_2)).</td>
<td></td>
</tr>
</tbody>
</table>

| public void drawRect( int x, int y, int width, int height ) |
| 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 \textit{Graphics} object’s color—the body of the rectangle is not filled with this color. |

| public void fillRect( int x, int y, int width, int height ) |
| 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 \textit{Graphics} object’s color. |

| public void clearRect( int x, int y, int width, int height ) |
| 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. |

| public void drawRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight ) |
| Draws a rectangle with rounded corners in the current color with the specified width and height. The \textit{arcWidth} and \textit{arcHeight} determine the rounding of the corners (see Fig. 12.20). Only the outline of the shape is drawn. |

**Fig. 12.17** | \textbf{Graphics methods that draw lines, rectangles and ovals.}  
(Part 1 of 2)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)</code></td>
<td></td>
</tr>
<tr>
<td>Draws a filled rectangle with rounded corners in the current color with the specified <code>width</code> and <code>height</code>. The <code>arcWidth</code> and <code>arcHeight</code> determine the rounding of the corners (see Fig. 12.20).</td>
<td></td>
</tr>
<tr>
<td><code>public void draw3DRect(int x, int y, int width, int height, boolean b)</code></td>
<td></td>
</tr>
<tr>
<td>Draws a three-dimensional rectangle in the current color with the specified <code>width</code> and <code>height</code>. The top-left corner of the rectangle has the coordinates <code>(x, y)</code>. The rectangle appears raised when <code>b</code> is true and lowered when <code>b</code> is false. Only the outline of the shape is drawn.</td>
<td></td>
</tr>
<tr>
<td><code>public void fill3DRect(int x, int y, int width, int height, boolean b)</code></td>
<td></td>
</tr>
<tr>
<td>Draws a filled three-dimensional rectangle in the current color with the specified <code>width</code> and <code>height</code>. The top-left corner of the rectangle has the coordinates <code>(x, y)</code>. The rectangle appears raised when <code>b</code> is true and lowered when <code>b</code> is false.</td>
<td></td>
</tr>
<tr>
<td><code>public void drawOval(int x, int y, int width, int height)</code></td>
<td></td>
</tr>
<tr>
<td>Draws an oval in the current color with the specified <code>width</code> and <code>height</code>. The bounding rectangle’s top-left corner is at the coordinates <code>(x, y)</code>. 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.</td>
<td></td>
</tr>
<tr>
<td><code>public void fillOval(int x, int y, int width, int height)</code></td>
<td></td>
</tr>
<tr>
<td>Draws a filled oval in the current color with the specified <code>width</code> and <code>height</code>. The bounding rectangle’s top-left corner is at the coordinates <code>(x, y)</code>. The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 12.21).</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 12.17 | Graphics methods that draw lines, rectangles and ovals. (Part 2 of 2)**
```java
// Fig. 12.18: LinesRectsOvalsJPanel.java
// Drawing lines, rectangles and ovals.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class LinesRectsOvalsJPanel extends JPanel {
    // display various lines, rectangles and ovals
    public void paintComponent(Graphics g) {
       super.paintComponent(g); // call superclass's paint method
       this.setBackground(Color.WHITE);

       g.setColor(Color.RED);
       g.drawLine(5, 30, 380, 30);

       g.setColor(Color.BLUE);
       g.drawRect(5, 40, 90, 55);
       g.fillRect(100, 40, 90, 55);
    }
}
```

- **Draw a straight line**
- **Draw an empty rectangle**
- **Draw a filled rectangle**
23       g.setColor( Color.CYAN );
24       g.fillRoundRect( 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

© 1992-2007 Pearson Education, Inc. All rights reserved.
// Fig. 12.19: LinesRectsOvals.java
// Drawing lines, rectangles and ovals.
import java.awt.Color;
import javax.swing.JFrame;

public class LinesRectsOvals {
    // execute application
    public static void main( String args[] )
    {
        // create frame for LinesRectsOvalsJPanel
        JFrame frame =
            new JFrame( "Drawing lines, rectangles and ovals" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
LinesRectsOvals = LinesRectsOvals =
      new LinesRectsOvals();
LinesRectsOvals.setBackground(Color.WHITE);
frame.add(LinesRectsOvals); // add panel to frame
frame.setSize(400, 210); // set frame size
frame.setVisible(true); // display frame
} // end main
} // end class LinesRectsOvals
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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td><code>public void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)</code></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 12.23** | **Graphics methods for drawing arcs.**
public class ArcsJPanel extends JPanel
{
    // draw rectangles and arcs
    public void paintComponent( Graphics g )
    {
        super.paintComponent( g ); // call superclass's paintComponent

        // start at 0 and sweep 360 degrees
        g.setColor( Color.RED );
        g.drawRect( 15, 35, 80, 80 );
        g.setColor( Color.BLACK );
        g.drawArc( 15, 35, 80, 80, 0, 360 );

        // start at 0 and sweep 110 degrees
        g.setColor( Color.RED );
        g.drawRect( 100, 35, 80, 80 );
        g.setColor( Color.BLACK );
        g.drawArc( 100, 35, 80, 80, 0, 110 );
    }
}

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

Width and height of bounding rectangle

Starting angle

Sweep angle

Draw empty arcs
Draw filled arcs

Negative values indicate arc should sweep clockwise
// Fig. 12.25: DrawArcs.java
// Drawing arcs.

import javax.swing.JFrame;

public class DrawArcs
{
    // execute application
    public static void main( String args[] )
    {

        // create frame for ArcsJPanel
        JFrame frame = new JFrame( "Drawing Arcs" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }
}
frame.add( arcsJPanel ); // add ArcsJPanel to frame
frame.setSize( 300, 210 ); // set frame size
frame.setVisible( true ); // display frame
} // end main
} // end class DrawArcs
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
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphics methods for drawing polygons</strong></td>
<td></td>
</tr>
<tr>
<td>public void drawPolygon( int xPoints[], int yPoints[], int points )</td>
<td>Draws a polygon. The x-coordinate of each point is specified in the xPoints array, and the y-coordinate of each point in the yPoints array. The last argument specifies the number of points. 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.</td>
</tr>
<tr>
<td>public void drawPolyline( int xPoints[], int yPoints[], int points )</td>
<td>Draws a sequence of connected lines. The x-coordinate of each point is specified in the xPoints array, and the y-coordinate of each point in the yPoints array. The last argument specifies the number of points. If the last point is different from the first, the polyline is not closed.</td>
</tr>
<tr>
<td>public void drawPolygon( Polygon p )</td>
<td>Draws the specified polygon.</td>
</tr>
<tr>
<td>public void fillPolygon( int xPoints[], int yPoints[], int points )</td>
<td>Draws a filled polygon. The x-coordinate of each point is specified in the xPoints array, and the y-coordinate of each point in the yPoints array. The last argument specifies the number of points. 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.</td>
</tr>
</tbody>
</table>

Fig. 12.26 | **Graphics methods for polygons and class Polygon methods.** (Part 1 of 2)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public void fillPolygon</strong> (Polygon p)</td>
<td>Draws the specified filled polygon. The polygon is closed.</td>
</tr>
</tbody>
</table>

**Polygon constructors and methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public Polygon()</strong></td>
<td>Constructs a new polygon object. The polygon does not contain any points.</td>
</tr>
<tr>
<td><strong>public Polygon(int xValues[], int yValues[], int numberOfPoints)</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>public void addPoint(int x, int y)</strong></td>
<td>Adds pairs of x- and y-coordinates to the Polygon.</td>
</tr>
</tbody>
</table>

---

**Fig. 12.26** | **Graphics methods for polygons and class Polygon methods.**  
(Part 2 of 2)
public class PolygonsJPanel extends JPanel {
    public void paintComponent( Graphics g ) {
        // draw polygon with Polygon object
        int xValues[] = { 20, 40, 50, 30, 20, 15 };  // sets of x-and y-coordinates
        int yValues[] = { 50, 50, 60, 80, 80, 60 };  // sets of x-and y-coordinates
        Polygon polygon1 = new Polygon( xValues, yValues, 6 );
        g.drawPolygon( polygon1 );

        // draw polyline from two arrays
        int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };  // sets of x-coordinates
        int yValues2[] = { 100, 100, 110, 110, 130, 110, 90 };  // sets of y-coordinates
        g.drawPolyline( xValues2, yValues2, 7 );
// fill polygon with two arrays
int xValues3[] = { 120, 140, 150, 190 };
int yValues3[] = { 40, 70, 80, 60 };
g.fillPolygon( xValues3, yValues3, 4 );

// draw filled polygon with Polygon object
Polygon polygon2 = new Polygon();
polygon2.addPoint( 165, 135 );
polygon2.addPoint( 175, 150 );
polygon2.addPoint( 270, 200 );
polygon2.addPoint( 200, 220 );
polygon2.addPoint( 130, 180 );
g.fillPolygon( polygon2 );
import javax.swing.JFrame;

public class DrawPolygons {
    // execute application
    public static void main( String args[] )
    {
        // create frame for PolygonsJPanel
        JFrame frame = new JFrame( "Drawing Polygons" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }
}
PolygonsJPanel polygonsJPanel = new PolygonsJPanel();
frame.add( polygonsJPanel ); // add polygonsJPanel to frame
frame.setSize( 280, 270 ); // set frame size
frame.setVisible( true ); // display frame

} // end main

} // end class DrawPolygons
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

© 1992-2007 Pearson Education, Inc. All rights reserved.
// Fig. 12.29: ShapesJPanel.java  
// Demonstrating some Java 2D shapes.
import java.awt.Color;
import java.awt.Graphics;
import java.awt.BasicStroke;
import java.awt.GradientPaint;
import java.awt.TexturePaint;
import java.awt.Rectangle;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.geom.RoundRectangle2D;
import java.awt.geom.Arc2D;
import java.awt.geom.Line2D;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;

public class ShapesJPanel extends JPanel {
    // draw shapes with Java 2D API
    public void paintComponent( Graphics g )
    {
        super.paintComponent( g ); // call superclass's paintComponent

        Graphics2D g2d = ( Graphics2D ) g; // cast g to Graphics2D

        Graphcs2D g2d = ( Graphics2D ) g; // cast g to Graphics2D
    }
}
// draw 2D ellipse filled with a blue-yellow gradient
// draw 2D rectangle in red
// draw 2D rounded rectangle with a buffered background

Set Graphics2D object to draw using a gradient from blue to yellow
Draw ellipse filled using gradient
Set width of border to 10 pixels
Create image to be used for TexturePaint object
52       // paint buffImage onto the JFrame
53       g2d.setPaint( new TexturePaint( buffImage,
54              new Rectangle( 10, 10 ) ) );
55       g2d.fill( new RoundRectangle2D.Double( 155, 30, 75, 100, 50, 50 ) );
58       // draw 2D pie-shaped arc in white
59       g2d.setPaint( Color.WHITE );
60       g2d.setStroke( new BasicStroke( 6.0f ) );
61       g2d.draw( new Arc2D.Double( 240, 30, 75, 100, 0, 270, Arc2D.PIE ) );
64       // draw 2D lines in green and yellow
65       g2d.setPaint( Color.GREEN );
66       g2d.draw( new Line2D.Double( 395, 30, 320, 150 ) );
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, BasicStroke.JOIN_ROUND, 10, dashes, 0 ) );
73       g2d.draw( new Line2D.Double( 320, 30, 395, 150 ) );
75 } // end method paintComponent
76 } // 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
// Fig. 12.30: Shapes.java
// Demonstrating some Java 2D shapes.
import javax.swing.JFrame;

public class Shapes
{
    // execute application
    public static void main( String args[] )
    {
        // create frame for ShapesJPanel
        JFrame frame = new JFrame( "Drawing 2D shapes" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        // create ShapesJPanel
        ShapesJPanel shapesJPanel = new ShapesJPanel();

        frame.add( shapesJPanel ); // add shapesJPanel to frame
        frame.setSize( 425, 200 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // 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
public class Shapes2JPanel extends JPanel {

    // draw general paths
    public void paintComponent( Graphics g ) {
        super.paintComponent( g ); // call superclass's paintComponent
        Random random = new Random(); // get random number generator

        int xPoints[] = { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
        int yPoints[] = { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };

        Graphics2D g2d = ( Graphics2D ) g;
        GeneralPath star = new GeneralPath(); // create GeneralPath object

        // set the initial coordinate of the GeneralPath
        star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );
    }
}

Create GeneralPath object
Set starting point of GeneralPath object
// create the star--this does not draw the star
for ( int count = 1; count < xPoints.length; count++ )
    star.lineTo( xPoints[ count ], yPoints[ count ] );

star.closePath(); // close the shape

g2d.translate( 200, 200 ); // translate the origin

// rotate around origin and draw stars in random colors
for ( int count = 1; count <= 20; count++ )
{
    g2d.rotate( Math.PI / 10.0 ); // rotate coordinate system

    // set random drawing color
    g2d.setColor( new Color( random.nextInt( 256 ),
                           random.nextInt( 256 ), random.nextInt( 256 ) ) );

    g2d.fill( star ); // draw filled star
}
} // end method paintComponent

} // 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
// Fig. 12.32: Shapes2.java
// Demonstrating a general path.
import java.awt.Color;
import javax.swing.JFrame;

public class Shapes2
{
    // execute application
    public static void main( String args[] )
    {
        // create frame for Shapes2JPanel
        JFrame frame = new JFrame( "Drawing 2D Shapes" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
Shapes2J Panel shapes2J Panel = new Shapes2J Panel();
frame.add( shapes2J Panel ); // add shapes2J Panel to frame
frame.setBackground( Color.WHITE ); // set frame background color
frame.setSize( 400, 400 ); // set frame size
frame.setVisible( true ); // display frame
} // end main
} // end class Shapes2