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`.**

Outline

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**

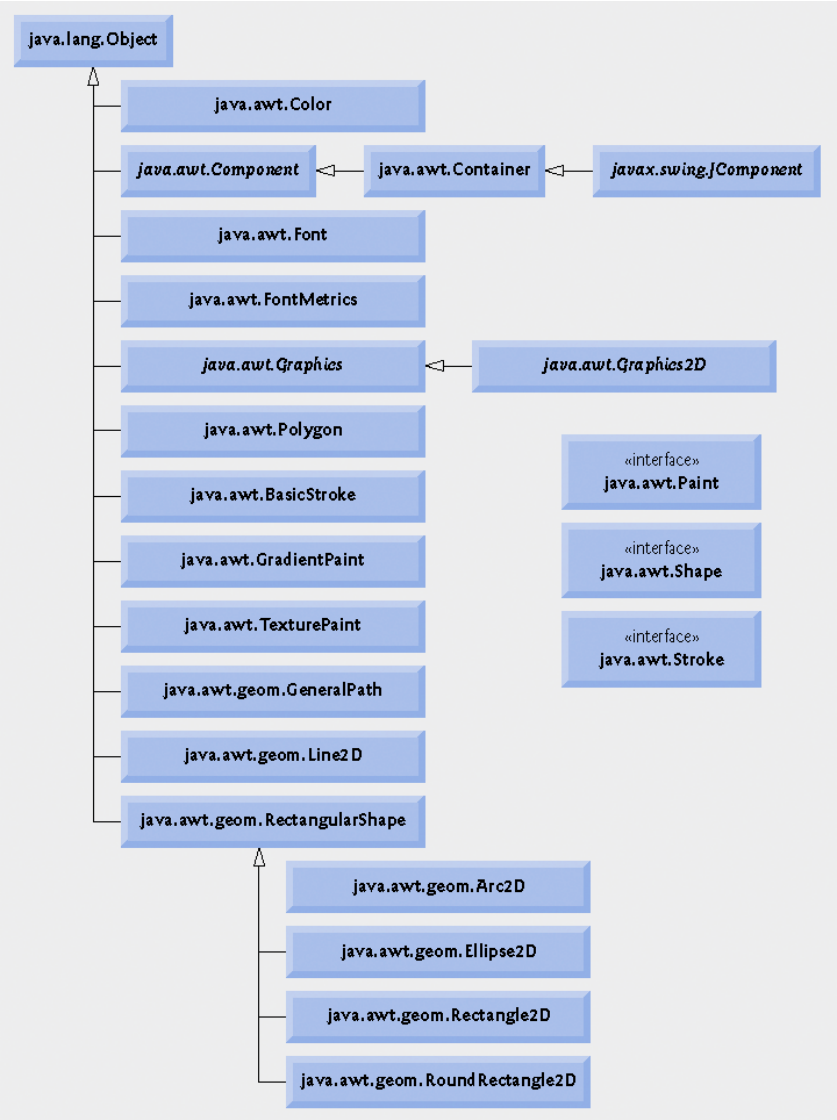


Fig. 12.1 | Classes and interfaces used in this chapter from Java's original graphics capabilities and from the Java 2D API. [*Note:* Class `Object` appears here because it is the superclass of the Java class hierarchy.]

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.**

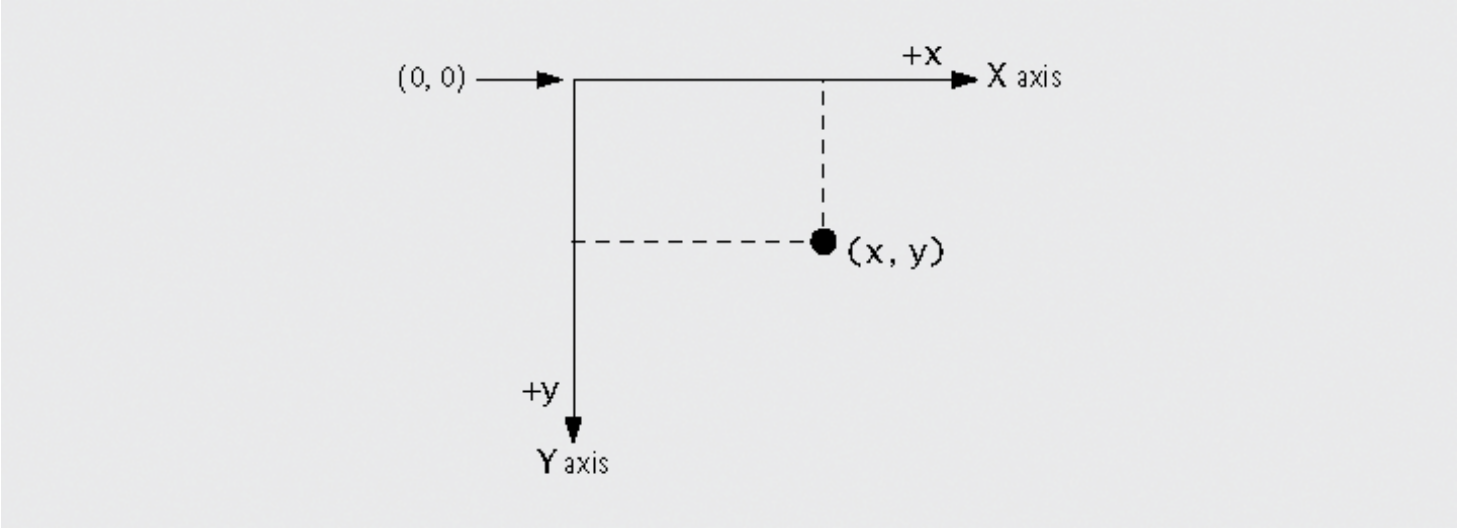


Fig. 12.2 | Java coordinate system. Units are measured in pixels.

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

Fig. 12.3 | Color constants and their RGB values.

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

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

```java
// 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 co
   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

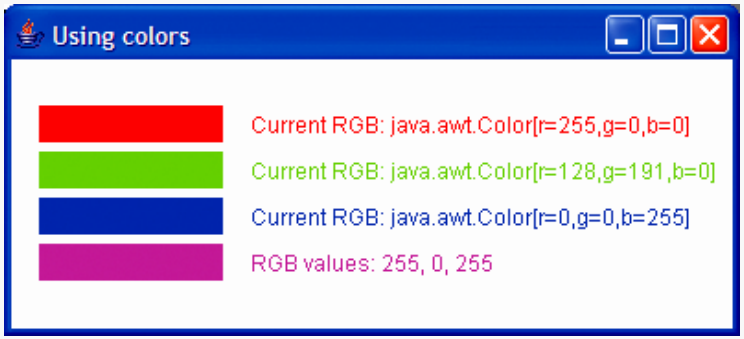
```
      // display individual RGB values
      Color color = Color.MAGENTA;
      g.setColor( color );
      g.fillRect( 15, 100, 100, 20 );
      g.drawString( "RGB values: " + color.getRed() + ", " +
         color.getGreen() + ", " + color.getBlue(), 130, 115 );
   } // end method paintComponent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

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

```java
// 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 );

      ColorJPanel colorJPanel = new ColorJPanel(); // create ColorJPanel
      frame.add( colorJPanel ); // add colorJPanel to frame
      frame.setSize( 400, 180 ); // set frame size
      frame.setVisible( true ); // displayt frame
   } // end main
} // 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

```java
// Fig. 12.7: ShowColors2JFrame.java
// Choosing colors with JColorChooser.
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JColorChooser;
import javax.swing.JPanel;

public class ShowColors2JFrame extends JFrame
{
   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(

```

Import `JColorChooser` class

```
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
4
41                   color = Color.LIGHT_GRAY;
42
43                // change content pane's background color
44                colorJPanel.setBackground( color );
45             } // end method actionPerformed
46          } // end anonymous inner class
47       ); // end call to addActionListener
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
```

Display `JColorChooser` dialog

Reference to parent component

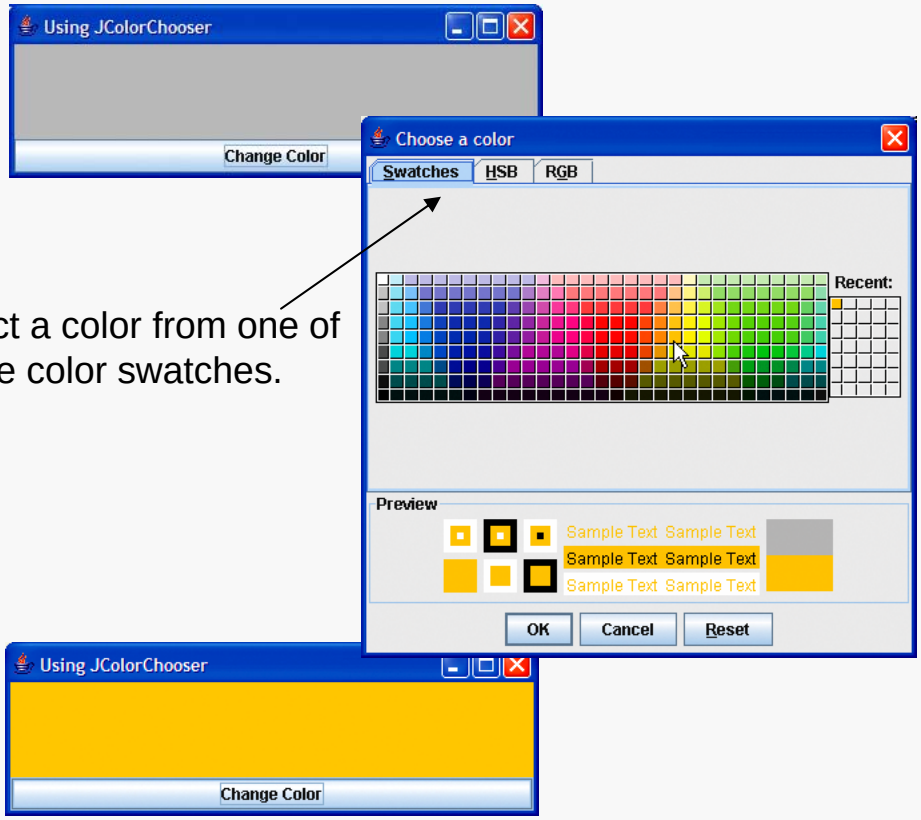
Title bar text

Initial selected color

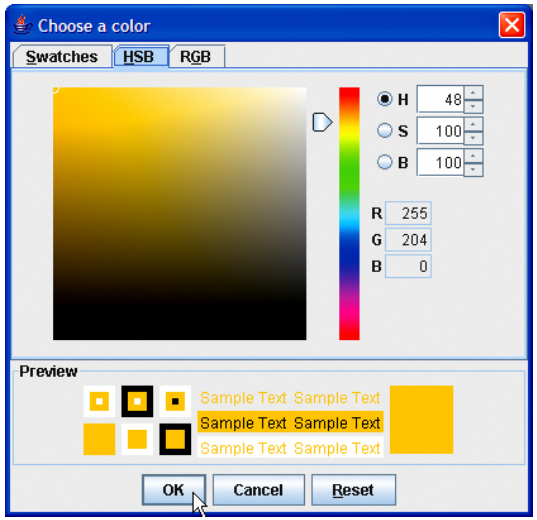
Change background color of `JPanel`

```java
// Fig. 12.8: ShowColors2.java
// Choosing colors with JColorChooser.
import javax.swing.JFrame;

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.



Sliders to select the red, green and blue color components

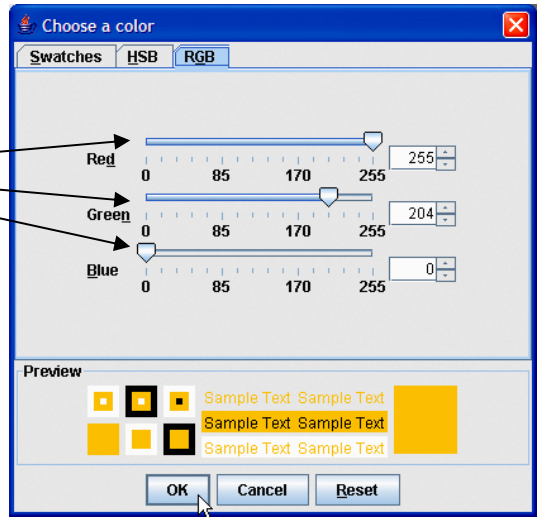


Fig. 12.9 | HSB and RGB tabs of the `JColorChooser` dialog.

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	
`public final static int PLAIN`	A constant representing a plain font style.
`public final static int BOLD`	A constant representing a bold font style.
`public final static int ITALIC`	A constant representing an italic font style.
`public Font( String name, int style, int size )`	Creates a `Font` object with the specified font name, style and size.
`public int getStyle()`	Returns an integer value indicating the current font style.
`public int getSize()`	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
`public String getName()`	Returns the current font name as a string.
`public String getFamily()`	Returns the font's family name as a string.
`public boolean isPlain()`	Returns `true` if the font is plain, else `false`.
`public boolean isBold()`	Returns `true` if the font is bold, else `false`.
`public boolean isItalic()`	Returns `true` if the font is italic, else `false`.
`Graphics` methods for manipulating `Fonts`	
`public Font getFont()`	Returns a `Font` object reference representing the current font.
`public void setFont( Font f )`	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.

```java
// 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 );

```

Font name

Font style

Font size

Creating and setting `Font` objects

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

Combining styles

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

```java
// Fig. 12.12: Fonts.java
// Using fonts.
import javax.swing.JFrame;

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

      FontJPanel fontJPanel = new FontJPanel(); // create FontJPanel
      frame.add( fontJPanel ); // add fontJPanel to frame
      frame.setSize( 420, 170 ); // set frame size
      frame.setVisible( true ); // display frame
   } // end main
} // 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

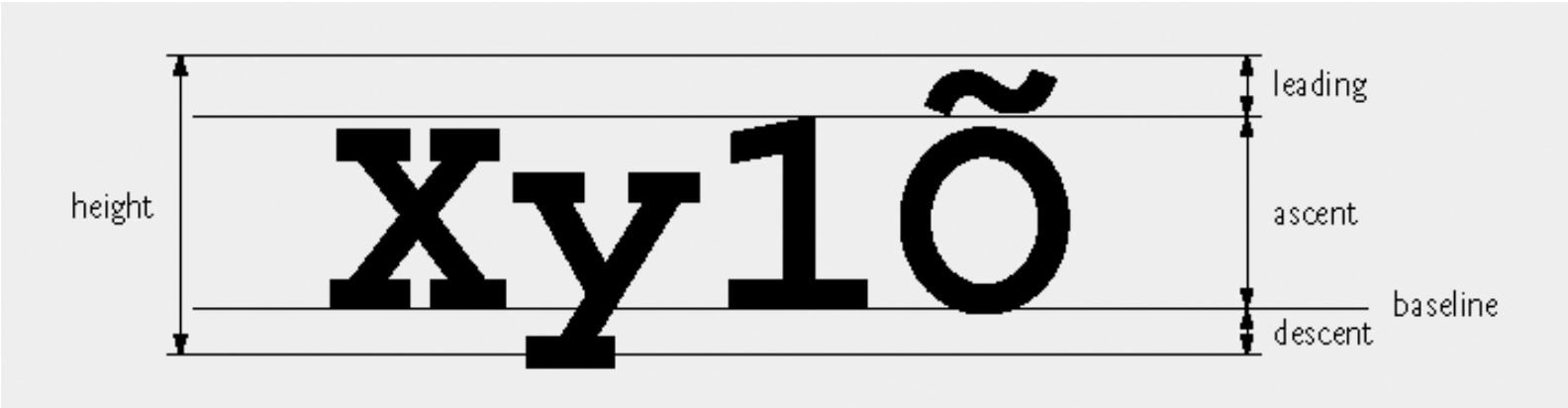


Fig. 12.13 | Font metrics.

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

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

```java
// Fig. 12.15: MetricsJPanel.java
// FontMetrics and Graphics methods useful 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 `FontMetrics` object of current `Font`

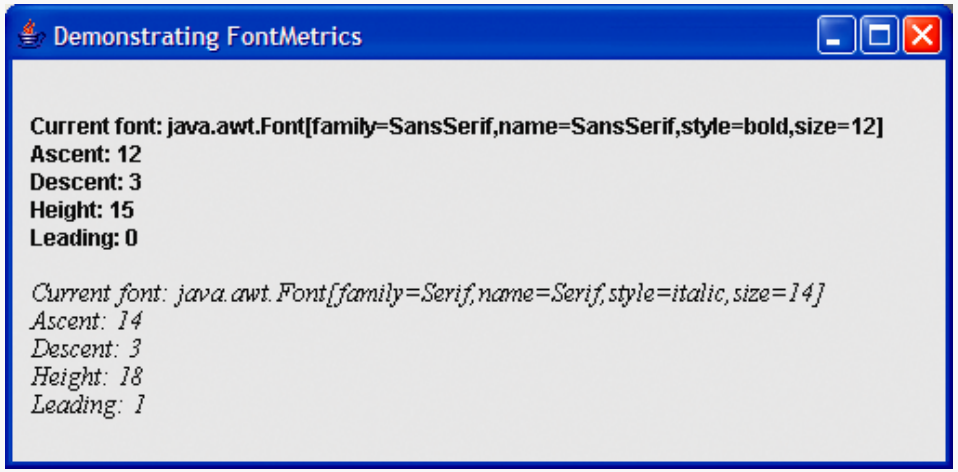
Retrieve font metric values

```
      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
```

```java
// 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
`public void drawLine( int x1, int y1, int x2, int y2 )`	Draws a line between the point (`x1`, `y1`) and the point (`x2`, `y2`).
`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 `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 `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 `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
`public void fillRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`	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).
`public void draw3DRect( int x, int y, int width, int height, boolean b )`	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.
`public void fill3DRect( int x, int y, int width, int height, boolean b )`	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.
`public void drawOval( int x, int y, int width, int height )`	Draws an oval in the current color with the specified `width` and `height`. The bounding rectangle's top-left corner is at the coordinates (`v`, `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.
`public void fillOval( int x, int y, int width, int height )`	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)

```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

```
      g.setColor( Color.CYAN );
      g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
      g.drawRoundRect( 290, 40, 90, 55, 20, 20 );

      g.setColor( Color.YELLOW );
      g.draw3DRect( 5, 100, 90, 55, true );
      g.fill3DRect( 100, 100, 90, 55, false );

      g.setColor( Color.MAGENTA );
      g.drawOval( 195, 100, 90, 55 );
      g.fillOval( 290, 100, 90, 55 );
   } // end method paintComponent
} // 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

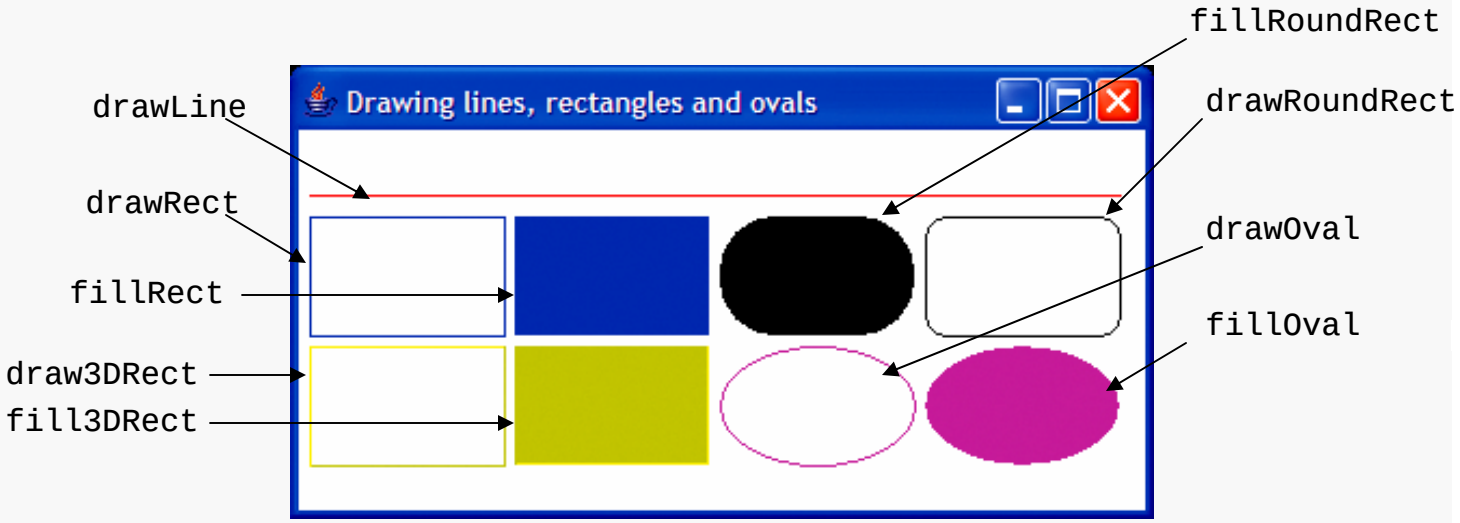
Draw a filled oval

```java
// 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 );

```

```
      LinesRectsOvalsJPanel linesRectsOvalsJPanel =
         new LinesRectsOvalsJPanel();
      linesRectsOvalsJPanel.setBackground( Color.WHITE );
      frame.add( linesRectsOvalsJPanel ); // add panel to frame
      frame.setSize( 400, 210 ); // set frame size
      frame.setVisible( true ); // display frame
   } // end main
} // end class LinesRectsOvals
```



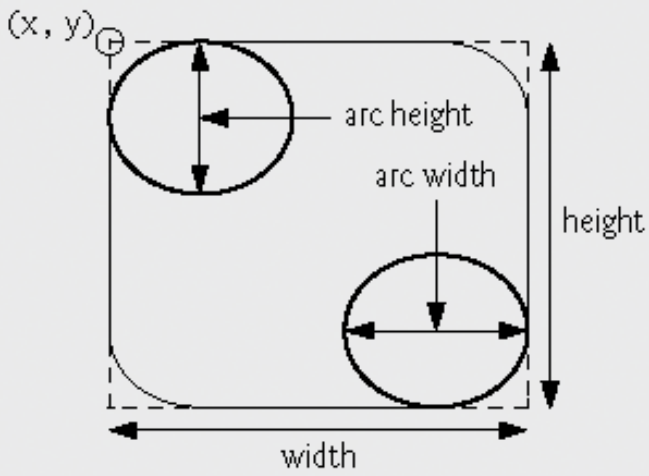


Fig. 12.20 | Arc width and arc height for rounded rectangles.

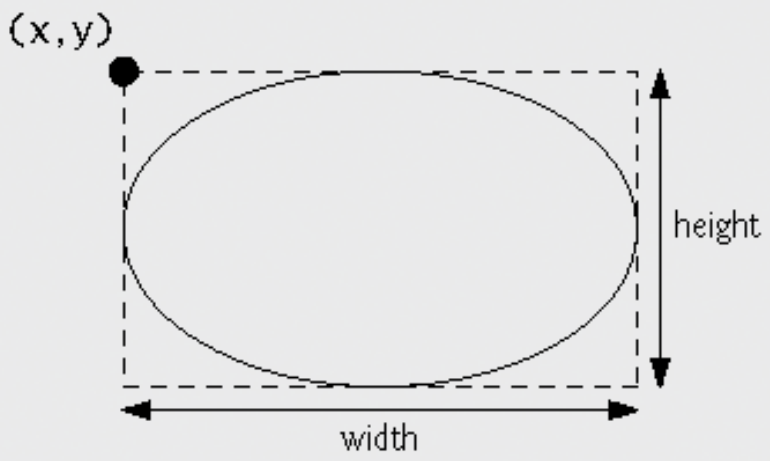


Fig. 12.21 | Oval bounded by a rectangle.

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**

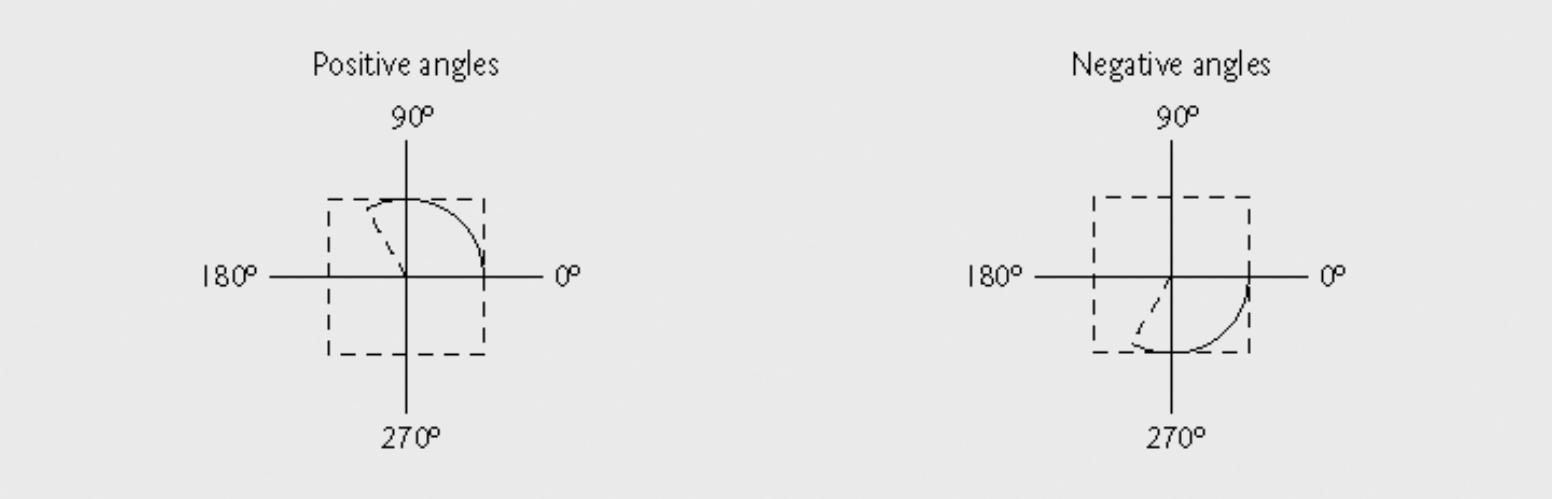


Fig. 12.22 | Positive and negative arc angles.

Method	Description
`public void drawArc( int x, int y, int width, int height, int startAngle, int arcAngle )`	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.
`public void fillArc( int x, int y, int width, int height, int startAngle, int arcAngle )`	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.

```java
// Fig. 12.24: ArcsJPanel.java
// Drawing arcs.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class ArcsJPanel extends JPanel
{
   // draw rectan
   public void pa
   {
      super.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

```
      // start at 0 and sweep -270 degrees
      g.setColor( Color.RED );
      g.drawRect( 185, 35, 80, 80 );
      g.setColor( Color.BLACK );
      g.drawArc( 185, 35, 80, 80, 0, -270 );

      // start at 0 and sweep 360 degrees
      g.fillArc( 15, 120, 80, 40, 0, 360 );

      // start at 270 and sweep -90 degrees
      g.fillArc( 100, 120, 80, 40, 270, -90 );

      // start at 0 and sweep -270 degrees
      g.fillArc( 185, 120, 80, 40, 0, -270 );
   } // end method paintComponent
} // end class ArcsJPanel
```

Draw filled arcs

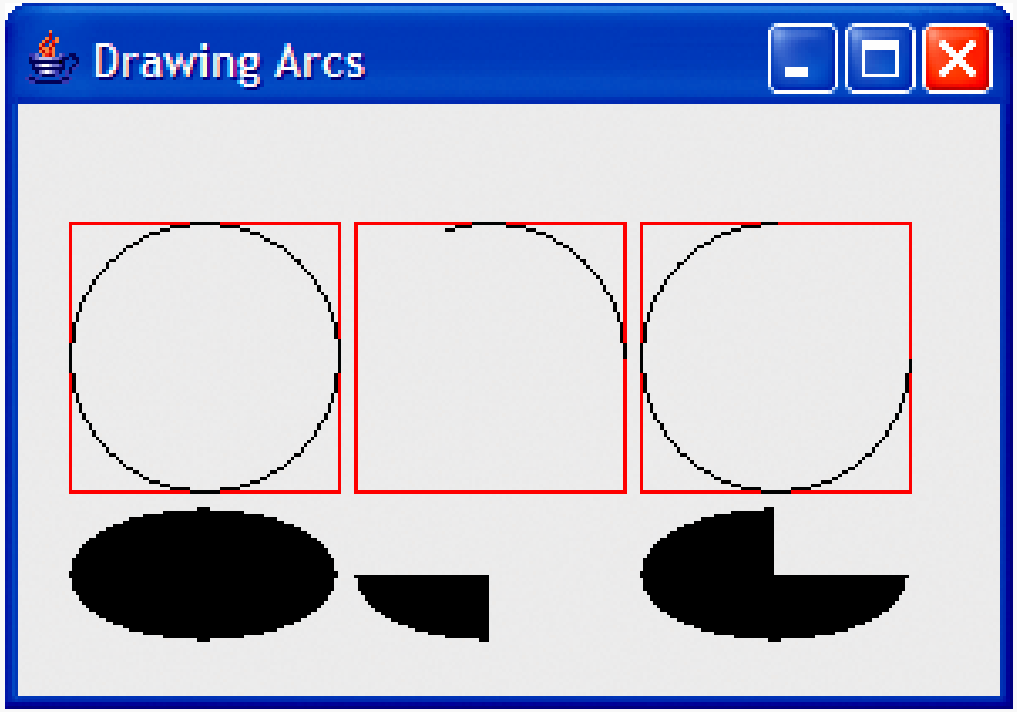
Negative values indicate arc should sweep clockwise

```java
// 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 );

```

```
      ArcsJPanel arcsJPanel = new ArcsJPanel(); // create ArcsJPanel
      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**

Method	Description
`Graphics` *methods for drawing polygons*	
`public void drawPolygon( int xPoints[], int yPoints[], int points )`	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.
`public void drawPolyline( int xPoints[], int yPoints[], int points )`	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.
`public void drawPolygon( Polygon p )`	Draws the specified polygon.
`public void fillPolygon( int xPoints[], int yPoints[], int points )`	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.

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)

```java
// Fig. 12.27: PolygonsJPanel.java
// Drawing polygons.
import java.awt.Graphics;
import java.awt.Polygon;
import javax.swing.JPanel;

public class PolygonsJPanel extends JPanel
{
   // draw polygons and polylines
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g ); // call superclass's paintComponent

      // draw polygon with Polygon object
      int xValues[] = { 20, 40, 50, 30, 20, 15 };
      int yValues[] = { 50, 50, 60, 80, 80, 60 };
      Polygon polygon1 = new Polygon( xValues, yValues, 6 );
      g.drawPolygon( polygon1 );

      // draw polylines with two arrays
      int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };
      int yValues2[] = { 100, 100, 110, 110, 130, 110,
      g.drawPolyline( xValues2, yValues2, 7 );

```

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

Draw an empty `Polygon`

Draw polyline from sets of *x*- and *y*-coordinates

```
      // 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 );
   } // end method paintComponent
} // end class PolygonsJPanel
```

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

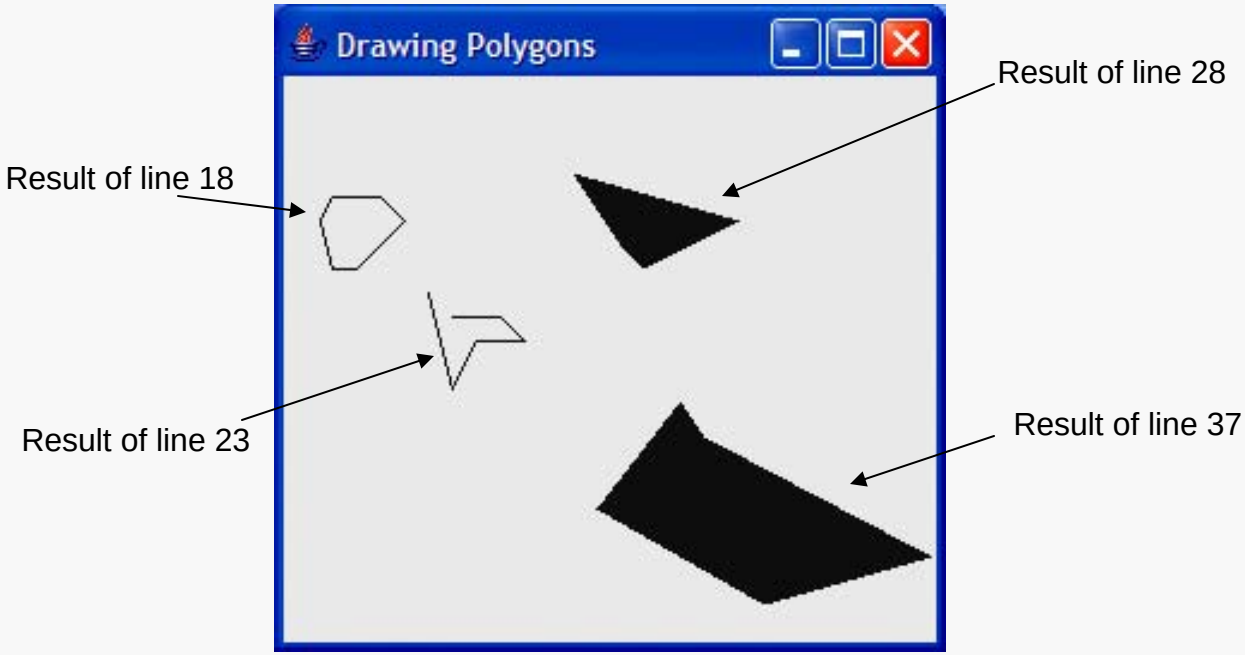
Add coordinates to `Polygon` with method `addPoint`

```java
// Fig. 12.28: DrawPolygons.java
// Drawing polygons.
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 );

```

```
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
```



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

```java
// 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

```

Java 2D API shape classes

Creating a `Graphics2D` reference

```
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 bufferImage 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

```
      // paint buffImage onto the JFrame
      g2d.setPaint( new TexturePaint( buffImage,
         new Rectangle( 10, 10 ) ) );
      g2d.fill(
         new RoundRectangle2D.Double( 155, 30, 75,

      // draw 2D pie-shaped arc in white
      g2d.setPaint( Color.WHITE );
      g2d.setStroke( new BasicStroke( 6.0f ) );
      g2d.draw(
         new Arc2D.Double( 240, 30, 75, 100, 0, 270, Arc2D.PIE ) );

      // draw 2D lines in green and yellow
      g2d.setPaint( Color.GREEN );
      g2d.draw( new Line2D.Double( 395, 30, 320, 150 ) );

      // draw 2D line using stroke
      float dashes[] = { 10 }; // specify dash pattern
      g2d.setPaint( Color.YELLOW );
      g2d.setStroke( new BasicStroke( 4, BasicStroke.CAP_ROUND,
         BasicStroke.JOIN_ROUND, 10, dashes, 0 ) );
      g2d.draw( new Line2D.Double( 320, 30, 395, 150 ) );
   } // end method paintComponent
} // 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

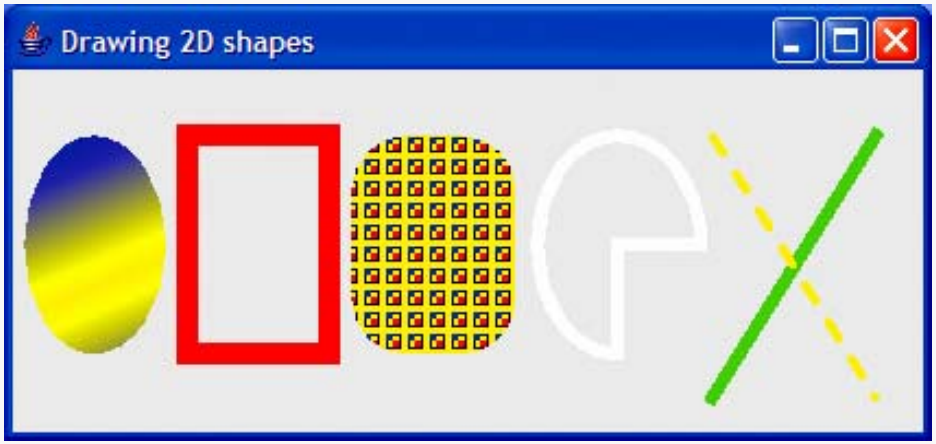
Draw dashed yellow line

```java
// 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

```java
// Fig. 12.31: Shapes2JPanel.java
// Demonstrating a general path.
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.GeneralPath;
import java.util.Random;
import javax.swing.JPanel;

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
      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 General Path
      star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );

```

Create `GeneralPath` object

Set starting point of `GeneralPath` object

```
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 origi
34
35       // rotate around origin and c
36       for ( int count = 1; count <= 20; 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.nextIn
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

```java
// 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 );

```

```
      Shapes2JPanel shapes2JPanel = new Shapes2JPanel();
      frame.add( shapes2JPanel ); // add shapes2JPanel 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
```