JAVA™ PROGRAMMING

# Chapter 16: Graphics

# Objectives

- Learn about the `paint()` and `repaint()` methods
- Use the `drawString()` method to draw `Strings` using various fonts and colors
- Draw lines and shapes
- Learn more about fonts
- Draw with Java 2D graphics

# Learning About the `paint()` and `repaint()` Methods – Part 1

- **Rerender**
  - To redisplay a display surface
- **Painting**
  - **System-triggered painting**
  - **Application-triggered painting**
- **`paint()` method**
  - Write your own method to override the default
  - Method header
    - `public void paint(Graphics g)`

# Learning About the `paint()` and `repaint()` Methods – Part 2

- `Graphics` object
  - Preconfigured with the appropriate values for drawing on the component
- **`repaint()` method**
  - Use when a window needs to be updated
  - Calls the `paint()` method
  - Creates a `Graphics` object

# Using the `setLocation()` Method

- Place a component at a specific location within a `JFrame`'s content pane
- Change the position of a component by using the **`setLocation()` method**
  - `pressMe.setLocation(100, 50);`

# Creating `Graphics` Objects

- Call the `paint()` method
  - Use the automatically created `Graphics` object
  - Instantiate any `Graphics` object
    - `Graphics draw = getGraphics();`
  - `getGraphics()` method

# Using the `drawString()` Method – Part 1

- **`drawString()` method**
  - Allows you to draw a `String` in a `JFrame` window
  - Requires three arguments:
    - `String`
    - x-axis coordinate
    - y-axis coordinate
  - Is a member of the `Graphics` class

# Using the `drawString()` Method – Part 2

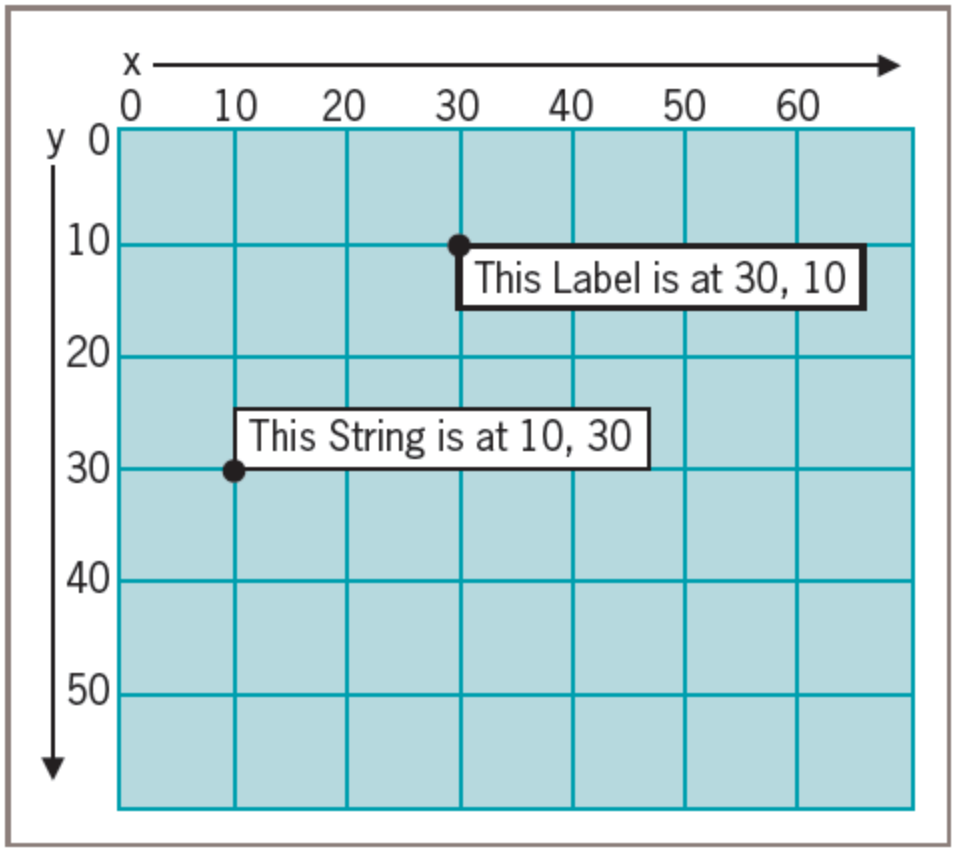


**Figure 16-5** Placement of `String` and `JLabel` objects on a frame

# Using the `setFont()` and `setColor()` Methods

- `setFont()` method
  - Requires a `Font` object
- You can instruct a `Graphics` object to use a font
  - `somegraphicsobject.setFont(someFont);`

# Using Color

- `setColor()` method
  - Designates a `Graphics` color
  - Use 13 `Color` class constants as arguments
    - `brush.setColor(Color.GREEN);`

# Drawing Lines and Shapes

- Java provides several methods for drawing a variety of lines and geometric shapes

# Drawing Lines

- **`drawLine()` method**
  - Draws a straight line between any two points
  - Takes four arguments:
    - x- and y-coordinates of the line's starting point
    - x- and y-coordinates of the line's ending point

# Drawing Rectangles – Part 1

- **`drawRect()` method**
  - Draws the outline of a rectangle
- **`fillRect()` method**
  - Draws a solid or filled rectangle
- Both require four arguments:
  - x- and y-coordinates of the upper-left corner of the rectangle
  - The width and height of the rectangle

# Drawing Rectangles – Part 2

- **`clearRect()` method**
  - Draws a rectangle
  - Requires four arguments:
    - x- and y-coordinates of the upper-left corner of the rectangle
    - The width and height of the rectangle
  - Appears empty or “clear”
- **`drawRoundRect()` method**
  - Creates rectangles with rounded corners
  - Requires six arguments

# Creating Shadowed Rectangles

- **`draw3DRect()` method**
  - A minor variation on the `drawRect()` method
  - Draws a rectangle that appears to have "shadowing" on two edges
  - Contains a Boolean value argument:
    - `true` if the rectangle is darker on the right and bottom
    - `false` if the rectangle is darker on the left and top
- **`fill3DRect()` method**
  - Creates filled three-dimensional rectangles

# Drawing Ovals

- **`drawOval()` and `fillOval()` methods**
  - Draw ovals using the same four arguments that rectangles use

# Drawing Arcs – Part 1

- **`drawArc()` method** arguments:
  - x- and y-coordinates of the upper-left corner of an imaginary rectangle that represents the bounds of the imaginary circle that contains the arc
  - The width and height of the imaginary rectangle that represents the bounds of the imaginary circle that contains the arc
  - The beginning arc position
  - The arc angle

# Drawing Arcs – Part 2

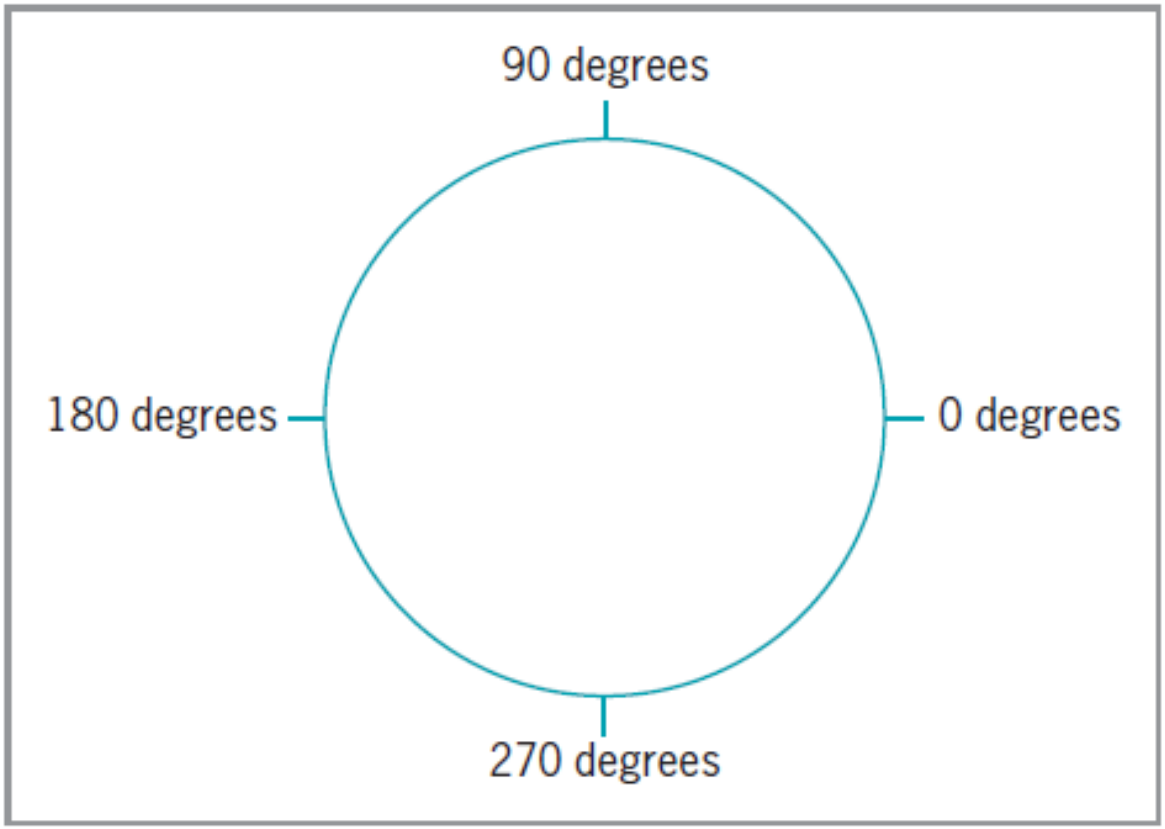


**Figure 16-18** Arc positions

# Drawing Arcs – Part 3

- **`fillArc()` method**
  - Creates a solid arc
    - Two straight lines are drawn from the arc endpoints to the center of the imaginary circle whose perimeter the arc occupies

# Creating Polygons – Part 1

- **`drawPolygon()` method**
  - Draws complex shapes
  - Requires three arguments:
    - The integer array, which holds a series of x-coordinate positions
    - The second array, which holds a series of corresponding y-coordinate positions
    - The number of pairs of points to connect

# Creating Polygons – Part 2

- **`fillPolygon()` method**
  - Draws a solid shape
  - If the beginning and ending points are not identical, two endpoints are connected by a straight line before the polygon is filled with color
- **`addPoint()` method**
  - Adds points to a polygon indefinitely

# Copying an Area

- **`copyArea()` method**
  - Requires six parameters:
    - The x- and y-coordinates of the upper-left corner of the area to be copied
    - The width and height of the area to be copied
    - The horizontal and vertical displacement of the destination of the copy

# Using the `paintComponent()` Method with `JPanels`

- Use the `paintComponent()` method when creating drawings on a `JPanel`
- `JFrame` is not a child of `JComponent`
  - Does not have its own `paintComponent()` method

# Learning More About Fonts

- **`getAvailableFontFamilyNames()` method**
  - Is part of the `GraphicsEnvironment` class defined in the `java.awt` package
  - Returns an array of `String` objects that are names of available fonts
- You cannot instantiate the `GraphicsEnvironment` object directly
  - Get the reference object to the current computer environment
    - Call the static `getLocalGraphicsEnvironment()` method

# Discovering Screen Statistics Using the `Toolkit` Class – Part 1

- **`getDefaultToolkit()` method**
  - Provides information about the system in use
- **`getScreenResolution()` method**
  - Returns the number of pixels as an integer
- You can create a `Toolkit` object and get the screen resolution using the following code:

```
Toolkit tk = Toolkit.getDefaultToolkit();
int resolution = tk.getScreenResolution();
```

# Discovering Screen Statistics Using the `Toolkit` Class – Part 2

- `Dimension` class
  - Use for representing the width and height of a user interface component
  - Constructors:
    - `Dimension()` creates an instance of `Dimension` with a width and height of 0
    - `Dimension(Dimension d)` creates an instance of `Dimension` whose width and height are the same as for the specified dimension
    - `Dimension(int width, int height)` constructs a `Dimension` and initializes it to the specified width and height

# Discovering Screen Statistics Using the `Toolkit` Class – Part 3

- **`getScreenSize()` method**
  - Is a member of the `Toolkit` object
  - Returns an object of type `Dimension`, which specifies the width and height of the screen in pixels

# Discovering Font Statistics – Part 1

- **Leading**
  - The amount of space between baselines
- **Ascent**
  - The height of an uppercase character from the baseline to the top of the character
- **Descent**
  - Measures the parts of characters that “hang below” the baseline
- **Height of a font**
  - The sum of leading, ascent, and descent

# Discovering Font Statistics – Part 2

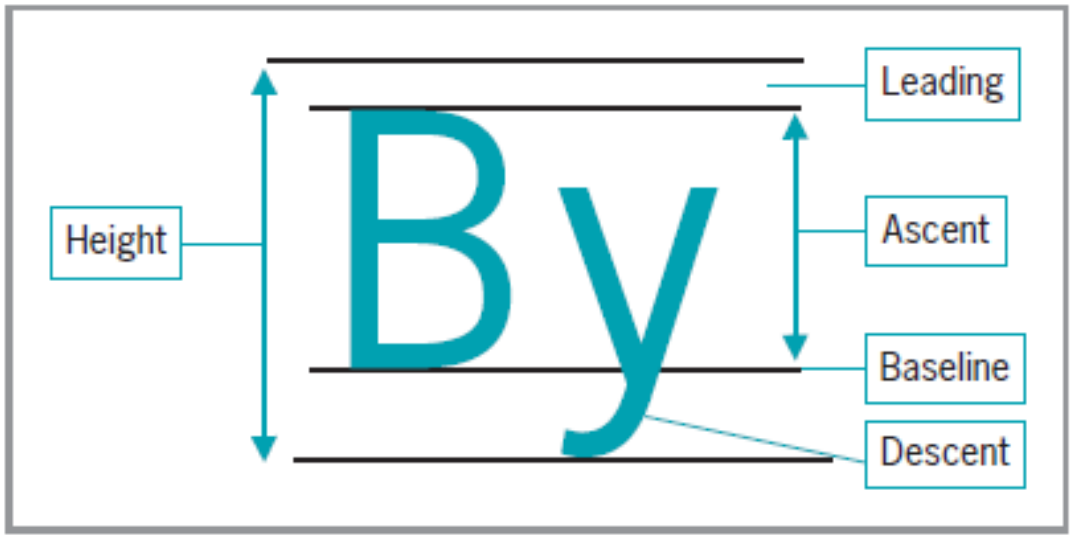


**Figure 16-29** Parts of a font's height

# Discovering Font Statistics – Part 3

- **`getFontMetrics()` method**
  - Discovers a font's height
  - Returns the `FontMetrics` object
- Use one of the `FontMetrics` class methods with the object to return one of a `Font`'s statistics:
  - `public int getLeading()`
  - `public int getAscent()`
  - `public int getDescent()`
  - `public int getHeight()`

# Discovering Font Statistics – Part 4

- **`stringWidth()` method**
  - Returns the integer width of a `String`
  - Requires the name of the `String`
  - Is a member of the `FontMetrics` class

# Drawing with Java 2D Graphics – Part 1

- Java 2D
  - Higher quality, two-dimensional (2D) graphics, images, and text
- **`Graphics2D` class**
  - Features include:
    - Fill patterns
    - Strokes
    - Anti-aliasing

# Drawing with Java 2D Graphics – Part 2

- **`Graphics2D` class** (cont'd.)
  - Found in the `java.awt` package
  - Produced by casting, or converting, and promoting a `Graphics` object
- The process of drawing with Java 2D objects:
  - Specify the rendering attributes
  - Set a drawing stroke
  - Create objects to draw

# Specifying the Rendering Attributes – Part 1

- Use the `setColor()` method
  - Specify 2D colors
  - Use a `Graphics2D` object and set the color to black
    - `gr2D.setColor(Color.BLACK);`
- **Fill patterns**
  - Control how a drawing object is filled in
  - Can be a solid, gradient, texture, or pattern
  - Created by using the `setPaint()` method of `Graphics2D` with a fill pattern object

# Specifying the Rendering Attributes – Part 2

- **Gradient fill**
  - A gradual shift from one color at one coordinate point to a different color at a second coordinate point
  - **Acyclic gradient**
  - **Cyclic gradient**

# Figure 16-32 The Jgradient class

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.Color;
public class JGradient extends JFrame
{
   public void paint(Graphics gr)
   {
      super.paint(gr);
      int x = 20, y = 40, x2 = 180, y2 = 100;
      Graphics2D gr2D = (Graphics2D)gr;
      gr2D.setPaint(new GradientPaint(x, y, Color.LIGHT_GRAY,
         x2, y2, Color.DARK_GRAY, false));
      gr2D.fill(new Rectangle2D.Double(x, y, x2, y2));
      x = 210;
      gr2D.setPaint(new GradientPaint(x, y, Color.LIGHT_GRAY,
         x2, y2, Color.DARK_GRAY, true));
      gr2D.fill(new Rectangle2D.Double(x, y, x2, y2));
   }
   public static void main(String[] args)
   {
      JGradient frame = new JGradient();
      frame.setSize(440, 180);
      frame.setVisible(true);
   }
}
```

**Figure 16-32** The JGradient class

# Setting a Drawing Stroke – Part 1

- **Stroke**
  - Represents a single movement
  - **`setStroke()` method**
- `Stroke` interface
- **`BasicStroke`** class
- **Endcap styles**
  - Apply to the ends of lines that do not join with other lines
- **Juncture styles**
  - For lines that join

# Setting a Drawing Stroke – Part 2

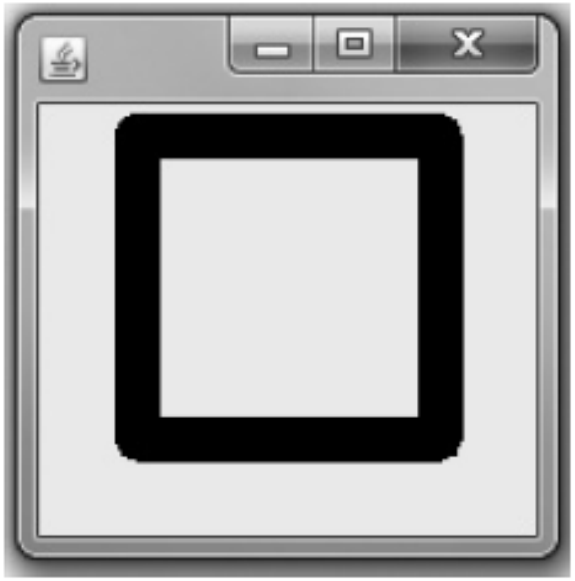

Figure 16-35 Output of the `JStroke` program

# Creating Objects to Draw – Part 1

- Objects drawn in Java 2D are created by defining them as geometric shapes
  - Use the `java.awt.geom` package classes
  - Define the shape
  - Use the shape as an argument to the `draw()` or `fill()` methods

***Lines***

- `Line2D.Float`
- `Line2D.Double`
- `Point2D.Float`
- `Point2D.Double`

# Creating Objects to Draw – Part 2

***Rectangles***

- `Rectangle2D.Float`
- `Rectangle2D.Double`
- `Rectangle2D.Float rect = new Rectangle2D.Float(10F, 10F, 50F, 40F);`

***Ovals***

- `Ellipse2D.Float`
- `Ellipse2D.Double`
- `Ellipse2D.Float ell = new Ellipse2D.Float(10F, 73F, 40F, 20F);`

# Creating Objects to Draw – Part 3

***Arcs***

- `Arc2D.Float`
- `Arc2D.Double`
- `Arc2D.PIE`
- `Arc2D.CHORD`
- `Arc2D.OPEN`
- `Arc2D.Float ac = new Arc2D.Float (10,133,30,33,30,120,Arc2D.PIE);`

# Creating Objects to Draw – Part 4

***Polygons***

- Define movements from one point to another
- `GeneralPath` object
- `GeneralPath pol = new GeneralPath();`
- `moveTo()`
- `lineTo()`
- `closePath()`

# You Do It – Part 2

- Using `FontMetrics` Methods to Compare Fonts
- Using `FontMetrics` Methods to Place a Border Around a `String`
- Using Drawing Strokes
- Working with Shapes

# Don’t Do It

- Don’t forget to call `super.paint()` as the first statement in the `paint()` method
- Don’t forget that the `setLocation()` method works correctly only when it is used after the layout manager has finished positioning all of the application’s components
- Don’t forget that the lower-left corner of a `String` is placed at the coordinates used when you call `drawString()`

# You Do It – Part 1

- Using the `drawString()` Method
- Using Fonts and Colors
- Creating Your Own `Graphics` Object
- Examining Screen Coordinates
- Creating a Drawing
- Copying an Area
- Don't forget to use `paintComponent()` rather than `paint()` when creating graphics on a `JPanel`

# Summary

- `paint()` method
- `drawString()` method
  - Draws a `String` in a `JApplet` window
- Methods for drawing a variety of lines and geometric shapes
- `getAvailableFontFamilyNames()` method
  - Discovers fonts available on a system
- Java 2D
  - Higher quality