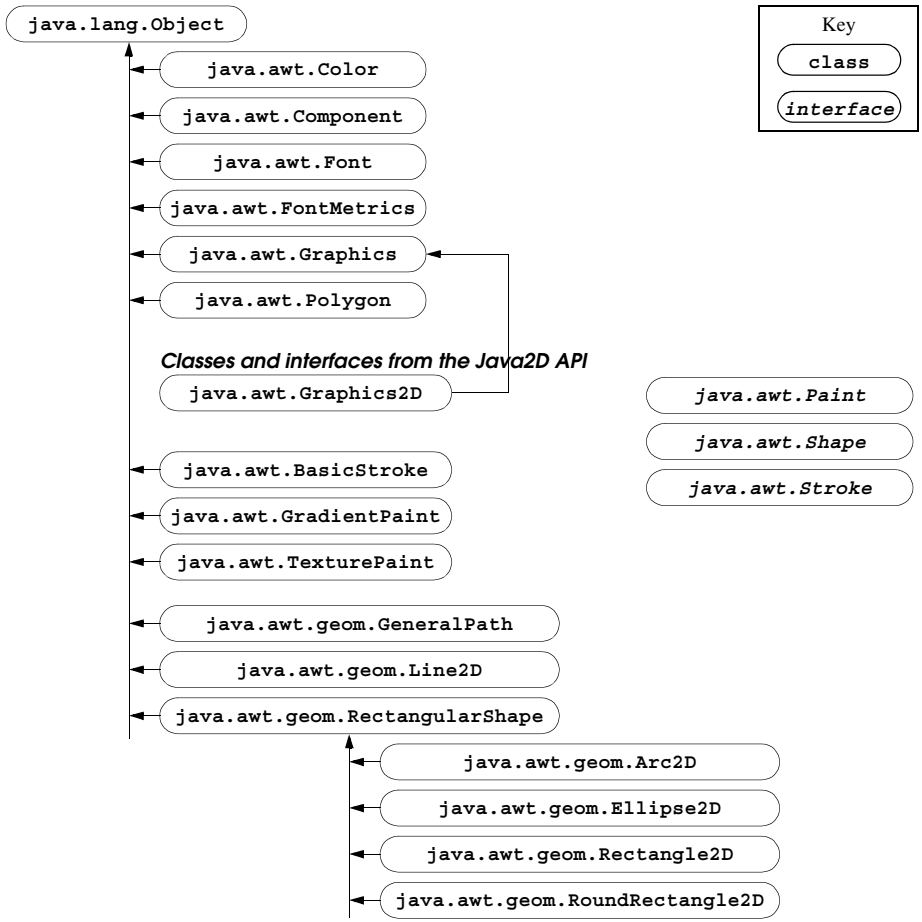

Fig. 11.1 Some classes and interfaces used in this chapter from Java's original graphics capabilities and from the Java2D API.

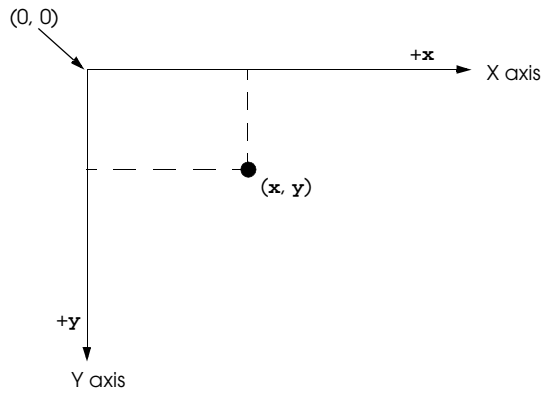

Fig. 11.2 Java coordinate system. Units are measured in pixels.

`Color` Constant	Color	RGB value
`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 lightGray`	light gray	192, 192, 192
`public final static Color darkGray`	dark gray	64, 64, 64
`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

Fig. 11.3 `Color` class `static` constants and RGB values .

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

Fig. 11.4 `Color` methods and color-related `Graphics` methods.

```
1  // Fig. 11.5: ShowColors.java
2  // Demonstrating Colors
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class ShowColors extends JFrame {
8     public ShowColors()
9     {
10       super( "Using colors" );
11
12       setSize( 400, 130 );
13       show();
14    }
15
16    public void paint( Graphics g )
17    {
18       // set new drawing color using integers
19       g.setColor( new Color( 255, 0, 0 ) );
20       g.fillRect( 25, 25, 100, 20 );
21       g.drawString( "Current RGB: " + g.getColor(), 130, 40 );
22
23       // set new drawing color using floats
24       g.setColor( new Color( 0.0f, 1.0f, 0.0f ) );
25       g.fillRect( 25, 50, 100, 20 );
26       g.drawString( "Current RGB: " + g.getColor(), 130, 65 );
27
28       // set new drawing color using static Color objects
29       g.setColor( Color.blue );
30       g.fillRect( 25, 75, 100, 20 );
31       g.drawString( "Current RGB: " + g.getColor(), 130, 90 );
32
33       // display individual RGB values
34       Color c = Color.magenta;
35       g.setColor( c );
36       g.fillRect( 25, 100, 100, 20 );
37       g.drawString( "RGB values: " + c.getRed() + ", " +
38          c.getGreen() + ", " + c.getBlue(), 130, 115 );
39    }
40
41    public static void main( String args[] )
42    {
43       ShowColors app = new ShowColors();
44
45       app.addWindowListener(
46          new WindowAdapter() {
47             public void windowClosing( WindowEvent e )
48             {
49                System.exit( 0 );
50             }
51          }
52       );
53    }
54 }
```

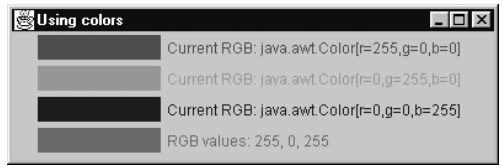

Fig. 11.5 Demonstrating setting and getting a **`Color`**.

```
1  // Fig. 11.6: ShowColors2.java
2  // Demonstrating JColorChooser
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class ShowColors2 extends JFrame {
8     private JButton changeColor;
9     private Color color = Color.lightGray;
10    private Container c;
11
12    public ShowColors2()
13    {
14       super( "Using JColorChooser" );
15
16       c = getContentPane();
17       c.setLayout( new FlowLayout() );
18
19       changeColor = new JButton( "Change Color" );
20       changeColor.addActionListener(
21          new ActionListener() {
22             public void actionPerformed( ActionEvent e )
23             {
24                color =
25                   JColorChooser.showDialog( ShowColors2.this,
26                      "Choose a color", color );
27
28                if ( color == null )
29                   color = Color.lightGray;
30
31                c.setBackground( color );
32                c.repaint();
33             }
34          }
35       );
36       c.add( changeColor );
37
38       setSize( 400, 130 );
39       show();
40    }
41
42    public static void main( String args[] )
43    {
44       ShowColors2 app = new ShowColors2();
45
46       app.addWindowListener(
47          new WindowAdapter() {
48             public void windowClosing( WindowEvent e )
49             {
50                System.exit( 0 );
51             }
52          }
53       );
54    }
55 }
```

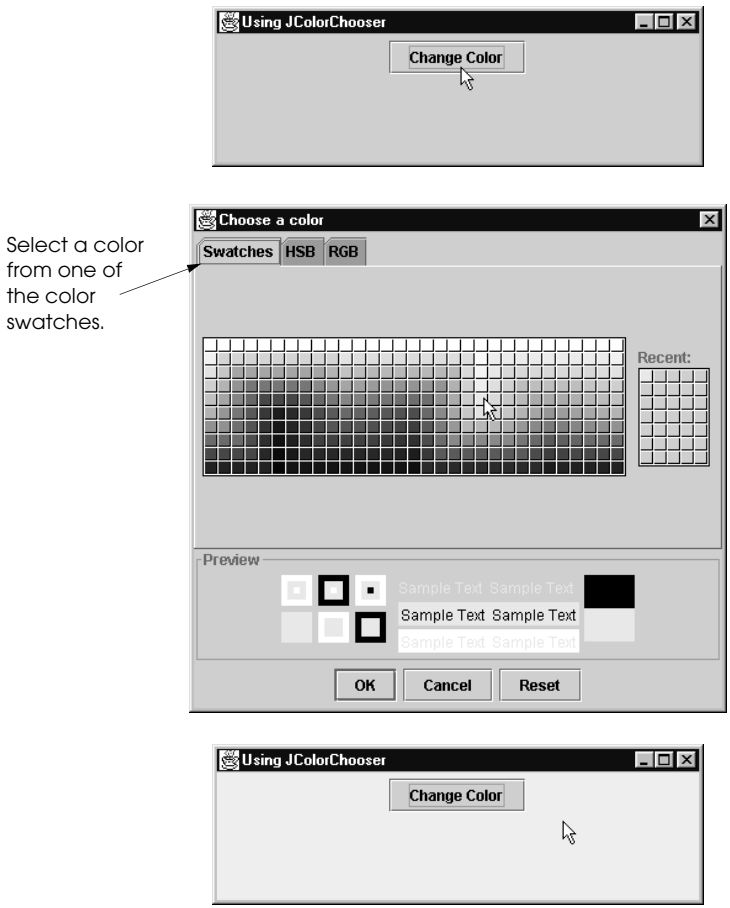

Fig. 11.6 Demonstrating the **`JColorChooser`** dialog.

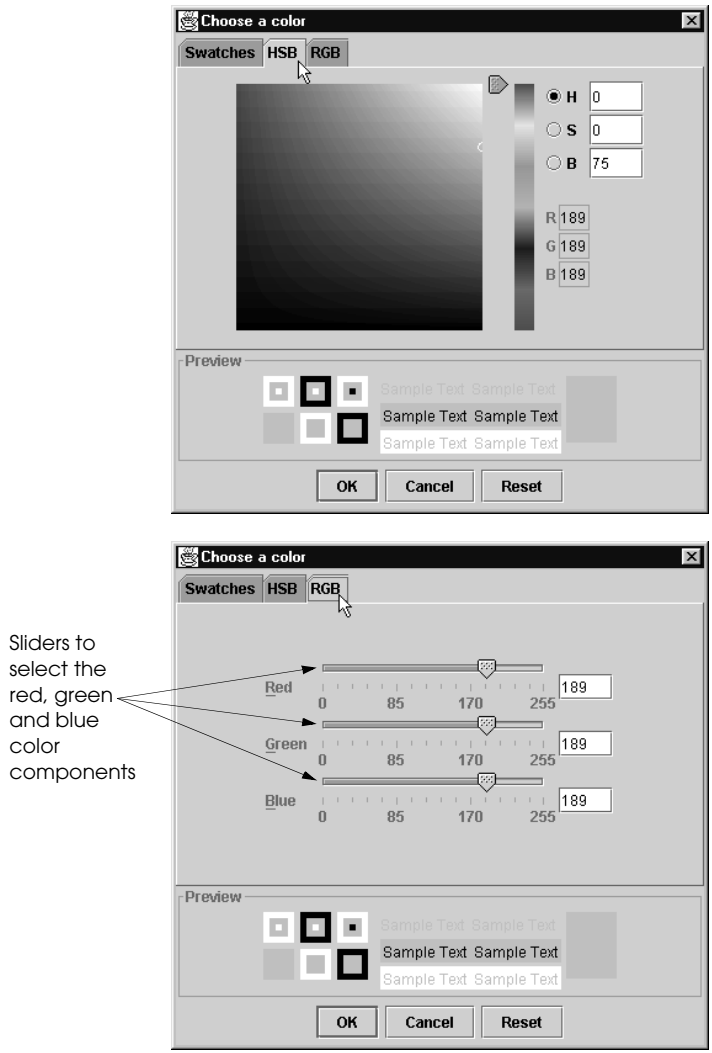

Fig. 11.7 The **HSB** and **RGB** tabs of the `JColorChooser` dialog.

.

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

Fig. 11.8 **`Font`** methods, constants and font-related **`Graphics`** methods.

```
1  // Fig. 11.9: Fonts.java
2  // Using fonts
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class Fonts extends JFrame {
8     public Fonts()
9     {
10       super( "Using fonts" );
11
12       setSize( 420, 125 );
13       show();
14    }
15
16    public void paint( Graphics g )
17    {
18       // set current font to Serif (Times), bold, 12pt
19       // and draw a string
20       g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
21       g.drawString( "Serif 12 point bold.", 20, 50 );
22
23       // set current font to Monospaced (Courier),
24       // italic, 24pt and draw a string
25       g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
26       g.drawString( "Monospaced 24 point italic.", 20, 70 );
27
28       // set current font to SansSerif (Helvetica),
29       // plain, 14pt and draw a string
30       g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
31       g.drawString( "SansSerif 14 point plain.", 20, 90 );
32
33       // set current font to Serif (times), bold/italic,
34       // 18pt and draw a string
35       g.setColor( Color.red );
36       g.setFont(
37          new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
38       g.drawString( g.getFont().getName() + " " +
39                     g.getFont().getSize() +
40                     " point bold italic.", 20, 110 );
41    }
42
43    public static void main( String args[] )
44    {
45       Fonts app = new Fonts();
46
47       app.addWindowListener(
48          new WindowAdapter() {
49             public void windowClosing( WindowEvent e )
50             {
51                System.exit( 0 );
52             }
53          }
54       );
55    }
```

```
56    }
```

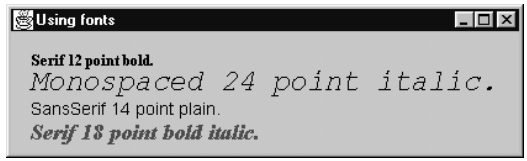

Fig. 11.9 Using **`Graphics`** method **`setFont`** to change **`Font`**s.

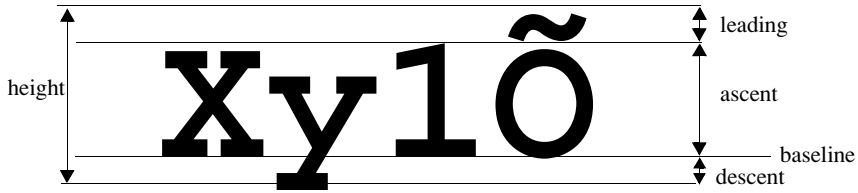

Fig. 11.10 Font metrics.

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

Fig. 11.11 `FontMetrics` and `Graphics` methods for obtaining font metrics.

```
1  // Fig. 11.12: Metrics.java
2  // Demonstrating methods of class FontMetrics and
3  // class Graphics useful for obtaining font metrics
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Metrics extends JFrame {
9     public Metrics()
10    {
11       super( "Demonstrating FontMetrics" );
12
13       setSize( 510, 210 );
14       show();
15    }
16
17    public void paint( Graphics g )
18    {
19       g.setFont( new Font( "SansSerif", Font.BOLD, 12 ) );
20       FontMetrics fm = g.getFontMetrics();
21       g.drawString( "Current font: " + g.getFont(), 10, 40 );
22       g.drawString( "Ascent: " + fm.getAscent(), 10, 55 );
23       g.drawString( "Descent: " + fm.getDescent(), 10, 70 );
24       g.drawString( "Height: " + fm.getHeight(), 10, 85 );
25       g.drawString( "Leading: " + fm.getLeading(), 10, 100 );
26
27       Font font = new Font( "Serif", Font.ITALIC, 14 );
28       fm = g.getFontMetrics( font );
29       g.setFont( font );
30       g.drawString( "Current font: " + font, 10, 130 );
31       g.drawString( "Ascent: " + fm.getAscent(), 10, 145 );
32       g.drawString( "Descent: " + fm.getDescent(), 10, 160 );
33       g.drawString( "Height: " + fm.getHeight(), 10, 175 );
34       g.drawString( "Leading: " + fm.getLeading(), 10, 190 );
35    }
36
37    public static void main( String args[] )
38    {
39       Metrics app = new Metrics();
40
41       app.addWindowListener(
42          new WindowAdapter() {
43             public void windowClosing( WindowEvent e )
44             {
45                System.exit( 0 );
46             }
47          }
48       );
49    }
50 }
```

Fig. 11.12 Obtaining font metric information (part 1 of 2).

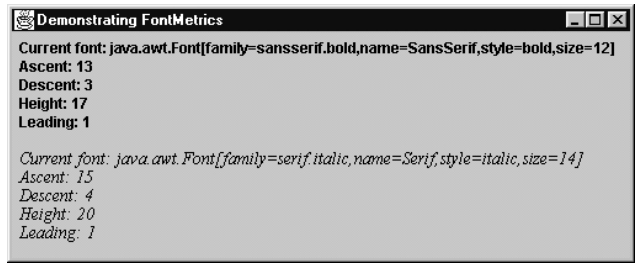

Fig. 11.12 Obtaining font metric information (part 2 of 2).

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`).
`public void fillRect( int x, int y, int width, int height )`	Draws a solid rectangle with the specified `width` and `height`. The top-left corner of the rectangle has the coordinate (`x`, `y`).
`public void clearRect( int x, int y, int width, int height )`	Draws a solid rectangle with the specified `width` and `height` in the current background color. The top-left corner of the rectangle has the coordinate (`x`, `y`).
`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. 11.15).
`public void fillRoundRect( int x, int y, int width, int height, int arcWidth, int arcHeight )`	Draws a solid 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. 11.15).
`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 is lowered when `b` is `false`.
`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 is 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 (`x`, `y`). The oval touches all four sides of the bounding rectangle at the center of each side (see Fig. 11.16).

Fig. 11.13 `Graphics` methods that draw lines, rectangles and ovals (part 1 of 2).

Method	Description
`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. 11.16).

Fig. 11.13 **`Graphics`** methods that draw lines, rectangles and ovals (part 2 of 2).

```
1  // Fig. 11.14: LinesRectsOvals.java
2  // Drawing lines, rectangles and ovals
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LinesRectsOvals extends JFrame {
8     private String s = "Using drawString!";
9
10    public LinesRectsOvals()
11    {
12       super( "Drawing lines, rectangles and ovals" );
13
14       setSize( 400, 165 );
15       show();
16    }
17
18    public void paint( Graphics g )
19    {
20       g.setColor( Color.red );
21       g.drawLine( 5, 30, 350, 30 );
22
23       g.setColor( Color.blue );
24       g.drawRect( 5, 40, 90, 55 );
25       g.fillRect( 100, 40, 90, 55 );
26
27       g.setColor( Color.cyan );
28       g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
29       g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
30
31       g.setColor( Color.yellow );
32       g.draw3DRect( 5, 100, 90, 55, true );
33       g.fill3DRect( 100, 100, 90, 55, false );
34
35       g.setColor( Color.magenta );
36       g.drawOval( 195, 100, 90, 55 );
37       g.fillOval( 290, 100, 90, 55 );
38    }
39
40    public static void main( String args[] )
41    {
42       LinesRectsOvals app = new LinesRectsOvals();
43
44       app.addWindowListener(
45          new WindowAdapter() {
46             public void windowClosing( WindowEvent e )
47             {
48                System.exit( 0 );
49             }
50          }
51       );
52    }
53 }
```

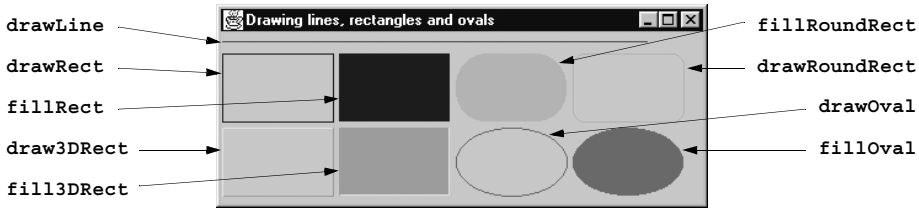

Fig. 11.14 Demonstrating some drawing methods of class **`Graphics`**.

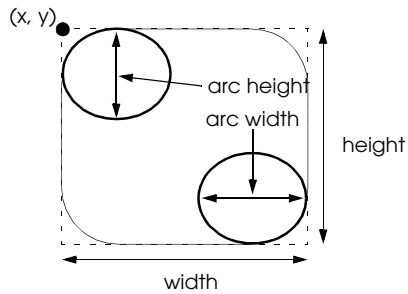

Fig. 11.15 The arc width and arc height for rounded rectangles.

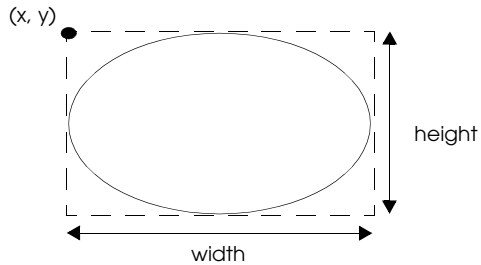

Fig. 11.16 An oval bounded by a rectangle.

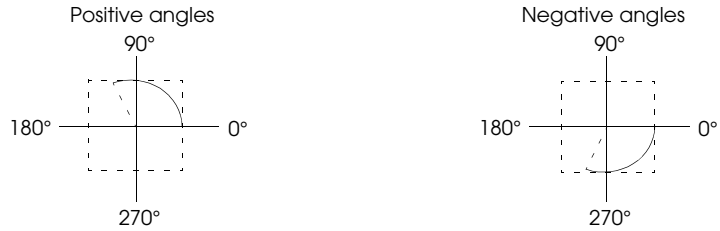

Fig. 11.17 Positive and negative arc angles.

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 coordinates *(**x**, **y**)* 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 solid arc (i.e., a sector) relative to the bounding rectangle's top-left coordinates *(**x**, **y**)* with the specified **`width`** and **`height`**. The arc segment is drawn starting at **`startAngle`** and sweeps **`arcAngle`** degrees.

Fig. 11.18 **`Graphics`** methods for drawing arcs.

```
1  // Fig. 11.19: DrawArcs.java
2  // Drawing arcs
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class DrawArcs extends JFrame {
8     public DrawArcs()
9     {
10       super( "Drawing Arcs" );
11
12       setSize( 300, 170 );
13       show();
14    }
15
16    public void paint( Graphics g )
17    {
18       // start at 0 and sweep 360 degrees
19       g.setColor( Color.yellow );
20       g.drawRect( 15, 35, 80, 80 );
21       g.setColor( Color.black );
22       g.drawArc( 15, 35, 80, 80, 0, 360 );
23
24       // start at 0 and sweep 110 degrees
25       g.setColor( Color.yellow );
26       g.drawRect( 100, 35, 80, 80 );
27       g.setColor( Color.black );
28       g.drawArc( 100, 35, 80, 80, 0, 110 );
29
30       // start at 0 and sweep -270 degrees
31       g.setColor( Color.yellow );
32       g.drawRect( 185, 35, 80, 80 );
33       g.setColor( Color.black );
34       g.drawArc( 185, 35, 80, 80, 0, -270 );
35
36       // start at 0 and sweep 360 degrees
37       g.fillArc( 15, 120, 80, 40, 0, 360 );
38
39       // start at 270 and sweep -90 degrees
40       g.fillArc( 100, 120, 80, 40, 270, -90 );
41
42       // start at 0 and sweep -270 degrees
43       g.fillArc( 185, 120, 80, 40, 0, -270 );
44    }
45
46    public static void main( String args[] )
47    {
48       DrawArcs app = new DrawArcs();
49
50       app.addWindowListener(
51          new WindowAdapter() {
52             public void windowClosing( WindowEvent e )
53             {
54                System.exit( 0 );
55             }
```

```
56              }
57           );
58        }
59     }
```

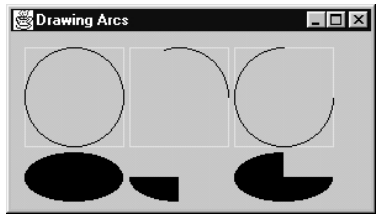

Fig. 11.19 Demonstrating **`drawArc`** and **`fillArc`**.

Method	Description
`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 is specified in the **yPoints** array. The last argument specifies the number of **points**. This method draws a closed polygon—even if the last point is different from the first point.
`public void drawPolyline( int xPoints[], int yPoints[], int points )`	Draws a series of connected lines. The *x*-coordinate of each point is specified in the **xPoints** array and the *y*-coordinate of each point is specified in the **yPoints** array. The last argument specifies the number of **points**. If the last point is different from the first point, the polyline is not closed.
`public void drawPolygon( Polygon p )`	Draws the specified closed polygon.
`public void fillPolygon( int xPoints[], int yPoints[], int points )`	Draws a solid polygon. The *x*-coordinate of each point is specified in the **xPoints** array and the *y*-coordinate of each point is specified in the **yPoints** array. The last argument specifies the number of **points**. This method draws a closed polygon—even if the last point is different from the first point.
`public void fillPolygon( Polygon p )`	Draws the specified solid polygon. The polygon is closed.
`public Polygon()` `// Polygon class`	Constructs a new polygon object. The polygon does not contain any points.
`public Polygon( int xValues[], int yValues[], int numberOfPoints )` `// Polygon class`	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**.

Fig. 11.20 **Graphics** methods for drawing polygons and class **Polygon** constructors.

```
1  // Drawing polygons
2  import java.awt.*;
3  import java.awt.event.*;
4  import javax.swing.*;
5
6  public class DrawPolygons extends JFrame {
7     public DrawPolygons()
8     {
9        super( "Drawing Polygons" );
10
11       setSize( 275, 230 );
12       show();
13    }
14
15    public void paint( Graphics g )
16    {
17       int xValues[] = { 20, 40, 50, 30, 20, 15 };
18       int yValues[] = { 50, 50, 60, 80, 80, 60 };
19       Polygon poly1 = new Polygon( xValues, yValues, 6 );
20
21       g.drawPolygon( poly1 );
22
23       int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };
24       int yValues2[] = { 100, 100, 110, 110, 130, 110, 90 };
25
26       g.drawPolyline( xValues2, yValues2, 7 );
27
28       int xValues3[] = { 120, 140, 150, 190 };
29       int yValues3[] = { 40, 70, 80, 60 };
30
31       g.fillPolygon( xValues3, yValues3, 4 );
32
33       Polygon poly2 = new Polygon();
34       poly2.addPoint( 165, 135 );
35       poly2.addPoint( 175, 150 );
36       poly2.addPoint( 270, 200 );
37       poly2.addPoint( 200, 220 );
38       poly2.addPoint( 130, 180 );
39
40       g.fillPolygon( poly2 );
41    }
42
43    public static void main( String args[] )
44    {
45       DrawPolygons app = new DrawPolygons();
46
47       app.addWindowListener(
48          new WindowAdapter() {
49             public void windowClosing( WindowEvent e )
50             {
51                System.exit( 0 );
52             }
53          }
54       );
55    }
```

```
56    }
```

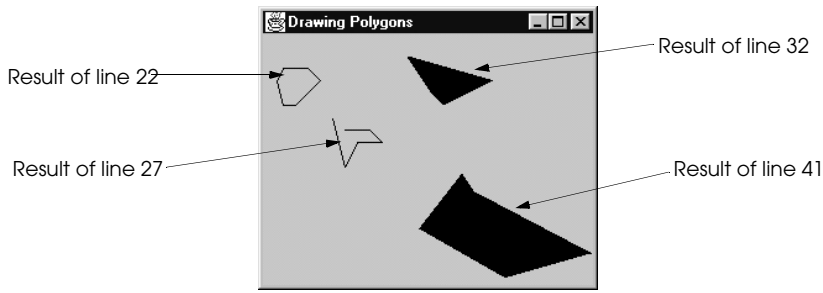

Fig. 11.21 Demonstrating **`drawPolygon`** and **`fillPolygon`**.

```
1  // Fig. 11.22: Shapes.java
2  // Demonstrating some Java2D shapes
3  import javax.swing.*;
4  import java.awt.event.*;
5  import java.awt.*;
6  import java.awt.geom.*;
7  import java.awt.image.*;
8
9  public class Shapes extends JFrame {
10    public Shapes()
11    {
12       super( "Drawing 2D shapes" );
13
14       setSize( 425, 160 );
15       show();
16    }
17
18    public void paint( Graphics g )
19    {
20       // create 2D by casting g to Graphics2D
21       Graphics2D g2d = ( Graphics2D ) g;
22
23       // draw 2D ellipse filled with a blue-yellow gradient
24       g2d.setPaint(
25          new GradientPaint( 5, 30,          // x1, y1
26                             Color.blue,     // initial Color
27                             35, 100,        // x2, y2
28                             Color.yellow,   // end Color
29                             true ) );       // cyclic
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(
36          new Rectangle2D.Double( 80, 30, 65, 100 ) );
37
38       // draw 2D rounded rectangle with a buffered background
39       BufferedImage buffImage =
40          new BufferedImage(
41             10, 10, BufferedImage.TYPE_INT_RGB );
42
43       Graphics2D gg = buffImage.createGraphics();
44       gg.setColor( Color.yellow ); // draw in yellow
45       gg.fillRect( 0, 0, 10, 10 ); // draw a filled rectangle
46       gg.setColor( Color.black );  // draw in black
47       gg.drawRect( 1, 1, 6, 6 );   // draw a rectangle
48       gg.setColor( Color.blue );   // draw in blue
49       gg.fillRect( 1, 1, 3, 3 );   // draw a filled rectangle
50       gg.setColor( Color.red );    // draw in red
51       gg.fillRect( 4, 4, 3, 3 );   // draw a filled rectangle
```

```
52
53          // paint buffImage onto the JFrame
54          g2d.setPaint(
55             new TexturePaint(
56                buffImage, new Rectangle( 10, 10 ) ) );
57          g2d.fill(
58             new RoundRectangle2D.Double(
59                155, 30, 75, 100, 50, 50 ) );
60
61          // draw 2D pie-shaped arc in white
62          g2d.setPaint( Color.white );
63          g2d.setStroke( new BasicStroke( 6.0f ) );
64          g2d.draw(
65             new Arc2D.Double(
66                240, 30, 75, 100, 0, 270, Arc2D.PIE ) );
67
68          // draw 2D lines in green and yellow
69          g2d.setPaint( Color.green );
70          g2d.draw( new Line2D.Double( 395, 30, 320, 150 ) );
71
72          float dashes[] = { 10 };
73
74          g2d.setPaint( Color.yellow );
75          g2d.setStroke(
76             new BasicStroke( 4,
77                              BasicStroke.CAP_ROUND,
78                              BasicStroke.JOIN_ROUND,
79                              10, dashes, 0 ) );
80          g2d.draw( new Line2D.Double( 320, 30, 395, 150 ) );
81       }
82
83       public static void main( String args[] )
84       {
85          Shapes app = new Shapes();
86
87          app.addWindowListener(
88             new WindowAdapter() {
89                public void windowClosing( WindowEvent e )
90                {
91                   System.exit( 0 );
92                }
93             }
94          );
95       }
96    }
```

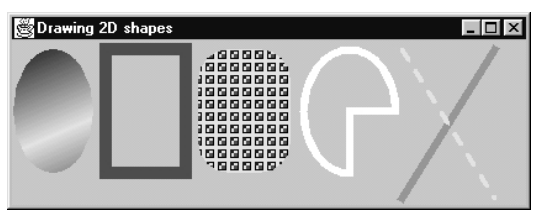

Fig. 11.22 Demonstrating some Java2D shapes.

```
1  // Fig. 11.23: Shapes2.java
2  // Demonstrating a general path
3  import javax.swing.*;
4  import java.awt.event.*;
5  import java.awt.*;
6  import java.awt.geom.*;
7
8  public class Shapes2 extends JFrame {
9     public Shapes2()
10    {
11       super( "Drawing 2D Shapes" );
12
13       setBackground( Color.yellow );
14       setSize( 400, 400 );
15       show();
16    }
17
18    public void paint( Graphics g )
19    {
20       int xPoints[] =
21          { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
22       int yPoints[] =
23          { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };
24
25       Graphics2D g2d = ( Graphics2D ) g;
26
27       // create a star from a series of points
28       GeneralPath star = new GeneralPath();
29
30       // set the initial coordinate of the General Path
31       star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );
32
33       // create the star--this does not draw the star
34       for ( int k = 1; k < xPoints.length; k++ )
35          star.lineTo( xPoints[ k ], yPoints[ k ] );
36
37       // close the shape
38       star.closePath();
39
40       // translate the origin to (200, 200)
41       g2d.translate( 200, 200 );
42
43       // rotate around origin and draw stars in random colors
44       for ( int j = 1; j <= 20; j++ ) {
45          g2d.rotate( Math.PI / 10.0 );
46          g2d.setColor(
47             new Color( ( int ) ( Math.random() * 256 ),
48                        ( int ) ( Math.random() * 256 ),
49                           ( int ) ( Math.random() * 256 ) ) );
50          g2d.fill( star );   // draw a filled star
51       }
52    }
53
```

```
54      public static void main( String args[] )
55      {
56         Shapes2 app = new Shapes2();
57
58         app.addWindowListener(
59            new WindowAdapter() {
60               public void windowClosing( WindowEvent e )
61               {
62                  System.exit( 0 );
63               }
64            }
65         );
66      }
67   }
```

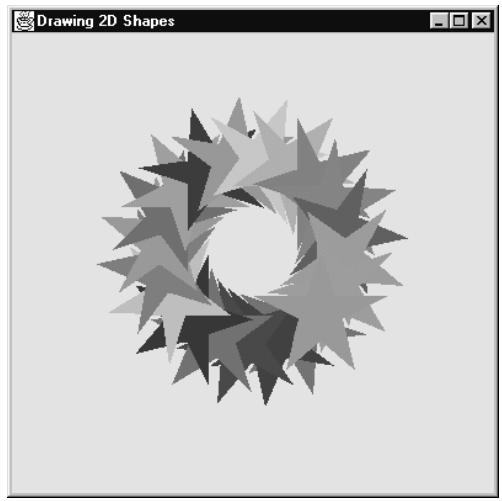

Fig. 11.23 Drawing a **GeneralPath**.