Chapter 11 - Graphics and Java 2D

11.1 Introduction

- Java's graphics capabilities
  - Drawing 2D shapes
  - Controlling colors
  - Controlling fonts
- Java 2D API
  - More sophisticated graphics capabilities
    - Drawing custom 2D shapes
    - Filling shapes with colors and patterns
- Java's coordinate system
  - Scheme for identifying all points on screen
  - Upper-left corner has coordinates (0,0)
  - Coordinate point composed of x-coordinate and y-coordinate

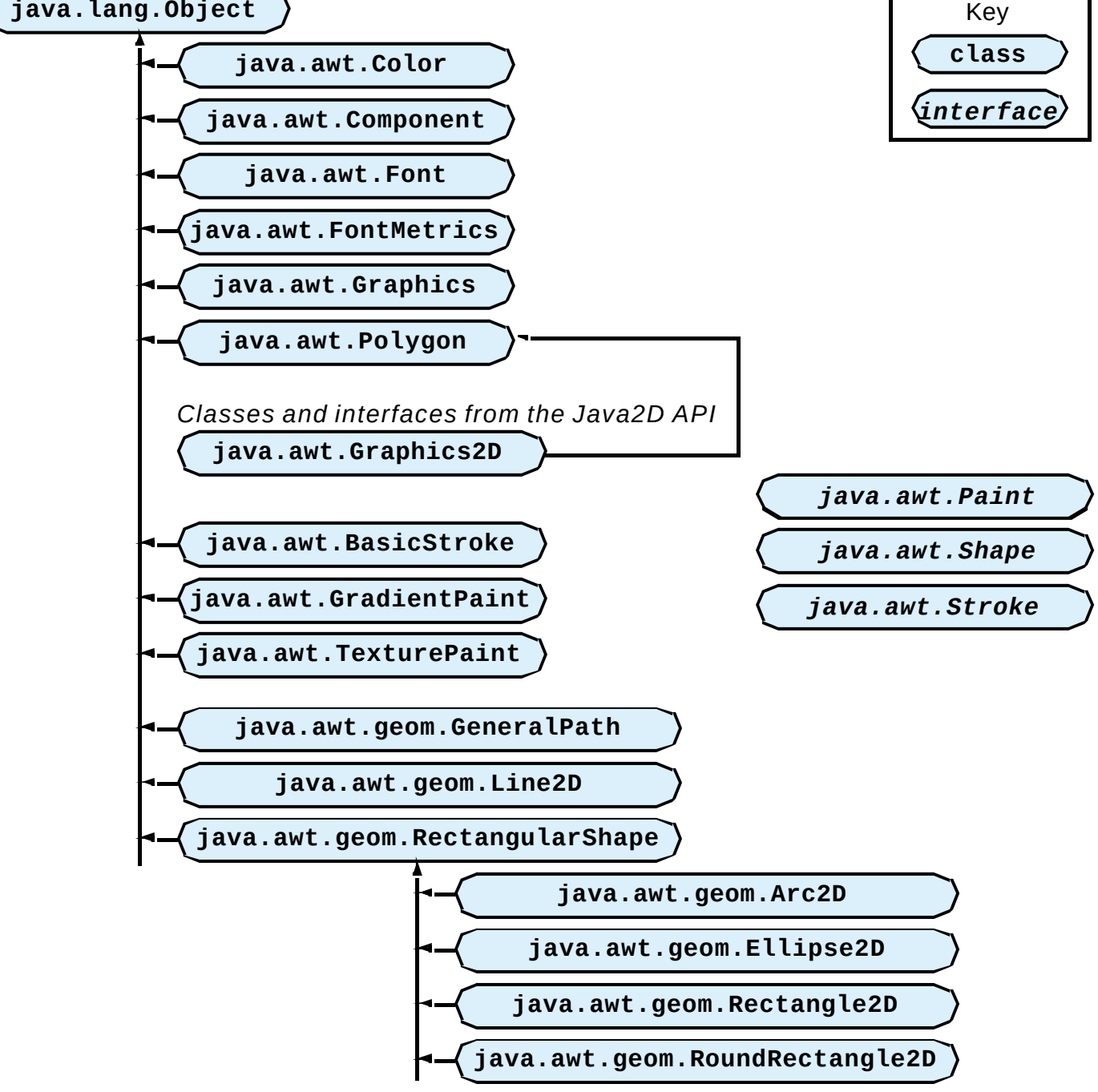
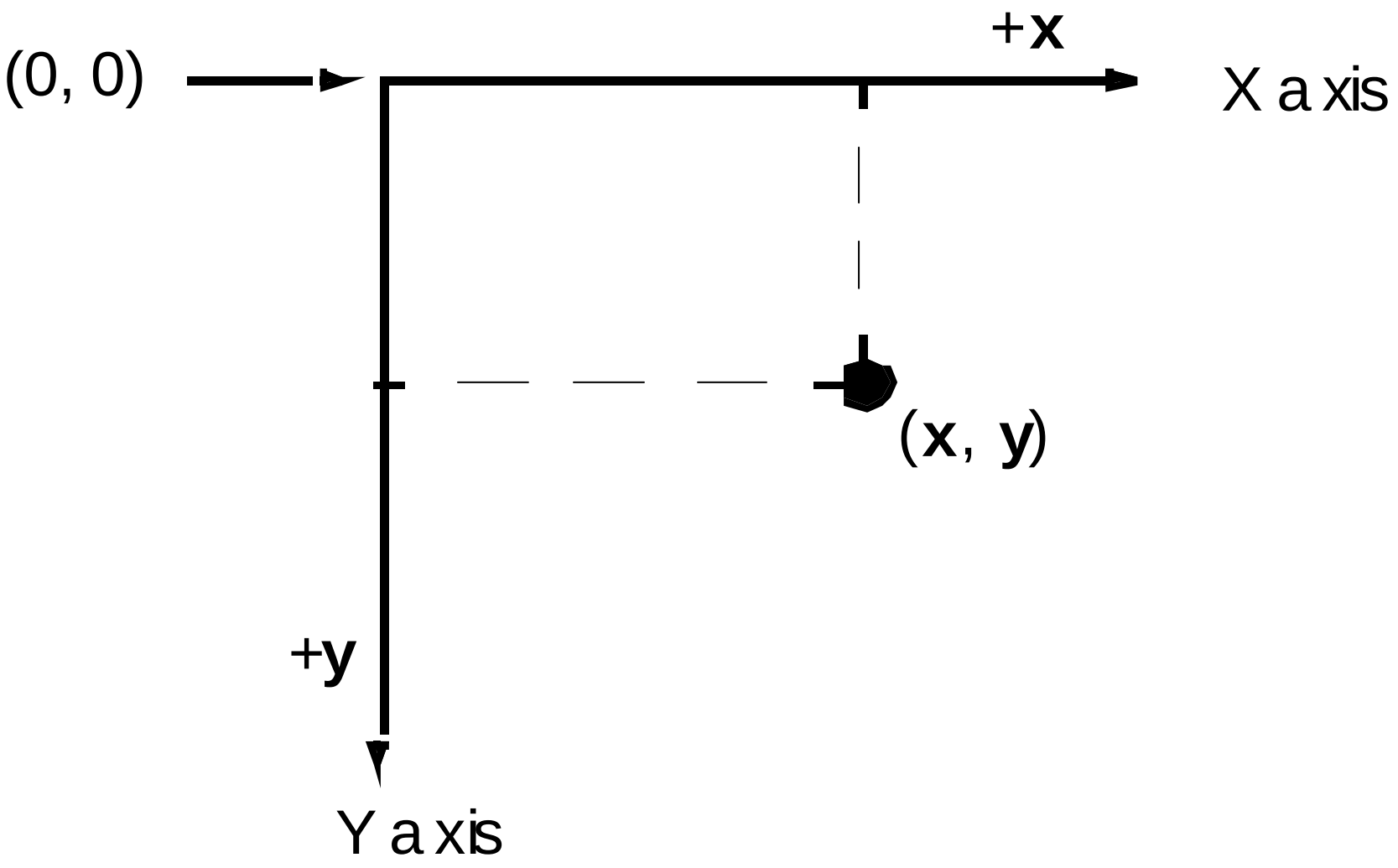


Fig 11.1 Some classes and interfaces used in this chapter from Java's original graphics capabilities and from the Java2D API

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



11.2 Graphics Context and Graphics Objects

- Graphics context
  - Enables drawing on screen
  - **`Graphics`** object manages graphics context
    - Controls how information is drawn in a platform-independent manner
  - Class **`Graphics`** is an ***abstract class***
    - Cannot be instantiated – cannot be used as ***new Graphics();***
    - Contributes to Java's **portability**
  - Class **`Component`** method **`paint`** takes **`Graphics`** object as an argument

```
public void paint( Graphics g )➔ event-driven process
public void repaint()➔ update()➔ paint( Graphics g )
```

  - Called through method **`repaint(cannot be overriden!!!)`**

11.3 Color Control

- Class **`Color`**
  - Defines methods and constants for manipulating colors
  - Colors are created from red, green and blue components
    - RGB values

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

`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

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

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
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class ShowColors extends JFrame {
12
13    // constructor sets window's title bar string and dimensions
14    public ShowColors()
15    {
16       super( "Using colors" );
17
18       setSize( 400, 130 );
19       setVisible( true );
20    }
21
22    // draw rectangles and Strings in different colors
23    public void paint( Graphics g )
24    {
25       // call superclass's paint method
26       super.paint( g );
27
28       // set new drawing color using integers
29       g.setColor( new Color( 255, 0, 0 ) );
30       g.fillRect( 25, 25, 100, 20 );
31       g.drawString( "Current RGB: " + g.getColor(), 130, 40 );
32
33       // set new drawing color using floats
34       g.setColor( new Color( 0.0f, 1.0f, 0.0f ) );
35       g.fillRect( 25, 50, 100, 20 );
```

Outline

ShowColors.java

Line 23

Line 29

Line 30

Line 31

Paint window when application begins execution

Method **setColor** sets color's RGB value

Method **fillRect** creates filled rectangle at specified coordinates using current RGB value

Method **drawString** draw colored text at specified coordinates

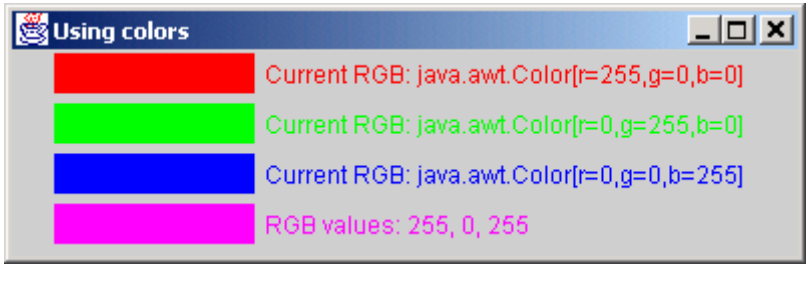
Outline

ShowColors.java

Lines 39 and 44

```
36       g.drawString( "Current RGB: " + g.getColor(), 130, 65 );
37
38       // set new drawing color using static Color objects
39       g.setColor( Color.blue );
40       g.fillRect( 25, 75, 100, 20 );
41       g.drawString( "Current RGB: " + g.getColor(), 130, 90 );
42
43       // display individual RGB values
44       Color color = Color.magenta;
45       g.setColor( color );
46       g.fillRect( 25, 100, 100, 20 );
47       g.drawString( "RGB values: " + color.getRed() + ", " +
48          color.getGreen() + ", " + color.getBlue(), 130, 115 );
49    }
50
51    // execute application
52    public static void main( String args[] )
53    {
54       ShowColors application = new ShowColors();
55
56       application.setDefaultCloseOperation(
57          JFrame.EXIT_ON_CLOSE );
58    }
59
60 }  // end class ShowColors
```

Use constant in class **Color** to specify current color



```
1  // Fig. 11.6: ShowColors2.java
2  // Demonstrating JColorChooser.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class ShowColors2 extends JFrame {
12    private JButton changeColorButton;
13    private Color color = Color.lightGray;
14    private Container container;
15
16    // set up GUI
17    public ShowColors2()
18    {
19       super( "Using JColorChooser" );
20
21       container = getContentPane();
22       container.setLayout( new FlowLayout() );
23
24       // set up changeColorButton and register its event handler
25       changeColorButton = new JButton( "Change Color" );
26
27       changeColorButton.addActionListener(
28
29          // anonymous inner class
30          new ActionListener() {
31
32             // display JColorChooser when user clicks button
33             public void actionPerformed( ActionEvent event )
34             {
35                color = JColorChooser.showDialog(
```

Outline

ShowColors2.jav

Line 35

Line 35

JColorChooser allows user to choose from among several colors

static method **showDialog** displays the color chooser dialog

Outline

ShowColors2.jav

```
36                   ShowColors2.this, "Choose a color", color );
37
38                // set default color, if no color is returned
39                if ( color == null )
40                   color = Color.lightGray;
41
42                // change content pane's background color
43                container.setBackground( color );
44             }
45
46          }  // end anonymous inner class
47
48       ); // end call to addActionListener
49
50       container.add( changeColorButton );
51
52       setSize( 400, 130 );
53       setVisible( true );
54    }
55
56    // execute application
57    public static void main( String args[] )
58    {
59       ShowColors2 application = new ShowColors2();
60
61       application.setDefaultCloseOperation(
62          JFrame.EXIT_ON_CLOSE );
63    }
64
65 }  // end class ShowColors2
```

ShowColors2.jav

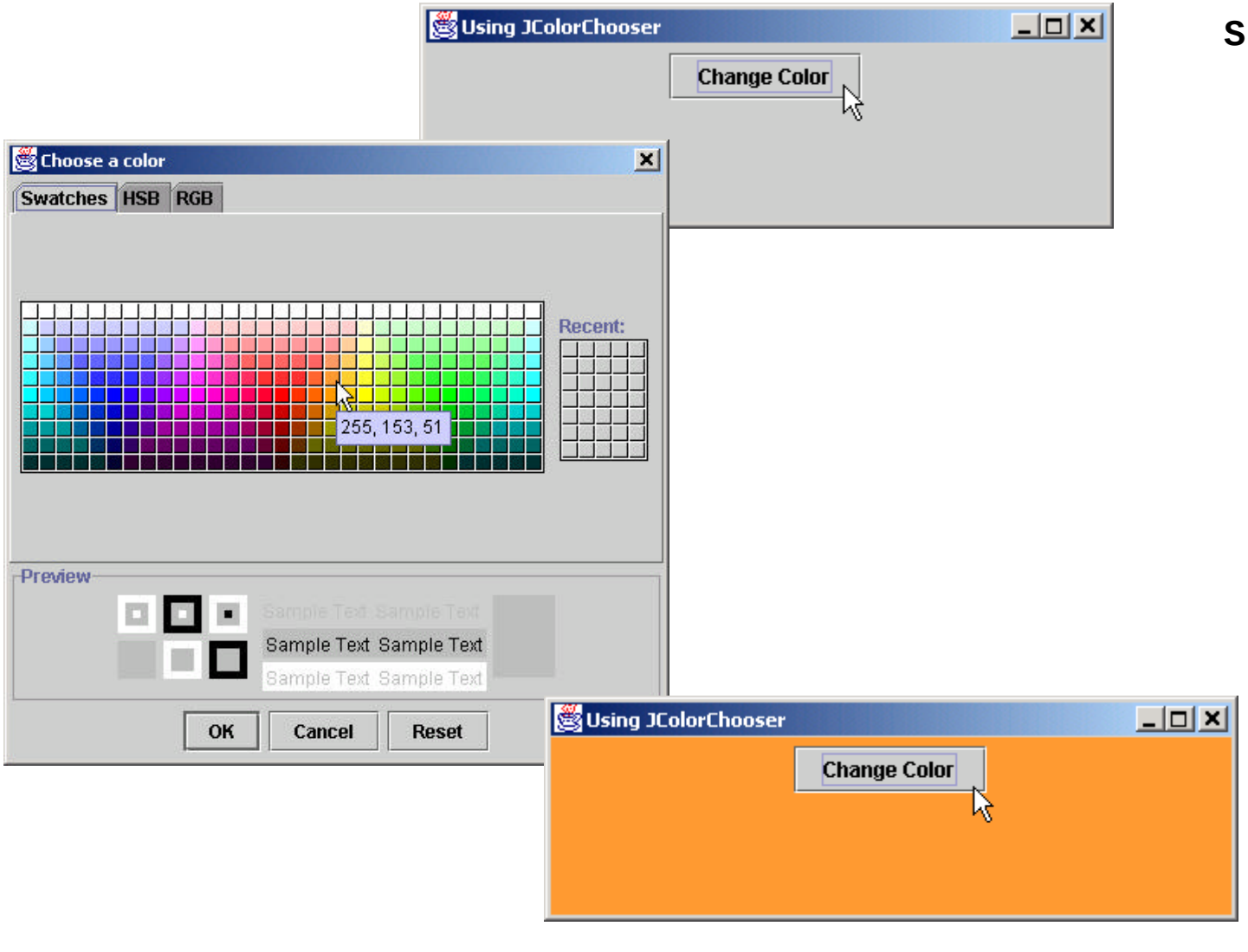
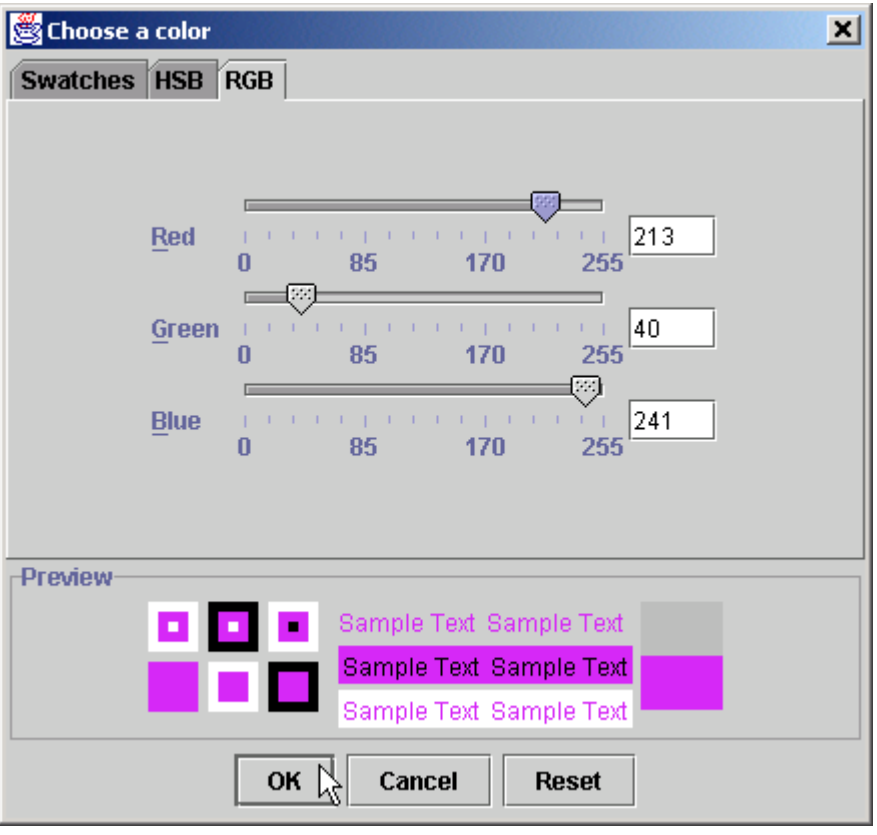
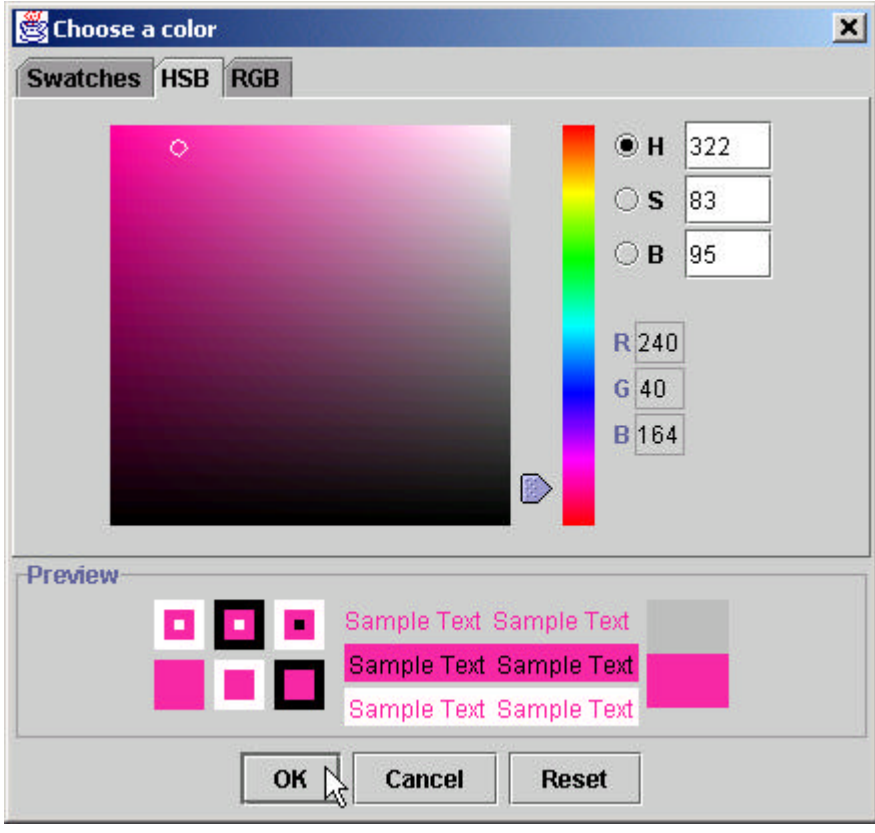


Fig. 11.7 The HSB and RGB of the `JColorChooser` dialog





11.4 Font Control

- Class **`Font`**
  - Contains methods and constants for font control
  - Font constructor takes three arguments
    - Font name→**standard Java font name**
      - **`Monospaced`**, **`SansSerif`**, **`Serif`**, etc.
    - Font style
      - **`Font.PLAIN`**, **`Font.ITALIC`** and **`Font.BOLD`**
    - Font size
      - Measured in points (**1/72** of inch per point)
- Font metrics
  - Height
  - Descent (amount character dips below baseline)
  - Ascent (amount character rises above baseline)
  - Leading (difference between descent and ascent)

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

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 bold 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.

Fig. 11.8 **Font** methods, constants and font-related **Graphics** methods (Part 1 of 2).

Fig. 11.8 `Font` methods, constants and font-related `Graphics` methods (Part 2).

Method or constant	Description
`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 (Part 2 of 2).

```
1  // Fig. 11.9: Fonts.java
2  // Using fonts
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class Fonts extends JFrame {
12
13    // set window's title bar and dimensions
14    public Fonts()
15    {
16       super( "Using fonts" );
17
18       setSize( 420, 125 );
19       setVisible( true );
20    }
21
22    // display Strings in different fonts and colors
23    public void paint( Graphics g )
24    {
25       // call superclass's paint method
26       super.paint( g );
27
28       // set current font to Serif (Times), bold, 12pt
29       // and draw a string
30       g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
31       g.drawString( "Serif 12 point bold.", 20, 50 );
32
33       // set current font to Monospaced (Courier),
34       // italic, 24pt and draw a string
35       g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
```

Outline

Fonts.java

Line 30

Line 31

Method **setFont** sets current font

Draw text using current font

Outline

Fonts.java

Line 41

```
36       g.drawString( "Monospaced 24 point italic.", 20, 70 );
37
38       // set current font to SansSerif (Helvetica),
39       // plain, 14pt and draw a string
40       g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
41       g.drawString( "SansSerif 14 point plain.", 20, 90 );
42
43       // set current font to Serif (times), bold/italic,
44       // 18pt and draw a string
45       g.setColor( Color.red );
46       g.setFont(
47          new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
48       g.drawString( g.getFont().getName() + " " +
49          g.getFont().getSize() +
50          " point bold italic.", 20, 110 );
51    }
52
53    // execute application
54    public static void main( String args[] )
55    {
56       Fonts application = new Fonts();
57
58       application.setDefaultCloseOperation(
59          JFrame.EXIT_ON_CLOSE );
60    }
61
62 }  // end class Fonts
```

Set font to SansSerif 14-point plain

Set font to Serif 18-point bold italic

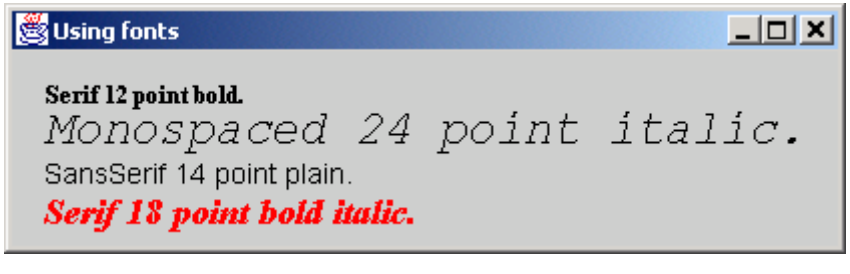


Fig. 11.10 Font metrics

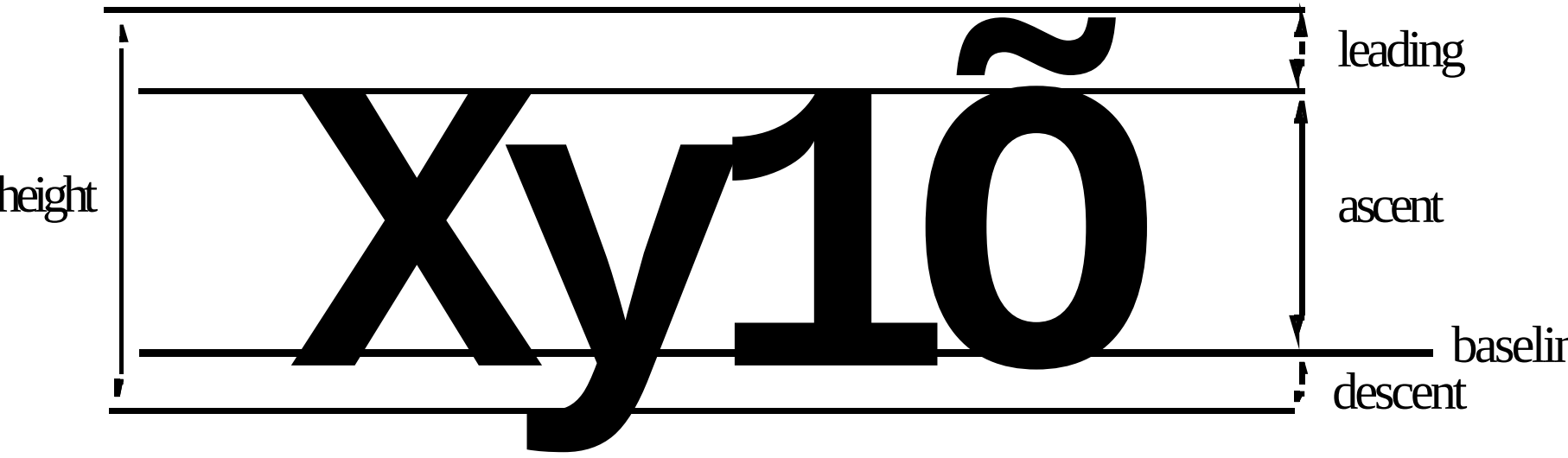


Fig. 11.11 `FontMetrics` and `Graphics` methods for obtaining font metrics.

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
5  // Java core packages
6  import java.awt.*;
7  import java.awt.event.*;
8
9  // Java extension packages
10 import javax.swing.*;
11
12 public class Metrics extends JFrame {
13
14    // set window's title bar String and dimensions
15    public Metrics()
16    {
17       super( "Demonstrating FontMetrics" );
18
19       setSize( 510, 210 );
20       setVisible( true );
21    }
22
23    // display font metrics
24    public void paint( Graphics g )
25    {
26       // call superclass's paint method
27       super.paint( g );
28
29       g.setFont( new Font( "SansSerif", Font.BOLD, 12 ) );
30       FontMetrics metrics = g.getFontMetrics();
31       g.drawString( "Current font: " + g.getFont(), 10, 40 );
32       g.drawString( "Ascent: " + metrics.getAscent(), 10, 55 );
33       g.drawString( "Descent: " + metrics.getDescent(), 10, 70 );
34       g.drawString( "Height: " + metrics.getHeight(), 10, 85 );
35       g.drawString( "Leading: " + metrics.getLeading(), 10, 100);
```

Outline

Metrics.java

Line 29

Line 30

Lines 32-35

Set font to SansSerif 12-point bold

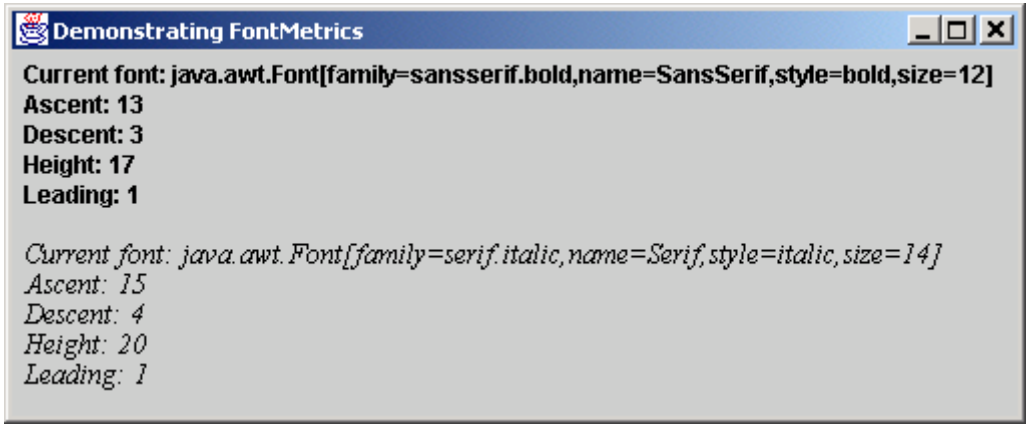
Obtain **FontMetrics** object for current font

Use **FontMetrics** to obtain ascent, descent, height and leading

```
36
37          Font font = new Font( "Serif", Font.ITALIC, 14 );
38          metrics = g.getFontMetrics( font );
39          g.setFont( font );
40          g.drawString( "Current font: " + font, 10, 130 );
41          g.drawString( "Ascent: " + metrics.getAscent(), 10, 145 );
42          g.drawString( "Descent: " + metrics.getDescent(), 10, 160);
43          g.drawString( "Height: " + metrics.getHeight(), 10, 175 );
44          g.drawString( "Leading: " + metrics.getLeading(), 10, 190);
45       }
46
47       // execute application
48       public static void main( String args[] )
49       {
50          Metrics application = new Metrics();
51
52          application.setDefaultCloseOperation(
53             JFrame.EXIT_ON_CLOSE );
54       }
55
56    }  // end class Metrics
```

Repeat same process for Serif 14-point italic font

Lines 37-44



11.5 Drawing Lines, Rectangles and Ovals

- Class **`Graphics`**
  - Provides methods for drawing lines, rectangles and ovals
    - All drawing methods require parameters **`width`** and **`height`**

Fig. 11.13 `Graphics` methods that draw lines, rectangle and 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**).
`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).

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

Fig. 11.13 `Graphics` methods that draw lines, rectangle and ovals (Part 2).

Method	Description
`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).
`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
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class LinesRectsOvals extends JFrame {
12
13    // set window's title bar String and dimensions
14    public LinesRectsOvals()
15    {
16       super( "Drawing lines, rectangles and ovals" );
17
18       setSize( 400, 165 );
19       setVisible( true );
20    }
21
22    // display various lines, rectangles and ovals
23    public void paint( Graphics g )
24    {
25       // call superclass's paint method
26       super.paint( g );
27
28       g.setColor( Color.red );
29       g.drawLine( 5, 30, 350, 30 );
30
31       g.setColor( Color.blue );
32       g.drawRect( 5, 40, 90, 55 );
33       g.fillRect( 100, 40, 90, 55 );
34
35       g.setColor( Color.cyan );
```


LinesRectsOvals.java

```
36        g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
37        g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
38
39        g.setColor( Color.yellow );
40        g.draw3DRect( 5, 100, 90, 55, true );
41        g.fill3DRect( 100, 100, 90, 55, false );
42
43        g.setColor( Color.magenta );
44        g.drawOval( 195, 100, 90, 55 );
45        g.fillOval( 290, 100, 90, 55 );
46     }
47
48     // execute application
49     public static void main( String args[] )
50     {
51        LinesRectsOvals application = new LinesRectsOvals();
52
53        application.setDefaultCloseOperation(
54           JFrame.EXIT_ON_CLOSE );
55     }
56
57  }  // end class LinesRectsOvals
```

Draw filled rounded rectangle

Draw (non-filled) rounded rectangle

Draw 3D rectangle

Draw filled 3D rectangle

Draw oval

Draw filled oval

Line 36

Line 37

Line 40

Line 41

Line 44

Line 45

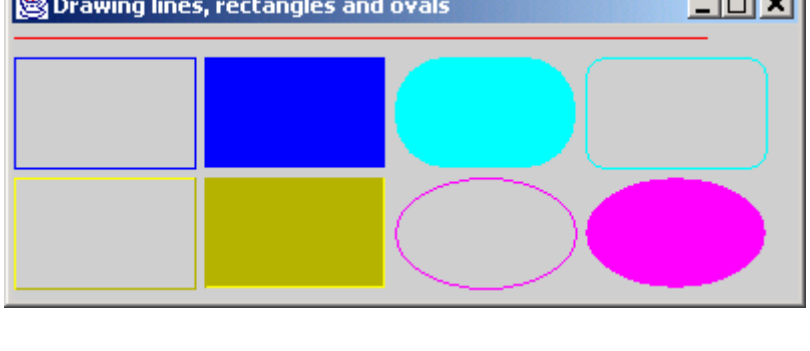


Fig. 11.15 The arc width and height for rounded rectangles.

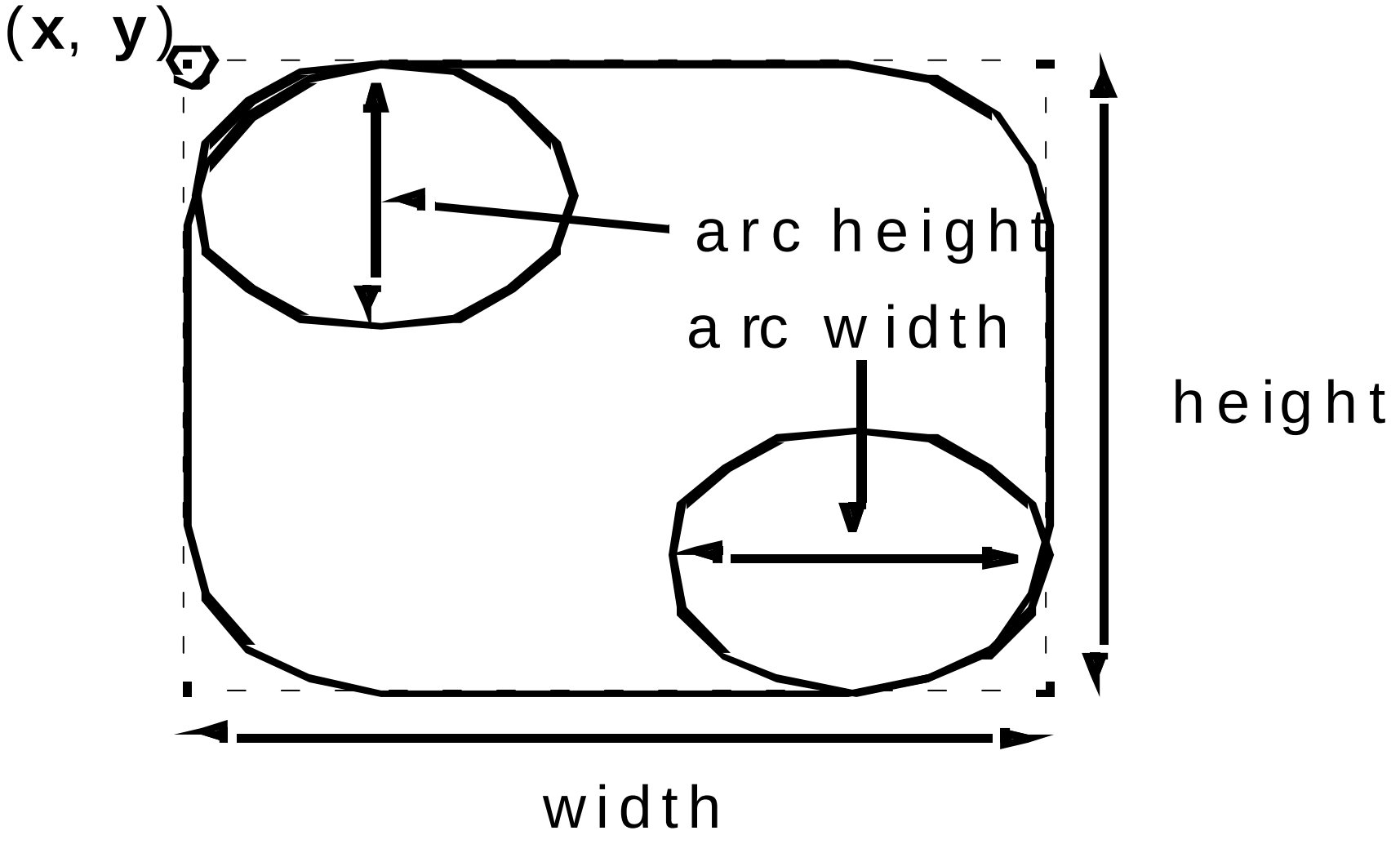
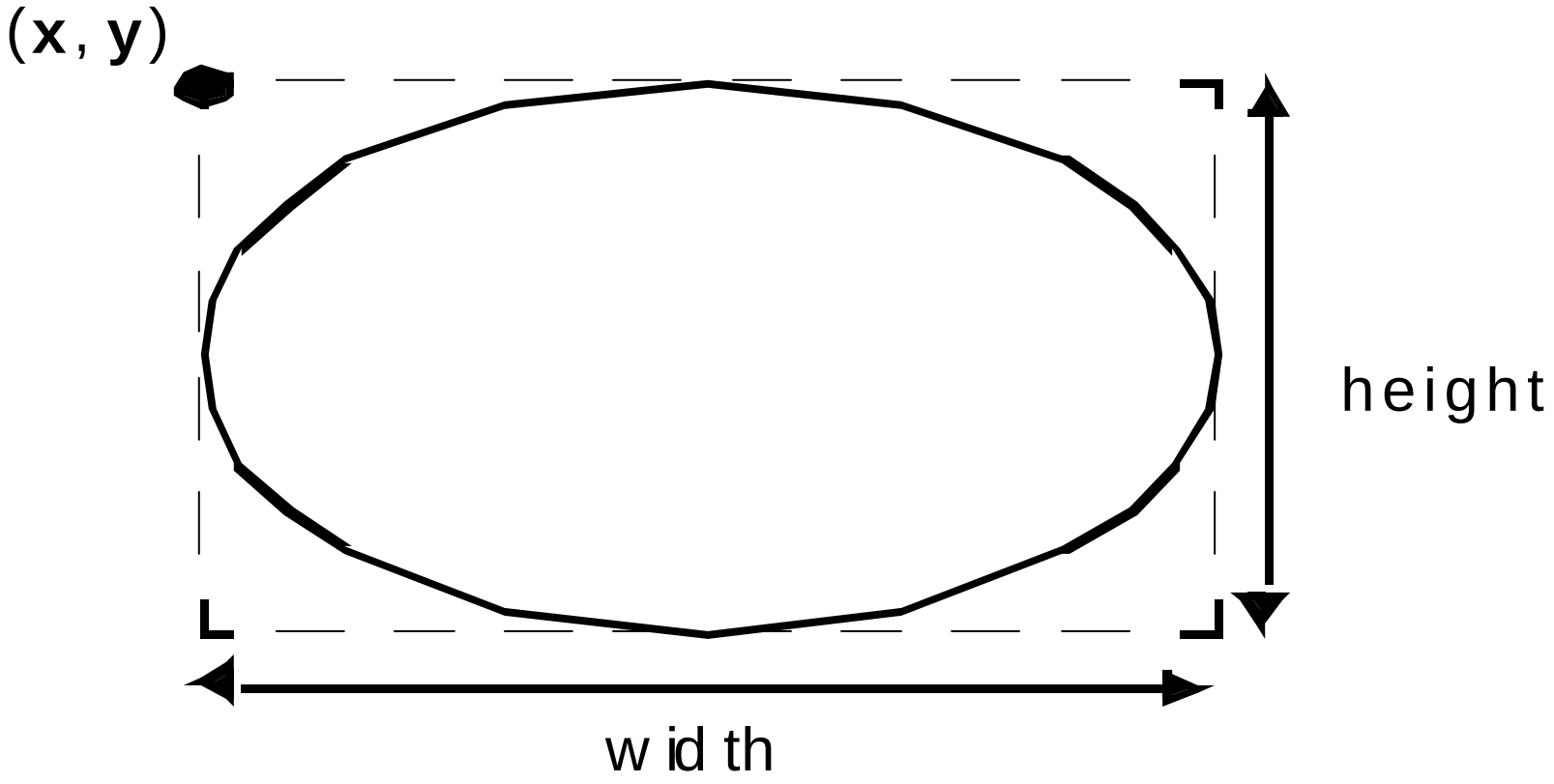


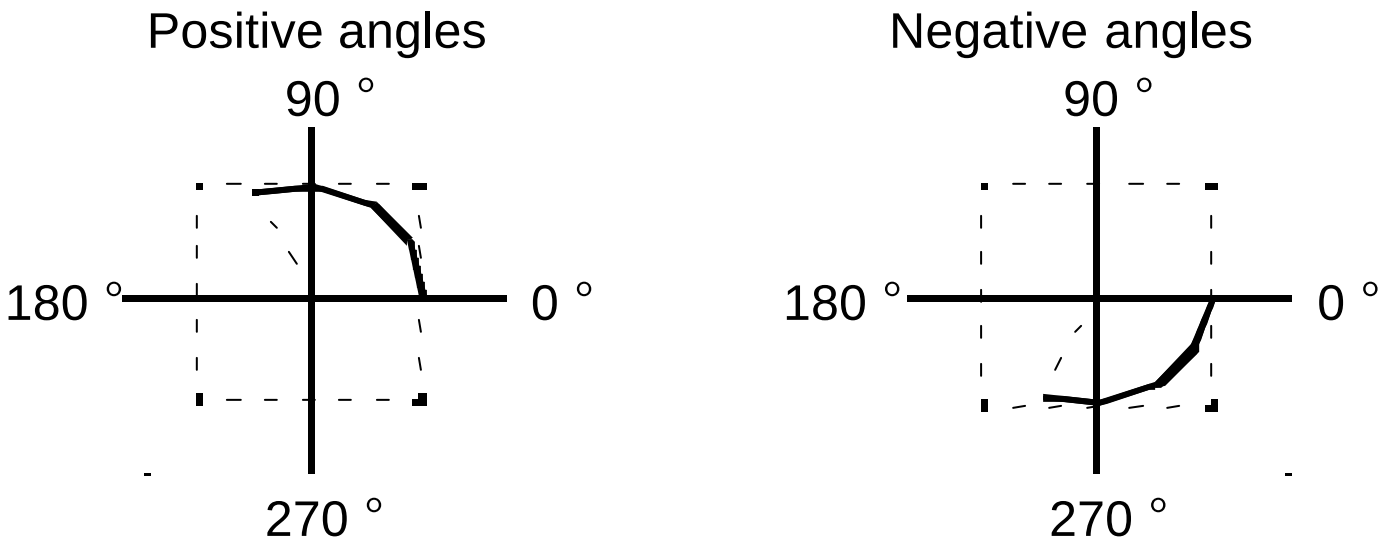
Fig. 11.16 An oval bounded by a rectangle.



11.6 Drawing Arcs

- Arc
  - Portion of oval
  - Measured in degrees
  - *Sweeps* the number of degrees in ***arc angle***
  - Sweep starts at starting angle
    - **Counterclockwise** sweep is measure in positive degrees
    - **Clockwise** sweep is measure in negative degrees

Fig. 11.17 Positive and negative arc angles.



11.18 Graphics methods for drawing arcs.

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.

Outline

DrawArcs.java

Lines 29-32

```
1  // Fig. 11.19: DrawArcs.java
2  // Drawing arcs
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class DrawArcs extends JFrame {
12
13    // set window's title bar String and dimensions
14    public DrawArcs()
15    {
16       super( "Drawing Arcs" );
17
18       setSize( 300, 170 );
19       setVisible( true );
20    }
21
22    // draw rectangles and arcs
23    public void paint( Graphics g )
24    {
25       // call superclass's paint method
26       super.paint( g );
27
28       // start at 0 and sweep 360 degrees
29       g.setColor( Color.yellow );
30       g.drawRect( 15, 35, 80, 80 );
31       g.setColor( Color.black );
32       g.drawArc( 15, 35, 80, 80, 0, 360 );
33
34       // start at 0 and sweep 110 degrees
35       g.setColor( Color.yellow );
```

Draw first arc that sweeps 360 degrees and is contained in rectangle

```
36          g.drawRect( 100, 35, 80, 80 );
37          g.setColor( Color.black );
38          g.drawArc( 100, 35, 80, 80, 0, 110 );
39
40          // start at 0 and sweep -270 degrees
41          g.setColor( Color.yellow );
42          g.drawRect( 185, 35, 80, 80 );
43          g.setColor( Color.black );
44          g.drawArc( 185, 35, 80, 80, 0, -270 );
45
46          // start at 0 and sweep 360 degrees
47          g.fillArc( 15, 120, 80, 40, 0, 360 );
48
49          // start at 270 and sweep -90 degrees
50          g.fillArc( 100, 120, 80, 40, 270, -90 );
51
52          // start at 0 and sweep -270 degrees
53          g.fillArc( 185, 120, 80, 40, 0, -270 );
54       }
55
56       // execute application
57       public static void main( String args[] )
58       {
59          DrawArcs application = new DrawArcs();
60
61          application.setDefaultCloseOperation(
62             JFrame.EXIT_ON_CLOSE );
63       }
64
65    }  // end class DrawArcs
```

Draw second arc that sweeps 110 degrees and is contained in rectangle

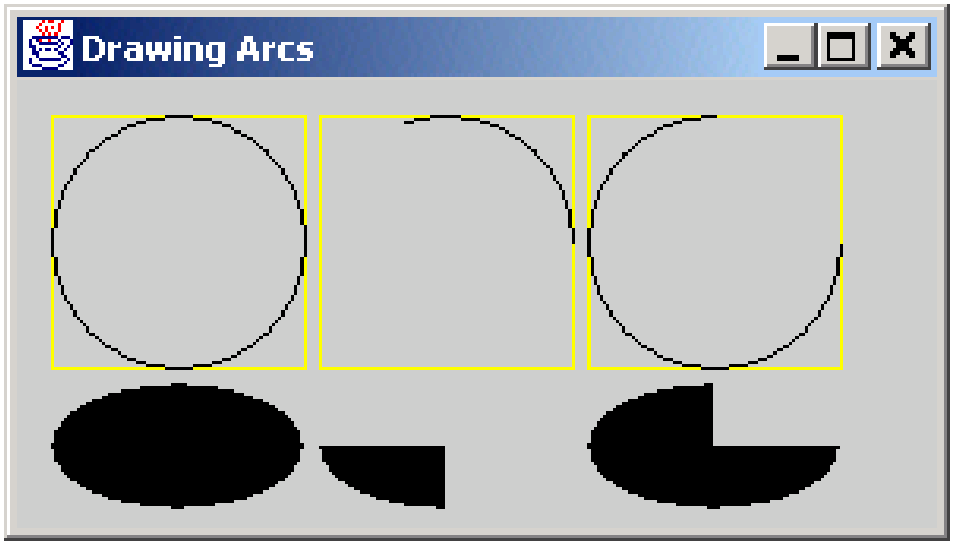
Draw third arc that sweeps -270 degrees and is contained in rectangle

Draw fourth arc that is filled, has starting angle 0 and sweeps 360 degrees

Draw fifth arc that is filled, has starting angle 270 and sweeps -90 degrees

Draw sixth arc that is filled, has starting angle 0 and sweeps -270 degrees

`DrawArcs.java`.



Output

11.7 Drawing Polygons and Polylines

- Class **`Polygon`**
  - Polygons
    - Multisided shapes
  - Polylines
    - Series of connected points

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

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  // Fig. 11.21: DrawPolygons.java
2  // Drawing polygons
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class DrawPolygons extends JFrame {
12
13    // set window's title bar String and dimensions
14    public DrawPolygons()
15    {
16       super( "Drawing Polygons" );
17
18       setSize( 275, 230 );
19       setVisible( true );
20    }
21
22    // draw polygons and polylines
23    public void paint( Graphics g )
24    {
25       // call superclass's paint method
26       super.paint( g );
27
28       int xValues[] = { 20, 40, 50, 30, 20, 15 };
29       int yValues[] = { 50, 50, 60, 80, 80, 60 };
30       Polygon polygon1 = new Polygon( xValues, yValues, 6 );
31
32       g.drawPolygon( polygon1 );
33
34       int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };
35       int yValues2[] = { 100, 100, 110, 110, 130, 110, 90 };
```

Outline

DrawPolygons.java

Lines 28-29

Line 32

Lines 34-35

int arrays specifying **Polygon polygon1** points

Draw **polygon1** to screen

int arrays specifying **Polygon polygon2** points

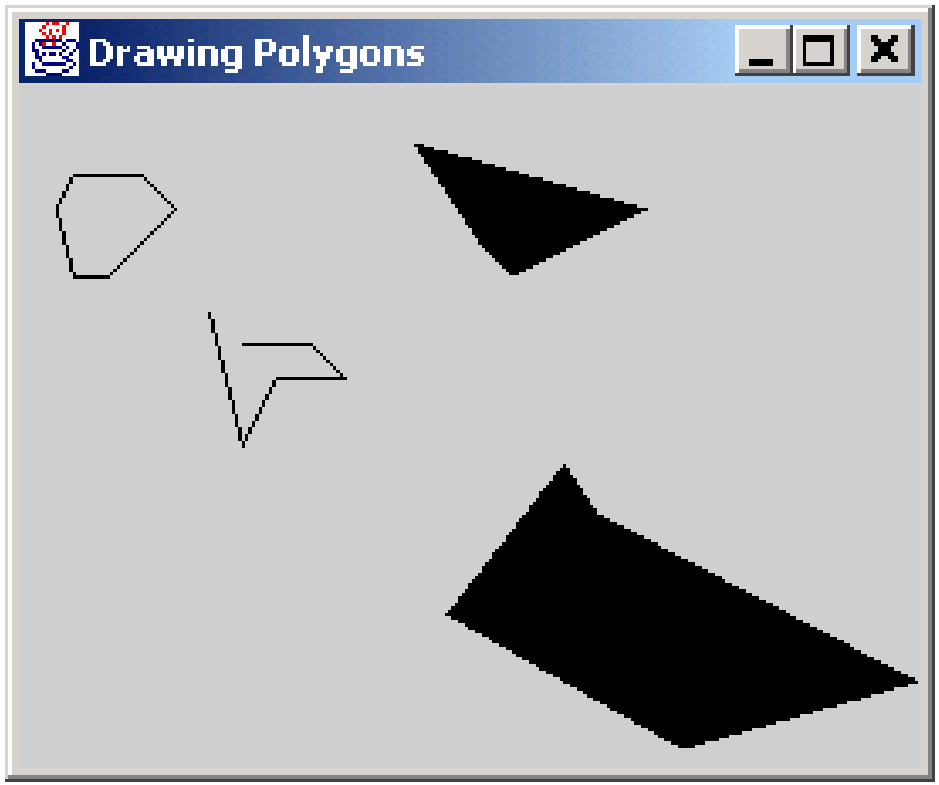
```
36
37          g.drawPolyline( xValues2, yValues2, 7 );
38
39          int xValues3[] = { 120, 140, 150, 190 };
40          int yValues3[] = { 40, 70, 80, 60 };
41
42          g.fillPolygon( xValues3, yValues3, 4 );
43
44          Polygon polygon2 = new Polygon();
45          polygon2.addPoint( 165, 135 );
46          polygon2.addPoint( 175, 150 );
47          polygon2.addPoint( 270, 200 );
48          polygon2.addPoint( 200, 220 );
49          polygon2.addPoint( 130, 180 );
50
51          g.fillPolygon( polygon2 );
52       }
53
54       // execute application
55       public static void main( String args[] )
56       {
57          DrawPolygons application = new DrawPolygons();
58
59          application.setDefaultCloseOperation(
60             JFrame.EXIT_ON_CLOSE );
61       }
62
63    }  // end class DrawPolygons
```

Draw **`polygon2`** to screen

Specify **`Polygon`** points and draw (filled) **`polygon3`** to screen

Method **`addPoint`** adds pairs of x-y coordinates to **`Polygon`**

DrawPolygons.java



Output

11.8 Java 2D API & 11.9 Java 2D Shapes

- Java 2D API
  - Provides advanced 2D graphics capabilities
    - **`java.awt`**
    - **`java.awt.image`**
    - **`java.awt.color`**
    - **`java.awt.font`**
    - **`java.awt.geom`**
    - **`java.awt.print`**
    - **`java.awt.image.renderable`**
  - Uses class **`java.awt.Graphics2D`**
    - Extends class **`java.awt.Graphics`**
- Java 2D shapes
  - Package **`java.awt.geom`**
    - **Ellipse2D.Double ➔ GradientPaint(change color gradually)**
    - **Rectangle2D.Double ➔ BufferedImage (create a texture image)**
    - **RoundRectangle2D.Double ➔ TexturePaint (create a image as texture)**
    - **Arc3D.Double ➔ BasicStroke (specify the width of the line)**
    - **Lines2D.Double**

```
1  // Fig. 11.22: Shapes.java
2  // Demonstrating some Java2D shapes
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7  import java.awt.geom.*;
8  import java.awt.image.*;
9
10 // Java extension packages
11 import javax.swing.*;
12
13 public class Shapes extends JFrame {
14
15    // set window's title bar String and dimensions
16    public Shapes()
17    {
18       super( "Drawing 2D shapes" );
19
20       setSize( 425, 160 );
21       setVisible( true );
22    }
23
24    // draw shapes with Java2D API
25    public void paint( Graphics g )
26    {
27       // call superclass's paint method
28       super.paint( g );
29
30       // create 2D by casting g to Graphics2D
31       Graphics2D g2d = ( Graphics2D ) g;
32
33       // draw 2D ellipse filled with a blue-yellow gradient
34       g2d.setPaint( new GradientPaint( 5, 30, Color.blue, 35,
35          100, Color.yellow, true ) );
```

Outline

Shapes.java

Lines 34-35

Use **GradientPaint** to fill shape with gradient

```
36       g2d.fill( new Ellipse2D.Double( 5, 30, 65, 100 ) );
37
38       // draw 2D rectangle in red
39       g2d.setPaint( Color.red );
40       g2d.setStroke( new BasicStroke( 10.0f ) );
41       g2d.draw( new Rectangle2D.Double( 80, 30, 65, 100 ) );
42
43       // draw 2D rounded rectangle with a buffered background
44       BufferedImage buffImage = new BufferedImage(
45          10, 10, BufferedImage.TYPE_INT_RGB );
46
47       Graphics2D gg = buffImage.createGraphics();
48       gg.setColor( Color.yellow ); // draw in yellow
49       gg.fillRect( 0, 0, 10, 10 ); // draw a filled rectangle
50       gg.setColor( Color.black );  // draw in black
51       gg.drawRect( 1, 1, 6, 6 );   // draw a rectangle
52       gg.setColor( Color.blue );   // draw in blue
53       gg.fillRect( 1, 1, 3, 3 );   // draw a filled rectangle
54       gg.setColor( Color.red );    // draw in red
55       gg.fillRect( 4, 4, 3, 3 );   // draw a filled rectangle
56
57       // paint buffImage onto the JFrame
58       g2d.setPaint( new TexturePaint(
59          buffImage, new Rectangle( 10, 10 ) ) );
60       g2d.fill( new RoundRectangle2D.Double(
61          155, 30, 75, 100, 50, 50 ) );
62
63       // draw 2D pie-shaped arc in white
64       g2d.setPaint( Color.white );
65       g2d.setStroke( new BasicStroke( 6.0f ) );
66       g2d.draw( new Arc2D.Double(
67          240, 30, 75, 100, 0, 270, Arc2D.PIE ) );
68
69       // draw 2D lines in green and yellow
70       g2d.setPaint( Color.green );
```

Fill ellipse with gradient

Use **BasicStroke** to draw 2D red-border rectangle

Line 36

BufferedImage produces image to be manipulated

Lines 44-45

Draw texture into **BufferedImage**

Lines 58-61

Lines 64-67

Use **BufferedImage** as texture for painting rounded rectangle

Use **Arc2D.PIE** to draw white-border 2D pie-shaped arc

```
71          g2d.draw( new Line2D.Double( 395, 30, 320, 150 ) );
72
73          float dashes[] = { 10 };
74
75          g2d.setPaint( Color.yellow );
76          g2d.setStroke( new BasicStroke( 4, BasicStroke.CAP_ROUND,
77             BasicStroke.JOIN_ROUND, 10, dashes, 0 ) );
78          g2d.draw( new Line2D.Double( 320, 30, 395, 150 ) );
79       }
80
81       // execute application
82       public static void main( String args[] )
83       {
84          Shapes application = new Shapes();
85
86          application.setDefaultCloseOperation(
87             JFrame.EXIT_ON_CLOSE );
88       }
89
90    }  // end class Shapes
```

Draw solid green line

Shapes.java

Draw dashed yellow line that crosses solid green line

Lines 75-78

```
1  // Fig. 11.23: Shapes2.java
2  // Demonstrating a general path
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7  import java.awt.geom.*;
8
9  // Java extension packages
10 import javax.swing.*;
11
12 public class Shapes2 extends JFrame {
13
14    // set window's title bar String, background color
15    // and dimensions
16    public Shapes2()
17    {
18       super( "Drawing 2D Shapes" );
19
20       getContentPane().setBackground( Color.yellow );
21       setSize( 400, 400 );
22       setVisible( true );
23    }
24
25    // draw general paths
26    public void paint( Graphics g )
27    {
28       // call superclass's paint method
29       super.paint( g );
30
31       int xPoints[] =
32          { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
33       int yPoints[] =
34          { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };
35
```

x-y coordinates that comprise star

Outline

Shapes2.java

Lines 31-34

```
36       Graphics2D g2d = ( Graphics2D ) g;
37
38       // create a star from a series of points
39       GeneralPath star = new GeneralPath();
40
41       // set the initial coordinate of the General Path
42       star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );
43
44       // create the star--this does not draw the star
45       for ( int count = 1; count < xPoints.length; count++ )
46          star.lineTo( xPoints[ count ], yPoints[ count ] );
47
48       // close the shape
49       star.closePath();
50
51       // translate the origin to (200, 200)
52       g2d.translate( 200, 200 );
53
54       // rotate around origin and draw stars in random colors
55       for ( int count = 1; count <= 20; count++ ) {
56
57          // rotate coordinate system
58          g2d.rotate( Math.PI / 10.0 );
59
60          // set random drawing color
61          g2d.setColor( new Color(
62             ( int ) ( Math.random() * 256 ),
63             ( int ) ( Math.random() * 256 ),
64             ( int ) ( Math.random() * 256 ) ) );
65
66          // draw filled star
67          g2d.fill( star );
68       }
69
70    }  // end method paint
```

`GeneralPath` is a shape constructed from straight lines and complex curves

utline

`Shapes2.java`

Line 39

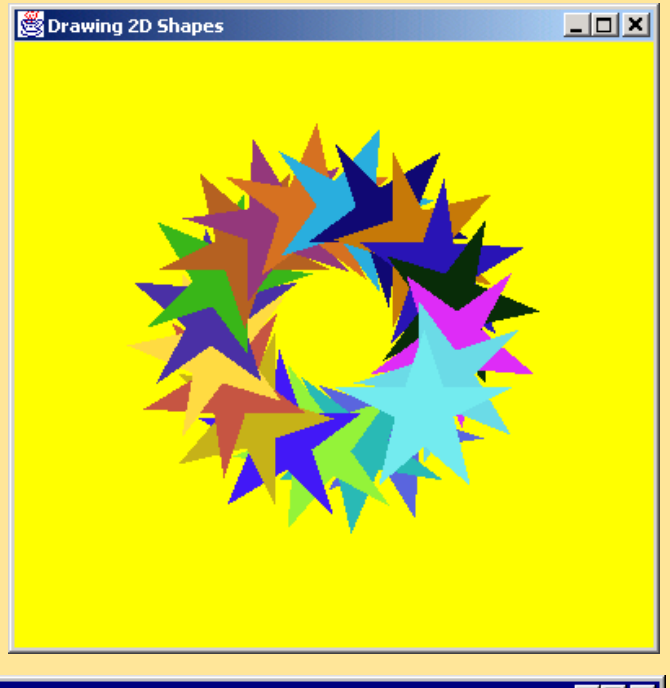
Create star

Lines 55-67

Draw filled, randomly colored star 20 times around origin

Shapes2.java

```
71
72    // execute application
73    public static void main( String args[] )
74    {
75       Shapes2 application = new Shapes2();
76
77       application.setDefaultCloseOperation(
78          JFrame.EXIT_ON_CLOSE );
79    }
80
81 }  // end class Shapes2
```



11.10 (Optional Case Study) Thinking About Objects: Designing Interfaces With the UML

- Use UML to represent listener interfaces
  - Class diagram modeling *realizations*
    - Classes *realize*, or implement, interface behaviors
    - **`Person`** realizes **`DoorListener`**
    - In Java, class **`Person`** implements interface **`DoorListener`**

Fig. 11.25 Class diagram that models class `Person` realizing interface `DoorListener`.

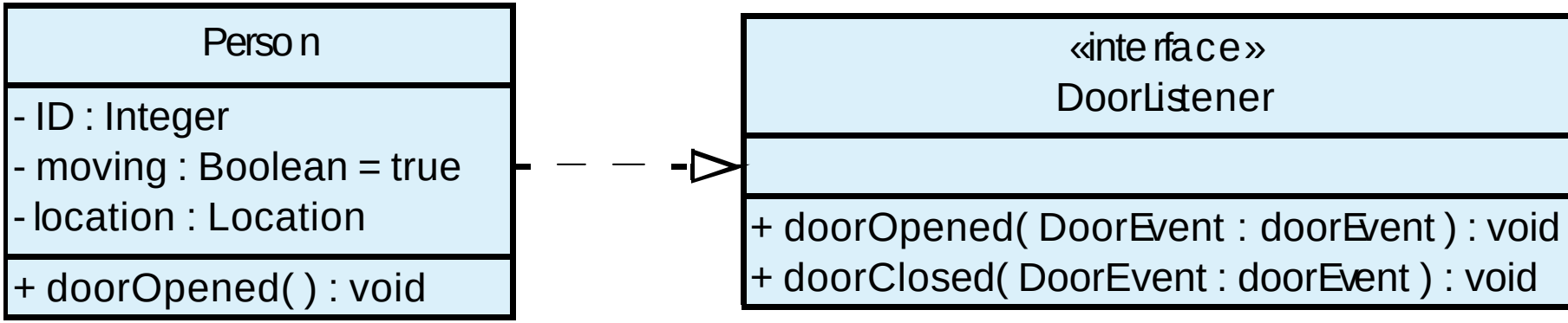
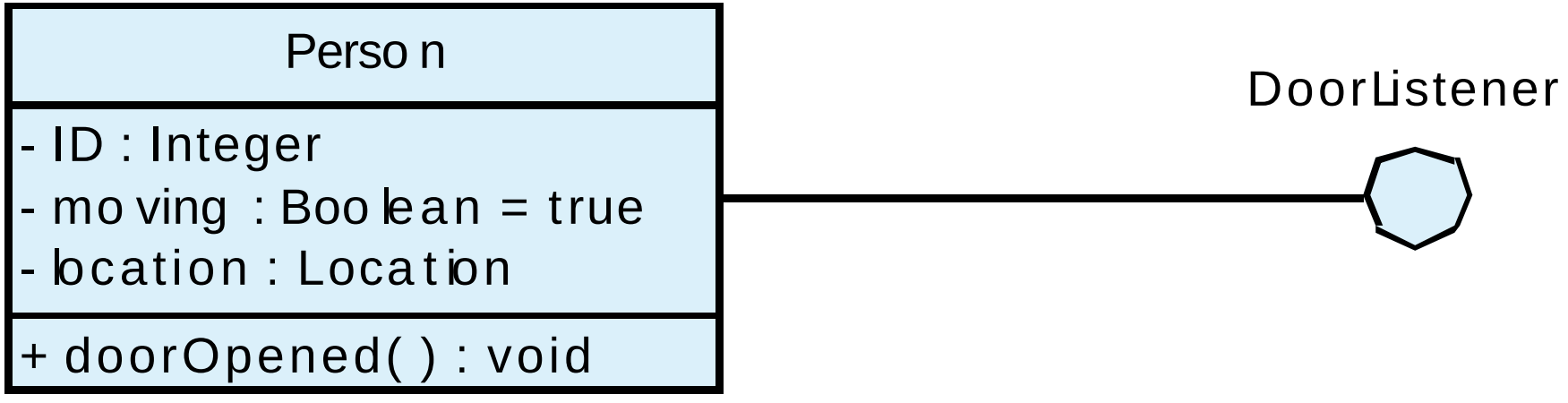


Fig. 11.26 Elided class diagram that models class `Person` realizing interface `DoorListener`



Outline

Person.java

Lines 3 and 14-15

```
1  // Person.java
2  // Generated from Fig. 11.24
3  public class Person implements DoorListener {
4
5     // attributes
6     private int ID;
7     private boolean moving = true;
8     private Location location;
9
10    // constructor
11    public Person() {}
12
13    // methods of DoorListener
14    public void doorOpened( DoorEvent doorEvent ) {}
15    public void doorClosed( DoorEvent doorEvent ) {}
16 }
```

Class **Person** must implement **DoorListener** methods

Fig. 11.27 Class diagram that models realizations in the elevator model.

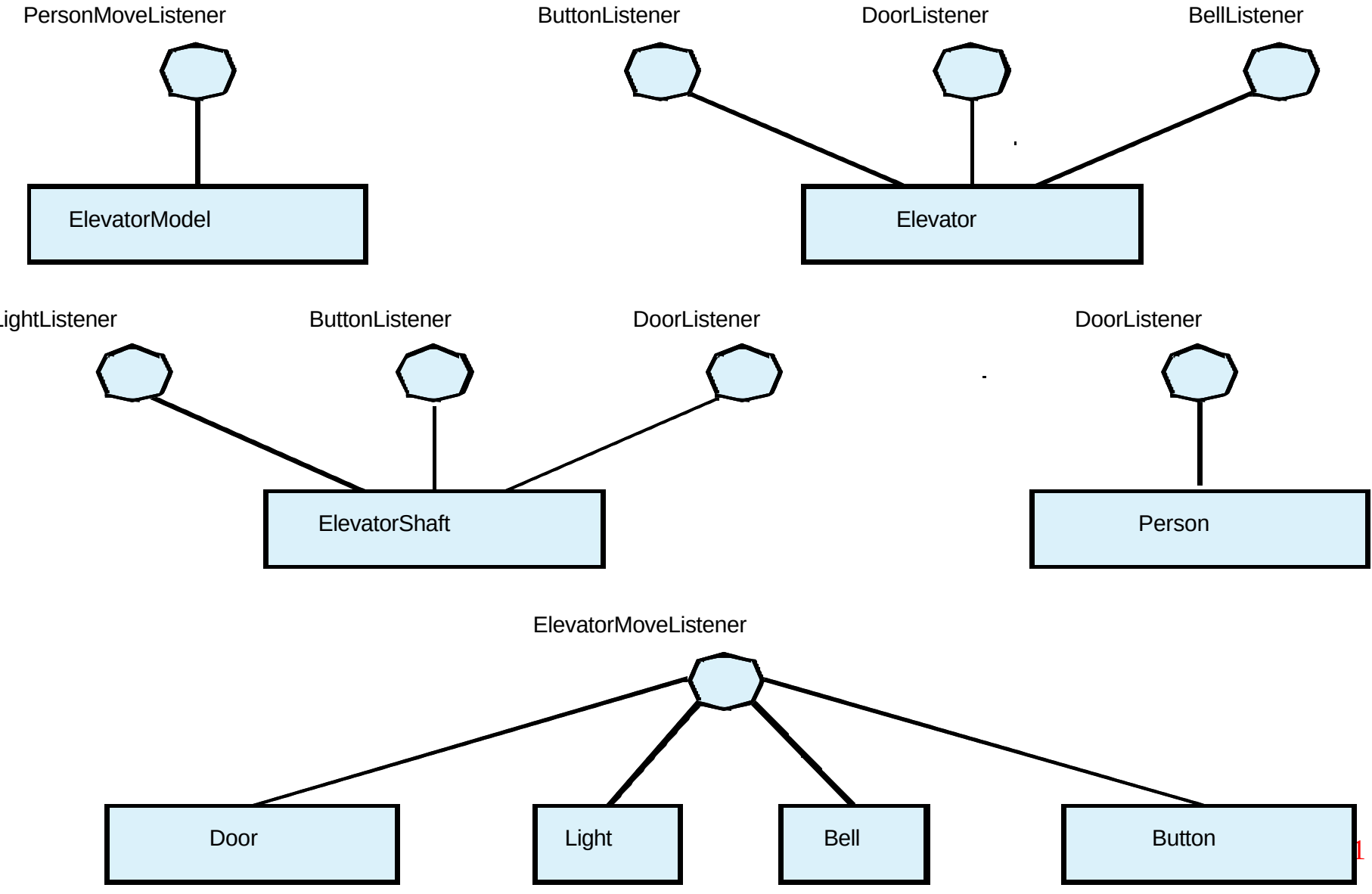


Fig. 11.28 Class diagram for listener interfaces