Capitolo 1 – Le immagini e Java2D

Outline

1.1 Introduction

- In this chapter
  - Draw 2D shapes
  - Colors
  - Fonts
- Java appealing for its graphics support
  - Has a class hierarchy for its graphics classes and 2D API classes
- Java coordinate system
  - (x,y) pairs
    - x - horizontal axis
    - y - vertical axis
  - Upper left corner is (0,0)
  - Coordinates measured in pixels (smallest unit of resolution)

1.2 Graphics Contexts and Graphics Objects

- Graphics context
  - Enables drawing on screen
  - **`Graphics`** object manages graphics context
    - Controls how information is drawn
    - Has methods for drawing, font manipulation, etc
    - We have used **`Graphics`** reference **`g`** for applets
  - **`Graphics`** an **`abstract`** class
    - Cannot instantiate objects
    - Implementation hidden - more portable
- Class **`Component`**
  - Superclass for many classes in **`java.awt`**
  - Method **`paint`** takes **`Graphics`** object as argument

1.2 Graphics Contexts and Graphics Objects

- Class **`Component`** (continued)
  - **`paint`** called automatically when applet starts
  - **`paint`** often called in response to an event
  - **`repaint`** calls **`update`**, which forces a **`paint`** operation
    - **`update`** rarely called directly
    - Sometimes overridden to reduce "flicker"

    Headers:

    ```
    public void repaint()
    public void update( Graphics g )
    ```
- In this chapter
  - Focus on **`paint`** method

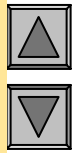
1.3 Color Control

- Class **`Color`**
  - Defines methods and constants for manipulating colors
  - Colors created from red, green, and blue component
    - RGB value: 3 integers from 0 to 255 each, or three floating point values from 0 to 1.0 each
    - Larger the value, more of that color
  - Color methods **`getRed`**, **`getGreen`**, **`getBlue`**

```
19        g.setColor( new Color( 255, 0, 0 ) );
```

  - Graphics method **`setColor`** sets drawing color
    - Takes **`Color`** object
    - Method **`getColor`** returns **`Color`** object with current settings

```
1  // Fig. 1.5: 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
```



Outline

1. `import`

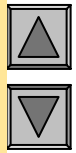
1.1 Class definition

1.2 Define `paint`

Call to **`setColor`**, takes a **`Color`** object.

`fillRect` fills a rectangle (more later)

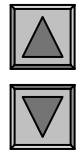
```
28       // set new drawing color using static Color objects
29       g.setColor( Color.blue );
30       g.fillRect( 25, 75, 100, 20 );
31       g.drawString( "Current RGB: " + g.getColor(), 130, 90 );
32
33       // display individual RGB values
34       Color c = Color.magenta;
35       g.setColor( 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 }
```



Outline

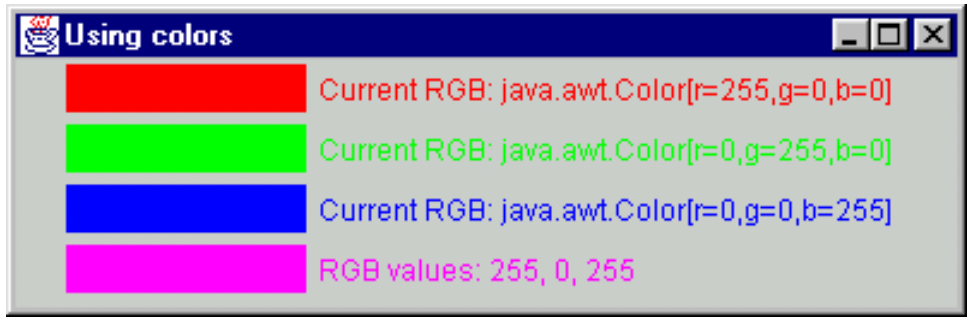
1.2 Define `paint`

1.3 `main`



Outline

Program Output



1.3 Color Control

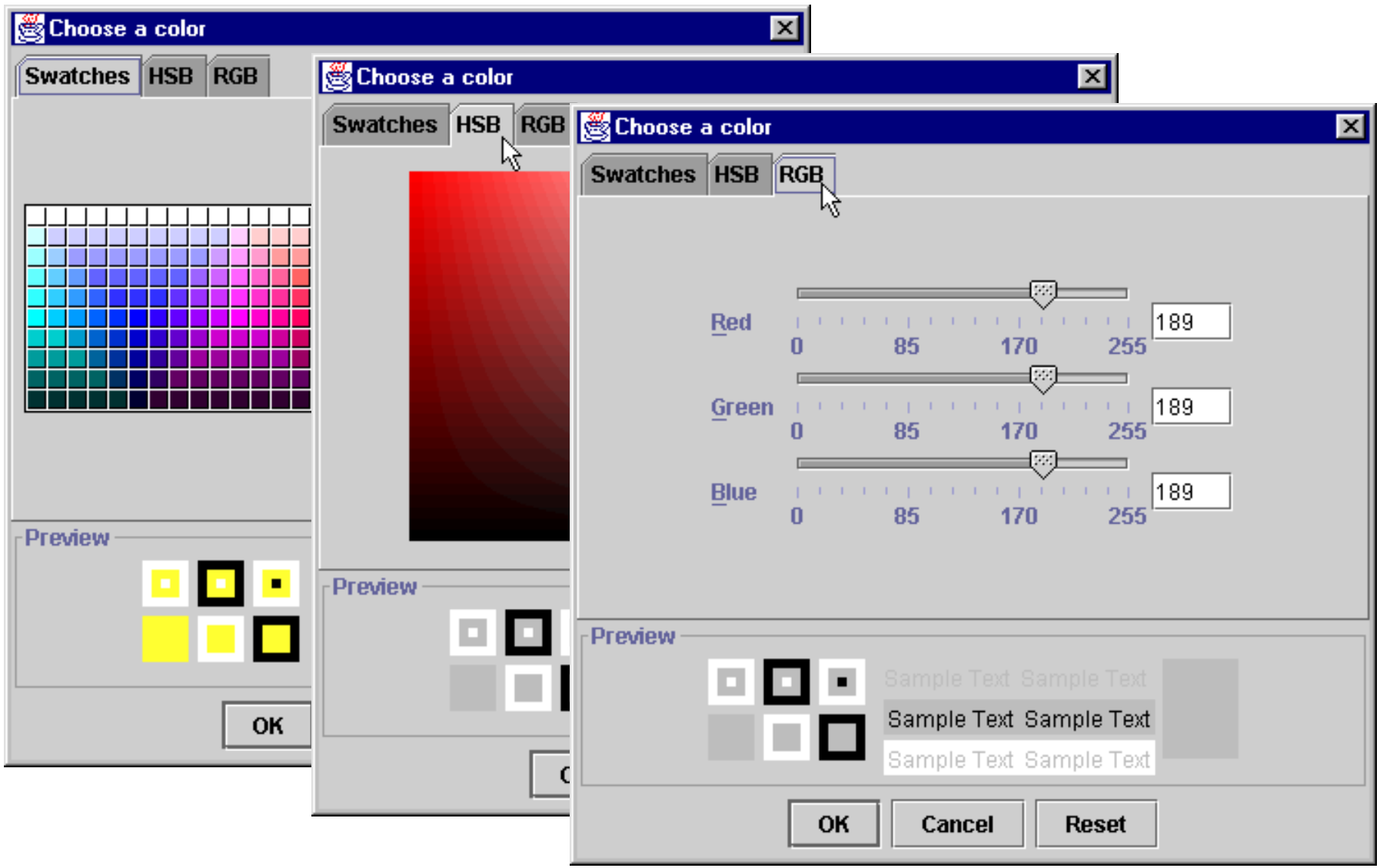
- Component **`JColorChooser`**
  - Displays dialog allowing user to select a color

```
24                color =
25                   JColorChooser.showDialog( ShowColors2.this,
26                      "Choose a color", color );
```

  - **`static`** method **`showDialog`**
    - First argument: reference to parent **`Component`** (window from which dialog being displayed)
      - Modal dialog - user cannot interact with other dialogs while active
    - Second argument: **`String`** for title bar
    - Third argument: Initial selected color
    - Returns a **`Color`** object

1.3 Color Control

JColorChooser



1.3 Color Control

```
10    private Container c;
16       c = getContentPane();
31                c.setBackground( color );
32                c.repaint();
```

- Class **`Container`**
  - Method **`setBackground`** - takes **`Color`** object
  - Sets background color
- **`repaint`**
  - Calls **`update`** method, ensures background gets changed

```
1  // Fig. 1.6: 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;
```

Use static method **showDialog** to select a color.

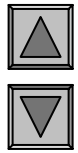
Outline

1. **import**

1.1 Class definition

2 Define **paint**

2.1 **showDialog**

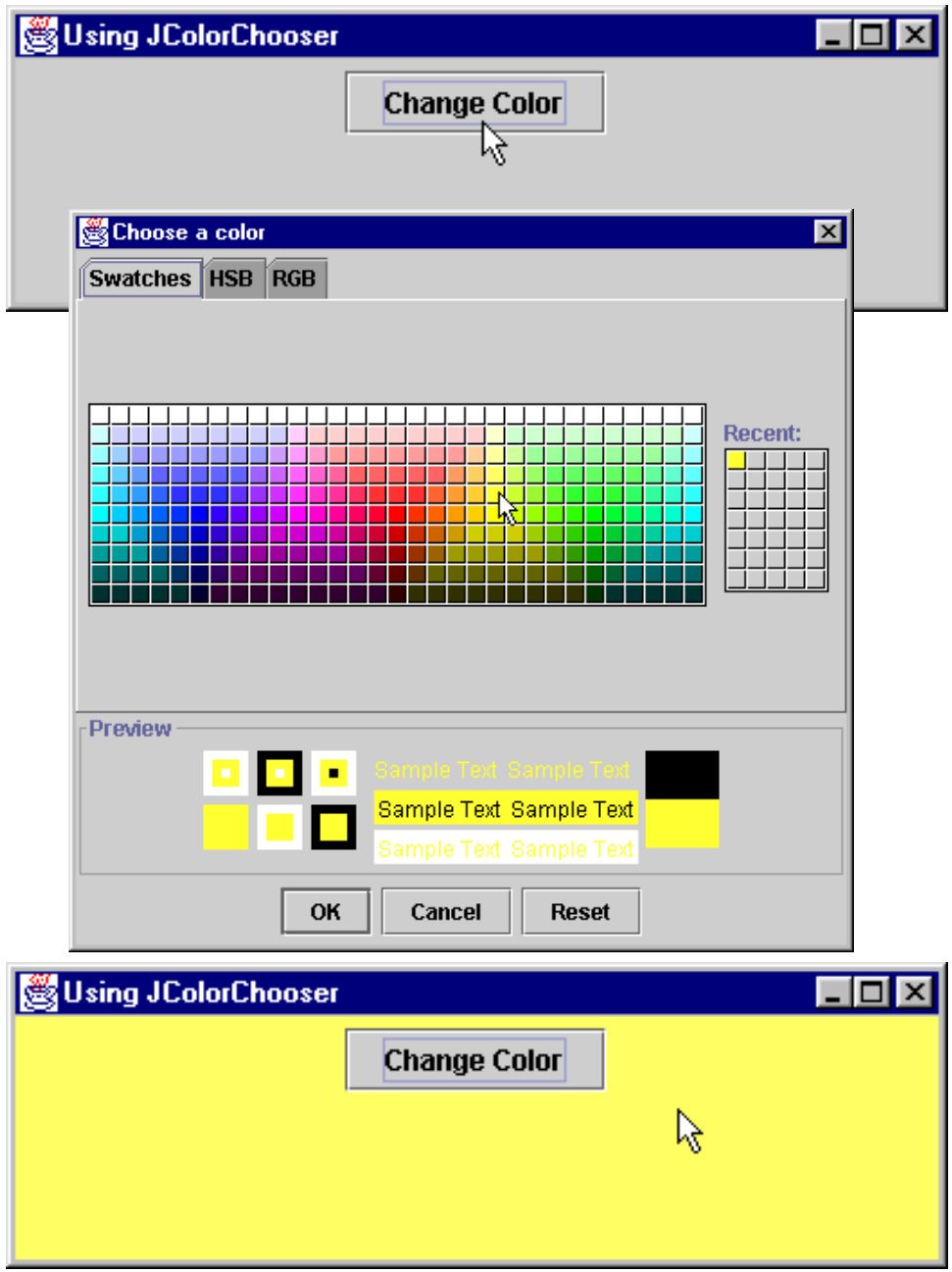


Outline

```
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 }
```

Container method **setBackground** takes a **Color** object.

Call **repaint** to update the screen.



Program Output

1.4 Font Control

- Class **`Font`**
  - Constructor takes three arguments

```
public Font( String name, int style, int size)
```

    - name: font supported by system (**`Serif`**, **`Monospaced`**, etc)
    - style: constants **`FONT.PLAIN`**, **`FONT.ITALIC`**, **`FONT.BOLD`**
      - Combinations: **`FONT.ITALIC + FONT.BOLD`**
    - size: measured in points (1/72 of an inch)

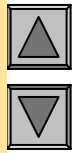
  - Use similar to **`Color`**
  - **`g.setFont( fontObject );`**

```
20        g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
```


1.4 Font Control

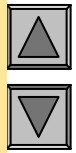
- Methods
  - `getStyle()`
  - `getSize()`
  - `getName()`
  - `getFamily()`
  - `isPlain()`
  - `isBold()`
  - `isItalic()`
  - `getFont()`
  - `setFont(Font f)`

```
1  // Fig. 1.9: 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
```



Notice use of **`Font`** objects and method calls.

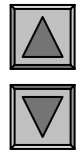
```
28       // set current font to SansSerif (Helvetica),
29       // plain, 14pt and draw a string
30       g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
31       g.drawString( "SansSerif 14 point plain.", 20, 90 );
32
33       // set current font to Serif (times), bold/italic,
34       // 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 }
```



Outline

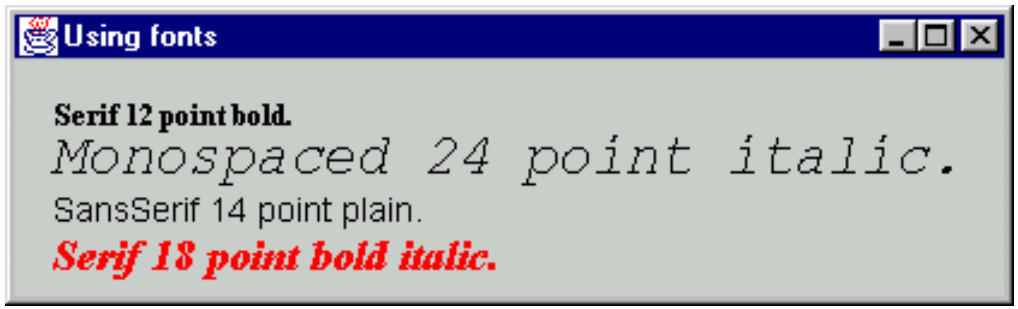
2. **paint**

3. **main**



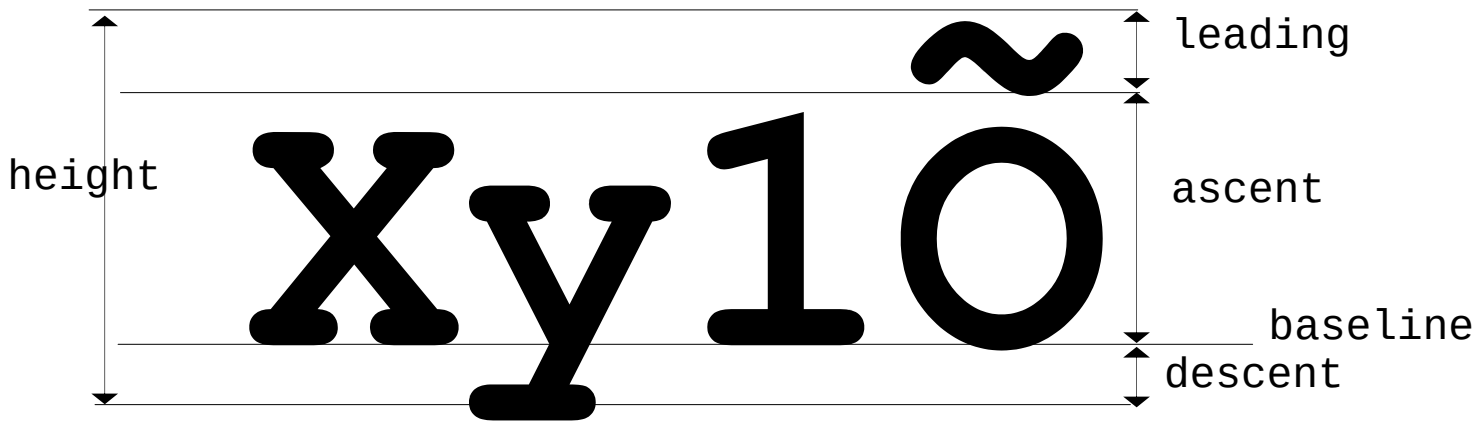
Outline

Program Output



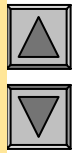
1.4 Font Control

- Class **`FontMetrics`**
  - Has methods for getting font metrics
  - **`g.getFontMetrics`** - returns **`FontMetrics`** object for current font



1.4 Font Control

- **`FontMetrics`** (and **`Graphics`**) methods
  - **`getAscent()`**
  - **`getDescent()`**
  - **`getLeading()`**
  - **`getHeight()`**
  - **`getFontMetrics()`**
  - **`getFontMetrics( Font f )`**



Outline

FontMetrics Example

1. `import`

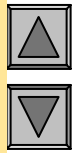
1.1 Constructor

1.2 `paint`

```
1  // Fig. 1.12: 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
```

Create new **`fontMetrics`** object, with fontmetrics for current font.

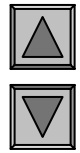
```
27       Font font = new Font( "Serif", Font.ITALIC, 14 );
28       fm = g.getFontMetrics( font );
29       g.setFont( font );
30       g.drawString( "Current font: " + font, 10, 130 );
31       g.drawString( "Ascent: " + fm.getAscent(), 10, 145 );
32       g.drawString( "Descent: " + fm.getDescent(), 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 }
```



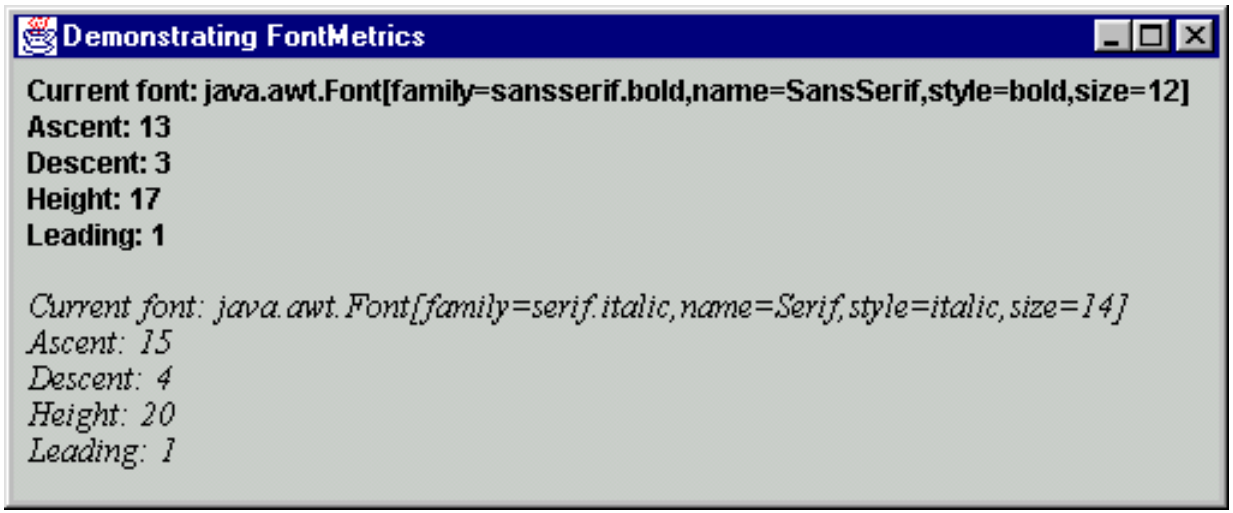
Outline

1.2 paint

1.3 main



Outline



Program Output

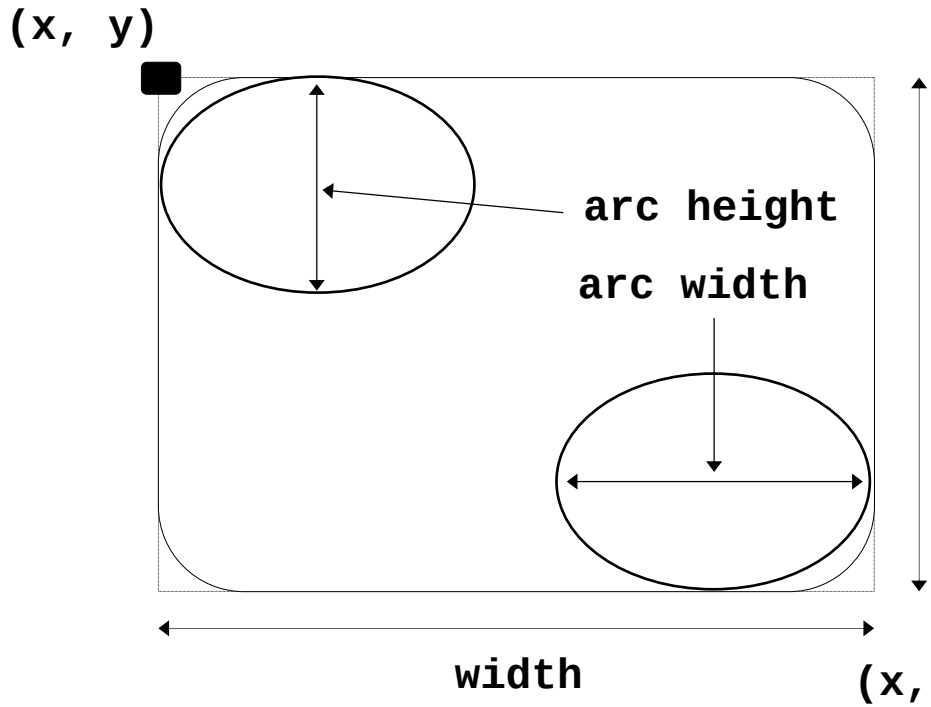
1.5 Drawing Lines, Rectangles and Ovals

- Graphics methods for drawing shapes
  - **`drawLine( x1, y1, x2, y2 )`**
    - Line from **`x1`**, **`y1`** to **`x2`**, **`y2`**
  - **`drawRect( x1, y1, width, height)`**
    - Draws rectangle with upper left corner **`x1, y1`**
  - **`fillRect( x1, y1, width, height)`**
    - As above, except fills rectangle with current color
  - **`clearRect (x1, y1, width, height)`**
    - As above, except fills rectangle with background color
  - **`draw3DRect(x1, y1, width, height, isRaised)`**
    - Draws **`3D`** rectangle, raised if **`isRaised true,`** else lowered

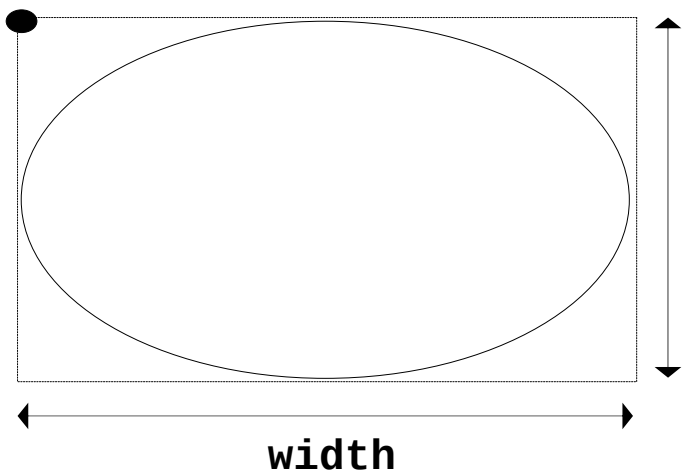
1.5 Drawing Lines, Rectangles and Ovals

- Graphics methods for drawing shapes (continued)
  - **`fill3DRect`**
    - As previous, but fills rectangle with current color
  - **`drawRoundRect( x, y, width, height, arcWidth, arcHeight )`**
    - Draws rectangle with rounded corners. See diagram next slide.
  - **`fillRoundRect( x, y, width, height, arcWidth, arcHeight )`**
  - **`drawOval( x, y, width, height )`**
    - Draws oval in bounding rectangle (see diagram)
    - Touches rectangle at midpoint of each side
  - **`fillOval ( x, y, width, height )`**

1.5 Drawing Lines, Rectangles and Ovals



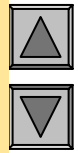
drawRoundRect parameters



drawOval parameters

```
1  // Fig. 1.14: 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
```

Draw various shapes using **`Graphics`** methods.



Outline

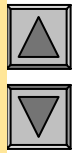
Lines, Rectangles, and Ovals Example

1. `import`

1.1 Constructor

2. `paint`

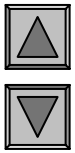
```
31       g.setColor( Color.yellow );
32       g.draw3DRect( 5, 100, 90, 55, true );
33       g.fill3DRect( 100, 100, 90, 55, false );
34
35       g.setColor( Color.magenta );
36       g.drawOval( 195, 100, 90, 55 );
37       g.fillOval( 290, 100, 90, 55 );
38    }
39
40    public static void main( 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 }
```



Outline

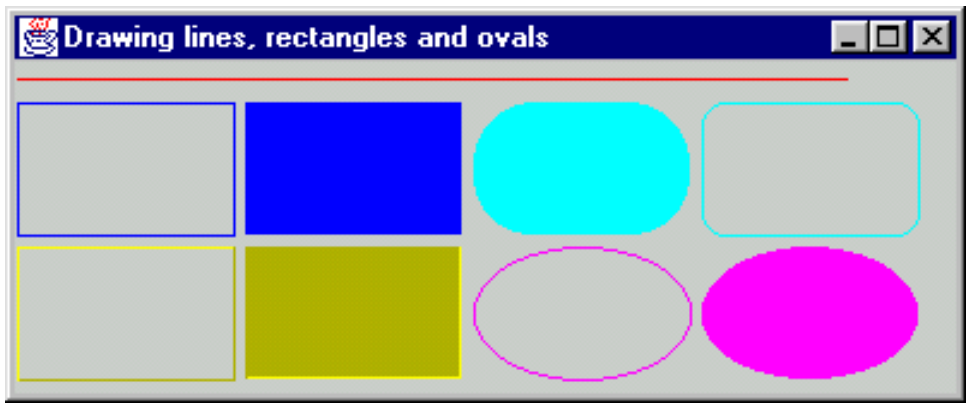
2. **paint**

3. **main**



Outline

Program Output



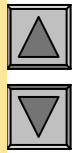
1.6 Drawing Arcs

- Arc
  - Portion of an oval
  - Measured in degrees
    - Starts at a starting angle and sweeps the number of degrees specified by arc angle
    - Positive - counterclockwise
    - Negative - clockwise
  - When drawing an arc, specify bounding rectangle for an oval
  - **`drawArc( x, y, width, height, startAngle, arcAngle )`**
  - **`fillArc`** - as above, but draws a solid arc (sector)

```
1  // Fig. 1.19: 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
```

Draw a bounding rectangle for the arc.

`Graphics` method **`drawArc( x, y, width, height, startAngle, arcAngle )`**.



Outline

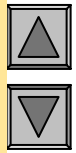
1. **`import`**

1.1 **Constructor**

2. **`paint`**

2.1 **`drawArc`**

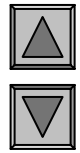
```
30       // start at 0 and sweep -270 degrees
31       g.setColor( Color.yellow );
32       g.drawRect( 185, 35, 80, 80 );
33       g.setColor( Color.black );
34       g.drawArc( 185, 35, 80, 80, 0, -270 );
35
36       // start at 0 and sweep 360 degrees
37       g.fillArc( 15, 120, 80, 40, 0, 360 );
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 }
```



Outline

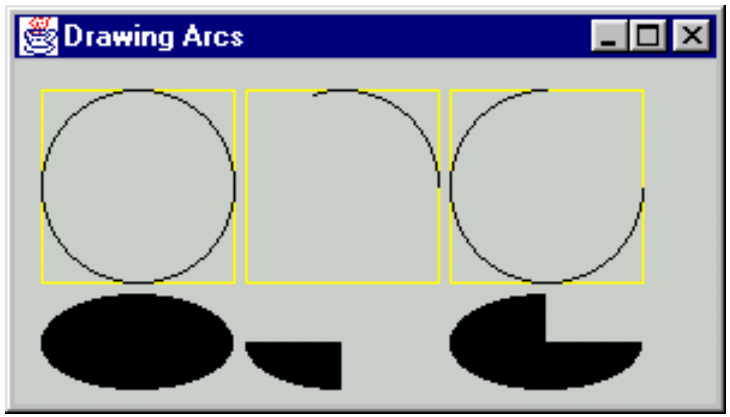
2.2 `fillArc`

3. `main`



Outline

Program Output



1.7 Drawing Polygons and Polylines

- Polygon
  - Multisided shape
- Polyline
  - Series of connected points
- Methods of class Polygon
  - **`drawPolygon( xPoints[], yPoints[], points )`**
    - Draws a polygon, with x and y points specified in arrays. Last argument specifies number of points
    - Closed polygon, even if last point different from first
  - **`drawPolyline ( xPoints[], yPoints, points )`**
    - As above, but draws a polyline
    - Open polyline

1.7 Drawing Polygons and Polylines

- Methods of class **`Polygon`**
  - **`drawPolygon( myPolygon )`**
    - Draws specified closed polygon
  - **`fillPolygon( xPoints[], yPoints[], points )`**
    - Draws a solid polygon
  - **`fillPolygon( myPolygon )`**
    - Draws specified solid polygon
  - **`Polygon( xValues[], yValues[], numberOfPoints )`**
    - Constructs a new polygon object
  - **`myPolygon.addPoint( x, y )`**
    - Add pairs of x-y coordinates to polygon object

```
1  // Fig. 1.21: DrawPolygons.java
2  // Drawing polygons
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class DrawPolygons extends JFrame {
8     public DrawPolygons()
9     {
10       super( "Drawing Polygons" );
11
12       setSize( 275, 230 );
13       show();
14    }
15
16    public void paint( Graphics g )
17    {
18       int xValues[] = { 20, 40, 50, 30, 20, 15 };
19       int yValues[] = { 50, 50, 60, 80, 80, 60 };
20       Polygon poly1 = new Polygon( xValues, yVal
21
22       g.drawPolygon( poly1 );
23
24       int xValues2[] = { 70, 90, 100, 80, 70, 65, 60 };
25       int yValues2[] = { 100, 100, 110, 110, 130, 110, 90 };
26
27       g.drawPolyline( xValues2, yValues2, 7 );
28
29       int xValues3[] = { 120, 140, 150, 190 };
30       int yValues3[] = { 40, 70, 80, 60 };
31
```

Initialize **Polygon** object with array of **x** and **y** coordinates.

Draw **Polyline** object using array of **x** and **y** coordinates.

One ver… …n

Outline

1. **import**

1.1 **Constructor**

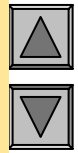
2. **paint**

Polygon

2.2 **drawPolyline**

```
32       g.fillPolygon( xValues3, yValues3, 4 );
33
34       Polygon poly2 = new Polygon();
35       poly2.addPoint( 165, 135 );
36       poly2.addPoint( 175, 150 );
37       poly2.addPoint( 270, 200 );
38       poly2.addPoint( 200, 220 );
39       poly2.addPoint( 130, 180 );
40
41       g.fillPolygon( poly2 );
42    }
43
44    public static void main( String args[] )
45    {
46       DrawPolygons app = new DrawPolygons();
47
48       app.addWindowListener(
49          new WindowAdapter() {
50             public void windowClosing( WindowEvent e )
51             {
52                System.exit( 0 );
53             }
54          }
55       );
56    }
57 }
```

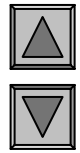
Use **addPoint** to add **x,y** pairs to the **Polygon**.



Outline

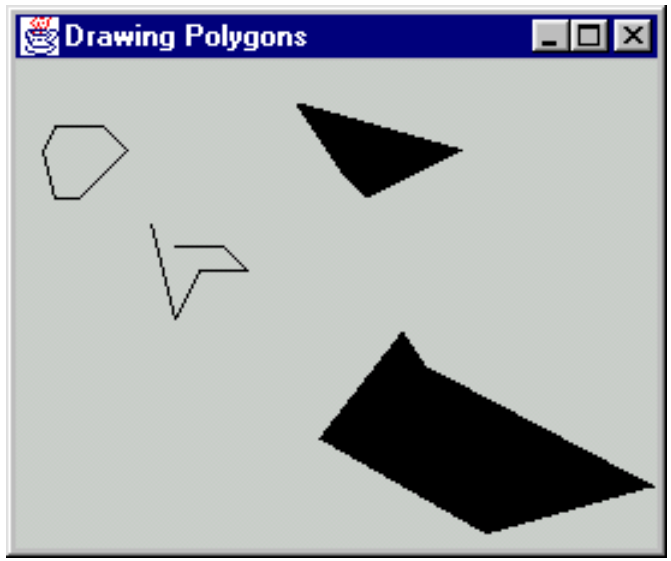
addPoint

3. main



Outline

Program Output



1.8 The Java2D API

- Java2D API
  - Advanced two dimensional graphics capabilities
  - Too many capabilities to cover (for overview, see Java2D demo in Chapter 3)
- Drawing with Java2D API
  - Use instance of class **`Graphics2D`** (package **`java.awt`**)
    - Subclass of **`Graphics`**
    - Has all graphics capabilities we have previously discussed
  - Must downcast **`Graphics`** reference passed to **`paint`**

```
21      Graphics2D g2d = ( Graphics2D ) g;
```

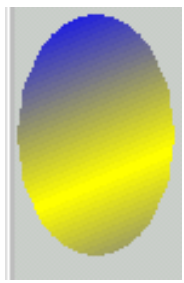
  - Technique used in programs of next section

1.9 Java2D Shapes

- Sample methods from **`Graphics2D`**
  - **`setPaint ( paintObject )`**
    - Determines color of shapes being displayed
    - **`paintObject`** is from a class that implements **`java.awt.Paint`**
    - Examples: class **`Color, GradientPaint, SystemColor, TexturePaint`**

1.9 Java2D Shapes

- Sample methods from **`Graphics2D`**
  - **`GradientPaint ( x1, y1, color1, x2, y2, color2, cyclic )`**



  - Creates gradient (slowly changing color) from **`x1`**, **`y1`**, to **`x2`**, **`y2`**, starting with **`color1`** and changing to **`color2`**
  - If **`cyclic true`**, then cyclic gradient (keeps transitioning colors)
    - If acyclic, only transitions colors once

1.9 Java2D Shapes

- Sample methods from **`Graphics2D`**
  - **`fill ( shapeObject )`**
    - Draws a filled **`Shape`** object
    - Class that implements **`Shape`** (**`java.awt`**)
      - **`Ellipse2D.Double`**, **`Rectangle2D.Double`**
    - Double-precision inner classes of **`Ellipse2D`**
  - **`setStroke( strokeObject )`**
    - Set shape's borders
    - Class that implements **`Stroke`** (**`java.awt`**)
    - **`BasicStroke( width )`** - One of many constructors
      - This constructor specifies width in pixels of border

```
34          g2d.setStroke( new BasicStroke( 10.0f ) );
```


1.9 Java2D Shapes

- Sample methods from **Graphics2D**
  - **draw (shapeObject)**
    - Draws specified **Shape** object

```
39       BufferedImage buffImage =
40          new BufferedImage(
41             10, 10, BufferedImage.TYPE_INT_RGB );
```

  - Class **BufferedImage**
    - Can produce images in color or grayscale
    - Create patterns by drawing into the **BufferedImage** object
    - 10 pixels high, 10 pixels wide
    - **TYPE_INT_RGB** - color image using RGB color scheme

1.9 Java2D Shapes

```
43       Graphics2D gg = buffImage.createGraphics();
44       gg.setColor( Color.yellow ); // draw in yellow
45       gg.fillRect( 0, 0, 10, 10 ); // draw a filled rectangle
```

- Class **BufferedImage**
  - Method **createGraphics**
    - Creates a **Graphics2D** object, used to draw into the **BufferedImage**

```
54       g2d.setPaint(
55          new TexturePaint(
56             buffImage, new Rectangle( 10, 10 ) ) );
```

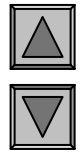
- Class **TexturePaint**
  - Constructor can take **BufferedImage** and area to use
  - **setPaint** - sets **Paint** object
    - Argument becomes fill for new objects

1.9 Java2D Shapes

- Class **`Arc2D.Double`**
  - Similar to normal arcs, except has another argument at end
    - **`Arc2D.PIE`** - close arc with two lines
    - **`Arc2D.CHORD`** - draws line from endpoints of arc
    - **`Arc2D.OPEN`** - arc not closed
- Class **`BasicStroke`**
  - Can be used to create customized dashed lines, set how lines end and join

```
1  // Fig. 1.22: 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 ) );
```

Use **GradientPaint** with method **setPaint**, then draw an ellipse.



Outline

1. **import**

1.1 Constructor

2 **paint**

2.1 Cast to **Graphics2D**

2.2 **GradientPaint**

Outline

2.3 BufferedImage

2.3.1 TexturePaint

```
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 ) );
```

Draw red rectangle.

Create new **BufferedImage**.

Use **TexturePaint** and draw a **RoundRectangle**.

```
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
```

Draw a white arc.

Draw a green line, yellow dashed line.

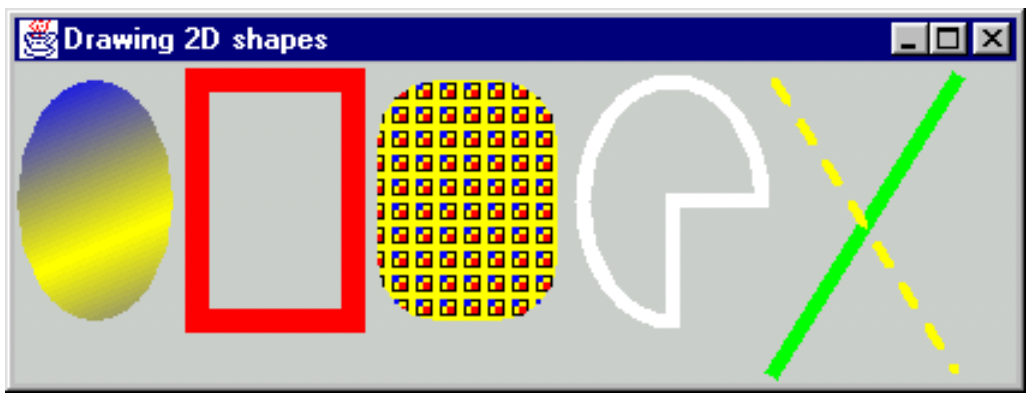
Outline

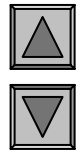
2.4 Arc2D.Double

2.5 Line2D.Double

3. main

```
87          app.addWindowListener(
88             new WindowAdapter() {
89                public void windowClosing( WindowEvent e )
90                {
91                   System.exit( 0 );
92                }
93             }
94          );
95       }
96    }
```





Outline

Program Output

1.9 Java2D Shapes

- Class **`GeneralPath`**
  - A general path is a shape made from lines and curves
  - Method **`moveTo`**
    - Specifies first point in a general path
  - Method **`lineTo`**
    - Draws a line to next point in general path
  - Method **`closePath`**
    - Draws line from last point to point specified in last call to **`moveTo`**

1.9 Java2D Shapes

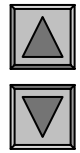
- Other methods
  - **`rotate( radians )`** - rotate next shape around origin
  - **`translate( x, y )`** - translates origin to **`x`**, **`y`**

- Upcoming program
  - Create a **`GeneralPath`** object for a star
    - Use two arrays to hold x and y values
  - Loop and use **`lineTo`** to add points to **`GeneralPath`**
  - Translate **`origin`**
  - Rotate around origin and draw **`GeneralPath`** object

```
50          g2d.fill( star );   // draw a filled star
```


```
1  // Fig. 1.23: 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
```

Arrays to hold x and y coordinates.



Outline

Java2D Example

1. `import`

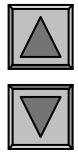
1.1 Constructor

2. `paint`

2.1 `Graphics2D` object

`eneralPath` object

```
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
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
```



Outline

Move to initial coordinates, create lines to points.

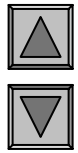
Randomize colors, rotate around origin, and draw star.

2.4 `lineTo`

2.5 Randomize colors

3 main

```
58       app.addWindowListener(
59          new WindowAdapter() {
60             public void windowClosing( WindowEvent e )
61             {
62                System.exit( 0 );
63             }
64          }
65       );
66    }
67 }
```



Outline

Program Output