

Supplement V.C: Java 2D

For Introduction to Java Programming, 6E
By Y. Daniel Liang

The text introduced how to draw lines, rectangles, ovals, arcs, and polygons, using the methods in the Graphics class. This supplement introduces Java 2D. Java 2D enables you to draw advanced and complex two-dimensional graphics.

1 Obtaining a Graphics2D Context

You used the drawing methods in the Graphics class in the text. The Graphics class is primitive. The Java 2D API provides the Graphics2D class (in the java.awt package) that extends Graphics with advanced capabilities. Normally, you write the code to draw graphics in the paintComponent method in a GUI component. The coding template for the method is as follows:

```
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
  
    // Use the method in Graphics to draw graphics  
    ...  
}
```

The parameter passed to the paintComponent method is actually also an instance of Graphics2D. So, to obtain a Graphics2D reference, you may simply cast the parameter g to Graphics2D as follows:

```
protected void paintComponent(Graphics g) {  
    super.paintComponent(g);  
  
    Graphics2D g2d = (Graphics2D)g;  
  
    // Use the method in Graphics2D to draw graphics  
    ...  
}
```

Since Graphics2D is a subclass of Graphics, all the methods in Graphics can be used in Graphics2D. Additionally, you can use the methods in Graphics2D.

2 Line2D, Rectangle2D, RoundRectangle2D, Arc2D, and Ellipse2D

You have used the simple methods in the Graphics class to draw lines, rectangles, arcs, and ellipses. The Java 2D API provides a more object-oriented approach for drawing various

shapes using the classes: Line2D.Double, Line2D.Float, Rectangle2D.Double, Rectangle2D.Float, RoundRectangle2D.Double, RoundRectangle2D.Float, Arc2D.Double, Arc2D.Float, Ellipses2D.Double, and Ellipses2D.Float. All these classes are static inner classes. For example, Line2D.Double refers to the static inner class Double defined in the Line2D class. You can use either Line2D.Double or Line2D.Float to create an object for modeling a line, depending on whether you want to use double or float for coordinates. Because double values are more convenient than the float values in Java, the examples will use the double values. These inner classes are also subclasses of their respective outer classes. So Line2D.Double is a subclass of Line2D. The shape classes Line2D, Rectangle2D, RoundRectangle2D, Arc2D, and Ellipse2D all implement the Shape interface. A Shape object can be displayed on a Graphics2D context using the draw(Shape) method.

Listing 1 gives a program that demonstrates how to draw various shapes on a Graphics2D context. Figure 1 shows a sample run of the program.

Listing 1 Graphics2DDemo.java (Drawing Shapes)

<Side remark: Line 2: import for shape classes>
<Side remark: Line 5: applet>
<Side remark: Line 11: main method>
<Side remark: Line 28: Graphics2D reference>
<Side remark: Line 30: draw a line>
<Side remark: Line 31: draw a rectangle>
<Side remark: Line 32: fill a rectangle>
<Side remark: Line 33: round rectangle>
<Side remark: Line 34: draw an arc>
<Side remark: Line 37: draw an ellipse>

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class Graphics2DDemo extends JApplet {
    public Graphics2DDemo() {
        add(new ShapePanel1());
    }

    /** Main method */
    public static void main(String[] args) {
        Graphics2DDemo applet = new Graphics2DDemo();
        applet.init();
        applet.start();
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("Graphics2DDemo");
    }
}
```

```

        frame.getContentPane().add(applet, BorderLayout.CENTER);
        frame.setSize(400, 130);
        frame.setVisible(true);
    }
}

class ShapePanel1 extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D)g;

        g2d.draw(new Line2D.Double(10, 10, 40, 80));
        g2d.draw(new Rectangle2D.Double(50, 10, 30, 70));
        g2d.fill(new Rectangle2D.Double(90, 10, 30, 70));
        g2d.fill(new RoundRectangle2D.Double(130, 10, 30, 70, 20, 30));
        g2d.draw(new Arc2D.Double(170, 10, 30, 70, 0, 270, Arc2D.OPEN));
        g2d.draw(new Arc2D.Double(220, 10, 30, 70, 0, 270, Arc2D.PIE));
        g2d.draw(new Arc2D.Double(260, 10, 30, 70, 0, 270, Arc2D.CHORD));
        g2d.draw(new Ellipse2D.Double(300, 10, 30, 70));
    }
}

```

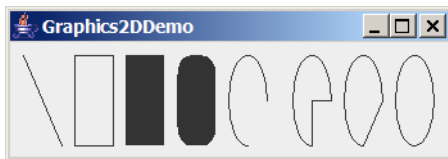


Figure 1

You can draw various shapes using Java 2D.

The shape classes Line2D, Rectangle2D, RoundRectangle2D, Arc2D, and Ellipse2D are in the java.awt.geom package. So, they are imported in line 2.

Graphics2DDemo is an applet. The main method (line 11) in the applet enables it to run standalone.

<Side remark: Line2D>

A Graphics2D reference is obtained in line 28 in order to invoke the methods in Graphics2D. The draw(Shape) method draws a shape and the fill(Shape) method draws a filled shape. The statement new Line2D.Double(10, 10, 40, 80) in line 30 creates an instance of Line2D.Double, which is also an instance of Line2D and Shape. The instance represents a line from (10, 10) to (40, 80).

<Side remark: Rectangle2D>

The statement new Rectangle.Double(50, 10, 30, 70) in line 31 creates an instance of Rectangle2D.Double, which is also an instance of Rectangle2D and Shape. The instance

represents a rectangle whose upper left corner point is (50, 10) with width 30 and height 70.

<Side remark: fill>

The fill(Shape) method in line 32 draws a filled rectangle.

<Side remark: RoundRectangle2D>

The statement new RoundRectangle2D.Double(130, 10, 30, 70, 20, 30) in line 33 creates an instance of RoundRectangle2D.Double, which is also an instance of RoundRectangle2D and Shape. The instance represents a round-cornered rectangle whose parameters are the same as in the drawRoundRect(int x, int y, int w, int h, int aw, int ah) method in the Graphics class.

<Side remark: Arc2D>

The statement new Arc2D.Double(170, 10, 30, 70, 0, 270, Arc2D.OPEN) in line 34 creates an instance of Arc2D.Double, which is also an instance of Arc2D and Shape. The instance represents an open arc. The parameters in this constructor are similar to the parameters in the drawArc(int x, int y, int w, int h, int startAngle, int arcAngle) method in the Graphics class, except that the last parameter specifies whether the arc is open or closed. The value Arc2D.OPEN specifies that the arc is open. The value Arc2D.PIE specifies that the arc is closed by drawing straight line segments from the start of the arc segment to the center of the full ellipse and from that point to the end of the arc segment. The value Arc2D.CHORD specifies that the arc is closed by drawing a straight line segment from the start of the arc segment to the end of the arc segment.

<Side remark: Ellipse2D>

The statement new Ellipse2D.Double(300, 10, 30, 70) in line 37 creates an instance of Ellipse2D.Double, which is also an instance of Ellipse2D and Shape. The instance represents an ellipse. The parameters in this constructor are the same as the parameters in the drawOval(int x, int y, int w, int h) method in the Graphics class.

3 GradientPaint

The GradientPaint class provides a way to fill a shape with gradually changing colors. To create a GradientPaint object, use the following constructor:

```
GradientPaint(float x1, float y1, Color color1, float x2, float y2,  
Color color2, boolean cyclic)
```

The parameters x1 and y1 specify the starting coordinate with color color1 and the parameters x2 and y2 specify the ending coordinate with color color2. The cyclic parameter

specifies whether the color pattern repeats.

Listing 2 gives a program that demonstrates the effect of changing colors using the GradientPaint class. Figure 2 shows a sample run of the program.

Listing 2 GradientPaintDemo.java (Demonstrating Gradient Colors)

<Side remark: Line 2: import for shape classes>
<Side remark: Line 5: applet>
<Side remark: Line 11: main method>
<Side remark: Line 28: Graphics2D reference>
<Side remark: Line 30: set Paint>
<Side remark: Line 32: fill a rectangle>
<Side remark: Line 34: set Paint>
<Side remark: Line 32: fill a rectangle>
<Side remark: Line 38: set Paint>
<Side remark: Line 41: fill a rectangle>
<Side remark: Line 43: set Paint>
<Side remark: Line 44: fill a rectangle>

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class GradientPaintDemo extends JApplet {
    public GradientPaintDemo() {
        add(new ShapePanel2());
    }

    /** Main method */
    public static void main(String[] args) {
        GradientPaintDemo applet = new GradientPaintDemo();
        applet.init();
        applet.start();
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("GradientPaintDemo");
        frame.getContentPane().add(applet, BorderLayout.CENTER);
        frame.setSize(400, 130);
        frame.setVisible(true);
    }
}

class ShapePanel2 extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D) g;

        g2d.setPaint(new GradientPaint(10, 10, Color.RED, 30, 30,
            Color.BLUE, false));
        g2d.fill(new Rectangle2D.Double(10, 10, 70, 70));
    }
}
```

```

        g2d.setPaint(new GradientPaint(90, 10, Color.YELLOW, 130, 130,
            Color.BLACK, true));
        g2d.fill(new Rectangle2D.Double(90, 10, 70, 70));

        g2d.setPaint(new GradientPaint(170, 10, Color.RED, 200, 200,
            Color.YELLOW, false));
        g2d.fill(new Rectangle2D.Double(170, 10, 70, 70));

        g2d.setPaint(Color.YELLOW);
        g2d.fill(new Rectangle2D.Double(250, 10, 70, 70));

        g2d.setColor(Color.GREEN);
        g2d.fill(new Rectangle2D.Double(330, 10, 70, 70));
    }
}

```



Figure 2

The rectangles are filled with gradient colors.

The statement `new GradientPaint(10, 10, Color.RED, 30, 30, Color.BLUE, false)` (line 30) creates an instance of `GradientPaint`, which is also an instance of the `Paint` interface. The `setPaint(Paint)` method sets a `Paint` object for the `Graphics2D` context. The program sets a new `Paint` object (lines 30, 34, 38) before drawing a filled rectangle (lines 32, 36, 40).

NOTE: You can also use the `setPaint` method to set a `Color` object (line 43) or use the `setColor` method in the `Graphics` class to set a color (line 46).

4 BasicStroke

The `BasicStroke` class defines a basic set of rendering attributes for drawing graphics. You can specify the width of the line, how the line ends (called *end caps*), how lines join together (called *line joins*), and whether the line is dashed. A line may end with a round cap (`CAP_ROUND`), a square cap (`CAP_SQUARE`), or no added decorations (`CAP_BUTT`).

Two line paths may be joined in three different ways:

- connecting the outer corners of their wide outlines with a straight segment (`JOIN_BEVEL`).
- joining path segments by extending their outside edges

until they meet (JOIN_MITER).

- joining path segments by rounding off the corner at a radius of half the line width (JOIN_ROUND).

You may use the constructors in Figure 3 to create a BasicStroke. In the last constructor, the dash parameter is an array that specifies a dashing pattern, alternating between opaque and transparent sections. The dash_phase parameter specifies the offset to start dashing pattern.

java.awt.BasicStroke	
+BasicStroke()	Constructs a BasicStroke with default attributes.
+BasicStroke(width: float)	Constructs a solid BasicStroke with the specified width.
+BasicStroke(width: float, cap: int, join: int)	Constructs a solid BasicStroke with the specified width, cap, and join.
+BasicStroke(width: float, cap: int, join: int, miterlimit: float)	Constructs a solid BasicStroke with the specified width, cap, join, and miter limit.
+BasicStroke(width: float, cap: int, join: int, miterlimit: float, dash: float[], dash_phase: float)	Constructs a solid BasicStroke with the specified width, cap, join, miter limit, dashing pattern, and the offset to start dashing pattern.

Figure 3

You can create a Stroke using the BasicStroke class.

Listing 3 gives a program that demonstrates how to draw shapes using basic strokes. Figure 4 shows a sample run of the program.

Listing 3 BasicStrokeDemo.java (Demonstrating Basic Strokes)

<Side remark: Line 2: import for shape classes>
<Side remark: Line 5: applet>
<Side remark: Line 11: main method>
<Side remark: Line 28: Graphics2D reference>
<Side remark: Line 30: set a stroke>
<Side remark: Line 32: draw a line>
<Side remark: Line 33: draw a rectangle>

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class BasicStrokeDemo extends JApplet {
    public BasicStrokeDemo() {
        add(new ShapePanel3());
    }

    /** Main method */
    public static void main(String[] args) {
        BasicStrokeDemo applet = new BasicStrokeDemo();
        applet.init();
    }
}
```

```

    applet.start();
    JFrame frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setTitle("BasicStrokeDemo");
    frame.getContentPane().add(applet, BorderLayout.CENTER);
    frame.setSize(400, 130);
    frame.setVisible(true);
}
}

class ShapePanel3 extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D)g;

        g2d.setStroke(new BasicStroke(15.0f, BasicStroke.CAP_ROUND,
            BasicStroke.JOIN_BEVEL));
        g2d.draw(new Line2D.Double(10, 10, 40, 80));
        g2d.draw(new Rectangle2D.Double(70, 10, 30, 70));

        g2d.setStroke(new BasicStroke(15.0f, BasicStroke.CAP_ROUND,
            BasicStroke.JOIN_MITER));
        g2d.draw(new Rectangle2D.Double(130, 10, 30, 70));

        g2d.setStroke(new BasicStroke(15.0f, BasicStroke.CAP_SQUARE,
            BasicStroke.JOIN_ROUND));
        g2d.draw(new Rectangle2D.Double(190, 10, 30, 70));

        g2d.setStroke(new BasicStroke(4.0f, BasicStroke.CAP_SQUARE,
            BasicStroke.JOIN_ROUND, 1.0f, new float[]{8}, 0));
        g2d.draw(new Line2D.Double(240, 10, 270, 80));
    }
}
}

```



Figure 4

You can specify the attributes for strokes.

The statement `new BasicStroke(15.0f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_BEVEL)` (line 30) creates an instance of `BasicStroke`, which is also an instance of the `Stroke` interface. The `setStroke(Stroke)` method sets a `Stroke` object for the `Graphics2D` context. The program sets a new `Stroke` object with width 15.0f, round end cap, and bevel join in lines 30-31, sets a new `Stroke` object with width 15.0f, round end cap, and miter join in lines 35-36, sets a new `Stroke` object with width 15.0f, round square cap, and round

join in lines 39-40, and finally sets a new Stroke object with width 4.0f, round square cap, round join, miter limit 1.0, dashing pattern {8}, and dash phase 0 in lines 43-44.

4 GeneralPath

The GeneralPath class represents a geometric path constructed from straight lines, and quadratic and cubic curves. It can contain multiple subpaths. You may create a GeneralPath using its no-arg constructor, use the moveTo(float x, float y) method to move the current point, use the lineTo(float x, float y) to add a point to the path by drawing a straight line from the current point to the specified new point, and use the closePath() method to connect the current point with the point in the last moveTo method.

Listing 4 gives a program that demonstrates general paths. Figure 5 shows a sample run of the program.

Listing 4 GeneralPathDemo.java (Demonstrating General Paths)

<Side remark: Line 2: import for shape classes>
<Side remark: Line 5: applet>
<Side remark: Line 11: main method>
<Side remark: Line 28: Graphics2D reference>
<Side remark: Line 29: a general path>
<Side remark: Line 30: initial point>
<Side remark: Line 31: draw a line>
<Side remark: Line 33: close general path>
<Side remark: Line 35: display general path>

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class GeneralPathDemo extends JApplet {
    public GeneralPathDemo() {
        add(new ShapePanel4());
    }

    /** Main method */
    public static void main(String[] args) {
        GeneralPathDemo applet = new GeneralPathDemo();
        applet.init();
        applet.start();
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("GeneralPathDemo");
        frame.getContentPane().add(applet, BorderLayout.CENTER);
        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}
```

```

class ShapePanel4 extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D)g;
        GeneralPath generalPath = new GeneralPath();
        generalPath.moveTo(10, 10); // Set the initial point at (10, 10)
        generalPath.lineTo(100, 10); // Draw a line to (100, 10)
        generalPath.lineTo(55, 90); // Draw a line to (55, 90)
        generalPath.closePath(); // Close the path

        g2d.draw(generalPath); // Draw the general path
    }
}

```

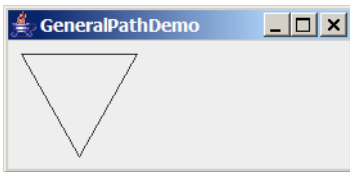


Figure 5

You can draw arbitrary paths using the GeneralPath class.

The statement new GeneralPath() (line 29) creates an empty general path. The moveTo(10, 10) method in line 30 sets the current point at (10, 10). The lineTo(100, 10) method draws a line from (10, 10) to (100, 10) and the current point is now at (100, 10). The closePath() method connects the current point with the point set by the last moveTo method, which is (10, 10) in this case. Line 35 displays the general path.

5 Translate

Java 2D is very powerful with many useful features. You can use the translate(double x, double y) method in the Graphics2D class to move to subsequent rendering in the Graphics object x pixels horizontally and y pixels vertically. For example, translate(5, -10) moves subsequent rendering 5 pixels to the right and 10 pixels up, and translate(-5, 10) moves all shapes 5 pixels to the left and 10 pixels down.

Listing 5 gives a program that demonstrates the translate method. Figure 6 shows a sample run of the program.

Listing 5 TranslateDemo.java (Demonstrating Translate)

<Side remark: Line 2: import for shape classes>

<Side remark: Line 5: applet>

<Side remark: Line 11: main method>

<Side remark: Line 28: Graphics2D reference>

<Side remark: Line 29: a rectangle>
<Side remark: Line 31: random number>
<Side remark: Line 33: set a new color>
<Side remark: Line 35: display rectangle>
<Side remark: Line 36: move rectangle>

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class TranslateDemo extends JApplet {
    public TranslateDemo() {
        add(new ShapePanel5());
    }

    /** Main method */
    public static void main(String[] args) {
        TranslateDemo applet = new TranslateDemo();
        applet.init();
        applet.start();
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("TranslateDemo");
        frame.getContentPane().add(applet, BorderLayout.CENTER);
        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}

class ShapePanel5 extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D)g;
        Rectangle2D rectangle = new Rectangle2D.Double(10, 10, 50, 60);

        java.util.Random random = new java.util.Random();
        for (int i = 0; i < 10; i++) {
            g2d.setColor(new Color(random.nextInt(256),
                random.nextInt(256), random.nextInt(256)));
            g2d.draw(rectangle);
            g2d.translate(20, 5);
        }
    }
}
```

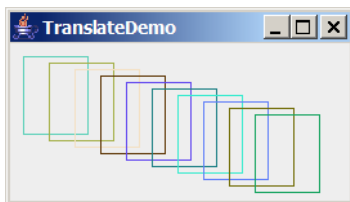


Figure 6

You can use the translate method to move a shape.

Line 31 creates a Random object. The Random class was introduced in §7.5.2, "The Random Class." Invoking random.nextInt(256) (line 33) generates a random int value between 0 and 255. The setColor method (line 33) sets a new color for subsequent rendering. Line 35 draws a rectangle. The translate(20, 5) method in line 36 moves the subsequent rendering 20 pixels to the right and 5 pixels down.

6 Rotate

You can use the rotate(double theta) method in the Graphics2D class to rotate subsequent rendering in the Graphics object by theta degrees from the origin, where theta is a double value in radians. By default the origin is (0, 0). You can use the translate(x, y) method to move the origin to a specified location. For example, rotate(Math.PI / 2) rotates subsequent rendering 45 degrees counterclockwise along the northern direction from the origin, as shown in Figure 7.

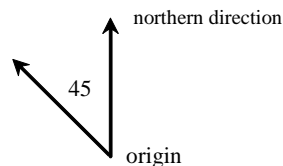


Figure 7

The rotate method rotates along the northern direction from the origin.

Listing 7 gives a program that demonstrates the rotate method. Figure 8 shows a sample run of the program.

Listing 7 RotateDemo.java (Demonstrating Translate)

```
<Side remark: Line 2: import for shape classes>
<Side remark: Line 5: applet>
<Side remark: Line 11: main method>
<Side remark: Line 28: Graphics2D reference>
<Side remark: Line 29: a rectangle>
<Side remark: Line 31: new origin>
<Side remark: Line 32: random number>
<Side remark: Line 34: set a new color>
<Side remark: Line 36: display rectangle>
<Side remark: Line 37: rotate>

import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
```

```

public class RotateDemo extends JApplet {
    public RotateDemo() {
        add(new ShapePanel6());
    }

    /** Main method */
    public static void main(String[] args) {
        RotateDemo applet = new RotateDemo();
        applet.init();
        applet.start();
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("RotateDemo");
        frame.getContentPane().add(applet, BorderLayout.CENTER);
        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}

class ShapePanel6 extends JPanel {
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D)g;
        Rectangle2D rectangle = new Rectangle2D.Double(20, 20, 50, 60);

        g2d.translate(200, 150);
        java.util.Random random = new java.util.Random();
        for (int i = 0; i < 10; i++) {
            g2d.setColor(new Color(random.nextInt(256),
                random.nextInt(256), random.nextInt(256)));
            g2d.draw(rectangle);
            g2d.rotate(Math.PI / 5);
        }
    }
}

```

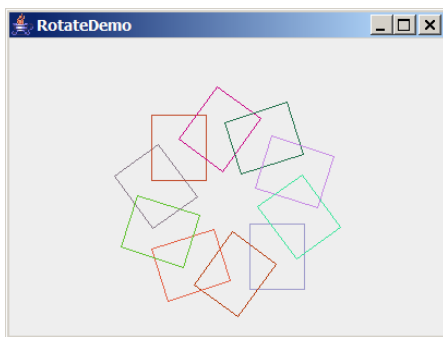


Figure 8

You can use the translate method to move a shape.

The translate(200, 150) method moves the origin from (0, 0) to (200, 150) in line 31. The loop is repeated ten times.

Each iteration sets a new color randomly (line 34), draws the rectangle (line 36), and rotates the coordinate system (line 37).