



# Computer Graphics Program in Java

Bachelor in Information Management

Banepa NIST College  
Computer Graphics  
*Compiled By:* Dinesh Ghemosu

July 29, 2023

**Program 1: Displaying Different Text**

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class DifferentText extends JFrame {
5     private final int WIDTH = 800;
6     private final int HEIGHT = 600;
7
8     public DifferentText() {
9         setSize(WIDTH, HEIGHT);
10        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        setLocationRelativeTo(null);
12        setVisible(true);
13    }
14
15    @Override
16    public void paint(Graphics g) {
17        super.paint(g);
18        displayText(g);
19    }
20
21    private void displayText(Graphics g) {
22        // Text 1: Color RED, Size 12
23        g.setColor(Color.RED);
24        g.setFont(new Font("Arial", Font.PLAIN, 12));
25        g.drawString("Text in RED, Size 12", 100, 100);
26
27        // Text 2: Color BLUE, Size 18
28        g.setColor(Color.BLUE);
29        g.setFont(new Font("Arial", Font.BOLD, 18));
30        g.drawString("Text in BLUE, Size 18", 100, 150);
31
32        // Text 3: Color GREEN, Size 24
33        g.setColor(Color.GREEN);
34        g.setFont(new Font("Arial", Font.ITALIC, 24));
35        g.drawString("Text in GREEN, Size 24", 100, 200);
36
37        // Text 4: Color ORANGE, Size 30
38        g.setColor(Color.ORANGE);
39        g.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 30));
40        g.drawString("Text in ORANGE, Size 30", 100, 250);
41    }
42
43    public static void main(String[] args) {
44        SwingUtilities.invokeLater(() -> new DifferentText());
45    }
46 }
```

**Program 2: Displaying Different Lines**

```
1 import java.awt.*;
2 import javax.swing.*;
3
```

```
4 public class DifferentLines extends JFrame {
5     private final int WIDTH = 800;
6     private final int HEIGHT = 600;
7
8     public DifferentLines() {
9         setSize(WIDTH, HEIGHT);
10        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        setLocationRelativeTo(null);
12        setVisible(true);
13    }
14
15    @Override
16    public void paint(Graphics g) {
17        super.paint(g);
18        drawLines(g);
19    }
20
21    private void drawLines(Graphics g) {
22        // Line 1: Color RED, Thickness 1
23        g.setColor(Color.RED);
24        g.drawLine(100, 100, 300, 100);
25
26        // Line 2: Color BLUE, Thickness 2
27        g.setColor(Color.BLUE);
28        Graphics2D g2d = (Graphics2D) g;
29        g2d.setStroke(new BasicStroke(2));
30        g2d.drawLine(100, 150, 300, 150);
31
32        // Line 3: Color GREEN, Thickness 3
33        g.setColor(Color.GREEN);
34        g2d.setStroke(new BasicStroke(3));
35        g2d.drawLine(100, 200, 300, 200);
36
37        // Line 4: Color ORANGE, Thickness 4
38        g.setColor(Color.ORANGE);
39        g2d.setStroke(new BasicStroke(4));
40        g2d.drawLine(100, 250, 300, 250);
41    }
42
43    public static void main(String[] args) {
44        SwingUtilities.invokeLater(() -> new DifferentLines());
45    }
46 }
```

### Program 3: Displaying Different Shapes

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class DifferentShapes extends JFrame {
5     private final int WIDTH = 800;
6     private final int HEIGHT = 600;
7
8     public DifferentShapes() {
```

```
9      setSize(WIDTH, HEIGHT);
10     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11     setLocationRelativeTo(null);
12     setVisible(true);
13 }
14
15 @Override
16 public void paint(Graphics g) {
17     super.paint(g);
18     drawShapes(g);
19 }
20
21 private void drawShapes(Graphics g) {
22     // Rectangle: Color RED
23     g.setColor(Color.RED);
24     g.fillRect(50, 50, 100, 80);
25
26     // Circle: Color BLUE
27     g.setColor(Color.BLUE);
28     g.fillOval(200, 50, 100, 100);
29
30     // Oval: Color GREEN
31     g.setColor(Color.GREEN);
32     g.fillOval(350, 50, 150, 80);
33
34     // Polygon: Color ORANGE
35     g.setColor(Color.ORANGE);
36     int[] xPoints = {550, 600, 650};
37     int[] yPoints = {50, 120, 50};
38     g.fillPolygon(xPoints, yPoints, 3);
39 }
40
41 public static void main(String[] args) {
42     SwingUtilities.invokeLater(() -> new DifferentShapes());
43 }
44 }
```

#### Program 4: Drawing Arc

```
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class ArcDrawingExample extends JPanel {
5
6     @Override
7     protected void paintComponent(Graphics g) {
8         super.paintComponent(g);
9         drawArcs(g);
10    }
11
12    private void drawArcs(Graphics g) {
13        int width = getWidth();
14        int height = getHeight();
15    }
```

```
16     int xCenter = width / 2;
17     int yCenter = height / 2;
18     int arcRadius = Math.min(width, height) / 4; // Arc radius as 1/4th of the
        smaller dimension
19
20     int startAngle = 45; // Starting angle for the arcs
21     int arcAngle = 90; // Total angle of the arc (in degrees)
22
23     // Draw a simple arc
24     g.drawArc(xCenter - arcRadius, yCenter - arcRadius, arcRadius * 2,
        arcRadius * 2, startAngle, arcAngle);
25
26     // Draw an arc with a larger angle
27     g.drawArc(xCenter - arcRadius, yCenter - arcRadius, arcRadius * 2,
        arcRadius * 2, 120, 180);
28
29     // Draw a filled arc
30     g.fillArc(xCenter - arcRadius, yCenter - arcRadius, arcRadius * 2,
        arcRadius * 2, 270, 135);
31 }
32
33 public static void main(String[] args) {
34     JFrame frame = new JFrame("Arc Drawing Example");
35     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
36     frame.setSize(400, 400);
37     frame.add(new ArcDrawingExample());
38     frame.setVisible(true);
39 }
40 }
```

### Program 5: Drawing Different Arc

```
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class DrawArcs extends JFrame {
5
6     public DrawArcs() {
7         setTitle("Drawing Arcs in Java");
8         setSize(400, 400);
9         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        setLocationRelativeTo(null);
11        setVisible(true);
12    }
13
14    @Override
15    public void paint(Graphics g) {
16        super.paint(g);
17
18        // Draw a full circle (360 degrees) starting from 0 degrees
19        g.setColor(Color.BLUE);
20        g.drawArc(50, 50, 300, 300, 0, 360);
21
22        // Draw a 90-degree arc starting from 45 degrees
```

```
23     g.setColor(Color.RED);
24     g.drawArc(75, 75, 250, 250, 45, 90);
25
26     // Draw a 180-degree arc starting from 180 degrees
27     g.setColor(Color.GREEN);
28     g.drawArc(100, 100, 200, 200, 180, 180);
29
30     // Draw a 270-degree arc starting from 270 degrees
31     g.setColor(Color.ORANGE);
32     g.drawArc(125, 125, 150, 150, 270, 270);
33 }
34
35 public static void main(String[] args) {
36     new DrawArcs();
37 }
38 }
```

### Program 5: Filling Different Arc

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class DrawFilledArcs extends JFrame {
5
6      public DrawFilledArcs() {
7          setTitle("Drawing Filled Arcs in Java");
8          setSize(400, 400);
9          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10         setLocationRelativeTo(null);
11         setVisible(true);
12     }
13
14     @Override
15     public void paint(Graphics g) {
16         super.paint(g);
17
18         // Draw a full circle (360 degrees) starting from 0 degrees
19         g.setColor(Color.BLUE);
20         g.fillArc(50, 50, 300, 300, 0, 360);
21
22         // Draw a 90-degree arc starting from 45 degrees
23         g.setColor(Color.RED);
24         g.fillArc(75, 75, 250, 250, 45, 90);
25
26         // Draw a 180-degree arc starting from 180 degrees
27         g.setColor(Color.GREEN);
28         g.fillArc(100, 100, 200, 200, 180, 180);
29
30         // Draw a 270-degree arc starting from 270 degrees
31         g.setColor(Color.ORANGE);
32         g.fillArc(125, 125, 150, 150, 270, 270);
33     }
34
35     public static void main(String[] args) {
```

```
36     new DrawFilledArcs();
37 }
38 }
```

### Program 6: Drawing Different Polygons

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class DrawPolygons extends JFrame {
5
6      public DrawPolygons() {
7          setTitle("Drawing Polygons in Java");
8          setSize(400, 400);
9          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10         setLocationRelativeTo(null);
11         setVisible(true);
12     }
13
14     @Override
15     public void paint(Graphics g) {
16         super.paint(g);
17
18         // Draw a triangle
19         int[] xPointsTriangle = {100, 200, 150};
20         int[] yPointsTriangle = {200, 200, 100};
21         g.setColor(Color.BLUE);
22         g.drawPolygon(xPointsTriangle, yPointsTriangle, 3);
23
24         // Draw a rectangle
25         int[] xPointsRectangle = {100, 300, 300, 100};
26         int[] yPointsRectangle = {250, 250, 350, 350};
27         g.setColor(Color.RED);
28         g.drawPolygon(xPointsRectangle, yPointsRectangle, 4);
29
30         // Draw a pentagon
31         int[] xPointsPentagon = {200, 250, 300, 250, 200};
32         int[] yPointsPentagon = {100, 50, 100, 150, 150};
33         g.setColor(Color.GREEN);
34         g.drawPolygon(xPointsPentagon, yPointsPentagon, 5);
35
36         // Draw a hexagon
37         int[] xPointsHexagon = {150, 200, 250, 250, 200, 150};
38         int[] yPointsHexagon = {300, 300, 250, 200, 200, 250};
39         g.setColor(Color.ORANGE);
40         g.drawPolygon(xPointsHexagon, yPointsHexagon, 6);
41     }
42
43     public static void main(String[] args) {
44         new DrawPolygons();
45     }
46 }
```

### Program 7: DDA Alogrithm

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class DDAAAlgorithm extends JFrame {
5     private final int WIDTH = 800;
6     private final int HEIGHT = 600;
7
8     public DDAAAlgorithm() {
9         setTitle("DDA Algorithm in Java");
10        setSize(WIDTH, HEIGHT);
11        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        setLocationRelativeTo(null);
13        setVisible(true);
14    }
15
16    @Override
17    public void paint(Graphics g) {
18        super.paint(g);
19        ddaAlgorithm(g, 0, 0, 500, 400); // Replace these coordinates with your
20        desired endpoints
21    }
22
23    private void ddaAlgorithm(Graphics g, int x1, int y1, int x2, int y2) {
24        int dx = x2 - x1;
25        int dy = y2 - y1;
26        int steps = Math.max(Math.abs(dx), Math.abs(dy));
27
28        float xIncrement = (float) dx / steps;
29        float yIncrement = (float) dy / steps;
30
31        float x = x1;
32        float y = y1;
33
34        for (int i = 0; i <= steps; i++) {
35            int roundX = Math.round(x);
36            int roundY = Math.round(y);
37            g.drawLine(roundX, roundY, roundX, roundY);
38            x += xIncrement;
39            y += yIncrement;
40        }
41    }
42
43    public static void main(String[] args) {
44        SwingUtilities.invokeLater(() -> new DDAAAlgorithm());
45    }
46 }
```

### Program 8: Bresenham Line Drawing Algorithm

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class BresenhamAlgorithm extends JFrame {
```



```
5 private final int WIDTH = 800;
6 private final int HEIGHT = 600;
7
8 public BresenhamAlgorithm() {
9     setTitle("Bresenham Line Drawing Algorithm in Java");
10    setSize(WIDTH, HEIGHT);
11    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12    setLocationRelativeTo(null);
13    setVisible(true);
14 }
15
16 @Override
17 public void paint(Graphics g) {
18     super.paint(g);
19     bresenhamAlgorithm(g, 100, 100, 500, 400); // Replace these coordinates
20     // with your desired endpoints
21 }
22
23 private void bresenhamAlgorithm(Graphics g, int x1, int y1, int x2, int y2) {
24     int dx = Math.abs(x2 - x1);
25     int dy = Math.abs(y2 - y1);
26     int signX = x1 < x2 ? 1 : -1;
27     int signY = y1 < y2 ? 1 : -1;
28
29     int x = x1;
30     int y = y1;
31
32     boolean interchange = false;
33     if (dy > dx) {
34         // Swap dx and dy to ensure we are always iterating over the major axis
35         int temp = dx;
36         dx = dy;
37         dy = temp;
38         interchange = true;
39     }
40
41     int decision = 2 * dy - dx;
42
43     for (int i = 0; i <= dx; i++) {
44         // Draw the pixel at (x, y)
45         g.drawLine(x, y, x, y);
46
47         while (decision >= 0) {
48             if (interchange)
49                 x += signX;
50             else
51                 y += signY;
52
53             decision -= 2 * dx;
54         }
55
56         if (interchange)
57             y += signY;
```

```

57         else
58             x += signX;
59
60         decision += 2 * dy;
61     }
62 }
63
64 public static void main(String[] args) {
65     SwingUtilities.invokeLater(() -> new BresenhamAlgorithm());
66 }
67 }

```

### Program 8: Midpoint Circle Drawing Algorithm

```

1  import java.awt.*;
2  import javax.swing.*;
3
4  public class MidpointCircleAlgorithm extends JFrame {
5      private final int WIDTH = 800;
6      private final int HEIGHT = 600;
7
8      public MidpointCircleAlgorithm() {
9          setTitle("Midpoint Circle Drawing Algorithm in Java");
10         setSize(WIDTH, HEIGHT);
11         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12         setLocationRelativeTo(null);
13         setVisible(true);
14     }
15
16     @Override
17     public void paint(Graphics g) {
18         super.paint(g);
19         int centerX = WIDTH / 2;    // Replace with desired center X-coordinate
20         int centerY = HEIGHT / 2;   // Replace with desired center Y-coordinate
21
22         int radius = 100; // Change the radius to draw a different circle
23
24
25         g.setColor(Color.RED);
26         g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
27         g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
28
29         midpointCircleAlgorithm(g, centerX, centerY, radius);
30     }
31
32     private void plotCirclePixel(Graphics g, int x, int y, int centerX, int
        centerY) {
33         // Draw the pixels at the corresponding positions
34         g.setColor(Color.BLACK);
35         g.drawLine(centerX + x, centerY + y, centerX + x, centerY + y);
36         g.drawLine(centerX - x, centerY + y, centerX - x, centerY + y);
37         g.drawLine(centerX + x, centerY - y, centerX + x, centerY - y);
38         g.drawLine(centerX - x, centerY - y, centerX - x, centerY - y);
39         g.drawLine(centerX + y, centerY + x, centerX + y, centerY + x);

```

```

40     g.drawLine(centerX - y, centerY + x, centerX - y, centerY + x);
41     g.drawLine(centerX + y, centerY - x, centerX + y, centerY - x);
42     g.drawLine(centerX - y, centerY - x, centerX - y, centerY - x);
43 }
44
45 private void midpointCircleAlgorithm(Graphics g, int centerX, int centerY, int
    radius) {
46     int x = 0;
47     int y = radius;
48     int decision = 1 - radius;
49
50     plotCirclePixel(g, x, y, centerX, centerY);
51
52     while (y > x) {
53         if (decision < 0) {
54             decision += 2 * x + 1;
55         } else {
56             decision += 2 * (x - y) + 1;
57             y--;
58         }
59         x++;
60         plotCirclePixel(g, x, y, centerX, centerY);
61     }
62 }
63
64 public static void main(String[] args) {
65     SwingUtilities.invokeLater(() -> new MidpointCircleAlgorithm());
66 }
67 }

```

### Program 9: 2D Translation Algorithm Example

```

1  import javax.swing.*;
2  import java.awt.*;
3
4  public class Translation2D extends JPanel {
5      private void drawTranslatedShape(Graphics2D g2d, int[] xPoints, int[] yPoints,
        int dx, int dy) {
6          int nPoints = xPoints.length;
7
8          // Translate the shape's coordinates by dx and dy
9          int[] translatedXPoints = new int[nPoints];
10         int[] translatedYPoints = new int[nPoints];
11         for (int i = 0; i < nPoints; i++) {
12             translatedXPoints[i] = xPoints[i] + dx;
13             translatedYPoints[i] = yPoints[i] + dy;
14         }
15
16         // Draw the original shape
17         g2d.setColor(Color.BLACK);
18         g2d.drawPolygon(xPoints, yPoints, nPoints);
19
20         // Draw the translated shape in red
21         g2d.setColor(Color.RED);

```

```

22     g2d.drawPolygon(translatedXPoints, translatedYPoints, nPoints);
23 }
24
25 @Override
26 protected void paintComponent(Graphics g) {
27     super.paintComponent(g);
28
29     Graphics2D g2d = (Graphics2D) g;
30
31     int[] xPoints = {100, 150, 200};
32     int[] yPoints = {200, 100, 200};
33
34     // Call drawTranslatedShape to draw the translated shape with dx and dy
35     drawTranslatedShape(g2d, xPoints, yPoints, 50, 50);
36 }
37
38 public static void main(String[] args) {
39     SwingUtilities.invokeLater(() -> {
40         JFrame frame = new JFrame("2D Translation");
41         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
42         frame.setSize(400, 400);
43         frame.setContentPane(new Translation2D());
44         frame.setVisible(true);
45     });
46 }
47 }

```

**Program 10:** 2D Translation Using Inbuilt Java Affine Transformation Example

```

1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.geom.AffineTransform;
4
5  public class TranslationExample extends JFrame {
6      private final int WIDTH = 800;
7      private final int HEIGHT = 600;
8
9      public TranslationExample() {
10         setTitle("Affine Transformation in Java");
11         setSize(WIDTH, HEIGHT);
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         setLocationRelativeTo(null);
14         setVisible(true);
15     }
16
17     @Override
18     public void paint(Graphics g) {
19         super.paint(g);
20         Graphics2D g2d = (Graphics2D) g;
21
22         int x = 100;
23         int y = 100;
24         int dx = 50;
25         int dy = 100;

```

```

26
27     // Original rectangle
28     g2d.setColor(Color.RED);
29     g2d.fillRect(x, y, 100, 80);
30
31     // Translated rectangle
32     AffineTransform translation = AffineTransform.getTranslateInstance(dx, dy);
33     g2d.setColor(Color.BLUE);
34     g2d.fill(translation.createTransformedShape(new Rectangle(x, y, 100, 80)));
35 }
36
37 public static void main(String[] args) {
38     SwingUtilities.invokeLater(() -> new TranslationExample());
39 }
40 }

```

### Program 10: 2D Rotation Example

```

1  import javax.swing.*;
2  import java.awt.*;
3
4  public class Rotation2DUsingFormula extends JPanel {
5      private void drawRotatedShape(Graphics2D g2d, int[] xPoints, int[] yPoints,
6          double angleDegrees, int centerX, int centerY) {
7
8          // Convert the angle to radians
9          double angleRadians = Math.toRadians(angleDegrees);
10
11         // Apply the rotation formula to each point
12         int[] rotatedXPoints = new int[nPoints];
13         int[] rotatedYPoints = new int[nPoints];
14         for (int i = 0; i < nPoints; i++) {
15             int x = xPoints[i];
16             int y = yPoints[i];
17             rotatedXPoints[i] = (int) (centerX + (x - centerX) *
18                 Math.cos(angleRadians) - (y - centerY) * Math.sin(angleRadians));
19             rotatedYPoints[i] = (int) (centerY + (x - centerX) *
20                 Math.sin(angleRadians) + (y - centerY) * Math.cos(angleRadians));
21         }
22
23         // Draw the original shape
24         g2d.setColor(Color.BLACK);
25         g2d.drawPolygon(xPoints, yPoints, nPoints);
26
27         // Draw the rotated shape in red
28         g2d.setColor(Color.RED);
29         g2d.drawPolygon(rotatedXPoints, rotatedYPoints, nPoints);
30     }
31
32     @Override
33     protected void paintComponent(Graphics g) {
34         super.paintComponent(g);
35     }
36 }

```

```

34     Graphics2D g2d = (Graphics2D) g;
35
36     int[] xPoints = {100, 150, 200};
37     int[] yPoints = {200, 100, 200};
38     double angleDegrees = 45; // Rotation angle in degrees
39     int centerX = 150; // Center of rotation x-coordinate
40     int centerY = 150; // Center of rotation y-coordinate
41
42     // Call drawRotatedShape to draw the rotated shape
43     drawRotatedShape(g2d, xPoints, yPoints, angleDegrees, centerX, centerY);
44 }
45
46 public static void main(String[] args) {
47     SwingUtilities.invokeLater(() -> {
48         JFrame frame = new JFrame("2D Rotation using Formula");
49         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50         frame.setSize(400, 400);
51         frame.setContentPane(new Rotation2DUsingFormula());
52         frame.setVisible(true);
53     });
54 }
55 }

```

### Program 11: 2D Scaling About an Origin

```

1  import javax.swing.*;
2  import java.awt.*;
3
4  public class Scaling2DUsingFormula extends JPanel {
5      private void drawScaledShape(Graphics2D g2d, int[] xPoints, int[] yPoints,
6          double Sx, double Sy) {
7          int nPoints = xPoints.length;
8
9
10         // Apply the rotation formula to each point
11         int[] scaledXPoints = new int[nPoints];
12         int[] scaledYPoints = new int[nPoints];
13         for (int i = 0; i < nPoints; i++) {
14             int x = xPoints[i];
15             int y = yPoints[i];
16             scaledXPoints[i] = (int) (x * Sx);
17             scaledYPoints[i] = (int) (y * Sy);
18         }
19
20         // Draw the original shape
21         g2d.setColor(Color.BLACK);
22         g2d.drawPolygon(xPoints, yPoints, nPoints);
23
24         // Draw the rotated shape in red
25         g2d.setColor(Color.RED);
26         g2d.drawPolygon(scaledXPoints, scaledYPoints, nPoints);
27     }
28 }

```

```

29  @Override
30  protected void paintComponent(Graphics g) {
31      super.paintComponent(g);
32
33      Graphics2D g2d = (Graphics2D) g;
34
35      int[] xPoints = {100, 150, 200};
36      int[] yPoints = {200, 100, 200};
37
38      double Sx = 1.5; // Scaling factor in x direction
39      double Sy = 1.5; // Scaling factor in y direction
40
41      // Call drawRotatedShape to draw the rotated shape
42      drawScaledShape(g2d, xPoints, yPoints, Sx, Sy);
43  }
44
45  public static void main(String[] args) {
46      SwingUtilities.invokeLater(() -> {
47          JFrame frame = new JFrame("2D Scaling about an origin using Formula");
48          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
49          frame.setSize(800, 600);
50          frame.setContentPane(new Scaling2DUsingFormula());
51          frame.setVisible(true);
52      });
53  }
54  }

```

**Program 12:** 2D Scaling About an a fixed point

```

1  import javax.swing.*;
2  import java.awt.*;
3
4  public class Scaling2DFixedPoint extends JPanel {
5      private void drawScaledShape(Graphics2D g2d, int[] xPoints, int[] yPoints,
6          double Sx, double Sy, int xf, int yf) {
7
8
9
10         // Apply the rotation formula to each point
11         int[] scaledXPoints = new int[nPoints];
12         int[] scaledYPoints = new int[nPoints];
13         for (int i = 0; i < nPoints; i++) {
14             int x = xPoints[i];
15             int y = yPoints[i];
16             scaledXPoints[i] = (int) (xf + (x - xf) * Sx);
17             scaledYPoints[i] = (int) (yf + (y - yf) * Sy);
18         }
19
20         // Draw the original shape
21         g2d.setColor(Color.BLACK);
22         g2d.drawPolygon(xPoints, yPoints, nPoints);
23
24         // Draw the rotated shape in red

```

```

25     g2d.setColor(Color.RED);
26     g2d.drawPolygon(scaledXPoints, scaledYPoints, nPoints);
27 }
28
29 @Override
30 protected void paintComponent(Graphics g) {
31     super.paintComponent(g);
32
33     Graphics2D g2d = (Graphics2D) g;
34
35     int[] xPoints = {100, 150, 200};
36     int[] yPoints = {200, 100, 200};
37
38     double Sx = 1.5; // Scaling factor in x direction
39     double Sy = 1.5; // Scaling factor in y direction
40
41     int xf = 150; //fixed point reference
42     int yf = 150; //fixed point reference
43
44     // Call drawRotatedShape to draw the rotated shape
45     drawScaledShape(g2d, xPoints, yPoints, Sx, Sy, xf, yf);
46 }
47
48 public static void main(String[] args) {
49     SwingUtilities.invokeLater(() -> {
50         JFrame frame = new JFrame("2D Scaling about an origin using Formula");
51         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
52         frame.setSize(800, 600);
53         frame.setContentPane(new Scaling2DFixedPoint());
54         frame.setVisible(true);
55     });
56 }
57 }

```

### Program 13: X-direction Shear

```

1  import java.awt.Graphics;
2  import javax.swing.JFrame;
3  import javax.swing.JPanel;
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class XShearingRectangle extends JPanel {
8
9      private final int WIDTH = 800;
10     private final int HEIGHT = 600;
11
12     private int[] originalX = {50, 250, 250, 50}; // x-coordinates of the original
13     rectangle
14     private int[] originalY = {50, 50, 150, 150}; // y-coordinates of the original
15     rectangle
16     private int shearingFactorX = 2; // Shearing factor along X-axis
17
18     @Override

```



```

17     protected void paintComponent(Graphics g) {
18         super.paintComponent(g);
19
20         g.setColor(Color.RED);
21         g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
22         g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
23
24         // Draw the original rectangle
25         g.setColor(Color.BLACK);
26         g.drawPolygon(originalX, originalY, 4);
27
28         // Shearing the rectangle along X-axis
29         int[] shearedX = new int[4];
30         int[] shearedY = new int[4];
31         for (int i = 0; i < 4; i++) {
32             shearedX[i] = originalX[i] + shearingFactorX * originalY[i];
33             shearedY[i] = originalY[i];
34         }
35
36         // Draw the sheared rectangle
37         g.setColor(Color.BLUE);
38         g.drawPolygon(shearedX, shearedY, 4);
39     }
40
41     public static void main(String[] args) {
42         JFrame frame = new JFrame("Shearing Rectangle");
43         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44         frame.setSize(800, 600);
45         frame.add(new XShearingRectangle());
46         frame.setVisible(true);
47     }
48 }

```

#### Program 14: Y-direction Shear

```

1  import java.awt.Graphics;
2  import javax.swing.JFrame;
3  import javax.swing.JPanel;
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class YShearingRectangle extends JPanel {
8
9      private final int WIDTH = 800;
10     private final int HEIGHT = 600;
11
12     private int[] originalX = {50, 250, 250, 50}; // x-coordinates of the original
13         rectangle
14     private int[] originalY = {50, 50, 150, 150}; // y-coordinates of the original
15         rectangle
16     private int shearingFactorY = 2; // Shearing factor along Y-axis
17
18     @Override
19     protected void paintComponent(Graphics g) {

```

```

18     super.paintComponent(g);
19
20     g.setColor(Color.RED);
21     g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
22     g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
23
24     // Draw the original rectangle
25     g.setColor(Color.BLACK);
26     g.drawPolygon(originalX, originalY, 4);
27
28     // Shearing the rectangle along X-axis
29     int[] shearedX = new int[4];
30     int[] shearedY = new int[4];
31     for (int i = 0; i < 4; i++) {
32         shearedX[i] = originalX[i] ;
33         shearedY[i] = originalX[i] * shearingFactorY + originalY[i];
34     }
35
36     // Draw the sheared rectangle
37     g.setColor(Color.BLUE);
38     g.drawPolygon(shearedX, shearedY, 4);
39 }
40
41 public static void main(String[] args) {
42     JFrame frame = new JFrame("Y-direction Shearing");
43     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44     frame.setSize(800, 600);
45     frame.add(new YShearingRectangle());
46     frame.setVisible(true);
47 }
48 }

```

### Program 15: Reflection about x-axis

```

1  import java.awt.Graphics;
2  import javax.swing.JFrame;
3  import javax.swing.JPanel;
4  import java.awt.*;
5
6  public class ReflectTriangleXAxis extends JPanel {
7
8      private final int WIDTH = 800;
9      private final int HEIGHT = 600;
10
11     private int[] originalX = {100, 200, 150}; // x-coordinates of the original
12         triangle
13     private int[] originalY = {100, 100, 50}; // y-coordinates of the original
14         triangle
15
16     @Override
17     protected void paintComponent(Graphics g) {
18         super.paintComponent(g);
19         g.setColor(Color.RED);
20         g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);

```

```

19 g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
20
21 // Draw the original triangle
22 g.setColor(Color.BLACK);
23 g.drawPolygon(originalX, originalY, 3);
24
25 // Reflecting the triangle along X-axis
26 int[] reflectedX = new int[3];
27 int[] reflectedY = new int[3];
28 for (int i = 0; i < 3; i++) {
29     reflectedX[i] = originalX[i];
30     reflectedY[i] = getHeight() - originalY[i];
31 }
32
33 // Draw the reflected triangle
34 g.setColor(Color.BLUE);
35 g.drawPolygon(reflectedX, reflectedY, 3);
36 }
37
38 public static void main(String[] args) {
39     JFrame frame = new JFrame("Reflect Triangle about X-axis");
40     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
41     frame.setSize(800, 600);
42     frame.add(new ReflectTriangleXAxis());
43     frame.setVisible(true);
44 }
45 }

```

#### Program 16: Reflection about y-axis

```

1 import java.awt.Graphics;
2 import javax.swing.JFrame;
3 import javax.swing.JPanel;
4 import java.awt.*;
5
6 public class ReflectTriangleYAxis extends JPanel {
7
8     private final int WIDTH = 800;
9     private final int HEIGHT = 600;
10
11     private int[] originalX = {100, 200, 150}; // x-coordinates of the original
12         triangle
13     private int[] originalY = {100, 100, 50}; // y-coordinates of the original
14         triangle
15
16     @Override
17     protected void paintComponent(Graphics g) {
18         super.paintComponent(g);
19         g.setColor(Color.RED);
20         g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
21         g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
22
23         // Draw the original triangle
24         g.setColor(Color.BLACK);

```

```

23     g.drawPolygon(originalX, originalY, 3);
24
25     // Reflecting the triangle along X-axis
26     int[] reflectedX = new int[3];
27     int[] reflectedY = new int[3];
28     for (int i = 0; i < 3; i++) {
29         reflectedX[i] = getWidth() - originalX[i];
30         reflectedY[i] = originalY[i];
31     }
32
33     // Draw the reflected triangle
34     g.setColor(Color.BLUE);
35     g.drawPolygon(reflectedX, reflectedY, 3);
36 }
37
38 public static void main(String[] args) {
39     JFrame frame = new JFrame("Reflect Triangle about Y-axis");
40     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
41     frame.setSize(800, 600);
42     frame.add(new ReflectTriangleYAxis());
43     frame.setVisible(true);
44 }
45 }

```

### Program 17: Drawing a Triangle with center as origin

```

1  import java.awt.Dimension;
2  import java.awt.Graphics;
3  import javax.swing.JFrame;
4  import javax.swing.JPanel;
5  import java.awt.*;
6
7  public class TriangleWithCenterOrigin extends JPanel {
8
9      @Override
10     protected void paintComponent(Graphics g) {
11         super.paintComponent(g);
12
13         // Get the dimensions of the display
14         Dimension screenSize = getSize();
15         int centerX = screenSize.width / 2;
16         int centerY = screenSize.height / 2;
17
18         int WIDTH = screenSize.width;
19         int HEIGHT = screenSize.height;
20
21         g.setColor(Color.RED);
22         g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
23         g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
24
25         // Define the vertices of the triangle with respect to the center
26         int[] verticesX = {centerX, centerX + 100, centerX - 100};
27         int[] verticesY = {centerY - 100, centerY + 50, centerY + 50};
28

```

```

29     // Draw the triangle
30     g.setColor(Color.BLACK);
31     g.drawPolygon(verticesX, verticesY, 3);
32 }
33
34 public static void main(String[] args) {
35     JFrame frame = new JFrame("Triangle with Center Origin");
36     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
37     frame.setSize(400, 400);
38     frame.add(new TriangleWithCenterOrigin());
39     frame.setVisible(true);
40 }
41 }

```

### Program 18: Reflection of a Triangle about x-axis( $y=0$ )

```

1  import java.awt.Dimension;
2  import java.awt.Graphics;
3  import javax.swing.JFrame;
4  import javax.swing.JPanel;
5  import java.awt.*;
6
7  public class ReflectionTriangleXaxis extends JPanel {
8      private int[] originalX = {50, 150, 100}; // x-coordinates of the original
          triangle
9      private int[] originalY = {100, 100, 50}; // y-coordinates of the original
          triangle
10
11     @Override
12     protected void paintComponent(Graphics g) {
13         super.paintComponent(g);
14
15         // Get the dimensions of the display
16         Dimension screenSize = getSize();
17         int centerX = screenSize.width / 2;
18         int centerY = screenSize.height / 2;
19
20         int WIDTH = screenSize.width;
21         int HEIGHT = screenSize.height;
22
23         g.setColor(Color.RED);
24         g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
25         g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
26
27         // Draw the original triangle with respect to the center
28         g.setColor(Color.BLACK);
29         drawTriangle(g, originalX, originalY, centerX, centerY);
30
31
32
33         // Reflecting the triangle along X-axis
34         int[] reflectedX = new int[3];
35         int[] reflectedY = new int[3];
36         for (int i = 0; i < 3; i++) {

```

```

37         reflectedX[i] = originalX[i];
38         reflectedY[i] = - originalY[i];
39     }
40
41     // Draw the reflected triangle
42     g.setColor(Color.BLUE);
43     drawTriangle(g, reflectedX, reflectedY, centerX, centerY);
44
45 }
46
47 private void drawTriangle(Graphics g, int[] x, int[] y, int centerX, int centerY)
48 {
49     // Shift the triangle to draw it with respect to the center
50     int[] shiftedX = new int[3];
51     int[] shiftedY = new int[3];
52     for (int i = 0; i < 3; i++) {
53         shiftedX[i] = centerX + x[i];
54         shiftedY[i] = centerY - y[i];
55     }
56
57     // Draw the triangle
58     g.drawPolygon(shiftedX, shiftedY, 3);
59 }
60
61 public static void main(String[] args) {
62     JFrame frame = new JFrame("Reflection of a triangle about x-axis");
63     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
64     frame.setSize(400, 400);
65     frame.add(new ReflectionTriangleXaxis());
66     frame.setVisible(true);
67 }

```

### Program 19: Reflection of a Triangle about y-axis( $x=0$ )

```

1  import java.awt.Dimension;
2  import java.awt.Graphics;
3  import javax.swing.JFrame;
4  import javax.swing.JPanel;
5  import java.awt.*;
6
7  public class ReflectionTriangleYaxis extends JPanel {
8      private int[] originalX = {50, 150, 100}; // x-coordinates of the original
9          triangle
10
11     private int[] originalY = {100, 100, 50}; // y-coordinates of the original
12         triangle
13
14     @Override
15     protected void paintComponent(Graphics g) {
16         super.paintComponent(g);
17
18         // Get the dimensions of the display
19         Dimension screenSize = getSize();
20         int centerX = screenSize.width / 2;

```

```
18     int centerY = screenSize.height / 2;
19
20     int WIDTH = screenSize.width;
21     int HEIGHT = screenSize.height;
22
23     g.setColor(Color.RED);
24     g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
25     g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
26
27     // Draw the original triangle with respect to the center
28     g.setColor(Color.BLACK);
29     drawTriangle(g, originalX, originalY, centerX, centerY);
30
31
32
33     // Reflecting the triangle along Y-axis
34     int[] reflectedX = new int[3];
35     int[] reflectedY = new int[3];
36     for (int i = 0; i < 3; i++) {
37         reflectedX[i] = - originalX[i];
38         reflectedY[i] = originalY[i];
39     }
40
41     // Draw the reflected triangle
42     g.setColor(Color.BLUE);
43     drawTriangle(g, reflectedX, reflectedY, centerX, centerY);
44
45 }
46
47 private void drawTriangle(Graphics g, int[] x, int[] y, int centerX, int centerY)
48 {
49     // Shift the triangle to draw it with respect to the center
50     int[] shiftedX = new int[3];
51     int[] shiftedY = new int[3];
52     for (int i = 0; i < 3; i++) {
53         shiftedX[i] = centerX + x[i];
54         shiftedY[i] = centerY - y[i];
55     }
56
57     // Draw the triangle
58     g.drawPolygon(shiftedX, shiftedY, 3);
59 }
60
61 public static void main(String[] args) {
62     JFrame frame = new JFrame("Reflection of a triangle about y-axis");
63     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
64     frame.setSize(800, 600);
65     frame.add(new ReflectionTriangleYaxis());
66     frame.setVisible(true);
67 }
```

**Program 20:** Reflection of a Triangle about an origin

```
1 import java.awt.Dimension;
2 import java.awt.Graphics;
3 import javax.swing.JFrame;
4 import javax.swing.JPanel;
5 import java.awt.*;
6
7 public class ReflectionTriangleAboutOrigin extends JPanel {
8     private int[] originalX = {50, 150, 100}; // x-coordinates of the original
        triangle
9     private int[] originalY = {100, 100, 50}; // y-coordinates of the original
        triangle
10
11     @Override
12     protected void paintComponent(Graphics g) {
13         super.paintComponent(g);
14
15         // Get the dimensions of the display
16         Dimension screenSize = getSize();
17         int centerX = screenSize.width / 2;
18         int centerY = screenSize.height / 2;
19
20         int WIDTH = screenSize.width;
21         int HEIGHT = screenSize.height;
22
23         g.setColor(Color.RED);
24         g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
25         g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
26
27         // Draw the original triangle with respect to the center
28         g.setColor(Color.BLACK);
29         drawTriangle(g, originalX, originalY, centerX, centerY);
30
31         // Reflecting the triangle about an origin
32         int[] reflectedX = new int[3];
33         int[] reflectedY = new int[3];
34         for (int i = 0; i < 3; i++) {
35             reflectedX[i] = - originalX[i];
36             reflectedY[i] = - originalY[i];
37         }
38
39         // Draw the reflected triangle
40         g.setColor(Color.BLUE);
41         drawTriangle(g, reflectedX, reflectedY, centerX, centerY);
42     }
43
44     private void drawTriangle(Graphics g, int[] x, int[] y, int centerX, int centerY)
        {
45         // Shift the triangle to draw it with respect to the center
46         int[] shiftedX = new int[3];
47         int[] shiftedY = new int[3];
48         for (int i = 0; i < 3; i++) {
49             shiftedX[i] = centerX + x[i];
50             shiftedY[i] = centerY - y[i];
```



```

51     }
52
53     // Draw the triangle
54     g.drawPolygon(shiftedX, shiftedY, 3);
55 }
56
57 public static void main(String[] args) {
58     JFrame frame = new JFrame("Reflection of a triangle about about an origin");
59     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
60     frame.setSize(400, 400);
61     frame.add(new ReflectionTriangleAboutOrigin());
62     frame.setVisible(true);
63 }
64 }

```

### Program 21: Reflection of a Triangle with respect to $y=x$

```

1  import java.awt.Dimension;
2  import java.awt.Graphics;
3  import javax.swing.JFrame;
4  import javax.swing.JPanel;
5  import java.awt.*;
6
7  public class ReflectionTriangleYequalX extends JPanel {
8      private int[] originalX = {50, 120, 50}; // x-coordinates of the original triangle
9      private int[] originalY = {150, 150, 200}; // y-coordinates of the original
10         triangle
11
12     @Override
13     protected void paintComponent(Graphics g) {
14         super.paintComponent(g);
15
16         // Get the dimensions of the display
17         Dimension screenSize = getSize();
18         int centerX = screenSize.width / 2;
19         int centerY = screenSize.height / 2;
20
21         int WIDTH = screenSize.width;
22         int HEIGHT = screenSize.height;
23
24         g.setColor(Color.RED);
25         g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
26         g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
27
28         // Lie Color GREEN, Thickness 1
29         g.setColor(Color.GREEN);
30         Graphics2D g2d = (Graphics2D) g;
31         g2d.setStroke(new BasicStroke(1));
32         g2d.drawLine(WIDTH, 0, 0, HEIGHT);
33         g.drawString("y = x", 650, 50);
34
35         //draw an original polygon with screen coordinates
36         g.setColor(Color.BLACK);
37         g.drawPolygon(originalX, originalY,3);

```

```

37     g.drawString("An original Rectangle", 50, 140);
38
39     // Draw the original triangle with respect to the center
40     g.setColor(Color.RED);
41     drawTriangle(g, originalX, originalY, centerX, centerY);
42
43     // Reflecting the triangle with respect to y = x
44     int[] reflectedX = new int[3];
45     int[] reflectedY = new int[3];
46     for (int i = 0; i < 3; i++) {
47         reflectedX[i] = originalY[i];
48         reflectedY[i] = originalX[i];
49     }
50
51     // Draw the reflected triangle with respect to the center
52     g.setColor(Color.BLUE);
53     drawTriangle(g, reflectedX, reflectedY, centerX, centerY);
54
55 }
56
57 private void drawTriangle(Graphics g, int[] x, int[] y, int centerX, int
58     centerY) {
59     // Shift the triangle to draw it with respect to the center
60     int[] shiftedX = new int[3];
61     int[] shiftedY = new int[3];
62     for (int i = 0; i < 3; i++) {
63         shiftedX[i] = centerX + x[i];
64         shiftedY[i] = centerY - y[i];
65     }
66
67     // Draw the triangle considering origin at center of a screen
68
69     g.drawPolygon(shiftedX, shiftedY, 3);
70
71 public static void main(String[] args) {
72     JFrame frame = new JFrame("Reflection of a triangle w.r.t to y = x");
73     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
74     frame.setSize(800, 600);
75     frame.add(new ReflectionTriangleYequalX());
76     frame.setVisible(true);
77 }
78 }

```

**Program 22:** Reflection of a Triangle with respect to  $y = -x$

```

1  import java.awt.Dimension;
2  import java.awt.Graphics;
3  import javax.swing.JFrame;
4  import javax.swing.JPanel;
5  import java.awt.*;
6
7  public class ReflectionTriangleYEqualsNegativeX extends JPanel {
8      private int[] originalX = {50, 150, 100}; // x-coordinates of the original

```

```

    triangle
9   private int[] originalY = {100, 100, 50}; // y-coordinates of the original
    triangle
10
11   @Override
12   protected void paintComponent(Graphics g) {
13       super.paintComponent(g);
14
15       // Get the dimensions of the display
16       Dimension screenSize = getSize();
17       int centerX = screenSize.width / 2;
18       int centerY = screenSize.height / 2;
19
20       int WIDTH = screenSize.width;
21       int HEIGHT = screenSize.height;
22
23       g.setColor(Color.RED);
24       g.drawLine(WIDTH/2, 0, WIDTH/2, HEIGHT);
25       g.drawLine(0, HEIGHT/2, WIDTH, HEIGHT/2 );
26
27       // Lie Color GREEN, Thickness 1
28       g.setColor(Color.GREEN);
29       Graphics2D g2d = (Graphics2D) g;
30       g2d.setStroke(new BasicStroke(1));
31       g2d.drawLine(0, 0, WIDTH, HEIGHT);
32       g.drawString("y = -x", 180, 160);
33
34       // Draw the original triangle with respect to the center
35       g.setColor(Color.BLACK);
36       drawTriangle(g, originalX, originalY, centerX, centerY);
37
38       // Reflecting the triangle about an origin
39       int[] reflectedX = new int[3];
40       int[] reflectedY = new int[3];
41       for (int i = 0; i < 3; i++) {
42           reflectedX[i] = - originalY[i];
43           reflectedY[i] = - originalX[i];
44       }
45
46       // Draw the reflected triangle
47       g.setColor(Color.BLUE);
48       drawTriangle(g, reflectedX, reflectedY, centerX, centerY);
49   }
50
51   private void drawTriangle(Graphics g, int[] x, int[] y, int centerX, int centerY)
52   {
53       // Shift the triangle to draw it with respect to the center
54       int[] shiftedX = new int[3];
55       int[] shiftedY = new int[3];
56       for (int i = 0; i < 3; i++) {
57           shiftedX[i] = centerX + x[i];
58           shiftedY[i] = centerY - y[i];
59       }

```

```
59     // Draw the triangle
60     g.drawPolygon(shiftedX, shiftedY, 3);
61 }
62
63 public static void main(String[] args) {
64     JFrame frame = new JFrame("Reflection of a triangle w.r.t to y = - x");
65     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
66     frame.setSize(800, 600);
67     frame.add(new ReflectionTriangleYEqualsNegativeX());
68     frame.setVisible(true);
69 }
70 }
```

### Program 23: Rotate a text

```
1  import java.awt.Font;
2  import java.awt.Color;
3  import java.awt.FontMetrics;
4  import java.awt.Graphics;
5  import java.awt.Graphics2D;
6  import javax.swing.JFrame;
7  import javax.swing.JPanel;
8
9  public class RotateTextInJava extends JPanel {
10
11     @Override
12     protected void paintComponent(Graphics g) {
13         super.paintComponent(g);
14
15         // Set the X and Y coordinates where you want to print the text
16         int xCoordinate = 100;
17         int yCoordinate = 150;
18
19         // Set the text to be printed
20         String text = "Hello, Java!";
21
22
23
24         // Set the rotation angle in degrees
25         double rotationAngleDegrees = 45;
26
27         // Create a new Font with the desired font family, style, and size
28         Font font = new Font("Arial", Font.BOLD, 26);
29
30         // Cast Graphics object to Graphics2D for additional functionality
31         Graphics2D g2d = (Graphics2D) g;
32
33         // Set the new Font
34         g2d.setFont(font);
35
36         //set color of text
37
38         g2d.setColor(Color.RED);
39     }
```

```
40 // Set the rotation angle
41 double rotationAngleRadians = Math.toRadians(rotationAngleDegrees);
42 g2d.rotate(rotationAngleRadians, xCoordinate, yCoordinate);
43
44 // Draw the text at the specified rotated coordinates
45 g2d.drawString(text, xCoordinate, yCoordinate);
46 }
47
48 public static void main(String[] args) {
49     JFrame frame = new JFrame("Rotate Text in Java");
50     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
51     frame.setSize(600, 400);
52     frame.add(new RotateTextInJava());
53     frame.setVisible(true);
54 }
55 }
```