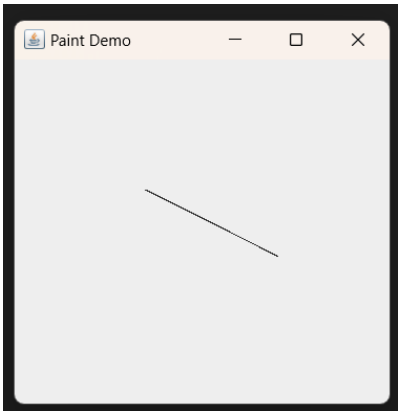
Lab Assignments – X
MCA Semester III
CG and Java Lab (CS3307)

Let's look into the following code.

```java
import java.awt.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class Test2 {
    private class Canvas extends JPanel {
        int x1, y1, x2, y2;

        Canvas(int x1, int y1, int x2, int y2) {
            this.x1 = x1;
            this.x2 = x2;
            this.y1 = y1;
            this.y2 = y2;
        }

        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.drawLine(x1, y1, x2, y2);
        }
    }

    Test2() {
        JFrame frm = new JFrame("Paint Demo");
        frm.setSize(300, 300);
        frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Canvas canvas = new Canvas(100, 100, 200, 150);
        frm.add(canvas);
        frm.setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Test2();
            }
        });
    }
}
```



Every lightweight component inherits the Component class's (which the JComponent extends) paint method. It renders the corresponding component with the help of three methods: paintComponent(), paintBorder() and paintChildren(). Remember, we do not override the paint() directly, rather to define how the component will be rendered, we override the paintComponent(). Moreover, we need to have the superclass portion of painting to take place, such as how eventually a JPanel will be rendered (remember, we are not controlling how JPanel will be rendered, it will always be rectangle with passed width and height). We are controlling only what will be rendered on the JPanel. So, provide your graphics after calling paintComponent of parent class. Then, a simple line is being drawn from (100, 100) to (200, 150). Observe, we are keeping the pixels within the dimension of the frame. The output of above code looks like as shown in the image on the left.

Unfortunately, there is no method to plot a single pixel in Swing (we need OpenGL for that). Anyway, we can simulate it as follows.

```
import java.awt.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class Test2 {
    private class Canvas extends JPanel {
        int w, h;

        Canvas(int w, int h) {
            this.w = w;
            this.h = h;
        }

        private void plotPixel(Graphics g, int x, int y) {
            g.fillRect(x+(w/2), (h/2)-y, 1, 1);
        }

        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            // This is where your algorithm will appear, call as method
            // such as drawBresenhamCircle(int radius, Graphics g) that will
            // internally call the plotPixel to plot the pixel. The plotPixel is
            // already translating the origin of the canvas from top-left corner
            // to center of the panel, so that you can focus on implementing the
            // algorithms only. So, consider the center of the canvas as (0, 0)
        }
    }
```

```
    Test2(int h, int w) {
        JFrame frm = new JFrame("Paint Demo");
        frm.setSize(w, h);
        frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Canvas canvas = new Canvas(w, h);
        frm.add(canvas);
        frm.setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Test2(300, 300);
            }
        });
    }
}
```

[In 1 and 2, the line will be drawn from (x1, y1) to (x2, y2). In 3 and 4, the center will be at (0, 0) and the circle will be of radius r]

1) Draw a line using DDA algorithm via
   drawDDALine(int x1, int y1, int x2, int y2, Graphics g)

2) Draw a line using Bresenham's algorithm via
   drawBresenhamLine(int x1, int y1, int x2, int y2, Graphics g)

3) Draw a circle using Bresenham's algorithm via
   drawBresenhamCircle(int r, Graphics g)

4) Draw a circle using Midpoint method via
   drawMidpointCircle(int r, Graphics g)

5) Draw a line using any of the above method, and then translate it 50 pixels to right and 50 pixels to up. The JPanel must show both lines. Then rotate the translated line by 30 degrees clockwise around its leftmost point. Add this third line to the JPanel [Hint: while drawing the line, start adding the plotted pixels to an array or list. While rotating or translating, apply the operation on the stored pixel values.]

6) Draw a Bezier Curve via three points using the method
   drawBezier(int x1, int y1, int x2, int y2, int x3, int y3, Graphics g)