CN LAB 1

Name: Rachit Nimje
Class: TY-CS-D
Roll no: 12

Batch: 1
PRN: 12210952

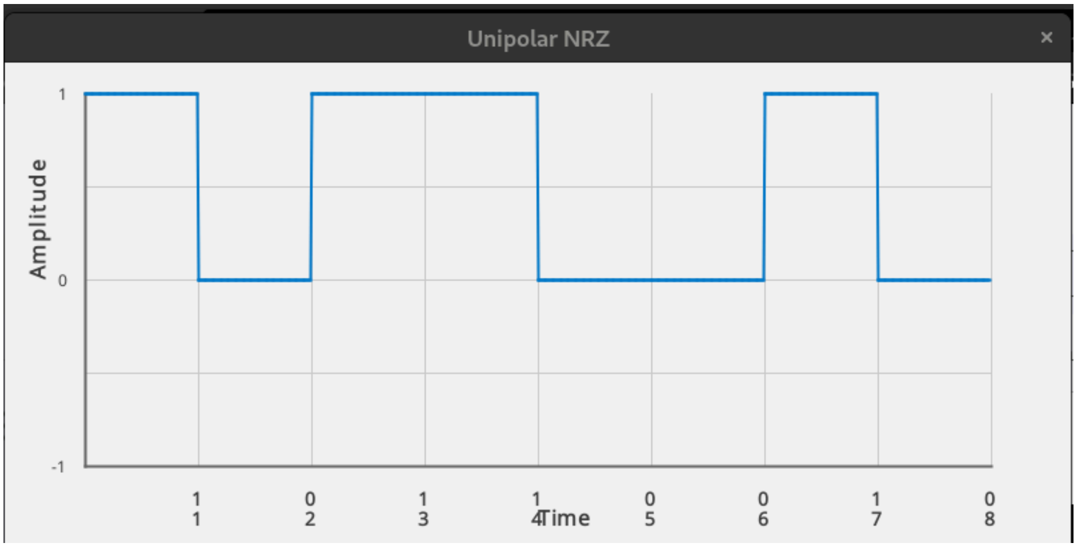
Title: Write a program in C++/JAVA to implement - Unipolar NRZ, Polar NRZ, NRZ Inverted, Bipolar Encoding, Manchester Encoding and Differential Manchester Encoding.

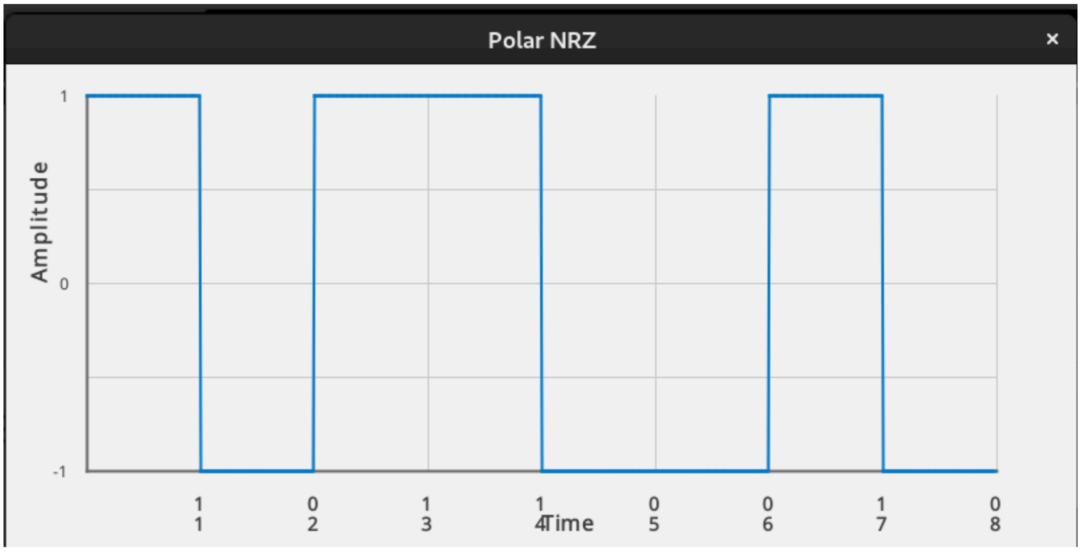
Code:

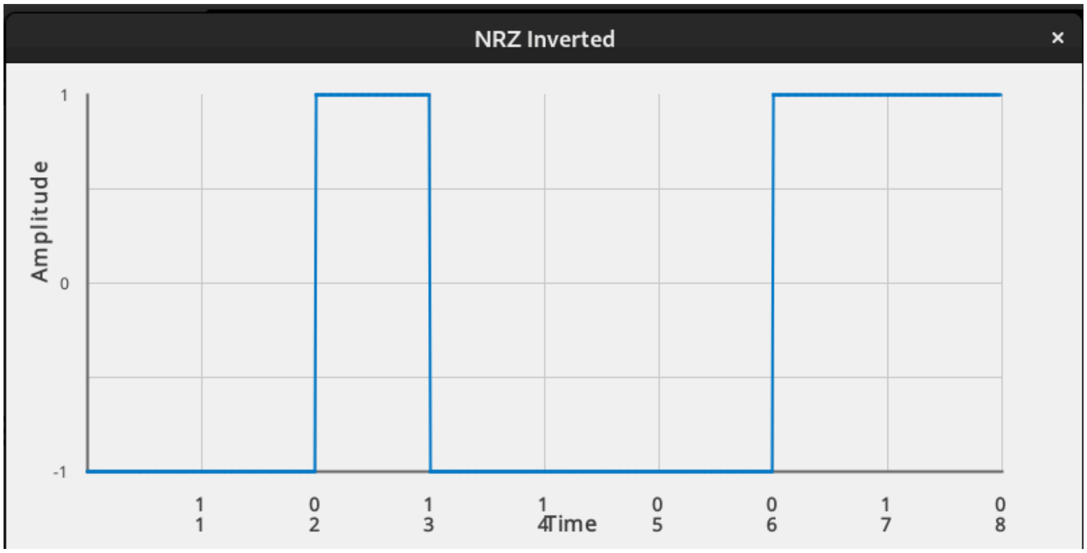
```java
import javax.swing.*;
import java.awt.*;
import java.awt.geom.Line2D;
import java.util.Arrays;

public class LineEncoding extends JFrame {
   private int[] bits;
   private double[] signal;
   private String title;

   // gui config
   private static final int WIDTH = 800;
   private static final int HEIGHT = 400;
   private static final int PADDING = 60;
   private static final Color BACKGROUND_COLOR = new Color(240, 240, 240);
   private static final Color AXIS_COLOR = new Color(100, 100, 100);
   private static final Color SIGNAL_COLOR = new Color(0, 120, 200);
   private static final Color GRID_COLOR = new Color(200, 200, 200);
   private static final Color TEXT_COLOR = new Color(60, 60, 60);

   public LineEncoding(int[] bits, double[] signal, String title) {
       this.bits = bits;
       this.signal = signal;
       this.title = title;
       setTitle(title);
       setSize(WIDTH, HEIGHT);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setLocationByPlatform(true);
       setResizable(false);
   }

   @Override
   public void paint(Graphics g) {
       super.paint(g);
       Graphics2D g2 = (Graphics2D) g;
```

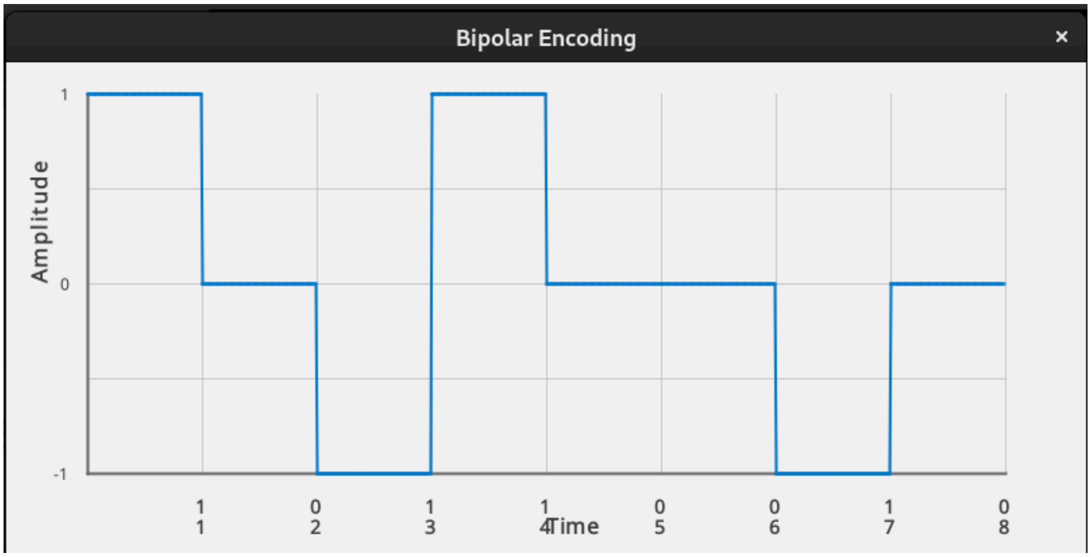
```
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        g2.setColor(BACKGROUND_COLOR);
        g2.fillRect(0, 0, WIDTH, HEIGHT);

        g2.setColor(GRID_COLOR);
        for (int i = 0; i <= bits.length; i++) {
            int x = PADDING + i * (WIDTH - 2 * PADDING) / bits.length;
            g2.drawLine(x, PADDING, x, HEIGHT - PADDING);
        }
        for (int i = -1; i <= 1; i++) {
            int y = HEIGHT / 2 + i * (HEIGHT - 2 * PADDING) / 4;
            g2.drawLine(PADDING, y, WIDTH - PADDING, y);
        }

        g2.setColor(AXIS_COLOR);
        g2.setStroke(new BasicStroke(2));
        g2.drawLine(PADDING, HEIGHT - PADDING, WIDTH - PADDING, HEIGHT -
PADDING);
        g2.drawLine(PADDING, PADDING, PADDING, HEIGHT - PADDING);

        g2.setColor(SIGNAL_COLOR);
        g2.setStroke(new BasicStroke(2));
        double xScale = (double) (WIDTH - 2 * PADDING) / signal.length;
        double yScale = (double) (HEIGHT - 2 * PADDING) / 2;
        for (int i = 1; i < signal.length; i++) {
            double x1 = PADDING + (i - 1) * xScale;
            double y1 = HEIGHT / 2 - signal[i - 1] * yScale;
            double x2 = PADDING + i * xScale;
            double y2 = HEIGHT / 2 - signal[i] * yScale;
            g2.draw(new Line2D.Double(x1, y1, x2, y2));
        }

        g2.setColor(TEXT_COLOR);
        g2.setFont(new Font("Arial", Font.BOLD, 14));
        for (int i = 0; i < bits.length; i++) {
            int x = PADDING + (i + 1) * (WIDTH - 2 * PADDING) / bits.length;
            g2.drawString(Integer.toString(bits[i]), x - 5, HEIGHT - PADDING /
2);
            g2.drawString(Integer.toString(i + 1), x - 5, HEIGHT - PADDING /
4);
        }

        g2.setFont(new Font("Arial", Font.PLAIN, 12));
        g2.drawString("0", PADDING - 20, HEIGHT / 2 + 5);
        g2.drawString("1", PADDING - 20, PADDING + 5);
        g2.drawString("-1", PADDING - 25, HEIGHT - PADDING + 5);
        g2.setFont(new Font("Arial", Font.BOLD, 16));
        g2.drawString("Time", WIDTH / 2, HEIGHT - PADDING / 4);
        g2.rotate(-Math.PI / 2);
        g2.drawString("Amplitude", -HEIGHT / 2, PADDING / 2);
        g2.rotate(Math.PI / 2);

        g2.setFont(new Font("Arial", Font.BOLD, 20));
        g2.drawString(title, WIDTH / 2 - g2.getFontMetrics().stringWidth(title)
/ 2, 30);
    }

    private static double[] unipolarNRZ(int[] bits) {
        double[] signal = new double[bits.length * 100];
        for (int i = 0; i < bits.length; i++) {
            Arrays.fill(signal, i * 100, (i + 1) * 100, bits[i] == 1 ? 1 : 0);
        }
        return signal;
    }

    private static double[] polarNRZ(int[] bits) {
        double[] signal = new double[bits.length * 100];
        for (int i = 0; i < bits.length; i++) {
            Arrays.fill(signal, i * 100, (i + 1) * 100, bits[i] == 1 ? 1 : -1);
        }
        return signal;
    }

    private static double[] nrzInverted(int[] bits) {
        double[] signal = new double[bits.length * 100];
        int currentLevel = 1;
        for (int i = 0; i < bits.length; i++) {
            if (bits[i] == 1) currentLevel = -currentLevel;
            Arrays.fill(signal, i * 100, (i + 1) * 100, currentLevel);
        }
        return signal;
    }

    private static double[] bipolarEncoding(int[] bits) {
        double[] signal = new double[bits.length * 100];
        int lastOne = 1;
        for (int i = 0; i < bits.length; i++) {
            if (bits[i] == 1) {
                Arrays.fill(signal, i * 100, (i + 1) * 100, lastOne);
                lastOne = -lastOne;
            }
        }
        return signal;
    }
    private static double[] manchesterEncoding(int[] bits) {
        double[] signal = new double[bits.length * 100];
        for (int i = 0; i < bits.length; i++) {
            Arrays.fill(signal, i * 100, i * 100 + 50, bits[i] == 0 ? 1 : -1);
            Arrays.fill(signal, i * 100 + 50, (i + 1) * 100, bits[i] == 0 ? -1
: 1);
        }
        return signal;
    }

    private static double[] differentialManchesterEncoding(int[] bits) {
        double[] signal = new double[bits.length * 100];
        boolean lastBitOne = true;
        for (int i = 0; i < bits.length; i++) {
            if (bits[i] == 0) lastBitOne = !lastBitOne;
            Arrays.fill(signal, i * 100, i * 100 + 50, lastBitOne ? 1 : -1);
            Arrays.fill(signal, i * 100 + 50, (i + 1) * 100, lastBitOne ? -1 :
1);
        }
        return signal;
    }

    public static void main(String[] args) {
        int[] bits = {1, 0, 1, 1, 0, 0, 1, 0};

        SwingUtilities.invokeLater(() -> {
            new LineEncoding(bits, unipolarNRZ(bits), "Unipolar
NRZ").setVisible(true);
            new LineEncoding(bits, polarNRZ(bits), "Polar
NRZ").setVisible(true);
            new LineEncoding(bits, nrzInverted(bits), "NRZ
Inverted").setVisible(true);
            new LineEncoding(bits, bipolarEncoding(bits), "Bipolar
Encoding").setVisible(true);
            new LineEncoding(bits, manchesterEncoding(bits), "Manchester
Encoding").setVisible(true);
            new LineEncoding(bits, differentialManchesterEncoding(bits),
"Differential Manchester Encoding").setVisible(true);
        });
    }
}
```

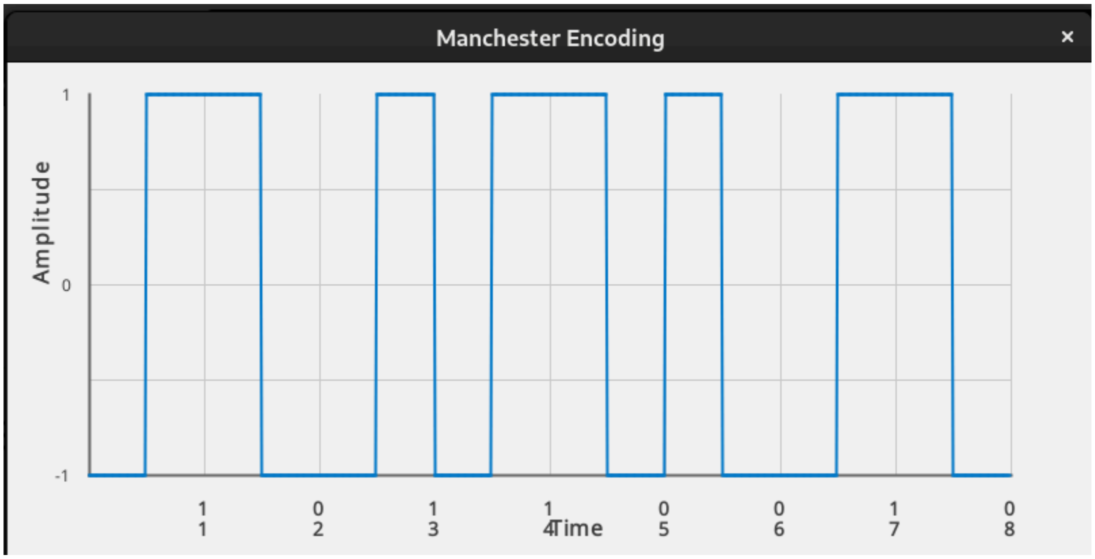
Output:

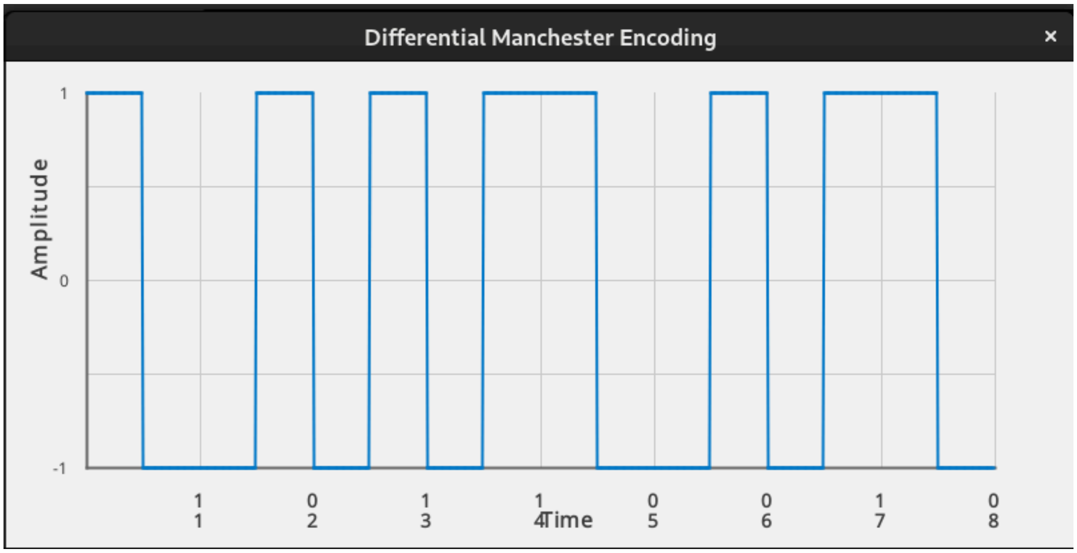

Unipolar NRZ
×
Amplitude
1
0
-1
1 1
0 2
1 3
1 4
Time
0 5
0 6
1 7
0 8


Polar NRZ
×
Amplitude
1
0
-1
1 1
0 2
1 3
1 4
Time
0 5
0 6
1 7
0 8


NRZ Inverted
×
Amplitude
1
0
-1
1 1
0 2
1 3
1 4
Time
0 5
0 6
1 7
0 8


Bipolar Encoding
×
Amplitude
1
0
-1
1 1
0 2
1 3
1 4Time
0 5
0 6
1 7
0 8


Manchester Encoding
×
Amplitude
1
0
-1
1 1
0 2
1 3
1 4Time
0 5
0 6
1 7
0 8


Differential Manchester Encoding
×
Amplitude
1
0
-1
1 1
0 2
1 3
1 4Time
0 5
0 6
1 7
0 8