ASSIGNMENT 2
Part II

CSc 221
Spring 2016

I. The program

Write a program that uses the three classes **MonteCarlo** (with the **main()** method), **Simulation**, and **Metrics** of Part I and a new class **Histogram** to visually represent the behavior of normally distributed random numbers.

Here is a template of the class **Histogram**:

```
import javax.swing.JPanel;

public class Histogram extends JPanel {

        final int TOP_MARGIN = 20;
        final int BOTTOM_MARGIN = 20;
        final int LEFT_MARGIN = 20;
        final int RIGHT_MARGIN = 20;

        // Declarations of instance variables:
        // private ...
        // private ...
        // ...

        // constructor
        public Histogram(Simulation s) {
                setBackground(Color.WHITE);

                // Set values of instance variables ...
        }

        // paintConponent draws the histogram
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                drawXAxis(g);
                drawYAxis(g);
                drawBins(g);
                drawXLabels(g);
                drawYLabels(g);
        }

        // drawXAxis Draws the x-axis
        private void drawXAxis(Graphics g) {
                int x1 = LEFT_MARGIN;
                int y1 = getHeight() - BOTTOM_MARGIN;
                int x2 = getWidth() - RIGHT_MARGIN;
                int y2 = y1;
        g.drawLine(x1, y1, x2, y2);
    }

    // drawYAxis Draws the y-axis
    private void drawYAxis(Graphics g) {
        int x1 = LEFT_MARGIN;
        int y1 = getHeight() - BOTTOM_MARGIN;
        int x2 = x1;
        int y2 = TOP_MARGIN;
        g.drawLine(x1, y1, x2, y2);
    }

    // drawBins draws the bins
    private void drawBins(Graphics g) {
        g.setColor(Color.GRAY);

        // Your code here
        // ...
        // ...
    }

    // drawXLabels draws the labels along the x-axis
    private void drawXLabels(Graphics g) {
        g.setColor(Color.BLACK);

        DecimalFormat formatter = new DecimalFormat();
        formatter.setMinimumFractionDigits(2);
        formatter.setMaximumFractionDigits(2);

        // Sample code (which you may or may not choose to use)
        double labelVal = min;
        String label = formatter.format(labelVal);
        int x = LEFT_MARGIN;
        int y = getHeight() - BOTTOM_MARGIN + 12;
        for (int b : bins) {
            g.drawString(label, x-12, y);
            x += binWidth;
            labelVal += binSize;
            label = formatter.format(labelVal);
        }
        g.drawString(label, x-12, y);
    }


    // drawYLabels draws the labels along the y-axis,
    // i.e., draws the count of the bins on top of the bins
    private void drawYLabels(Graphics g) {
        g.setColor(Color.BLUE);

        DecimalFormat formatter = new DecimalFormat("#,###");

        // Your code here
        // ...
        // ...
    }
```

```
}
```

You will also likely use a scaling method (to scale actual values to their graphical equivalents), with signature as follows:

scaleY(int, double)

scaleY will take as input an actual value and a ratio representing your scale and return the value's graphical equivalent.

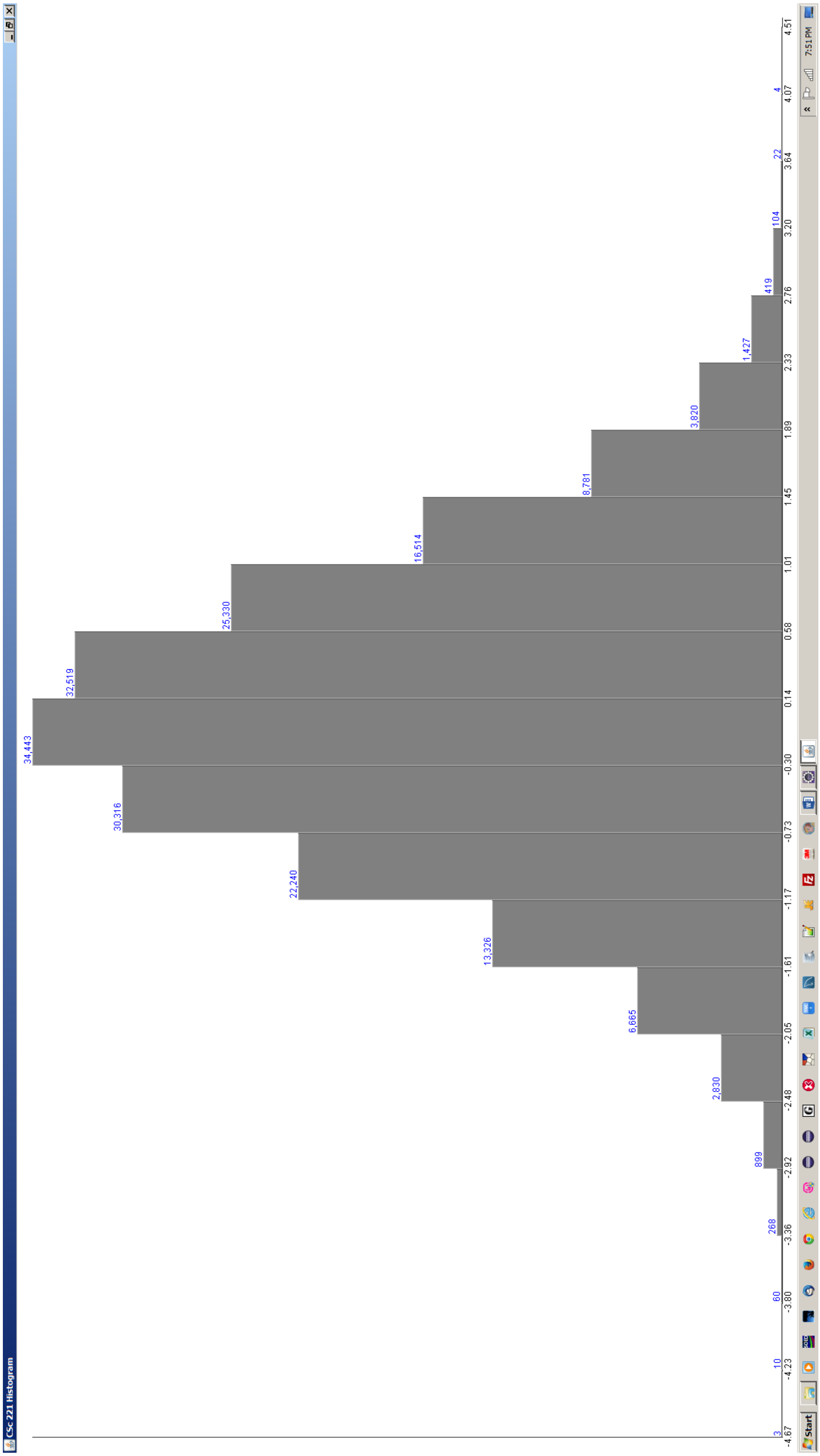
II. Output

Invoke **Histogram** from **MonteCarlo** (from the **main()** method), as follows:

```
Histogram h = new Histogram(s);
JFrame visuals = new JFrame();

visuals.setTitle("CSc 221 Histogram");
visuals.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
visuals.add(h);
visuals.setSize(1200, 800);
visuals.setVisible(true);
```

Your output would have the appearance of the following output (see also the uploaded JPEG file "output.jpg"):


CSc 221 Histogram
3
10
60
268
899
2,830
6,665
13,326
22,240
30,316
34,443
32,519
25,330
16,514
8,781
3,820
1,427
419
104
22
4
-4.67
-4.23
-3.80
-3.36
-2.92
-2.48
-2.05
-1.61
-1.17
-0.73
-0.30
0.14
0.58
1.01
1.45
1.89
2.33
2.76
3.20
3.64
4.07
4.51
Start
7:51 PM

III. Some guidelines

You may make *minor* changes to classes used in Part I. These changes should be *superficial* only, such as changes to the scope of variables (**private** to **public**, for example).

Use methods for calculations that are repeatedly invoked (as in the above **scaleY()** case).

Do not use **double** for **Double** (even if you might be getting away with it), or vice versa.

Use the *enhanced* **for** loop wherever possible.

Modularize code.

Comment code.

Submit *entire* project (all four classes) exported into a zip file.