PSS3 Report

Mouaz Ali

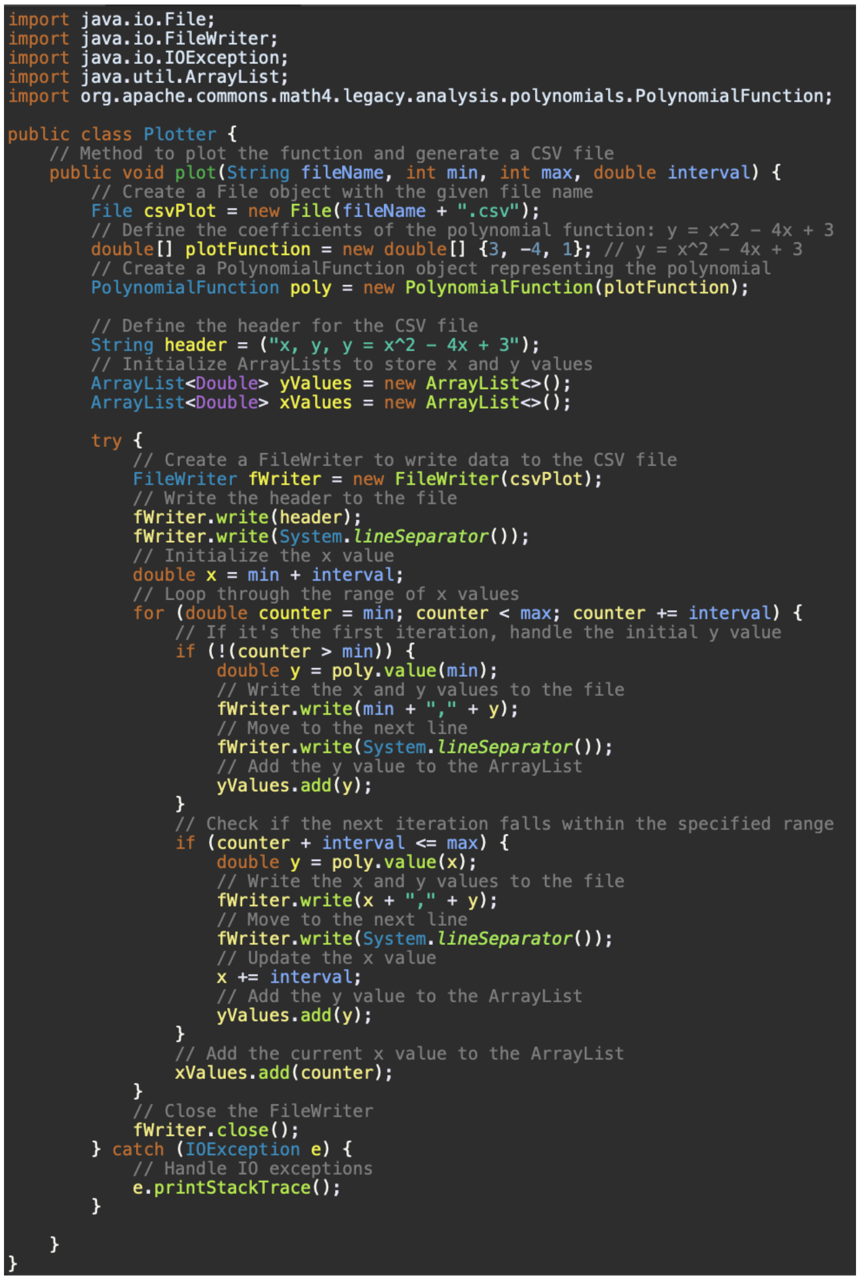
Table of Contents

This program consists of five Java classes. Each class has a specific role in generating, manipulating, and visualizing data. Together, they form a statistical toolset.

1. **Plotter:**
This class is responsible for generating a CSV file containing x and y values of a polynomial function, plotting the function using JFreeChart, and displaying the plot on a GUI.

- Functionalities:
  - Generates a CSV file with x and y values of a predefined polynomial function.
  - Uses JFreeChart to create a plot of the function.
  - Displays the plot on a GUI window.

Class Code:

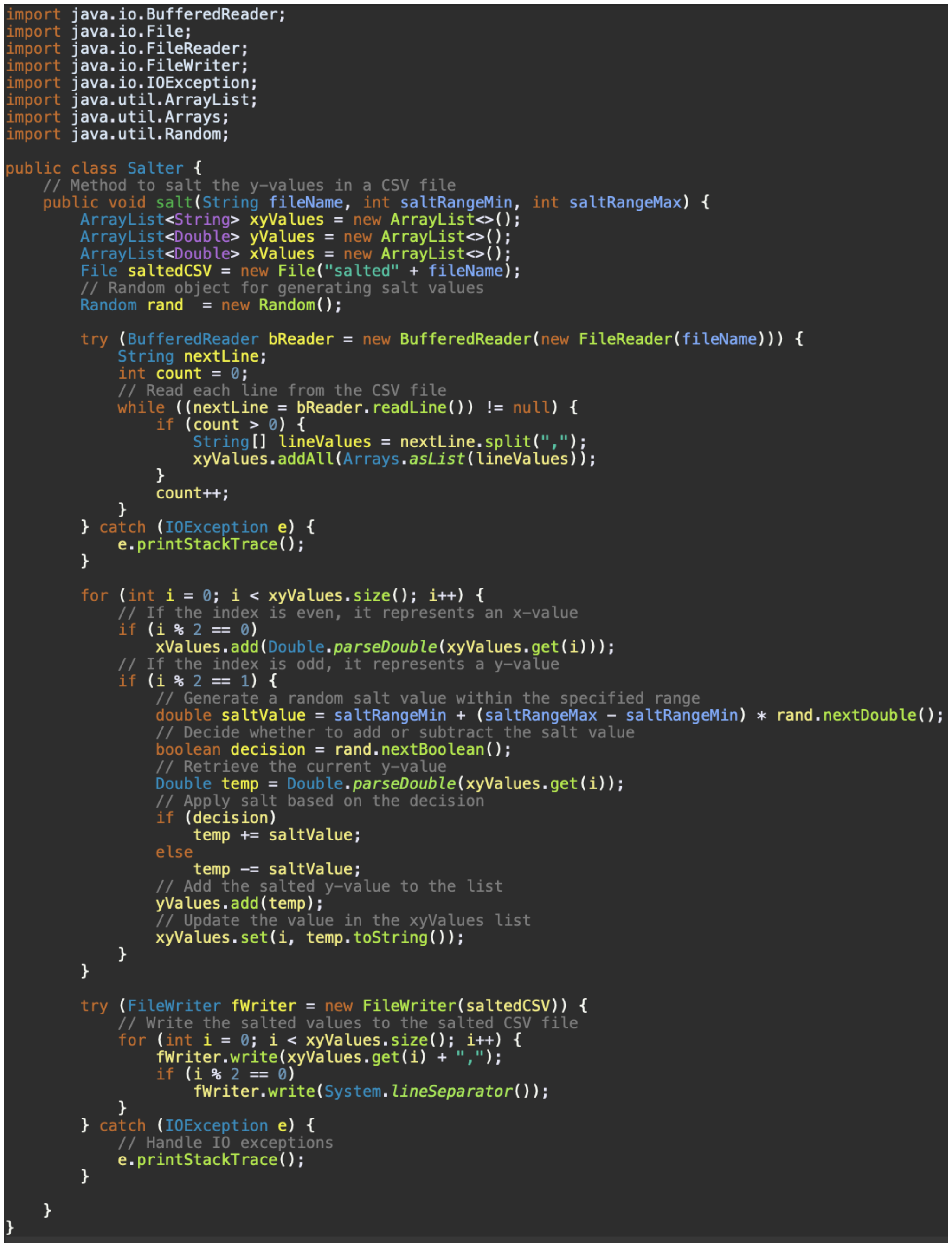


```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.commons.math4.legacy.analysis.polynomials.PolynomialFunction;

public class Plotter {
    // Method to plot the function and generate a CSV file
    public void plot(String fileName, int min, int max, double interval) {
        // Create a File object with the given file name
        File csvPlot = new File(fileName + ".csv");
        // Define the coefficients of the polynomial function: y = x^2 - 4x + 3
        double[] plotFunction = new double[] {3, -4, 1}; // y = x^2 - 4x + 3
        // Create a PolynomialFunction object representing the polynomial
        PolynomialFunction poly = new PolynomialFunction(plotFunction);

        // Define the header for the CSV file
        String header = ("x, y, y = x^2 - 4x + 3");
        // Initialize ArrayLists to store x and y values
        ArrayList<Double> yValues = new ArrayList<>();
        ArrayList<Double> xValues = new ArrayList<>();

        try {
            // Create a FileWriter to write data to the CSV file
            FileWriter fWriter = new FileWriter(csvPlot);
            // Write the header to the file
            fWriter.write(header);
            fWriter.write(System.lineSeparator());
            // Initialize the x value
            double x = min + interval;
            // Loop through the range of x values
            for (double counter = min; counter < max; counter += interval) {
                // If it's the first iteration, handle the initial y value
                if (!(counter > min)) {
                    double y = poly.value(min);
                    // Write the x and y values to the file
                    fWriter.write(min + "," + y);
                    // Move to the next line
                    fWriter.write(System.lineSeparator());
                    // Add the y value to the ArrayList
                    yValues.add(y);
                }
                // Check if the next iteration falls within the specified range
                if (counter + interval <= max) {
                    double y = poly.value(x);
                    // Write the x and y values to the file
                    fWriter.write(x + "," + y);
                    // Move to the next line
                    fWriter.write(System.lineSeparator());
                    // Update the x value
                    x += interval;
                    // Add the y value to the ArrayList
                    yValues.add(y);
                }
                // Add the current x value to the ArrayList
                xValues.add(counter);
            }
            // Close the FileWriter
            fWriter.close();
        } catch (IOException e) {
            // Handle IO exceptions
            e.printStackTrace();
        }

    }
}
```

2. **Salter:**
This class takes an existing CSV file, adds random noise (salt) to the y values within a specified range, and saves the modified data to a new CSV file. It also plots the salted data using JFreeChart and displays the plot on a GUI.

- Functionalities:
  - Reads an existing CSV file containing x and y values.
  - Adds random noise to the y values within a specified range.
  - Saves the modified data to a new CSV file.
  - Plots the salted data using JFreeChart and displays the plot on a GUI window.

Class Code:

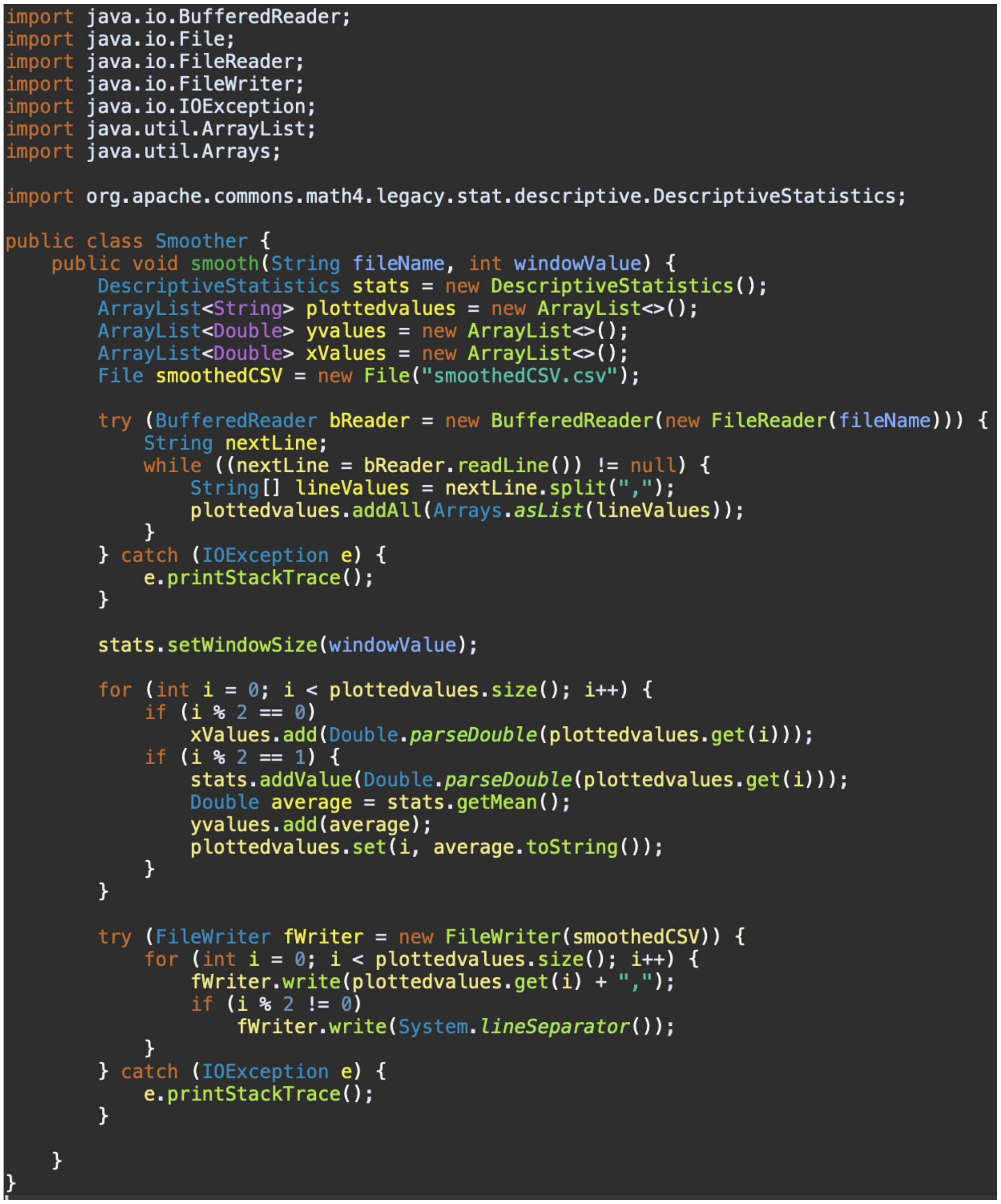


```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

public class Salter {
    // Method to salt the y-values in a CSV file
    public void salt(String fileName, int saltRangeMin, int saltRangeMax) {
        ArrayList<String> xyValues = new ArrayList<>();
        ArrayList<Double> yValues = new ArrayList<>();
        ArrayList<Double> xValues = new ArrayList<>();
        File saltedCSV = new File("salted" + fileName);
        // Random object for generating salt values
        Random rand  = new Random();

        try (BufferedReader bReader = new BufferedReader(new FileReader(fileName))) {
            String nextLine;
            int count = 0;
            // Read each line from the CSV file
            while ((nextLine = bReader.readLine()) != null) {
                if (count > 0) {
                    String[] lineValues = nextLine.split(",");
                    xyValues.addAll(Arrays.asList(lineValues));
                }
                count++;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        for (int i = 0; i < xyValues.size(); i++) {
            // If the index is even, it represents an x-value
            if (i % 2 == 0)
                xValues.add(Double.parseDouble(xyValues.get(i)));
            // If the index is odd, it represents a y-value
            if (i % 2 == 1) {
                // Generate a random salt value within the specified range
                double saltValue = saltRangeMin + (saltRangeMax - saltRangeMin) * rand.nextDouble();
                // Decide whether to add or subtract the salt value
                boolean decision = rand.nextBoolean();
                // Retrieve the current y-value
                Double temp = Double.parseDouble(xyValues.get(i));
                // Apply salt based on the decision
                if (decision)
                    temp += saltValue;
                else
                    temp -= saltValue;
                // Add the salted y-value to the list
                yValues.add(temp);
                // Update the value in the xyValues list
                xyValues.set(i, temp.toString());
            }
        }

        try (FileWriter fWriter = new FileWriter(saltedCSV)) {
            // Write the salted values to the salted CSV file
            for (int i = 0; i < xyValues.size(); i++) {
                fWriter.write(xyValues.get(i) + ",");
                if (i % 2 == 0)
                    fWriter.write(System.lineSeparator());
            }
        } catch (IOException e) {
            // Handle IO exceptions
            e.printStackTrace();
        }

    }
}
```

3. **Smoother:**
This class reads a CSV file containing x and y values, applies a smoothing algorithm to the y values, and saves the smoothed data to a new CSV file. It also plots the smoothed data using JFreeChart and displays the plot on a GUI.

- Functionalities:
    - Reads a CSV file containing x and y values.
    - Applies a smoothing algorithm to the y values using Apache Commons Math.
    - Saves the smoothed data to a new CSV file.
    - Plots the smoothed data using JFreeChart and displays the plot on a GUI window.

Class Code:

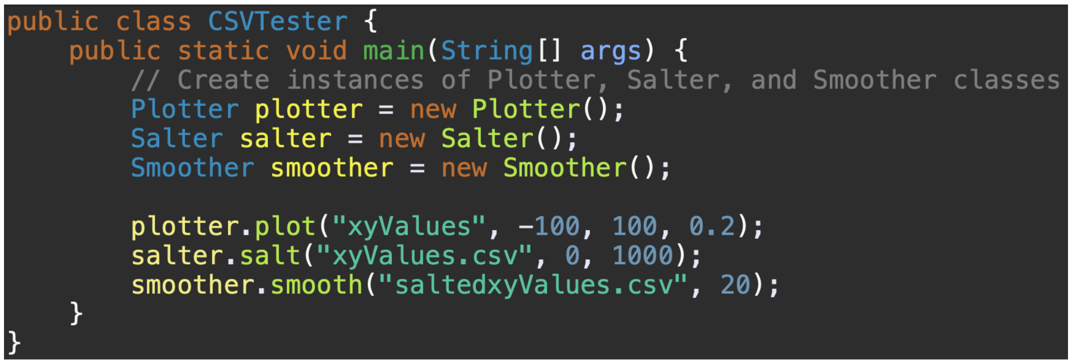


```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;

import org.apache.commons.math4.legacy.stat.descriptive.DescriptiveStatistics;

public class Smoother {
    public void smooth(String fileName, int windowValue) {
        DescriptiveStatistics stats = new DescriptiveStatistics();
        ArrayList<String> plottedvalues = new ArrayList<>();
        ArrayList<Double> yvalues = new ArrayList<>();
        ArrayList<Double> xValues = new ArrayList<>();
        File smoothedCSV = new File("smoothedCSV.csv");

        try (BufferedReader bReader = new BufferedReader(new FileReader(fileName))) {
            String nextLine;
            while ((nextLine = bReader.readLine()) != null) {
                String[] lineValues = nextLine.split(",");
                plottedvalues.addAll(Arrays.asList(lineValues));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        stats.setWindowSize(windowValue);

        for (int i = 0; i < plottedvalues.size(); i++) {
            if (i % 2 == 0)
                xValues.add(Double.parseDouble(plottedvalues.get(i)));
            if (i % 2 == 1) {
                stats.addValue(Double.parseDouble(plottedvalues.get(i)));
                Double average = stats.getMean();
                yvalues.add(average);
                plottedvalues.set(i, average.toString());
            }
        }

        try (FileWriter fWriter = new FileWriter(smoothedCSV)) {
            for (int i = 0; i < plottedvalues.size(); i++) {
                fWriter.write(plottedvalues.get(i) + ",");
                if (i % 2 != 0)
                    fWriter.write(System.lineSeparator());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

4. **CSVTester:**
This class serves as the main entry point for the program. It instantiates objects of the Plotter, Salter, and Smoother classes, calls their respective methods to perform plotting, salting, and smoothing operations, and displays the resulting plots on GUI windows.

- Functionalities:
  - Instantiates objects of the Plotter, Salter, and Smoother classes.
  - Calls methods of these classes to perform plotting, salting, and smoothing operations.
  - Displays the resulting plots on GUI windows.

Class Code:

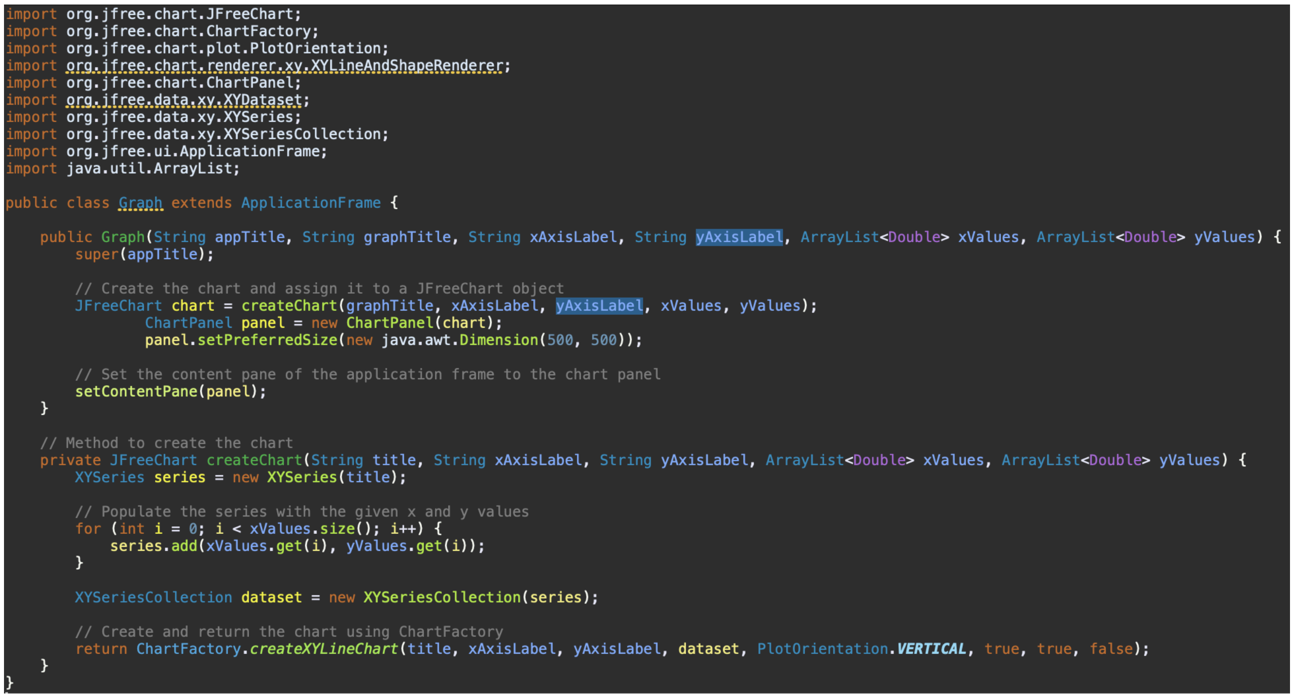


```java
public class CSVTester {
    public static void main(String[] args) {
        // Create instances of Plotter, Salter, and Smoother classes
        Plotter plotter = new Plotter();
        Salter salter = new Salter();
        Smoother smoother = new Smoother();

        plotter.plot("xyValues", -100, 100, 0.2);
        salter.salt("xyValues.csv", 0, 1000);
        smoother.smooth("saltedxyValues.csv", 20);
    }
}
```

5. **Graph:**
This class extends the JFreeChart ApplicationFrame class and is responsible for creating and displaying line plots using JFreeChart.

- Functionalities:
  - Creates and displays line plots using JFreeChart.
  - Accepts x and y values as input for plotting.

Class Code:



```java
import org.jfree.chart.JFreeChart;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.chart.ChartPanel;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.ui.ApplicationFrame;
import java.util.ArrayList;

public class Graph extends ApplicationFrame {

    public Graph(String appTitle, String graphTitle, String xAxisLabel, String yAxisLabel, ArrayList<Double> xValues, ArrayList<Double> yValues) {
        super(appTitle);

        // Create the chart and assign it to a JFreeChart object
        JFreeChart chart = createChart(graphTitle, xAxisLabel, yAxisLabel, xValues, yValues);
                ChartPanel panel = new ChartPanel(chart);
                panel.setPreferredSize(new java.awt.Dimension(500, 500));

        // Set the content pane of the application frame to the chart panel
        setContentPane(panel);
    }

    // Method to create the chart
    private JFreeChart createChart(String title, String xAxisLabel, String yAxisLabel, ArrayList<Double> xValues, ArrayList<Double> yValues) {
        XYSeries series = new XYSeries(title);

        // Populate the series with the given x and y values
        for (int i = 0; i < xValues.size(); i++) {
            series.add(xValues.get(i), yValues.get(i));
        }

        XYSeriesCollection dataset = new XYSeriesCollection(series);

        // Create and return the chart using ChartFactory
        return ChartFactory.createXYLineChart(title, xAxisLabel, yAxisLabel, dataset, PlotOrientation.VERTICAL, true, true, false);
    }
}
```

Results

The polynomial function and data parameters in the Java Apache and JFreeCharts trials were the same as in the earlier program implementations to ensure consistency across different environments. In the first trial, the population was a range of -10 to 10 with 0.1increments. Using PolynomialFunction made equation representation easier and made calculations for data plotting more efficient. The first graph was produced by the plotter method. After that, the salter class added randomness, ranging from 0 to 250, to the data. The resulting graph showed a random distribution of data points with the expected parabola trend. A window size of 8 was applied through the smoother. Though it still showed dispersed data points, the first graph from the smoother approach seemed smoother than the salter graph, much like the original Java version and the MATLAB experiments. Two more runs of the smoother were conducted to ensure compatibility with the other parts. The generated graphs showed smoother curves similar to the MATLAB and Excel graphs.

In the second trial, a population ranging from -100 to 100 was used, with increments of 0.2. Similar to the initial trial, the resulting graph closely resembled the plotter graph from the first trial. Next, the salter method was run with a salt range of 1000. Similar to previous iterations, the salter graph showed a spikey curve that was almost like a "U." The smoother was applied with a window size of 20 to the salted data, producing a graph that seemed less successful than the initial smoother run in the first Java program with a larger scale. This could possibly be due to the differences in detail between Excel and JFreeCharts graphs. The final graphs showed results similar to the second and third smoother runs in the larger data MATLAB trials. But one thing that stood out about this trial was how difficult it was for the smoother to reduce the big data spikes at the beginning of the curve. However, when compared to the salted graph, the smoothed graphs stayed much closer to the original curve.

Discussion:

JFreeChart serves as a powerful tool for data visualization in Java applications. It offers extensive features for creating high-quality charts and graphs, making it well-suited for a wide range of data visualization tasks. A real-world application of data processing and visualization is demonstrated by this program. It first creates CSV files with plotted data, which are then salted and smoothed, all while being visually represented using JFreeChart.

A key component of this program is the Graph class. The data from the CSV files is displayed in an interactive graphical user interface that is created by utilizing JFreeChart's features. A powerful framework for making a variety of chart formats, such as line charts, bar charts, and scatter plots, is offered by JFreeChart. JFreeChart's adaptability and flexibility are two of its main benefits. Because of this adaptability, extremely informative and eye-catching charts that are specific to the requirements of the user or application can be created.

On top of that, JFreeChart also offers interactive elements that improve user experience. Users have the ability to examine data points for in-depth research, pan across the chart, and zoom in and out. The graph class in this program creates the line charts. It accomplishes this by taking in x-values and y-values from the CSV files as ArrayLists and creates a line chart representing the relationship between these values. JFreeChart's ApplicationFrame is utilized to display the chart in a windowed environment, allowing users to interact with and explore the data visually.

All things considered, JFreeChart's incorporation into the Graph class improves the program's functionality by offering a user-friendly data visualization interface. This makes it easier for users to interpret and analyze the plotted, salted, and smoothed data produced by the other classes, allowing them to observe the graph with ease. In conclusion, JFreeChart's incorporation into the Graph class improves the program's usefulness by offering a smooth and user-friendly interface for data display.