JFreeChart and Apache Stats Library

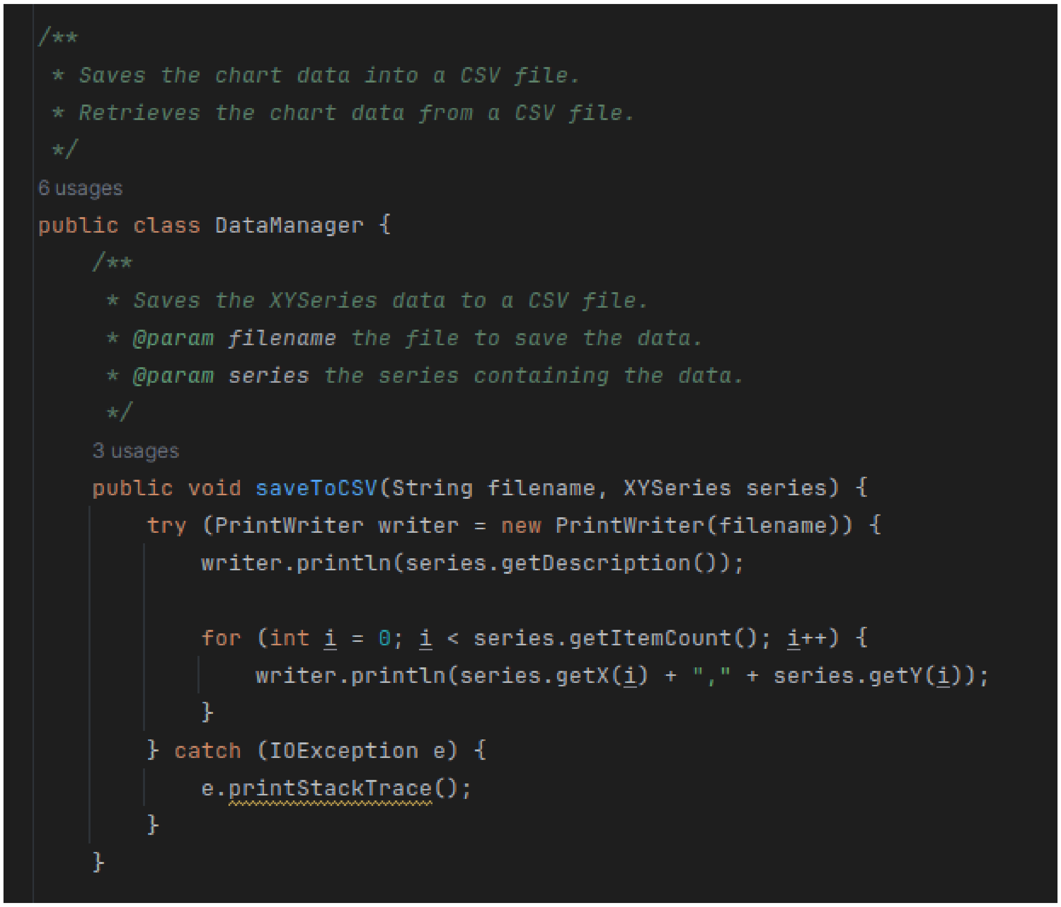
I downloaded **JFreeChart** from this website: https://www.jfree.org/jfreechart/download.html and the **Apache Stats Library** from the Downloads link on this site: https://commons.apache.org/proper/commons-math/userguide/stat.html

Then I created two helper classes:

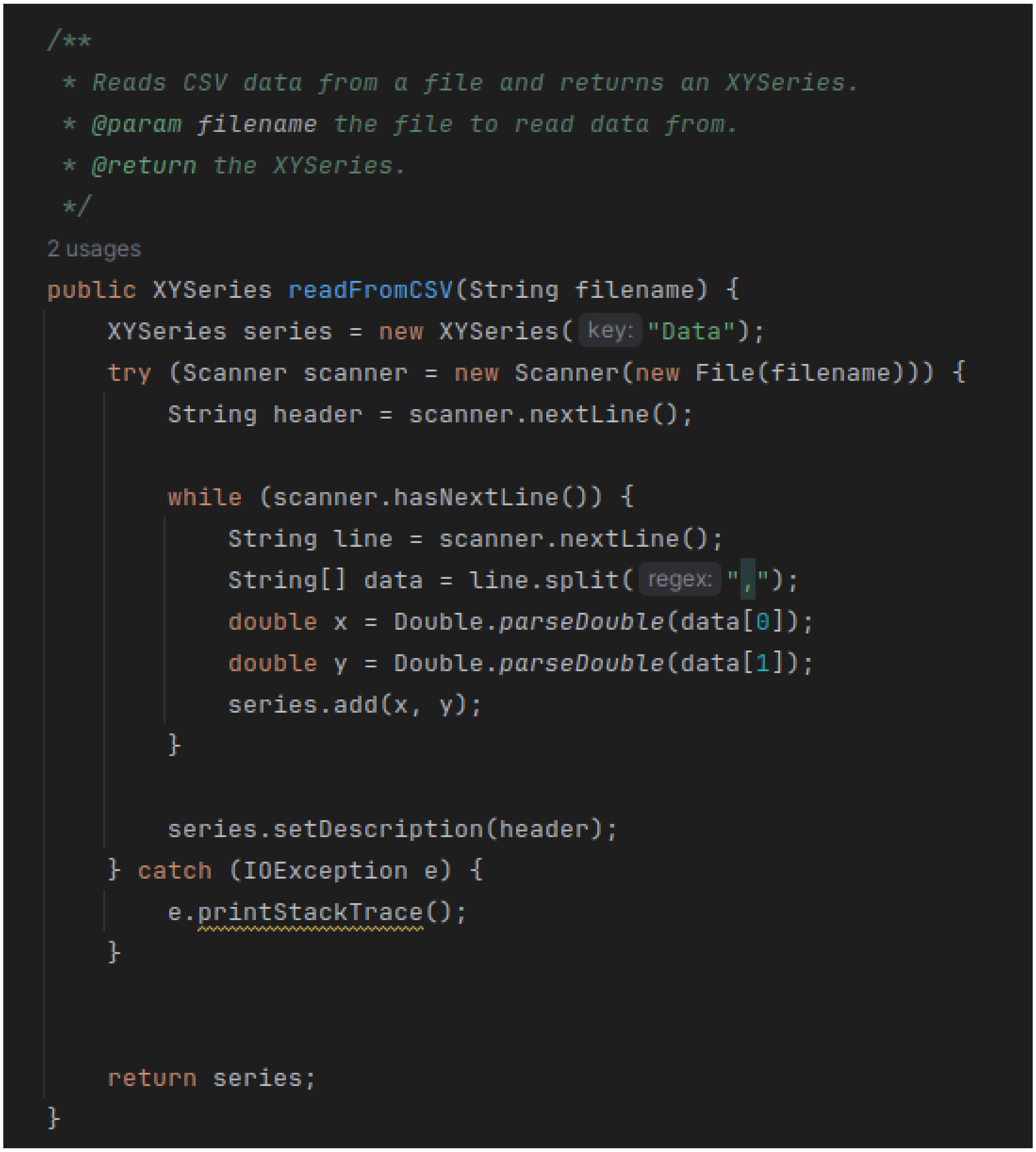
- **DataManager**: is responsible for reading data from **csv** file and writing data into a **csv** file
- **Displayer**: is responsible for plotting the data and saving the plot in a **png** file

DataManager

In place of **double[]** array or **List<Double>** data types, **XYSeries** from the **JFreeChart** library is used:



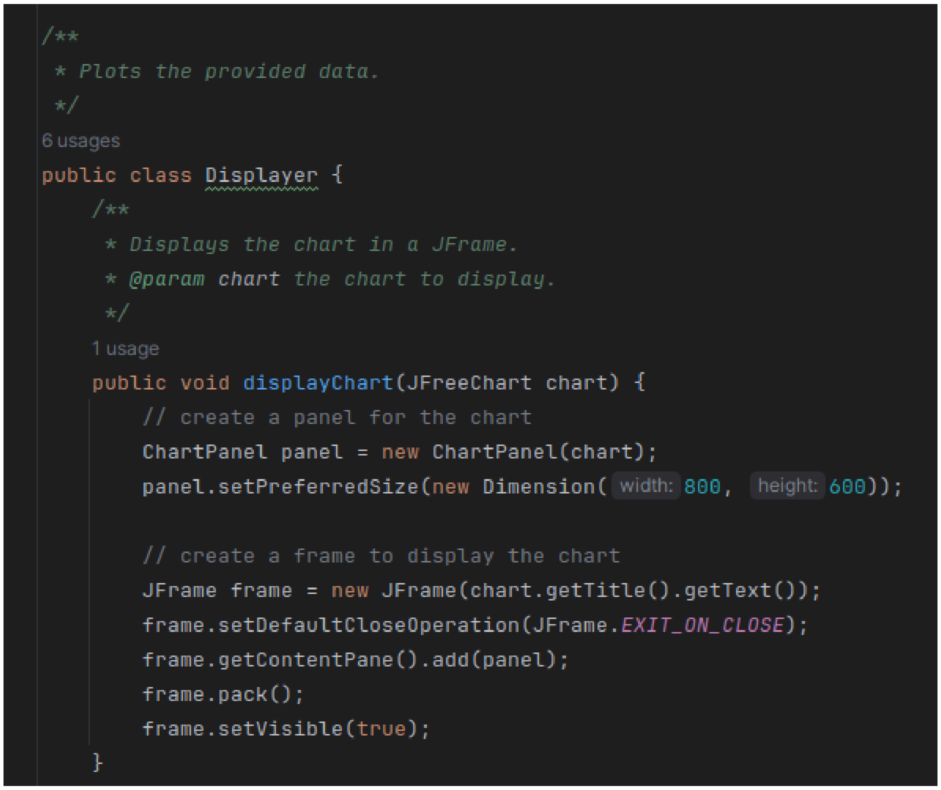
```java
/**
 * Saves the chart data into a CSV file.
 * Retrieves the chart data from a CSV file.
 */
6 usages
public class DataManager {
    /**
     * Saves the XYSeries data to a CSV file.
     * @param filename the file to save the data.
     * @param series the series containing the data.
     */
    3 usages
    public void saveToCSV(String filename, XYSeries series) {
        try (PrintWriter writer = new PrintWriter(filename)) {
            writer.println(series.getDescription());

            for (int i = 0; i < series.getItemCount(); i++) {
                writer.println(series.getX(i) + "," + series.getY(i));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```



```
/**
 * Reads CSV data from a file and returns an XYSeries.
 * @param filename the file to read data from.
 * @return the XYSeries.
 */
2 usages
public XYSeries readFromCSV(String filename) {
    XYSeries series = new XYSeries( key: "Data");
    try (Scanner scanner = new Scanner(new File(filename))) {
        String header = scanner.nextLine();

        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] data = line.split( regex: ",");
            double x = Double.parseDouble(data[0]);
            double y = Double.parseDouble(data[1]);
            series.add(x, y);
        }

        series.setDescription(header);
    } catch (IOException e) {
        e.printStackTrace();
    }


    return series;
}
```

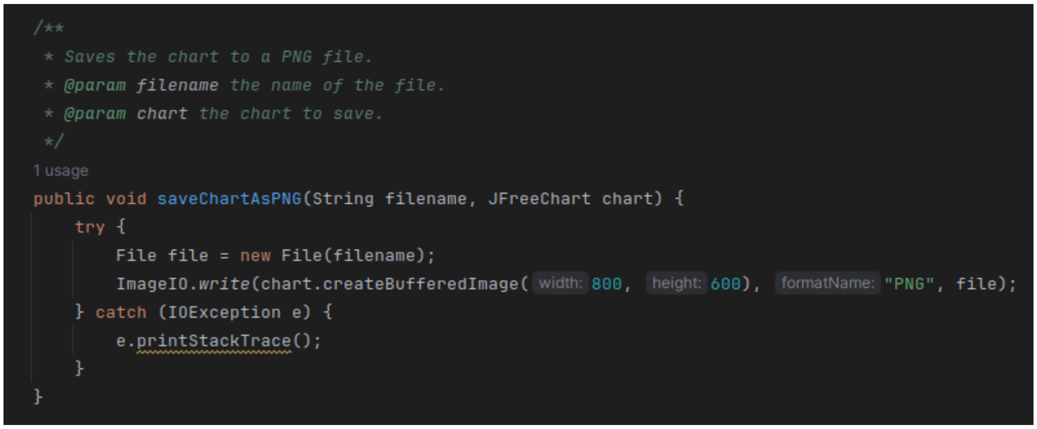
Displayer

Again, the chart data is passed using the **JFreeChart** classes. Classes from **java.awt** package are used to create the **JFrame** inside which the **ChartPanel** is placed:



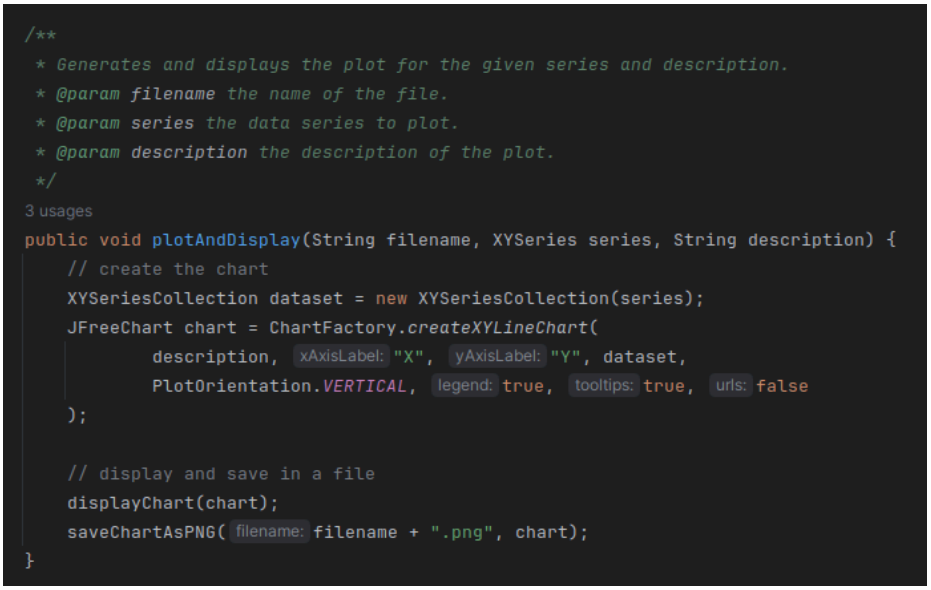
```
/**
 * Plots the provided data.
 */
6 usages
public class Displayer {
    /**
     * Displays the chart in a JFrame.
     * @param chart the chart to display.
     */
    1 usage
    public void displayChart(JFreeChart chart) {
        // create a panel for the chart
        ChartPanel panel = new ChartPanel(chart);
        panel.setPreferredSize(new Dimension( width: 800,  height: 600));

        // create a frame to display the chart
        JFrame frame = new JFrame(chart.getTitle().getText());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
```

The **ImageIO** class is used to save the chart to a **png** file:



```
/**
 * Saves the chart to a PNG file.
 * @param filename the name of the file.
 * @param chart the chart to save.
 */
1 usage
public void saveChartAsPNG(String filename, JFreeChart chart) {
    try {
        File file = new File(filename);
        ImageIO.write(chart.createBufferedImage( width: 800,  height: 600),  formatName: "PNG", file);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

The **XYSeries** class is again used as a representation of the chart data and is used to create **XYSeriesCollection** which is then used to create a **JFreeChart** object:



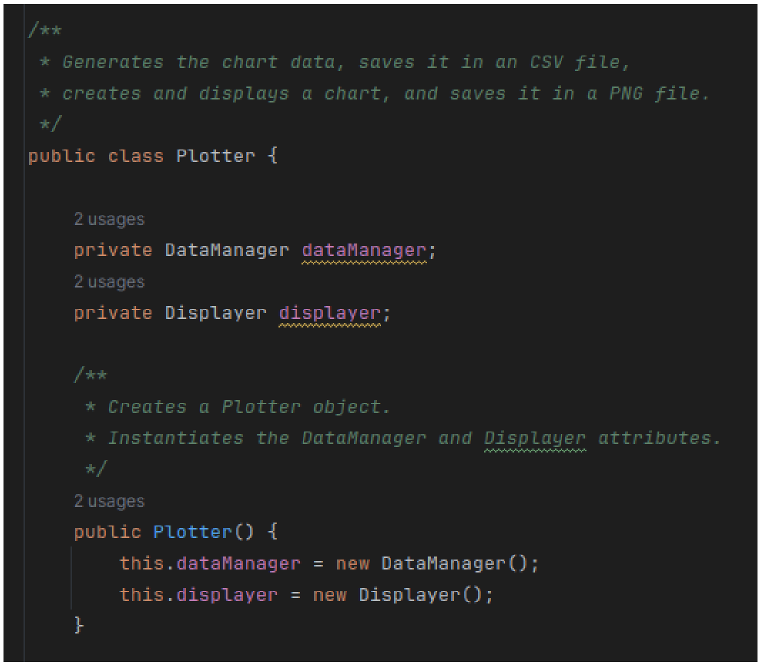
```
/**
 * Generates and displays the plot for the given series and description.
 * @param filename the name of the file.
 * @param series the data series to plot.
 * @param description the description of the plot.
 */
3 usages
public void plotAndDisplay(String filename, XYSeries series, String description) {
    // create the chart
    XYSeriesCollection dataset = new XYSeriesCollection(series);
    JFreeChart chart = ChartFactory.createXYLineChart(
            description, xAxisLabel: "X", yAxisLabel: "Y", dataset,
            PlotOrientation.VERTICAL, legend: true, tooltips: true, urls: false
    );

    // display and save in a file
    displayChart(chart);
    saveChartAsPNG( filename: filename + ".png", chart);
}
```

The other classes are the same as before:

- **Plotter**: for generating and plotting the original data
- **Salter**: for salting the original data
- **Smoother**: for smoothing the salted data

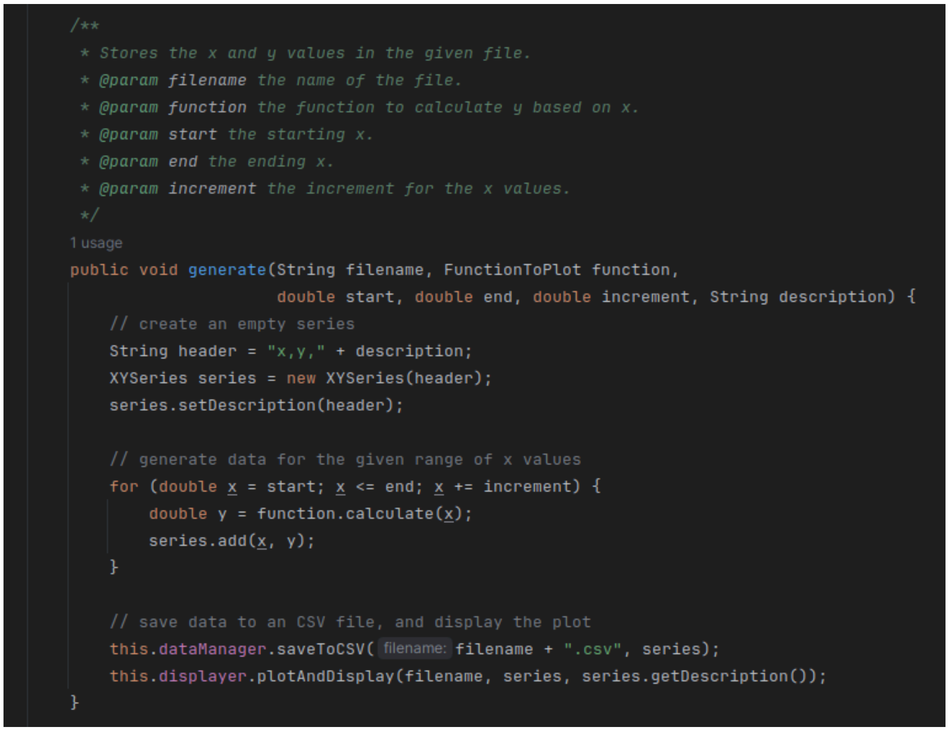
Plotter

A non-arg constructor was added to the class, it instantiates the **DataManager** and **Displayer** attributes:



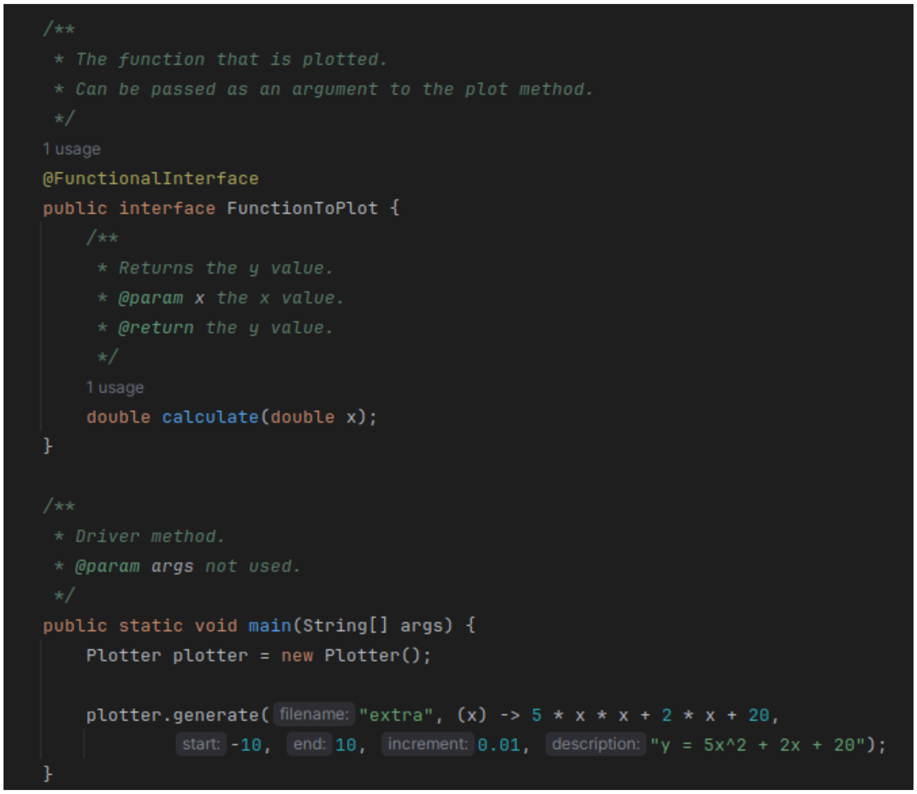
```java
/**
 * Generates the chart data, saves it in an CSV file,
 * creates and displays a chart, and saves it in a PNG file.
 */
public class Plotter {

    2 usages
    private DataManager dataManager;
    2 usages
    private Displayer displayer;

    /**
     * Creates a Plotter object.
     * Instantiates the DataManager and Displayer attributes.
     */
    2 usages
    public Plotter() {
        this.dataManager = new DataManager();
        this.displayer = new Displayer();
    }
```

The **generate** method now delegates to the **dataManager** to store the data in a **csv** file and to **dislpayer** to plot the data:



```java
    /**
     * Stores the x and y values in the given file.
     * @param filename the name of the file.
     * @param function the function to calculate y based on x.
     * @param start the starting x.
     * @param end the ending x.
     * @param increment the increment for the x values.
     */
    1 usage
    public void generate(String filename, FunctionToPlot function,
                         double start, double end, double increment, String description) {
        // create an empty series
        String header = "x,y," + description;
        XYSeries series = new XYSeries(header);
        series.setDescription(header);

        // generate data for the given range of x values
        for (double x = start; x <= end; x += increment) {
            double y = function.calculate(x);
            series.add(x, y);
        }

        // save data to an CSV file, and display the plot
        this.dataManager.saveToCSV( filename: filename + ".csv", series);
        this.displayer.plotAndDisplay(filename, series, series.getDescription());
    }
```

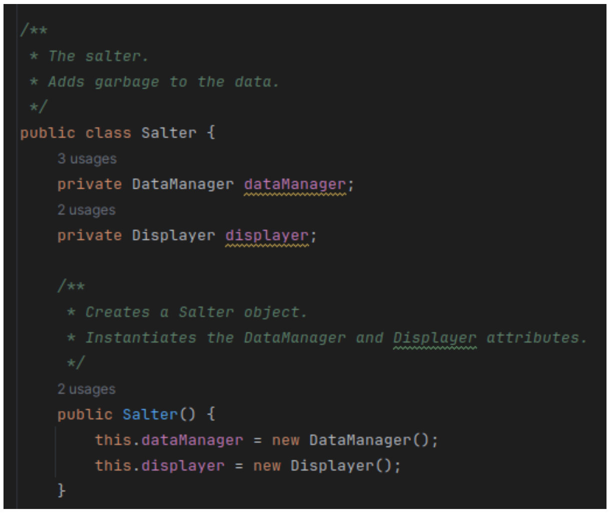
The **FunctionToPlot** interface and the **main** method remain unchanged:



```
/**
 * The function that is plotted.
 * Can be passed as an argument to the plot method.
 */
1 usage
@FunctionalInterface
public interface FunctionToPlot {
    /**
     * Returns the y value.
     * @param x the x value.
     * @return the y value.
     */
    1 usage
    double calculate(double x);
}

/**
 * Driver method.
 * @param args not used.
 */
public static void main(String[] args) {
    Plotter plotter = new Plotter();

    plotter.generate( filename: "extra", (x) -> 5 * x * x + 2 * x + 20,
            start: -10, end: 10, increment: 0.01, description: "y = 5x^2 + 2x + 20");
}
```

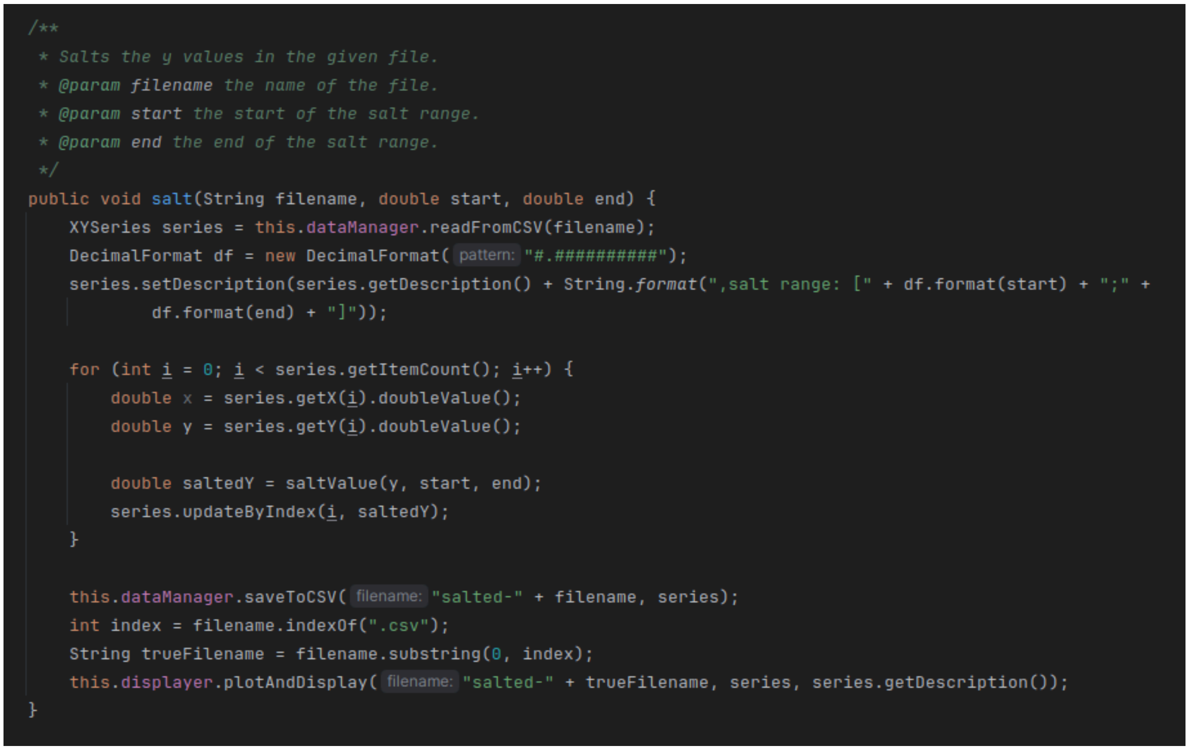
Salter

This class too has the **Displayer** and **DataManager** attributes:



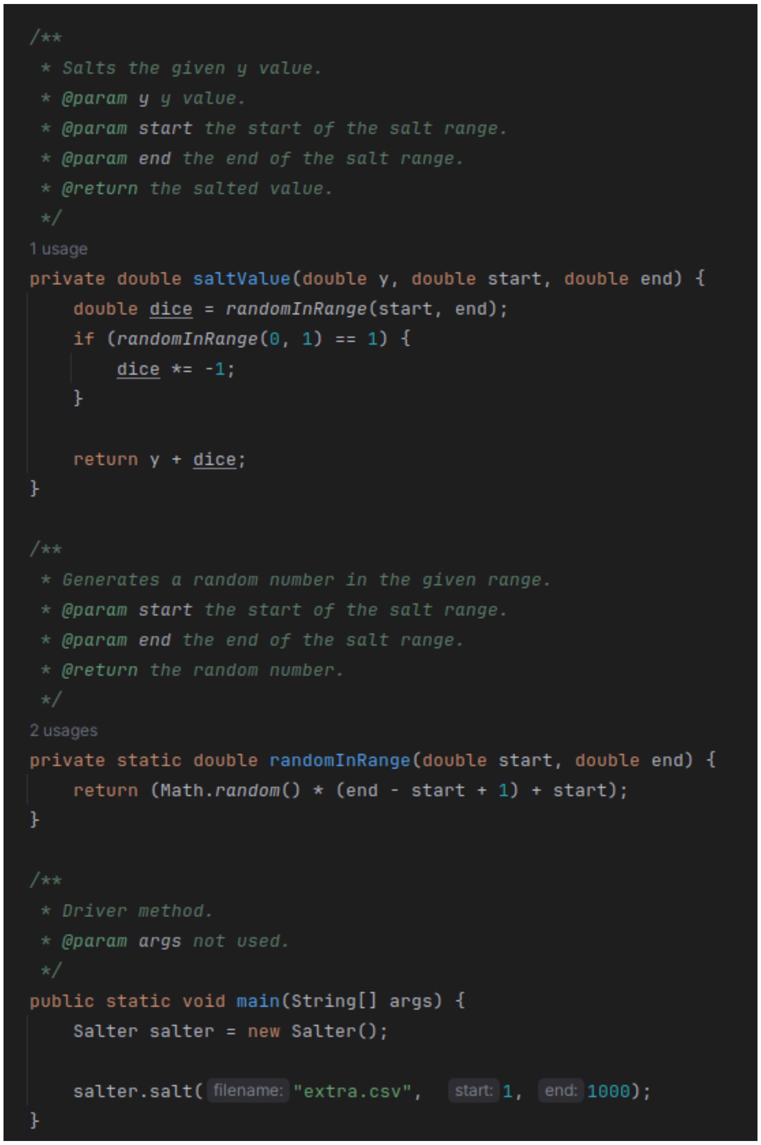
```
/**
 * The salter.
 * Adds garbage to the data.
 */
public class Salter {
    3 usages
    private DataManager dataManager;
    2 usages
    private Displayer displayer;

    /**
     * Creates a Salter object.
     * Instantiates the DataManager and Displayer attributes.
     */
    2 usages
    public Salter() {
        this.dataManager = new DataManager();
        this.displayer = new Displayer();
    }
```

And later delegates to those classes:



```java
/**
 * Salts the y values in the given file.
 * @param filename the name of the file.
 * @param start the start of the salt range.
 * @param end the end of the salt range.
 */
public void salt(String filename, double start, double end) {
    XYSeries series = this.dataManager.readFromCSV(filename);
    DecimalFormat df = new DecimalFormat("#.##########");
    series.setDescription(series.getDescription() + String.format(",salt range: [" + df.format(start) + ";" +
            df.format(end) + "]"));

    for (int i = 0; i < series.getItemCount(); i++) {
        double x = series.getX(i).doubleValue();
        double y = series.getY(i).doubleValue();

        double saltedY = saltValue(y, start, end);
        series.updateByIndex(i, saltedY);
    }

    this.dataManager.saveToCSV("salted-" + filename, series);
    int index = filename.indexOf(".csv");
    String trueFilename = filename.substring(0, index);
    this.displayer.plotAndDisplay("salted-" + trueFilename, series, series.getDescription());
}
```

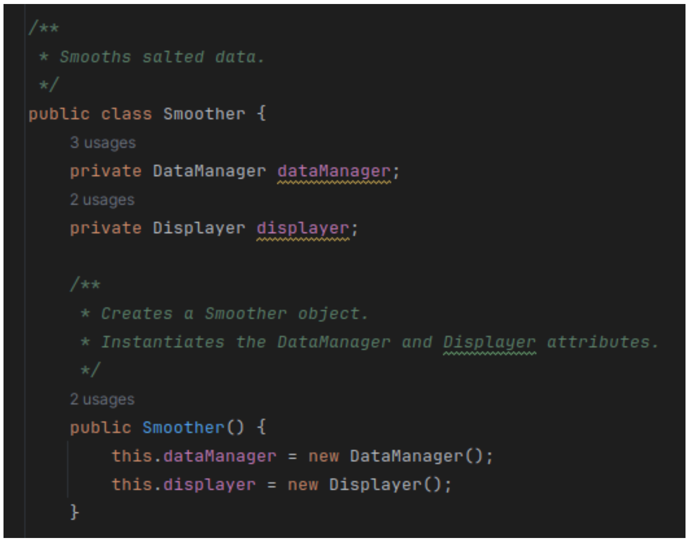
The rest of the methods remain unchanged:



```
/**
 * Salts the given y value.
 * @param y y value.
 * @param start the start of the salt range.
 * @param end the end of the salt range.
 * @return the salted value.
 */
1 usage
private double saltValue(double y, double start, double end) {
    double dice = randomInRange(start, end);
    if (randomInRange(0, 1) == 1) {
        dice *= -1;
    }

    return y + dice;
}

/**
 * Generates a random number in the given range.
 * @param start the start of the salt range.
 * @param end the end of the salt range.
 * @return the random number.
 */
2 usages
private static double randomInRange(double start, double end) {
    return (Math.random() * (end - start + 1) + start);
}

/**
 * Driver method.
 * @param args not used.
 */
public static void main(String[] args) {
    Salter salter = new Salter();

    salter.salt( filename: "extra.csv",  start: 1,  end: 1000);
}
```

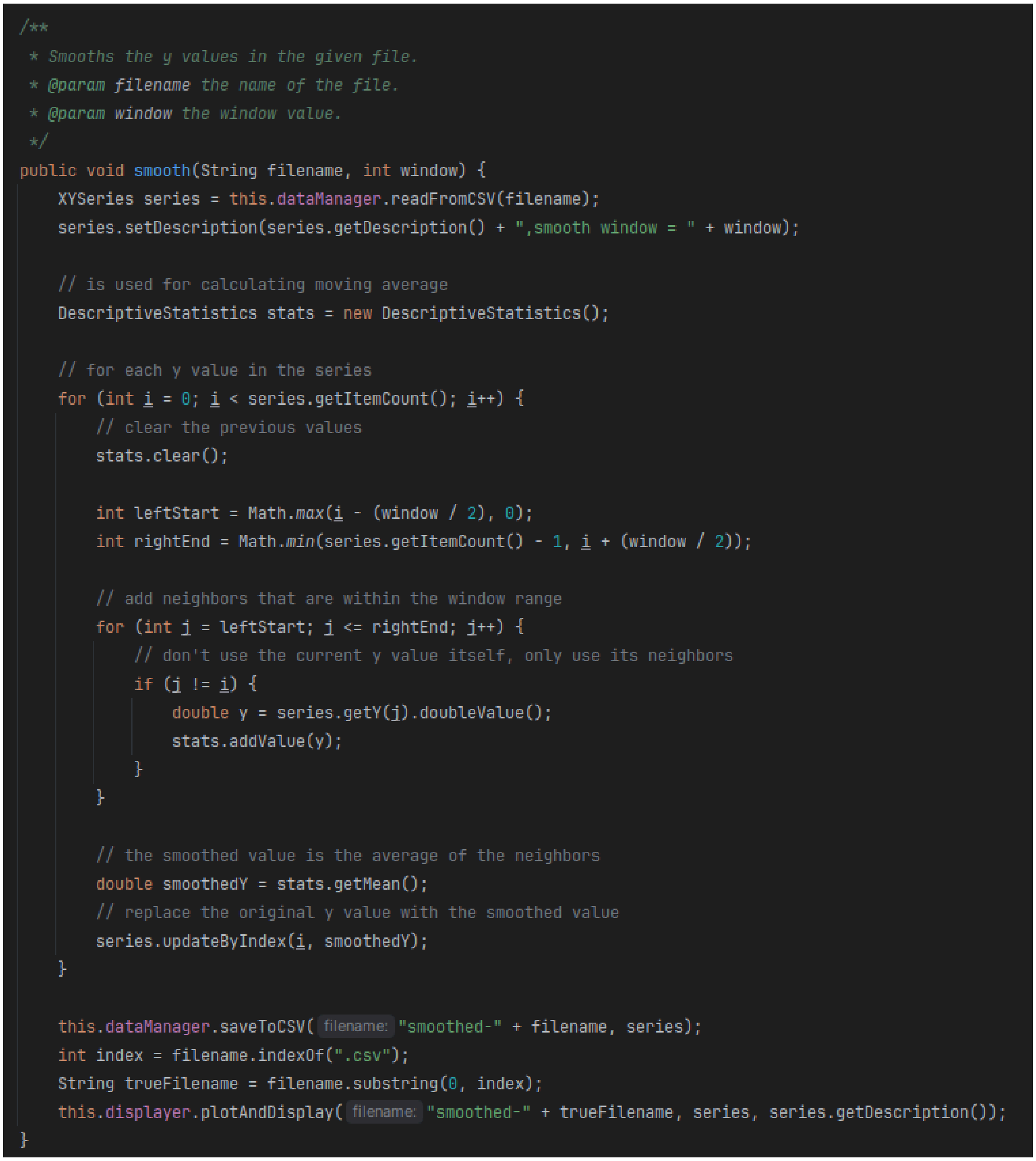
Smoother

The constructor logic is the same as in the other classes:



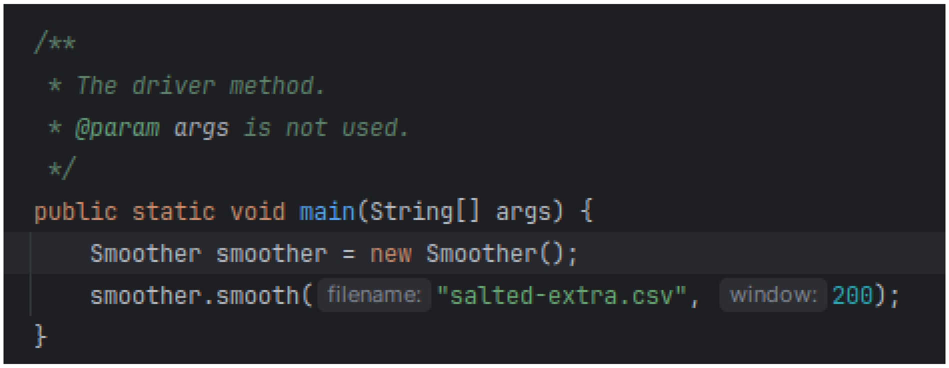
```
/**
 * Smooths salted data.
 */
public class Smoother {
    3 usages
    private DataManager dataManager;
    2 usages
    private Displayer displayer;

    /**
     * Creates a Smoother object.
     * Instantiates the DataManager and Displayer attributes.
     */
    2 usages
    public Smoother() {
        this.dataManager = new DataManager();
        this.displayer = new Displayer();
    }
```

The **smooth** method makes use of the **DescriptiveStatistics** class from the **Apache** library to calculate the moving average values:



```java
/**
 * Smooths the y values in the given file.
 * @param filename the name of the file.
 * @param window the window value.
 */
public void smooth(String filename, int window) {
    XYSeries series = this.dataManager.readFromCSV(filename);
    series.setDescription(series.getDescription() + ",smooth window = " + window);

    // is used for calculating moving average
    DescriptiveStatistics stats = new DescriptiveStatistics();

    // for each y value in the series
    for (int i = 0; i < series.getItemCount(); i++) {
        // clear the previous values
        stats.clear();

        int leftStart = Math.max(i - (window / 2), 0);
        int rightEnd = Math.min(series.getItemCount() - 1, i + (window / 2));

        // add neighbors that are within the window range
        for (int j = leftStart; j <= rightEnd; j++) {
            // don't use the current y value itself, only use its neighbors
            if (j != i) {
                double y = series.getY(j).doubleValue();
                stats.addValue(y);
            }
        }

        // the smoothed value is the average of the neighbors
        double smoothedY = stats.getMean();
        // replace the original y value with the smoothed value
        series.updateByIndex(i, smoothedY);
    }

    this.dataManager.saveToCSV( filename: "smoothed-" + filename, series);
    int index = filename.indexOf(".csv");
    String trueFilename = filename.substring(0, index);
    this.displayer.plotAndDisplay( filename: "smoothed-" + trueFilename, series, series.getDescription());
}
```

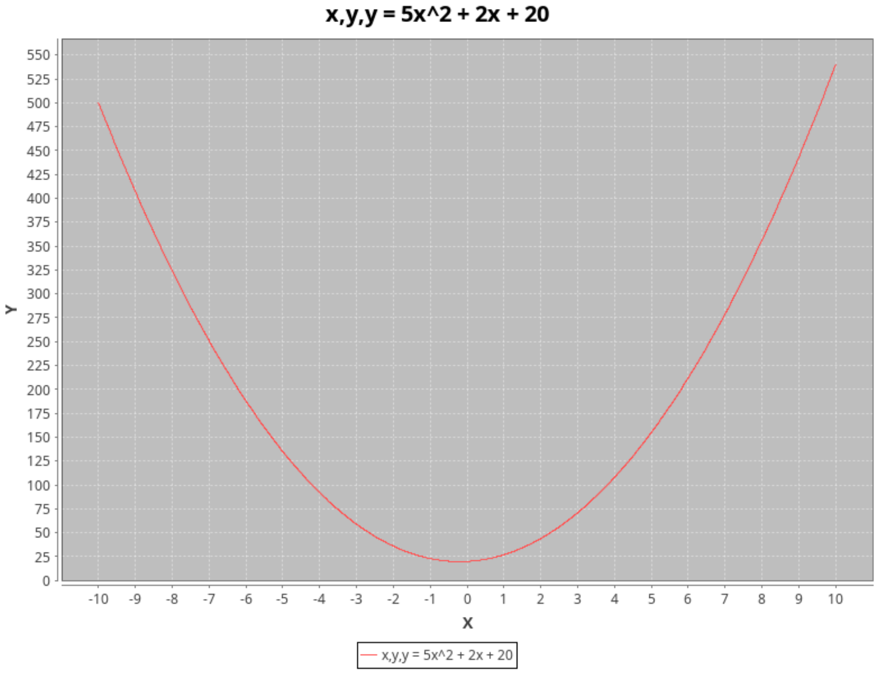
The logic in the **main** method remained unchanged:



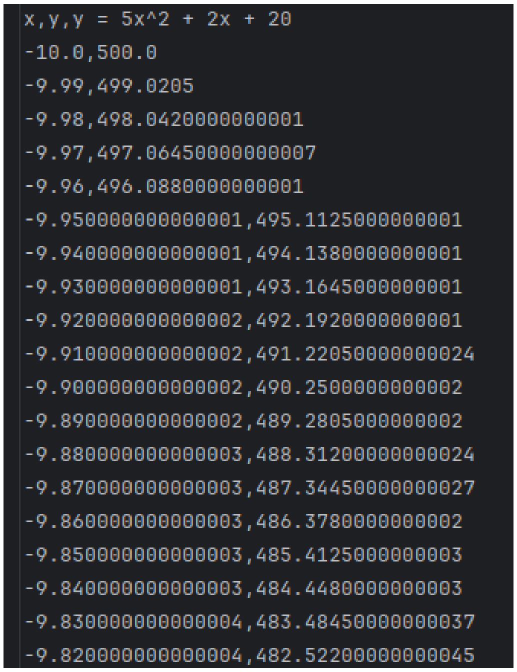
```
/**
 * The driver method.
 * @param args is not used.
 */
public static void main(String[] args) {
    Smoother smoother = new Smoother();
    smoother.smooth( filename: "salted-extra.csv",  window: 200);
}
```

Generated charts and data

The **Plotter** class generates the following chart:

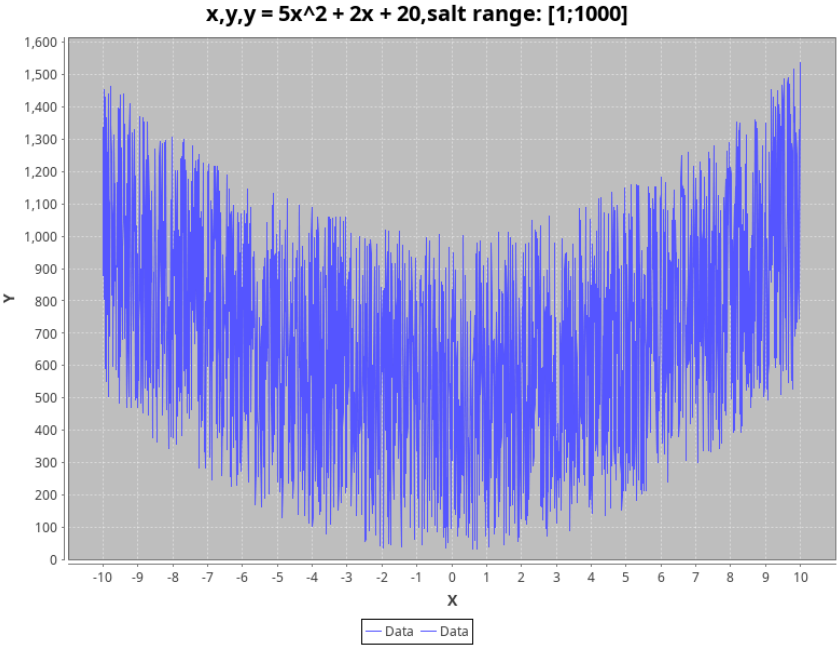


And the following **csv** file:

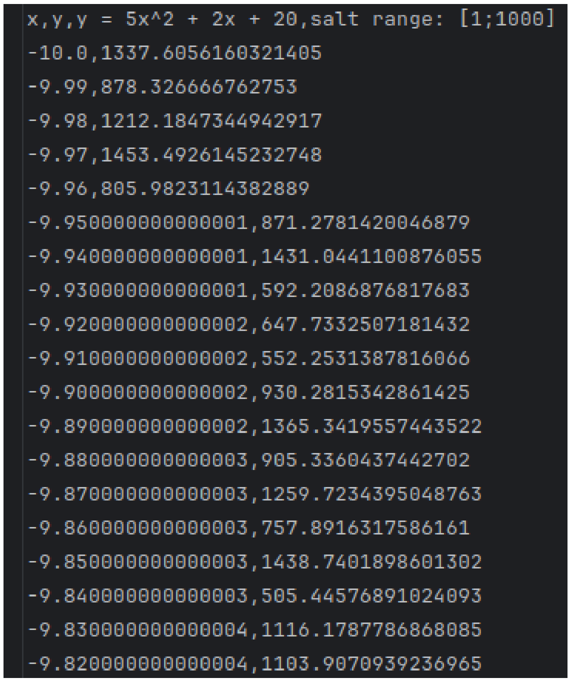


```
x,y,y = 5x^2 + 2x + 20
-10.0,500.0
-9.99,499.0205
-9.98,498.0420000000001
-9.97,497.06450000000007
-9.96,496.0880000000001
-9.950000000000001,495.1125000000001
-9.940000000000001,494.1380000000001
-9.930000000000001,493.1645000000001
-9.920000000000002,492.1920000000001
-9.910000000000002,491.22050000000024
-9.900000000000002,490.2500000000002
-9.890000000000002,489.2805000000002
-9.880000000000003,488.31200000000024
-9.870000000000003,487.34450000000027
-9.860000000000003,486.3780000000002
-9.850000000000003,485.4125000000003
-9.840000000000003,484.4480000000003
-9.830000000000004,483.48450000000037
-9.820000000000004,482.52200000000045
```

The **Salter** class generates the following chart:

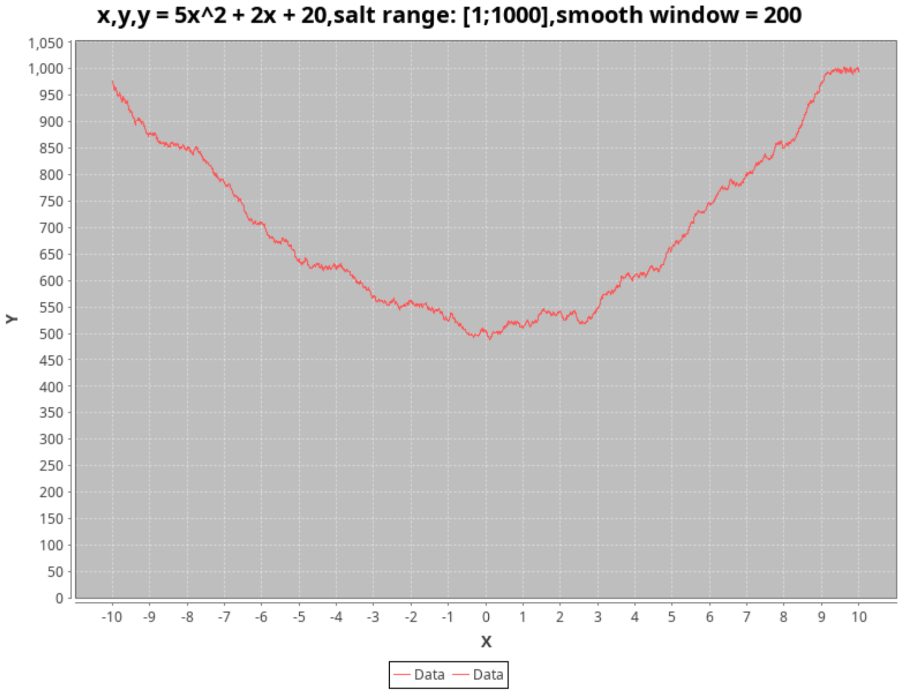


And the following **csv** file:

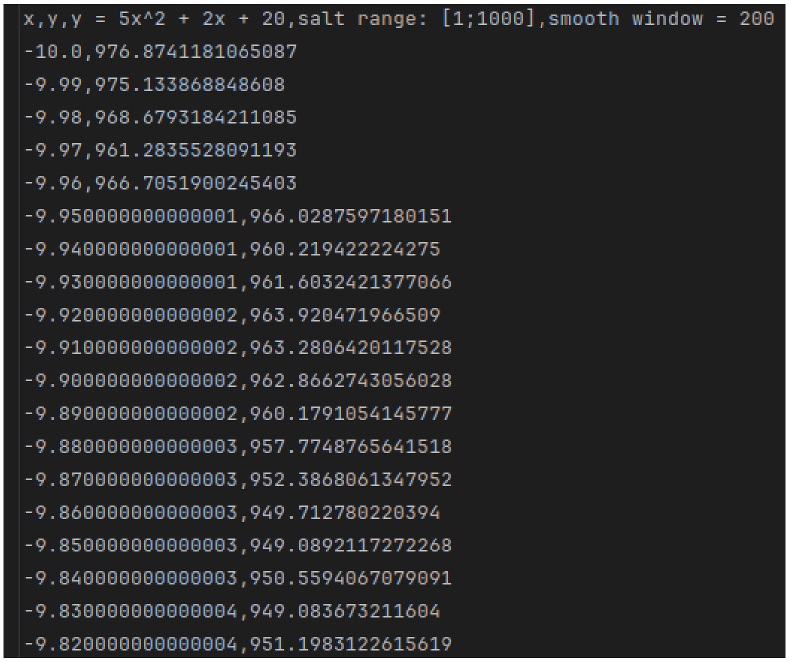


```
x,y,y = 5x^2 + 2x + 20,salt range: [1;1000]
-10.0,1337.6056160321405
-9.99,878.326666762753
-9.98,1212.1847344942917
-9.97,1453.4926145232748
-9.96,805.9823114382889
-9.950000000000001,871.2781420046879
-9.940000000000001,1431.0441100876055
-9.930000000000001,592.2086876817683
-9.920000000000002,647.7332507181432
-9.910000000000002,552.2531387816066
-9.900000000000002,930.2815342861425
-9.890000000000002,1365.3419557443522
-9.880000000000003,905.3360437442702
-9.870000000000003,1259.7234395048763
-9.860000000000003,757.8916317586161
-9.850000000000003,1438.7401898601302
-9.840000000000003,505.44576891024093
-9.830000000000004,1116.1787786868085
-9.820000000000004,1103.9070939236965
```

The **Smoother** class generates the following chart:



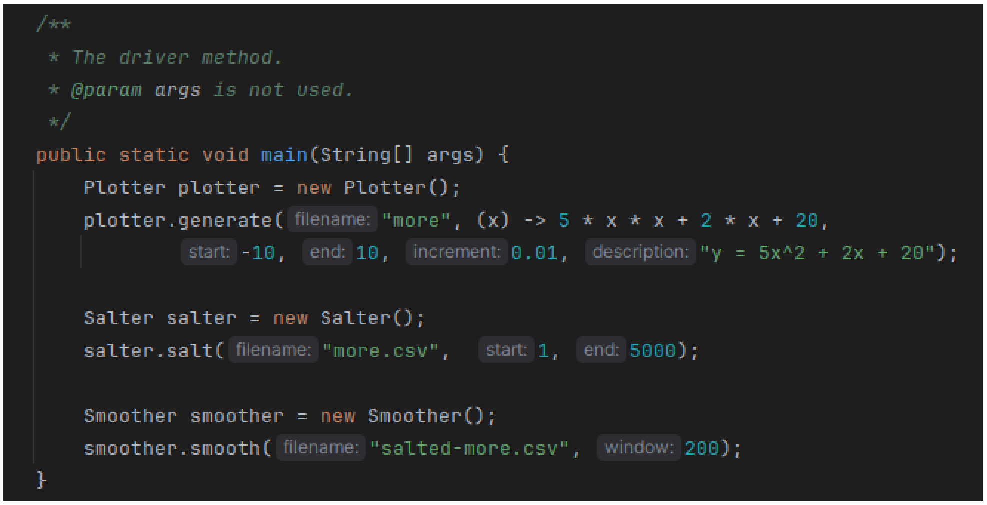
And the following **csv** file:



```
x,y,y = 5x^2 + 2x + 20,salt range: [1;1000],smooth window = 200
-10.0,976.8741181065087
-9.99,975.133868848608
-9.98,968.6793184211085
-9.97,961.2835528091193
-9.96,966.7051900245403
-9.950000000000001,966.0287597180151
-9.940000000000001,960.219422224275
-9.930000000000001,961.6032421377066
-9.920000000000002,963.920471966509
-9.910000000000002,963.2806420117528
-9.900000000000002,962.8662743056028
-9.890000000000002,960.1791054145777
-9.880000000000003,957.7748765641518
-9.870000000000003,952.3868061347952
-9.860000000000003,949.712780220394
-9.850000000000003,949.0892117272268
-9.840000000000003,950.5594067079091
-9.830000000000004,949.083673211604
-9.820000000000004,951.1983122615619
```

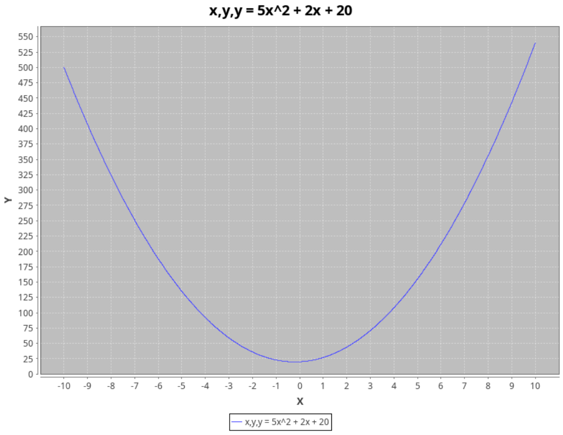
All the **csv** files listed above are truncated and include only the first 20 rows.

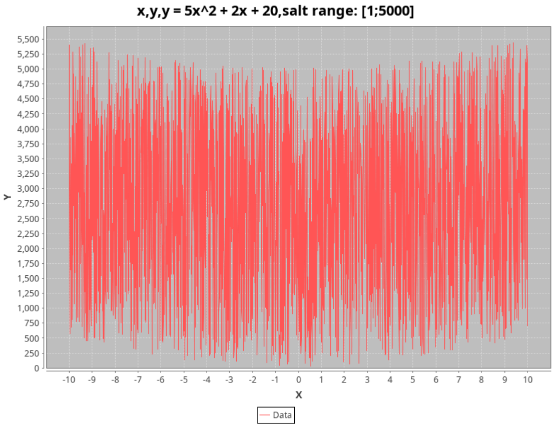
Changing the **main** method in the **Smoother** class as follows:

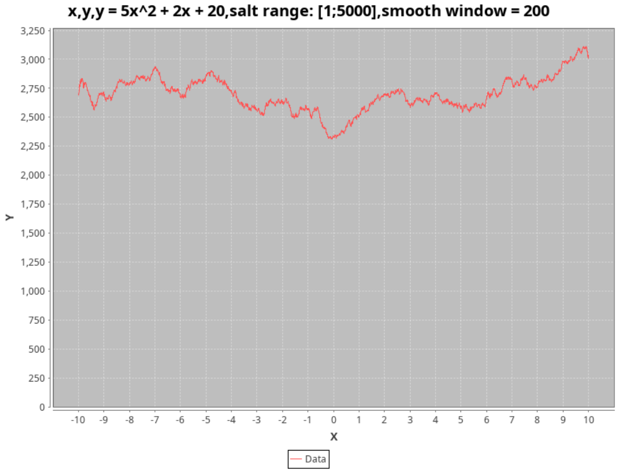


```java
/**
 * The driver method.
 * @param args is not used.
 */
public static void main(String[] args) {
    Plotter plotter = new Plotter();
    plotter.generate( filename: "more", (x) -> 5 * x * x + 2 * x + 20,
             start: -10,  end: 10,  increment: 0.01,  description: "y = 5x^2 + 2x + 20");

    Salter salter = new Salter();
    salter.salt( filename: "more.csv",   start: 1,  end: 5000);

    Smoother smoother = new Smoother();
    smoother.smooth( filename: "salted-more.csv",  window: 200);
}
```

Results in the following charts:

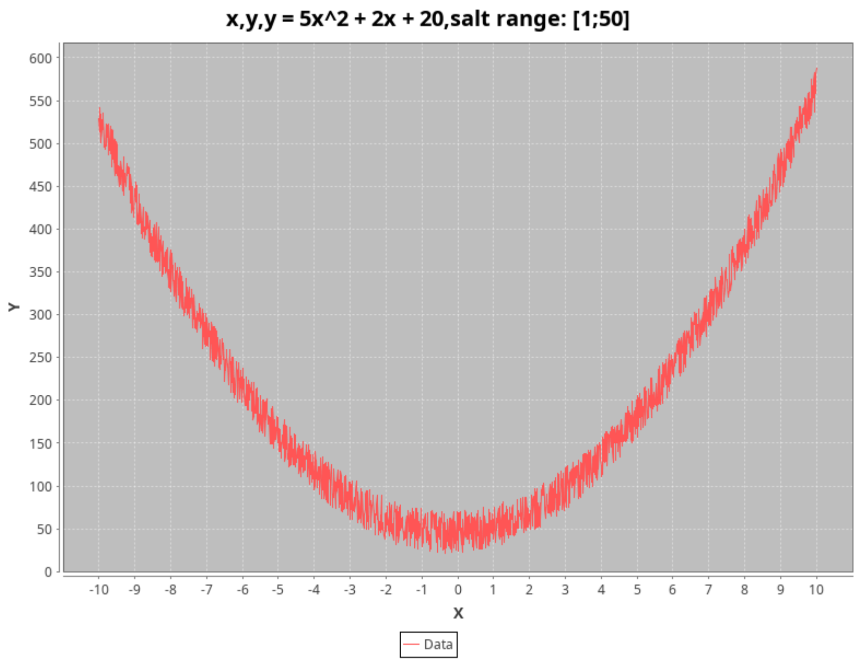


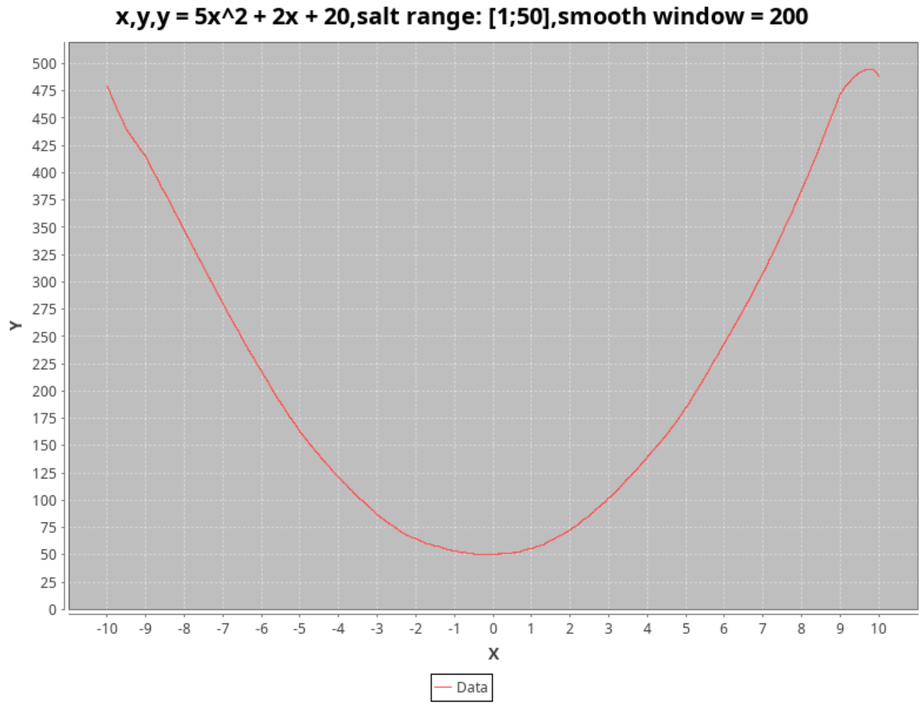




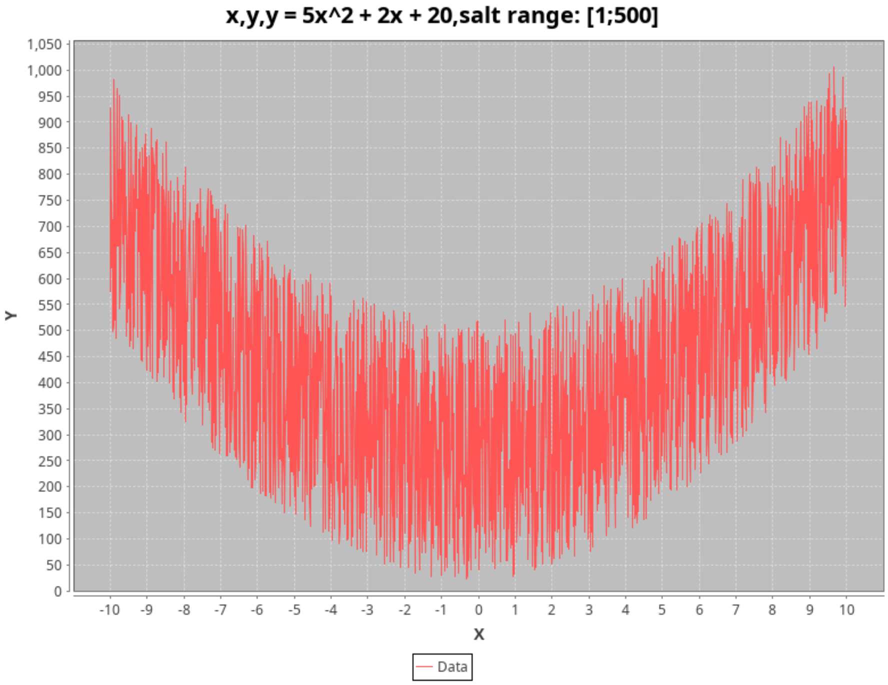
The wider salt range produces an unrecognizable salted chart.

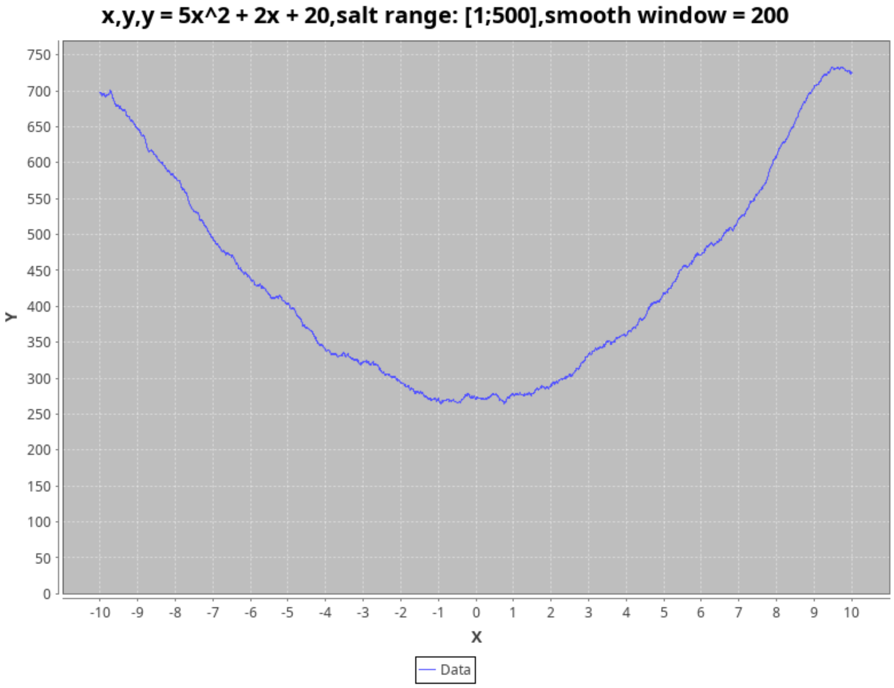
With the upper salt range of 50:



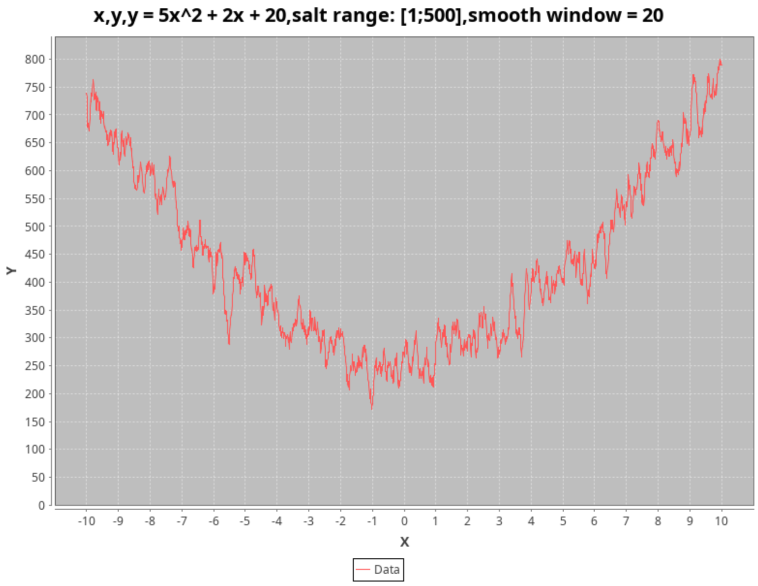


With an upper salt range of 500:





With a window of 20:



The smaller window made the graph less smooth and more similar to the salted chart.