Creating and Reading CSV Files with X and Y Values in Java

Melvin Vazquez Andujar

April 26th, 2024

About Program:

**The objective of this program is to gain a deeper understanding of how CSV’s work in Java and to gain a better understanding of real-life problems such as having messed up data and being able to fix that data so that it is readable. The first part of this project was to create a plotting class that plots a formula decided by me and puts the x and y values inside a CSV file. The second objective is to call that CSV inside another class called Salter, and this class gets the y values and messes up the data to replicate a real-life scenario where the data received is messy and filled with trash. The last objective is to mend the data and try to get as close as possible to the original data graph, and this process is called Smoothing.**

Table of Contents

[Results: 1](#_Toc165029074)

[Plotter: 1](#_Toc165029075)

[Creating Plotter Class: 1](#_Toc165029076)

[Methods: 1](#_Toc165029077)

[Salter: 1](#_Toc165029078)

[Creating Salter Class: 1](#_Toc165029079)

[Methods: 1](#_Toc165029080)

[Smoothing: 1](#_Toc165029081)

[Creating Smoothing Class: 1](#_Toc165029082)

[Methods: 1](#_Toc165029083)

[Conclusion: 1](#_Toc165029084)

# Results:

##### Small Dataset: 30 Values

##### A screenshot of a graph Description automatically generatedA screen shot of a graph Description automatically generatedA screenshot of a graph Description automatically generated

#### Medium Sized Dataset: 100 Values, smoothed has two different graphs to showcase when it is smoothed only once and when it is smoothed three times. The accompanying data is next to the graph.

A graph on a screen

Description automatically generated

A graph on a white background

Description automatically generatedA screenshot of a graph

Description automatically generatedA graph showing a line

Description automatically generated

A screenshot of a computer

Description automatically generatedA graph with a line

Description automatically generated

#### Big Dataset: The smoothed data in this one uses 5 as window value and smooths the data 3 times, This one has 200 values

A graph on a white sheet

Description automatically generatedA graph on a white background

Description automatically generatedA screen shot of a graph

Description automatically generated

# Plotter:

## Creating Plotter Class:

Before I started coding the plotter class, I had to first do research on how to create a Comma Separated Values file and how I would insert data into these files that I created. So, I looked up a video and used a couple websites to help me in the creation of this class. This class uses the quadratic formula, , to find y-values, and the x-values are created in a separate method that asks the user for input on the minimum and maximum of their graph. I created a total of three different plotter CSV’s that handled a small, medium, and large dataset.

### Methods:

#### quadraticFormula(double start, double end, double increment):

In this method a total of three parameters are being set which is the start of the first x value, the end of the last x-value, and by how much we are incrementing by. With those three parameters we set up a for loop that begins at the first x given and increments until x equals the last x-value given. Inside the for loop we input the equation through the use of a y variable, the built in method for power, and variables that were set at the beginning of the class, which . After each iteration the x and y value each get added to an ArrayList that is specific to the value being put in.

#### bounds():

In this method we call the Scanner class because I wanted to get the userInput here and in this class is where we get the start, end, and increment parameters that are being used in the Quadratic Formula method. For this method I used a while loop and a try catch that asks the user to input a value for where they would like to start, end, and by how much they would like to increment by. The catch is there to make sure that the input is what the code is looking for and not an invalid input, and the while loop is to make sure that part of the code keeps running until all fields are answered correctly.

#### checkBounds(double start, double end, double increment):

In this method I use the parameters that we get from Bounds to check whether it makes logical sense, so it checks that the start is not greater than the end, and that it is incrementing by more than 0. If all is good then this method calls the csvPlotter method at the end.

#### csvPlotter():

This is the hardest method because I had to do research on how to create a CSV file and how to input array values into the file in the correct format that it is readable. I first had to understand which build in methods to use and how to use them correctly and in the correct format. But once I learned how to do that, it isn’t too difficult and not too long a process in creating. I import the built-in method for File, printWriter, and exceptions before starting the code. After I do that I create a try catch that runs through the code until an error occurs, however, if no error occurs then the code runs fully through. This code is first creating a file called, dataPlotter.csv, and then using printWriter to write to that file. Then we use a variation of the printf method but for the printWrite and input x and y into the file. The error will never occur because the file is created inside the try block and the catch is set to see if a file is found, which it will always find.

# Salter:

## Creating Salter Class:

The difficulty of creating this class is a lot more compared to the plotter class because I had to figure out how to read a csv file and how to input the x values into an ArrayList and the y values into another ArrayList. This took me the longest time since I just could not find anything on how to do it, so I kept trying to do it through looking at different ways online and trying to use that but in the way I wanted to do it. It took a while but eventually I was able to find a way to import that data into the two ArrayList and I could finally start coding the rest of the methods I would use for this code. Same as Plotter I created three different CSV’s that handles small, medium, and a large dataset.

### Methods:

#### saltingGraph():

This method is basically a copy of the plotter one in which it creates the CSV for the salter and inputs the x and new y value inside the CSV.

#### setArrayList():

This method is the one that took the longest because I had to figure out how to read the CSV file. So, to start I had to import three new built in methods called BufferedReader, FileReader, and IOException. The file reader is what helps read the content of the file and buffered reader is what helps to get specific content inside the file, like reading each line. After that I had to figure how to add the values into an ArrayList and the way to do that was by first setting a variable x and a variable y equal to the value of where the index of the file starts at. Then I had to parse it to convert it to a double since it is a string, then with the two ArrayList I created at the beginning of the class I add that y and x to their respective list. This method is put inside a while loop and two try catch blocks. The first try catch is to check that there is no problem with finding file or an IO exception where class gets interrupted by something. Then inside that try is a while loop and another try catch. Inside of these two is the code that parses the string and puts it in an ArrayList. The catch in the second is there to check if there is a value that can’t be parsed since it is not a number, which does happen since the heading is set to be X-Values and Y-Values inside the CSV. After the code finishes running it calls two methods, the setRange() and then the messedUpData method.

#### setRange():

This method is just like the one in the plotter class which gets userInput to find the range in which the user wants to mess up the data from, so if they put 30-70, it will get a random value from 30-70 and add or subtract from the value we are looking at in the index. But this method only gets the range, the math is done in another method.

#### messedUpData(double min, double max):

In this method the math mentioned in setRange() is done here, using the two parameters from setRange() I use a do while loop that runs the code while the count I made does not equal the size of the y points ArrayList. Inside the do part of the loop is a for loop and if statement. The for loop runs through the y point ArrayList and the if statement checks whether the num I created in the method is 1 or 2. If it is 1 then we will add to the index at that point using a random number from the range given, otherwise, it will subtract if it is 2.

# Smoothing:

## Creating Smoothing Class:

This class was not hard to make compared to the other two because I already knew how to write and read the CSV, so it was mostly just copy and pasting the code from the salter class and just changing some small details to match what the code is looking for. This class is to fix the data from salter to try and match it back to the original data. The only hard part for this code was figuring out the logic behind the math and figuring out how to put that logic into code.

### Methods:

#### getData():

This method is the same as the setArrayList() method in the Salter class, just changed the path since we are looking for a different CSV, but other than that everything is basically the same.

#### smoothingData(int windowVal, int smoothingTimes):

This method uses the parameters to get the average value of y at each index. The window value goes left and right however many times it tells the code and adds up all the numbers. After it is all added up, the count divides by however many numbers were added and gets the average at that point. That point is then added to an ArrayList and this keeps repeating until an average is found for every point and that counts as smoothing the data once. This process will repeat as many times as the smoothingTimes tells the code and the y points used is the one that is added to the ArrayList so that it gets the average of the first averaged points. This method uses three for loops, first one is for the smoothing, the second is for the y-points ArrayList, and the third is for the window value. I had a hard time figuring out how to do the third for loop, but when I talked to a friend, they suggested subtracting the index of the second for loop from window value so the index starts however far back it should, and to add window value to j so that we go to the right however many times it tells the code to. After that the math portion was very easy and the code was done.

#### setRangeAndTime():

This method is getting the windowVal and smoothingTimes parameters that will be used inside the smoothingData method. These parameters are what tell the code how many values to look at from the index to find an average and then smoothing times is how many times we want to smooth the data. After we find the values from userInput we call the method for smoothing the data.

#### smoothingCSV():

This method is the same as the salter and plotter in which it creates the CSV File, only difference is that the y points being put inside the file is the smoothed points.

# Conclusion:

This code was fun to do since it taught me real life skills that I might use in my field whenever I get a job in the future, and it was interesting playing with the code because I noticed that depending on the size of the y-values if there are huge values you need to mess up the data through a bigger range or the data won’t change as much. Plus, I also noticed that the smoothing changes depending on how many values you look at and how many times you smooth the data. I did an experiment where I did the same number of smoothing times but changed the number of values looked at, and there was a substantial difference from each other. The more values I looked at was more accurate to the original compared to the one with less values looked at. However, this won’t work if we try to add all values since there is a big difference in size of the numbers since, I am using quadratic formula and one of the numbers gets squared. Other than that learning about CSVs was very fun and taught me more ways to go about my coding projects and making it easier to get data from just inputting it straight to a file from the code.

# Bibliography:

This is a list of websites, and videos I visited trying to learn CSVs and methods to do some math portions and logic

CSDoctor. “Writing to a CSV File Example with PrintWriter in Java.” *YouTube*, YouTube, 1 Apr. 2020, www.youtube.com/watch?v=dHZaqMmQNO4&t=73s.

Educative. “How to Generate Random Numbers in Java.” *Educative*, www.educative.io/answers/how-to-generate-random-numbers-in-java. Accessed 26 Apr. 2024.

“Java DO-While Loop with Examples.” *GeeksforGeeks*, GeeksforGeeks, 22 Mar. 2023, www.geeksforgeeks.org/java-do-while-loop-with-examples/.

Lee, Alex. “Java: Read a CSV File into an Array #49.” *YouTube*, YouTube, 18 June 2020, www.youtube.com/watch?v=-Aud0cDh-J8.

TheodoreTheodore                      5544 bronze badges, et al. “How Can I Make the Code Catch the Error When the Input Provided Is Not a Number?” *Stack Overflow*, Theodore, 1 Mar. 1965, stackoverflow.com/questions/56523561/how-can-i-make-the-code-catch-the-error-when-the-input-provided-is-not-a-number.

w3SChools. “Java Exceptions - Try...Catch.” *Java Exceptions (Try...Catch)*, w3Schools, 1999, www.w3schools.com/java/java\_try\_catch.asp.