**CSC110 Class Final Project**

Carlos Hernandez

Department of Computer Science, Gateway Community College, Arizona

**Date:** March 10, 2021

**Technical Report:** NUIM-CS-TR2002-05

**Software Design Description (SDD)**

Cover Page

Table of Contents

1 PROJECT DESCRIPTION

1.1 Design Overview

1.2 Current Exceptions/Errors

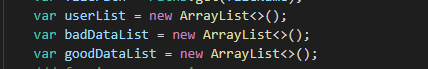
2 Conclusion/Additional Information

1. **Project Description**

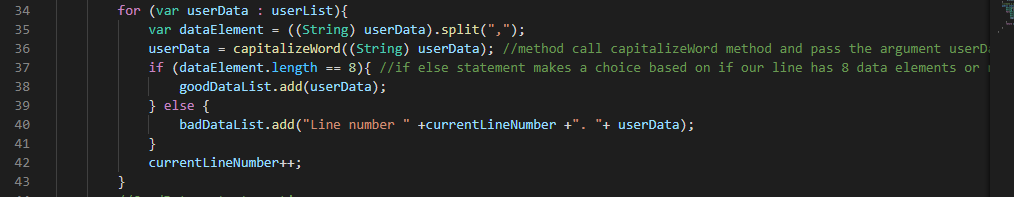
Using Visual Studio Code and Java, we will create a functional program that takes a csv, excel, or text file, and breaks into 2 data files using different formatting techniques based on a requirement. This time we’ll be using a hard requirement of accepting ONLY 8 different data elements separated by a comma per line, any more or less than the required data elements will get sorted into it’s own data file named “BadData.txt”. As an additional requirement for this project, we’ll be making every single word capitalized in the data lines before storing them in a data structure/outputting, and making sure to report the line number of the data lines that make it into the badData.

* 1. **Design Overview**

In order to design this project, we needed multiple data structures to store different data depending on if they fit the requirement or not. This was the only way we could differentiate and make different outputs based on the same input file path. We used ArrayLists for every data need we had. We set up 3 different ones, one to store the generic data, one to store the good data and one to store the bad data. The generic one would be used to build the other 2 and the 2 would be used to output/write different files.

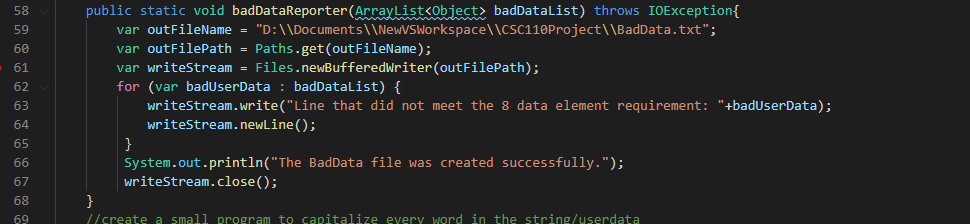


We also used basic Paths.get(fileName) and while (line!= null) loops to add all the required data into the generic ArrayList first, excluding the header. We would then make a var currentLineCounter before making a for loop, that says for every line in the data the loop would run. The loop would split the data, using a String.split(“,”), and then capitalize the userData variable defined in the for loop using a method call for a small method later in the program that basically just capitalizes every word in a string. We then use the split data by using a nested if else statement, with it making sure the amount of data elements is 8 using the substring.length method. Using that method, we check if the length is equal to 8, using the == operator, and if it is, we add it to the good ArrayList, if it’s not, we add it to the bad ArrayList.



As you can see, the bad data also gets the currentLineCounter, and a small string with “Line number” to make it more readable when it gets added to the ArrayList. This comes in handy because it will report/write the lines that failed the requirement later in the program using that logic. At the end we increment the previously defined currentLineNumber so when the loop runs again for each new line, the number increases by 1 to match it.

After this we made the first basic output section for the good data, it uses a newBufferedWriter to write a split version of the goodDataList or the ArrayList we used to store good data, and then writes a “|” to split the data/format it. Originally, I was going to make a switch case until I realized I needed both of them to run always, and then I realized I couldn’t return the 2 different dataLists needed to pass along to the methods, so I created the first output inside the aggregateGoodData method. After that is done, and it creates the file, it will let the user know using a System.out.println() function, and then move on using a method call to another method named the badDataReporter. This method gets passed along the badDataList, or the ArrayList we used to store the data that didn’t fit the requirements. Instead of having to split the data like GoodData, this one just needs to fetch the ArrayList, convert it to String using the for loop, and then write the new string variable in a newBufferedWriter.write (); method. We add the explanation before for readability, but it’s not required.



Finally, we close the bufferedWriter using writeStream.close, and then move on with our day. I hope this explained the design choices and why I made them.

* 1. **Current Exceptions/Errors**

Currently, this program we built functions and compiles with no problems. There are no Exceptions or Errors.

1. **Conclusion/Additional Information**

Overall, this program hit all the marks I needed it to, and it does it in a pretty concise way too. Using only one method call in the main method. If I was to do it in the future, I would like to learn about a better logic path, as I relied a lot on for loops in order to get the job done and while not bad, I would like to get some efficiency in there. Also, I think my object-oriented programming could be better. I would like to make this program more readable if I had a second chance on it, I went for functionality, but I think OOP with Java is important regardless. Regardless, I think I did pretty well, and it’s a statement of how far I’ve come in this semester when in the beginning I wasn’t declaring my class and struggling with System.out.println() methods.