Lab 7 – Files (12 pts)

Lab Objectives

- Be able to use file streams for I/O
- Be able to write a loop that reads until the end of a file
- Be able to implement an accumulator and a counter

~~Deliverables~~

~~This lab has three tasks. When you have all tasks done, run the report in Blackboard. The report is a Blackboard test with short-answer, file-response, multiple-answer, and other types of questions.~~

~~In the report, you may be asked to provide code segments, Java source code files (must have extension *.java*), screenshot of program execution, files in PDF format, and your analysis of the results.~~

~~If a short answer question requests a code segment, please ensure that your input is readable: all **new lines and indents** are in place.~~

~~Screenshots in your report **must show a full screen**, so your computer can be identified. Please resize your IDE panels the way that the required dialog or output is visible along with the source code. Show as much source code as possible.~~

~~NOTE:~~

- ~~Use **Blackboard only** to submit your work; **no email** submission unless your instructor directs it.~~
- ~~If Blackboard gives you multiple submission attempts (usually three), the **last one** will be evaluated and graded.~~
- ~~**No late submissions**, **no changes** in your submission after the due date.~~

Introduction

You will be introduced to file input and output. Before doing this lab please read section 4.10 of the textbook and p.p.1-5 of additional reading: *Unit 8. Files and input/output streams* from *Lecture Notes for Introduction to Programming* by Diego Calvanese.

We will read a file, line by line, converting each line into a number. We will then use the numbers to calculate the mean and standard deviation.

First, we will learn how to use file output to get results printed to a file.

Next, we will use file input to read the numbers from a file and calculate the mean. The approach for file input that we will use is different from what is in the textbook and the lecture slides. This alternative approached is based on the stream input, not on `Scanner` class and it is discussed in suggested additional reading.

Finally, we will see that when the file is closed, and then reopened, we will start reading from the top of the file again so that we can calculate the standard deviation.

An important comment on the file names: make sure that your code and files you are working with are all in the same folder (directory). Then `javac` (or IDE Dr. Java) should have no problem with opening files. If you are on IDE Eclipse, the file to open must be in the project directory at the same level as folders *bin* and *src*.

If you are experiencing problems, you may need to pass in the absolute path of the file. On Windows, if file *Numbers.txt* is located in your *Desktop* folder, the absolute path could be *C:/Users/YourName/Desktop/Numbers.txt*.

To manage the absolute path if you are prompting the user to enter the file name, the user must type in

- for Windows: *C:/Users/YourName/Desktop/Numbers.txt* ,
- for MacOS: */Users/YourName/Desktop/Numbers.txt* .

To manage the absolute path if you are hard coding the file name, e.g. `FileReader file = new FileReader("`*file name*`");` you must pass in the absolute file path like this:

- for Windows: `FileReader file = new FileReader("C:\\Users\\YourName\\Desktop\\Numbers.txt");`
- for MacOS: `FileReader file = new FileReader("//Users//YourName//Desktop//Numbers.txt");`

The absolute path is individual for each file and above examples only illustrate the idea.

Task #1 Writing Output to a File (4 pts)

1. Copy the files *StatsDemo.java* and *Numbers.txt* as directed by your instructor. Correct syntax errors if any, and improve programming style when necessary (indents, newlines, etc.).
2. First, we will write output to a file:
   a. Create a `FileWriter` object passing it the filename *Results.txt* (don't forget the needed `import` statement).
   b. Since you are using a `FileWriter` object, add a `throws` clause to the `main` method header.
   c. Create a `PrintWriter` object passing it the `FileWriter` object.
   d. Print the current value of mean and standard deviation to the output file using a three-decimal format, labeling each.
   e. Close the output file.
3. Compile, debug, and run. You will need to type in the filename ***Numbers.txt***. You should get no output to the console but running the program will create a file called ***Results.txt*** with your output. The output you should get at this point is: **mean = 0.000, standard**

deviation = 0.000. This is not the correct mean or standard deviation for the data, but we will fix this in the next tasks.

Task #2 Calculating the Mean (4 pts)

1. Now we need to add lines to allow us to read from the input file and calculate the mean.
    a. Create a `FileReader` object passing it the filename.
    b. Create a `BufferedReader` object passing it the `FileReader` object.
2. Write a priming read to read the first line of the file.
3. Write a loop that continues until you are at the end of the file.
4. The body of the loop will:
    a. convert the line into a double value and add the value to the accumulator
    b. increment the counter
    c. read a new line from the file
5. When the program exits the loop close the input file.
6. Calculate and store the mean. The mean is calculated by dividing the accumulator by the counter.
7. Compile, debug, and run. You should now get a mean of **77.444**, but the standard deviation will still be **0.000**.

Task #3 Calculating the Standard Deviation (4 pts)

1. We need to reconnect to the file, so we can start reading from the top again.
    a. Create a `FileReader` object passing it the filename.
    b. Create a `BufferedReader` object passing it the `FileReader` object.
2. Reinitialize the sum and count to 0.
3. Write a priming read to read the first line of the file.
4. Write a loop that continues until you are at the end of the file.
5. The body of the loop will:
    a. convert the line into a double value and subtract the mean, store the result in difference
    b. add the square of the difference to the accumulator
    c. increment the counter
    d. read a newline from the file.
6. When the program exits the loop close the input file.
7. The variance is calculated by dividing the accumulator (sum of the squares of the difference) by the counter. Calculate the standard deviation by taking the square root of the variance (Use the `Math.sqrt` method to take the square root).
8. Compile, debug, and run. You should get a mean of **77.444** and standard deviation of **10.021**.