COMP2402B (Fall 2015)

Assignment 1 of 5

Revised September 14, 2015

Caution

You will be uploading your submission to cuLearn – this assingnment does not have a written component.

Only the latest submission received before the deadline will be used to determine your mark on this assignment.

You must adhere to the following rules (as your submission may be subjected to automatic marking system):

- ***Download** the compressed file **"COMP2402-F15-A1-(Start Skeleton).zip"** from cuLearn.*
- ***Retain the directory structure** (i.e., if you find a file in subfolder "comp2402a1", you must not remove it).*
- ***Retain the package directives** (i.e., if you see "package comp2402a1;" in a file, you must not remove it).*
- ***Do not rename any of the methods already present** in the files provided.*
- ***Do not change the visibility of any of the methods provided** (e.g., do not change private to public).*
- ***Do not change the main method of any class**; on occasion the main method provided will use command line arguments and/or open input and output files – you must not change this behaviour).*

*Please also note that **your code may be marked for efficiency as well as correctness** – this is accomplished by placing a hard limit on the amount of time your code wil be permitted for execution. If you select/apply your data structures correctly, your code will easily execute within the time limit, but **if your choice or usage of a data structure is incorrect**, your code may be **judged to be too slow** and **it may receive a grade of zero**.*

*You are expected to **demonstrate good programming practices at all times** (e.g., choosing appropriate variable names, provide comments in your code, etc.) and **your code may be penalized if it is poorly written**.*

Instructions

This assignment is about using the Java Collections Framework to accomplish some basic text-processing tasks. These questions involve choosing the right abstraction (Collection, Set, List, Queue, SortedSet, Map, or SortedMap) to efficiently accomplish the task at hand. The best way to do these is to read the question and then think about what type of Collection is best to use to solve it. There are only a few lines of code you need to write to solve each of them. It should be noted that, unless otherwise specified, **the phrase "sorted order" refers to the natural sorted order on Strings**, as would be exhibited by a call to the String.compareTo(s) method.

Each of the following questions refers to an input that will take the form of a file that was redirected into your program over the command line (i.e., java YourProgramName < input.txt); the start skeleton downloaded from cuLearn will include source that will allow you to read this data.

Read both "Part I – Coding" and "Part II – Testing" before starting this assignment.

Part I – Coding

1. [5 marks] Read the input one line at a time and write each line to the output **(in the same order they appeared in input)** if and only if it is not a duplicate of some previous input line. Ensure so that a file with a lot of duplicate lines does not use more memory than what is required for the number of unique lines.

2. [5 marks] Read each of the lines of the input and output them in sorted order. Duplicate lines should be printed only once. Ensure that a file with a lot of duplicate lines does not use more memory than what is required for the number of unique lines.

3. [5 marks] Read each of the lines of the input and output them in sorted order. Duplicate lines should be printed the same number of times they occur in the input. Ensure that a file with a lot of duplicate lines does not use more memory than what is required for the number of unique lines.

4. [5 marks] Read the first 50 lines of input and then write them out in reverse order. After that, read the next 50 lines and then write them out in reverse order. Repeat these steps until there are no more lines left to be read. (Top clarify, your output will start with the 50th line, then the 49th, then the 48th, etc. until the 1st line is reached. This line will be followed by the 100th line, followed by the 99th, etc. until the 51st line is reached.) Please note that your code should never store more than 50 lines at any given time. **When you reach the end of the file make sure to write out the remaining lines in reverse order, even if there were fewer than 50 lines at the end of the file.**

5. [5 marks] Read the entire input file one line at a time and then output all lines in sorted order according to line length, with the shortest lines first. In the case where two lines have the same length, resolve their order using the standard "sorted order" (i.e., the natural sorted order on Strings).

6. [5 marks] Read the entire input one line at a time and then output the even numbered lines (starting with the first line, which should be considered line 0) followed by the odd-numbered lines.

7. [5 marks] Read the input one line at a time. At any point after reading the first 42 lines, if some line is blank (i.e., a string of length 0) then output the line that occured 42 lines prior to that one. For example, if line 242 is blank, then your program should output line 200. This program should be implemented so that it never stores more than 43 lines of the input at any given time. No bonus marks will be awarded for recognizing the significance of the number 42.

8. [5 marks] Read the entire input one line at a time and randomly permute the lines before producing the output. To clarify, the lines should appear unchanged, but in a random order. The random order in which your lines are printed is expected to change virtually every time you run your program.

Part II – Testing

Create a collection of test input files (to be accompanied by a collection of expected output files) for the first seven questions of "Part I – Coding" **(n.b., [2 marks] each for a total of [14 marks])**. These files should be placed in the "tests/" directory of the archive and should be named "testX-YY.ZZZ" where X is the number of the question being tested, YY is the two digit test number (a number between 01 and 99, inclusively) ZZZ is the word "in" or "out", depending upon whether the file is an input file or an expected output file. To clarify, test1-01.in and test2-03.out, would be the input file for the first test of the first question and the expected output file of the third test of the second question, respectively.

After you submit, a correct implementation will be run against each pair of files to make sure that that the expected output file (e.g., test1-01.out) corresponds to the test input file (e.g., test1-01.in). If this task completes successfully, an incorrect implementation will be run against these test files as well. In order to receive full marks for this question, one of your test files should expose the error in the implementation, so that the implementation either crashes or fails to produce correct output. The maximum number of tests you are allowed to submit for each of the questions being tested is 99 (i.e., the two digit test number is a number between 01 and 99, inclusively) but this is much than required/expected. Please note the error in each of the incorrect implementations can be exposed with a relatively small input (i.e., 1000 lines or less). Please also note that there is an upper bound on the maximum size of the zip file you can upload, so don't exceed this limit.