Assignment1 (15 pts).

Exercise 1. (8 pts) This exercise is designed to help familiarize the student with choosing the right data structure for the right problem. If implemented, the parts of this exercise should be done by making use of an implementation of the relevant interface (Stack, Queue, Deque, Map, SortedMap, USet, or SSet) provided by the Java Collections Framework.

Solve the following three problems by reading a text file one line at a time and performing operations on each line in the appropriate data structure(s). Your implementations should be fast enough that even files containing a million lines can be processed in a few seconds. Different solutions are possible your solution will be evaluated based on correctness and efficiency.

For reasons of unification and readability, and in order to spare you the effort of elaborating code related to I/O streams, please reuse and customize the file ExampleInpoutOutputStreams.java provided with this assignment.

In this exercise, the term "the input" refers to a text file or a sequence of lines you get from the standard input. You can notice this is implemented in the provided Java file ExampleInpoutOutputStreams.java.

1.1 (2 pts) Read the first 50 lines of the input and then write them out in reverse order. Read the next 50 lines and then write them out in reverse order. Do this until there are no more lines left to read, at which point any remaining lines should be output in reverse order.

In other words, your output will start with the 50th line, then the 49th, then the 48th, and so on down to the first line. This will be followed by the 100th line, followed by the 99th, and so on down to the 51st line. And so on.

Your code should never have to store more than 50 lines at any given time.

1.2 (2 pts) Imagine each input line is numbered, starting from 0. Read the input one line at a time. Output the odd-numbered lines in ~~descending~~ ascending order of line number, then output the even-numbered lines in ascending order of line number. Your implementation should never store more than n/2 +1 lines where n represents the number of lines of the input.

1.3 (4 pts) You are asked to develop a simple application that provides the following functionalities.

1. Read the input one line at a time and, than provide an output consisting in:

a. a text file named ?CompFile.txt (e.g. File1CompFile.txt) including all the input lines with no duplicates and ordered according to their first occurrences in the input. For example the sequence of lines consisting of karim, bob, lol, bob, alice, bob, lol will be transformed to karim, bob, lol, alice.

b. a text file named ?UncompArg.txt (e.g. File1UncompArg.txt) that stores supplementary information allowing to retrieve the input before processing as required in the following functionality.

2. Process any two files output of the previous functionality (e.g. File1CompFile.txt and File1UnompArg.txt) producing the original input that may contain duplicated lines (e.g. File1.txt).

Exercise 2 (3 pts) Priority Queue implementation.
In this exercise you are asked to develop a reusable implementation of the Abstract Data Type (ADT) Priority Queue (PQ). Your implementation must reuse existing implementations of ADT of the Java Collections Framework mentioned at http://docs.oracle.com/javase/tutorial/collections/implementations/index.html.

We assume that PQ accepts any element having a property named priorityValue that can be accessed as a public attribute value. This property represents the priority of an element. The element having the lowest priority will be removed whenever the PQ instance receives a remove() call. When more than one elements have the same priority, the PQ ADT allows to remove any of them after a remove() with no supplementary constraints. We assume also that the frequency of removal is about 1/10 of the frequency of adding elements, so your implementation may consider this to improve efficiency minimizing the overall cost of theses operations.

Exercise 3 (2 pts) A Dyck word is a sequence of +1's and -1's with the property that the sum of any prefix of the sequence is never negative. For example, +1, -1, +1, -1 is a Dyck word, but +1, -1, -1, +1 is not a Dyck word since the prefix +1-1-1 < 0. Describe how to use push(x) and pop() operations of a stack in order to determine if a sequence of +1's and -1's is a Dyck word. Your description should consist in a commented pseudocode or java code.

Exercise 4 (2 pts) A matched string is a sequence of {, }, (,), [, and] characters that are properly matched. For example, ``{{()[]}}" is a matched string, but this ``{{()]}" is not, since the second { is matched with a]. Show how to use a stack so that, given a string of length n, you can determine if it is a matched string in O(n) time. Your answer should consist in a commented pseudocode or Java code.

The submission is through cuLearn in one zip file. The solution of each exercise must be included in a separate directory named “Part” followed by the exercise number. Thus, the directories names are: Part1.1, Part1.2, Part1.3, Part2, Part3 and Part4. The name of your zip file must be a concatenation of the student full name and Assign1-2402.zip (e.g. BrandonThomsonAssign1-2402.zip).