CSCD 300 Homework 2
Streaming Text Processing, Tokenization, Counting Word Occurrence, List Sorting, Comparator, Comparable and file I/O.

Due: Jan.26 2014 23:59 pm

Turn in: On EWU Canvas, CSCD300→ Assignments→HW2→Submit.
Please put all your source code(.java) files and output files, the PDF file that shows your unit test together into a zip file. Name the zip file with your last name followed by first initial plus hw2.zip. For example, smithjhw2.zip is for John Smith.

If you forget to include your source code in the zip file, you get a zero credit for this homework. If your code shows a compile-time error, you get a zero credit.

Problems Description
You are required to use a linked list (singular or doubly) to count word occurrence in a large text file, in which we have arbitrary type of punctuations. The list and its node structure have already been declared in the provided source file. You are required to use a linked list structure, you **could** change the provided MyLinkedList class to a Doubly Linked List. If you use array to store all tokenized words, you get a ZERO credit.

What is provided?

Two text files are provided for processing, testfile1 and testfile2. You can use testfile1 to debug your code because it is smaller. You have to run your program on testfile2 to see if it works. The testfile2 is a big text file, which is a subset of Wikipedia and contains around half of million English words.

You have to implement and use four classes that are included in the **start-up zip file**. They are **MyLinkedList, WordItem, WordProcessor, and Tester**. **Please carefully read the comments on top of each these classes and content in this document**. Please do NOT change the Tester class, the toString() method in MyLinkedList and in WordItem and writeToFile() method in the WordProcessor class, because we like to keep the output format consistent for all the students.

Basic Idea to Extract Word
We assume all words in the provided text are correct English words. Suppose we have a line of text, **without** a line feed at the end(no other terminal characters).
"You are a student. Who's your advisor? i.e.   teacher. I'm your friends."

We can see that English words are delimited by white space or punctuations. Basically, starting from the first letter of a word, we remember a **start** position of a word. As we scan down the next few characters, if we encounter a space or punctuation character, we know that from the **start** position we remembered to the predecessor of the current character, defines an English word.

After we find an English word in the line of text, we increment a counter for that word (if the word has been in the list), and update the list of line numbers at which the word appeared in the original text file. Otherwise we make a new WordItem object and call addOrdered() in the List to insert the object into a proper location in the Linked List. NOTE that, when check whether a word has already existed in the list or not, you are required to use CASE-INSENSITIVE comparison.

After we extracted a word, if its length is one, (contains only one letter) we throw them away, (that is, we will not insert the one-letter word into the linked list) EXCEPT THE WORD "I" and "A".

For example, in the text above, after "Who" we get an apostrophe ', that means we get a English word "who". Then, after we throw away the apostrophe following word 'who', we get a new word starting at letter 's'. Following 's' we got a white space. According to our algorithm, the single letter 's' is a word. But here we throw away this single-letter word. The same is for the 'e' in "i.e. " and 'm' in "I'm".

In order to extract all English words in a piece of text, we have to specifically look at each character in the text from the first line to the last line. It is like processing a stream of ASCII characters.

In this Homework 2, you **CANNOT** use Regular Expression or build-in method split() to extract English words out of text. You have to implement the provided algorithm above yourself. We will have more information in Class about this algorithm.

What you should do?

1) Compile a PDF file that clearly shows that you have done the unit test for each method **you implemented**. Provides at least 3 test cases for each method and show the output for each test case. You have to show me in the PDF file that each method works well as expected.
2) Run your program on the testfile2 file, the big one. You have to generate two text files with required format. The first output file contains word occurrence for each word in that file, and its records are sorted according to words in an alphabetical order. The second output file is also word occurrence for each word in the file, but records are sorted by the word occurrence in descending order. That is, the most frequently used words are listed first in the file. File format are fixed and enforced by the provided toString() method.

Bonus Extra Credit

1) **25 points,** write a subclass that implements Comparator, so that your linked list sort() method can take a Comparator object as a parameter. In this way, you can write one sort method, but you can sort the list according to English Word name or according to word occurrence in one sort method. The

sort() method behaves differently based on the Comparator object. Basically, Comparator class defines how would you compare two WordItem Object.

2) **25 points,** Improve your sort method**.** When the linked list is sorted according to the word occurrence in descending order, if two or more words have a same occurrence, we like to sort **these words (words with the same occurrence)** in an alphabetic order as a second sorting rule.

Test Cases
For the input file testfile1, two output files are provided in the start-up folder. Your task is to calculate the similar output files for testfile2.