SE480 – Software Architecture I
Autumn 2018
Homework 4
Due Date: Sunday, November 4 11:59PM
100 points possible. Will be scaled to 10% of final grade.

Submit your answers to D2L by Sunday, November 4 11:59PM. Late homework will not be accepted.

Assignment Overview

This assignment focuses on the active pipe-and-filter architecture. You will be required to construct and implement a pipe-and-filter architecture for a text processing platform.

You must program this assignment in Java.

This assignment has two parts:

Part 1

Design an *active* pipe-and-filter architecture which reads in a text file, removes stop words (see explanation below), removes all non-alphabetical text, stems words into their root form (from now on referred to as terms), computes the frequency of each term, and prints out the 10 most commonly occurring terms (not including stop words) in descending order of frequency. If there are ties, then print the tied terms in alphabetical order.

The challenge is to complete the processing of a single file as fast as possible, while considering other architectural goals such as maintainability, reuse, and understandability.

SOME parts of the solution include:

- **Data Source**: Input files will be provided. Your program must be able to run on all the following input files without breaking:
  - The Declaration of Independence (`usdeclar.txt`)
  - The complete text of Alice in Wonderland (`alice30.txt`)
  - The text of the King James version of the Bible (`kjbible.txt`)
- **Stopword Removal**: We will be using the "very long list" from `https://www.ranks.nl/stopwords`. For your convenience, I have saved the words into the file `stopwords.txt`.
- **Porter Stemming Algorithm**: You will need to apply the Porter stemming algorithm in order to stem all words to a morphological root (e.g. jumping, jumps, jumped → jump). You can find the description of the algorithm here: `http://snowball.tartarus.org/algorithms/porter/stemmer.html` and Java source code for the algorithm here: `https://tartarus.org/martin/PorterStemmer/java.txt`. I have not tested the above source code to determine if it is suitable – such is the problem of reusing other people's code.
- **Data Sink**: Outputs the 10 most frequently occurring terms listed in descending order of frequency. In the case of a tie, continue outputting all terms tied for the position in alphabetical order.

Note: I have deliberately NOT listed all possible filters here. This is for you to consider.

For part 1 of your assignment, you must submit a fully working solution.

It is acceptable to download and reuse code that you find on the internet for individual filters i.e. Porter's stemming algorithm, data structure for the pipe etc. ALL reused code must be marked as such. It is NOT acceptable to borrow code written by your class mate, or to download the entire code (if you should find it) for the homework. Each person is responsible for developing their own application.

This part must be submitted as a ZIP file to D2L. The ZIP file must contain:

1. The source code for the solution

2. The compiled executable (except for Python)
3. A README explaining how to run your program

Part 2

In the second part of the assignment you will analyze the extensibility and response-time of your solution.

1. **Extensibility**: Explain how your solution would support the following extensibility goal: The customer wishes to redesign the system to handle text files written in languages other than English (each file is one language). The design time modification must take less than one day. The ultimate solution must be configurable automatically at runtime.
2. **Response Time**: Using the "King James version" text file, instrument your code and report on the response time of your application under conditions of no contention. Do not include infrastructure setup time (e.g. loading stopwords), but do include everything related to loading, processing, and saving results. Also evaluate the individual response times of each filter.

   Note that this is going to be a little tricky, because the runtime of each filter is impacted by the performance of its incoming buffer. You will have to figure out how to get around this problem. Some possible solutions include:

   (a) Determining the time needed to process a single element (i.e. read from incoming pipe, process, and output to outgoing pipe), or

   (b) Set up a driver component.

   Discuss your results in at least one paragraph. Identify bottlenecks (pipes, filters, etc.)
3. **Based on your answer from #2 "Response Time"** redesign your solution to improve response time while balancing the need for maintainability and reuse. Implement your solution and deliver a comparative graph to show how it improves performance over the original solution.

Submit this part as a second ZIP file to D2L.