George,

Thank you for the fun project. I enjoyed working on it. I found it an interesting problem to solve.

The source code and output files can be found by cloning the GitHub url:

https://github.com/eleric/DRNLP

The branches are:

eleric/version1

eleric/version2

eleric/version3

Each for the respective exercises.

I used a simple Main class to run the program: dr.nlp.Main(String[] args)

I have created a jar archive to make it easier to run.

.\DRNLP\DRNLP\_jar\java -jar DRNLP.jar <followed by argument strings>

Coded in Java 8.  It will not run on an earlier JVM.

Parameters for version 1

String inputFilename - paths and name of the document file

String outputFilename - path and name of the output file

Parameters for version 2

String inputFilename - paths and name of the document file

String outputFilename - path and name of the output file

String properNounFilename - paths and name of the text file that will list proper nouns found in the document

Parameters for version 3

String inputFilename - paths and name of the zipped file

String outputFilename - path and name of the output file

String properNounFilename - paths and name of the text file that will list proper nouns found in the documents

I made a few assumptions.

1. The document parsed must be UTF-8 encoded.
2. Digits are components of words. For examples would all be words: 19, B2B, U2
3. White space is every non-visible character. For example: space, delete, null
4. Everything not alphanumeric or a white space is considered a punctuation character.
5. Punctuation tokens could be made up of multiple characters. For example: …, ?”
6. White space tokens could be made up of multiple characters as well. For example: space followed by line return would be 1 token.
7. Order matters for sentences in a document and tokens in a sentence.
8. Order does not matter for documents in a folder.
9. Sentences terminate after a termination token is encountered followed by a white space. For example: W.W will not end a sentence.
10. Special case sentence termination if the end of the file is encountered, even without a valid termination token.
11. Words are case sensitive. For example: Bob and bob are treated as not equal.

The program limitations.

1. Punctuation as part of a word will confuse the parser. For example: Mr. will be treated as sentence terminator. C.B.S. will be treated as 3 separate words separated by the punctuation period, then terminate the sentence.
2. Everything is held in memory until done. For large documents or folders, this could potentially cause the program to run out of space. I built the program flexible enough to make it possible swap in implementations that would better handle a larger memory load.

Tests

I wrote upper level integration tests that work with resource files. I considered writing some mock tests, but I don’t think they would have added much value, since programs don’t use a lot of injected services. If this was a production solution, most likely some framework would have been used to support a more service based strategy.

All tests for each version pass. I used JUnit 4 for my unit tests.

Employed strategies.

1. I made Tokens, the smallest unit stored, immutable so they would be inherently thread safe.
2. I came up with the concept of a Master Dictionary. This would store every unique token in one place in memory. A sentence data structure would reference the token word “the”. A new “the” token would not need to be created every time the word is encountered. This does a few things of value. It effectively conserves memory. Also, if at some point meta-data wanted to be attached to tokens, it could be stored here as well.

I thoroughly enjoyed working on this project. Though my dog wasn’t happy I spent most of the weekend working on it, instead of playing with her. I am familiar with Git, but have not used GitHub before, so if you have any problem access my code, let me know.

Thank you,

Marcus Waldman