Project Documentation

for

Lexical Analyzer

Student:

Kevin Kolcheck

Teacher:

Andy Harris

Class:

CSCI 240

Date:

4/25/2016

Contents

[1. Project Overview 3](#_Toc91412612)-4

[1.1 Disclosure 3](#_Toc91412614)

[1.2 Description of project 3](#_Toc91412614)

[1.3 Project title 3](#_Toc91412614)

[1.4 Intended user 3](#_Toc91412614)

[1.5 What problem is being solved? 4](#_Toc91412614)

[1.6 Technologies used 4](#_Toc91412614)

[2. Data Design 4](#_Toc91412648)-5

[2.1 The data 4](#_Toc91412614)

[2.2 Data representation 4](#_Toc91412614)

[2.3 Data persistence 5](#_Toc91412614)

[2.4 Data diagram 5](#_Toc91412614)

[3. UI Design 5-6](#_Toc91412648)

[3.1 Command Line 5](#_Toc91412614)-6

[3.2 UI diagram 6](#_Toc91412614)

[4. Algorithm 6](#_Toc91412648)-11

[5. Revision 11](#_Toc91412648)

# Project Overview

## Disclosure

The idea behind the project is not my own. Earlier today (03/25/16) I had read a couple of really interesting articles that pertain to text analysis. The first article was about Microsoft’s recent failed A.I. tweetbot called Tay. <http://www.forbes.com/sites/kalevleetaru/2016/03/24/how-twitter-corrupted-microsofts-tay-a-crash-course-in-the-dangers-of-ai-in-the-real-world/#6c3256cc32cb>. While not something nearly as complicated, the idea that a piece of software interacts and changes with things people write sounded really cool. While that is not what I am trying to accomplish with this program, the process behind analysing their tweets, is.

The second article I read was about an A.I. that wrote a Sci-Fi book (with a input from the creators) that almost won a literary prize <http://www.latimes.com/books/jacketcopy/la-et-jc-novel-computer-writing-japan-20160322-story.html>.

As I was digging for something in this vein that was more on my level, I came across the base idea for this project on this website: <http://www.sciencebuddies.org/science-fair-projects/project_ideas/CompSci_p022.shtml#summary>. As I worked on the project, the scope changed some and there are features that I added, I felt like I made it my own. For the sake of time, I had to set a stopping point for myself, but I actually have a lot ideas for how I could take this project further.

## Description of project

The intent of this program is to take a piece of writing and pull it apart to measure things about it. Word frequency, total sentence length, word length, and so on. With this information you can do different things like find averages, key words, and by feeding in word lists (I use function words, for example) to find out things about their word choice.

## Title

Lexical Analyser

## Intended user

The user will likely be a typical adult. It is command line driven, so I try to give helpful feedback on how to use the menus. I kept it simple, so just about anyone could use it.

## What problem is being solved?

The project is designed to heavily analyse a piece of writing, so it will be able to extract information about ANY snippet of English writing that would take a person hours or longer to do on their own

## Technologies used

* Java
* IO library, for reading and writing files
* Map/Hashmap library
* String.split(), worth noting this method is the primary means I use to parse strings

# Data Design

## The data

The program reads a .txt file and stores every sentence. From those sentences, lots of different calculations can be made about the individual sentences: total words in the sentence, total characters in the sentence, and can return the number of letters in each word. It also makes many calculations on the text in it’s entirety: total sentences, total words, total characters, word frequency for every unique word, which can be manipulated to return key words (I allow the user to choose whether to include stop words or not) and function words.

## Data representation and aggregation

The program is object oriented (and not just because it’s written in Java).

There are a few data structures at work in this program. I rely on a linked list to store each parsed sentence. It only made sense that I use a linked list for this task, as I have no way of knowing ahead of time how many sentences might be in a piece of writing, so the dynamic nature was the obvious solution to me.

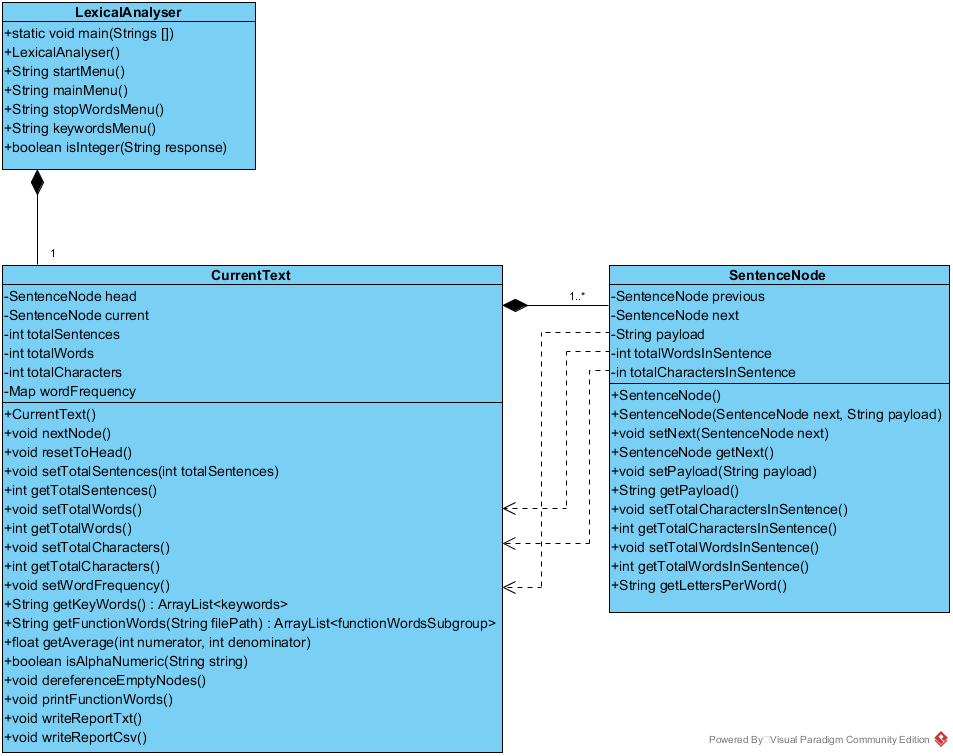
Another data structure I rely on for keeping track of word frequency is a hash map. Each unique word becomes a key in the map, and each key contains the frequency that it appears in the writing. It’s an ideal structure for this problem, because it allows for very fast searching.

I also rely heavily on string arrays to hold parsed strings, typically to iterate through for whatever I am actually after. This is how I will aggregate the words in each sentence into the hashmap.

## Data persistence

The program can generate two different reports, one is a .csv and the other a txt. They both contain much of the same information, except the text file has a each parsed sentence printed out, followed by information about each individual sentence. I didn’t figure this would work well in a csv file, so only the measurements/keywords/function words are in there. I was originally planning on serializing the classes, but after I changed my scope to focus on only one piece of writing at a time, as opposed to the program keeping tabs on multiple authors containing multiple pieces of writing (to compare to each other), it didn’t feel needed anymore.

## Data diagram

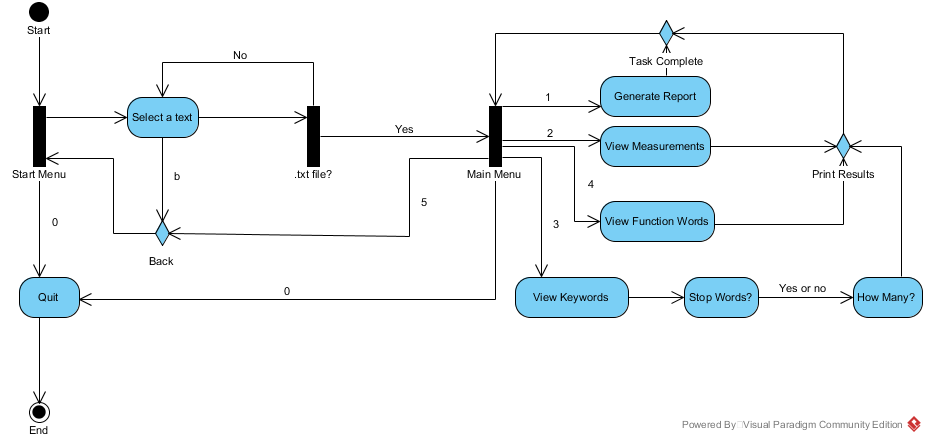


# UI Design

## Command Line

I opted for command line this time around. I wrote the program in Java specifically for the option to have a GUI eventually, but I haven’t had a great deal of time to work on this project, so I wanted to focus my efforts on the functionality of the program. Although after putting the command line in, I don’t know how much time it really saved me. There’s the main menu which has all of the options, but there three smaller menus that allow the user to make other choices. A start menu, which either allows the user to load a txt file, or quit. The other two are associated with making choices for keywords.

## UI diagram



# Algorithm

CLASS: LexicalAnalyser

* Public static void main(String[] args)
  + Create a new instance of LexicalAnalyser
* LexicalAnalyser()
  + Print out description and instructions
  + Call start menu
    - Load a file
    - Quit
  + Display all .txt files in subdirectory text
  + Allow user to type which file he/she wants, or go back to start menu
    - Check to make sure they select a .txt file and nothing else
    - Verify that file exists, if so, continue
  + Try and load the .txt file to a Buffered reader to parse line by line
  + While loop, read until end of the file
    - Split each line into an array of tokens based on standard sentence ending marks (.!?)
    - Iterate over the list of tokens based on the length of the array
      * Assign each string (which will mostly be full length sentences) in the string array to payload in its own node
      * Calculate words in that sentence and save it to the node
      * Calculate total characters in the sentence and save it to the node
      * Increment CurrentText’s total sentences by 1
  + One loop is done, check for empty nodes
  + Now set CurrentText’s total word and total character count
  + Set wordFrequency hashmap to contain unique words as keys, with the value reflecting how often it appears in the text
  + Call main menu
    - 1) Print Reports
      * Print a report in a .txt file and a .csv file
    - 2) View Measurements
      * Print the following: total sentences
      * Total words
      * Unique words
      * Average # of words per sentence
      * Average word length
    - 3) View Keywords
      * Call stopWordsMenu and see if the user wants stop words included or excluded from results
      * Ask the user how many words to return
        + Call isInteger() to check and make sure it’s legit
      * Return keyword results
    - 4) View Function Words
      * Call CurrentText’s printFunctionWords method to display all function words
    - 5) Back
      * Go back to start menu
    - 0) Quit
* String startMenu()
  + Print instructions
  + 1) Select a text
  + 0) Quit
  + Return user’s response
* String mainMenu()
  + Print instructions
  + 1) Print Reports
  + 2) View Measurements
  + 3) View Keywords
  + 4) View Function Words
  + 5) Back
  + 0) Quit
  + Return user’s response
* String stopWordsMenu()
  + Print instructions
  + 1)Yes
  + 2)No
  + Return user’s response
* String keywordsMenu()
  + Print instructions
  + Return user’s response
* boolean isInteger(String response)
  + use a boolean to test if the user’s string is valid
  + return result

CLASS: CurrentText

* Private properties
  + SentenceNode head
  + SentenceNode current
  + totalSentences
  + totalWords
  + totalCharacters
  + Map<String, Integer> wordFrequency
* CurrentText()
  + Instantiate new SentenceNode
  + Set current node
  + Set head to current node
  + Set totalSentences, totalWords, totalCharacters, to 0
* void nextNode()
  + Check if current node’s next property is null.
    - If it isn’t, change current to next
    - If it is, create a temporary node to store the current
    - Current gets a new node
    - Save the location of that node to a temporary node
    - Add new node location to previous nodes “next”
    - Switch back to the new node so that you are on the tail
* Void resetToHead()
  + set current node’s location to the location of head
* void setTotalSentences(int totalSentences)
  + set property this.totalSentences to totalSentences
* int getTotalSentences()
  + return totalSentences
* void setTotalWords()
  + loop through linked list of nodes starting at head and return node’s totalWordsInSentence property, adding them all together
  + set this.totalWords to sum
* int getTotalWords()
  + return totalWords
* void setTotalCharacters()
  + loop through linked list of nodes starting at head and return node’s totalCharactersInSentence property, adding them all together
  + set totalCharacters to sum
* int getTotalCharacters()
  + return totalCharacters
* void setWordFrequency()
  + loop through words and put them as keys in the Hashmap, increment the value everytime the same key appears.
  + Delete any empty characters that made it into map
* String[] getKeyWords(int limit, boolean stopWords)
  + The user indicates how many keywords they would like to see and whether they want stop words included in the list.
  + Make a copy of wordFrequency,
  + If user ignores stop words, load in file and delete those files from copy of hashmap
  + Check the user’s requested number of keywords. If it exceeds the number in the map to the max number in the map, or is a 0 or a negative, set to 10.
  + For loop for the number of keywords to be selected
    - Get a list of all of the keys, step through each word, keep the word with the highest frequency
    - Add keywork to a string array
    - Delete keyword from the copy of the hash map so it will not show up again.
  + Step through loop until all keywords have been acquired
  + Return string array of all keywords
* String[] getFunctionWords(String filePath)
  + Try to read in whichever file is being fed into the function (ex: conjunctions) and loop until it ends
    - Check if word is in hash map
    - If so, append it to a string
  + Split string of words into an array of strings
  + Return string array
* float getAverage(int numerator, int denominator)
  + cast numerator and denominator as floats and divide
  + return average
* boolean isAlphaNumeric(String string)
  + check string against pattern that looks for characters and numbers.
  + Return boolean
* void dereferenceEmptyNodes()
  + reset to head
  + create a previous node and save current to it
  + loop through the number of sentences(nodes) in linked list
  + If it’s payload has a string which passes the AlphaNumeric test, go to next node
  + Otherwise, go to next node, save new node as current
  + Previous contains reference to the last node that contained a sentence, so go back to that one, and set it to point to next. Return to next node
* void printFunctionWords()
  + Print label
  + Set a word count and a counter
  + Loop through string array returned by getFunctionWords based on whatever text file you’re looking at
    - Print out results, add together total of words and increment counter
  + Print total counter,
  + Rinse and repeat for the five other lists of function words
* writeReportCsv(String inputFileName)
  + Create an outfile based on the name of the inputfile, except replace .txt with .csv.
  + Write all the measurements to the file in csv style (see Lexical Analyser, Main Menu option 2): Description, data.
  + Print top 50 keywords with no stopwords: keyword, frequency.
  + Print all function words. Same as above, in format: Keyword, frequency
* writeReportTxt(String inputFileName)
  + Create an outfile based on the name of the inputfile
  + Write all the measurements to the file (see Lexical Analyser, Main Menu option 2):
  + Print top 50 keywords with no stopwords
  + Print all function words. Same as printFunctionWords()
  + Print out a sentence of the text, followed by the number of characters per word, total characters in sentence and number of words in the sentence
  + Repeat for every sentence until it ends.

CLASS: SentenceNode

* Private properties:
  + SentenceNode next
  + String Payload
  + int totalWordsInSentence
  + int totalCharactersInSentence
* Sentence Node()
  + Set empty properties
* SentenceNode(SentenceNode next, String payload)
  + Set properties based on what’s passed in
  + Set totalWords and totalCharacters
* Void setNext()
  + This.next = next
* SentenceNode getNext()
  + Return next
* Void setPayload(String payload)
  + This.payload = payload
* String getPayload()
  + Return payload
* Void setTotalCharactersInSentence()
  + Split payload into tokens
  + Iterate through and save the length of each word
  + Save totalCharactersInSentence to match sum
* Int getTotalCharactersInSentence()
  + Return totalCharactersInSentence
* Void setTotalWordsInSentence()
  + Split payload into tokens
  + Save tokens length as totalWordsInSentence
* Int getTotalWordsInSentence()
  + Return totalWordsInSentence
* String getLettersPerWord()
  + Split payload into tokens
  + Loop through tokens and append the length of each word onto the string
  + Return the string of numbers

# Revision

|  |  |  |
| --- | --- | --- |
| Ver. | Description | Date |
| 1.0 | original proposal | 2016-03-25 |
| 1.1 | Revised proposal to submit | 2016-04-25 |