



UNIVERSITY OF MARYLAND

DEPARTMENT OF COMPUTER SCIENCE

CMSC131 - OBJECT-ORIENTED PROGRAMMING I

SECTION 010X, 030X, AND 040X

DR. ROGER EASTMAN AND DR. ILCHUL YOON

Project 4

Text Analysis

Assigned: Apr. 16th, 2018

Due: Apr. 24th, 2018 11:00 PM

Late Due: Apr. 25th, 2018 11:00 PM — 20% penalty

Good Faith Attempt Due: May 1st, 2018 11:00 PM

Drop Option: You CANNOT drop this project.

1 Overview

In this project, you will practice writing code to manipulate textual data, using `arrays`, `ArrayLists` and methods provided by the `String` class.

You will use Eclipse to work on this project. Check out first the `TextAnalysis` project from the CVS repository and then implement the methods given in the starter files (Do NOT modify the method signatures). You will declare instance variables and write additional methods. You have some freedom to create variables and methods of your own design, that will not be subject to unit test.

The project asks you to tally textual statistics on a text file. You are to compute statistics like number of sentences, average sentence length, number of unique words, average word length, frequency of letters, among other values. These statistics can be used to distinguish texts, authors, genres and in general fingerprint text files.

2 Requirements and Assumptions

In this project, you are required to implement the methods in the following classes:

- **WordMain** class

The **WordMain** class contains the **main** method. It already contains the code to open an input file and read the header lines and also the code to print the statistics (i.e., **readHeader** and **statToString** method). It also contain the **analyzeBookText** method. You will complete the method to read (every words, starting from the book title) and update the analysis results. The letter, word, and sentence analytics data (e.g., frequency) should be kept in the **LetterTally**, **WordTally**, and **SentenceTally** object, respectively. The three objects are instantiated in the constructor of the **BookMain** class. Process book text data until you read the token: **\$\$\$END\$\$\$**. The end marker is added to the file not to process the license information attached at the end of the book file.

- **LetterTally** class

The **LetterTally** class keeps track of the frequencies of individual letters in the book. A letter in this project is defined as **a character between A and Z, or between a and z**. When you update the frequencies, assume lower-case letters. That is, you will increase the count of **a** when **A** occurs. Do not count any non-letter symbols (e.g., numbers, period, comma, and so on). You will declare instance variables needed to maintain the information and implement the following methods. See the class source code for the method signatures.

- **LetterTally**
- **totalCount**
- **freqChar**
- **computeEntropy**

For the **computeEntropy** method, use the following formula:

$$\sum_{i=1}^{26} p_i \times \log\left(\frac{1}{p_i}\right)$$

where p_i is the ratio of frequency count of a letter to the total count of the letters in the book.

Implement additional methods if needed. For example, you may want to write a method that takes a word and analyzes/updates the letter frequency data of the **LetterTally** object.

- **WordTally** class

The **WordTally** class keeps track of the frequencies of individual words in the book – **in specific, you must use an `ArrayList of Words`**. You will declare instance variables needed to maintain the information and implement the following methods. See the class source code for the method signatures. Note that an example token **example1904** is a word, although 1, 9, 0, and 4 will not be counted for the letter frequency.

- **WordTally**
- **getRawCount**
- **getUniqueCount**
- **avgWordLength**
- **longestWord**
- **freqWord**

Like `LetterTally` class, implement additional methods if needed. For example, you may want to write a method that takes a word and update the word frequency information maintained in the `WordTally` object.

- **Word class**

A `Word` object encapsulates a single word and the number of occurrences of the word in the book. In this project, a word is defined as **a token that has letters (See `LetterTally` class description for the definition), and is not a number or just punctuation or an HTML link**. Contractions should be counted as one word, not two – e.g., “don’t” should be modified to “dont” and be counted as one word.

As described above, the word data is maintained in the `WordTally` object using an `ArrayList` of `Word` objects. You will find that an empty constructor is given in the `Word` class source code. Complete the constructor and also write additional methods if needed – e.g., (1) a method that returns the occurrence count of the word or (2) a method that returns the length of the word.

- **SentenceTally class**

The `SentenceTally` class keeps track of the number of sentences in the book. You will declare instance variables needed to maintain the information and/or needed for implementing the following methods.

- `SentenceTally`
- `getCount`
- `avgSentenceLength`

Like `LetterTally` class, implement additional methods if needed. See the class source code for the method signatures.

- **WordLib class**

The `WordLib` class is a library class. That is, all methods defined in this class are *static*. You are requested to implement the following methods of which signatures are given in the class code.

- `cleanWord`
- `splitDash`

The methods are to pre-process (or clean) the input data, and would be called from the `analyzeBookText` method or from methods in three Tally objects. See the class source code for the details.

Consider the following assumptions and constraints.

- Declare all variables as **private** unless the **final** modifier is used in the declaration.
- Use **double** for the floating point numbers.
- Assume that a sentence is completed if a token (space-separated letter sequence in a book) *ends with* '.', '?', or '!'.
- Assume that no three or more words are connected with double dashes.
- The submit server applies a time limit when it runs a test. If you implement your code very inefficiently, tests could fail. For example, it can take long if a book has to be read multiple times.
- See in Section 4 that the average word length and the average sentence length are printed until the second decimal digit. The `String.format` method can be used for the formatted printing. See the `WordTally` and the `SentenceTally` class for details.

3 Grading and Good Faith Attempt Policy

Your submission will be evaluated based on the public, release, and/or secret tests in the submit server. **All public tests must be passed to meet the GFA requirements for this project.**

4 Example

An example output from a fully implemented project is given below for your reference. The data is from the book “Pride and Prejudice”.

```
Raw letter count: 536408
Letter entropy: 3
Raw word count: 121941
Unique word count: 6868
Longest word: 19
Average word length: 4.40
Sentence count: 5774
Average sentence length: 21.05
```

5 Requirements on readability and coding style

Follow a good coding style (e.g., <http://www.cs.umd.edu/~nelson/classes/resources/javastyleguide/>). Your code should be properly indented (highly encourage you to use the `AutoFormat` feature in Eclipse), and well documented with appropriate comments. It is mandatory to use meaningful variable names, rather than “magic numbers” (literal constants). There are no explicit requirements for how many and what type of variables. For readability including variable naming, we suggest you to use these rules:

- Format the code clearly with blank lines, spacing and indents
- Use self-documenting, meaningful variable names
- Use variable names that begin with lower case and then camel case (e.g., `roomTemp`) if two words
- Use capital letters (e.g., `ALL_CAPS`) for variables that encodes constant values that aren’t updated
- Use appropriate comments that indicate what a section, or a complicated line, does

As you have done for other projects, first few lines of your code should be the comments with the project, date, and author (you) information. You are **required** to write your name, directory ID, university ID, and section number as a comment at the top of each file you submit from this project forward. Additionally, we expect you to write the honor pledge (***I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination.***) as a comment near the top of each file you submit.

6 Submission

Submit your work to P4 in the submit server by the due date/time. It is strongly recommended to submit directly using the ‘Submit Project’ menu in Eclipse. After submission, check that your work is correctly submitted and also that your code passed all tests. For the full credits, your submission must pass all release/secret tests.

7 Academic Integrity

Make sure you read the academic integrity section of the syllabus so you understand what you must not do. Note that we check your submission against other students’ submissions, and that we are required to report academic dishonesty cases to the University’s Office of Student Conduct. As stated in section 2, you are **required** to write the honor pledge as a comment near the top of each file you submit.