Introduction

One statistical measure used in fields like document clustering, web search, and natural language processing is word frequency. The word frequencies of a given text document is simply a list of each unique word in the document along with the number of times it appears in the document (that is, the word’s frequency). For this project, you will create a program that can generate the word frequencies list for any given text document. The project will test your understanding of how to create and manipulate single dimensional arrays, and it will assess your understanding of packages and your ability to define and use interrelated classes in order to accomplish a complex task.

You will write a program called DocumentProcessor which takes a single command line argument. The argument is the URL of the text document to process. DocumentProcessor should process the text of the document, producing the word frequencies list, with words sorted in ascending lexicographical. For example, running the program using http://cobweb.cs.uga.edu/~cs1301/files/Project5/GettysburgAddress.txt as the command line argument should produce the output recorded in http://cobweb.cs.uga.edu/~cs1301/files/Project5/GettysburgAddress_output.txt.

The file DocumentProcessor.java has been partially implemented and provided for you. Completion of the program, however, requires the creation of other classes and packages. Specifically, the classes Word and Document must be created, both being part of the project5.documentprocessing.data package.

General program requirements are shown below, followed by a detailed specification of the classes to be implemented.

Program Requirements

1. Completion of the project5.documentprocessing.data.Word class. A partial class skeleton has been provided. You must implement the instance methods as indicated in the instructions below.

2. Completion of the project5.documentprocessing.data.Document. A partial class skeleton has been provided. You must implement the instance methods as indicated in the instructions below.

3. Use of the project5.documentprocessing.DocumentProcessor class. A partial implementation of this class has been provided for you, and you must use to test your finished program.

4. Use of the project5.documentprocessing.reader.DocumentReader class. Document and DocumentProcessor will need this class. This class has been provided for you in the docreader.jar library file on our course webpage. The webpage also contains instructions on how to add jars to an Eclipse Java project. A link to the API for the DocumentReader class is http://cobweb.cs.uga.edu/~cs1301/files/Project5/DocumentReader.htm.

5. All instructions, specifications, and notes in this project must be followed in order to receive full credit.

6. Thoroughly and rigorously test your program with all of the examples provided. In addition, test the program with your own examples.

7. Your program should never get stuck in an infinite loop.

8. Each submitted file must include the comment below. Copy the comment into each file, and fill in the name of your .java file, your name, submission date, and the program’s purpose. The comment must be placed in the lines between the import statements and the class declaration in your source code.

    /*
    * [Class name here].java
    * Author:  [Your name here]
Class Specifications

Class: **Word**

An instance of the **Word** class represents a single word found in a text document. The class must be defined as part of the package **project5.documentprocessing.data**, and it must have the following members.

**Instance variables:**
1. A string representing the word.
2. An integer representing the number of times that word has been seen in the document (its frequency).

**Methods:**
1. The constructor should take a single **String** parameter (representing the string value of the word).
2. **public String getWordValue():** Returns the string value represented by this **Word** object.
3. **public int getFrequency():** Returns the frequency of the **Word** object. This is the number of times the word has been found in a given document.
4. **public void incrementFrequency():** Increments the frequency value of the **Word** object by 1.
5. **public boolean equals(Word w):** Determines whether two **Word** objects are equal. Two **Word** objects are equal if they store lexicographically identical strings (ignoring case).
6. **public int compareTo(Word w):** Should return a negative integer, zero, or a positive integer when this **Word** is less than, equal, or greater to the supplied **Word** w, respectively. This method should return the compareTo value obtained by comparing the two strings stored by the **Word** objects.

Class: **Document**

The **Document** class represents a processed document. It must be defined in the package **project5.documentprocessing.data** and must have the members described below.

**Instance variables:**
1. An array of **Word** objects representing the unique words found in document. In these instructions, we will call this the **words array**.
2. An array of `String` objects representing strings (e.g., common words) that should be ignored when processing the document. In these instructions, we will call this array the `ignore list`.

3. A `DocumentReader` (which reads the lines of the document).

4. An integer which counts the number of unique words seen (strings in the ignore list should not be counted or included in the word array).

5. An integer that stores the numbers of lines in the document.

Methods:

1. You must define two constructors. The first takes a single `String` URL parameter and uses it to initialize the `DocumentReader` instance variable. It also initializes the array of `Word` objects to a size of 10 and then processes the document (this is done by invoking `processDocument`, described below). Importantly, after processing the document, the method should shrink and sort the words array (methods to do this are also described below).

2. The second constructor accepts the URL parameter described above as the first parameter, but it also accepts a `String` array containing the strings to ignore as the second parameter. This array should be used to initialize the ignore list (by creating a new array and copying the values of the parameter array to it). In other respects, this constructor functions like the first.

3. `public int getWordsSize()`: Returns the current size of the words array.

4. `public int getWordCount()`: Returns the number of unique words in the document, excluding those that have been ignored.

5. `public int getLineCount()`: Returns the total number of lines in the document.

6. `private void expandWords()`: Creates a new array having double the current word array's size, and copies all of the elements from the old array into the new one. Afterwards, the words array instance variable is updated to point to the newly created array.

7. `private void shrink()`: Shrinks the words array down to exactly the length needed to store all of the words seen in the document and nothing else.

8. `private String stripPunctuation(String w)`: Given a `String w`, it returns a string with all of the non-letter or digit characters removed from `w`.

9. `public boolean seenWord(String w)`: Returns true if a `Word` object for `w` has already been created and stored in the words array.

10. `private int findWord(String w)`: Returns the index in the words array of the `Word` object representing the word string parameter `w`. If it is not in the words array, the method returns -1.

11. `private int findString(String w, String[] strings)`: Returns the index in the strings array of `w`. If `w` is not in the array, the method returns -1. The search is case-insensitive (e.g., "cat" should match "Cat").

12. `private void addWord(String w)`: If a `Word` object does not exist in the words array corresponding to `w`, then a new `Word` object is created and added to the array. If such an object already exists in the array, then its frequency is increased.

13. `private void processDocument()`: Reads all of the lines from the `DocumentReader` and parses out each word, adding each word to the words array. This method is called by the constructor. Importantly, strings contained in the ignore list should not be added as words to the words array.

14. `private void sort()`: This has already been implemented in the provided class skeleton. The method calls `shrink()` and then sorts the `Word` objects in the words array according to the ascending lexicographical ordering of the strings they store. This method must be called by the constructor.
15. **public String toString()**: Returns a string with the document line count and word count on the first line and then the word string and frequency for each unique word found in the document (in ascending order). The last line should contain the length of the words array. See the examples for sample values.

**Class: DocumentProcessor**

This class is the program driver, and it has already been partially implemented. Observe that it is defined in the package `project5.documentprocessing`. The class contains the following method:

- **public static void main(String[] args)**: Here, `args` is an array of command line arguments. The 0th argument is assumed to be the URL of the text document to process. The method takes this URL and displays the word frequencies of the text document at that URL.

The program `DocumentProcessor` does not test the use of ignore lists in your `Document` class. To test your program with an ignore list, you should modify the `DocumentProcessor` to create a `Document` instance making use of an ignore list.

**Further Notes**

1. For the purposes of this project, words consist of only letters and numbers, and they must contain at least one character. Punctuation marks (non-letter or non-number characters) should not appear (they should be removed). Also, words are not case-sensitive (“Hello” is the same word as “hello”).

2. Similarly, the strings in the ignore list are to be considered case insensitive. That is, if “hello” appears in the ignore list, then “Hello”, “HELLO”, etc., should be ignored when encountered in a document. Word objects should not be created for them.

3. Words in a line are separated by spaces. You can use the `split` method of the `String` class to extract all of the words from a line. For example, if `samplestr` is a `String` representing a line of text then the following line of code will return a `String` array with all of the words in `samplestr`.

   ```java
   String[] linewords = samplestr.split(" ");
   ```

4. When removing the punctuation marks (non-letter or non-number characters), `Character.isLetterOrDigit(char ch)` can be used to determine if the character `ch` is an acceptable character to be in a word.

5. There are many algorithms to sort an array of elements. The one used in `Document.sort()` is called selection sort, which will be discussed during lecture class before the end of the semester.

6. Sample text files and the output are available online (see the examples below for their URLs).

7. In your code, you are NOT allowed to use any java.util.Arrays methods or the java stream API (if you are familiar with either of these). Using the java.util.Arrays class or Java stream in any way will result in a grade of zero on this assignment.

8. Note that in order to complete the project, you will need to create new packages. In order to create a new package in `Eclipse`, please follow the steps below:

   - Create the project in `Eclipse`.
   - Click on `File ➔ New ➔ Package` in the main menu.
   - A window titled `New Java Package` will open up. In the textbox `name` type the name of the package you want to create, for example, you’ll need to create the `project5.documentprocessing` package, and then create the `project5.documentprocessing.data` package.
   - When you have both packages created in Eclipse, place the correct files in those packages, and you should see the following package structure in the Package Explorer of Eclipse in your Java project for this assignment. If you don’t see something like the following, then you may have setup the packages incorrectly.
Project Submission

After you have thoroughly tested your program with all of the examples provided plus your own examples, you should submit the two files `Document.java` and `Word.java` to our course website. You do not need to submit `DocumentProcessor.java`.

Project Grading

This project cannot be submitted late. Programs not submitted by the deadline will receive a grade of zero. It is graded out of a possible 100 points. Programs that do not compile will receive a grade of zero. Using the java.util.Arrays class or Java stream in any way will result in a grade of zero on this assignment. You must make absolutely certain your program compiles before submitting, and you must thoroughly test your program with different inputs to verify that it is working correctly.

This project will be graded for both correctness and style:

**Style [20pts]**
- 5 points for including the class comment required for all projects in all of your submitted `.java` files
- 5 points for commenting all methods, constructors, instance variables, and class variables
- 10 points for naming all classes correctly AND defining their packages correctly

**Correctness [80pts]**
- 80 points for correct output on tests cases. Submitted programs must compile, run, and work correctly with any program that we use to test them.

Examples

**Example 1**
Output URL: [http://cobweb.cs.uga.edu/~cs1301/files/Project5/feynman_output_1.txt](http://cobweb.cs.uga.edu/~cs1301/files/Project5/feynman_output_1.txt)

Ignore List: none (first constructor used).

**Output**

- Line Count: 4
- Word Count: 5
- Word: feynman Frequency: 1
- Word: i Frequency: 8
- Word: richard Frequency: 1
- Word: why Frequency: 6
- Word: wonder Frequency: 8
- Words Size: 5

**Example 2**
Output URL: [http://cobweb.cs.uga.edu/~cs1301/files/Project5/feynman_output_2.txt](http://cobweb.cs.uga.edu/~cs1301/files/Project5/feynman_output_2.txt)
Ignore List: {"why", "I", "wonder"};

Output

Line Count: 4 Word Count: 2
Word: feynman Frequency: 1
Word: richard Frequency: 1
Words Size: 2

Example 3
URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/feynman.txt
Output URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/feynman_output_3.txt

Ignore List: {"I", "wonder", "why", "richard", "feynman"};

Output

Line Count: 4 Word Count: 0
Words Size: 0

Example 4
URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/fox.txt
Output URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/fox_output_1.txt

Ignore List: none (first constructor used).

Output

Line Count: 1 Word Count: 8
Word: brown Frequency: 1
Word: dog Frequency: 1
Word: fox Frequency: 1
Word: jumps Frequency: 1
Word: lazy Frequency: 1
Word: over Frequency: 1
Word: quick Frequency: 1
Word: the Frequency: 2
Words Size: 8

Example 5
URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/fox.txt
Output URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/fox_output_2.txt

Ignore List: {"the", "quick"};

Output

Line Count: 1 Word Count: 6
Word: brown Frequency: 1
Word: dog Frequency: 1
Word: fox Frequency: 1
Word: jumps Frequency: 1
Word: lazy Frequency: 1
Word: over Frequency: 1
Words Size: 6

Example 6
Ignore List: none (first constructor used).

Output

Line Count: 10 Word Count: 1
Word: remember Frequency: 10
Words Size: 1

Example 7

URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/remember.txt
Output URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/remember_output_2.txt

Ignore List: \{"you", "must", "remember", "this"\};

Output

Line Count: 10 Word Count: 0
Words Size: 0

Example 8

URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/empty.txt
Output URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/empty_output_1.txt

Ignore List: none (first constructor used).

Output

Line Count: 0 Word Count: 0
Words Size: 0

Example 9

URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/unreal.txt
Output URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/unreal_output.txt

Ignore List: none (first constructor used).

Output

Line Count: 9 Word Count: 9
Word: a Frequency: 1
Word: b Frequency: 2
Word: c Frequency: 3
Word: d Frequency: 4
Word: e Frequency: 5
Word: f Frequency: 6
Word: g Frequency: 7
Word: h Frequency: 8
Word: i Frequency: 9
Words Size: 9

Example 10

URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/GettysburgAddress.txt
Output URL: http://cobweb.cs.uga.edu/~cs1301/files/Project5/GettysburgAddress_output.txt

Ignore List: none (first constructor used).
Output: See URL above.

Copyright © Sal LaMarca and the University of Georgia. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License to students and the public. The content and opinions expressed on this Web page do not necessarily reflect the views of nor are they endorsed by the University of Georgia or the University System of Georgia.