**Lab 1.** **(Please read all pages before starting to code**)

You are to write a program in Java. Examine the file documents.txt. It contains a bunch of documents about flying. Your job it to extract data from the file. In the main method, you will create a DocumentCollection object that stores a HashMap, where the *key* will be the value for I (see the I attribute in the file) and will range from 1 to 1400. The *value* of the hash map will be an object of type TextVector. This object will contain a HashMap of its own. The *key* will be the word and the *value* will be its frequency. A word can contain only letters (either lower case or upper case). A word must have two or more letters. Please do not store information for the following noise words.

public static String noiseWordArray[] = {"a", "about", "above", "all", "along",

"also", "although", "am", "an", "and", "any", "are", "aren't", "as", "at",

"be", "because", "been", "but", "by", "can", "cannot", "could", "couldn't",

"did", "didn't", "do", "does", "doesn't", "e.g.", "either", "etc", "etc.",

"even", "ever", "enough", "for", "from", "further", "get", "gets", "got", "had", "have",

"hardly", "has", "hasn't", "having", "he", "hence", "her", "here",

"hereby", "herein", "hereof", "hereon", "hereto", "herewith", "him",

"his", "how", "however", "i", "i.e.", "if", "in", "into", "it", "it's", "its",

"me", "more", "most", "mr", "my", "near", "nor", "now", "no", "not", "or", "on", "of", "onto",

"other", "our", "out", "over", "really", "said", "same", "she",

"should", "shouldn't", "since", "so", "some", "such",

"than", "that", "the", "their", "them", "then", "there", "thereby",

"therefore", "therefrom", "therein", "thereof", "thereon", "thereto",

"therewith", "these", "they", "this", "those", "through", "thus", "to",

"too", "under", "until", "unto", "upon", "us", "very", "was", "wasn't",

"we", "were", "what", "when", "where", "whereby", "wherein", "whether",

"which", "while", "who", "whom", "whose", "why", "with", "without",

"would", "you", "your", "yours", "yes"};

Assume that anything that is not a letter (e.g. ‘, -, (,), space, tab, cartridge return, etc.) is a word separator. (Read about line.split("[^a-zA-Z]+"))

For simplicity, please convert all words to lower case when storing them in the TextVector. Here are some examples of how the system should work.

Mexico’s debt was: extract words: mexico and debt.  
Farmer-owned reserve national five-day average. : extract words farmer, owned, reserve, national, five, day, and average.  
Bread wheat prev 1,655.8, Feb 872.0, March 164.6, total 2,692.4 (4,161.0).: extract bread, wheat, prev, feb, march, and total.

Note that each document contains meta data, title, and body. You only need to extract data from the body (the text after .W)

At the end of the main method, serialize the DocumentCollection object to a file. For testing purposes, please print on the screen the word with the highest single document frequency and the frequency, the sum of the distinct number of words in each document over all documents, and the sum of the frequencies of all non-noise words that are stored. Save your work for the next lab.

Expected output:

Word = is

Frequency = 33

Distinct Number of Words = 90535

Total word count = 132267

Here is an example of how to store the data to a file, where docs is our DocumentCollection object. Note that all the objects that will be serialized must implement the Serializable interface.

try(ObjectOutputStream os = new ObjectOutputStream(new

FileOutputStream(new File("./files/docvector")))){

os.writeObject(docs);

} catch(Exception e){

System.out.println(e);

}

You must follow the following design. Create two packages: labs and DocumentClasses. Note that you need to implement all the methods below, where some methods will become useful in the next labs. You may have to create additional methods or add additional parameters to some of the methods.

File **labs.Lab1.java**: contains only the main method. Expected size is about 40 lines of code. Here, you create an object of type DocumentCollection. The method getDocumennts() of the object will return a set of TextVector objects.

File DocumentClasses**.**DocumentCollection.java: Expected size about 130 lines of code. Contains a variable documents of type HashMap<Integer, TextVector> that maps each document ID to a TextVector. Implements Serializable. Must contain the following methods:

Public methods:

* getDocumentById(int): returns the TextVector for the document with the ID that is given.
* getAverageDocumentLength() : returns the average length of a document not counting noise words. Use the method getTotalWordCount() on each document to calculate the number of non-noise words in each document. Add up the numbers and divide by the total number of documents.
* getSize() : returns number of documents
* getDocuments(): returns a Collection<TextVector>
* getEntrySet(): returns a mapping of document id to Text Vector, that is an object of type Set<Map.Entry<Integer, TextVector>>. Use the method entrySet on the HashMap to get the result.
* getDocumentFrequency(String): returns the number of documents that contain the input word.
* DocumentCollection(String): a constructor that reads the file that is specified as input and it uses the data in the file to populate the documents variable. //about 50 lines of code

Private methods:

* isNoiseWord(String): is the input a noise word.

File DcoumentClasses.TextVector.java: Expected size about 110 lines of code. Implements Serializable. Contains the variable rawVector of type HashMap<String, Integer> that stores the frequency for each non-noise word. Contains the following methods.

* getRawVectorEntrySet(): returns an object of type Set<Map.Entry<String, Integer>>. This is a mapping from each word to its frequency. Use the method entrySet on the HashMap to get the result.
* add(String): adds a word to the rawVector. If the word is not new, frequency is incremented by one.
* contains(String): returns true if the word is in the rawVector and false otherwise.
* getRawFrequency(String): returns the frequency of the word.
* getTotalWordCount(): returns the total number of non-noise words that are stored for the document (e.g., if frequency =2, then count the word twice).
* getDistinctWordCount(): returns the number of distinct words that are stored.
* getHighestRawFrequency(): returns the highest word frequency.
* getMostFrequentWord(): returns the word with the highest frequency.