CIT 594 Module 4 Programming Assignment

In this assignment, you will write a program that will analyze the sentiment (positive or negative) of a sentence based on the words it contains by implementing methods that use the List, Set, and Map interfaces from the Java Collections Framework.

Learning Objectives

In completing this assignment, you will:

- Become familiar with the methods in the java.util.List, java.util.Set, and java.util.Map interfaces
- Continue working with abstract data types by using only the interface of an implementation
- Apply what you have learned about how lists, sets, and maps work
- Get a better understanding of the difference between lists and sets
- Demonstrate that you can use lists, sets, and maps to solve real-world problems
- Gain experience writing Java code that reads an input file

Background

Sentiment analysis is a task from the field of computational linguistics that seeks to determine the general attitude of a given piece of text. For instance, we would like to have a program that could look at the text “This assignment was joyful and a pleasure” and realize that it was a positive statement while “It made me want to pull out my hair” is negative.

One algorithm that we can use for this is to assign a numeric value to any given word based on how positive or negative that word is and then determine the overall sentiment of the statement based on the average values of the words.

To determine the sentiment of an individual word, we can use a corpus of statements, each of which has **an overall score already assigned to it**. The sentiment of an individual word equals the **average** of the statements in which that word appears.

For instance, our corpus may look like this:

```
0 This was not as much fun as I thought it would be .
1 I had a lot of fun on this and learned a lot .
-1 It would be more fun if we had more time to work on it .
2 I didn't think programming in Java could be so much fun !
-2 I would have preferred an easier assignment .
2 I can't think of anything more fun than learning Java !
```

Each statement is labeled with a score from -2 to 2 as follows:
-2: very negative
-1: somewhat negative
0: neutral
1: somewhat positive
2: very positive

To determine the overall sentiment of the word "fun," we take the average of the sentences in which it appears. In this case, it would be (0 + 1 + -1 + 2 + 2) / 5 = 0.8.

Then, given a new sentence, we can determine its sentiment by computing the average of the sentiments of the individual words it contains. The sentiment of any previously unseen word would be 0.

Your program will be evaluated using a set of ~8,500 movie reviews that have been provided to you in the file **reviews.txt**. You can, of course, create your own input file for testing, but the correctness of your program will be determined using this file.

Getting Started

Download **Sentence.java** and **Word.java**, which represent a single sentence from the input file and a distinct word in the file, respectively.

Last, download **Analyzer.java**, which contains the "main" method for the program, as well as the unimplemented methods for the code that you will write in this assignment.

Activity Part 1. Reading the corpus as an input file

Implement the **readFile** method in **Analyzer.java**.

This method should take the name of the file to read and read it one line at a time, creating Sentence objects and putting them into the List. Note that the method returns a List containing Sentence objects.

For a valid sentence such as:
`2 I am learning a lot .`
then the **score** field of the Sentence object should be set to 2, and the **text** field should be "`I am learning a lot .`"

Your code should ignore (and not create a Sentence object for) any line that is not well-formatted, which we assume to mean "starts with an int between -2 and 2 (inclusive), has a single whitespace, and then is followed by more text."

However, if the file cannot be opened for reading or if the input filename is null, this method should return an empty List.

Note that it is up to you to determine which List implementation to return.

If you do not have prior experience writing Java code to read a text file, feel free to look online for help. There is good documentation at
https://docs.oracle.com/javase/tutorial/essential/io/file.html

Please do not change the signature of the *readFile* method and do not modify *Sentence.java.* Also, please do not create new .java files. If you need to create new classes, add them to *Analyzer.java*. Last, please make sure that your *Analyzer* class is in the default package, i.e. there is no "package" declaration at the top of the source code.

Activity Part 2. Calculating the sentiment of each word

Implement the **allWords** method in Analyzer.java (the same file as Part 1).

This method should find all of the individual tokens/words in the text field of each Sentence in the List and create Word objects for each distinct word. The Word objects should keep track of the number of occurrences of that word in all Sentences, and the total **cumulative** score of all Sentences in which it appears. This method should then return a Set of those Word objects.

If the input List of Sentences is null or is empty, the allWords method should return an empty Set.

If a Sentence object in the input List is null, this method should ignore it and process the non-null Sentences.

As you can see, **allWords** needs to tokenize/split the text of each Sentence to get the individual words. Consult the Java documentation for help with this:

- https://docs.oracle.com/javase/8/docs/api/java/lang/String.html
- https://docs.oracle.com/javase/8/docs/api/java/util/StringTokenizer.html

Keep in mind that when you tokenize the text of each Sentence, you will be getting Strings, but the Set that this method returns needs to include Word objects. However, if two Strings are equal, they should be combined into the same Word object, which should track the cumulative score of all Sentences containing that String.

As you may have noticed in the data file we provided, some tokens start with punctuation and the first word of each sentence starts with a capital letter. In producing the Set of Words, **your program should ignore any token that does not start with a letter**. Also, this method should convert all strings to **lowercase** so that it is case-insensitive. Do not assume that the strings in the Sentence objects have already been converted to lowercase.

As an example, consider this text:
It 's a lot of fun to learn about data structures.

Your program should convert "**It**" to "**it**" (to make it lowercase) and ignore " **'s** " and the period at the end of the sentence since those tokens do not start with a letter.

As in Part 1, it is up to you to determine which Set implementation to return.

Hint: although this method needs to return a Set of Words, you may find it easier to use a different data structure while processing each Sentence, and then put the combined results into a Set before the method returns. There is not necessarily a single "correct" way to implement this method, as long as you return a Set of Words at the end.

Please do not change the signature of the allWords method and do not modify Sentence.java or Word.java. Also, please do not create new .java files. If you need to create new classes, add them to Analyzer.java. Last, please make sure that your *Analyzer* class is in the default package, i.e. there is no “package” declaration at the top of the source code.

Activity Part 3. Storing the sentiment of each word

Implement the **calculateScores** method in Analyzer.java (the same file as Parts 1 and 2).

This method should iterate over each Word in the input Set, use the Word's **calculateScore** method to get the average sentiment score for that Word, and then place the text of the Word (as key) and calculated score (as value) in a Map.

If the input Set of Words is null or is empty, the calculateScores method should return an empty Map.

If a Word object in the input Set is null, this method should ignore it and process the non-null Words.

As above, it is up to you to determine which Map implementation to return.

Please do not change the signature of the *calculateScores* method and do not modify *Sentence.java* or *Word.java.* Also, please do not create new .java files. If you need to create new classes, add them to *Analyzer.java*. Make sure your *Analyzer* class is in the default package, i.e. there is no "package" declaration at the top of the source code.

Activity Part 4. Determining the sentiment of a sentence

Finally, implement the **calculateSentenceScore** method in Analyzer.java.

This method should use the Map to calculate and return the average score of all the words in the input sentence.

Note that you will need to tokenize/split the sentence, as you did in Part 2.

If a word in the sentence does not start with a letter, then you can ignore it, but if it starts with a letter and is not present in the Map, assign it a score of 0.

You may assume that all words in the Map consist only of lowercase letters (since this would have been done in previous steps) but do not assume that all words in the input sentence consist only of lowercase letters; you should convert them to lowercase before checking whether they're contained in the Map.

If the input Map is null or empty, or if the input sentence is null or empty or does not contain any valid words, this method should return 0.

Although you can (should!) test each method individually, you can test the entire program using the *main* method in *Analyzer.java*. Be sure to specify the name of the input file as the argument to main.

Please do not change the signature of the *calculateSentenceScore* method and do not modify *Sentence.java* or *Word.java.* Also, please do not create new .java files. If you need to create new classes, add them to *Analyzer.java*. Last, please make sure that your *Analyzer* class is in the default package, i.e. there is no "package" declaration at the top of the source code.

Helpful Hints

Documentation about the methods in the List, Set, and Map interfaces are available as part of the Java API docs:

- https://docs.oracle.com/javase/8/docs/api/java/util/List.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Set.html
- https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

Refer to this documentation if you need help understanding the methods that are available to you.

In implementing this program, we recommend that you implement and test each of the four methods one at a time. That is, rather than implementing all four methods and then hoping it all works correctly, do Part 1 and convince yourself that it works, then do Part 2 (either using the output of Part 1, or input that you create by hand) and convince yourself that it works, and so on. It's a lot easier to debug this way!

Before You Submit

Please be sure that:

- your *Analyzer* class is in the default package, i.e. there is no "package" declaration at the top of the source code
- your *Analyzer* class compiles and you have not changed the signatures of any of the four methods you implemented
- you have not created any additional .java files and have not made any changes to *Sentence.java* or *Word.java* (you do not need to submit these files)

How to Submit

After you have finished implementing the *Analyzer* class, go to the "Module 4 Programming Assignment Submission" item and click the "Open Tool" button to go to the Codio platform.

Once you are logged into Codio, read the submission instructions in the README file. Be sure you upload your code to the "submit" folder.

To test your code before submitting, click the "Run Test Cases" button in the Codio toolbar.

As in the previous assignment, **this will run some but not all of the tests that are used to grade this assignment.** That is, there **are** "hidden tests" on this assignment!

The test cases we provide here are "sanity check" tests to make sure that you have the basic functionality working correctly, but **it is up to you to ensure that your code satisfies all of the requirements described in this document.** Just because your code passes all the tests when you click "Run Test Cases" doesn't mean you'd get 100% if you submit the code for grading!

When you click "Run Test Cases," you'll see quite a bit of output, even if all tests pass, but at the bottom of the output you will see the number of successful test cases and the number of failed test cases.

You can see the name and error messages of any failing test cases by scrolling up a little to the "Failures" section.

Assessment

This assignment is scored out of a total of 75 points.

Part 1 (readFile method) is worth a total of 20 points, based on whether the method correctly reads the input file and creates a List of Sentence objects, and whether it correctly handles errors. Note that some of the input files used for grading are available in the "tests" folder in Codio; the others are not made available prior to submission.

Part 2 (allWords method) is worth a total of 28 points, based on whether the method correctly converts the List of Sentences to a Set of Words, and whether it correctly handles errors. The correctness of this method is evaluated purely based on its own implementation, and does not assume a correctly functioning readFile method.

Part 3 (calculateScores method) is worth a total of 9 points, based on whether it correctly converts the Set of Words to a Map of Strings and doubles, and whether it correctly handles

errors. The correctness of this method is evaluated purely based on its own implementation, and does not assume a correctly functioning allWords method.

Part 4 (calculateSentenceScore method) is worth a total of 18 points, based on whether it correctly calculates the score of a sentence, based on the input Map, and whether it correctly handles errors. The correctness of this method is evaluated purely based on its own implementation, and does not assume a correctly functioning calculateScores method.

As noted above, the tests that are executed when you click “Run Test Cases” are **not** all of the tests that are used for grading. There are “hidden” tests for each of the three methods described here.

After submitting your code for grading, you can go back to this assignment in Codio and view the “results.txt” file, which should be listed in the Filetree on the left. This file will describe any failing test cases.