Homework 8

TextAnalyzer

1 Introduction

Text documents are everywhere in the world. Whether its a newspaper article, a product review, academic paper, or a post on Facebook, text documents play a vital role in our lives. For that reason, we wanted to give you practice manipulating them using Java. Broadly, in this homework youll be doing some rudimentary text mining. What does that mean? Text mining is process of running different kinds of analytic measures of the words and text within a document. Youll be doing things like word counting, sentence counting, and sentiment analysis. What is sentiment analysis, you ask. Thats determining whether a particular text document is happy and positive or it is negative and angry. Well try to come up with a quantitative value for that.

Additionally, be sure to read the **entire document** as there are key details throughout that will help you succeed on this assignment.

2 Problem Description

This is another assignment where we want you to design the program yourself. Were not going to tell you which classes or methods to write. That's up to you. But of course, you should follow sound principles of object-oriented programming including encapsulation and modularity.

Your program should prompt the user to enter the name of a text file to be read. This will be the "source" file. Next it should prompt the user to enter the name of a text file of target words. This file should have one word (or phrase) per line. Additionally, your program should take two command line arguments, the names of two files containing our positive sentiment vocabulary words and our negative sentiment vocabulary words, respectively.

Your program then should run and compute a number of different metrics which it then reports as output (details are explained below). These metrics are:

- The total count of words and sentences in the source document.
- The longest sentence (measured by number of words) within the source document.
- The counts of all the target words in the source document (this is written to an external file).
- The summary sentiment analysis score for the document, which is simply number of positive words minus number of negative words.

For the purposes of this assignment, we will consider a sentence to be any sequence of words separated by a period, question mark, or exclamation point. A word is a sequence of letters and or numbers delimited by whitespace or

punctuation.

To determine a sentiment score, you should check every word in the source document against the positive and negative word lists, which are provided as part of the homework description on T-square. For each positive word you find, think of adding 1 to the score, while subtracting one for each negative word you find.

3 Sample Output

```
$ java TextCheck goodWords.txt badWords.txt
What is the name of source file to analyze?
tester.txt
What is the name of the target words file?
Mywords.txt
Sorry, but that file doesnt exist. Please retype the name.
mywords.txt

Analyzing
Your file contains 932 words and 111 sentences.

The longest sentence in this document is:
Wow, this is a really long sentence with a whole lot of words in it and I hope I get it right.

Writing out target word counts to file targetCount.txt

The sentiment analysis score is +12.
```

4 Solution Description

As mentioned above, the design of this program is up to you. We require that your program have a class named TextCheck which includes the `main()` method, but that's all it should include. Think about what you should add from there. The example programs we looked at earlier this term (Palindrome checker, Pig Latin translator, Syllable counting) may provide you with some helpful ideas. Your grade will depend not only on correct output, but also good program style.

5 Helpful Hints

- As the sample output showed, users can enter names of files that don't exist. Your program should handle these situations in an intelligent manner.
- Case sensitivity shouldn't count, that is, "Truck" and "truck" are the same word.
- When we test your program, our test files will have different levels of difficulty. At the first level, the source text only will include words with letters and the simple sentence-ending punctuation (!?.). If you get your program working correctly for this level, you will earn 80 points. At the second level, sentences can include other types of punctuation. Think about the single sentence:
  `"He bought shoes, ties, and belts at the store," said the narrator; I disagreed, however.`
  Note that a word here is "ties" not "ties,". If you get your program working correctly for this level, you will earn 95 points. Finally, at level three, there is additional tricky stuff. For example, beware contractions. "Don't" is

one word. Also, abbreviations are a pain.
`Dr.  Patel went to class.`
Is just one sentence. If you handle level three reasonably well, you will earn the full 100 points.

6 Example Calculations

Below are some example files and the appropriate output.

Source file contents:

```
The beginning.  This is the middle sentence.  This is the end.
1234 1234 1234?  This is another line of input.
Here is another. Happy happy joy joy!
```

Target word file contents:

```
the
this
happy
```

Appropriate output:

```
Analyzing
Your file contains 27 words and 7 sentences.

The longest sentence in this document is:
This is another line of input.

Writing out target word counts to file targetCount.txt

The sentiment analysis score is +4.
```

The output file targetCount.txt contents:

```
the: 3
this: 3
happy: 2
```

7 Javadocs

For this assignment you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online documentation for them is very detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 13.31
 */
public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */
    public Dog(){
        ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */
    public int add(int a, int b){
        ...
    }
}
```

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

7.1 Javadoc and Checkstyle

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.1.jar -j *.java
Audit done. Errors (potential points off):
0
```

8 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points.
Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar -a *.java
Audit done. Errors (potential points off):
0
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **20** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

9 Turn-in Procedure

Non-compiling or missing submissions will receive a zero. NO exceptions

Submit all of the Java source files you modified and resources your program requires to run to T-Square. **Do not submit** any compiled bytecode (`.class` files) or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

9.1 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!