# Word Counter API documentation

## Introduction

This was my first time creating an API, I have prior knowledge of using Java in education and in industry. I had to conduct some research to be able to use the right tools for this project. Since the project brief mentions using Maven, I needed to investigate getting that set up. Following setting up Maven I discovered that I required spring boot to create an API along with the requirement of Junit to be able to conduct unit tests on the code. Having a lack of experience in these areas proved to be the most challenging part of the test. I believe my quality of code can make up for the lack of experience in working with Maven and Junit, I am a quick learner, this can be seen by some of my previous work experience. With the right guidance I can grasp a good understanding of using Maven and Junit, although in this documentation I will show my uses of it and attempts to set it up for a working API. I believe creating efficient, elegant code, along with thoroughly detailed documentation is vital to software development.

## The task

Please write in Java an API to read the contents of a plain text file and enable the display of the total number of words, the average word length, the most frequently occurring word length and a list of the number of words of each length.  
Submit your code along with unit tests and maven POM file, in a zip file via email along with instructions for its installation and use along with any assumptions you have made about what defines a word (using as a basis the rules that can be deduced from the example below)   
For example, given a file that contains the following text:

Hello world & good morning. The date is 18/05/2016

We would expect the following output:

Word count = 9

Average word length = 4.556

Number of words of length 1 is 1

Number of words of length 2 is 1

Number of words of length 3 is 1

Number of words of length 4 is 2

Number of words of length 5 is 2

Number of words of length 7 is 1

Number of words of length 10 is 1

The most frequently occurring word length is 2, for word lengths of 4 & 5

## Assumptions

The only assumptions that had to be made here was what defines a word and how many decimal places to use in the average. I used Regex to dictate what characters remained in the project. Going on the example set in the task brief, I assumed that an “&” symbol along with a “/” could be counted as words. Since they contributed to the word count in the example. The only time this could be a problem is if a “/” is used anywhere else besides a formatted date.

static String *removeSpecialCharacters* = "[^\\w\\s\\/\\&]|\_";

We can see from line 21 of the program. This removes special characters except for normal characters, &, / and any whitespaces.   
If white spaces needed to be removed, for example in the word count, they are explicitly removed later in the code.

The example also gives the average word length in 3 decimal places, this is another assumption I made that the average should be rounded to this amount of decimal places, since it’s the only variable that will be using decimal places this assumption couldn’t be applied to any other part of the task.

## Main code

Outside of the methods in the program. There is the file input, followed by the removing of special characters.

The input file needs to be specified on line 17: It will then be read from the filereader in the later methods.

static File *input* = new File("INPUT");

I tackled the problem by splitting up the problem into the following 2 methods:

public static void wordAndCharCount() throws IOException{

Lines: 23-57

public static void numberOfWords() throws IOException{

Lines: 59-108

I found this to be an effective way, since the first method “wordAndCharCount” gathers the Word Count and the Character count. The character count is required to make the average. Therefore, this method handles the word count and the average word length outputs.   
Followed by the second method “numberOfWords” which handles the number of word length and the most frequently occurring word lengths.

wordAndCharCount

This method uses 2 while loops to count the words and the characters, followed by a calculation that finds the average word length and prints it in 3 decimal places.  
This method reads the file by using the file reader and buffered reader objects.

numberOfWords

This method uses 2 maps to get the most frequently occurring word length and the word lengths. Followed by a series of lambda functions to filter through the word lengths and pair any that are matching.   
This method reads the file by using the buffered reader, file reader then a string builder.

## Testing

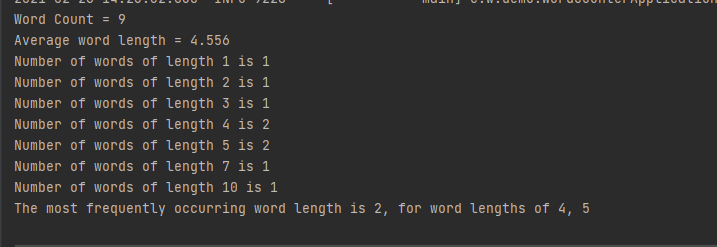
From my understanding, the tests needed to include all possible outcomes that can be expected in the program. Therefore, it was important to think about everything that could happen, so it is important to include tests that are expected to fail, as well as ones that are expected to pass.   
I tried to figure out how to get the test cases to work using Junit. I believe the main issue with them not working was because you can not retrieve a variable from a static method, since there is no return statement. I kept running into issues trying to test against variables. I then tried changing them to “public int” to be able to complete the tests. However, this didn’t fix the issue that I kept encountering.

“Cannot resolve method 'wordAndCharCount' in 'WordCounterApplicationTests'”

I spent a lot of time trying to figure this out but kept finding myself going in circles. I believe with more time this can be fixed and I’m quite possibly overlooking something simple.

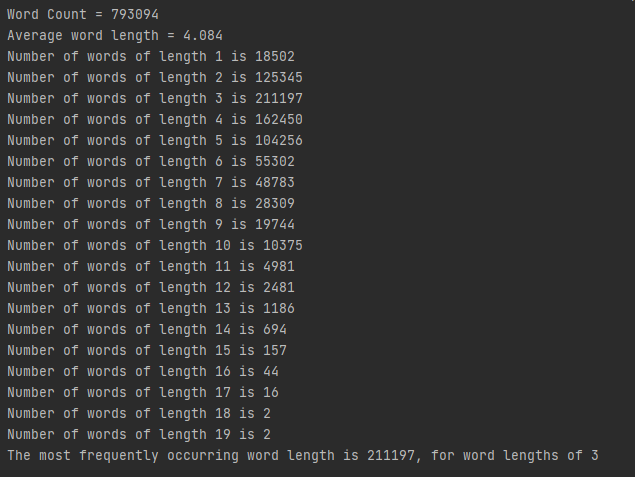
I have included the test file in the github repository with my attempts at writing test cases.

Outside of the test cases, I ran the code several times to get the expected results. Going from the example given in the task brief, we can see that expected outcome was given.

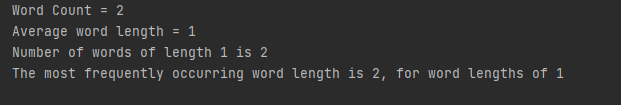


I proceeded to test the bible that was given in the task brief as well. However, I wasn’t sure how to test the accuracy of this. I put the file in microsoft word, where it had a different word count. This is down to the fact that Microsoft word can include “\*\*\*” in the word count, which I assumed didn’t count as a word.

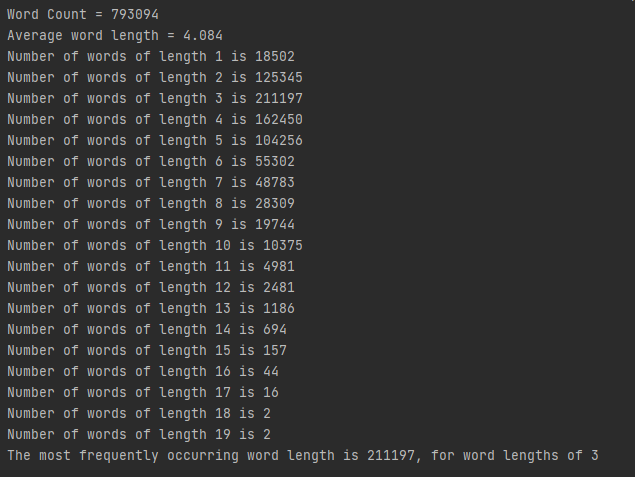
This was the expected outcome of running the bible:



After running this test, it seemed appropriate to test that the special characters were getting removed correctly.   
To do this, the input “! " £ $ % ^ & \* ( ) - = # [ ] ' / . ,” was used to see which ones would be removed. As mentioned in the assumptions, the only ones which should remain would be “&” and “/”.



Finally, testing the bible input seemed necessary. This is the output that was given, the issue with this I believe is what defines a word. With the assumptions I had made, this was the output that was given.



## XML

I used spring boot framework, however as mentioned before this was completely new to me and had to learn how to write XML files properly. It seemed to work as expected since it gave the spring boot launch message when running the program.

