Austin Pena (est892)  
CS 5103  
5/2/2021

Design Change Report for Word Statistics Project

Initial Design

The initial requirement of the project was to create software which counts the frequency of each unique word in a file. I chose Java to code this software, because Java has a HashMap data structure to keep track of unique elements and their corresponding values.

My design approach was similar to the top-down strategy, so that I could continually test the entire program as a whole instead of testing with small, isolated components. I started by coding and testing the main Java function, and then I added the features one at a time. The 3 main features were: attaining a file from the user, scanning through the file to count the words, and printing the word-frequencies.

The Java main() component mostly involved setting up a directory for the project and making the Java file with an appropriate name and working main function. Once I could compile and run the main function, I could then move onto the components one at a time, so that any bugs which occurred could be traced to that specific component. None of the 3 components were big enough to be broken into smaller tasks. I built the components sequentially for simplicity, however they could have been built in any order. Comments were added at the end; the program was too simple to warrant focusing on comments before the functional code was finalized.

First Design Change

The first design change was to have the program also print a Character Count and Line Count. It seemed most efficient to count the characters by incrementing a Char Count integer by the length of each word being scanned. As for the Line Count, the simple solution was to perform a secondary scan of the file to count its lines one at a time and store that count in a Line Count integer. Performing a second scan is inefficient, however the time complexity of the code would remain O(n) with respect to the size of the file, so I felt this was an appropriate solution. I then added comments related to these new items.

These changes created a few potential vulnerabilities. If the Character Count or Line Count reached 2^31, then an integer overflow error is guaranteed. Potentially, these errors could be prevented by using a Big Integer object, however these edge cases seemed too extreme to warrant the additional code complexity.

Second Design Change

The second design change involved allowing the user to replace a chosen word with another word. This could be broken up into 3 software components: prompting the user if they want to swap a word, prompting the user for their chosen words, and swapping out the old word when it is encountered in the file. I built the components in this order.

There were no problems with implementing this design change. I simplified the first request from the user down to a Y/N question, so that the user would not have difficulty understanding what to respond. Alternatively, I could have requested a numerical response from the user, such as ‘1’ for option 1 and ‘2’ for option 2. However, this was unnecessary with only 2 options. Any responses other than ‘y’, ‘Y’, ‘n’, or ‘N’ are discarded.

The second prompt was simpler, since the user can state any words they want. Even if the word is not present in the text file, that will not cause any errors. One limitation is that a word cannot be replaced with an empty string, because an empty string response will not be recognized by this Scanner. However, that edge case is not part of the requested design change “replace a word with another word”, so it is outside of the scope of this change request.

Finally, an if-statement was added to the code to check for the replaced word while scanning the file. The if-statement appears before character count and word frequencies totals are updated, so that the new word is calculated into those values. If replacement was not requested, then nothing will change. I then updated the comments to reflect all of the new features.

I also had to refactor the code to fit it into multiple methods rather than run everything in the “main” method. I previously had everything run in the main method for simplicity, however the JUnit tests warranted the use of individual methods that could be called for an object.