**Word Occurrences Unit Testing Implementation**

Unit testing was introduced to the word occurrences program (labeled as WordCounterGUI in my repository) by adding JUnit testing classes to the project file. The brunt of the processing happens within the Main.java class, so the unit tests were designed for select methods within that class. Below, I document what each of those unit testing classes are aiming to test, and what the expected result should be.

**WordSearchTest.java**

This class is created to test the method *WordSearch* within the *main* method of the Main.java class. The intended purpose of this *WordSearch* method is to take a string that a user inputs, and search through a specified Map to find that string as a key, then retrieve that key’s value (if the string was found), and finally return a Boolean flag indicating whether or not the string was found. As an example, suppose the Map contains the following key value pairs: (“Dog”, 4, “Cat”, 2, “Bird”, 1), and suppose the user entered the text “Cat”. The method will take the entered string “Cat”, the Map, and a class called FetchedWords (this class holds the content of the key and the value that are extracted from the list). The method will then compare the string “Cat” against the Map, and upon finding a matching string, will set the Boolean flag to true and update the properties of the FetchedWords class to equal “Cat” and 2. The method only returns a Boolean value of true. From here, the rest of the program will make use of the updated properties in FetchedWords and the Boolean flag.

The test is designed exactly as the example above reads. A sample map, user input string, and FetchedWords class are created and passed into the *WordSearch* method (in this case the map’s keys and values are populated as shown above, and the sample input string is “Cat” also as shown above). The Boolean value is returned and compared against a value of true since the text “Cat” exists within the sample map. This Boolean value is then used in the *assertEquals* method to perform the test. The result should be true, so if a value of true has not been returned by the *WordSearch* method then the test fails.

**SortedDictionaryTest.java**

This class is created to test the method *SortedDictionary* within the *main* method of the Main.java class. The intended purpose of this method is to take an unsorted map with string keys and integer values and sort them in descending order of values. This method takes a map of string keys and integer values as input and returns a (sorted) map of the same type. As an example, for an input map containing the key value pairs: (“Bird”, 1, “Cat”, 2, “Dog”, 4), the output map would look like: (“Dog”, 4, “Cat”, 2, “Bird”, 1). This is what the test does, it takes the exact unsorted map as described above, and returns the sorted map as described above. After the sorting happens, the first index of the sorted map is passed through the *assertEquals* method where both the key and the value are compared. The correct result should contain “Dog”, 4 as the value. Additionally, the first entries of each map (sorted and unsorted) are printed to the console for comparison.

**Additional Comments**

The final method for which a unit test was attempted to be implemented was the *ParseText* method, in the *main* method of the Main.java class. This method does not take any input, but returns a map containing string keys and integer values. The reason that this method was not tested is that implementing a test would itself exactly duplicate the method being tested. The *ParseText* method internally imports text from a hard-coded webpage and runs dozens of regex comparisons against the text contained on that webpage, looking for patterns in the text. The only thing that could be tested from this method is the returned map. In this case, the test would not be able to send any sample text to the method. Additionally, if the method were changed such that sample text could be sent to the method, the regex text parsers would need to be re-tuned to match whatever patterns the sample text might have. In re-tuning these regex parsers, the original regex parsers would be altered, which would render testing of the current regex parsers meaningless. Such results would not be useful in verifying the correctness of the method, only the effectiveness of the re-tuned regex parsers.