Practice Coding Assignment 16 : Word Counter

This assignment is entirely optional and designed to give you practice writing code and applying lessons and topics for the current module.

This homework focuses on the topics:

- File I/O
- Text parsing
- Collections
- Unit testing

The Assignment

In this assignment, you will build a word counter. The program will read and parse text from a .txt file, get the counts of words in the file, and then write the results to a new file.

There are four main files in the program: *Main.java*, *MyFileReader.java*, *MyFileWriter.java*, and *WordCounter.java*. The *MyFileReader* class will read the provided "war_and_peace.txt" file and return the cleaned up contents of the file as an ArrayList<String>. The *WordCounter* class will process the array list provided by *MyFileReader* and calculate the count of each word in the list of lines. It will keep track of the words and their counts in a HashMap<String, Integer>. *WordCounter* will also provide a way (method) of getting a list of words that are repeated in the text a specific number of times, and the *MyFileWriter* class will write the list of words to a new file "output.txt".

The *Main* class will drive the entire program, and the "main" method calls have been provided for you. Nothing needs to be completed in the main method.

The program logic looks like this:

1. Using the *MyFileReader* class, read the "war_and_peace.txt" file and get the cleaned up lines of text as an ArrayList<String>
2. Pass the ArrayList<String> from the *MyFileReader* class to the *WordCounter* class
    a. Traverse the list of lines, and keep track of the count of each word
    b. Store the word counts in a HashMap<String, Integer> called *wordCount*
3. Inside the "main" method in the *Main* class, load the file and then get the words that are repeated a specific number of times
    a. Add this information to the ArrayList<String> *words*
4. Write the ArrayList<String> *words* to the "output.txt" file using the *MyFileWriter* class

Implement all methods with "//TODO" in all classes.

MyFileReader Class

Instance Variables

The *MyFileReader* class has the following instance variable defined for you:

- String filename - Name of file being read

Constructor

The constructor in the *MyFileReader* class has been defined and **implemented for you**. You don't need to make any changes to the constructor in this class file.

The constructor has one parameter which represents the name of the file being loaded and read. The code in the constructor sets the value of the corresponding instance variable in the *MyFileReader* class.

Below is the code for the constructor in the *MyFileReader* class:

```
/**
 * Creates MyFileReader with given filename to read.
 * @param filename to read
 */
public MyFileReader(String filename) {
    this.filename = filename;
}
```

Methods

There is 1 method defined in the *MyFileReader* class that **needs to be implemented**:

- getCleanContent() - Opens the file specified by filename and reads the text line by line. Cleans the contents by trimming whitespace from the beginning and end of each line. Adds each line to an ArrayList<String> which is returned from the method.

Below is the code for the "getCleanContent" method in the *MyFileReader* class. For now, the method just returns an empty ArrayList. Read the javadoc, which explains what the method is supposed to do. Then write your code where it says "// TODO". We have provided hints and example method calls to help you get started.

```
/**
 * Opens the file specified by filename and reads the text line by
 * line.
 * Cleans up each line by trimming whitespace from the beginning and
 * end of each line.
```

```
 * Adds each line to an ArrayList<String> which is returned from the
 * method.
 * If a line in the file is empty (does not contain any text), it's
 * skipped and is not added to the ArrayList<String>.
 *
 * @return list of lines with no empty spaces at the beginning or end
 * of each line
 */
public ArrayList<String> getCleanContent() {
    ArrayList<String> lines = new ArrayList<String>();

    // TODO Implement method

    return lines;
}
```

WordCounter Class

Instance Variables

The *WordCounter* class has the following instance variables defined for you:

- ArrayList<String> lines - List of lines of words to count.
- Map<String, Integer> wordCount - Map storing the count of each word in the list of lines. Each word will be a key, and the associated counts of each word will be the values.

Constructor

The constructor in the *WordCounter* class has been defined and **implemented for you**. You don't need to make any changes to the constructor in this class file.

The constructor has one parameter which represents the list of lines to process. The code in the constructor sets the value of the corresponding instance variable in the *WordCounter* class and starts the process of generating the count of each word in the list of lines.

Below is the code for the constructor in the *WordCounter* class:

```
/**
 * Creates WordCounter based on the given list of lines.
 * Starts the process of generating the count of each word in the
 * list.
```

```
 * @param lines of words to count
 */
public WordCounter(ArrayList<String> lines) {

    this.lines = lines;
    this.wordCount = new HashMap<String, Integer>();
    this.generateWordCounts();
}
```

Methods

There are 3 methods defined in the *WordCounter* class that **need to be implemented**:

- generateWordCounts() - Calculates the count of each word in the list of lines. Traverses the list of lines, and keeps track of the count of each word. Stores each word as a key and its associated count as a value in the HashMap<String, Integer> *wordCount*.
- getWordCounter() - Returns the HashMap<String, Integer> wordCount.
- getWordsOccuringMoreThan(int threshold) - Gets a list of words that appear a particular number of times, indicated by the given threshold (minimum word count). Each word in the returned list will have a word count >= threshold.

Again, each method has been defined for you in the *WordCounter* class, but without the code. See the javadoc for each method for instructions on what the method is supposed to do and how to write the code. It should be clear enough. In some cases, we have provided hints and example method calls to help you get started.

For example, below is the code for the "getWordsOccuringMoreThan" method in the *WordCounter* class. The method gets a list of words that appear a particular number of times, indicated by the given threshold. For now, the method just returns an empty ArrayList. Read the javadoc, which explains what the method is supposed to do. Then write your code where it says "// TODO". You'll do this for the other methods in the *WordCounter* class as well.

```
/**
 * Gets a list of words that appear a particular number of times,
 * indicated by the given threshold.
 *
 * @param threshold (minimum word count) for words to include in the
 * returned list, where each word has a word count >= threshold.
 * @return list of words, where each has a count >= threshold
 */
public ArrayList<String> getWordsOccuringMoreThan(int threshold) {
    ArrayList<String> result = new ArrayList<String>();
```

```
    // TODO Implement method

    return result;
}
```

MyFileWriter Class

Instance Variable

The *MyFileWriter* class has the following instance variable defined for you:

- String filename - Name of file being written to

Constructor

The constructor in the *MyFileWriter* class has been defined and **implemented for you**. You don't need to make any changes to the constructor in this class file.

The constructor has one parameter which represents the name of the file being written to. The code in the constructor sets the value of the corresponding instance variable in the *MyFileWriter* class.

Below is the code for the constructor in the *MyFileWriter* class:

```
/**
 * Creates MyFileWriter with given filename to write to.
 * @param filename to write to
 */
public MyFileWriter(String filename) {
    this.filename = filename;
}
```

Methods

There is 1 method defined in the *MyFileWriter* class that **needs to be implemented**:

- writeToFile(ArrayList<String> words) - Opens the file specified by filename and writes each word in the given list of words to the file. Each word is written to a new line.

Below is the code for the "writeToFile" method in the *MyFileWriter* class. Read the javadoc, which explains what the method is supposed to do. Then write your code where it says "// TODO". We have provided hints and example method calls to help you get started.

```
/**
 * Opens the file specified by filename and writes each word in the
 * given list of words to the file.
 * Each word is written to a new line in the file.
 *
 * @param list of words to write to the file
 */
public void writeToFile(ArrayList<String> words) {

    // TODO Implement method

}
```

Unit Testing

In addition, you will write unit tests to test all method implementations in the program. Each unit test method has been defined for you, including some test cases. First make sure you pass all of the provided tests, then write **additional and distinct test cases** for each unit test method.

For example, in *MyFileReaderTest.java* we have defined 3 instances of *MyFileReader* for testing. We then initialize them inside of the "setUp" method, referencing the test files in the constructor.

```
public class MyFileReaderTest {

    MyFileReader myFileReader1;
    MyFileReader myFileReader2;
    MyFileReader myFileReader3;

    @BeforeEach
    void setUp() throws Exception {

        // original war_and_peace.txt file
        this.myFileReader1 = new MyFileReader("war_and_peace.txt");

        // test file containing some text from war_and_peace.txt,
with
        // different characters and info
        this.myFileReader2 = new MyFileReader("test1.txt");

        // test file containing some text from war_and_peace.txt,
with
        // empty lines
```

```
        this.myFileReader3 = new MyFileReader("test2.txt");

    }
```

We have also defined a “testGetCleanContent” method for you (see below) which tests the “getCleanContent” method. It does this by using the test instances of *MyFileReader* and calling the “getCleanContent” method. First it uses “myFileReader1” to load text and compare a subset of the resulting lines to a test ArrayList<String> of expected lines. Then it uses “myFileReader2” to load text and compare the entire result to a test ArrayList<String> of expected lines. Pass the tests provided then write additional tests where it says “// TODO”.

```
@Test
public void testGetCleanContent() {

    // test original war_and_peace.txt file
    ArrayList<String> actual = myFileReader1.getCleanContent();
    ArrayList<String> expected = new ArrayList<String>();
    expected.add("Title: War and Peace");
    expected.add("Author: Leo Tolstoy");
    expected.add("Translators: Louise and Aylmer Maude");
    expected.add("Posting Date: January 10, 2009 [EBook #2600]");
    expected.add("Last Updated: January 21, 2019");
    expected.add("Language: English");
    expected.add("WAR AND PEACE");
    expected.add("By Leo Tolstoy/Tolstoi");

    // assert first 8 lines containing text
    for (int i = 0; i < expected.size(); i++) {
        assertEquals(expected.get(i), actual.get(i));
    }

    // test shorter file
    actual = myFileReader2.getCleanContent();
    expected = new ArrayList<String>();
    expected.removeAll(expected);
    expected.add("Lines Other Info");
    expected.add("\"The Project Gutenberg EBook of War and Peace, by
Leo Tolstoy\"");
    expected.add("Author: Leo Tolstoy");
    expected.add("CHAPTER I");
```

```
    expected.add("\"Well, Prince, so Genoa and Lucca are now just
family estates of the\"");
    expected.add("\"Buonapartes. But I warn you, if you don't tell me
that this means war,\"");
    expected.add("if you still try to defend the infamies and horrors
perpetrated by that");
    assertEquals(expected, actual);

    // TODO write at least 1 additional test case using a different
    // MyFileReader

}
```

In *WordCounterTest.java* we have defined 2 instances of *MyFileReader* for testing. We then initialize them inside of the "setUp" method, referencing the test files in the constructor.

```
class WordCounterTest {

     MyFileReader fr1;
     MyFileReader fr2;

     @BeforeEach
     void setUp() throws Exception {

          // original war_and_peace.txt file
          fr1 = new MyFileReader("war_and_peace.txt");

          // test file containing some text from war_and_peace.txt,
with different characters and info
          fr2 = new MyFileReader("test1.txt");
     }
```

We have also defined the "testWordCounter" method which first loads the "test1.txt" file, gets the clean contents of the file, generates the word counts, and tests the counts of different words in the generated HashMap<String, Integer> of word counts. It then loads the "war_and_peace.txt" file and tests the counts of different words in that file. Pass the tests provided then write additional tests where it says "// TODO". You'll do this for each unit test method noted in *WordCounterTest.java*

```
@Test
void testWordCounter() {
```

```
    // Get clean lines from test1.txt
    ArrayList<String> linesSol = fr2.getCleanContent();

    // Create new Word Counter
    WordCounter wc = new WordCounter(linesSol);

    // Get map of words and counts
    Map<String, Integer> counters = wc.getWordCounter();

    // Test the count of different words in map
    assertEquals(1, counters.get("Gutenberg"));
    assertEquals(3, counters.get("and"));


    // Get clean lines from war_and_peace.txt
    linesSol = fr1.getCleanContent();

    // Create new Word Counter
    wc = new WordCounter(linesSol);

    // Get map of words and counts
    counters = wc.getWordCounter();

    // Test the count of different words in map
    int atCount = counters.get("at");
    System.out.println("atCount: " + atCount);
    assertTrue(atCount >= 3700 & atCount <= 4700);


    // TODO write at least 3 additional test cases
    // Recommended: Test the counts of other words in the map.
    // (Remember, words are case-sensitive.)

}
```

In *MyFileWriterTest.java* we have defined 1 instance of *MyFileWriter* for testing. We then initialize it inside of the "setUp" method, specifying a unique output file name in the constructor.

```
public class MyFileWriterTest {
```

```
    MyFileWriter myFileWriter1;

    @BeforeEach
    void setUp() throws Exception {
        // test output file
        this.myFileWriter1 = new MyFileWriter("output_test.txt");
}
    }
```

We have also defined a "testWriteToFile" method for you (see below) which is for testing the "writeToFile" method. Write the tests where it says "// TODO" by creating a test ArrayList<String> of words, and using the instance of *MyFileWriter* to call the "writeToFile" method. Then call the "readWrittenFile" helper method to open, read, and test the outputted file and compare the contents with the original test ArrayList<String> of words.

```
@Test
public void testWriteToFile() {

    // TODO write at least 1 test case using myFileWriter1
    // Hint(s):
    // - Create an ArrayList<String> to store a list of words to
write
    // to the test file "output_test.txt"
    // - Call writeToFile to write the list of words to the file
    // - Call readWrittenFile to read the written file and compare
its
    // contents to the defined ArrayList of words above

}
```

Main Method

The "main" method in the *Main* class has also been defined and **implemented for you** (see example below). It takes care of calling all of the necessary methods, in order, in the program. Nothing needs to be completed in the main method.

```
/**
 * Main class to control the flow of the program.
 */
public class Main {

    public static void main(String[] args) {
```

```
        // Create new File Reader
        MyFileReader fr = new MyFileReader("war_and_peace.txt");

        // Create new File Writer
        MyFileWriter fw = new MyFileWriter("output.txt");

        // Get clean lines from the file
        ArrayList<String> lines = fr.getCleanContent();

        // Create new Word Counter with the clean lines
        WordCounter wc = new WordCounter(lines);

        // Get word count map
        Map<String, Integer> counters = wc.getWordCounter();

        // Get and print the counts of some words
        System.out.println(counters.get("Still"));
        System.out.println(counters.get("still"));
        System.out.println(counters.get("in"));

        // Get the words repeated 5000 times or more
        ArrayList<String> words =
    wc.getWordsOccuringMoreThan(5000);

        // Write the words to a file
        fw.writeToFile(words);
    }

}
```

Tips for this Assignment

In this assignment, some tips are given as follows:

- String Methods:
  - For removing leading and trailing spaces from Strings, consider the trim() method.
  - For splitting a String on a delimiter, consider the split() method
  - See additional String methods here: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html
- ArrayLists:
  - ArrayLists have many useful methods, such as add(), etc. For a list of all methods available with ArrayLists, see: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

- Maps:
    - Maps have many useful methods, such as get(), put(), etc. For a list of all methods available with Maps, see: https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

Submission

You have been provided with *Main.java*, *WordCounter.java*, *MyFileReader.java*, and *MyFileWriter.java*, which are the main program files. In addition, you have been provided with *war_and_peace.txt*, which is the file to load and read. You have also been provided with *WordCounterTest.java*, *MyFileReaderTest.java*, and *MyFileWriterTest.java* for testing. There are multiple test files to be used by *MyFileReaderTest.java* for loading and reading. These include: *test1.txt* and *test2.txt*.

To complete the assignment, implement the methods in *WordCounter.java*, *MyFileReader.java*, and *MyFileWriter.java*, making sure you pass all the tests in *WordCounterTest.java*, *MyFileReaderTest.java*, and *MyFileWriterTest.java* respectively. (*MyFileReaderTest.java* will load and read the test files noted above.) Then write **additional and distinct test cases** for each unit test method noted in *WordCounterTest.java*, *MyFileReaderTest.java*, and *MyFileWriterTest.java*. Do not modify the name of the methods in any of the files or the automated testing will not recognize it.

You will submit the following files for this assignment: *Main.java*, *WordCounter.java*, *MyFileReader.java*, *MyFileWriter.java*, *war_and_peace.txt*, and *output.txt*. You will also submit all of the testing files, including: *WordCounterTest.java*, *MyFileReaderTest.java*, *MyFileWriterTest.java*, *test1.txt*, *test2.txt*, and any other files you created for testing. Make sure your program and the unit testing files run without errors! Submit the completed program using the steps outlined in the assignment in Coursera.

Evaluation

This assignment is evaluated, so that you can assess how well you understand the material. However, the evaluation of the assignment will not affect your course grade.

Points

- *MyFileReader* class (5 pts)
  - getCleanContent() - 5 pts
    - Did you parse the file and store the clean content to the array list correctly?
- *MyFileWriter* class (5 pts)
  - writeToFile(ArrayList<String> words) - 5 pts
    - Did you write the file correctly?
- *WordCounter* class (13 pts)
  - WordCounter() - 3 pts
  - generateWordCounts() - 5 pts
  - getWordsOccuringMoreThan(int threshold) - 5 pts
- Did you include at least the required number of additional distinct and valid test cases for each test method? Do all of your tests pass? (10 pts)
  - MyFileReaderTest => testGetCleanContent() - 2 pts
    - [**1 additional test**]
  - MyFileWriterTest => testWriteToFile() - 2 pts
    - [**1 additional test**]
  - WordCounterTest => testWordCounter() - 2 pts
    - [**3 additional tests**]
  - WordCounterTest => testGetWordsOccuringMoreThan() - 2 pts
    - [**3 additional tests**]
  - WordCounterTest => testGenerateWordCounts() - 2 pts
    - [**3 additional tests**]