

# Assignment#3

# Objects, Hashing and Searching

## Submission Requirements

Complete the following assignment and submit electronically in the assignments folder on myCanvas as an IntelliJ Project – Zip the entire folder (Assignment3.zip). This assignment must be completed individually. You may not use ChatGPT, any other AI coding assistant or external source.

## Background / Problem Summary

You are to write a program that will count the words in the book **War and Peace by Leo Tolstoy**. In addition to counting words, you must spell check the document and report how many words are not found in the provided dictionary. You will also match occurrences of the word “War” with the word “Peace” and find the average distance between the closest set of each pair of these words.

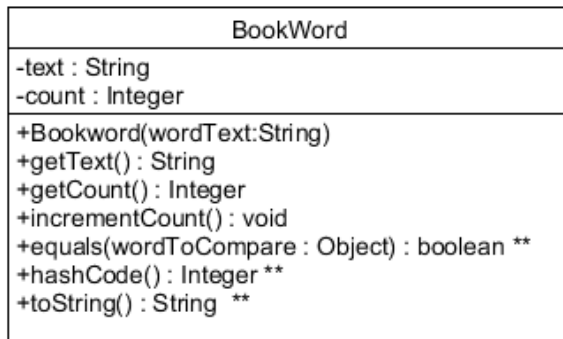
You must use a class called BookWord (provided as UML below) to store each word for the words in the novel and for the words in the provided dictionary. The words must be stored without concern for case sensitivity (all characters must be converted to lowercase).

A word is considered to be one or more characters in length separated by a period, question mark, exclamation point, white space (space character, tab, new line character), double quote, either round bracket, a comma, underscore, hyphen, colon, semicolon or carriage return. The regex expression below can be used to implement this functionality with the `.useDelimiter` method of the scanner object.

```
String regex = "\\.|\\?|\\!|\\s|\\\"|\\(|\\)|\\,|\\_|\\-|\\:|\\;|\\n":
```

Other punctuation marks such as single quote, apostrophe, forward slash and backslash may appear in the input, but for the purpose of this assignment we will treat these as words with spelling errors. For example, the string "it's" is a contraction which is to be read as "it is" or "it has", and is actually 2 words. This is too complicated for this assignment, we will just say anything like this is a word with a spelling error.

## BookWord Class UML



\*\* These methods exist in the Object class and **must be overridden** in your BookWord class

BookWord will be used to hold the characters for each unique word as a string along with a count which will contain the number of occurrences of that word in the novel. The main method will create an ArrayList of BookWord to hold the Words for the novel. The starting code should be used to read the words from the file.

The **us.txt** file is a dictionary of words. You must use the dictionary to find the number of misspelled words (defined as not present in the provided dictionary) in the novel. You will need to create 2 dictionaries for this lab.

- The first must be an ArrayList<BookWord>. Sort the ArrayList using the Collections.sort method once all dictionary has been stored in the ArrayList.
- The second must be a SimpleHashSet<BookWord>. Do not use the HashSet or HashMap that are supplied as part of the Java Collection Framework.

### Steps:

- Create a project in IntelliJ.
- Download SimpleHashSet.java, the novel (WarandPeace.txt), and the dictionary (US.txt) from MyCanvas. Add the US.txt and the WarAndPeace.txt file to the src folder.
- Complete the BookWord class based on the UML provided.
- Provide correct implementation for toString, equals and hashCode.
  - The hashCode method return a hash value for each BookWord. The hashCode must only use constant components in the class for the calculation. Find a good hashing algorithm on the Internet. You may not use a built-in hashing method.
  - Do not use String.hashCode() or Object for this assignment.
  - Site your reference in the source code for the hashing algorithm used.
    - You may not cite ChatGPT, you must find an explanation of the hash that you use, and include the URL in a comment in your code.
- Confirm that your algorithm produces good hashes. If you have less than 10% empty buckets in your HashSet on the Dictionary Words your algorithm is acceptable. The public methods in the SimpleHashSet class will calculate this value.

## Part A – Hashing / Counting and Spelling

Use formatted print statements so all numbers are readable, i.e.

```
System.out.printf("There are a total of %,d words in %s.\n",
                  uniqueWords, filename);
```

Ensure your application prints all of the following information to the standard output.

- Total # of words in the novel.
- Total # of unique words in the file.
- The # of words that are not contained in the dictionary. All three methods below should yield the same number of words. You must implement all three techniques and measure the performance of each. At the top of the code you must include a discussion of the results including an analysis of the performance and whether or not this lines up with the theory for the data structure and algorithms used.
  1. Use the **contains** method of `ArrayList` to find matches.
  2. Use the **binarySearch** method of the `Collections` class (**`Collections.binarySearch`**) to search the `ArrayList` dictionary. Supply a Lambda expression for the binary search of the Dictionary.
  3. Use the `SimpleHashSet` dictionary and the **contains** method provided.
- The run time in ms for Part A

Example output for Part A:

```
Assignment #3, analysis of src/WarAndPeace.txt
```

```
PART A
```

```
=====
```

```
There are a total of ???,??? words in src/WarAndPeace.txt.
```

```
There are a total of 20,213 unique words in src/WarAndPeace.txt
```

```
Dictionary Search, timing for 3 methods:
```

```
LINEAR SEARCH - ?,??? words not found in dictionary - time = ?,??? ms
```

```
BINARY SEARCH - ?,??? words not found in dictionary - time = ?,??? ms
```

```
HASHSET SEARCH - ?,??? words not found in dictionary - time = ?,??? ms
```

```
TOTAL TIME for all of PART A = ?,??? ms
```

## Part B – War and Peace

The words 'war' and 'peace' appear in the novel many times. Starting at the beginning of the novel you are to match each occurrence of the word peace with the closest occurrence of the word war. War may appear before or after peace. Once a pair has been made the occurrence of each word must not be used again in other matches. Note that the order can make a difference so to match the expected results you must find **peace first** and then look for the closest occurrence of war.

One way to determine this is to perform a Proximity Search ([https://en.wikipedia.org/wiki/Proximity\\_search\\_\(text\)](https://en.wikipedia.org/wiki/Proximity_search_(text))). One method to calculate a numerical proximity is to determine the proximity distance between one word and another in terms of word position. To do this, each word must be numbered starting from 1 in the order they appear in the text. As an example, if we wanted to find the proximity distance between war and peace in the text below, we would number each word starting at 1 from the beginning of the text.

**War and peace, peace and war. War, War, War. Give Peace a chance, we all want peace**  
(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14)(15)(16) (17)

We could then start pairing off each occurrence of peace with the closest war in the text. The first pair would be 3 and 1, The second pair would be 4 and 6, The third pair would be 11 and 9. And the last pair would be 17 and 8. The occurrence of war at position 7 is ignored as there is no occurrence of peace to match with.

Sample analysis for the test case shown above.

```
shortest distance between war at pos(3) and peace(1) = 2
shortest distance between war at pos(4) and peace(6) = 2
shortest distance between war at pos(11) and peace(9) = 2
shortest distance between war at pos(17) and peace(8) = 9
```

Sample output. Your submission should generate just two lines for Part B that show the following two metrics. Also report the total run time for the whole program, Part A + Part B.

**The total sum of distances between the matched pairs 'war' and 'peace' = 15**  
**The average distance between the matched pairs 'war' and 'peace' = 3.75**

**TOTAL RUNTIME (Part A + Part B) = ?,??? ms**

For Part B, the only output you should show are the summary results “total sum” and “average distance” and run time. The example #s 15 and 3.75 are for the test sentence, you should expect larger #s. You may find it helpful to generate some debugging messages like those highlighted in yellow, but remove these print statements before you submit your work.

**Your program must run in a reasonable amount of time.** If the total execution time for Part A and Part B combined is longer than **10 seconds** you will not receive full credit. If your program is still running after the 10 second mark, your professor may simply stop the program, and grade the output generated so far, treating the other parts as incomplete / not working.

## Marking Scheme

Make sure you include a statement of authorship at the top of your code.

Reminder, you may not use ChatGPT or any other AI tool. Your work must be your own. Your program should not take longer than 10 seconds to complete. Your professor may terminate execution of your program at their discretion and mark any output generated so far.

Program structure – Comments, follows best programming practices - 15%

Program Implementation – BookWord class implemented as required and complete and includes hashCode method with a cited reference for the hashing method – 10%

Program stores novel words into an ArrayList of BookWord and stores dictionary into both an ArrayList and SimpleHashSet of BookWord – 15%

Part A) - Output matches requirement. Clean output formatting, < 10 secs run time – 20%

Part B) – Metrics for matched pairs output to screen with the total distance and the average distance output to 2 decimal places. Clean output formatting, < 10 secs run time -20%

Discussion in comment at top of code – Dictionary lookup comparison discussion completed for ArrayList (linear search and binary search) and using the SimpleHashSet data structure – 20%