**Data Structures and Concurrency : Continuous Assessment 1**

**October 2018**

**CA Report Due Friday , November 9th 2018**

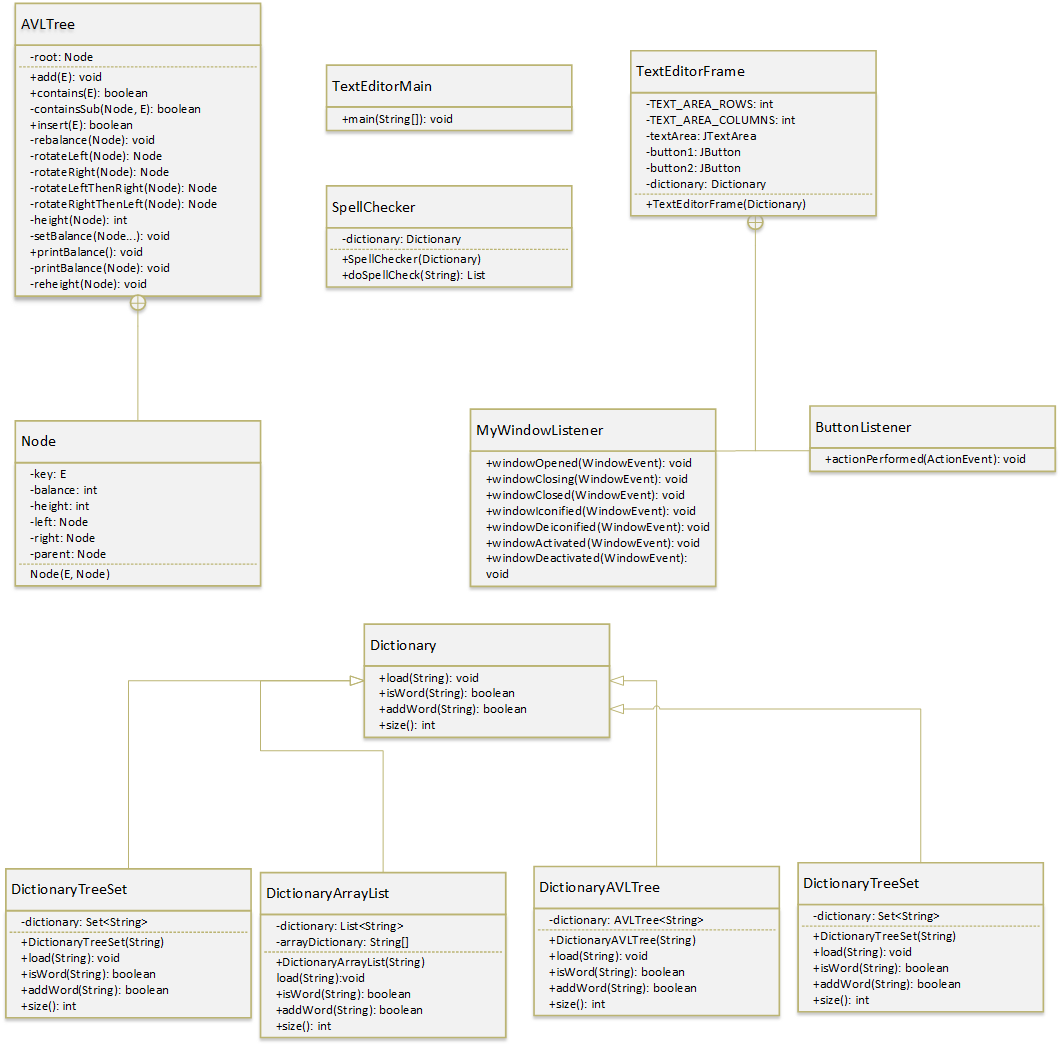
Please email this report to me by this date.

**N.B. Please save all the code you develop as you may be requested to demonstrate it.**

Student Name Michael Edgar

Submission Date November 9th 2018

Note: When you are asked to do unit testing, you should use JUnit5 and your test should be in a folder named tests. (This is how we did unit testing in class)

**1.** Draw a class diagram for the application, either by hand, or use whatever case tool you are familiar with.

**2.** AVLTree class has a recursive contains method.

(a) Write an **iterative** version of this method. Insert the method here:

**public boolean** containsIterative (E word) {  
 **boolean** results = **false**;  
 **int** result;  
  
 **if**(**root** != **null**) {  
 Node currentNode = **root**;  
  
 **while**(currentNode != **null**) {  
 result = currentNode.**key**.compareTo(word);  
  
 **if**(result > 0) {  
 *//Go Left* **if**(currentNode.**left** == **null**) {  
 currentNode = **null**;  
 }  
 **else** {  
 currentNode = currentNode.**left**;  
 }  
 }  
 **else if** (result < 0) {  
 *//Go Right* **if**(currentNode.**right** == **null**){  
 currentNode = **null**;  
 }  
 **else** {  
 currentNode = currentNode.**right**;  
 }  
 }  
  
 **else** {  
 results = **true**;  
 currentNode = **null**;  
 }  
 }  
 }  
 **return** results;  
}

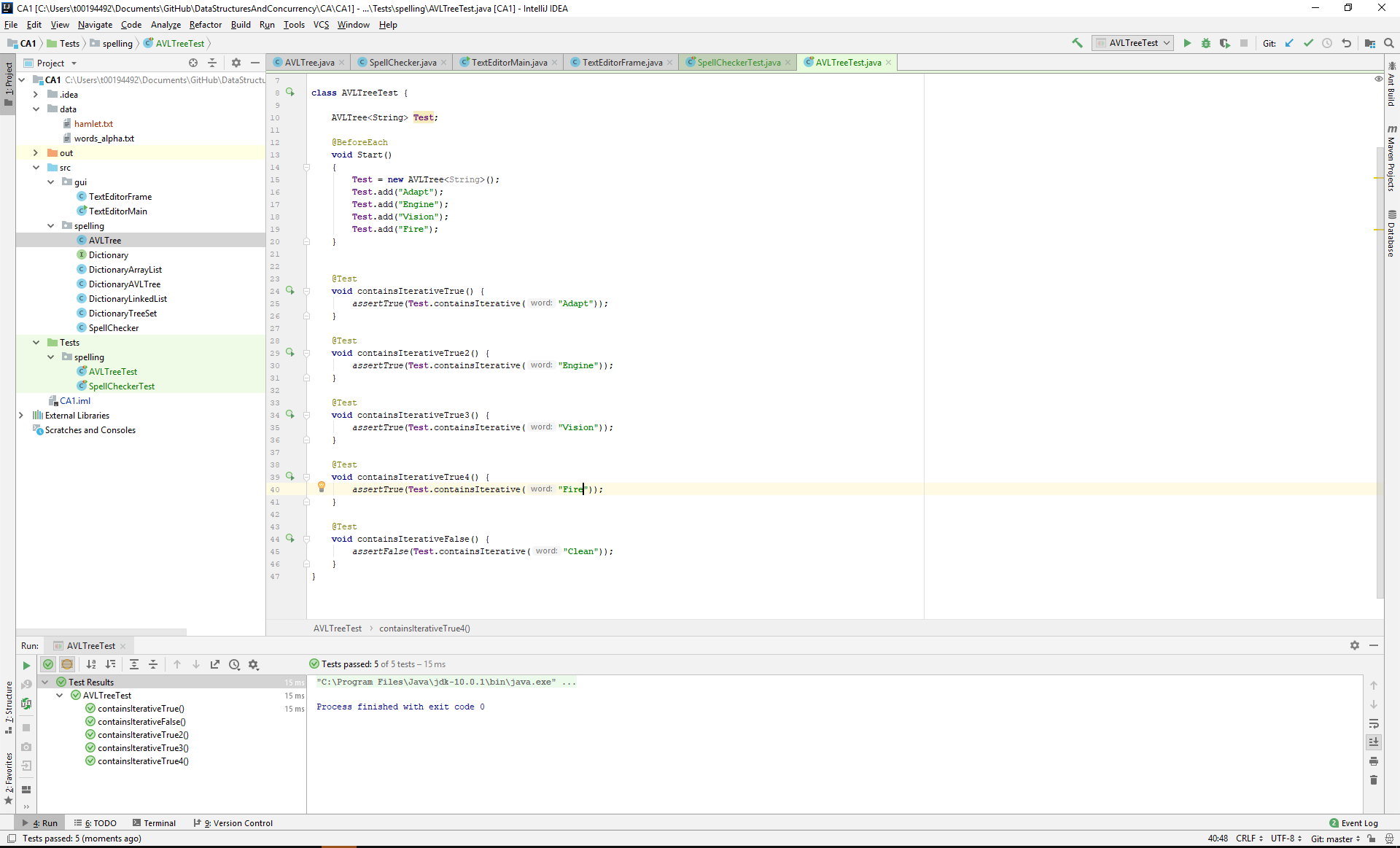
(b) Do unit testing of this method by populating an AVLTree instance with a few values and then testing the method.

Insert the unit test code here:

**package** spelling;  
  
 **import** org.junit.jupiter.api.BeforeEach;  
 **import** org.junit.jupiter.api.Test;  
  
 **import static** org.junit.jupiter.api.Assertions.\*;  
  
**class** AVLTreeTest {  
  
 AVLTree<String> **Test**;  
  
 @BeforeEach  
 **void** Start()  
 {  
 **Test** = **new** AVLTree<String>();  
 **Test**.add(**"Adapt"**);  
 **Test**.add(**"Engine"**);  
 **Test**.add(**"Vision"**);  
 **Test**.add(**"Fire"**);  
 }  
  
  
 @Test  
 **void** containsIterativeTrue() {  
 *assertTrue*(**Test**.containsIterative(**"Adapt"**));  
 }  
  
 @Test  
 **void** containsIterativeTrue2() {  
 *assertTrue*(**Test**.containsIterative(**"Engine"**));  
 }  
  
 @Test  
 **void** containsIterativeTrue3() {  
 *assertTrue*(**Test**.containsIterative(**"Vision"**));  
 }  
  
 @Test  
 **void** containsIterativeTrue4() {  
 *assertTrue*(**Test**.containsIterative(**"Fire"**));  
 }  
  
 @Test  
 **void** containsIterativeFalse() {  
 *assertFalse*(**Test**.containsIterative(**"Clean"**));  
 }  
}

Specify if the method passed. Yes

Also insert a screenshot of the results of the test:



(c) Which method, iterative or recursive, do you think will be more efficient?

Give reasons for your answer.

The iterative method will be more efficient, as it calls less methods than the recursive method. This ensures it doesn’t overflow the runtime stack with data.

**3.** Do unit testing of the doSpellCheck method of SpellChecker class. The method returns a list of the misspelt words.

(a) For this use a small text file to run the spelling check on e.g. Hamlet.txt is included in the

CA 1 October 2018 folder or create a small file of your own. Give the name and contents of the file here:

Hamlet.txt ,

“To be, or not to be,

That is the question.”

(b) Create a very small dictionary file – call it small\_dictionary.txt. It should contain some of the words in your small text file created in (a) but not all. Also it should contain some words not in your text file. There should be only one word per line. Give the contents of the file here:

apple

be

engine

not

oboe

that

the

question

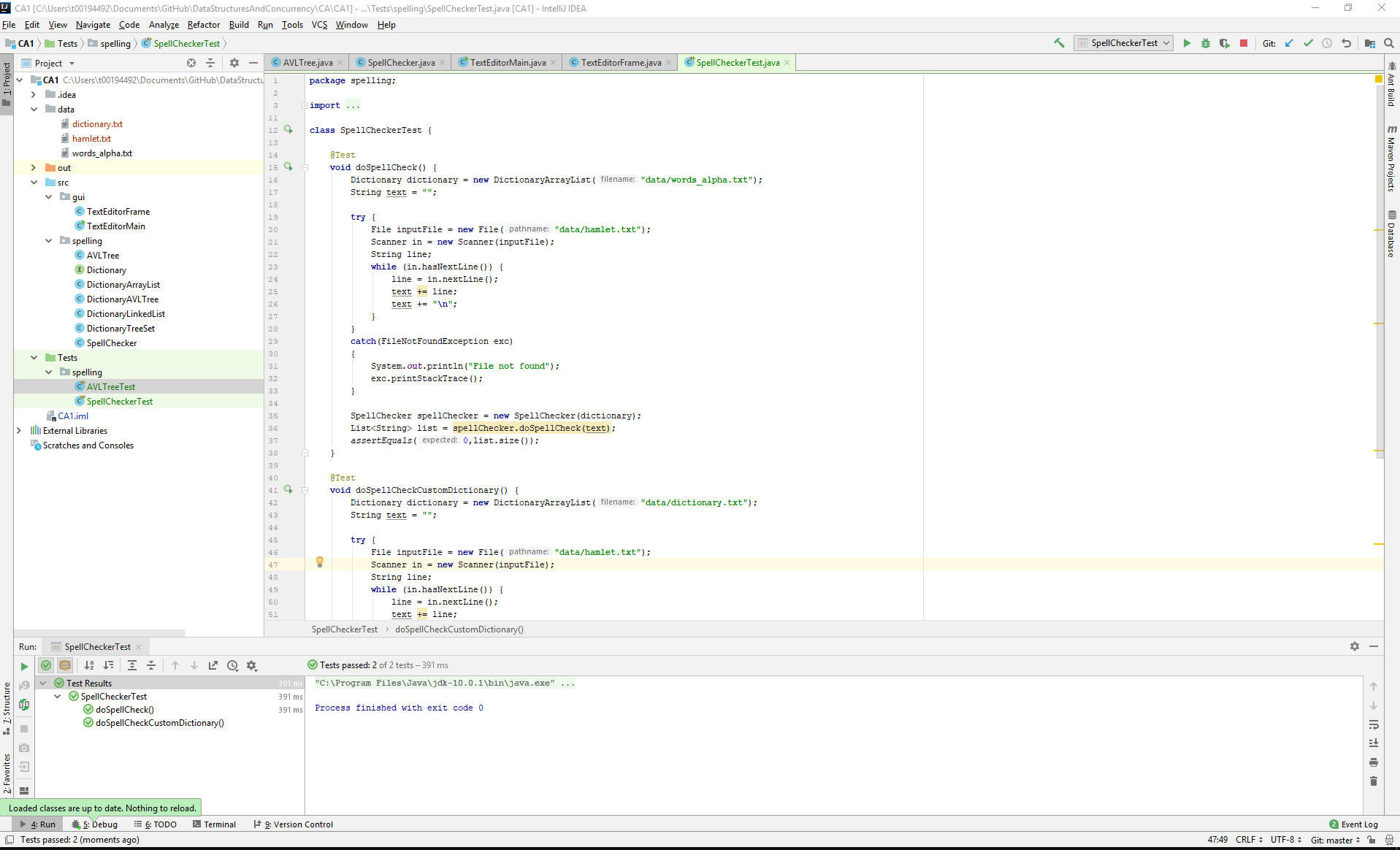
(c) Now do unit testing of the doSpellcheck method.

Insert the unit test code here:

**package** spelling;  
  
**import** org.junit.jupiter.api.Test;  
  
**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.List;  
**import** java.util.Scanner;  
  
**import static** org.junit.jupiter.api.Assertions.\*;  
  
**class** SpellCheckerTest {  
  
 @Test  
 **void** doSpellCheck() {  
 Dictionary dictionary = **new** DictionaryArrayList(**"data/words\_alpha.txt"**);  
 String text = **""**;  
  
 **try** {  
 File inputFile = **new** File(**"data/hamlet.txt"**);  
 Scanner in = **new** Scanner(inputFile);  
 String line;  
 **while** (in.hasNextLine()) {  
 line = in.nextLine();  
 text += line;  
 text += **"\n"**;  
 }  
 }  
 **catch**(FileNotFoundException exc)  
 {  
 System.***out***.println(**"File not found"**);  
 exc.printStackTrace();  
 }  
  
 SpellChecker spellChecker = **new** SpellChecker(dictionary);  
 List<String> list = spellChecker.doSpellCheck(text);  
 *assertEquals*(0,list.size());  
 }  
  
 @Test  
 **void** doSpellCheckCustomDictionary() {  
 Dictionary dictionary = **new** DictionaryArrayList(**"data/dictionary.txt"**);  
 String text = **""**;  
  
 **try** {  
 File inputFile = **new** File(**"data/hamlet.txt"**);  
 Scanner in = **new** Scanner(inputFile);  
 String line;  
 **while** (in.hasNextLine()) {  
 line = in.nextLine();  
 text += line;  
 text += **"\n"**;  
 }  
 }  
 **catch**(FileNotFoundException exc)  
 {  
 System.***out***.println(**"File not found re-enter file"**);  
 exc.printStackTrace();  
 }  
  
 SpellChecker spellChecker = **new** SpellChecker(dictionary);  
 List<String> list = spellChecker.doSpellCheck(text);  
 *assertNotEquals*(0,list.size());  
 }  
}

Specify if the method passed. yes

Also insert a screenshot of the results of the test:



**4.** There are a few implementations of the Dictionary interface.

Explain what algorithm the contains method of DictionaryArrayList uses. Give code from the application to support your answer (Specify name of class and method):

The isWord method of DictionaryArrayList uses a recursive algorithm. This is done by calling the binarySearch method in the Arrays class. This is shown in the code below.

@Override  
**public boolean** isWord(String s) {  
 **int** result = Arrays.*binarySearch*(**arrayDictionary**, s);  
 **if** (result >= 0)  
 **return true**;  
 **else return false**;  
}

**public static int** binarySearch(Object[] a, Object key) {  
 **return** *binarySearch0*(a, 0, a.**length**, key);  
}

*// Like public version, but without range checks.***private static int** binarySearch0(Object[] a, **int** fromIndex, **int** toIndex,  
 Object key) {  
 **int** low = fromIndex;  
 **int** high = toIndex - 1;  
  
 **while** (low <= high) {  
 **int** mid = (low + high) >>> 1;  
 @SuppressWarnings(**"rawtypes"**)  
 Comparable midVal = (Comparable)a[mid];  
 @SuppressWarnings(**"unchecked"**)  
 **int** cmp = midVal.compareTo(key);  
  
 **if** (cmp < 0)  
 low = mid + 1;  
 **else if** (cmp > 0)  
 high = mid - 1;  
 **else  
 return** mid; *// key found* }  
 **return** -(low + 1); *// key not found.*}

**5.** Profiling of doSpellCheck method

See explanation in Notes 6 in “CA October 2018 Information”. Complete the following table:

**Time taken to do the spell check**

|  |  |  |  |
| --- | --- | --- | --- |
| Dictionary using: | Big Oh value | Time 1 – see below | Time from profiler |
| ArrayList | O(n) | 16 |  |
| LinkedList | O(n) | 516 |  |
| AVLTree | O(log n) | 0 |  |
| TreeSet | O(log n) | 0 |  |

Name of text file used: PrideAndPrejudice.txt

Source of file used: Gutenberg.org

Obtained with intel core i7 3770 processor, 3.4 GHz, Java Version 10 , Windows 10 (or specify if not…)

**Notes:**

The values you insert in the Big Oh column are the values for the efficiency of the isWord method of the relevant dictionary i.e. DictionaryArrayList etc.

To find the Big Oh for TreeSet, look up TreeSet in the java api and see the efficiency for its contains method.

**Time 1** is the time got by getting the time before and after the relevant code executes and computing the difference between them.

Like we did in Lab 1 – use System.*currentTimeMillis*();

**6.** Any comments/suggestions for improvements, additions etc:

**7.** References/Sources of information. Specify any sources you used.

***Important Note: Plagiarism and Academic Dishonesty***

Please familiarize yourself with IT Tralee Anti-plagiarism Policy and Procedures document. There is a copy of the document available in blackboard under Admin- Academic Affairs and Student Supports>Policies and Protocols>Anti-Plagiarism Policy.

Please note that if the work you submit is not your own, a mark of 0 will be awarded.