

Week 1: Java Refresher Exercise

Due date: Friday 19th September, 23:59

130016030
University of St Andrews
September 18, 2014

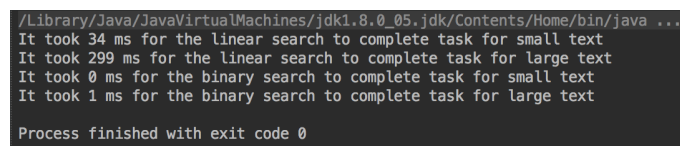
Introduction

This project is a part of a program called "Spelling checker". The checker repeatedly prompts the user to enter a word to be checked. If the word is in the dictionary it is reported as "correct". If it is not, it is reported as "missing" and the words which would have come before and after it in alphabetical order (if any) are printed.

Structure

1 Benchmark package

Benchmark package contains **SpellCheckBenchmark.java** file. It is one of the suggested extensions. This is implementation of timing performance comparator. It has small and large texts, calculates the time required for the binary and linear search algorithm to perform the task.



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_05.jdk/Contents/Home/bin/java ...  
It took 34 ms for the linear search to complete task for small text  
It took 299 ms for the linear search to complete task for large text  
It took 0 ms for the binary search to complete task for small text  
It took 1 ms for the binary search to complete task for large text  
Process finished with exit code 0
```

Figure 1: Comparison between linear and binary searches

This is a good example of the time complexity for both search algorithms.

Linear is $T(n) = \Theta(n)$, whereas binary search performance $\Omega(\log n)$

2 UI package

UI package contains two user interfaces: console-based and graphical. Both **SpellCheckUI.java** and **SpellCheckGUI.java** has a main method as it was said in README file. However, GUI is using binary search algorithm, whereas console-based use linear. As it was described before, it is done due to performance improvement.

Additionally, it has been decided to make own extensions. Firstly, user can choose the path to the dictionary or use default one. Secondly, user can set more than a one word and the appropriate message will be displayed.

```

/Library/Java/JavaVirtualMachines/jdk1.8.0_05.jdk/Contents/Home/bin/java ...
Would you like to use a default path "/usr/share/dict/words" or your own path?
Please type:
  1: If you want to use default.
  2: If you want to use your own.
Your choice: 1
OK. Please enter words to check. Enter "quit" to finish.
Next word(s): computerz zz comrade computing aab
computerz not found. Nearest neighbour(s): computer and computist
zz not found. Nearest neighbour(s): zyzzogeton
comrade correct
computing not found. Nearest neighbour(s): computer and computist
aab not found. Nearest neighbour(s): aa and aal
Next word(s): quit
Goodbye
Process finished with exit code 0

```

Figure 2: Typical session in console-based interface

It can be clearly seen that the program deals with several words and show messages. When the "zz" word was typed only one word was suggested because it was at the end of the dictionary. Even if it was the beginning of the dictionary, only one word will be displayed.

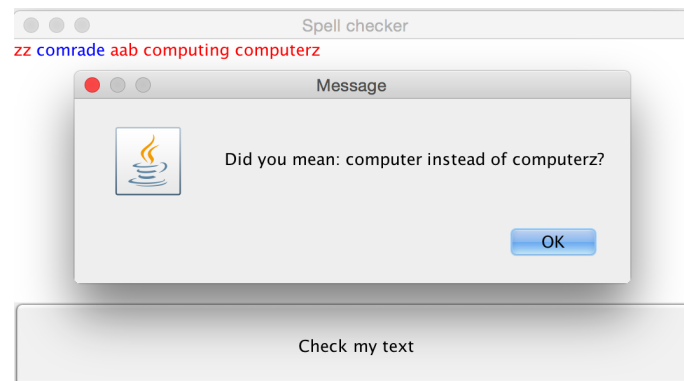


Figure 3: Typical session in graphical user interface

GUI is reasonably simple. It has only one button, which is used to check the text. If the word is correct it will appear blue, otherwise the error message will be displayed suggesting possible alternatives and text will be red. After that, next text you type will be black, but if you delete a previous word, the color will change as well. These are another extensions.

3 Main package

ISpellChecker.java is an interface defining the check method of SpellChecker. As it was required in specification, this interface was not changed.

DictionaryLoader.java provides a static method `loadTheDictionary` which reads the words from a file and returns them as an array of strings in alphabetical order.

SpellChecker.java stores the array of strings and provides a method `check` to search it. There are two constructors: first one sets default path and another use custom path specified by user, which is another extension. Check method is an implementation of the linear search algorithm, which is used by console-based interface. `BsCheck()` is the recursive binary search algorithm used by GUI. It has been decided to use `String.compareTo()` method to compare the two strings.

Another extension is to provide possible suggestions for the misspelled word. The algorithm will be discussed at the next page.

```

int index = bsCheck(word).getIndex();

// If word is at the beginning of a dictionary
if (index <= 10){
    for (int i = 0; i < 10; i++) {
        suggestions[i] = new SpellCheckerSuggestion
            (wordsInDictionary[i],
             findNumOfSimilarChar(word, wordsInDictionary[i]));
    }
}
// If word is at the end of a dictionary
else if (wordsInDictionary.length - index <= 10) {
    for (int i = 0; i < 10; i++) {
        suggestions[i] = new SpellCheckerSuggestion
            (wordsInDictionary[index-i-1],
             findNumOfSimilarChar(word, wordsInDictionary[index-i-1]));
    }
}
// If word is somewhere else
else {
    // Look up 5 words before and after the given word
    for (int i = 0; i < 10; i++) {
        suggestions[i] = new SpellCheckerSuggestion
            (wordsInDictionary[index - 5 + i],
             findNumOfSimilarChar(word, wordsInDictionary[index-5+i]));
    }
}

```

From the code above it can be seen that there are three main cases. First of all, define index using binary search. If the word at the beginning of a dictionary, then all 10 words after it will be added to the array. If the word at the end, 10 words before will be added. Otherwise, 5 words before the word and 5 words after it will be added to the array.

All of the words are passed through the function that compares two strings and finds number of similar characters. After that, the array will be sorted to find max value and the most suitable suggestion.

SpellCheckerSuggestion.java is used to store number of similar characters and the given word. This class maps those two value which is required to find the most possible word.

SpellCheckResult.java is a class used by SpellChecker to return results. A SpellCheckResult object has public fields indicating whether the word was correctly spelt and, if not, what were the words immediately before and after it. If it was before the first dictionary entry or after the last one then one of these fields might be null. It was slightly modified to store the index of the word. This is also used to provide a suggestion.

4 Parser package

SpellCheckerParser.java contains a parse method which splits the string into individual words. It also checks array to make sure that it contains only words and not integers or doubles. This is another extension to this practical. It is done to ensure that text contains valid values.

Testing

To ensure that this program does not break, input values were from at the beginning, middle and the end of the dictionary. For instance, word "zz" is not in the dictionary, but if it was it will be at the end if it. In this case, only one word will be printed (Figure 2). The same test was done if the word was possibly at the beginning of the page.

Another extension of this project is ensure that input does not have any numbers in it. This check can be seen in the Parser class. For now the error message will be displayed. However, the actual spell checker does not behave in this way. For the future improvement, these numerical values might be ignored to improve performance of the algorithm.

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_05.jdk/Contents/Home/bin/java ...  
Would you like to use a default path "/usr/share/dict/words" or your own path?  
Please type:  
1: If you want to use default.  
2: If you want to use your own.  
Your choice: 1  
OK. Please enter words to check. Enter "quit" to finish.  
Next word(s): qwerty 1234 23 5  
1234 is a number. Please enter the word instead  
Process finished with exit code 2
```

Figure 4: Error numerical value in console-based interface

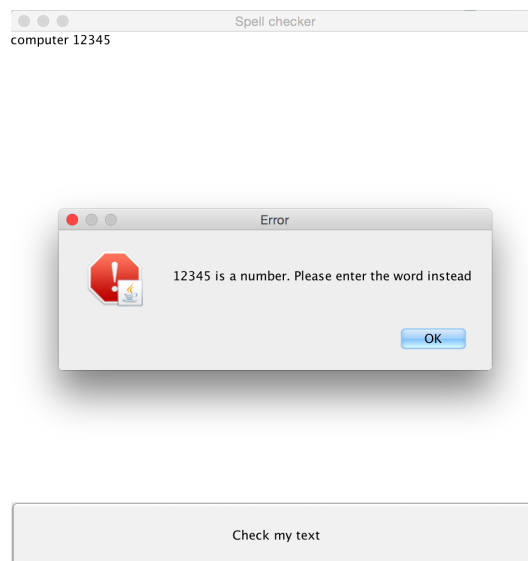


Figure 5: Error numerical value in graphical user interface

Conclusion

This project includes javadoc, Apache ANT build, source code and this report. Please make sure that you have Java 8 to run this program. There are three main methods in: SpellCheckerUI, SpellCheckerGUI and SpellCheckBenchmark. Images in this report can be also accessed in the /img folder in case you cannot see it. Please make sure that you have a **Java 8** to run it.