COMP 2140 Assignment 4: Spellchecking with a Hash Table

Cuneyt Akcora

Programming Standards

When writing code for this course, follow the programming standards, available on this course's website on Desire2Learn. Failure to do so will result in the loss of marks.

Objective

To check the spelling in a document. Note that in this assignment you will start without a given Java file. You will write everything from scratch.

Your Program

General Overview: Most spell checkers use hashing to determine if a given word in a document is correct (that's why they can check your spelling as you type).

The Dictionary: First, you must set up a dictionary in which to look up words. Use either `JFileChooser` or a `Scanner` to allow the user to specify the dictionary file to load. (The dictionary file is `words.txt`, which can be found in the Content Browser on D2L with this assignment in an Assignment 3 folder under Assignments.) Read in the dictionary file. There is one word per line. Insert each word in the dictionary into a Table ADT.

Implement the Table ADT as follows:

- You will create your own hash table ADT. Do not use existing HashTable or HashMap from Java collections. Do not implement a hash table every time you need one. Implement once, and create instances with different load factors.
- Use a hash table of size 94321 (a prime number that gives a load factor of about 0.5).
- Use separate chaining (in which each table slot is a linked list of nodes) to resolve collisions.
- Use the polynomial hashing algorithm discussed in class to hash each word — use Horner's method to compute the polynomial hash code and take the result of each operation mod the table size to avoid integer overflow.

Checking the spelling in a file: Next, you must check the spelling of all words in another file, a document file.

Again, use either `JFileChooser` or a `Scanner` to allow the user to specify the document file to load. (The file is `GeorgeEliot_SilasMarner.txt`, which can be found in the Content Browser on D2L with this assignment in an Assignment folder.) The document file contains multiple words per line.

For each word in the document file, perform a Table search. If a word isn't present in the Table, then it is a spelling error. (Note that `words.txt` is not a complete dictionary, so some words that are spelled correctly will be flagged as erroneous.)

Indicate the spelling errors by printing each incorrect word **once**. With each word, include the line numbers on which the word appears (in the order in which you found them). So an incorrectly-spelled word might generate the following line in the output:

```
Invalid word "criator" found on lines 59 21 21 1
```

Use another hash table, an incorrect-word table, to keep track of the incorrect words (and where they were found in the document):

- This hash table should be size 2797.
- An incorrect word will only be added to the incorrect-word table if the word is not already in the incorrect-word table. Thus, you must do a search for the word, and only add the word to the incorrect-word table if it is not found.
- Each element of the incorrect-word table will contain two things: a word and a stack of the line numbers on which this incorrect word was found in the document file. The stack of line numbers is an ordinary stack: line numbers are stacked at the top of the stack.
- If an incorrect word is already in the incorrect-word table, simply add the new line number to the top of the line number stack for that word.

After the document file is completely processed, traverse the incorrect-word table. For each incorrect word found in the table, print out both the word and the numbers of all lines on which it appeared (removing the line numbers one-by-one from the line number stack).

You should consider the following when implementing your solution:

- The document file that you will read has punctuation that must be handled. You must ensure that no word is seen as error because it had punctuation attached to it. Note that an apostrophe is a valid part of a word and should not be considered punctuation.

  Suggestion: If `String` `inLine` is an input line from the document file, then `inLine.trim().split( "[^a-zA-Z']+" )` splits the line into tokens using one or more characters that are NOT letters or an apostrophe as the splitting pattern ("^" means "not").

- Hashing `"A"` and `"a"` produces different values, but you would not expect them to be different words in a dictionary. To manage this difference, convert all text to all lower case when processing (this comment also applies to the dictionary words). This conversion simplifies things and makes the spell checker more reliable.

  Suggestion: The helpful `String` method `toLowerCase()` will do the job for you.

- You must organize this program into appropriate classes.

Sample output: Your output should take the following form:

```
There are a total of 10 invalid words:
Invalid word "cant" found on lines 1
Invalid word "pewter" found on lines 1
Invalid word "turbine" found on lines 1
Invalid word "Helen" found on lines 59 21 21 1
Invalid word "fortune" found on lines 1
Invalid word "fashionable" found on lines 13 11 11 9 5 3
Invalid word "Popcorn" found on lines 104 90 5
Invalid word "10" found on lines 139 7
Invalid word "Zapp" found on lines 139 7
Invalid word "haberdasher" found on lines 13 13 9
```

Hand-in Instructions

Go to COMP2140 in Desire2learn. Please note the following:

- Submit ONE .java file. The .java file must contain all the source code. The .java file must be named `A4<your last name><your first name>.java` (e.g., `A4AkcoraCuneyt.java`).
- Please do not submit anything else.
- We only accept homework submissions via D2L. Please DO NOT try to email your homework to the instructor or TAs.
- We reserve the right to refuse to grade the homework or to deduct marks if these instructions are not followed.