CS 3345 Data Structures Project 2

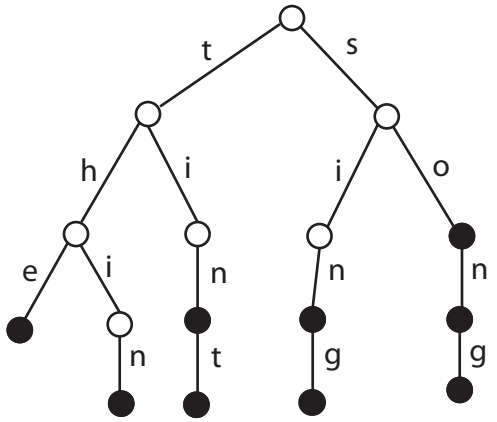
Write a class to implement a Trie data structure and a program to test it

The trie efficiently stores words, as in a spell-checker. To simplify matters, words will be restricted to strings of lower-case characters, excluding spaces or punctuation marks.

The member functions of the Trie class must include:

```
boolean insert(String s);      // returns false if s is already present, true otherwise
boolean isPresent(String s);   // returns true if s is present, false otherwise
boolean delete(String s);      // returns false if s is not present, true otherwise
int membership();              // returns the number of words in the data structure
void listAll();                // list all members of the Trie in alphabetical orber
```

A Trie is a tree of degree 26. The following Trie contains the words: the, thin, tin, tint, sin, sing, so, son, song.



The black Nodes have a boolean variable set to indicate the end of a word.

Here are the class variables for the class Node:

```
class Node {
   boolean terminal;
   int outDegree;
   children Node[];
}
```

Each Node includes an array of 26 references to Node. When a Node is created, all the array elements will be initialized to null and the outDegree will be set to zero. Do not add ANY other variables to the Node class.

In the above Trie, the root node has only two children, corresponding to characters 's' and 't', or positions 19 and 20 in the array of references.

A search begins at the root Node and follows the links, one level at a time. One level for each character in the given word. To find the word "tin" we begin with array element 20 in the root Node and, if it is present,

we check element 8 in the array of the second level node. Finally, we check element 14 in the array of the third level Node. If that Node is marked as a word-terminal, we return "true".

The insert function begins with a search for the given word. If a null reference is found, a sequence of new Nodes is created corresponding to the missing suffix.

To insert the word "soap" to the above Trie, two new nodes would be added, corresponding to 'a' and 'p'. This sequence would be referenced by the third level Node corresponding to the prefix "so".

Deletion also begins with a search for the given word. If found, there are two cases. Say we are deleting "sin" from the above Trie. Since the terminal character of the given word has children, deletion only requires flipping the boolean variable that indicates the end of a word.

If the word "thin" is deleted, the suffix "in" must be unlinked from the third level node. To do this, when recursing out from the end of the given word, remove links to the characters of that word until a Node is found with degree greater than 1, or a terminal node is found. The former case occurs when deleting the word "thin". The word "the" must remain. The latter case occurs when deleting the word "song." The word "so" must remain.

Your program will read a text file from System.in that contains text commands. In response to each command your program will output one or more lines of text to system.out.

Here are the commands:

```
N         // Print your name followed by a newline.
A soap    // Insert the word 'soap'
          // Print one of the strings "Word inserted" or "Word already exists"
          // followed by a newline.
D sin     // Delete the word 'sin'.
          // Print "Word deleted" or "Word not present" followed by a newline.
S fortune // Search for the word "fortune".
          // Print "Word found" or "Word not found" followed by a newline.
M         // Print "Membership is ####" where #### is the number of words in the Trie
C  wa wb wc wd we  // Check the space separated sequence of words wa, wb, wc, wd, we, ...
          // up to the end of the line for presence in the Trie and list any that are
          // not present, one per line, each preceded by the phrase "Spelling mistake".
          // The words in the list may each be up to 20 characters long and the line may
          // be up to 200 characters long.
L         // Print a list of the elements of the Trie in alphabetical order, one word per line.
E         // The end of the input file
```

Here are sample input and output data files:

Sample Input	Sample Output
N	Ivor Page
A ant	Word inserted
A goat	Word inserted
A frog	Word inserted
A art	Word inserted
A goad	Word inserted
A antler	Word inserted
A go	Word inserted
A from	Word inserted
A part	Word inserted
A past	Word inserted
A arts	Word inserted
A part	Word already exists
A frond	Word inserted
A fries	Word inserted
A text	Word inserted
A message	Word inserted

```
A mess                            Word inserted
A mean                            Word inserted
A goal                            Word inserted
A interpretation                  Word inserted
A coat                            Word inserted
M                                 Membership is 20
D art                             Word deleted
D art                             Word not present
D mess                            Word deleted
S art                             Word not found
M                                 Membership is 18
S antler                          Word found
S part                            Word found
S text                            Word found
S freis                           Word not found
C pert post past text             Spelling mistake pert
C coat frond interpretacion       Spelling mistake post
C anteler massage cat mouse goal  Spelling mistake interpretacion
L                                 Spelling mistake anteler
E                                 Spelling mistake massage
                                  Spelling mistake cat
                                  Spelling mistake mouse
                                  ant
                                  antler
                                  arts
                                  coat
                                  fries
                                  frog
                                  from
                                  frond
                                  go
                                  goad
                                  goal
                                  goat
                                  interpretation
                                  mean
                                  message
                                  part
                                  past
                                  text
```

RULES FOR PROGRAMMING AND SUBMISSION:

1. Your submitted code must be your own, except for any code that you copy from this specification.
2. Write your program as one source file and do not use the "package" construct in your Java source.
3. Name your source file as $N_1N_2F_1F_2$P2.java where your given name begins with the characters N_1N_2 and your family name begins with the characters F_1F_2. For example my name is Ivor Page, so my source file will be called IVPAP2.java. Note that in all but the "java" extension, all characters are upper case.
4. Your program must not output any prompts. Its output must exactly match the format of the **Sample Output** above.
5. Do not use any Java Collection Classes, except the arrays to store references in each node. Strings are allowed for input/output.
6. You program must read from System.in and output to System.out.
7. Use good style and layout and comment your code well.

8. Use file redirection to test your program. That's how I will test it.

9. Use the test files provided on the eLearning webpage for this class to test your program. Sample input and corresponding output files will be given.

10. Submit your ONE source code file to the eLearning Assignment Dropbox for this project. Don't submit a compressed file. Don't submit a .class file.

11. **There will be a 1% penalty for each minute of lateness, up to 60 minutes. After 60 minutes of lateness a grade of zero will be recorded.**