**COMP2080**

**Assignment 2 (15%)**

**GROUP SIZE: 2 MAXIMUM**

**Due: 3rd April 2022 11:30PM**

**Given: 17th March 2022**

Submission Requirements:

**The St**u**dent ID number for each member of the group must be commented at the top of the file.**

**Paste the code for each class and main program into this document.**

**Choose one member of the group to make submissions on behalf of the group.**

**The chosen member can make as many submissions (versions) as they like. Only the last submission will be marked.**

**You must upload 1 item :**

**The completed submission document . It must be in Word format with the code neat and visible.**

Group Information:

|  |  |  |
| --- | --- | --- |
|  | Full Name | Student Number |
| Member 1 | Minh Nhat Vo | 101224717 |
| Member 2 (if any) |  |  |

ALL CLASSES FOR THIS ASSIGNMENT MUST BE CREATED FROM SCRATCH. NO BUILT IN DATA STRUCTURES MUST BE USED.

You are required to create a personal dictionary and spellcheck program. There must only be two core classes. A class “wordInfo” and a class “dictionary”.

The information stored on a **wordInfo object** consists of:

A word: a string

A meaning: a string

**The dictionary class** is responsible for holding all the words and supports the following operations:

**public bool add (string word, string meaning)** – adds a new word to the dictionary . **No duplicate words are allowed**. **You must store all the word using a binary search tree**. All words should be stored in lowercase.

**public bool delete (string word**) – deletes the wordInfo object with the matching word.

**public bool exists(string word)** – returns true if the word is in the dictionary and false otherwise.

**public string getMeaning(string word)**- returns the meaning of the word.

**public int getCount()** – returns the number of words in the dictionary. Note that is must be implemented even if not used.

**public string printWordList ()** – returns a list of all the words stored in the dictionary in alphabetical order (only the words, not the meanings).

**public void printDictionary()**- prints the full word and meaning for each wordInfo object in the dictionary (in ascending order).

Your program must **first load all known words from the text file “wordlist.txt” into the dictionary class** you created (which stores all the word as a binary tree). You must find a way to ensure that the initial binary tree is as close to a balanced tree as possible.

NOTE that this list does not contain any definitions and none need to be created. Only new words need definitions. The words loaded from the file should have “Undefined word” as the definition.

Create a main program with the following menu options:

1: Add new word

2: Delete word

3: Get meaning

4: Dictionary list

5: Spell check a text file.

6: Exit

Notes:

**Add new word** – This option should request a word and its meaning. Insert the word into the dictionary if it is new. No duplicates are allowed**. All words are to be stored in lowercase letters**.

**Delete word** – This option should request a word and delete it from the dictionary.

**Get meaning** - This option should request a word and print its meaning if found in the dictionary. If it is not found an appropriate message must be shown.

**Dictionary List**- This option must simply list all the words contained in the dictionary. **Note that the meanings should not be listed.**

**Spell check a text file**- This option should **prompt the user for a text file**. You may assume the text file contains only sentences with words and the only punctuation allowed are periods (.) , commas (,). **You must print all the word that are not in the dictionary**. Please note this must still work regardless of the case of the letters.

**Exit** - This option should exit the program.

Paste the code for you word Info Class here

package assignment2;

public class wordInfo {

public String word;

public String meaning;

wordInfo left, right;

public wordInfo(String word, String meaning) {

this.word = word.toLowerCase();

this.meaning = meaning;

left = right = null;

}

public String getWord() {

return word;

}

}

Paste the Code for your dictionary class **and its associated node** class here

package assignment2;

public class dictionary {

public wordInfo core;

private int numEle;

public dictionary() {

core = null;

numEle = 0;

}

public boolean add(String word, String mean) {

boolean val = true;

wordInfo newWord = null;

//Test duplicate

if (!search2Test(word)) {

newWord = new wordInfo(word, mean);

// Case1 empty

if (core == null) {

core = newWord;

numEle++;

return val;

}

//Case2 leaves

wordInfo explorer, follower;

explorer = follower = core;

while (explorer != null) {

follower = explorer;

if (explorer.word.compareToIgnoreCase(word) > 0) {

explorer = explorer.left;

} else {

explorer = explorer.right;

}

}

if (follower.word.compareToIgnoreCase(word) > 0) {

follower.left = newWord;

numEle++;

val = true;

} else {

follower.right = newWord;

numEle++;

val = true;

}

} else if (search2Test(word)) {

System.out.println("\*\*\* This word EXSISTED. Please! Enter OTHER word \*\*\*");

val = false;

}

return val;

}

public boolean exists(String word) {

boolean val = true;

wordInfo explorer = core;

//Case1 empty

if (explorer == null) {

System.out.println("Binary Tree EMPTY!!!");

return false;

}

//Case2 leaves

while (explorer != null && !explorer.word.equals(word)) {

if (explorer.word.compareToIgnoreCase(word) > 0) {

explorer = explorer.left;

} else {

explorer = explorer.right;

}

}

if (explorer != null) {

System.out.println("SUCCESSFULLY FOUND: " + explorer.word);

val = true;

} else {

System.out.println("\*\*\*NOT FOUND the WORD\*\*\*");

val = false;

}

return val;

}

public boolean search2Test(String word) {

boolean val = true;

wordInfo explorer = core;

//Case1 empty

if (explorer == null) {

// System.out.println("Binary Tree EMPTY!!!");

return false;

}

//Case2 leaves

while (explorer != null && !explorer.word.equals(word)) {

if (explorer.word.compareToIgnoreCase(word) > 0) {

explorer = explorer.left;

} else {

explorer = explorer.right;

}

}

if (explorer != null) {

//System.out.println(" Successfully FOUND: " + explorer.word);

val = true;

} else {

//System.out.println("Not FOUND the WORD!!!");

val = false;

}

return val;

}

public String search(String word) {

String val = null;

wordInfo explorer = core;

//Case1 empty

if (explorer == null) {

// System.out.println("Binary Tree EMPTY!!!");

return val;

}

//Case2 leaves

while (explorer != null && !explorer.word.equals(word)) {

if (explorer.word.compareToIgnoreCase(word) > 0) {

explorer = explorer.left;

} else {

explorer = explorer.right;

}

}

if (explorer != null) {

//System.out.println(" Successfully FOUND: " + explorer.word);

val = explorer.meaning;

} else {

//System.out.println("Not FOUND the WORD!!!");

val = null;

}

return val;

}

public String search2(String word) {

String val = null;

wordInfo explorer = core;

//Case1 empty

if (explorer == null) {

// System.out.println("Binary Tree EMPTY!!!");

return val;

}

//Case2 leaves

while (explorer != null && !explorer.word.equals(word)) {

if (explorer.word.compareToIgnoreCase(word) > 0) {

explorer = explorer.left;

} else {

explorer = explorer.right;

}

}

if (explorer != null) {

//System.out.println(" Successfully FOUND: " + explorer.word);

val = explorer.word;

} else {

//System.out.println("Not FOUND the WORD!!!");

val = null;

}

return val;

}

public boolean delete(String word) {

if (!search2Test(word)) {

System.out.println("");

System.out.println("CANNOT Delete!!!");

return false;

}

core = recursiveDelete(core, word);

System.out.println("\nDeleted SUCCESSFULLY");

return true;

}

private wordInfo recursiveDelete(wordInfo core, String word) {

//Case1 empty

if (core == null) {

return null;

}

//Case2 leaves

if (core.word.compareToIgnoreCase(word) > 0) {

core.left = recursiveDelete(core.left, word);

} else if (core.word.compareToIgnoreCase(word) < 0) {

core.right = recursiveDelete(core.right, word);

} else if (core.word.compareToIgnoreCase(word) == 0) {

if (core.left == null) {

return core.right;

} else if (core.right == null) {

return core.left;

} //else if (core.right != null && core.left != null)

//Case3

wordInfo receiver = core.right;

while (receiver.left != null) {

receiver = receiver.left;

}

core.word = receiver.word;

core.right = recursiveDelete(core.right, receiver.word);

}

return core;

}

public String getMeaning(String word) {

String holder = null;

String searcher = search(word);

// if (core == null) {

// return holder;

// }

if (searcher != null) {

holder = searcher;

} else {

holder = null;

}

return holder;

}

public int getCount() {

return numEle;

}

public void printDictionary() {

System.out.println("\n---- DICTIONARY ----");

producer(core);

System.out.println("");

}

private void producer(wordInfo word) {

if (word == null) {

return;

}

producer(word.left);

System.out.println(word.word + " - " + word.meaning);

producer(word.right);

}

public String printWordList() {

System.out.println("\n---- WORD LIST ----");

return producer2(core);

}

private String producer2(wordInfo word) {

String res = "";

if (word != null) {

res += producer2(word.left);

res += word.word + "\n";

res += producer2(word.right);

}

return res;

}

}

Paste all OTHER code here (includes the main program)

package assignment2;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.\*;

public class Assignment2 {

public static void main(String[] args) throws FileNotFoundException {

String choice = null;

boolean entryMain = true;

Scanner sc = new Scanner(System.in);

dictionary dic = new dictionary();

try {

File fl = new File("");

Scanner sc1 = new Scanner(fl);

while (sc1.hasNextLine()) {

String data = sc1.nextLine();

dic.add(data, "Undefined word");

}

sc1.close();

// dic.printDictionary();

System.out.println("There are " + dic.getCount() + " words LOADED in DICTIONARY.\n");

} catch (FileNotFoundException e) {

System.out.println("An Error occurred.");

e.printStackTrace();

}

while (entryMain) {

System.out.println("----- MENU -----");

System.out.println("1: Add new word");

System.out.println("2: Delete word");

System.out.println("3: Get meaning");

System.out.println("4: Dictionary list");

System.out.println("5: Spell check a text file");

System.out.println("6: Exit");

System.out.print("\nPLEASE CHOOSE ONE: ");

try {

choice = sc.nextLine();

} catch (InputMismatchException e) {

choice = null;

}

if (choice.equals("1")) {

boolean entry = true;

boolean repeat = true;

String enteredWord = null;

String enteredMeaning = null;

while (entry) {

while (true) {

System.out.println("\nEnter a WORD: ");

enteredWord = sc.nextLine();

if (!isAlpha(enteredWord)) {

break;

} else if (isAlpha(enteredWord)) {

System.out.println("\*\*\*Please enter CORRECT TYPE\*\*\*");

enteredWord = null;

continue;

}

}

while (true) {

System.out.println("Enter MEANING of WORD: ");

enteredMeaning = sc.nextLine();

if (!isAlpha(enteredMeaning)) {

break;

} else if (isAlpha(enteredMeaning)) {

System.out.println("\*\*\*Please enter CORRECT TYPE\*\*\*");

enteredMeaning = null;

continue;

}

}

System.out.println("");

boolean testDup = dic.add(enteredWord, enteredMeaning);

if (testDup) {

System.out.println("Add SUCCESSFULLY!!");

while (repeat) {

System.out.println("\nDo you wanna add more? y(YES) - n(NO): ");

String confirmRet = sc.nextLine().toLowerCase();

if ("n".equals(confirmRet)) {

System.out.println("");

dic.printDictionary();

entry = false;

break;

} else if ("y".equals(confirmRet)) {

entry = true;

break;

} else if (confirmRet != "y" || confirmRet != "n") {

System.out.println("\*\*\*Please enter CORRECT TYPE\*\*\*");

repeat = true;

}

}

} else if (!testDup) {

entry = true;

}

}

} else if (choice.equals("2")) {

boolean entry = true;

boolean repeat = true;

int count = dic.getCount();

while (entry) {

if (count == 0) {

System.out.println("\nThese is NOTHING!!!\n");

entry = true;

break;

} else {

System.out.println("\nEnter WORD wanna DELETE: ");

String enteredWord = sc.nextLine();

if (!isAlpha(enteredWord)) {

boolean test = dic.delete(enteredWord);

if (test) {

while (repeat) {

System.out.println("\nDo you wanna DELETE one more WORD? y(YES) - n(NO): ");

String confirmRet = sc.nextLine().toLowerCase();

if ("n".equals(confirmRet)) {

System.out.println("");

entry = false;

break;

} else if ("y".equals(confirmRet)) {

entry = true;

break;

} else if (confirmRet != "y" || confirmRet != "n") {

System.out.println("\*\*\*Please enter CORRECT TYPE\*\*\*");

repeat = true;

}

}

} else if (!test) {

entry = true;

}

} else if (isAlpha(enteredWord)) {

System.out.println("\*\*\*Please enter CORRECT TYPE\*\*\*");

enteredWord = null;

entry = true;

}

}

}

} else if (choice.equals("3")) {

boolean entry = true;

boolean repeat = true;

int count = dic.getCount();

while (entry) {

if (count == 0) {

System.out.println("\nThese is NOTHING!!!\n");

entry = true;

break;

} else {

System.out.println("\nWhich WORD you wanna to TRANSLATE: ");

String enteredWord = sc.nextLine();

if (!isAlpha(enteredWord)) {

String test = dic.getMeaning(enteredWord);

if (test != null) {

System.out.println(enteredWord + " - " + test);

while (repeat) {

System.out.println("\nDo you wanna DELETE one more WORD? y(YES) - n(NO): ");

String confirmRet = sc.nextLine().toLowerCase();

if ("n".equals(confirmRet)) {

System.out.println("");

entry = false;

break;

} else if ("y".equals(confirmRet)) {

entry = true;

break;

} else if (confirmRet != "y" || confirmRet != "n") {

System.out.println("\*\*\*Please enter CORRECT TYPE\*\*\*");

repeat = true;

}

}

} else if (test == null) {

System.out.println("\nCANNOT FIND the WORD!!!");

entry = true;

}

} else if (isAlpha(enteredWord)) {

System.out.println("\*\*\*Please enter CORRECT TYPE\*\*\*");

enteredWord = null;

entry = true;

}

}

}

} else if (choice.equals("4")) {

if (dic.getCount() == 0) {

System.out.println("\nThese is NOTHING!!!\n");

} else {

System.out.println(dic.printWordList());

}

} else if (choice.equals("5")) {

boolean entry = true;

while (entry) {

System.out.println("Enter TEXT FILE:");

String path = sc.nextLine();

if (path.isEmpty()) {

System.out.println("\*\*\*Please enter CORRECT TYPE\*\*\*");

path = null;

entry = true;

}

File fl = new File(path);

Scanner sc1 = new Scanner(fl);

String line = "";

String[] line2;

while (sc1.hasNextLine()) {

line += sc1.nextLine() + " ";

}

line2 = line.split(" ");

for (int x = 0; x < line2.length; x++) {

String newItem = line2[x].replace(",", "");

newItem = newItem.replace(".", "");

if (!dic.search2Test(newItem) && !newItem.equals("")) {

System.out.println(newItem);

}

}

break;

}

} else if (choice.equals("6")) {

System.out.println("(^\_^)Thanks for using(^\_^)\n");

entryMain = false;

} else if (!choice.equals("1") && !choice.equals("2") && !choice.equals("3")

&& !choice.equals("4") && !choice.equals("5") && !choice.equals("6")) {

System.out.println("\n\*\*\*Please enter option between 1-6 \*\*\*\n");

}

}

}

public static boolean isAlpha(String str) {

try {

if (str.isEmpty()) {

return true;

}

double d = Double.parseDouble(str);

} catch (NumberFormatException nfe) {

return false;

}

return true;

}

}