**COMP2080**

**Assignment 1 (10%)**

**Due: 18th February 2023 11:30PM**

**Given: 30th January 2023**

Submission Requirements:

**Your St**u**dent ID number must be commented at the top of the file.**

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

**Only the last submission will be marked.**

**You must upload 2 items :**

* **This completed submission document . It must be in Word format with the code neat and visible.**
* **The compressed (zipped) program**

Student Information:

|  |  |
| --- | --- |
| **Full Name** | **Student Number** |
| Bruno Ramirez | 101380203 |

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 words in all array of size 1500**. 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 in the array as sorted).

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.

Code Submission :

**WordInfo.java**

// Made with love by Bruno 🧡

package entities;  
  
public class WordInfo {  
 private final String word;  
 private String meaning;  
  
 public WordInfo(String \_word) {  
 this.word = \_word.toLowerCase();  
 this.meaning = "Undefined";  
 }  
  
 public WordInfo(String \_word, String \_meaning) {  
 this.word = \_word.toLowerCase();  
 this.meaning = \_meaning;  
 }  
  
 public String getWord() {  
 return word;  
 }  
  
 public String getMeaning() {  
 return meaning;  
 }  
  
 public void setMeaning(String meaning) {  
 this.meaning = meaning;  
 }  
  
 public boolean isGreaterThan(WordInfo \_other) {  
 if (\_other == null) return false;  
 return this.word.compareToIgnoreCase(\_other.getWord()) > 0;  
 }  
  
 public boolean isLessThan(WordInfo \_other) {  
 if (\_other == null) return false;  
 return this.word.compareToIgnoreCase(\_other.getWord()) < 0;  
 }  
  
 public boolean isGreaterThan(String \_other) {  
 return this.word.compareToIgnoreCase(\_other) > 0;  
 }  
  
 public boolean isLessThan(String \_other) {  
 return this.word.compareToIgnoreCase(\_other) < 0;  
 }  
}

**Dictionary.java**

// Made with love by Bruno 🧡

package entities;  
  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.function.Consumer;  
  
public class Dictionary {  
 private final int MAX\_SIZE = 1500;  
 private final WordInfo[] dictionary;  
 private int current;  
  
 public Dictionary() {  
 this.dictionary = new WordInfo[MAX\_SIZE];  
 this.current = 0;  
 }  
  
 public boolean add(String \_word) {  
 return add(\_word, "Undefined");  
 }  
  
 public boolean add(String \_word, String \_meaning) {  
 if (this.current >= this.MAX\_SIZE) return false;  
 if (exists(\_word)) return false;  
  
 dictionary[this.current++] = new WordInfo(\_word, \_meaning);  
 quickSort(0, current - 1);  
 return true;  
 }  
  
 public boolean delete(String \_word) {  
 \_word = \_word.toLowerCase();  
 int item\_index = binSearch(\_word);  
 if (item\_index == -1) return false;  
  
 dictionary[item\_index] = dictionary[--current];  
 quickSort(0, current - 1);  
 return true;  
 }  
  
 public boolean exists(String \_word) {  
 \_word = \_word.toLowerCase();  
 return binSearch(\_word) != -1;  
 }  
  
 public String getMeaning(String \_word) {  
 \_word = \_word.toLowerCase();  
 int item\_index = binSearch(\_word);  
 if (item\_index == -1) return "Word not found!";  
  
 return this.dictionary[item\_index].getMeaning();  
 }  
  
 public int getCount() {  
 return this.current;  
 }  
  
 public String printWordList() {  
 StringBuilder sb = new StringBuilder();  
 stringDictionaryIterator(x -> sb.append(x.getWord()).append("\n"));  
 return sb.toString();  
 }  
  
 public String printDictionary() {  
 StringBuilder sb = new StringBuilder();  
 stringDictionaryIterator(x -> sb.append(x.getWord()).append(" - ").append(x.getMeaning()).append("\n"));  
 return sb.toString();  
 }  
  
 private void stringDictionaryIterator(Consumer<WordInfo> \_action) {  
 for (int i = 0; i < this.current; i++) {  
 \_action.accept(this.dictionary[i]);  
 }  
 }  
  
 private int binSearch(String \_word) {  
 int hi = current - 1, lo = 0, mid;  
 while (lo <= hi) {  
 mid = (hi + lo) / 2;  
 if (dictionary[mid].isGreaterThan(\_word)) hi = mid - 1;  
 else if (dictionary[mid].isLessThan(\_word)) lo = mid + 1;  
 else return mid;  
 }  
 return -1;  
 }  
  
 private void quickSort(int \_lo, int \_hi) {  
 if (\_lo > \_hi) return;  
  
 int mid = partition(\_lo, \_hi);  
 quickSort(mid + 1, \_hi);  
 quickSort(\_lo, mid - 1);  
 }  
  
 private int partition(int \_lo, int \_hi) {  
 int ordered = \_lo - 1;  
  
 for (int i = \_lo; i < \_hi; i++) {  
 if (this.dictionary[i].isLessThan(this.dictionary[\_hi])) {  
 swap(i, ++ordered);  
 }  
 }  
  
 swap(\_hi, ++ordered);  
 return ordered;  
 }  
  
 private void swap(int \_prev, int \_new) {  
 WordInfo tmp = this.dictionary[\_prev];  
 this.dictionary[\_prev] = this.dictionary[\_new];  
 this.dictionary[\_new] = tmp;  
 }  
  
 public static Dictionary loadDictionary() {  
 Dictionary dict = new Dictionary();  
 Path filePath = Path.*of*("/words.txt");  
 String[] content;  
 try {  
 content = Files.*readString*(Path.*of*("./out/production/assignment1/words.txt")).split("\n");  
 } catch (IOException e) {  
 return dict;  
 }  
  
 int lines = Integer.*parseInt*(content[0].trim());  
 for (int i = 1; i < lines + 1; i++) {  
 dict.add(content[i].trim());  
 }  
  
 return dict;  
 }  
}

**Main.java**

// Made with love by Bruno 🧡

import entities.Dictionary;  
  
import java.io.\*;  
import java.nio.charset.StandardCharsets;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Scanner;  
  
public class Main {  
 public static int *menu\_current* = 0;  
 public static String[] *menu\_options* = new String[]{"1. Add new word", "2. Delete word", "3. Get meaning", "4. Dictionary list", "5. Spell check a text file", "6. Exit"};  
 public static Runnable[] *menu\_actions* = new Runnable[]  
 {  
 Main::*addNewWord*,  
 Main::*deleteWord*,  
 Main::*getMeaning*,  
 Main::*dictionaryList*,  
 Main::*spellCheck*,  
 Main::*exit* };  
 public static Scanner *scanner* = new Scanner(System.*in*);  
 public static Dictionary *dict*;  
  
 public static void main(String[] args) {  
 *dict* = Dictionary.*loadDictionary*();  
  
 do {  
 *showMenu*();  
 *menu\_current* = *getIntInput*("Enter an option: ", "Input was not in the correct format!");  
 *menu\_current*--;  
 if (*menu\_current* >= 0 && *menu\_current* < *menu\_options*.length) {  
 *menu\_actions*[*menu\_current*].run();  
 } else {  
 System.*out*.println("Not a valid option!");  
 }  
 } while (*menu\_current* != 5);  
 }  
  
 private static int getIntInput(String \_message, String \_error\_message) {  
 Integer res = null;  
 while (res == null) {  
 try {  
 System.*out*.print(\_message);  
 res = Integer.*parseInt*(*scanner*.nextLine().trim());  
 } catch (NumberFormatException ex) {  
 System.*out*.println(\_error\_message);  
// res = null;  
 }  
 }  
 return res;  
 }  
  
 private static void showMenu() {  
 for (int i = 0; i < *menu\_options*.length; i++) {  
 if (*menu\_current* == i) System.*out*.println("> " + *menu\_options*[i]);  
 else System.*out*.println(*menu\_options*[i]);  
 }  
 }  
  
 private static void addNewWord() {  
 String word = *getInput*("Enter a word: ");  
 if (*dict*.exists(word)) {  
 System.*out*.println("The word already exist in the dictionary!");  
 return;  
 }  
 String meaning = *getInput*("Enter a meaning: ");  
 if (*dict*.add(word, meaning)) {  
 System.*out*.println("Successfully created!");  
 } else {  
 System.*out*.println("Word '" + word + "' could not be added!");  
 }  
 }  
  
 private static void deleteWord() {  
 String word = *getInput*("Enter a word: ");  
 if (*dict*.delete(word)) {  
 System.*out*.println("Successfully deleted!");  
 } else {  
 System.*out*.println("Word '" + word + "' could not be deleted!");  
 }  
 }  
  
 private static void getMeaning() {  
 String word = *getInput*("Enter a word: ");  
 System.*out*.println(*dict*.getMeaning(word));  
 }  
  
 private static void dictionaryList() {  
 System.*out*.println(*dict*.printDictionary());  
 }  
  
 private static void spellCheck() {  
 String path = *getInput*("Enter a file path: ");  
 File file = new File(path);  
 String data = null;  
 try (FileInputStream fis = new FileInputStream(file)) {  
 byte[] info = fis.readAllBytes();  
 data = new String(info, StandardCharsets.*UTF\_8*);  
 } catch (IOException e) {  
 System.*out*.println("File not found!");  
 return;  
 }  
  
 data = data.replace(", ", " ").replace(".", " ").trim();  
 List<String> words = Arrays.*stream*(data.split(" ")).filter(x -> !x.isBlank() && !x.isEmpty()).toList();  
 StringBuilder sb = new StringBuilder();  
 sb.append("Words that I don't have in my dictionary: ").append("\n");  
 for (String word : words)  
 if (!word.isBlank() && !word.isEmpty())  
 if (!*dict*.exists(word.trim()))  
 sb.append(word.trim()).append("\n");  
  
 System.*out*.println(sb);  
 }  
  
 private static String getInput(String \_message) {  
 System.*out*.print(\_message);  
 return *scanner*.nextLine().trim();  
 }  
  
 private static void exit() {  
 System.*exit*(0);  
 }  
}