**Dictionary Code:**

package com.exceptionhandling.dictionary\_app;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

class DictionaryNode {

String word;

String meaning;

DictionaryNode left;

DictionaryNode right;

public DictionaryNode(String w, String m) {

word = w;

meaning = m;

left = null;

right = null;

}

}

class Dictionary\_app {

DictionaryNode root;

public Dictionary\_app() {

root = null;

}

private void destructor\_help(DictionaryNode temp) {

if (temp == null) {

return;

}

destructor\_help(temp.left);

destructor\_help(temp.right);

root = null;

}

public void destructor() {

destructor\_help(root);

}

public void LoadDictionary() {

try {

BufferedReader file = new BufferedReader(new FileReader("C:\\Users\\Administrator\\Downloads\\Dic.txt"));

ClearTree(root);

String line;

while ((line = file.readLine()) != null) {

String[] words = line.split(" ");

Insert(words[0], words[1]);

}

file.close();

System.out.println("Dictionary loaded successfully.");

} catch (IOException e) {

System.out.println("Error: Unable to open file Dictionary.txt");

}

}

public void AddWord(String word, String meaning) {

if (Search(root, word) != null) {

System.out.println("Word already exists in the dictionary.");

} else {

Insert(word, meaning);

UpdateFile();

System.out.println("Word added successfully.");

}

}

public void SearchWord(String word) {

DictionaryNode result = Search(root, word);

if (result != null) {

System.out.println("Meaning: " + result.meaning);

} else {

System.out.println("Word not found in the dictionary.");

}

}

public void DeleteWord(String word) {

DictionaryNode result = Search(root, word);

if (result != null) {

System.out.print("Do you want to delete the word? (Yes/No): ");

String response = System.console().readLine();

if (response.equalsIgnoreCase("Yes")) {

root = Delete(root, word);

UpdateFile();

System.out.println("Word deleted successfully.");

}

} else {

System.out.println("Word not found in the dictionary.");

}

}

public void UpdateWord(String word, String newMeaning) {

DictionaryNode result = Search(root, word);

if (result != null) {

System.out.print("Do you want to update the word? (Yes/No): ");

String response = System.console().readLine();

if (response.equalsIgnoreCase("Yes")) {

result.meaning = newMeaning;

UpdateFile();

System.out.println("Word updated successfully.");

}

} else {

System.out.println("Word not found in the dictionary.");

}

}

public void WordSuggestion(String partialWord) {

System.out.print("Suggested words for \"" + partialWord + "\": ");

SuggestWords(root, partialWord);

System.out.println();

}

private void SuggestWords(DictionaryNode node, String partialWord) {

if (node != null) {

if (node.word.startsWith(partialWord)) {

System.out.println();

System.out.print(node.word);

}

SuggestWords(node.left, partialWord);

SuggestWords(node.right, partialWord);

}

}

private void ClearTree(DictionaryNode node) {

if (node != null) {

ClearTree(node.left);

ClearTree(node.right);

node = null;

}

root = null;

}

private void Insert(String word, String meaning) {

root = Insert(root, word, meaning);

}

private DictionaryNode Insert(DictionaryNode node, String word, String meaning) {

if (node == null) {

return new DictionaryNode(word, meaning);

}

if (word.compareTo(node.word) < 0) {

node.left = Insert(node.left, word, meaning);

} else if (word.compareTo(node.word) > 0) {

node.right = Insert(node.right, word, meaning);

}

return node;

}

private DictionaryNode Search(DictionaryNode node, String word) {

if (node == null || node.word.equals(word)) {

return node;

}

if (word.compareTo(node.word) < 0) {

return Search(node.left, word);

} else {

return Search(node.right, word);

}

}

private DictionaryNode FindMin(DictionaryNode node) {

while (node.left != null) {

node = node.left;

}

return node;

}

private DictionaryNode Delete(DictionaryNode node, String word) {

if (node == null) {

return node;

}

if (word.compareTo(node.word) < 0) {

node.left = Delete(node.left, word);

} else if (word.compareTo(node.word) > 0) {

node.right = Delete(node.right, word);

} else {

if (node.left == null) {

DictionaryNode temp = node.right;

node = null;

return temp;

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

DictionaryNode temp = node.left;

node = null;

return temp;

}

DictionaryNode temp = FindMin(node.right);

node.word = temp.word;

node.meaning = temp.meaning;

node.right = Delete(node.right, temp.word);

}

return node;

}

private void UpdateFile() {

try {

BufferedWriter outFile = new BufferedWriter(new FileWriter("C:\\Users\\Administrator\\Downloads\\Dic.txt"));

WriteToFile(outFile, root);

outFile.close();

} catch (IOException e) {

System.out.println("Error: Unable to open file Dictionary.txt for writing");

}

}

private void WriteToFile(BufferedWriter outFile, DictionaryNode node) throws IOException {

if (node != null) {

WriteToFile(outFile, node.left);

outFile.write(node.word + " " + node.meaning);

outFile.newLine();

WriteToFile(outFile, node.right);

}

}

private void sleep(int milliseconds) {

try {

Thread.sleep(milliseconds);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

private void setConsoleColor(int color) {

// Not supported in Java

}

private void resetConsoleColor() {

// Not supported in Java

}

private void printHeader() {

System.out.println("=============================================");

System.out.println(" DICTIONARY APPLICATION ");

System.out.println("=============================================");

}

private void printOptions() {

System.out.println("\t1. Add Word");

System.out.println("\t2. Search Word");

System.out.println("\t3. Delete Word");

System.out.println("\t4. Update Word");

System.out.println("\t5. Word Suggestion");

System.out.println("\t6. Print Dictionary");

System.out.println("\t7. Exit");

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

System.out.print("\tEnter your choice: ");

}

public void InOrderPrint() {

System.out.println("In-Order Print of Dictionary:");

InOrderPrint(root);

System.out.println();

}

private void InOrderPrint(DictionaryNode node) {

if (node != null) {

InOrderPrint(node.left);

System.out.println(node.word + ": " + node.meaning);

InOrderPrint(node.right);

}

}

public static void main(String[] args) {

Dictionary\_app dictionary = new Dictionary\_app ();

dictionary.LoadDictionary();

String option;

do {

try {

new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();

} catch (IOException | InterruptedException e) {

e.printStackTrace();

}

dictionary.printHeader();

dictionary.printOptions();

option = System.console().readLine();

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

String word, meaning;

System.out.print("Enter word: ");

word = System.console().readLine();

System.out.print("Enter meaning: ");

meaning = System.console().readLine();

System.out.print("Adding word...");

dictionary.sleep(1000);

dictionary.resetConsoleColor();

dictionary.AddWord(word, meaning);

System.out.println("Word added successfully.");

dictionary.sleep(2000);

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

String word;

System.out.print("Enter word to search: ");

word = System.console().readLine();

dictionary.SearchWord(word);

dictionary.sleep(2000);

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

String word;

System.out.print("Enter word to delete: ");

word = System.console().readLine();

dictionary.DeleteWord(word);

dictionary.sleep(2000);

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

String word, meaning;

System.out.print("Enter word to update: ");

word = System.console().readLine();

System.out.print("Enter new meaning: ");

meaning = System.console().readLine();

dictionary.sleep(2000);

dictionary.UpdateWord(word, meaning);

dictionary.sleep(5000);

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

String partialWord;

System.out.print("Enter partial word: ");

partialWord = System.console().readLine();

dictionary.sleep(2000);

dictionary.WordSuggestion(partialWord);

dictionary.sleep(5000);

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

dictionary.InOrderPrint();

dictionary.sleep(10000);

} else if (option.equals("7")) {

System.out.println("Exiting the program.");

dictionary.sleep(5000);

} else {

System.out.println("Invalid option. Please try again.");

dictionary.sleep(5000);

}

} while (!option.equals("7"));

dictionary.destructor();

}

}