**Spell Checker using HashSet and TreeSet**

**Overview**

This Java project implements a spell checker using two classes from the Java Collection Framework: HashSet and TreeSet. The spell checker reads a dictionary from a file, stores it in a HashSet for efficient lookup, and then checks the spelling of words in a user-selected file. Suggestions for correct spellings are provided using a TreeSet.

**Project Structure**

* **SpellChecker.java**: Main class containing the program logic.
* **words.txt**: File containing the English words used as the dictionary.

**Set Methods**

The project utilizes methods from the Set interface, including **size()**, **add(item)**, **contains(item)**, and **isEmpty()**. Iteration through the set is achieved using either an iterator or a for-each loop.

**Reading a Dictionary**

The project reads the dictionary from the **words.txt** file using a Scanner. Words are stored in a HashSet<String>, and all words are converted to lowercase for consistency.

HashSet<String> dictionary = readDictionary();

**Checking Words in a File**

The user is prompted to select an input file using JFileChooser. The program then reads words from the file, checks their spelling against the dictionary, and prints misspelled words along with suggested corrections.

File inputFile = getInputFileNameFromUser();

checkFile(inputFile, dictionary);

**Providing a List of Possible Correct Spellings**

The **corrections** method generates a TreeSet<String> containing variations on a misspelled word that are present in the dictionary. Suggestions include deleting, changing, inserting, swapping, and inserting a space in the misspelled word.

TreeSet<String> suggestions = corrections(word, dictionary);

**Program:**

import java.io.File;

import java.io.FileNotFoundException;

import java.util.HashSet;

import java.util.Scanner;

import java.util.TreeSet;

import javax.swing.JFileChooser;

public class App {

    public static void main(String[] args) {

        HashSet<String> dictionary = readDictionary();

        if (dictionary.size() != 72875) {

            System.out.println("Error reading dictionary.");

            return;

        }

        File inputFile = getInputFileNameFromUser();

        if (inputFile == null) {

            System.out.println("No file selected. Exiting.");

            return;

        }

        checkFile(inputFile, dictionary);

    }

    private static HashSet<String> readDictionary() {

        HashSet<String> dictionary = new HashSet<>();

        try {

            Scanner filein = new Scanner(

                    new File("src/words.txt"));

            while (filein.hasNext()) {

                String word = filein.next().toLowerCase();

                dictionary.add(word);

            }

            filein.close();

        } catch (FileNotFoundException e) {

            e.printStackTrace();

        }

        return dictionary;

    }

    private static void checkFile(File inputFile, HashSet<String> dictionary) {

        try {

            Scanner in = new Scanner(inputFile);

            in.useDelimiter("[^a-zA-Z]+");

            while (in.hasNext()) {

                String word = in.next().toLowerCase();

                if (!dictionary.contains(word)) {

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

                    TreeSet<String> suggestions = corrections(word, dictionary);

                    if (suggestions.isEmpty()) {

                        System.out.println("(no suggestions)");

                    } else {

                        suggestions.forEach(System.out::println);

                    }

                    System.out.println();

                }

            }

            in.close();

        } catch (FileNotFoundException e) {

            e.printStackTrace();

        }

    }

    private static File getInputFileNameFromUser() {

        JFileChooser fileDialog = new JFileChooser();

        fileDialog.setDialogTitle("Select File for Input");

        int option = fileDialog.showOpenDialog(null);

        if (option != JFileChooser.APPROVE\_OPTION) {

            return null;

        } else {

            return fileDialog.getSelectedFile();

        }

    }

    private static TreeSet<String> corrections(String badWord, HashSet<String> dictionary) {

        TreeSet<String> suggestions = new TreeSet<>();

        // Delete any one of the letters from the misspelled word

        for (int i = 0; i < badWord.length(); i++) {

            String deletedChar = badWord.substring(0, i) + badWord.substring(i + 1);

            if (dictionary.contains(deletedChar)) {

                suggestions.add(deletedChar);

            }

        }

        // Change any letter in the misspelled word to any other letter

        for (int i = 0; i < badWord.length(); i++) {

            for (char ch = 'a'; ch <= 'z'; ch++) {

                String changedChar = badWord.substring(0, i) + ch + badWord.substring(i + 1);

                if (dictionary.contains(changedChar)) {

                    suggestions.add(changedChar);

                }

            }

        }

        // Insert any letter at any point in the misspelled word

        for (int i = 0; i <= badWord.length(); i++) {

            for (char ch = 'a'; ch <= 'z'; ch++) {

                String insertedChar = badWord.substring(0, i) + ch + badWord.substring(i);

                if (dictionary.contains(insertedChar)) {

                    suggestions.add(insertedChar);

                }

            }

        }

        // Swap any two neighboring characters in the misspelled word

        for (int i = 0; i < badWord.length() - 1; i++) {

            String swappedChars = badWord.substring(0, i) + badWord.charAt(i + 1) + badWord.charAt(i) +

                    badWord.substring(i + 2);

            if (dictionary.contains(swappedChars)) {

                suggestions.add(swappedChars);

            }

        }

        // Insert a space at any point in the misspelled word

        for (int i = 0; i <= badWord.length(); i++) {

            String withSpace = badWord.substring(0, i) + " " + badWord.substring(i);

            if (dictionary.contains(withSpace)) {

                suggestions.add(withSpace);

            }

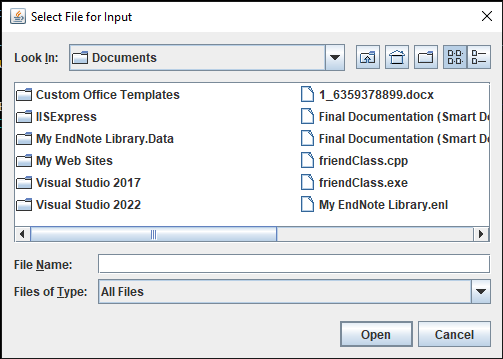
        }

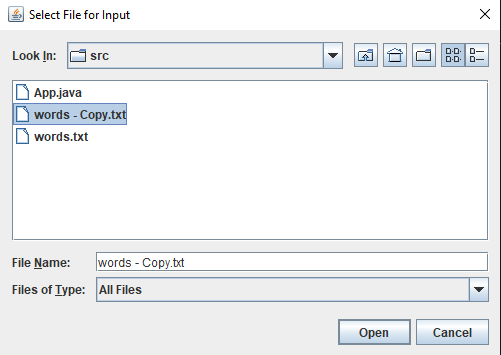
        return suggestions;

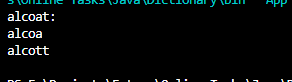
    }

}

**Outputs:**







**References:**

1. Brose, G., Vogel, A., & Duddy, K. (2001). *Java programming with CORBA: advanced techniques for building distributed applications*. John Wiley & Sons.
2. Brose, G., Vogel, A., & Duddy, K. (2001). *Java programming with CORBA: advanced techniques for building distributed applications*. John Wiley & Sons.
3. Zukowski, J. (2006). *Mastering Java 2, J2SE 1.4*. John Wiley & Sons.
4. Mehta, B. (2014). *RESTful Java Patterns and Best Practices*. Packt Publishing.

**The End**