```java
/**
 * EECS233 Written HW3
 * Tung Ho Lin
 */

public class PriorityQueue<T extends Comparable<T>> {

  private T[] items;

  private int numItems;

  private int maxItems;

  public PriorityQueue(int maxSize) {
    items = (T[]) new Comparable[maxSize];
    maxItems = maxSize;
    numItems = 0;
  }

  private boolean isEmpty() {
    return numItems==0;
  }

  public void insert(T item) {
    if(numItems == maxItems) {
      T[] olditems = items;
      items = (T[]) new Comparable[numItems*2 + 1];
      for(int i=0; i<olditems.length; i++)
        items[i] = olditems[i];
    }
    items[numItems] = item;
    numItems++;
    siftUp(numItems-1);
  }

  public T removeMax() {
    T toRemove = items[0];
    items[0] = items[numItems-1];
    numItems--;
    siftDown(0);
    return toRemove;
  }

  public void siftUp(int i) {
    T toSift = items[i];
    int child = i;
    int parent = (i-1)/2;
    while(parent > 0 && items[child].compareTo(items[parent]) > 0) { //if the child is larger than the parent
      items[child] = items[parent];
      items[parent] = toSift;
      child = parent;
      parent = (child-1)/2;
    }
    items[parent] = toSift;
  }

  public void siftDown(int i) {
    T toSift = items[i];
    int parent = i;
    int child = parent*2 + 1;  //child to compare with; start with left child
    while(child < numItems) {
      if(child + 1 < numItems && items[child].compareTo(items[child + 1]) < 0) //if the right child exists and is larger than the left child
        child += 1;
      if(toSift.compareTo(items[child]) >= 0) //if the parent is larger or equal to the child
        break;  //siftDown is complete
      items[parent] = items[child];
      items[child] = toSift;
```

```
68        parent = child;
69        child = parent*2 + 1;
70      }
71        items[parent] = toSift;
72    }
73  }
74
75
76
77
```

```java
/**
 * EECS233 WrittenHW3
 * Tung Ho Lin
 * Spelling Checker: Dictionary
 */
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Hashtable;

//using a built in hashtable to implement the words database
/**
 * The Dictionary text file has to be written in: one word one line format
 * for the build function to work
 */
public class Dictionary {

  private Hashtable<String, String> dict;

  public Dictionary() {
    dict = new Hashtable<String, String>();
  }

  public void addWord(String input) {
    String word = input.toLowerCase();
    word = word.replaceAll("\\s*\\p{Punct}+\\s*$", "");  //remove all whitespaces and punctuation at the end of the word
    if(findWord(word))
      return;
    else
      dict.put(word, word);
  }

  public boolean findWord(String input) {
    String word = input.toLowerCase();
    return dict.contains(word);
  }

 //limitation of this method, see top
  public void build(String inputfile) {
    try {
    BufferedReader reader = new BufferedReader(new FileReader(inputfile));
    String word;
    while((word = reader.readLine()) != null)
      addWord(word);
    reader.close();
    }
    catch (IOException e) {
      System.err.println("File not found!");
    }
  }
}
```

```java
/**
 * EECS233 WrittenHW3
 * Tung Ho Lin
 * Spelling Checker: SpellChecker
 */
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.ArrayList;

//D:\\School Documents\\EECS233\\HW5\\input2.txt
public class SpellChecker {

  private String inputfile;

  private Dictionary dict;

  public SpellChecker(String main, String personal, String input) {
    this.inputfile = input;
    dict = new Dictionary();
    dict.build(main);
    dict.build(personal);
  }

  //main method
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Please enter the path of the main dictionary file.");
    String main = sc.nextLine();
    System.out.println("Please enter the path of the personal dictionary file.");
    String personal = sc.nextLine();
    System.out.println("Please enter the path of the text file to be spell checked.");
    String input = sc.nextLine();
    SpellChecker checker = new SpellChecker(main, personal, input);
    checker.spellCheck();
  }

  //method to spell check and print out the errors
  public void spellCheck() {
    try {
    Scanner linesc = new Scanner(new File(inputfile));
    int linenumber = 0;
    String line;
    String word;
    while(linesc.hasNextLine()) {
      line = linesc.nextLine();   //scan a whole line of text
      linenumber++;  //increment line number
      Scanner wordsc = new Scanner(line);
      while(wordsc.hasNext()) {
        word = wordsc.next().toLowerCase();  //scan a word from the line of text
        word = word.replaceAll("\\s*\\p{Punct}+\\s*$", "");
        if(dict.findWord(word) == false) {  //if the word is not found in the built dictionary
          System.out.println(word + " : in Line " + linenumber + " is not spelled correctly.");
          System.out.println(appendSuggestions(word) + "\n");
        }
      }
      wordsc.close();
    }
    linesc.close();
    System.out.println("Spellcheck completed!");
    }
    catch (FileNotFoundException e) {
      System.err.println("File not found!");
    }
  }

  //collect all the suggestions from the 3 methods and delete identical suggestions
```

```
68    public String appendSuggestions(String input) {
69      ArrayList<String> add = addChar(input);
70      ArrayList<String> remove = removeChar(input);
71      ArrayList<String> swap = swapChar(input);
72      ArrayList<String> suggestions = new ArrayList<String>();
73      String output = "Suggestions: ";
74      for(int i=0; i<add.size(); i++)  //add all suggestions from first method
75        suggestions.add(add.get(i));
76      for(int i=0; i<remove.size(); i++) {  //add non-recurrent suggestions
77        if(suggestions.contains(remove.get(i)) == false)
78          suggestions.add(remove.get(i));
79      }
80      for(int i=0; i<swap.size(); i++) {  //add non-recurrent suggestions
81        if(suggestions.contains(swap.get(i)) == false)
82          suggestions.add(swap.get(i));
83      }
84      for(int i=0; i<suggestions.size()-1; i++) {  //append a String of suggestions
85        output += suggestions.get(i) + ", ";  //do not print out ", " on the last
    word
86      }
87      try{
88      output += suggestions.get(suggestions.size()-1);  //deal with the last word in
    the list
89      }
90      catch(ArrayIndexOutOfBoundsException e){
91        System.err.println("Suggestions cannot be generated by the built-in methods,
    Sorry!");  //if the misspelled word cannot be fixed by the 3 methods
92        output += "N/A";
93      }
94      return output;
95    }
96
97    //create suggestions by adding a character to anywhere in the word each time
98    public ArrayList<String> addChar(String input) {
99      ArrayList<String> suggestions = new ArrayList<String>();
100     char[] alphabets = new char[26];
101     for(int i=0; i<alphabets.length; i++)  //create an array of all 26 alphabets
102       alphabets[i] = (char)('a'+ i);
103     for(char c : alphabets) {  //for each alphabet to be inserted in the word
104       for(int i=0; i<=input.length(); i++) {  //for each space in the word to be
    inserted
105         String suggest = input.substring(0, i) + c + input.substring(i, input.
    length()); //insert the character between each adjacent characters
106         if(dict.findWord(suggest))  //check if it is a correct word
107           suggestions.add(suggest);
108       }
109     }
110     return suggestions;
111   }
112
113   //create suggestions by removing a character from anywhere in the word each time
114   public ArrayList<String> removeChar(String input) {
115     ArrayList<String> suggestions = new ArrayList<String>();
116     for(int i=0; i<input.length(); i++) {
117       StringBuilder builder = new StringBuilder(input);
118       builder.deleteCharAt(i);  //delete one character each time
119       String suggest = builder.toString();
120       if(dict.findWord(suggest))  //compare to the dictionary if it is a correct
    word
121         suggestions.add(suggest);
122     }
123     return suggestions;
124   }
125
126   //create suggestions by swapping 2 characters in a word
127   public ArrayList<String> swapChar(String input) {
128     ArrayList<String> suggestions = new ArrayList<String>();
129     for(int i=0; i<input.length()-1; i++) {
130       char[] decon = input.toCharArray();
131       char swap = decon[i];
```

```
132       decon[i] = decon[i+1];  //swapping the characters
133       decon[i+1] = swap;
134       String suggest = new String(decon);  //back to String
135       if(dict.findWord(suggest))  //check if it is a correct word
136         suggestions.add(suggest);
137     }
138     return suggestions;
139   }
140 }
141
142
143
144
145
```