A spelling checker reads the words in a document and outputs a list of misspelled words. For each misspelled word, the program outputs the word followed by a list of the line numbers where the word is found in the document.

The program decides which words are misspelled by using a dictionary of correct words. The program searches the dictionary for each word in the document. If a word from the document cannot be found in the dictionary, the word is added to the list of misspelled words.

## **Project Files**

The following files are required to complete the project. You should download these files from Google Drive. You are not allowed to modify these files in any way except for implementing methods that were intentionally left empty. Specifically, interface definitions may not be modified, and all code must remain in the spellcheck folder.

SpellCheckerInterface.java

|  |
| --- |
| import java.io.\*;  public interface SpellCheckInterface {  /\*\*  \* Loads the dictionary contained in the specified file  \*  \* @param filename The name of the dictionary file to be loaded  \* @return true If the file was successfully loaded, false  \* if the file could not be loaded or was invalid  \* @throws IOException if filename does not exist  \*/  public boolean loadDictionary(String fileName) throws IOException;    /\*\*  \* Check the document for misspelled words  \*  \* @return true if the file was successfully spell checked, false  \* if the file had misspelled words  \* @throws IllegalArgumentException if filename is null  \*/  public boolean checkDocument(String fileName) throws IOException;  } |

Word.java

|  |
| --- |
| import java.util.\*; public class Word implements Comparable<Word> {  private String word; // the misspelled word  private ArrayList<Integer> lines; // line numbers    public Word() {  lines = new ArrayList<>();  }    public Word(String w, int lineNumber) {  word = w;  lines = new ArrayList<>();  lines.add(lineNumber);  }    public String getWord() {  return word;  }    public ArrayList<Integer> getLines() {  return lines;  }    public void addLine(int lineNumber) {  lines.add(lineNumber);  }    public String toString() {  return word + " : "+lines.toString();  }    /\*\*  \*@return true if two words are equal, false otherwise  \*/  @Override  public boolean equals(Object w) {  // Left as exercise  }    /\*\*  \*@compare two words and return the corresponding integer value (<0, 0, >0)  \*/  @Override  public int compareTo(Word w) {  return this.word.compareTo(w.word);  } } |

SpellChecker.java

|  |
| --- |
| import java.util.\*;  import java.io.\*;  public class SpellChecker implements SpellCheckInterface {    LinkedList<String> dict;  LinkedList<Word> misspelled;    public SpellChecker() {  dict = new LinkedList<>();  misspelled = new LinkedList<Word>();  }    /\*\*  \* Loads the dictionary contained in the specified file  \*  \* @param filename The name of the dictionary file to be loaded  \* @return true If the file was successfully loaded, false  \* if the file could not be loaded or was invalid  \* @throws IOException if filename does not exist  \*/  @Override  public boolean loadDictionary(String fileName) throws IOException  {  // Left as exercise    return true;  }    /\*\*  \* Check the document for misspelled words  \*  \* @return A list of misspelled words and  \* the line numbers where they occur.  \* @throws @throws IOException if filename does not exist  \*/  @Override  public boolean checkDocument(String fileName) throws IOException {  misspelled = new LinkedList<>(); // Initialize for each file  int lineNumber = 0;  try {  File infile = new File(fileName);  System.out.println("\n\nFile Name:"+ fileName);  try ( Scanner in = new Scanner(infile); )    {  in.useDelimiter("[^A-Za-z]+"); // split input by words  while (in.hasNext()){  String myWord = in.next().toLowerCase();    lineNumber++;  // Left as exercise  // if dictionary does not contain myWord:  // myWord is misspelled  // if misspelled list contains myWord, update the lineNumber  // otherwise add a new Word to misspelled list        }    }  } catch (IOException ex) {  System.out.println(ex);  return false;    // sort misspelled list    if(list.length==0)  return true;  else {  System.out.println("Misspelled words are");  for(Object w : list)  System.out.println(w);  return false;  }        }  } |

Driver.java

|  |
| --- |
| import java.io.\*; public class Driver {   public static void main(String[] args) throws IOException{  // TODO Auto-generated method stub  SpellCheckInterface check = new SpellChecker();  check.loadDictionary("dictionary.txt");  if(check.checkDocument("test.txt"))  System.out.println("No misspelled words found");    if(check.checkDocument("short.txt"))  System.out.println("No misspelled words found");    } |

Dictionary File Format (dictionary.txt)

The spelling checker reads one or more dictionaries of correct words. Each dictionary is a text file that contains a list of words, one word per line as shown in this example.

|  |
| --- |
| abalone abalones abandon abandoned abandoning abandonment |

## **Document Files**

short.txt

test.txt

The documents to be spell checked are just regular text files (i.e., sequences of characters). The spelling checker reads in the file to be spell checked and extracts all of the words from the file. Each word in the file is checked against the words in the currently loaded dictionaries in order to detect misspelled words. If a word is not found in the currently loaded dictionaries, it is considered to be misspelled. A document file might contain English prose, or it could just be a list of words. The program does the same thing in either case:

1) extracts the words in the file and

2) checks to see if they're in the dictionary.

How does the program find the words in the document file? Words are defined as sequences of letters that are separated by characters that are not letters. The characters that you need to check for are numbers, white space (spaces, tabs, new lines, etc.), periods, commas, colons, semi-colons, parentheses, apostrophes, hyphens, equals, pluses, percents, slashes, quotes, asterisks, square braces, curly braces, less than, greater than, question marks, exclamation marks, ampersands, vertical bars and underscores.

The output of the spelling checker is a sorted list of misspelled words, one word per line. Use a \n character to indicate a new line. Each word is lowercase and followed by a colon and a comma-separated list of line numbers where the word is found in the document. Make sure to remember the space after the comma.

Sample output with “test.txt” file

|  |
| --- |
| as : [10, 40]  batlefield : [9]  be : [20, 22, 49]  by : [27, 54]  dedcated : [49]  foeur : [3]  foor : [20, 22, 33]  natiion : [4, 8]  nicolay : [31]  peple : [54, 54] |

Sample output with “short.txt” file

|  |
| --- |
| No misspelled words found |

|  |
| --- |
| **Tips**  1. Remember that Java is case-sensitive. If your dictionary is all in lower-case, and your file has some words in upper-case, the upper-case words won't be in the dictionary! 2. When using the Scanner, you want to tokenize by non-letter characters. You tell the Scanner to delimit by non-letter characters by using the following pattern: "**[^A-Za-z]+**". See the API for how to use patterns with the Scanner class. 3. Remember, it should be possible to use the same instantiated SpellingChecker object to check several documents, make sure that misspelled words found in one run of checkDocument() do not show up in the list of misspelled words for subsequent runs unless they really should be there (ie. make sure that you clear your list of misspelled words every time you scan a new document)   **NOTE:**   1. **Do not add any new classes** 2. **Do not alter the data files.** 3. **Do not change the algorithm.** 4. **Your program should work with the given specification and algorithm to get credit.** |