```java
import java.util.Comparator;

import components.map.Map;
import components.map.Map1L;
import components.queue.Queue;
import components.queue.Queue1L;
import components.set.Set;
import components.set.Set1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Program to ask the user for an input file and then on the basis of that input
 * file, creates an output HTML file which contains a table with the words in
 * the input file and their number of occurrences in the input file.
 *
 * @author Ansh Pachauri
 */
public final class Project1 {

    /**
     * No argument constructor--private to prevent instantiation.
     */
    private Project1() {
        // no code needed here
    }

    /**
     * Compare {@code String}s in Alphabetical order.
     */
    private static class StringLT implements Comparator<String> {
        @Override
        public int compare(String str1, String str2) {
            return str1.toLowerCase().compareTo(str2.toLowerCase());
        }
    }

    /**
     * Generates the set of characters in the given {@code String} into the
     * given {@code Set}.
     *
     * @param str
     *            the given {@code String}
     * @param charSet
     *            the {@code Set} to be replaced
     * @replaces charSet
     * @ensures charSet = entries(str)
     */
    private static void generateElements(String str, Set<Character> charSet) {
        for (int i = 0; i < str.length(); i++) {
            char strChar = str.charAt(i);
            if (!charSet.contains(str.charAt(i))) {
                charSet.add(strChar);
            }
        }
    }

    /**
     * Returns the first "word" (maximal length string of characters not in
     * {@code separators}) or "separator string" (maximal length string of
     * characters in {@code separators}) in the given {@code text} starting at
     * the given {@code position}.
     *
     * @param text
     *            the {@code String} from which to get the word
```

```
or separator
     *            string
     * @param position
     *            the starting index
     * @param separators
     *            the {@code Set} of separator characters
     * @return the first word or separator string found in {@code text} starting
     *         * at index {@code position}
     * @requires 0 <= position < |text|
     * @ensures <pre>
     * nextWordOrSeparator =
     * text[position, position + |nextWordOrSeparator|) and
     * if entries(text[position, position + 1)) intersection separators = {} * then
     * entries(nextWordOrSeparator) intersection separators = {} and
     * (position + |nextWordOrSeparator| = |text| or
     * entries(text[position, position + |nextWordOrSeparator| + 1))
     * intersection separators /= {})
     * else
     * entries(nextWordOrSeparator) is subset of separators and
     * (position + |nextWordOrSeparator| = |text| or
     * entries(text[position, position + |nextWordOrSeparator| + 1))
     * is not subset of separators)
     * </pre>
     */
    private static String nextWordOrSeparator(String text, int position,
            Set<Character> separators) {
        String str = "";
        if (!separators.contains(text.charAt(position))) {
            for (int i = 0; i < text.substring(position).length(); i++) {
                char strChar = text.charAt(i + position);
                if (!separators.contains(text.charAt(i + position))) {
                    str = str + strChar;
                } else {
                    i = text.substring(position).length();
                }
            }
        } else {
            for (int j = 0; j <
text.substring(position).length(); j++) {
                char strChar = text.charAt(j + position);
                if (separators.contains(text.charAt(j +
position))) {
                    str = str + strChar;
                } else {
                    j = text.substring(position).length();
                }
            }
        }
        return str;
    }

    /**
     * Outputs the HTML page with the table of words and their
corresponding
     * counts. Expected elements from this method:
     *
     * <html> <head> <title> title of the page </title> </head>
<body>
     * <h2>title</h2>
     * <hr>
     * <table>
     * <tr>
     * <th>Words</th>
     * <th>Counts</th>
     * </tr>
     * </table>
     * </body></html>
     *
     * @param termMap
     *            the map of terms and their occurrences
     * @param out
     *            the output stream
     * @param title
     *            the string of the file name
     * @param termQueue
     *            the queue of unique words
     * @updates out.content
     * @requires out.is_open
     * @ensures out.content = #out.content * [the HTML tags]
     */
    private static void outputHTML(Map<String, Integer>
termMap,
            SimpleWriter out, String title, Queue<String>
termQueue) {

        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>" + title + "</h2>");
        out.println("<hr>");
        // creating the table
        out.println("<table border = 1>");
        out.println("<tr>");
        out.println("<th>" + "Words" + "</th>");
        out.println("<th>" + "Counts" + "</th>");
        out.println("</tr>");
        // adding each word and value to the table
        int queueLength = termQueue.length();
        for (int i = 0; i < queueLength; i++) {
            String word = termQueue.dequeue();
            out.println("<tr>");
            // word
            out.println("<td>" + word + "</td>");
            // value
            out.println("<td>" + termMap.value(word) + "</
td>");
            out.println("</tr>");
```

```java
        }

        out.println("</table>");
        out.println("</body>");
        out.println("</html>");

    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments; unused here
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        out.print("Name of the input file: ");
        String inputFile = in.nextLine();
        SimpleReader inFile = new SimpleReader1L(inputFile);

        out.print("Name of the output file: ");
        String outputFile = in.nextLine();
        SimpleWriter outFile = new SimpleWriter1L(outputFile);

        Map<String, Integer> termMap = new Map1L<>();
        //characters for separating
        String separators = " \t~`!@#$%^&*()-_+={}[]|;:'<>,.?/";
        Set<Character> separatorSet = new Set1L<>();
        generateElements(separators, separatorSet);
        //adding the words and their corresponding counts in the map
        while (!inFile.atEOS()) {
            String line = inFile.nextLine();
            //starting position for each line
            int lineStart = 0;
            while (lineStart < line.length()) {
                //find character/word
                String charOrWord = nextWordOrSeparator(line, lineStart,
                        separatorSet);
                //check if the string is a word
                if (!separatorSet.contains(charOrWord.charAt(0))) {
                    //if it is a word then check if it is already in the map
                    if (!termMap.hasKey(charOrWord)) {
                        //if no, then add to the map
                        termMap.add(charOrWord, 1);
                    } else {
                        //if yes, then update the count of that word in the map
                        int val = termMap.value(charOrWord);
                        val++;
                        termMap.replaceValue(charOrWord, val);
                    }

                }
                //moving the next potential word or character in the line
                lineStart += charOrWord.length();
            }
        }
        //making a queue with all the words from map
        Queue<String> termQueue = new Queue1L<>();
        Map<String, Integer> tempMap = new Map1L<>();
        tempMap.transferFrom(termMap);
        while (tempMap.size() > 0) {
            Map.Pair<String, Integer> tempPair = tempMap.removeAny();
            String key = tempPair.key();
            int value = tempPair.value();
            termQueue.enqueue(key);
            termMap.add(key, value);
        }
        //arranging the words in the queue alphabetically
        Comparator<String> order = new StringLT();
        termQueue.sort(order);
        //title of the table
        String title = "Words Counted in " + inputFile;
        //creating the HTML document with the table
        outputHTML(termMap, outFile, title, termQueue);
        out.print("Done!");

        outFile.close();
        inFile.close();
        out.close();
        in.close();
    }
}

```