```java
import java.util.Comparator;

/**
 * Program to take in an input file of words and outputs an HTML file of the
 * words and the amount of times each word appeared in the input file.
 *
 * @author Shyam Sai Bethina
 */
public final class WordCounter {

    /**
     * Default constructor--private to prevent instantiation.
     */
    private WordCounter() {
    }

    /**
     *
     * A comparator that orders the queue of words.
     *
     */

    private static class StringLT implements Comparator<String> {
        @Override
        /*
         * Compares two strings and returns them in alphabetical sequence, which
         * is used to order the words queue later on
         */
        public int compare(String one, String two) {
            return one.compareTo(two);
        }
    }

    /**
     * Gets the lines from the input file stream and returns a queue of the
     * lines.
     *
     * @param in
     *            The input file stream
     * @return A queue of lines from the input file
     * @ensures Returned Queue is filled with lines from the input file stream
     *
     */
    public static Queue<String> getLines(SimpleReader in) {
        //Creates an empty queue to add in lines
        Queue<String> lines = new Queue1L<>();

        /*
         * While the input file stream is not at the end, it gets the next line
         * within the input, and if the is not empty, it gets added to the Queue
         */
        while (!in.atEOS()) {
            String line = in.nextLine();
            if (!(line.isEmpty())) {
                lines.enqueue(line);
            }

        }

        return lines;
    }

    /**
     * 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}. Adds only words to the Queue.
     *
     * @param text
     *            the {@code String} from which to get the word or separator
     *            string
     * @param words
     *            Queue to be replaced with only words from the text
     * @requires 0 <= position < |text|
     * @ensures <pre>
     * The returned Queue will have words, but not words with
```

```java
separators
     * </pre>
     */
    public static void nextWordOrSeparator(Queue<String> words, String text) {

        /*
         * Define all possible separator characters
         */
        final String separatorStr = " \t,!.?(){}[];:'-";
        Set<Character> separatorSet = new Set1L<>();
        /*
         * Goes through each character of the string and adds the non-duplicates
         * to the set
         */
        for (int i = 0; i < separatorStr.length(); i++) {
            char charTemp = separatorStr.charAt(i);
            if (!separatorSet.contains(charTemp)) {
                separatorSet.add(charTemp);
            }
        }

        //Indexes to get the substring of words or separators
        int firstIndex = 0;
        int secondIndex = 0;
        while (firstIndex < text.length()) {
            String subString;
            /*
             * This boolean will be used to determine whether the string we are
             * indexing is a word or separators
             */
            boolean word = false;

            /*
             * If the character at firstIndex is a separator, then subString
             * will equal the string with only separators until the character is
             * a letter. If the character at firstIndex is a letter, then
             * subString will equals the string with only letter until character
             * is a separator
             */
            if (separatorSet.contains(text.charAt(firstIndex))) {
                while (secondIndex < text.length()
                        && separatorSet.contains(text.charAt(secondIndex))) {
                    secondIndex++;
                }
            } else {
                while (secondIndex < text.length()
                        && !separatorSet.contains(text.charAt(secondIndex))) {
                    secondIndex++;
                }
                /*
                 * Since the characters in this block don't belong to the
                 * separate, they belong to words, which will make the word
                 * boolean true
                 */
                word = true;
            }

            /*
             * If the resulting subString is a word, this block enqueues the
             * word to Queue words, and firstIndex will equal to secondIndex in
             * order reset the count
             */
            if (word) {
                subString = text.substring(firstIndex, secondIndex);
                words.enqueue(subString.toLowerCase());
            }

            firstIndex = secondIndex;
        }

    }

    /**
     *
     * @param counts
     *             The map to add the words as the keys and the number of times
     *             the words appear as the value
     * @param lines
     *             The queue of lines from the input file
     * @ensures <pre>
     * The map will have all the words from the input file and the occurrences
     * of each word in the file as the value
     * </pre>
     */
    public static void addToMap(Map<String, Integer> counts,
            Queue<String> lines) {
        /*
         * Creates a queue and adds in all the words from all the lines from the
         * input file
         */
        Queue<String> words = new Queue1L<String>();
        for (String line : lines) {
            nextWordOrSeparator(words, line);
        }

        /*
         * Goes through each words to check if it is in the map already. If it
         * is, then it updates the count value of the word. If it is not in the
         * map, then it adds it into the map with a value of 1
         */
        for (String word : words) {
            if (!counts.hasKey(word)) {
                counts.add(word, 1);
            } else {
                int temp = counts.value(word);
                temp++;
                counts.replaceValue(word, temp);
            }
        }

    }

    /**
```

```java
     * Outputs the header for the index HTML file.
     *
     * @param out
     *            The output file stream
     * @param fileName
     *            The name of the file the user desired
     * @requires out.is_open
     * @ensures output file has the header for the index HTML file
     */
    public static void outputHeader(SimpleWriter out, String fileName) {
        /*
         * Outputs the beginning code of the index HTML file to the output file
         * stream
         */
        out.println("<html>");
        out.println("   <head>");
        out.println("       <title>Words Counted in " + fileName + "</title>");
        out.println("   </head>");
        out.println("   <body>");
        out.println("       <h2>Words Counted in " + fileName + "</h2>");
        out.println("       <hr/>");
        out.println("       <table border='1'>");
        out.println("           <tbody>");
        out.println("               <tr><th>Words</th><th>Counts</th></tr>");

    }

    /**
     * Outputs the footer for the index HTML file.
     *
     * @param out
     *            The output file stream
     * @requires out.is_open
     * @ensures output file has the closing braces for the index HTML file
     */
    public static void outputFooter(SimpleWriter out) {
        /*
         * Outputs the closing code of the index HTML file to the
```

```
output file
         * stream
         */
        out.println("           </tbody>");
        out.println("       </table>");
        out.println("   </body>");
        out.print("</html>");
    }

    /**
     * Outputs the words and corresponding counts to the table in
the index HTML
     * file.
     *
     * @param counts
     *            The map of the words and their corresponding
occurrences in
     *            the input file.
     * @param out
     *            The output file stream
     * @requires out.is_open
     * @ensures output file has the code to output the table of
words and counts
     *          in the HTML file.
     */
    public static void outputCounts(Map<String, Integer> counts,
            SimpleWriter out) {
        /*
         * This queue has all the words within in the map
         */
        Queue<String> words = new Queue1L<String>();
        for (Map.Pair<String, Integer> pair : counts) {
            words.enqueue(pair.key());
        }

        /*
         * The queue gets sorted in alphabetical order/
         */
        Comparator<String> order = new StringLT();
        words.sort(order);

        /*
         * We go through the queue, and output the word and the
corresponding
```

```java
         * count of the word. Since the queue is in order, the
list will appear
         * in alphabetical order on the HTML file
         */
        for (String word : words) {
            int count = counts.value(word);
            out.println("               <tr><th>" + word + "</
th><th>" + count
                    + "</th></tr>");
        }
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments; unused here
     */
    public static void main(String[] args) {
        /*
         * Creates input file stream for user input and output
file stream to
         * ask questions
         */
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        /*
         * Gets the input file name from user, and inputName
becomes the answer
         */
        out.println("Enter location and name of input file: ");
        String inputName = in.nextLine();

        /*
         * This input file stream reads the input file using the
name the user
         * inputed
         */
        SimpleReader inputFile = new SimpleReader1L(inputName);

        /*
         * Asks for the name of the output file name, and fileName
becomes the
```

```java
         * answer
         */
        out.println("Enter name of output file: ");
        String fileName = in.nextLine();

        /*
         * This output file stream creates a new file with the name the user
         * wanted
         */
        SimpleWriter outFile = new SimpleWriter1L(fileName);

        /*
         * Queue lines is filled up with the lines from the input file, and
         * inputFile stream is closed because it is not needed anymore
         */
        Queue<String> lines = getLines(inputFile);

        /*
         * The counts map has all the words and counts of each word
         */
        Map<String, Integer> counts = new Map1L<String, Integer>();
        addToMap(counts, lines);

        /*
         * The next three lines outputs the header, the list, and the footer of
         * the HTML file to the desire file
         */
        outputHeader(outFile, inputName);
        outputCounts(counts, outFile);
        outputFooter(outFile);

        /*
         * Closes all the input and output streams
         */
        inputFile.close();
        in.close();
        out.close();
        outFile.close();
```

```
    }

}

```