```java
import java.util.Comparator;

/**
 * Reads input file and outputs list words w/o duplicates and their frequency.
 *
 * @author Gabe Azzarita
 *
 */
public final class WordCounter {

    /**
     * No argument constructor--private to prevent instantiation.
     */
    private WordCounter() {
    }

    /**
     * Compare {@code String}s in lexicographic order.
     */
    private static class StringLT implements Comparator<String> {
        @Override
        /**
         * @param o1
         *            first string
         * @param o2
         *            second string
         * @ensures positive int, zero, or negative int if o1 is greater than,
         *          equal to, or less than o2
         * @return integer signaling which string is bigger
         */
        public int compare(String o1, String o2) {
            return o1.compareTo(o2);
        }
    }

    /**
     * Outputs the "opening" tags in the generated HTML file.
     *
     * @param out
     *           the output stream
     * @param file
     *           input file
     * @ensures out.content = #out.content * [the HTML "opening" tags]
     *
     */
    public static void outputHeader(SimpleWriter out, String file) {
        out.println("<html>");
        out.println("  <head>");
        out.println("    <title> Word Counter </title>");
        out.println("  </head>");
        out.println("    <body>");
        out.println("      <h2> Words Counted in " + file + "</h2>");
        out.println("<hr>");
        out.println("        <table border = 1>");
        out.println("          <tr>");
        out.println("            <td><b>Word</b></td>");
        out.println("            <td><b>Count</b></td>");
        out.println("          </tr>");

    }

    /**
     * Fills the separator set with whatever desired separators.
     *
     * @param separators
     *            set of separators to fill
     * @ensures separators set is filled with new separators
     *
     * @updates separators
     */
    public static void fillSeparators(Set<Character> separators) {
        separators.add(' ');
        separators.add(',');
        separators.add('.');
        separators.add('!');
        separators.add('?');
        separators.add('"');
        separators.add(';');
        separators.add(':');
        separators.add('-');
        separators.add('\t');
    }

    /**
     * Reads input file, and copies words in wordQueue.
     *
     * @param inRead
     *            reader for the input file
     * @param wordQueue
     *            queue that will be filled with words from file
     * @param separators
     *            set used to make sure we only add words to queue
     * @ensures wordQueue contains all the words from input file
     *
     * @updates wordQueue
     *
     */
    public static void fillWordQueue(SimpleReader inRead,
            Queue<String> wordQueue, Set<Character> separators) {

        String tempString = "";
        String tempWord = "";
        // Keep reading lines until we reach the end
        while (!inRead.atEOS()) {
            tempString = inRead.nextLine();
            // Go through string until we have find a separator
            for (int i = 0; i < tempString.length(); i++) {
                // If character is a letter, we add it to tempWord
                if (!separators.contains(tempString.charAt(i))) {
                    tempWord += tempString.charAt(i);
                    // for last character in line, add word to queue
                    // and clear tempWord
                    if (i == tempString.length() - 1) {
                        wordQueue.enqueue(tempWord.toLowerCase());
                        tempWord = "";
                    }
                } else {
                    // else if character is a separator, we add tempWord
                    // to queue if it's greater than 0 and clear tempWord
                    if (tempWord.length() > 0) {
                        wordQueue.enqueue(tempWord.toLowerCase());
                    }
                    tempWord = "";
                }
            }
        }
    }

    /**
     * Fills map, and keeps track of how many times a word shows up.
     *
     * @param wordQueue
     *            queue that Map is filled from
     * @param noDupeQueue
     *            queue that will fill with only non duplicate words
     * @param pairMap
     *            the map containing words and its count
     * @ensures pairMap is filled and words counts are updated
     *
     * @updates pairMap
     * @clears wordQueue
     *
     */
    public static void fillMap(Queue<String> wordQueue,
            Queue<String> noDupeQueue, Map<String, Integer> pairMap) {
        String tempWord = "";
        int tempValue = 0;
        // run until wordQueue is empty
        while (wordQueue.length() > 0) {
            tempWord = wordQueue.dequeue();

            // if wordQueue already contains word, increase value by 1
            if (pairMap.hasKey(tempWord)) {
                tempValue = pairMap.value(tempWord);
                pairMap.replaceValue(tempWord, tempValue + 1);
            } else {
                // else add non duplicate to pairMap and noDupeQueue
                noDupeQueue.enqueue(tempWord);
                pairMap.add(tempWord, 1);
            }

        }
    }

    /**
     * Outputs the table tags in the generated HTML file.
     *
     * @param outWrite
     *            output stream to output file
     * @param noDupeQueue
     *            queue used to access words in map
     * @param pairMap
     *            map of words and their count used to print table contents
     *
     */
    public static void printTable(SimpleWriter outWrite,
            Queue<String> noDupeQueue, Map<String, Integer> pairMap) {
```

```java
        String tempWord = "";
        while (noDupeQueue.length() > 0) {
            tempWord = noDupeQueue.dequeue();
            outWrite.println("          <tr>");
            outWrite.println("            <td>" + tempWord + "</td>");
            outWrite.println(
                    "            <td>" + pairMap.value(tempWord) + "</td>");
            outWrite.println("          </tr>");
        }
    }

    /**
     * Outputs the "closing" tags in the generated HTML file.
     *
     * @param out
     *            the output stream
     * @ensures out.content = #out.content * [the HTML "closing" tags]
     *
     */
    public static void outputFooter(SimpleWriter out) {
        assert out != null : "Violation of: out is not null";
        assert out.isOpen() : "Violation of: out.is_open";
        out.println("    </table>");
        out.println("  </body>");
        out.println("</html>");
    }

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

        // Grab input and output files and create respective reader/writer
        out.print("Input file: ");
        String inFile = in.nextLine();
        SimpleReader inRead = new SimpleReader1L(inFile);
        out.print("Output file: ");
        String outFile = in.nextLine();
        SimpleWriter outWrite = new SimpleWriter1L(outFile);

        // Create data structures
        // Queue is used to store all words in text file, allowing duplicates
        Queue<String> wordQueue = new Queue1L<>();
        Queue<String> noDupeQueue = new Queue1L<>();

        // Map is used to store words and their count, no duplicates
        Map<String, Integer> pairMap = new Map1L<>();

        // Set is used to store separators needed to separate words
        Set<Character> separators = new Set1L<>();
        fillSeparators(separators);

        // Output header
        outputHeader(outWrite, inFile);

        // Process file and fill queue
        fillWordQueue(inRead, wordQueue, separators);

        //fill pairMap using wordQueue and fill noDupeQueue using map
        fillMap(wordQueue, noDupeQueue, pairMap);

        // sort noDupeQueue
        noDupeQueue.sort(new StringLT());

        printTable(outWrite, noDupeQueue, pairMap);

        // Output footer
        outputFooter(outWrite);

        // Close simple readers and writers
        inRead.close();
        in.close();
        outWrite.close();
        out.close();
    }

}

```