```java
import java.util.Comparator;

/**
 * Put a short phrase describing the program here.
 *
 * @author Jonathan (Jonny) Pater
 *
 */
public final class WordCounter {

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

    /**
     * Implements the compare method for the Comparator<String> interface.
     *
     * @author Jonathan (Jonny) Pater
     *
     */
    private static class StringLT implements Comparator<String> {
        // implement the compare method to be used by the Queue sort method
        @Override
        public int compare(String o1, String o2) {
            return o1.compareToIgnoreCase(o2);
        }

    }

    /**
     * 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)
```

```
53      */
54     private static void generateElements(String str,
   Set<Character> charSet) {
55         Set<Character> temp1 = new Set1L<>();
56         for (int i = 0; i < str.length(); i++) {
57             char temp = str.charAt(i);
58             if (!temp1.contains(temp)) {
59                 //checks if a given character is already in the
   set
60                 temp1.add(temp);
61             }
62         }
63         charSet.transferFrom(temp1);
64     }
65 
66     /**
67      * Returns the first "word" (maximal length string of
   characters not in
68      * {@code separators}) or "separator string" (maximal length
   string of
69      * characters in {@code separators}) in the given {@code text}
   starting at
70      * the given {@code position}.
71      *
72      * @param text
73      *            the {@code String} from which to get the word or
   separator
74      *            string
75      * @param position
76      *            the starting index
77      * @param separators
78      *            the {@code Set} of separator characters
79      * @return the first word or separator string found in {@code
   text} starting
80      *         at index {@code position}
81      ** @ensures <pre>
82      * nextWordOrSeparator =
83      * text[position, position + |nextWordOrSeparator|) and
84      * if entries(text[position, position + 1)) intersection
   separators = {}
85      * then
86      *    entries(nextWordOrSeparator) intersection separators =
   {} and
87      *      (position + |nextWordOrSeparator| = |text| or
```

```
 88      *        entries(text[position, position + |
    nextWordOrSeparator| + 1))
 89      *          intersection separators /= {})
 90      * else
 91      *    entries(nextWordOrSeparator) is subset of separators and
 92      *     (position + |nextWordOrSeparator| = |text| or
 93      *      entries(text[position, position + |
    nextWordOrSeparator| + 1))
 94      *       is not subset of separators)
 95      * </pre>
 96      */
 97
 98     private static String nextWordOrSeparator(String text, int
    position,
 99             Set<Character> separators) {
100         StringBuilder returned = new StringBuilder("");
101         String subStr = text.substring(position);
102         char firstChar = subStr.charAt(0);
103         boolean contain = separators.contains(firstChar);
104         //checks the first character of the substring to see
105         //if its a separator or not
106         int i = 0; //i is a loop counter
107         if (contain) {
108             while (i < subStr.length()
109                     //loop adds separators together to the
    returned word
110                     && separators.contains(subStr.charAt(i))) {
111                 returned.append(subStr.charAt(i));
112                 i++;
113             }
114         } else {
115             while (i < subStr.length()
116                     //loops adds characters together to form a
    word.
117                     //Is terminated by a separator
118                     && !separators.contains(subStr.charAt(i))) {
119                 returned.append(subStr.charAt(i));
120                 i++;
121             }
122         }
123         return returned.toString();
124     }
125
126     /**
```

```
127     * Returns a map containing all of the unique words found in a
    given input
128     * text file along with the number of times each unique word
    occurs in the
129     * text document.
130     *
131     * @param input
132     *            The input SimpleReader object containing the
    text file to be
133     *            used
134     * @return A map object wordCount
135     * @requires The input SimpleReader object contains a valid
    text file to be
136     *           used and that the array words is sorted
    alphabetically
137     * @ensures The returned map object contains keys for all
    unique words found
138     *          in the input text file and contains values for
    each key
139     *          representing the number of occurrences for each
    unique word
140     */
141    private static Map<String, Integer> countWords(SimpleReader
    input) {
142        Map<String, Integer> wordCount = new Map1L<>();
143        int position = 0;
144        final String separatorStr = " \t, \n . - ? ! = + ( ) & @#$
    %^&{[}] |/ \" "
145                + ": ; ><`~ 1 2 3 4 5 6 7 8 9 0";
146        //All characters that are considered separators for the
147        //nextWordOrSeparator method
148        Set<Character> separatorSet = new Set1L<>();
149        generateElements(separatorStr, separatorSet);
150        while (!input.atEOS()) {
151            String line = input.nextLine();
152            while (position < line.length()) {
153                String word = nextWordOrSeparator(line, position,
    separatorSet);
154                position = position + word.length();
155                //increases position to the starting index of the
    next word or separator
156                char firstChar = word.charAt(0);
157                if (Character.isLetter(firstChar)) {
158                    //checks if the returned string is a word,
```

```
159                     //indicated with a letter in the first index
    of the string
160                     if (!wordCount.hasKey(word)) {
161                         //checks if the word is a key or not
    before adding
162                         //it to the map object
163                         wordCount.add(word, 1);
164                     } else {
165                         int count = wordCount.value(word);
166                         wordCount.replaceValue(word, count + 1);
167                     }
168                 }
169             }
170             position = 0;
171         }
172         return wordCount;
173     }
174 
175     /**
176      * Generates an HMTL page containing a table of all the unique
    words found
177      * in the input text document with each word's corresponding
    number of
178      * occurrences in the text file.
179      *
180      * @param out
181      *            the output stream
182      * @param words
183      *            the array containing all unique words found in
    the input text
184      *            file
185      * @param wordCount
186      *            map object containing all unique words and their
    corresponding
187      *            word count
188      * @param inFile
189      *            the input stream
190      * @requires words != null, wordCount !=null, out.is.open, and
    in.is.open
191      * @ensures a valid HTML document will be the output of the
    method
192      *
193      */
194     private static void outputHTML(SimpleWriter out, Queue<String>
```

```
    words,
195             Map<String, Integer> wordCount, String inFile) {
196         out.println("<html> <head> <title>Words Counted in " +
    inFile
197                 + "</title> </head> <body> <h2>Words Counted in "
    + inFile
198                 + "</h2> <hr>");
199         out.println("<table border = \"1\"> <tbody>");
200         out.println("<tr> <th>Words</th> <th>Counts</th>  </tr>");
201         int length = words.length();
202         for (int i = 0; i < length; i++) {
203             //loop to add each unique word to the table
204             String newWord = words.dequeue();
205             out.println("<tr> <td>" + newWord + "</td>");
206             out.println("<td>" + wordCount.value(newWord) + "</td>
    </tr>");
207         }
208         out.println("</tbody> </table> </body> </html>");
209     }
210
211     /**
212      * Constructs and returns a Queue containing all unique words
    found in the
213      * input text document sorted alphabetically.
214      *
215      * @param words
216      *            the {@code Map} containing all unique words to
    be placed in
217      *            the Queue and sorted.
218      * @return the {@code Queue} containing all unique words from
    the input text
219      *         document sorted alphabetically
220      * @requires words /= null
221      * @ensures the {@code Queue} returned contains all unique
    words and is
222      *          sorted alphabetically
223      */
224
225     private static Queue<String> generateSortedQueue(
226             Map<String, Integer> words) {
227         Queue<String> sortedWords = new Queue1L<>();
228         for (Map.Pair<String, Integer> temp : words) {
229             String word = temp.key();
230             sortedWords.enqueue(word);
```

```
231         }
232         Comparator<String> order = new StringLT();
233         sortedWords.sort(order);
234         return sortedWords;
235     }
236 
237     /**
238      * Main method.
239      *
240      * @param args
241      *            the command line arguments
242      */
243     public static void main(String[] args) {
244         SimpleReader in = new SimpleReader1L();
245         SimpleWriter out = new SimpleWriter1L();
246         out.print("Enter an input file name: ");
247         String input = in.nextLine();
248         SimpleReader inFile1 = new SimpleReader1L(input);
249         out.print("\nEnter an output file name (with a .html at
    the end): ");
250         String output = in.nextLine();
251         SimpleWriter outFile = new SimpleWriter1L(output);
252         Map<String, Integer> wordCount = countWords(inFile1);
253         //get words and their number of occurrences in the
    document
254         out.println(wordCount); // TEST
255         Queue<String> sortedWords =
    generateSortedQueue(wordCount);
256         //sorts all of the unique words alphabetically in a Queue
    object
257         out.println(sortedWords); // TEST
258         outputHTML(outFile, sortedWords, wordCount, input);
259         in.close();
260         out.close();
261         inFile1.close();
262         outFile.close();
263     }
264 
265 }
266 
```