```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;

/**
 * Put a short phrase describing the program here.
 *
 * @author Bashir Ali
 *
 */
public final class WordCounter {

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

    /**
     * Comparator class used to sort strings alphabetically.
     *
     * @param o1
     *            The first string to be compared
     * @param o2
     *            The second string to be compared
     * @return A negative number if o1 is 'less than' o2,
positive if o1 is
     *         'greater than' o2, and zero if o1 and o2 are
'equal'
```

```
37      */
38     private static class Sort implements Comparator<String> {
39         @Override
40         public int compare(String o1, String o2) {
41             return
   o1.toLowerCase().compareTo(o2.toLowerCase());
42         }
43     }
44 
45     /**
46      * Returns the next word or separator from the text
   starting from the
47      * specified position.
48      *
49      * @param text
50      *            The text to get the next word or separator
   from
51      * @param position
52      *            The starting position to search for the next
   word or separator
53      * @param separators
54      *            A set of symbols considered as separators
55      * @return The next word or separator found in the text
   starting from the
56      *         given position.
57      */
58     private static String nextWordOrSeparator(String text, int
   position,
59             Set<Character> separators) {
60         assert text != null : "Violation of: text is not null";
61         assert separators != null : "Violation of: separators
   is not null";
62         assert 0 <= position : "Violation of: 0 <= position";
63         assert position <= text.length() : "Violation of:
   position < |text|";
64 
65         int i = position;
66         int n = text.length();
67 
```

```
68          // Find the end of the word or separator
69          if (!separators.contains(text.charAt(i))) {
70              while (i < n && !
    separators.contains(text.charAt(i))) {
71                  i++;
72              }
73          } else {
74              while (i < n &&
    separators.contains(text.charAt(i))) {
75                  i++;
76              }
77          }
78
79          return text.substring(position, i);
80
81      }
82
83      /**
84       * Reads each word and adds it to a map with corresponding
    occurrences.
85       * Words are also put into a queue and sorted by
    alphabetical order
86       *
87       * @param input
88       *            The SimpleReader to read input from.
89       * @param map
90       *            The Map to populate with key-value pairs.
91       * @requires format Term and previous definition to be
    separated by a single
92       *           line.
93       * @return A sorted Queue containing the keys from the Map.
94       */
95      public static Queue<String> getKeysAndValues(SimpleReader
    input,
96              Map<String, Integer> m) {
97          // empty queue for keys to be stored in
98          Queue<String> queue = new Queue1L<>();
99          Comparator<String> comparator = new Sort();
100
```

```
101         // set of separator
102         Set<Character> separators = new Set1L<>();
103         separators.add(':');
104         separators.add(';');
105         separators.add('|');
106         separators.add('~');
107         separators.add('!');
108         separators.add(' ');
109         separators.add(',');
110
111         /*
112          * while not at the end, store a line of the text file as a string.
113          * Then, using next word or separator, go through the string assigning
114          * words to a map with their amount of occurrences and skipping over
115          * separators
116          */
117         while (!input.atEOS()) {
118             int index = 0;
119             String line = input.nextLine();
120             while (index < line.length()) {
121                 String word = nextWordOrSeparator(line, index, separators);
122                 index = index + word.length();
123                 int occurences = 1;
124                 if (m.hasKey(word) && !separators.contains(word.charAt(0))) {
125                     Map.Pair<String, Integer> pair = m.remove(word);
126                     occurences = pair.value();
127                     occurences++;
128                     m.add(word, occurences);
129                 } else if (!m.hasKey(word)
130                         && !separators.contains(word.charAt(0))) {
131                     queue.enqueue(word);
132                     m.add(word, occurences);
```

```
133                 }
134             }
135         }
136
137         // sort the words in the queue
138         queue.sort(comparator);
139         input.close();
140         return queue;
141     }
142
143     /**
144      * generate output page in HTML containing a table with
    each word in
145      * alphabetic order and the number of times it occurs
146      *
147      * @param outFile
148      *            file to print HTML lines in
149      * @param queue
150      *            alphabet ordered words in a queue
151      * @param map
152      *            map containing word and how much it occurs
153      */
154     public static void generateHTMLPage(SimpleWriter outFile,
    String inFileName,
155             Queue<String> queue, Map<String, Integer> map) {
156         outFile.println("<html>");
157         outFile.println("<head>");
158         outFile.println(
159                 "<title>" + "Words counted in " + inFileName +
    "</title>");
160         outFile.println("</head>");
161         outFile.println("<body>");
162         outFile.println("<h2>" + "Words counted in " +
    inFileName + "</h2>");
163         outFile.println("<hr />");
164         outFile.println("<table border=\"1\">");
165         outFile.println("<tr>");
166         outFile.println("<th>Words</th>");
167         outFile.println("<th>Counts</th>");
```

```
168         outFile.println("</tr>");
169
170         while (queue.length() != 0) {
171             String current = queue.dequeue();
172             outFile.println("<tr>");
173             outFile.println("<td>" + current + "</td>");
174             outFile.println("<td>" + map.value(current) + "</
    td>");
175             outFile.println("</tr>");
176         }
177
178         outFile.println("</table>");
179         outFile.println("</body>");
180         outFile.println("</html>");
181     }
182
183     /**
184      * Main method.
185      *
186      * @param args
187      *            the command line arguments
188      */
189     public static void main(String[] args) {
190         SimpleReader in = new SimpleReader1L();
191         SimpleWriter out = new SimpleWriter1L();
192
193         out.print("Enter a text file: ");
194         String inputFile = in.nextLine();
195         out.print("Enter an output file: ");
196         String outputFile = in.nextLine();
197
198         SimpleReader fileIn = new SimpleReader1L(inputFile);
199         SimpleWriter fileOut = new SimpleWriter1L(outputFile);
200
201         Map<String, Integer> map = new Map1L<>();
202
203         Queue<String> queue = getKeysAndValues(fileIn, map);
204         generateHTMLPage(fileOut, inputFile, queue, map);
205
```

```
206         in.close();
207         out.close();
208     }
209
210 }
211
```