

```

1  import java.util.Comparator;
13
14 /**
15  * Reads input file and outputs list words w/o duplicates and their frequency.
16  *
17  * @author Gabe Azzarita
18  *
19  */
20 public final class WordCounter {
21
22     /**
23      * No argument constructor--private to prevent instantiation.
24      */
25     private WordCounter() {
26     }
27
28     /**
29      * Compare {@code String}s in lexicographic order.
30      */
31     private static class StringLT implements Comparator<String> {
32         @Override
33         /**
34          * @param o1
35          *         first string
36          * @param o2
37          *         second string
38          * @ensures positive int, zero, or negative int if o1 is greater than,
39          *         equal to, or less than o2
40          * @return integer signaling which string is bigger
41          */
42         public int compare(String o1, String o2) {
43             return o1.compareTo(o2);
44         }
45     }
46
47     /**
48      * Outputs the "opening" tags in the generated HTML file.
49      *
50      * @param out
51      *         the output stream
52      * @param file
53      *         input file
54      * @ensures out.content = #out.content * [the HTML "opening" tags]
55      *
56      */
57     public static void outputHeader(SimpleWriter out, String file) {
58         out.println("<html>");
59         out.println("    <head>");
60         out.println("        <title> Word Counter </title>");
61         out.println("    </head>");
62         out.println("    <body>");
63         out.println("        <h2> Words Counted in " + file + "</h2>");
64         out.println("    <hr>");
65         out.println("        <table border = 1>");
66         out.println("            <tr>");
67         out.println("                <td><b>Word</b></td>");
68         out.println("                <td><b>Count</b></td>");
69         out.println("            </tr>");
70

```

```
71     }
72
73     /**
74      * Fills the separator set with whatever desired separators.
75      *
76      * @param separators
77      *        set of separators to fill
78      * @ensures separators set is filled with new separators
79      *
80      * @updates separators
81      */
82     public static void fillSeparators(Set<Character> separators) {
83         separators.add(' ');
84         separators.add(',');
85         separators.add('.');
86         separators.add('!');
87         separators.add('?');
88         separators.add('"');
89         separators.add(';');
90         separators.add(':');
91         separators.add('-');
92         separators.add('\t');
93     }
94
95     /**
96      * Reads input file, and copies words in wordQueue.
97      *
98      * @param inRead
99      *        reader for the input file
100     * @param wordQueue
101     *        queue that will be filled with words from file
102     * @param separators
103     *        set used to make sure we only add words to queue
104     * @ensures wordQueue contains all the words from input file
105     *
106     * @updates wordQueue
107     */
108     public static void fillWordQueue(SimpleReader inRead,
109         Queue<String> wordQueue, Set<Character> separators) {
110
111         String tempString = "";
112         String tempWord = "";
113         // Keep reading lines until we reach the end
114         while (!inRead.atEOS()) {
115             tempString = inRead.nextLine();
116             // Go through string until we have find a separator
117             for (int i = 0; i < tempString.length(); i++) {
118                 // If character is a letter, we add it to tempWord
119                 if (!separators.contains(tempString.charAt(i))) {
120                     tempWord += tempString.charAt(i);
121                     // for last character in line, add word to queue
122                     // and clear tempWord
123                     if (i == tempString.length() - 1) {
124                         wordQueue.enqueue(tempWord.toLowerCase());
125                         tempWord = "";
126                     }
127                 } else {
128                     // else if character is a separator, we add tempWord
```

```

130         // to queue if it's greater than 0 and clear tempWord
131         if (tempWord.length() > 0) {
132             wordQueue.enqueue(tempWord.toLowerCase());
133         }
134         tempWord = "";
135     }
136 }
137 }
138 }
139
140 /**
141  * Fills map, and keeps track of how many times a word shows up.
142  *
143  * @param wordQueue
144  *         queue that Map is filled from
145  * @param noDupeQueue
146  *         queue that will fill with only non duplicate words
147  * @param pairMap
148  *         the map containing words and its count
149  * @ensures pairMap is filled and words counts are updated
150  *
151  * @updates pairMap
152  * @clears wordQueue
153  *
154  */
155 public static void fillMap(Queue<String> wordQueue,
156     Queue<String> noDupeQueue, Map<String, Integer> pairMap) {
157     String tempWord = "";
158     int tempValue = 0;
159     // run until wordQueue is empty
160     while (wordQueue.length() > 0) {
161         tempWord = wordQueue.dequeue();
162
163         // if wordQueue already contains word, increase value by 1
164         if (pairMap.containsKey(tempWord)) {
165             tempValue = pairMap.value(tempWord);
166             pairMap.replaceValue(tempWord, tempValue + 1);
167         } else {
168             // else add non duplicate to pairMap and noDupeQueue
169             noDupeQueue.enqueue(tempWord);
170             pairMap.add(tempWord, 1);
171         }
172     }
173 }
174 }
175
176 /**
177  * Outputs the table tags in the generated HTML file.
178  *
179  * @param outWrite
180  *         output stream to output file
181  * @param noDupeQueue
182  *         queue used to access words in map
183  * @param pairMap
184  *         map of words and their count used to print table contents
185  *
186  */
187 public static void printTable(SimpleWriter outWrite,
188     Queue<String> noDupeQueue, Map<String, Integer> pairMap) {

```

```
189     String tempWord = "";
190     while (noDupeQueue.length() > 0) {
191         tempWord = noDupeQueue.dequeue();
192         outWrite.println("                <tr>");
193         outWrite.println("                <td>" + tempWord + "</td>");
194         outWrite.println(
195             "                <td>" + pairMap.value(tempWord) + "</td>");
196         outWrite.println("                </tr>");
197     }
198 }
199
200 /**
201  * Outputs the "closing" tags in the generated HTML file.
202  *
203  * @param out
204  *     the output stream
205  * @ensures out.content = #out.content * [the HTML "closing" tags]
206  *
207  */
208 public static void outputFooter(SimpleWriter out) {
209     assert out != null : "Violation of: out is not null";
210     assert out.isOpen() : "Violation of: out.is_open";
211     out.println("        </table>");
212     out.println("    </body>");
213     out.println("</html>");
214 }
215
216 /**
217  * Main method.
218  *
219  * @param args
220  *     the command line arguments
221  */
222 public static void main(String[] args) {
223     SimpleReader in = new SimpleReader1L();
224     SimpleWriter out = new SimpleWriter1L();
225
226     // Grab input and output files and create respective reader/writer
227     out.print("Input file: ");
228     String inFile = in.nextLine();
229     SimpleReader inRead = new SimpleReader1L(inFile);
230     out.print("Output file: ");
231     String outFile = in.nextLine();
232     SimpleWriter outWrite = new SimpleWriter1L(outFile);
233
234     // Create data structures
235     // Queue is used to store all words in text file, allowing duplicates
236     Queue<String> wordQueue = new Queue1L<>();
237     Queue<String> noDupeQueue = new Queue1L<>();
238
239     // Map is used to store words and their count, no duplicates
240     Map<String, Integer> pairMap = new Map1L<>();
241
242     // Set is used to store separators needed to separate words
243     Set<Character> separators = new Set1L<>();
244     fillSeparators(separators);
245
246     // Output header
247     outputHeader(outWrite, inFile);
```

```
248
249     // Process file and fill queue
250     fillWordQueue(inRead, wordQueue, separators);
251
252     //fill pairMap using wordQueue and fill noDupeQueue using map
253     fillMap(wordQueue, noDupeQueue, pairMap);
254
255     // sort noDupeQueue
256     noDupeQueue.sort(new StringLT());
257
258     printTable(outWrite, noDupeQueue, pairMap);
259
260     // Output footer
261     outputFooter(outWrite);
262
263     // Close simple readers and writers
264     inRead.close();
265     in.close();
266     outWrite.close();
267     out.close();
268 }
269
270 }
271
```