```
 1 import java.util.Comparator;
14
15 /**
16  * The program will count word occurrences in an HTML document based on a given
17  * input file. The HTML document will include a table with the words and a
18  * count, along with being listed in alphabetical order.
19  *
20  * @author Chloe Feller
21  *
22  */
23 public final class WordCounter {
24
25     /**
26      * No argument constructor--private to prevent instantiation.
27      */
28     private WordCounter() {
29     }
30
31     /**
32      * Reads words from the input file and adds them to a {@code Map}. Words are
33      * not alphabetized yet
34      *
35      * @param words
36      *            the {@code Map} of words
37      * @param file
38      *            file input by user
39      *
40      * @requires file.isOpen
41      * @requires words != null
42      * @replaces words
43      *
44      */
45     private static void readFile(Map<String, Integer> words,
46             SimpleReader file) {
47         assert file.isOpen() : "Violation of: file is open";
48         assert words != null : "Violation of: words is not null";
49
50         String separator = " \t,.-;'/\"@#$%&()";
51         Set<Character> charSet = new Set1L<Character>();
52
53         generateElements(separator, charSet);
54
55         /*
56          * Read through the file until all lines are read, while adding words to
57          * the Map
58          */
59         while (!file.atEOS()) {
60             String line = file.nextLine();
61             int i = 0;
62
63             while (i < line.length()) {
64                 String text = nextWordOrSeparator(line, i, charSet);
65                 if (!charSet.contains(text.charAt(0))) {
66                     /*
67                      * Sees if words contains the word. If it does not, the word
68                      * is added. If it does, the number of times it has appeared
69                      * is increased.
70                      */
71                     if (words.hasKey(text)) {
```

```
72                          int numberAppear = words.value(text);
73                          numberAppear++;
74                          words.replaceValue(text, numberAppear);
75                      } else {
76                          words.add(text, 1);
77                      }
78                  }
79                  // Skip to the next word/separator
80                  i += text.length();
81              }
82          }
83
84      }
85
86      /**
87       * Generates the set of characters in the given {@code String} into the
88       * given {@code Set}.
89       *
90       * @param str
91       *            the given {@code String}
92       * @param charSet
93       *            the {@code Set} to be replaced
94       * @replaces charSet
95       * @ensures charSet = entries(str)
96       */
97      public static void generateElements(String str, Set<Character> charSet) {
98          for (int i = 0; i < str.length(); i++) {
99              if (!charSet.contains(str.charAt(i))) {
100                 charSet.add(str.charAt(i));
101             }
102         }
103     }
104
105     /**
106      * Returns the first "word" (maximal length string of characters not in
107      * {@code separators}) or "separator string" (maximal length string of
108      * characters in {@code separators}) in the given {@code text} starting at
109      * the given {@code position}.
110      *
111      * @param text
112      *            the {@code String} from which to get the word or separator
113      *            string
114      * @param position
115      *            the starting index
116      * @param separators
117      *            the {@code Set} of separator characters
118      * @return the first word or separator string found in {@code text} starting
119      *         at index {@code position}
120      * @requires 0 <= position < |text|
121      * @ensures <pre>
122      * nextWordOrSeparator =
123      *   text[position, position + |nextWordOrSeparator|)  and
124      * if entries(text[position, position + 1)) intersection separators = {}
125      * then
126      *   entries(nextWordOrSeparator) intersection separators = {}  and
127      *   (position + |nextWordOrSeparator| = |text|  or
128      *    entries(text[position, position + |nextWordOrSeparator| + 1))
129      *      intersection separators /= {})
130      * else
```

```
131      *   entries(nextWordOrSeparator) is subset of separators  and
132      *   (position + |nextWordOrSeparator| = |text|  or
133      *    entries(text[position, position + |nextWordOrSeparator| + 1))
134      *      is not subset of separators)
135      * </pre>
136      */
137     public static String nextWordOrSeparator(String text, int position,
138             Set<Character> separators) {
139         assert text != null : "Violation of: text is not null";
140         assert position >= 0 : "Violation of: position is not >= 0";
141         assert position < text
142                 .length() : "Violation of: position is not < |text|";
143         assert separators != null : "Violation of: separators is not null";
144
145         String str = "";
146         char returnedChar = 'a';
147
148         if (separators.contains(text.charAt(position))) {
149             for (int i = 0; i < text.substring(position, text.length())
150                     .length(); i++) {
151                 returnedChar = text.charAt(position + i);
152                 if (separators.contains(returnedChar)) {
153                     str = str + returnedChar;
154                 } else {
155                     i = text.substring(position, text.length()).length();
156                 }
157             }
158         } else {
159             for (int i = 0; i < text.substring(position, text.length())
160                     .length(); i++) {
161                 returnedChar = text.charAt(position + i);
162                 if (!separators.contains(returnedChar)) {
163                     str = str + returnedChar;
164                 } else {
165                     i = text.substring(position, text.length()).length();
166                 }
167             }
168         }
169
170         return str;
171     }
172
173     /**
174      * Separates the words in the given {@code Map} in alphabetical order.
175      *
176      * @param words
177      *            the given {@code Map}
178      * @param sorter
179      *            a {@code Queue} used to help sort the {@code Map}
180      *
181      * @requires words != null
182      * @requires sorter != null
183      * @updates words
184      */
185     private static void alphabetMap(Map<String, Integer> words,
186             Queue<String> sorter) {
187         assert words != null : "Violation of : words is not null";
188         assert sorter != null : "Violation of : sorter is not null";
189
```

```
190         Map<String, Integer> tempMap = new Map1L<String, Integer>();
191         Comparator<String> order = new StringLT();
192         Queue<String> tempSorter = new Queue1L<String>();
193
194         /*
195          * Add the words from the Map to the Queue
196          */
197         while (words.iterator().hasNext()) {
198             Pair<String, Integer> newPair = words.removeAny();
199             tempMap.add(newPair.key(), newPair.value());
200             sorter.enqueue(newPair.key());
201         }
202
203         /*
204          * Sort the Queue
205          */
206         sorter.sort(order);
207
208         /*
209          * Add the words in alphabetical order back to the Map
210          */
211         while (sorter.iterator().hasNext()) {
212             String text = sorter.dequeue();
213             Pair<String, Integer> sortedPair = tempMap.remove(text);
214             words.add(sortedPair.key(), sortedPair.value());
215             tempSorter.enqueue(text);
216         }
217
218         sorter.transferFrom(tempSorter);
219     }
220
221     /**
222      *
223      * Compares two {@code Strings} and returns whether they are in alphabetical
224      * order or not.
225      *
226      */
227     private static class StringLT implements Comparator<String> {
228         @Override
229         public int compare(String one, String two) {
230             return one.toLowerCase().compareTo(two.toLowerCase());
231         }
232     }
233
234     /**
235      * Outputs the alphabetized words into an HTML document, which includes a
236      * table with the words and how many times they appear.
237      *
238      * @param out
239      *            the output file
240      * @param file
241      *            the input file the user gave
242      * @param words
243      *            the {@code Map} containing the words and how many times they
244      *            appear
245      * @param sorter
246      *            the {@code Queue} containing the words
247      *
248      * @requires out and file are open
```

```
249      * @requires words and sorter != null
250      */
251     private static void outputFile(SimpleWriter out, SimpleReader file,
252             Map<String, Integer> words, Queue<String> sorter) {
253         assert out.isOpen() : "Violation of : out is open";
254         assert file.isOpen() : "Violation of : file is open";
255         assert words != null : "Violation of : words is not null";
256         assert sorter != null : "Violation of : sorter is not null";
257
258         /*
259          * Print out beginning of HTML file
260          */
261         out.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\" ?>");
262         out.println("<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//EN'"
263                 + " 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>");
264         out.println("<html xmlns='http://www.w3.org/1999/xhtml'>");
265
266         /*
267          * Print out title
268          */
269         out.println("<head>");
270         out.println("<title>" + file.name() + "</title>");
271
272         /*
273          * Print out body
274          */
275         out.println("<body>");
276         out.println("<h1>Words Counted in " + file.name() + "</h2>");
277
278         /*
279          * Print out table
280          */
281         out.println("<table border=\"1\">");
282         out.println("<tr>");
283         out.println("<th>Words</th>");
284         out.println("<th>Counter</th>");
285         out.println("</tr>");
286
287         while (words.iterator().hasNext()) {
288             Pair<String, Integer> word = words.remove(sorter.dequeue());
289             out.println("<tr>");
290             out.println("<td>" + word.key() + "</td>");
291             out.println("<td>" + word.value() + "</td>");
292             out.println("</tr>");
293         }
294
295         out.println("</table>");
296         out.println("</body>");
297         out.println("</head>");
298         out.println("</html>");
299     }
300
301     /**
302      * Main method.
303      *
304      * @param args
305      *            the command line arguments
306      */
307     public static void main(String[] args) {
```

```
308         SimpleReader in = new SimpleReader1L();
309         SimpleWriter out = new SimpleWriter1L();
310
311         /*
312          * Ask user for an input and output file
313          */
314         out.print("Please give an input file: ");
315         String input = in.nextLine();
316         SimpleReader file = new SimpleReader1L(input);
317
318         out.print("Please give an output file: ");
319         String output = in.nextLine();
320         SimpleWriter write = new SimpleWriter1L(output);
321
322         /*
323          * Puts words in the input file to Strings / Queues (look into to see
324          * which is better)
325          */
326         Map<String, Integer> words = new Map1L<String, Integer>();
327         readFile(words, file);
328
329         /*
330          * Separate words based on alphabetical order
331          */
332         Queue<String> sorter = new Queue1L<String>();
333         alphabetMap(words, sorter);
334
335         /*
336          * Make a method for the output file's code, including a table and the
337          * words in alphabetical order
338          */
339         outputFile(write, file, words, sorter);
340
341         /*
342          * Close input and output streams
343          */
344         file.close();
345         write.close();
346         in.close();
347         out.close();
348     }
349
350 }
351
```