```java
import java.util.Comparator;

import components.map.Map;
import components.map.Map.Pair;
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;

/**
 * Program that reads a text file and then generates an HTML file with words'
 * occurrence in the file.
 *
 * @author Zhuoyang Li
 *
 */

public final class WordCounter {
    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private WordCounter() {
    }

    /**
     * Generate a HTML page with words' occurrence in the file.
     *
     * @param in
     *            the input stream
     * @param out
     *            the output stream
     * @param word
     *            the map of words and their occurrences
     * @param wordQueue
     *            the queue of words
     * @param fileName
     *            the name of the input file
     * @clear {@code wordQueue}, {@code word}
     * @ensures <pre>
     * {@code wordQueue = < >}
     * {@code word = < >}
     * </pre>
     */

    private static void html(SimpleReader in, SimpleWriter out,
            Map<String, Integer> word, Queue<String> wordQueue,
            String fileName) {
        int numberOfWords = 0;
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Word Counter : " + fileName + "</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>Word Counter " + fileName + "</h2>");
        out.println("<hr />");
```

```
 61         out.println("<table border=\"1\">");
 62         out.println("<tr>");
 63         out.println("<th>Words</th>");
 64         out.println("<th>Counts</th>");
 65         out.println("</tr>");
 66         //print the words and their occurrences in the table
 67         while (!wordQueue.length() == 0)) {
 68             String word1 = wordQueue.dequeue();
 69             Pair<String, Integer> pair = word.remove(word1);
 70             out.println("<tr>");
 71             out.println("<td>" + pair.key() + "</td>");
 72             out.println("<td>" + pair.value() + "</td>");
 73             out.println("</tr>");
 74             numberOfWords += pair.value();
 75         }
 76         out.println("</table>");
 77         out.println("<hr />");
 78         out.println("<p> Total number of words: " + numberOfWords + "</
    p>");
 79         out.println("</body>");
 80         out.println("</html>");
 81 
 82     }
 83 
 84     /**
 85      * Generates the set of characters in the given {@code String} into
    the
 86      * given {@code Set}.
 87      *
 88      * @param str
 89      *            the given {@code String}
 90      * @param charSet
 91      *            the {@code Set} to be replaced
 92      * @replaces charSet
 93      * @ensures charSet = entries(str)
 94      */
 95     private static void generateElements(String str, Set<Character>
    charSet) {
 96         assert str != null : "Violation of: str is not null";
 97         assert charSet != null : "Violation of: charSet is not null";
 98 
 99         for (int i = 0; i < str.length(); i++) {
100             if (!charSet.contains(str.charAt(i))) {
101                 charSet.add(str.charAt(i));
102             }
103         }
104 
105     }
106 
107     /**
108      * Returns the first "word" (maximal length string of characters not
    in
109      * {@code separators}) or "separator string" (maximal length string of
110      * characters in {@code separators}) in the given {@code text}
    starting at
111      * the given {@code position}.
112      *
113      * @param text
114      *            the {@code String} from which to get the word or
    separator
115      *            string
```

```
116      * @param position
117      *            the starting index
118      * @param separators
119      *            the {@code Set} of separator characters
120      * @return the first word or separator string found in {@code text}
   starting
121      *         at index {@code position}
122      * @requires 0 <= position < |text|
123      * @ensures <pre>
124      * nextWordOrSeparator =
125      *   text[position, position + |nextWordOrSeparator|)  and
126      * if entries(text[position, position + 1)) intersection separators =
   {}
127      * then
128      *   entries(nextWordOrSeparator) intersection separators = {}  and
129      *   (position + |nextWordOrSeparator| = |text|  or
130      *    entries(text[position, position + |nextWordOrSeparator| + 1))
131      *      intersection separators /= {})
132      * else
133      *   entries(nextWordOrSeparator) is subset of separators  and
134      *   (position + |nextWordOrSeparator| = |text|  or
135      *    entries(text[position, position + |nextWordOrSeparator| + 1))
136      *      is not subset of separators)
137      * </pre>
138      */
139     private static String nextWordOrSeparator(String text, int position,
140             Set<Character> separators) {
141         assert text != null : "Violation of: text is not null";
142         assert separators != null : "Violation of: separators is not
   null";
143         assert 0 <= position : "Violation of: 0 <= position";
144         assert position < text.length() : "Violation of: position < |
   text|";
145 
146         //
147         String result = "";
148         boolean isSeparator = separators.contains(text.charAt(position));
149         int i = position;
150 
151         while (i < text.length()
152                 && (isSeparator == separators.contains(text.charAt(i)))) {
153             result += text.charAt(i);
154             i++;
155         }
156 
157         return result;
158 
159     }
160 
161     /**
162      * Generate a map of words and their occurrences.
163      *
164      * @param file
165      *            the input stream
166      * @param word
167      *            the map of words and their occurrences
168      * @updates {@code word}
169      * @ensures <pre>
170      * {@code wordQueue = < >}
171      * </pre>
172      */
```

```
173     public static void mapGenerate(SimpleReader file,
174             Map<String, Integer> word) {
175         assert file.isOpen() : "Violation of: file is open";
176         assert word != null : "Violation of: word is not null";
177
178         word.clear();
179         int i = 0; //position
180         Set<Character> separatorSet = new Set1L<Character>();
181
182         //common separators
183         String separators = " ,.?!;:-()[]{}'\"/\\@#$%^&*";
184         generateElements(separators, separatorSet);
185
186         while (!file.atEOS()) {
187             String line = file.nextLine();
188             i = 0;
189             while (i < line.length()) {
190                 String word1 = nextWordOrSeparator(line, i, separatorSet);
191                 if (!separatorSet.contains(word1.charAt(0))) {
192                     if (word.hasKey(word1)) {
193                         // word1 is in the set
194                         int value = word.value(word1);
195                         word.replaceValue(word1, value + 1);
196                     } else {
197                         // word1 is not in the set
198                         word.add(word1, 1);
199                     }
200                 }
201                 i += word1.length(); //update position for the next
    nextWordOrSeparator
202             }
203         }
204
205     }
206
207     /**
208      * Compare {@code String}s in alphabetical order.
209      */
210     public static class WordComparator implements Comparator<String> {
211
212         /**
213          * Compare {@code String}s in alphabetical order.
214          *
215          * @param o1
216          *            the first {@code String} to compare
217          * @param o2
218          *            the second {@code String} to compare
219          * @return -1 if o1 is less than o2, 1 if o1 is greater than o2, 0
    if o1
220          *         is equal to o2
221          */
222         @Override
223         public int compare(String o1, String o2) {
224             return o1.compareToIgnoreCase(o2);
225         }
226     }
227
228     /**
229      * Sort the map of words and their occurrences in alphabetical order.
230      *
231      * @param word
```

```
232      *            the map of words and their occurrences
233      * @param order
234      *            the comparator of {@code String}s
235      * @param wordQueue
236      *            the queue of words
237      * @updates {@code keyQueue}
238      * @clear {@code words}
239      * @ensures <pre>
240      * {@code words = < >}
241      * </pre>
242      */
243     public static void sortMap(Map<String, Integer> word,
244             Comparator<String> order, Queue<String> wordQueue) {
245         assert word != null : "Violation of: words is not null";
246         assert order != null : "Violation of: order is not null";
247         assert wordQueue != null : "Violation of: wordQueue is not null";
248 
249         Map<String, Integer> temp = word.newInstance();
250         Queue<String> tempQueue = wordQueue.newInstance();
251         wordQueue.clear();
252 
253         while (word.iterator().hasNext()) {
254             Pair<String, Integer> pair = word.removeAny();// get one word
255             //add the pair to the temp map
256             temp.add(pair.key(), pair.value());
257             //create a queue of words with same order as the map
258             wordQueue.enqueue(pair.key());
259         }
260 
261         wordQueue.sort(order);
262         while (wordQueue.iterator().hasNext()) {
263             //get one word by alphabetical order
264             String word1 = wordQueue.dequeue();
265             //get the pair of the word from the map
266             Pair<String, Integer> pair = temp.remove(word1);
267             word.add(pair.key(), pair.value());
268             tempQueue.enqueue(word1);
269         }
270         wordQueue.transferFrom(tempQueue);
271     }
272 
273     /**
274      * Main method.
275      *
276      * @param args
277      *            the command line arguments
278      */
279     public static void main(String[] args) {
280         SimpleReader in = new SimpleReader1L();
281         SimpleWriter out = new SimpleWriter1L();
282         Map<String, Integer> word = new Map1L<String, Integer>();
283         Queue<String> wordQueue = new Queue1L<String>();
284         Comparator<String> order = new WordComparator();
285 
286         out.print("Please enter the name of the input file(included file):
    ");
287         String fileName = in.nextLine();
288         SimpleReader file = new SimpleReader1L(fileName);
289         //users should type "data/" and ".txt" in the input file name
290         out.print("Please enter the name of the output file: ");
291         SimpleWriter output = new SimpleWriter1L(in.nextLine());
```

```
292
293         mapGenerate(file, word);
294         sortMap(word, order, wordQueue);
295         html(file, output, word, wordQueue, fileName);
296
297         in.close();
298         out.close();
299         file.close();
300         output.close();
301
302     }
303 }
304
```