```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;

/**
 * Program to take a file of text, count the times of every word, and make a
 * page displaying each word and number of words that appear.
 *
 * @author Yiming Cheng
 *
 */
public final class WordCounter {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private WordCounter() {
        // no code needed here
    }

    /**
     * In order to sort the terms in an alphabetic order.
     */
    private static class stringList implements Comparator<String> {
        @Override
        public int compare(String o1, String o2) {
            return o1.toLowerCase().compareTo(o2.toLowerCase());
        }

    }

    /**
     * Returns the first "word" (maximal length string of characters not in
     * {@code separators}) or "separator string" (maximal length string of
     * characters in {@code separators}) in the given {@code text} starting at
     * the given {@code position}.
     *
     * @param text
     *            the {@code String} which are got from the word
     * @param position
     *            the starting position of index
     * @param separators
     *            the certain punctuation which is from the list.
     * @return the first word or separator string in the index position
     * @requires 0 <= position < |text|
     * @ensures <pre>
     * nextWordOrSeparator =
     *   text[position, position + |nextWordOrSeparator|)  and
     * if entries(text[position, position + 1)) intersection separators = {}
     * then
     *   entries(nextWordOrSeparator) intersection separators = {}  and
     *   (position + |nextWordOrSeparator| = |text|  or
     *    entries(text[position, position + |nextWordOrSeparator| + 1))
```

```
63      *      intersection separators /= {})
64      * else
65      *   entries(nextWordOrSeparator) is subset of separators  and
66      *   (position + |nextWordOrSeparator| = |text|  or
67      *    entries(text[position, position + |nextWordOrSeparator| + 1))
68      *      is not subset of separators)
69      * </pre>
70      */
71     public static String nextWordOrSeparator(String text, int position,
72             Set<Character> separators) {
73         assert text != null : "Violation of: text is not null";
74         assert separators != null : "Violation of: separators is not null";
75         assert 0 <= position : "Violation of: 0 <= position";
76         assert position < text.length() : "Violation of: position < |text|";
77
78         int endPos = -1;
79         String word = "";
80         int i = position;
81         /*
82          * find the corresponding substrings by separator
83          */
84         while (i < text.length()) {
85             if (separators.contains(text.charAt(i)) && endPos == -1) {
86                 endPos = i;
87             }
88             if (endPos == -1) {
89                 word = text.substring(position, text.length());
90             } else if (endPos == position) {
91                 word = text.substring(position, position + 1);
92             } else {
93                 word = text.substring(position, endPos);
94             }
95             i++;
96         }
97
98         return word;
99
100    }
101
102    /**
103     * Outputs the main page index.html. Expected elements from this method:
104     *
105     * @param map
106     *            the map of terms and their occurrences
107     * @param out
108     *            the output stream
109     * @param title
110     *            the string of the file name
111     * @param titlelist
112     *            the queue of unique words
113     * @updates out.content
114     * @requires out.is_open
115     * @ensures out.content = #out.content * [the HTML tags]
116     */
117    private static void outputhtml(Map<String, Integer> map, SimpleWriter out,
118            String title, Queue<String> titlelist) {
119        assert out.isOpen() : "Violation of: out.is_open";
120        //print the whole formats for the page.
121        out.print("<html>\r\n" + "<head>\r\n" + "<title> " + title
122                + "</title>\r\n" + "\r\n" + "</head>\r\n" + "\r\n");
123
124        out.print(
```

```
125                 "<body>\r\n" + "<h2>" + title + "</h2>\r\n" + "<hr>" + "\r\n");
126
127         out.print("<table border= \"1\"> \r\n");
128         out.print("<tbody>\r\n");
129
130         out.print("<tr>\r\n");
131         out.print("<th>" + "Words" + "</th>\r\n");
132         out.print("<th>" + "Counts" + "</th>\r\n");
133         out.print("</tr>\r\n");
134         int counter = titlelist.length();
135         int i = 0;
136         while (i < counter) {
137             String word = titlelist.dequeue();
138             out.print("<tr>\r\n");
139             out.print("<td>" + word + "</td>\r\n");
140             out.print("<td>" + map.value(word) + "</td>\r\n");
141             out.print("</tr>\r\n");
142             i++;
143         }
144
145         out.print("</tbody>\r\n" + "</table>\r\n" + "</body>\r\n" + "</html>");
146     }
147
148     /**
149      * Generates the set of characters in the given {@code String} into the
150      * given {@code Set}.
151      *
152      * @param str
153      *            the given {@code String}
154      * @param strList
155      *            the {@code Set} to be replaced
156      * @replaces strSet
157      * @ensures strSet = entries(str)
158      */
159     private static void generateElements(String str, Set<Character> strList) {
160         assert str != null : "Violations of: str is not null";
161         assert strList != null : "Violation of: strSet is not null";
162         int i = 0;
163         while (i < str.length()) {
164             char c = str.charAt(i);
165             if (!strList.contains(c)) {
166                 strList.add(c);
167             }
168             i++;
169         }
170     }
171
172     /**
173      * process the data in the list and map into the new queue.
174      *
175      * @param list
176      *            the original list that need to be move
177      * @param map
178      *            the map that store the words and the times that the words
179      *            would appear
180      * @return the queue that would be displayed.
181      */
182     private static Queue<String> processItems(Queue<String> list,
183             Map<String, Integer> map) {
184         int iteratorNum = list.length();
185
186         Queue<String> temp = list.newInstance();
```

```
187         for (int i = 0; i < iteratorNum; i++) {
188             String word = list.dequeue();
189             temp.enqueue(word);
190             list.enqueue(word);
191         }
192         int a = 0;
193         while (a < iteratorNum) {
194             String word = list.dequeue();
195             int number = 1;
196             if (map.hasKey(word)) {
197                 int count = map.value(word);
198                 map.replaceValue(word, count + 1);
199             } else {
200                 map.add(word, number);
201             }
202             a++;
203         }
204 
205         Set<String> words = new Set1L<>();
206         for (String x : temp) {
207             if (!words.contains(x)) {
208                 words.add(x);
209             }
210         }
211         for (String x : words) {
212             list.enqueue(x);
213         }
214 
215         return list;
216 
217     }
218 
219     /**
220      * Main method.
221      *
222      * @param args
223      *            the command line arguments; unused here
224      */
225     public static void main(String[] args) {
226         SimpleReader in = new SimpleReader1L();
227         SimpleWriter out = new SimpleWriter1L();
228         //ask the users about the file.
229 
230         out.print("Enter file that will be used to obtain the words: ");
231         String text = in.nextLine();
232         SimpleReader inFile = new SimpleReader1L(text);
233         //ask the users about the name of the html page.
234         out.print("Enter the name of a of the html page to write to: ");
235         String htmlpage = in.nextLine();
236         SimpleWriter outputhtml = new SimpleWriter1L(htmlpage);
237 
238         Queue<String> list = new Queue1L<>();
239         Map<String, Integer> map = new Map1L<>();
240         //separate the string into the different parts.
241         final String separatorStr = " \t, .-";
242         Set<Character> separatorList = new Set1L<>();
243         generateElements(separatorStr, separatorList);
244         while (!inFile.atEOS()) {
245             String line = inFile.nextLine();
246             int i = 0;
247             while (i < line.length()) {
248                 String word = nextWordOrSeparator(line, i, separatorList);
```

```
249                 boolean isWord = true;
250                 for (int j = 0; j < word.length(); j++) {
251                     char c = word.charAt(j);
252                     if (separatorList.contains(c)) {
253                         isWord = false;
254                     }
255                 }
256                 if (isWord) {
257                     list.enqueue(word);
258                 }
259                 i += word.length();
260             }
261         }
262 
263         // make the right order
264         Comparator<String> cs = new stringList();
265 
266         Queue<String> temp = list.newInstance();
267         temp = processItems(list, map);
268         temp.sort(cs);
269 
270         String title = "Words Counted in " + text;
271         outputhtml(map, outputhtml, title, temp);
272 
273         inFile.close();
274         out.close();
275         in.close();
276     }
277 
278 }
```