```java
import java.util.Comparator;

import components.map.Map;
import components.map.Map.Pair;
import components.map.Map1L;
import components.set.Set;
import components.set.Set1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
import components.sortingmachine.SortingMachine;
import components.sortingmachine.SortingMachine1L;
import components.utilities.Reporter;

/**
 * Read txt and find the amount words that appear most frequently. The amount
 is
 * an integer entered by the user.
 *
 * @author Qinuo Shi & Yiming Cheng
 *
 */
public final class TagCloudGenerator {

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

    /**
     * there separator are used in countingWords method.
     */
    public static final String Separator = "\\ \t\n\r,-.!?[]';:/()@&~`\"";

    /**
     * read the txt file and enter all the words into set and map.
     *
     * @param contentIn
     *            SimpleReader
     * @param content
     *            the map include all words and their occurrences
     * @param wordNum
     *            the set include all words
     * @requires contentIn should not be empty
     * @update content and wordNum
```

```
47      * @ensures content and wordNum should not be empty, and all keys in the
48      *          content must be the same as wordNum.
49      */
50     public static void contentInSetAndMap(SimpleReader contentIn,
51             Set<String> content, Map<String, Integer> wordNum) {
52         assert contentIn != null : "Violation of: txt file is not null";
53
54         /*
55          * Store all possible non-alphabetic symbols in a set.
56          */
57         Set<Character> sepSet = new Set1L<Character>();
58         for (int i = 0; i < Separator.length(); i++) {
59             sepSet.add(Separator.charAt(i));
60         }
61
62         /*
63          * Read content in the file and write them into set and map.
64          */
65         while (!contentIn.atEOS()) {
66             String eachLine = contentIn.nextLine();
67             int pos = 0;
68             while (pos < eachLine.length()) {
69                 String word = nextWordOrSeparator(eachLine, pos, sepSet);
70                 if (!sepSet.contains(word.charAt(0))) {
71                     if (!(wordNum.hasKey(word))) {
72                         wordNum.add(word, 1);
73                         content.add(word);
74                     } else {
75                         /*
76                          * If a word is already recorded, change the map
   value,
77                          * and leaving the set unchanged.
78                          */
79                         int num = wordNum.value(word) + 1;
80                         wordNum.replaceValue(word, num);
81                     }
82                 }
83                 pos += word.length();
84             }
85         }
86     }
87
88     /**
89      * Returns the first "word" (maximal length string of characters not in
90      * {@code separators}) or "separator string" (maximal length string of
91      * characters in {@code separators}) in the given {@code text} starting at
92      * the given {@code position}.
```

```
 93      *
 94      * @param str
 95      *            the {@code String} from which to get the word or separator
 96      *            string
 97      * @param pos
 98      *            the starting index
 99      * @param sepSet
100      *            the {@code Set} of separator characters
101      * @return the first word or separator string found in {@code str}
    starting
102      *         at index {@code pos}
103      * @requires 0 <= pos < |str|
104      * @ensures <pre>
105      * nextWordOrSeparator =
106      *   str[pos, pos + |nextWordOrSeparator|)  and
107      * if entries(str[pos, pos + 1)) intersection separators = {}
108      * then
109      *   entries(nextWordOrSeparator) intersection separators = {}  and
110      *   (pos + |nextWordOrSeparator| = |str|  or
111      *    entries(str[pos, pos + |nextWordOrSeparator| + 1))
112      *      intersection separators /= {})
113      * else
114      *   entries(nextWordOrSeparator) is subset of separators  and
115      *   (pos + |nextWordOrSeparator| = |str|  or
116      *    entries(str[pos, pos + |nextWordOrSeparator| + 1))
117      *      is not subset of separators)
118      * </pre>
119      */
120     public static String nextWordOrSeparator(String str, int pos,
121             Set<Character> sepSet) {
122         assert str != null : "Violation of: str is not null";
123         assert sepSet != null : "Violation of: separators is not null";
124         assert 0 <= pos : "Violation of: 0 <= pos";
125         assert pos < str.length() : "Violation of: pos < |str|";
126
127         int endPos = -1;
128         String word = "";
129         /*
130          * Use for loop to find the term's position.
131          */
132         for (int i = pos; i < str.length(); i++) {
133             if (sepSet.contains(str.charAt(i)) && endPos == -1) {
134                 endPos = i;
135             }
136         }
137         /*
138          * Depending on the case, intercepts the corresponding substring.
```

```
139          */
140         if (endPos == pos) {
141             word = str.substring(pos, endPos + 1);
142         } else if (endPos == -1) {
143             word = str.substring(pos);
144         } else {
145             word = str.substring(pos, endPos);
146         }
147 
148         return word;
149     }
150 
151     /**
152      * compare two integers and return a value.
153      */
154     private static class CompareNum
155             implements Comparator<Map.Pair<String, Integer>> {
156         @Override
157         public int compare(Map.Pair<String, Integer> p1,
158                 Map.Pair<String, Integer> p2) {
159             return p2.value() - p1.value();
160         }
161     }
162 
163     /**
164      * compare two strings and return a value.
165      */
166     private static class CompareString
167             implements Comparator<Map.Pair<String, Integer>> {
168         @Override
169         public int compare(Map.Pair<String, Integer> p1,
170                 Map.Pair<String, Integer> p2) {
171             return p1.key().toLowerCase().compareTo(p2.key().toLowerCase());
172         }
173     }
174 
175     /**
176      * read the txt file and enter all the words into set and map.
177      *
178      * @param wordWithOccurrence
179      *            the map include all words and their occurrences
180      * @param sortingInt
181      *            a SortingMachine for ints
182      * @param sortingString
183      *            a SortingMachine for strings
184      * @param number
185      *            the amount of the most frequent words
```

```
186      * @return a String that record the maximum and the minimum occurrences
187      * @requires wordWithOccurrence should not be empty
188      * @ensures MaxMin need record "maximum value | minimum value"
189      */
190     private static String sortingMapKeyValue(
191             Map<String, Integer> wordWithOccurrence,
192             SortingMachine<Pair<String, Integer>> sortingInt,
193             SortingMachine<Pair<String, Integer>> sortingString, int number) {
194         assert wordWithOccurrence != null : "Violation of: the map is not
    null";
195         assert number > 0 : "Violation of: Number must be positive";
196 
197         String maxMin = "";
198 
199         /*
200          * Add all ints in SortingMachine.
201          */
202         for (Pair<String, Integer> p : wordWithOccurrence) {
203             sortingInt.add(p);
204         }
205 
206         /*
207          * If the input number is larger than the amount of words in file,
208          * report an error.
209          */
210         Reporter.assertElseFatalError(number <= sortingInt.size(),
211                 "number is too large");
212         sortingInt.changeToExtractionMode();
213 
214         /*
215          * Sort all ints and Strings.
216          */
217         for (int i = 0; i < number; i++) {
218             Pair<String, Integer> p = sortingInt.removeFirst();
219             if (i == 0) {
220                 maxMin = p.value().toString() + "|";
221             }
222             if (i == number - 1) {
223                 maxMin += p.value().toString();
224             }
225             sortingString.add(p);
226         }
227         sortingString.changeToExtractionMode();
228 
229         return maxMin;
230     }
231 
```

```
232     /**
233      * read the txt file and enter all the words into set and map.
234      *
235      * @param sortingString
236      *            a SortingMachine for strings
237      * @param out
238      *            the output
239      * @param htmlName
240      *            the name of output file
241      * @param maxMin
242      *            a String that record the maximum and the minimum occurrences
243      * @requires sortingString should not be empty, htmlName should not be
244      *           empty, maxMin should not be empty
245      */
246     public static void writeHtml(
247             SortingMachine<Pair<String, Integer>> sortingString,
248             SimpleWriter out, String htmlName, String maxMin) {
249         assert sortingString != null : "Violation of: the sortingString is not
    null";
250         assert htmlName != null : "Violation of: the htmlName is not null";
251         assert maxMin != null : "Violation of: the maxMin is not null";
252
253         /*
254          * Output front part of html.
255          */
256         out.println("<html>");
257         out.println("  <head>");
258         out.println("    <title>" + "Top " + sortingString.size() + " words in
    "
259                 + htmlName + "</title>");
260         out.println(
261                 "    <link href=\"http://web.cse.ohio-
    state.edu/software/2231/web-sw2/assignments/projects/tag-cloud-
    generator/data/tagcloud.css\" rel =\"stylesheet\" type=\"text/css\"");
262         out.println("  </head>");
263         out.println("  <body>");
264         out.println("    <h2> Top " + sortingString.size() + " words in "
265                 + htmlName + "</h2>");
266         out.println("    <hr>");
267         out.println("    <div class = \"cdiv\">");
268         out.println("      <p class=\"cbox\">");
269
270         /*
271          * Subtract the maximum and minimum value from maxMin.
272          */
273         int symbolpos = maxMin.indexOf('|');
274         int max = Integer.parseInt(maxMin.substring(0, symbolpos));
```

```
275         int min = Integer.parseInt(maxMin.substring(symbolpos + 1));
276
277         /*
278          * Output each line of middle part of html.
279          */
280         final int maxTypeSize = 48;
281         final int minTypeSize = 11;
282         int length = sortingString.size();
283         for (int i = 0; i < length; i++) {
284             Pair<String, Integer> wordCount = sortingString.removeFirst();
285             String word = wordCount.key();
286             int numOfWords = wordCount.value();
287
288             /*
289              * Use a formula to calculate the type size of each word.
290              */
291             int typeSize = ((maxTypeSize - minTypeSize) * (numOfWords - min)
292                     / (max - min)) + minTypeSize;
293
294             out.println("        <span style=\"cursor:default\" class=\"" +
    "f"
295                     + typeSize + "\" title=\"count: " + numOfWords + "\">"
296                     + word + "</span>");
297         }
298
299         /*
300          * Output last part of html.
301          */
302         out.println("      </p>");
303         out.println("    </div>");
304         out.println("  </body>");
305         out.println("</html>");
306
307     }
308
309     /**
310      * Main method.
311      *
312      * @param args
313      *            the command line arguments
314      */
315     public static void main(String[] args) {
316         SimpleReader in = new SimpleReader1L();
317         SimpleWriter out = new SimpleWriter1L();
318
319         /*
320          * Ask the users to enter the txt file name they want to check.
```

```
321          */
322         out.println("Enter a txt file name: ");
323         String fileName = in.nextLine();
324 
325         /*
326          * Ask the users to enter the html file name they want to write in.
327          */
328         out.println("Enter a html file name: ");
329         String htmlName = in.nextLine();
330 
331         /*
332          * Ask the users to enter a number for the amount of the most frequent
333          * words they want to check.
334          */
335         out.println(
336                 "Enter a positive number for the amount of the most frequent
    words: ");
337         int num = in.nextInteger();
338         while (!(num > 0)) {
339             out.println(
340                     num + " is not a positive number, enter another number:
    ");
341             num = in.nextInteger();
342         }
343 
344         /*
345          * Build a set and map which will store the contents in txt files.
346          */
347         SimpleReader fileContent = new SimpleReader1L(fileName);
348         Set<String> word = new Set1L<>();
349         Map<String, Integer> wordWithOccurrence = new Map1L<>();
350         contentInSetAndMap(fileContent, word, wordWithOccurrence);
351 
352         /*
353          * If users' value is lager the amount of all word, just arrange all
    the
354          * words in the txt file.
355          */
356         if (num > word.size()) {
357             num = word.size();
358         }
359 
360         /*
361          * Sort ints and strings in map by using SortingMachine
362          */
363         Comparator<Map.Pair<String, Integer>> orderInt = new CompareNum();
364         SortingMachine<Map.Pair<String, Integer>> orderIntMap = new
```

```
   SortingMachine1L<>(
365                 orderInt);
366         Comparator<Map.Pair<String, Integer>> orderString = new CompareString
   ();
367         SortingMachine<Map.Pair<String, Integer>> orderStringMap = new
   SortingMachine1L<>(
368                 orderString);
369
370         /*
371          * Return a String that record the maximum and the minimum occurrences
372          * int the map which called wordWithOccurrence.
373          */
374         String maxMin = sortingMapKeyValue(wordWithOccurrence, orderIntMap,
375                 orderStringMap, num);
376
377         /*
378          * Output the content in html format to a html file.
379          */
380         SimpleWriter htmlContent = new SimpleWriter1L(htmlName);
381         writeHtml(orderStringMap, htmlContent, htmlName, maxMin);
382
383         /*
384          * Close all things.
385          */
386         fileContent.close();
387         htmlContent.close();
388         in.close();
389         out.close();
390     }
391 }
392
```