```
 1 import java.util.Comparator;
15
16 /**
17  * Creates a tag cloud from a given file input text.
18  *
19  * @author Chloe Feller
20  * @author Krish Patel
21  *
22  */
23 public final class TagCloudGenerator {
24
25     /**
26      * No argument constructor--private to prevent instantiation.
27      */
28     private TagCloudGenerator() {
29     }
30
31     /**
32      * This is a numerical ordering system which orders the largest numbers over
33      * the smaller numbers.
34      */
35     private static class Sort implements Comparator<Map.Pair<String, Integer>> {
36         @Override
37         public int compare(Map.Pair<String, Integer> one,
38                 Map.Pair<String, Integer> two) {
39             int compared = 0;
40             if (one.value().equals(two.value())) {
41                 compared = one.key().compareToIgnoreCase(two.key());
42             } else {
43                 compared = two.value().compareTo(one.value());
44             }
45             return compared;
46         }
47
48     }
49
50     /**
51      * This is an alphabetical ordering system which orders words starting from
52      * a all the way to z.
53      */
54     private static class SortTwo
55             implements Comparator<Map.Pair<String, Integer>> {
56         @Override
57         public int compare(Map.Pair<String, Integer> one,
58                 Map.Pair<String, Integer> two) {
59             return one.key().compareToIgnoreCase(two.key());
60         }
61     }
62
63     /**
64      * Reads words from the input file and adds them to a {@code Map}. Words are
65      * not alphabetized yet.
66      *
67      * @param words
68      *            the {@code Map} of words
69      * @param file
70      *            file input by user
71      *
72      * @requires file.isOpen
```

```
 73      * @requires words != null
 74      * @replaces words
 75      *
 76      */
 77     private static void readFile(Map<String, Integer> words,
 78             SimpleReader file) {
 79         assert file.isOpen() : "Violation of: file is open";
 80         assert words != null : "Violation of: words is not null";
 81
 82         String separator = " \t,.-;'/\"@#$%&()*`";
 83         Set<Character> charSet = new Set1L<Character>();
 84
 85         generateElements(separator, charSet);
 86
 87         /*
 88          * Read through the file until all lines are read, while adding words to
 89          * the Map
 90          */
 91         while (!file.atEOS()) {
 92             String line = file.nextLine();
 93             int i = 0;
 94
 95             while (i < line.length()) {
 96                 String text = nextWordOrSeparator(line, i, charSet);
 97                 if (!charSet.contains(text.charAt(0))) {
 98                     /*
 99                      * Sees if words contains the word. If it does not, the word
100                      * is added. If it does, the number of times it has appeared
101                      * is increased.
102                      */
103                     if (words.hasKey(text)) {
104                         int numberAppear = words.value(text);
105                         numberAppear++;
106                         words.replaceValue(text, numberAppear);
107                     } else {
108                         words.add(text, 1);
109                     }
110                 }
111                 // Skip to the next word/separator
112                 i += text.length();
113             }
114         }
115
116     }
117
118     /**
119      * Generates the set of characters in the given {@code String} into the
120      * given {@code Set}.
121      *
122      * @param str
123      *            the given {@code String}
124      * @param charSet
125      *            the {@code Set} to be replaced
126      * @replaces charSet
127      * @ensures charSet = entries(str)
128      */
129     private static void generateElements(String str, Set<Character> charSet) {
130         for (int i = 0; i < str.length(); i++) {
131             if (!charSet.contains(str.charAt(i))) {
```

```
132                 charSet.add(str.charAt(i));
133             }
134         }
135     }
136
137     /**
138      * Returns the first "word" (maximal length string of characters not in
139      * {@code separators}) or "separator string" (maximal length string of
140      * characters in {@code separators}) in the given {@code text} starting at
141      * the given {@code position}.
142      *
143      * @param text
144      *            the {@code String} from which to get the word or separator
145      *            string
146      * @param position
147      *            the starting index
148      * @param separators
149      *            the {@code Set} of separator characters
150      * @return the first word or separator string found in {@code text} starting
151      *         at index {@code position}
152      * @requires 0 <= position < |text|
153      * @ensures <pre>
154      * nextWordOrSeparator =
155      *   text[position, position + |nextWordOrSeparator|)  and
156      * if entries(text[position, position + 1)) intersection separators = {}
157      * then
158      *   entries(nextWordOrSeparator) intersection separators = {}  and
159      *   (position + |nextWordOrSeparator| = |text|  or
160      *    entries(text[position, position + |nextWordOrSeparator| + 1))
161      *      intersection separators /= {})
162      * else
163      *   entries(nextWordOrSeparator) is subset of separators  and
164      *   (position + |nextWordOrSeparator| = |text|  or
165      *    entries(text[position, position + |nextWordOrSeparator| + 1))
166      *      is not subset of separators)
167      * </pre>
168      */
169     private static String nextWordOrSeparator(String text, int position,
170             Set<Character> separators) {
171         assert text != null : "Violation of: text is not null";
172         assert position >= 0 : "Violation of: position is not >= 0";
173         assert position < text
174                 .length() : "Violation of: position is not < |text|";
175         assert separators != null : "Violation of: separators is not null";
176
177         String str = "";
178         char returnedChar = 'a';
179
180         if (separators.contains(text.charAt(position))) {
181             for (int i = 0; i < text.substring(position, text.length())
182                     .length(); i++) {
183                 returnedChar = text.charAt(position + i);
184                 if (separators.contains(returnedChar)) {
185                     str = str + returnedChar;
186                 } else {
187                     i = text.substring(position, text.length()).length();
188                 }
189             }
190         } else {
```

```
191             for (int i = 0; i < text.substring(position, text.length())
192                     .length(); i++) {
193                 returnedChar = text.charAt(position + i);
194                 if (!separators.contains(returnedChar)) {
195                     str = str + returnedChar;
196                 } else {
197                     i = text.substring(position, text.length()).length();
198                 }
199             }
200         }
201
202         return str;
203     }
204
205     /**
206      * Outputs the opening tags for the output HTML file.
207      *
208      * @param out
209      *            output stream
210      * @param file
211      *            input file given by user
212      * @param x
213      *            number of words given by user
214      * @updates {@code out}
215      * @requires <pre>
216      * {@code file} is open and not null and {@code out} is open
217      * </pre>
218      * @ensures <pre>
219      * {@code out = #out * tags}
220      * </pre>
221      */
222     private static void outputHeader(SimpleWriter out, String file, int x) {
223         assert out != null : "Violation of : out is not null";
224         assert out.isOpen() : "Violation of : out is not open";
225         assert file != null : "Violation of : file is not null";
226
227         /**
228          * Print out beginning of HTML file
229          */
230         out.println("<html>");
231         out.println("<head>");
232
233         /**
234          * Print out title
235          */
236         out.println("<title>Top " + x + " words in " + file + "</title>");
237         out.println("<link href=\"http://web.cse.ohio-state.edu/software/2231/"
238                 + "web-sw2/assignments/projects/tag-cloud-generator/data/"
239                 + "tagcloud.css\" rel=\"stylesheet\" type=\"text/css\">");
240         out.println("<link href=\"doc/tagcloud.css\" "
241                 + "rel=\"stylesheet\" type=\"text/css\">");
242         out.println("</head>");
243
244         /**
245          * Print out body
246          */
247         out.println("<body>");
248         out.println("<h2>Top " + x + " Words Counted in " + file + "</h2>");
249         out.println("<hr>");
```

```
250         out.println("<div class=\"cdiv\">");
251         out.println("<p class=\"cbox\">");
252
253     }
254
255     /**
256      * Prints footer for the output HTML file.
257      *
258      * @param out
259      *            output stream
260      * @updates {@code out}
261      * @requires <pre>
262      * {@code out} is open
263      * </pre>
264      * @ensures <pre>
265      * {@code out = #out * tags}
266      * </pre>
267      */
268     private static void outputFooter(SimpleWriter out) {
269         out.println("</p>");
270         out.println("</div>");
271         out.println("</body>");
272         out.println("</html>");
273     }
274
275     /**
276      * This sorts the words into two different groups. It starts sorting through
277      * a comparator in numerical order and then a second ordering method being
278      * alphabetically. It would then print to the output file in HTML format
279      * with the appropriate fonts.
280      *
281      * @param mapCount
282      *            This is the map of words and numbers that show up
283      * @param out
284      *            output file stream
285      * @param words
286      *            number of words given by user
287      */
288     private static void sortingAndFonts(Map<String, Integer> mapCount,
289             SimpleWriter out, int words) {
290
291         //numerical order
292         Comparator<Pair<String, Integer>> nums = new Sort();
293         SortingMachine<Map.Pair<String, Integer>> sort;
294         sort = new SortingMachine1L<Pair<String, Integer>>(nums);
295
296         while (mapCount.size() > 0) {
297             Pair<String, Integer> r = mapCount.removeAny();
298             sort.add(r);
299         }
300
301         sort.changeToExtractionMode();
302
303         //alphabetical ordering
304         Comparator<Pair<String, Integer>> numsTwo = new SortTwo();
305         SortingMachine<Pair<String, Integer>> sortTwo;
306         sortTwo = new SortingMachine1L<Pair<String, Integer>>(numsTwo);
307
308         int min = 0;
```

```
309         int max = 0;
310 
311         //loop ordering
312         for (int i = 0; (i < words) && (1 < sort.size()); i++) {
313             Pair<String, Integer> wording = sort.removeFirst();
314             int neg = words - 1;
315             if (i == 0) {
316                 max = wording.value();
317             } else if (i == neg) {
318                 min = wording.value();
319             }
320             sortTwo.add(wording);
321         }
322         sortTwo.changeToExtractionMode();
323 
324         //alphabetical + printing to the output stream
325         while (sortTwo.size() > 0) {
326             Pair<String, Integer> removed = sortTwo.removeFirst();
327 
328             final int eleven = 11;
329             final int fortyeight = 48;
330             int sizeFont = 0;
331             if (removed.value() == min) {
332                 sizeFont = eleven;
333             } else if (removed.value() == max) {
334                 sizeFont = fortyeight;
335             } else {
336                 sizeFont = eleven
337                         + ((removed.value() * (fortyeight - eleven)) / (max));
338             }
339 
340             String f = "f" + sizeFont;
341 
342             out.println("<span style=\"cursor:default\" class=\"" + f
343                     + "\" title=\"count: " + removed.value() + "\">"
344                     + removed.key() + "</span>");
345         }
346 
347     }
348 
349     /**
350      * Main method.
351      *
352      * @param args
353      *            the command line arguments
354      */
355     public static void main(String[] args) {
356         SimpleReader in = new SimpleReader1L();
357         SimpleWriter out = new SimpleWriter1L();
358 
359         /*
360          * Ask for input and output file, along with number of words.
361          */
362         out.print("Enter an input file: ");
363         String fileIn = in.nextLine();
364         out.print("Enter an output file: ");
365         String fileOut = in.nextLine();
366         out.print("Enter number of words: ");
367         int words = in.nextInteger();
```

```
368
369         Reporter.assertElseFatalError(words > 0,
370                 "Number of words must be greater than 0");
371
372         /*
373          * Create output file and print header of HTML file.
374          */
375         SimpleWriter output = new SimpleWriter1L(fileOut);
376         outputHeader(output, fileIn, words);
377
378         /*
379          * Create input file.
380          */
381         SimpleReader input = new SimpleReader1L(fileIn);
382
383         /*
384          * Read file and sort values in Map in both alphabetical and decreasing
385          * order.
386          */
387         Map<String, Integer> tmpMap = new Map1L<String, Integer>();
388         readFile(tmpMap, input);
389         sortingAndFonts(tmpMap, output, words);
390
391         /*
392          * Print footer of HTML file.
393          */
394         outputFooter(output);
395
396         /*
397          * Close input and output streams
398          */
399         in.close();
400         out.close();
401         output.close();
402         input.close();
403     }
404
405 }
406
```