

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

```

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

```

```

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

```

```

193         str = str + returnedChar;
194     } else {
195         i = text.substring(position, text.length()).length();
196     }
197 }
198 } else {
199     for (int i = 0; i < text.substring(position, text.length())
200         .length(); i++) {
201         returnedChar = text.charAt(position + i);
202         if (!separators.contains(returnedChar)) {
203             str = str + returnedChar;
204         } else {
205             i = text.substring(position, text.length()).length();
206         }
207     }
208 }
209
210     return str;
211 }
212
213 /**
214  * Outputs the opening tags for the output HTML file.
215  *
216  * @param out
217  *     output stream
218  * @param file
219  *     input file given by user
220  * @param x
221  *     number of words given by user
222  * @updates {@code out}
223  * @requires <pre>
224  * {@code file} is open and not null and {@code out} is open
225  * </pre>
226  * @ensures <pre>
227  * {@code out = #out * tags}
228  * </pre>
229  */
230 private static void outputHeader(PrintWriter out, String file, int x) {
231     assert out != null : "Violation of : out is not null";
232     assert file != null : "Violation of : file is not null";
233
234     /*
235      * Print out beginning of HTML file
236      */
237     out.println("<html>");
238     out.println("<head>");
239
240     /*
241      * Print out title
242      */
243     out.println("<title>Top " + x + " words in " + file + "</title>");
244     out.println("<link href=\"http://web.cse.ohio-state.edu/software/2231/"
245         + "web-sw2/assignments/projects/tag-cloud-generator/data/"
246         + "tagcloud.css\" rel=\"stylesheet\" type=\"text/css\">");
247     out.println("<link href=\"doc/tagcloud.css\" "
248         + "rel=\"stylesheet\" type=\"text/css\">");
249     out.println("</head>");
250
251     /*

```

```

252         * Print out body
253         */
254         out.println("<body>");
255         out.println("<h2>Top " + x + " Words Counted in " + file + "</h2>");
256         out.println("<hr>");
257         out.println("<div class=\"cdiv\">");
258         out.println("<p class=\"cbox\">");
259
260     }
261
262     /**
263      * Prints footer for the output HTML file.
264      *
265      * @param out
266      *      output stream
267      * @updates {@code out}
268      * @requires <pre>
269      * {@code out} is open
270      * </pre>
271      * @ensures <pre>
272      * {@code out = #out * tags}
273      * </pre>
274      */
275     private static void outputFooter(PrintWriter out) {
276         out.println("</p>");
277         out.println("</div>");
278         out.println("</body>");
279         out.println("</html>");
280     }
281
282     /**
283      * This sorts the words into two different groups. It starts sorting through
284      * a comparator in numerical order and then a second ordering method being
285      * alphabetically. It would then print to the output file in HTML format
286      * with the appropriate fonts.
287      *
288      * @param mapCount
289      *      This is the map of words and numbers that show up
290      * @param out
291      *      output file stream
292      * @param words
293      *      number of words given by user
294      */
295     private static void sortingAndFonts(Map<String, Integer> mapCount,
296         PrintWriter out, int words) {
297
298         Iterator<Map.Entry<String, Integer>> sorting = mapCount.entrySet()
299             .iterator();
300         Map.Entry<String, Integer> pair;
301
302         //numerical order
303         Comparator<Map.Entry<String, Integer>> nums = new Sort();
304         List<Map.Entry<String, Integer>> numberOrder;
305         numberOrder = new LinkedList<Map.Entry<String, Integer>>();
306
307         while (mapCount.size() > 0) {
308             pair = sorting.next();
309             sorting.remove();
310

```

```
311         numberOrder.add(pair);
312     }
313
314     numberOrder.sort(nums);
315
316     Iterator<Map.Entry<String, Integer>> sort2 = numberOrder.iterator();
317
318     //alphabetical ordering
319     Comparator<Map.Entry<String, Integer>> numsTwo = new SortTwo();
320     List<Map.Entry<String, Integer>> sortTwo;
321     sortTwo = new LinkedList<Map.Entry<String, Integer>>();
322
323     int min = 0;
324     int max = 0;
325
326     //loop ordering
327     for (int i = 0; (i < words) && (1 < numberOrder.size()); i++) {
328         Map.Entry<String, Integer> wording = sort2.next();
329         sort2.remove();
330
331         int neg = words - 1;
332
333         if (i == 0) {
334             max = wording.getValue();
335         } else if (i == neg) {
336             min = wording.getValue();
337         }
338         sortTwo.add(wording);
339     }
340
341     sortTwo.sort(numsTwo);
342
343     //alphabetical + printing to the output stream
344     while (sortTwo.size() > 0) {
345         Map.Entry<String, Integer> removed = sortTwo.remove(0);
346
347         final int eleven = 11;
348         final int fortyeight = 48;
349         int sizeFont = 0;
350         if (removed.getValue() == min) {
351             sizeFont = eleven;
352         } else if (removed.getValue() == max) {
353             sizeFont = fortyeight;
354         } else {
355             sizeFont = eleven
356                 + ((removed.getValue() * (fortyeight - eleven))
357                    / (max));
358         }
359
360         String f = "f" + sizeFont;
361
362         out.println("<span style=\"cursor:default\" class=\"" + f
363             + "\" title=\"count: " + removed.getValue() + "\">"
364             + removed.getKey().toLowerCase() + "</span>");
365     }
366 }
367
368
369 /**
```

```
370     * Main method.
371     *
372     * @param args
373     *     the command line arguments
374     */
375     public static void main(String[] args) {
376         /*
377          * Open input file.
378          */
379         BufferedReader in = new BufferedReader(
380             new InputStreamReader(System.in));
381         BufferedReader input = null;
382         String inRead = "";
383
384         try {
385             System.out.print("Enter an input file: ");
386             inRead = in.readLine();
387             input = new BufferedReader(new FileReader(inRead));
388         } catch (IOException e) {
389             System.err.println("Unable to open input file");
390         }
391
392         /*
393          * Open output file.
394          */
395         PrintWriter output = null;
396         String outRead;
397
398         try {
399             System.out.print("Enter an output file: ");
400             outRead = in.readLine();
401             output = new PrintWriter(
402                 new BufferedWriter(new FileWriter(outRead)));
403         } catch (IOException e) {
404             System.err.println("Unable to open output file");
405         }
406
407         /*
408          * Gather the number of words in the tag cloud.
409          */
410         System.out.print("Enter number of words: ");
411
412         int words = -1;
413
414         try {
415             while (words < 0) {
416                 words = Integer.parseInt(in.readLine());
417                 if (words < 0) {
418                     System.out.println("Must enter a positive number");
419                 }
420             }
421         } catch (IOException e) {
422             System.err.println("Error reading number");
423         } catch (NumberFormatException e) {
424             System.err.println("Number is not in the correct format");
425         }
426
427         /*
428          * Output the header of the HTML file.
```

```
429         */
430         Map<String, Integer> tmpMap = new HashMap<>();
431
432         /*
433         * Generate tags and sort them.
434         */
435         try {
436             readFile(tmpMap, input);
437         } catch (IOException e) {
438             System.err.println("Error reading lines in the input file");
439         }
440         outputHeader(output, inRead, words);
441         sortingAndFonts(tmpMap, output, words);
442
443         /*
444         * Output footer of HTML file and close output stream.
445         */
446         if (output != null) {
447             outputFooter(output);
448             output.close();
449         }
450
451         /*
452         * Close input streams
453         */
454         try {
455             in.close();
456         } catch (IOException e) {
457             System.err.println("Error closing files");
458         }
459
460         if (input != null) {
461             try {
462                 input.close();
463             } catch (IOException e) {
464                 System.err.println("Error closing files");
465             }
466         }
467     }
468 }
469 }
470
```