```
 1 import java.io.BufferedReader;
15
16 /**
17  * Java program that generates a tag cloud from a given input text.
18  *
19  * @author Gabe Azzarita and Ty Fredrick
20  *
21  */
22 public final class TagCloudJAVA {
23
24     /**
25      * No argument constructor--private to prevent instantiation.
26      */
27     private TagCloudJAVA() {
28     }
29
30     /**
31      * Compare entry values in numerical order.
32      */
33     private static class NumOrder
34             implements Comparator<Entry<String, Integer>> {
35
36         @Override
37         /**
38          * @param o1
39          *            first pair
40          * @param o2
41          *            second pair
42          * @ensures positive, negative, or zero int if o2.value is larger than,
43          *          less than, or equal to o1.value
44          *
45          * @return int signaling which pair.value() is larger
46          */
47         public int compare(Entry<String, Integer> o1,
48                 Entry<String, Integer> o2) {
49             return o2.getValue().compareTo(o1.getValue());
50         }
51
52     }
53
54     /**
55      * Compare entry keys in alphabetical order.
56      */
57     private static class AlphaOrder
58             implements Comparator<Entry<String, Integer>> {
59
60         @Override
61         /**
62          * @param o1
63          *            first pair
64          * @param o2
65          *            second pair
66          * @ensures positive, negative, or zero int if o1.key is larger than,
67          *          less than, or equal to o2.key
68          *
69          * @return int signaling which pair.key() is alphabetically larger
70          */
71         public int compare(Entry<String, Integer> o1,
72                 Entry<String, Integer> o2) {
```

```
 73             return o1.getKey().compareTo(o2.getKey());
 74         }
 75
 76     }
 77
 78     /**
 79      * Minimum font value.
 80      */
 81     private static final int MIN_FONT = 11;
 82
 83     /**
 84      * Maximum font value.
 85      */
 86     private static final int MAX_FONT = 48;
 87
 88     /**
 89      * Fill set of separators with standard separators.
 90      *
 91      * @param s
 92      *            set of separators to fill
 93      * @ensures s contains ASCII characters [0, 65) + [91, 97) + [123, 127]
 94      *
 95      */
 96     private static void fillSeparator(HashSet<Character> s) {
 97         final int uppercaseStart = 65;
 98         final int uppercaseEnd = 91;
 99         final int lowercaseStart = 97;
100         final int lowercaseEnd = 123;
101         final int asciiEnd = 127;
102
103         // Using ASCII values we can add all non-letters to our separator set
104         for (int i = 0; i < uppercaseStart; i++) {
105             s.add((char) i);
106         }
107         for (int k = uppercaseEnd; k < lowercaseStart; k++) {
108             s.add((char) k);
109         }
110         for (int j = lowercaseEnd; j < asciiEnd; j++) {
111             s.add((char) j);
112         }
113     }
114
115     /**
116      * Fill map by reading lines of input file.
117      *
118      * @param inFile
119      *            valid text file to read
120      * @param terms
121      *            Map to fill with words and count of words
122      * @param separators
123      *            Set of separators
124      * @throws IOException
125      * @ensures "terms" contains all unique non-separator strings in text file
126      *          as keys, while keeping count of words in value
127      *
128      */
129     private static void fillMap(String inFile, HashMap<String, Integer> terms,
130             HashSet<Character> separators) throws IOException {
131
```

```
132         BufferedReader read = new BufferedReader(new FileReader(inFile));
133         String tempLine = read.readLine();
134         String tempWord = "";
135         int position = 0;
136 
137         while (tempLine != null) {
138 
139             // Read each line of input using nextWordOrSeparator
140             position = 0;
141             while (position < tempLine.length()) {
142                 tempWord = nextWordOrSeparator(tempLine, position, separators);
143 
144                 // Check that tempWord is NOT a separator
145                 if (!separators.contains(tempWord.charAt(0))) {
146                     // make all words lowercase
147                     tempWord = tempWord.toLowerCase();
148 
149                     // Check if tempWord is already in the map
150                     if (!terms.containsKey(tempWord)) {
151                         terms.put(tempWord, 1);
152                     } else {
153                         int value = terms.remove(tempWord);
154                         terms.put(tempWord, value + 1);
155                     }
156                 }
157                 position += tempWord.length();
158             }
159             tempLine = read.readLine();
160 
161         }
162         read.close();
163 
164     }
165 
166     /**
167      * Returns the first "word" (maximal length string of characters not in
168      * {@code separators}) or "separator string" (maximal length string of
169      * characters in {@code separators}) in the given {@code text} starting at
170      * the given {@code position}.
171      *
172      * @param text
173      *            the {@code String} from which to get the word or separator
174      *            string
175      * @param position
176      *            the starting index
177      * @param separators
178      *            the {@code Set} of separator characters
179      * @return the first word or separator string found in {@code text} starting
180      *         at index {@code position}
181      * @requires 0 <= position < |text|
182      * @ensures <pre>
183      * nextWordOrSeparator =
184      * text[position, position + |nextWordOrSeparator|) and
185      * if entries(text[position, position + 1)) intersection separators = {}
186      * then
187      * entries(nextWordOrSeparator) intersection separators = {} and
188      * (position + |nextWordOrSeparator| = |text| or
189      * entries(text[position, position + |nextWordOrSeparator| + 1))
190      * intersection separators /= {})
```

```
191      * else
192      * entries(nextWordOrSeparator) is subset of separators and
193      * (position + |nextWordOrSeparator| = |text| or
194      * entries(text[position, position + |nextWordOrSeparator| + 1))
195      * is not subset of separators)
196      * </pre>
197      */
198     public static String nextWordOrSeparator(String text, int position,
199             HashSet<Character> separators) {
200         assert text != null : "Violation of: text is not null";
201         assert separators != null : "Violation of: separators is not null";
202         assert 0 <= position : "Violation of: 0 <= position";
203         assert position < text.length() : "Violation of: position < |text|";
204 
205         String resultStr = "";
206         String subStr = text.substring(position);
207         char ch = text.charAt(position);
208         boolean containsCh = separators.contains(ch);
209 
210         if (!containsCh) {
211             // If first char is not separator, loop until separator is found
212             for (int i = 0; i < subStr.length(); i++) {
213                 ch = text.charAt(position + i);
214                 if (!separators.contains(ch)) {
215                     resultStr += ch;
216                 } else { // As soon as next char is separator
217                     // Essentially a break
218                     i = text.substring(position).length();
219                 }
220             }
221         } else { // Separator found, return separator
222             resultStr += ch;
223         }
224 
225         return resultStr;
226     }
227 
228     /**
229      * Print HTML file.
230      *
231      * @param o
232      *            PrintWriter
233      * @param inFile
234      *            valid file cloudTag is based off
235      * @param n
236      *            generate tagCloud for top n words
237      * @param maxCount
238      *            largest count
239      * @param pQ
240      *            alphabetically ordered priority queue containing top n words
241      * @throws IOException
242      * @ensures printed HTML file "outFile" containing tagCloud from "inFile"
243      *
244      */
245     private static void printHTML(PrintWriter o, String inFile, int n,
246             PriorityQueue<Entry<String, Integer>> pQ, double maxCount) {
247 
248         // print header
249         o.println("<html>");
```

```
250         o.println("<head>");
251         o.println("<title> Top " + n + " Words In " + inFile + "</title>");
252         o.println("<link href=\"http://web.cse.ohio-state.edu/software/"
253                 + "2231/web-" + "sw2/assignments/projects/tag-cloud-generator/"
254                 + "data/tagcloud.css\" rel=\"stylesheet\" type=\"text/css\">");
255
256         o.println("<link href=\"tagcloud.css\" rel=\"stylesheet\" "
257                 + "type=\"text/css\">");
258         o.println("</head>");
259         o.println("<body>");
260         o.println("<h2> Top " + n + " Words In " + inFile + "</h2>");
261
262         o.println("<hr>");
263
264         o.println("<div class = \"cdiv\">");
265         o.println("<p class = \"cbox\">");
266
267         // Empty priority queue and print with font size calculated
268         if (pQ.size() > 0) {
269
270             int multiplier = (MAX_FONT - MIN_FONT);
271
272             while (pQ.size() > 0) {
273                 Entry<String, Integer> tempPair = pQ.remove();
274
275                 //calculate font size
276                 double fontSize = ((tempPair.getValue() / maxCount) * multiplier
277                         + MIN_FONT);
278
279                 o.println("<span style = \"cursor:default\" class = \"f"
280                         + (int) fontSize + "\" title = \"count: "
281                         + tempPair.getValue() + "\">" + tempPair.getKey()
282                         + "</span>");
283             }
284         }
285
286         // close each section of HTML
287         o.println("</p>");
288         o.println("</div>");
289         o.println("</body>");
290         o.println("</html>");
291     }
292
293     /**
294      * Main method.
295      *
296      * @param args
297      *            the command line arguments
298      */
299     public static void main(String[] args) {
300
301         // read user input
302         BufferedReader inUser = new BufferedReader(
303                 new InputStreamReader(System.in));
304
305         /*
306          * try opening files, vars created before try catch for scope
307          */
308
```

```
309         BufferedReader inF;
310         PrintWriter out;
311         String inFile = "";
312         String outFile = "";
313         try {
314             System.out.print("Please enter an input file: ");
315             inFile = inUser.readLine();
316             inF = new BufferedReader(new FileReader(inFile));
317 
318             System.out.print("Please enter an output file: ");
319             outFile = inUser.readLine();
320             out = new PrintWriter(new BufferedWriter(new FileWriter(outFile)));
321         } catch (IOException e) {
322             System.err.println("Error opening file.");
323             return;
324         }
325 
326         /*
327          * get N and check using parseInt. try catch for reading user input
328          */
329 
330         int n = 0;
331         try {
332             System.out.print("Please enter number of words for tag cloud: ");
333             n = Integer.parseInt(inUser.readLine());
334             Reporter.assertElseFatalError(n > -1, "Number cannot be negative.");
335         } catch (IOException e) {
336             System.err.println("Unable to read input.");
337         }
338 
339         /*
340          * create and fill set with desired separators, then use set to fill a
341          * map with unique words, while keeping count of word frequency
342          */
343 
344         HashSet<Character> separators = new HashSet<Character>();
345         fillSeparator(separators);
346 
347         HashMap<String, Integer> terms = new HashMap<String, Integer>();
348         try {
349             fillMap(inFile, terms, separators);
350         } catch (IOException e) {
351             System.err.println("Error reading file.");
352         }
353 
354         /*
355          * create a priority queue with descending numerical ordering and one
356          * with alphabetical order. empty map into numerical PQ then remove the
357          * first N items and add to alphabetical PQ
358          */
359 
360         Comparator<Entry<String, Integer>> numericalSort = new NumOrder();
361         PriorityQueue<Entry<String, Integer>> num = new PriorityQueue<>(
362                 numericalSort);
363 
364         for (Entry<String, Integer> p : terms.entrySet()) {
365             num.add(p);
366         }
367 
```

```
368         Comparator<Entry<String, Integer>> alphabetSort = new AlphaOrder();
369         PriorityQueue<Entry<String, Integer>> alpha = new PriorityQueue<>(
370                 alphabetSort);
371
372         double maxCount = 0; // used to calculate font size later
373
374         for (int i = 0; i < n && num.size() > 0; i++) {
375             Entry<String, Integer> p = num.remove();
376
377             if (p.getValue() > maxCount) {
378                 maxCount = p.getValue();
379             }
380
381             alpha.add(p);
382         }
383
384         // try printing HTML page
385         try {
386             PrintWriter o = new PrintWriter(
387                     new BufferedWriter(new FileWriter(outFile)));
388
389             printHTML(o, inFile, n, alpha, maxCount);
390             o.close();
391         } catch (IOException e) {
392             System.err.println("Error printing file.");
393         }
394
395         // try closing readers and writers
396         try {
397             inUser.close();
398             inF.close();
399             out.close();
400         } catch (IOException e) {
401             System.err.print("Error closing resources.");
402         }
403
404     }
405
406 }
407
```