```
 1 import java.io.BufferedReader;
15
16 /**
17  * Tag cloud generator program using normal java components.
18  *
19  * @author Bashir Ali and Kwasi Fosu
20  *
21  */
22 public final class TagCloudGeneratorWithJavaComponents {
23
24     /**
25      * No argument constructor--private to prevent
   instantiation.
26      */
27     private TagCloudGeneratorWithJavaComponents() {
28     }
29
30     /**
31      * font variables.
32      */
33     private static final int MAX_FONT = 48;
34     /**
35      * font variables.
36      */
37     private static final int MIN_FONT = 11;
38
39     /**
40      * set of separators.
41      */
42     private static final Set<Character> SEPARATORS =
   createSeparatorsSet();
43
44     /**
45      * @return separator set.
46      */
47     private static Set<Character> createSeparatorsSet() {
48         Set<Character> separators = new HashSet<Character>();
49         separators.add(':');
50         separators.add(';');
```

```
51          separators.add('|');
52          separators.add('~');
53          separators.add('!');
54          separators.add(' ');
55          separators.add('.');
56          separators.add(',');
57          separators.add('[');
58          separators.add(']');
59          separators.add('{');
60          separators.add('}');
61          separators.add('-');
62          separators.add('/');
63          separators.add('*');
64          separators.add('\'');
65          separators.add('\t');
66          separators.add('\n');
67          separators.add('\r');
68          return separators;
69      }
70
71      /**
72       * Comparator class used to sort strings alphabetically.
73       */
74      private static class SortKey
75              implements Comparator<Map.Entry<String, Integer>> {
76          @Override
77          public int compare(Map.Entry<String, Integer> o1,
78                  Map.Entry<String, Integer> o2) {
79              return o1.getKey().toLowerCase()
80                      .compareTo(o2.getKey().toLowerCase());
81          }
82      }
83
84      /**
85       * Comparator class used to sort strings alphabetically.
86       */
87      private static class SortValue
88              implements Comparator<Map.Entry<String, Integer>> {
89          @Override
```

```
 90          public int compare(Map.Entry<String, Integer> o1,
 91                  Map.Entry<String, Integer> o2) {
 92              return Integer.compare(o2.getValue(),
    o1.getValue());
 93          }
 94      }
 95
 96      /**
 97       * Returns the next word or separator from the text
    starting from the
 98       * specified position.
 99       *
100       * @param text
101       *            The text to get the next word or separator
    from
102       * @param position
103       *            The starting position to search for the next
    word or separator
104       * @param separators
105       *            A set of symbols considered as separators
106       * @return The next word or separator found in the text
    starting from the
107       *         given position.
108       */
109      private static String nextWordOrSeparator(String text, int
    position,
110              Set<Character> separators) {
111          assert text != null : "Violation of: text is not null";
112          assert separators != null : "Violation of: separators
    is not null";
113          assert 0 <= position : "Violation of: 0 <= position";
114          assert position <= text.length() : "Violation of:
    position < |text|";
115          int i = position;
116          int n = text.length();
117
118          // Find the end of the word or separator
119          if (!separators.contains(text.charAt(i))) {
120              while (i < n && !
```

```
    separators.contains(text.charAt(i))) {
121                 i++;
122             }
123         } else {
124             while (i < n &&
    separators.contains(text.charAt(i))) {
125                 i++;
126             }
127         }
128 
129         return text.substring(position, i);
130 
131     }
132 
133     /**
134      * Reads each word and adds it to a map with corresponding
    occurrences.
135      *
136      * @param input
137      *            The BufferedReader to read input from.
138      * @param m
139      *            The Map to populate with key-value pairs.
140      * @requires format Term and previous definition to be
    separated by a single
141      *           line.
142      */
143     public static void getKeysAndValues(BufferedReader input,
144             Map<String, Integer> m) throws IOException {
145         /*
146          * puts a word and its occurrences into a map as a pair
147          */
148         String line = input.readLine();
149         while (line != null) {
150             int index = 0;
151             while (index < line.length()) {
152                 String word = nextWordOrSeparator(line, index,
    SEPARATORS)
153                         .toLowerCase();
154                 index = index + word.length();
```

```
155                 int occurences = 1;
156                 if (m.containsKey(word)
157                         && !
    SEPARATORS.contains(word.charAt(0))) {
158                     occurences = m.get(word) + 1;
159                 }
160                 m.put(word, occurences);
161             }
162             line = input.readLine();
163         }
164     }
165 
166     /**
167      * generate output page in HTML containing a table with
    each word in
168      * alphabetic order and the number of times it occurs.
169      *
170      * @param outFile
171      *            file to print HTML lines in
172      * @param inFileName
173      *            name of the input file
174      * @param list
175      *            empty list to populate with ordered keys
176      * @param m
177      *            map containing word and how much it occurs
178      * @param n
179      *            amount of words to print
180      */
181     public static void generateHTMLPage(PrintWriter outFile,
    String inFileName,
182             List<Map.Entry<String, Integer>> list, Map<String,
    Integer> m,
183             int n) throws IOException {
184         outFile.println("<html>");
185         outFile.println("<head>");
186         outFile.println("<title>" + "Top " + n + " words in " +
    inFileName
187                 + "</title>");
188         outFile.println(
```

```
189                 "<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\">");
190         outFile.println(
191                 "<link href=\"tagcloud.css\" rel=\"stylesheet\"
    type=\"text/css\">");
192         outFile.println("</head>");
193         outFile.println("<body>");
194         outFile.println("<h2 style='color: blue;'>" + "Top " +
    n + " words in "
195                 + inFileName + "</h2>");
196         outFile.println("<div class=\"cdiv\">");
197         outFile.println("<p class=\"cbox\">");
198 
199         //sort the pairs in a list by value
200         Comparator<Map.Entry<String, Integer>> compareValue =
    new SortValue();
201         for (Map.Entry<String, Integer> entry : m.entrySet()) {
202             list.add(entry);
203         }
204         Collections.sort(list, compareValue);
205 
206         //remove all the pairs until list is same as n
207         while (list.size() > n) {
208             list.remove(list.size() - 1);
209         }
210 
211         //sort alphabetically and find max and min counts
212         Comparator<Map.Entry<String, Integer>> compareKey = new
    SortKey();
213         int minCount = Integer.MAX_VALUE;
214         int maxCount = 0;
215         for (Map.Entry<String, Integer> current : list) {
216             if (current.getValue() < minCount) {
217                 minCount = current.getValue();
218             }
219             if (current.getValue() > maxCount) {
220                 maxCount = current.getValue();
221             }
```

```
222         }
223         Collections.sort(list, compareKey);
224 
225         //print each word in the list
226         for (Map.Entry<String, Integer> current : list) {
227             int occurences = ((MAX_FONT - MIN_FONT)
228                     * (current.getValue() - minCount) /
    (maxCount - minCount))
229                     + MIN_FONT;
230             outFile.println("<span style=\"cursor:default\"
    class=\"f"
231                     + occurences + "\" title=\"count: " +
    current.getValue()
232                     + "\">" + current.getKey() + "</span>");
233         }
234         outFile.println("</p>");
235         outFile.println("</div>");
236         outFile.println("</body>");
237         outFile.println("</html>");
238     }
239 
240     /**
241      * Main method.
242      *
243      * @param args
244      *            the command line arguments
245      * @throws IOException
246      */
247     public static void main(String[] args) throws IOException {
248         Scanner in = new Scanner(System.in);
249 
250         System.out.print("Enter a text file: ");
251         String inputFile = in.nextLine();
252         System.out.print("Enter an output file: ");
253         String outputFile = in.nextLine();
254         int n = -1;
255         while (n < 0) {
256             try {
257                 System.out.print(
```

```
258                         "Enter how many words from the text
    file you want to print: ");
259                 n = Integer.parseInt(in.nextLine());
260                 if (n < 0) {
261                     System.out.println(
262                             "No negative numbers. Enter a
    positive number.");
263                 }
264             } catch (NumberFormatException e) {
265                 System.err.println("Error: Must type in an
    integer.");
266                 in.close();
267                 return;
268             }
269         }
270 
271         in.close();
272 
273         BufferedReader fileIn = null;
274         try {
275             fileIn = new BufferedReader(new
    FileReader(inputFile));
276         } catch (IOException e) {
277             System.err
278                     .println("Error: Program error while
    opening input file.");
279         }
280 
281         Map<String, Integer> map = new HashMap<>();
282         if (fileIn != null) {
283             try {
284                 getKeysAndValues(fileIn, map);
285             } catch (IOException e) {
286                 System.err.println(
287                         "Error: Program error while reading
    input file.");
288             }
289             try {
290                 fileIn.close();
```

```
291             } catch (IOException e) {
292                 System.err.println(
293                         "Error: Program error while closing
    input file.");
294             }
295         }
296 
297         PrintWriter fileOut = null;
298         try {
299             fileOut = new PrintWriter(new
    FileWriter(outputFile));
300         } catch (IOException e) {
301             System.err
302                     .println("Error: Program error while
    opening output file.");
303         }
304 
305         List<Map.Entry<String, Integer>> list = new
    ArrayList<>();
306         if (fileOut != null) {
307             try {
308                 generateHTMLPage(fileOut, inputFile, list, map,
    n);
309             } catch (IOException e) {
310                 System.err.println(
311                         "Error: Program error while reading
    input file.");
312             }
313             fileOut.close();
314         }
315     }
316 }
317 
```