

```

1 import java.util.Comparator;
14
15 /**
16  * This Java program generates a HTML file with tag cloud from a given input
17  * text.
18  *
19  * @author Zhuoyang Li + Xinci Ma
20  *
21  */
22 public final class TagCloudGenerator {
23
24     /**
25      * Set up the frequency for each word.
26      */
27     private static int minFrequency = 0;
28     /**
29      * Set up the frequency for each word.
30      */
31     private static int maxFrequency = 100;
32
33     /**
34      * Set up the font size for each word.
35      */
36     private static int minFontSize = 10;
37     /**
38      * Set up the font size for each word.
39      */
40     private static int maxFontSize = 50;
41
42     /**
43      * No argument constructor--private to prevent instantiation.
44      */
45     private TagCloudGenerator() {
46     }
47
48     /**
49      * Sort the words with alphabetical order.
50      */
51
52     private static Comparator<Map.Pair<String, Integer>> alphaOrder = new
    Comparator<Map.Pair<String, Integer>>() {
53         @Override
54         public int compare(Pair<String, Integer> o1, Pair<String, Integer> o2) {
55             // ignore case a = A
56             return o1.key().compareToIgnoreCase(o2.key());
57         }
58     };
59
60     /**
61      * Sort the words with frequency order Positive for descending order.
62      * Negative for ascending order
63      */
64
65     private static Comparator<Map.Pair<String, Integer>> frequencyOrder = new
    Comparator<Map.Pair<String, Integer>>() {
66         @Override
67         public int compare(Pair<String, Integer> o1, Pair<String, Integer> o2) {
68             return o2.value() - o1.value();
69         }
70     };
71
72     /**
73      * Returns the first "word" (maximal length string of characters not in
74      * {@code separators}) or "separator string" (maximal length string of
75      * characters in {@code separators}) in the given {@code text} starting at
76      * the given {@code position}.
77      *
78      *
79      * @param text
80      *         the {@code String} from which to get the word or separator
81      *         string
82      * @param position
83      *         the starting index
84      * @param separators
85      *         the {@code Set} of separator characters

```

```

86     * @return the first word or separator string found in {@code text} starting
87     *         at index {@code position}
88     * @requires <pre>
89     * { @code 0 <= position < |text| }
90     * </pre>
91     * @ensures <pre>
92     * { @code nextWordOrSeparator =
93     *   text[ position .. position + |nextWordOrSeparator| ] and
94     *   if elements(text[ position .. position + 1 ]) intersection separators = {}
95     *   then
96     *     elements(nextWordOrSeparator) intersection separators = {} and
97     *     ( position + |nextWordOrSeparator| = |text| or
98     *       elements(text[ position .. position + |nextWordOrSeparator| + 1 ])
99     *       intersection separators /= {} )
100    *   else
101    *     elements(nextWordOrSeparator) is subset of separators and
102    *     ( position + |nextWordOrSeparator| = |text| or
103    *       elements(text[ position .. position + |nextWordOrSeparator| + 1 ])
104    *       is not subset of separators ) }
105    * </pre>
106    */
107    private static String nextWordOrSeparator(String text, int position,
108        Set<Character> separators) {
109        assert text != null : "Violation of: text is not null";
110        assert separators != null : "Violation of: separators is not null";
111        assert 0 <= position : "Violation of: 0 <= position";
112        assert position < text.length() : "Violation of: position < |text|";
113
114        int i = position;
115
116        if (!separators.contains(text.charAt(position))) {
117            while (i < text.length() && !separators.contains(text.charAt(i))) {
118                i++;
119            }
120        } else {
121            while (i < text.length() && separators.contains(text.charAt(i))) {
122                i++;
123            }
124        }
125
126        return text.substring(position, i);
127    }
128
129    /**
130     * Generate the HTML file with tag cloud.
131     *
132     * @param file
133     *         the input file
134     * @param m
135     *         the map to store the frequency of each word
136     */
137    private static void countFrequency(SimpleReader file,
138        Map<String, Integer> m) {
139        assert file != null : "Violation of: file is not null";
140        assert file.isOpen() : "Violation of: file is open";
141        assert m.size() == 0 : "Violation of: m.size() = 0 (m is empty)";
142
143        String separators = "\\t/()?!.,<>;|[]{}~@#$$%-\\\"";
144        Set<Character> notIn = new Set1L<>();
145        for (int i = 0; i < separators.length(); i++) {
146            notIn.add(separators.charAt(i));
147        }
148        while (!file.atEOS()) {
149            String line = file.nextLine();
150            int i = 0;
151            while (i < line.length()) {
152                String word = nextWordOrSeparator(line, i, notIn);
153                if (!m.containsKey(word)) {
154                    m.add(word, 1);
155                } else {
156                    m.replaceValue(word, m.value(word) + 1);
157                }
158                i += word.length();
159            }
160        }

```

```

161     }
162 }
163
164 /**
165  * calculate the font size of the word.
166  *
167  * @param frequency
168  * @return the font size of the word
169  */
170 private static int wordSize(int frequency) {
171     int size = 0;
172     if (frequency == minFrequency) {
173         return minFontSize;
174     }
175     if (frequency == maxFrequency) {
176         return maxFontSize;
177     }
178
179     // Precompute these values to avoid recalculating them for each word
180     final int frequencyRange = maxFrequency - minFrequency;
181     final int fontSizeRange = maxFontSize - minFontSize;
182
183     // Linear interpolation between minFontSize and maxFontSize
184     return (int) Math
185         .floor(minFontSize + (double) (frequency - minFrequency)
186             / frequencyRange * fontSizeRange);
187 }
188
189 /**
190  * Sort the map with frequency order.
191  *
192  * @param m
193  *         the map to store the frequency of each word
194  * @param cloudSize
195  *         the size of the cloud
196  * @return the sorted map
197  */
198
199 public static Map<String, Integer> sortFrequency(Map<String, Integer> m,
200     int cloudSize) {
201
202     SortingMachine<Map.Pair<String, Integer>> sm = new SortingMachine1L<>(
203         frequencyOrder);
204     for (Map.Pair<String, Integer> pair : m) {
205         sm.add(pair);
206     }
207     sm.changeToExtractionMode();
208     Map<String, Integer> sorted = new Map1L<>();
209     int i = 0;
210     while (i < cloudSize && sm.size() > 0) {
211         Pair<String, Integer> pair = sm.removeFirst();
212         sorted.add(pair.key(), pair.value());
213         i++;
214     }
215     return sorted;
216 }
217
218 /**
219  * Sort the map with alphabetical order.
220  *
221  * @param m
222  *         the map to store the frequency of each word
223  * @return the sorted map
224  */
225 public static SortingMachine<Map.Pair<String, Integer>> generateAlphabeticSortedMap(
226     Map<String, Integer> m) {
227     SortingMachine<Map.Pair<String, Integer>> sm = new SortingMachine1L<>(
228         alphaOrder);
229     for (Map.Pair<String, Integer> pair : m) {
230         sm.add(pair);
231     }
232     sm.changeToExtractionMode();
233     return sm;
234 }
235

```

```

236  /**
237   * Generate the HTML file with tag cloud.
238   *
239   * @param file
240   *       the input file
241   * @param sm
242   *       the sorting machine
243   * @param cloudSize
244   *       the size of the cloud
245   * @param outName
246   *       the output file name
247   */
248
249  public static void printHTML(SimpleWriter outName, String file,
250                              int cloudSize, SortingMachine<Map.Pair<String, Integer>> sm) {
251      // Generate the Title
252      outName.println "<html>";
253      outName.println "<head>";
254      outName.println "<title>Tag Cloud Generator</title>";
255      outName.println
256          "<link href=\"tagcloud.css\" rel=\"stylesheet\" type=\"text/css\">";
257      outName.println "</head>";
258
259      // Generate the body
260      outName.println "<body>";
261      outName.println "<h2>Top " + cloudSize + " Words in " + file + "</h2>";
262      outName.println "<hr>";
263      outName.println "<div class=\"cdiv\">";
264      outName.println "<p class=\"cbox\">";
265      int i = 0;
266      while (i < cloudSize && sm.size() > 0) {
267          Map.Pair<String, Integer> pair = sm.removeFirst();
268          outName.println "<span style=\"cursor:default\" class=\"f"
269              + wordSizs(pair.value()) + "\" title=\"frequency: "
270              + pair.value() + "\">" + pair.key() + "</span>";
271          i++;
272      }
273      outName.println "</p>";
274      outName.println "</div>";
275      outName.println "</body>";
276      outName.println "</html>";
277  }
278
279  /**
280   * Main method.
281   *
282   * @param args
283   *       the command line arguments
284   */
285
286  public static void main(String[] args) {
287      SimpleReader in = new SimpleReader1L();
288      SimpleWriter out = new SimpleWriter1L();
289      out.println "Enter the file name done with txt: ";
290      String fileName = in.nextLine();
291      SimpleReader input = new SimpleReader1L(fileName);
292      out.println "Please enter the name of output file: ";
293      SimpleWriter output = new SimpleWriter1L("data/" + in.nextLine());
294      out.println "Please enter the size for the tag cloud: ";
295      int cloudSize = in.nextInt();
296      Map<String, Integer> str = new Map1L<String, Integer>();
297      countFrequency(input, str);
298      Map<String, Integer> topWords = sortFrequency(str, cloudSize);
299      SortingMachine<Map.Pair<String, Integer>> sort = generateAlphabeticSortedMap(
300          topWords);
301      printHTML(output, fileName, cloudSize, sort);
302
303      in.close();
304      out.close();
305      input.close();
306      output.close();
307  }
308
309
310

```