THE OHIO STATE UNIVERSITY

PROJECT 10: Tag Cloud Generator with Standard Java Components

Ansh Pachauri

Daniil Gofman

SW 2: Dev & Dsgn

Paolo Bucci

Yiyang Chen

Shivam Gupta

December 5, 2023

```
 1 import java.io.BufferedReader;
 2 import java.io.BufferedWriter;
 3 import java.io.FileReader;
 4 import java.io.FileWriter;
 5 import java.io.IOException;
 6 import java.io.InputStreamReader;
 7 import java.io.PrintWriter;
 8 import java.util.ArrayList;
 9 import java.util.Collections;
10 import java.util.Comparator;
11 import java.util.HashMap;
12 import java.util.HashSet;
13 import java.util.List;
14 import java.util.Map;
15 import java.util.Set;
16
17 /**
18  * Project 10.
19  *
20  * @author Daniil Gofman
21  * @author Ansh Pachauri
22  *
23  */
24 public final class TagCloudGeneratorJCF {
25
26     /**
27      * No argument constructor--private to prevent instantiation.
28      */
29     private TagCloudGeneratorJCF() {
30     }
31
32     /**
33      * The maximum font size.
34      */
35     private static final int FONT_NUMBER = 37;
36     /**
37      * The maximum font size.
38      */
39     private static final int MIN_FONT_SIZE = 11;
40
41     /**
42      * Compare {@code String}s in alphabetical order.
43      */
44     private static class StringLT
45             implements Comparator<Map.Entry<String, Integer>> {
46         @Override
47         public int compare(Map.Entry<String, Integer> str1,
48                 Map.Entry<String, Integer> str2) {
49
50             String s1 = str1.getKey();
```

```
51              String s2 = str2.getKey();
52
53              int compare = s1.compareToIgnoreCase(s2);
54
55              if (compare == 0) {
56                  compare = s1.compareTo(s2);
57              }
58              return compare;
59          }
60      }
61
62      /**
63       * Compare {@code Integer}i in decreasing order.
64       */
65      private static class IntegerLT
66              implements Comparator<Map.Entry<String, Integer>> {
67          @Override
68          public int compare(Map.Entry<String, Integer> int1,
69                  Map.Entry<String, Integer> int2) {
70
71              Integer i1 = int1.getValue();
72              Integer i2 = int2.getValue();
73
74              int compare = i2.compareTo(i1);
75
76              if (compare == 0) {
77                  compare = int1.getKey().compareTo(int2.getKey());
78              }
79              return compare;
80          }
81      }
82
83      /**
84       * Returns the first "word" (maximal length string of characters not in
85       * {@code separators}) or "separator string" (maximal length string of
86       * characters in {@code separators}) in the given {@code text} starting
    at
87       * the given {@code position}.
88       *
89       * @param text
90       *              the {@code String} from which to get the word or
    separator
91       *              string
92       * @param position
93       *              the starting index
94       * @param separators
95       *              the {@code Set} of separator characters
96       * @return the first word or separator string found in {@code text}
    starting
97       *          * at index {@code position}
```

```
 98      * @requires 0 <= position < |text|
 99      * @ensures <pre>
100      * nextWordOrSeparator =
101      * text[position, position + |nextWordOrSeparator|) and
102      * if entries(text[position, position + 1)) intersection separators =
    {} * then
103      * entries(nextWordOrSeparator) intersection separators = {} and
104      * (position + |nextWordOrSeparator| = |text| or
105      * entries(text[position, position + |nextWordOrSeparator| + 1))
106      * intersection separators /= {})
107      * else
108      * entries(nextWordOrSeparator) is subset of separators and
109      * (position + |nextWordOrSeparator| = |text| or
110      * entries(text[position, position + |nextWordOrSeparator| + 1))
111      * is not subset of separators)
112      * </pre>
113      */
114     private static String nextWordOrSeparator(String text, int position,
115             Set<Character> separators) {
116
117         int end = position;
118         if (!separators.contains(text.charAt(position))) {
119             //find length of word
120             while (end < text.length()
121                     && !separators.contains(text.charAt(end))) {
122                 end++;
123             }
124         } else {
125             //find length of separator
126             while (end < text.length()
127                     && separators.contains(text.charAt(end))) {
128                 end++;
129             }
130         }
131         return text.substring(position, end);
132     }
133
134     /**
135      * Generates the set of characters in the given {@code String} into the
136      * given {@code Set}.
137      *
138      * @param str
139      *            the given {@code String}
140      * @param strSet
141      *            the {@code Set} to be replaced
142      * @replaces strSet
143      * @ensures strSet = entries(str)
144      */
145     private static void generateElements(String str, Set<Character> strSet)
    {
```

```
146         assert str != null : "Violations of: str is not null";
147         assert strSet != null : "Violation of: strSet is not null";
148
149         for (int i = 0; i < str.length(); i++) {
150             char c = str.charAt(i);
151             if (!strSet.contains(c)) {
152                 strSet.add(c);
153             }
154         }
155     }
156
157     /**
158      * Takes the file and makes a map with each key as the word in lower
    case
159      * and the value as the number of occurrences of that word.
160      *
161      * @param input
162      *            the BufferedReader file
163      * @ensures all words from the file will be in the map with the count
    of
164      *          each word
165      * @return Map<String, Integer> of words of the file and their counts
166      * @throws IOException
167      */
168     private static Map<String, Integer> fileToMap(BufferedReader input)
169             throws IOException {
170         assert input != null : "Violation of: inFile is not null";
171
172         Map<String, Integer> result = new HashMap<>();
173         // define separators for the words
174         final String separators = "'., ()-_?\"/!@#$%^&*\t1234567890:"
175                 + ";[]{}+=~`><";
176         Set<Character> separatorSet = new HashSet<>();
177         generateElements(separators, separatorSet);
178         try {
179             // read file, separate words and compile all words into the map
180             String line = input.readLine();
181             while (line != null) {
182                 int i = 0;
183                 while (i < line.length()) {
184                     String word = nextWordOrSeparator(line, i,
    separatorSet)
185                             .toLowerCase();
186                     boolean isWord = true;
187                     for (int j = 0; j < word.length(); j++) {
188                         char c = word.charAt(j);
189                         if (separatorSet.contains(c)) {
190                             isWord = false;
191                         }
192                     }
```

```
193                     if (isWord) {
194                         //add word to map or increase the count
195                         if (result.containsKey(word)) {
196                             int count = result.get(word);
197                             result.replace(word, count + 1);
198                         } else {
199                             result.put(word, 1);
200                         }
201                     }
202                     i += word.length();
203                 }
204                 line = input.readLine();
205             }
206         } catch (IOException e) {
207             System.err.println("Error reading data from the file");
208         }
209 
210         return result;
211     }
212 
213     /**
214      * Takes a map and first sorts the it with sortingMachine by
    occurrences of
215      * each word in decreasing order. Then makes a second sortingMachine,
    and
216      * sorts the map alphabetically. numDisplay is the number of words to
    be put
217      * in first sortingMachine.
218      *
219      * @param map
220      *             map of all of the words and their counts
221      * @param numDisplay
222      *             the amount of words that will be in the first
    sortingMachine
223      * @requires map is not null
224      * @ensures all Map.Entrys is sorted in decreasing order by their
    values
225      * @return SortingMachine<Map.Entry<String, Integer>> of words and
    their
226      *         counts in alphabetical order
227      */
228     public static List<Map.Entry<String, Integer>> mapToListAlpha(
229             Map<String, Integer> map, Integer numDisplay) {
230         assert map != null : "Violation of: map is not null";
231 
232         // Comparator for sorting by count
233         Comparator<Map.Entry<String, Integer>> countSort = new IntegerLT();
234 
235         // Comparator for sorting alphabetically by keys
236         Comparator<Map.Entry<String, Integer>> alphaSort = new StringLT();
```

```
237
238         // Create a List to store map entries
239         List<Map.Entry<String, Integer>> listInt = new ArrayList<>();
240
241         // Add all entries from the map to the list
242         listInt.addAll(map.entrySet());
243
244         // Sort the list by count using the countSort comparator
245         Collections.sort(listInt, countSort);
246
247         // Create a sublist containing the specified number of entries to
    display
248         List<Map.Entry<String, Integer>> listAlpha = listInt.subList(0,
249                 numDisplay);
250
251         // Sort the sublist alphabetically using the alphaSort comparator
252         Collections.sort(listAlpha, alphaSort);
253
254         // Return the final sorted list
255         return listAlpha;
256     }
257
258     /**
259      * Output the header of the file.
260      *
261      * @param out
262      *            the output file.
263      *
264      * @param inFile
265      *            the input file.
266      *
267      * @param numWords
268      *            number of words in the HTML file.
269      */
270     private static void header(PrintWriter out, String inFile, int
    numWords) {
271         assert out != null : "Violation of: out is not null";
272         assert inFile != null : "Violation of: inFile is not null";
273         /*
274          * Output the index header HTML text.
275          */
276         out.println("<html>");
277         out.println("<head>");
278         out.println(
279                 "<title>Top " + numWords + " words in " + inFile +
    "</title>");
280         out.println("<link href=\"http://web.cse.ohio-state.edu/software"
281                 + "/2231/web-sw2/assignments/projects/tag-cloud-
    generator/data/"
282                 + "tagcloud.css\" rel=\"stylesheet\" type=\"text/css\">");
```

```
283         out.println(
284                 "<link href=\"tagcloud.css\" rel=\"stylesheet\" type=
    \"text/css\">");
285         out.println("</head>");
286         out.println("<body>");
287         out.println("<h2>Top " + numWords + " words in " + inFile +
    "</h2>");
288         out.println("<hr>");
289         out.println("<div class = \"cdiv\">");
290         out.println("<p class = \"cbox\">");
291     }
292 
293     /**
294      * Output the ending tags in the generated HTML file.
295      *
296      * @param output
297      *            the output file.
298      */
299     private static void footer(PrintWriter output) {
300         assert output != null : "Violation of: out is not null";
301         //output the closing tags
302         output.println("</p>");
303         output.println("</div>");
304         output.println("</body>");
305         output.println("</html>");
306     }
307 
308     /**
309      * Method to generate the main body of 'tag cloud'.
310      *
311      * @param output
312      * @param alphaSort
313      */
314     private static void generateList(PrintWriter output,
315             List<Map.Entry<String, Integer>> alphaSort) {
316         // Initialize a variable to keep track of the maximum count
317         int countMax = 0;
318 
319         // Define a constant for the minimum value
320         final int min = 100;
321 
322         // Initialize a variable to keep track of the minimum count
323         int countMin = min;
324 
325         // Iterate through the entries in the 'alphaSort' map
326         for (Map.Entry<String, Integer> i : alphaSort) {
327             /*
328              * Check if the current entry's value is greater than the
    current
329              * maximum count
```

```
330             */
331            if (i.getValue() > countMax) {
332                // Update the maximum count if the condition is met
333                countMax = i.getValue();
334            }
335
336            // Check if the current entry's value is less than the current
   minimum count
337            if (i.getValue() < countMin) {
338                // Update the minimum count if the condition is met
339                countMin = i.getValue();
340            }
341        }
342
343        // Calculate the difference between the maximum and minimum counts
344        int difference = countMax - countMin;
345
346        // Calculate an intermediate difference
347        int interDifference = difference / FONT_NUMBER;
348
349        // If the intermediate difference is zero, set it to 1 to avoid
   division by zero
350        if (interDifference == 0) {
351            interDifference = 1;
352        }
353
354        // Iterate through entries in alphaSort while it's not empty
355        while (alphaSort.size() > 0) {
356            // Remove and retrieve the first entry
357            Map.Entry<String, Integer> temp = alphaSort.remove(0);
358
359            // Calculate the font size based on the entry's value
360            int font = ((temp.getValue() - countMin) / interDifference)
361                    + MIN_FONT_SIZE;
362
363            // Print HTML code with the calculated font size and additional
   information
364            output.println("<span style=\"cursor:default\" class=\"f" +
   font
365                    + "\"title=\"count:" + temp.getValue() + "\">"
366                    + temp.getKey() + "</span>");
367        }
368    }
369
370    /**
371     * Main method.
372     *
373     * @param args
374     *            the command line arguments
375     */
```

```
376     public static void main(String[] args) {
377         try (BufferedReader in = new BufferedReader(
378                 new InputStreamReader(System.in))) {
379             System.out.print("Enter the name of the input text file: ");
380             String inputFile = in.readLine();
381 
382             try (BufferedReader input = new BufferedReader(
383                     new FileReader(inputFile))) {
384                 System.out.print("Enter the name of the output file: ");
385                 String outputFile = in.readLine();
386 
387                 try (PrintWriter output = new PrintWriter(
388                         new BufferedWriter(new FileWriter(outputFile)))) {
389                     System.out.print("Enter number of words to be included
    "
390                             + "in the generated tag cloud: ");
391                     String wordNumStr = in.readLine();
392 
393                     if (wordNumStr != null) {
394                         int wordNum = Integer.parseInt(wordNumStr);
395                         // Output header of an html-file
396                         header(output, inputFile, wordNum);
397                         // Output body of an html-file
398                         Map<String, Integer> map = fileToMap(input);
399                         List<Map.Entry<String, Integer>> alphaSort =
    mapToListAlpha(
400                                 map, wordNum);
401                         generateList(output, alphaSort);
402                         // Output footer of an html-file
403                         footer(output);
404                         System.out.println("Program completed");
405                     }
406                 }
407             } catch (IOException e) {
408                 System.err.printf("Error %s occurred", e.getMessage());
409             }
410 
411         } catch (IOException e) {
412             System.err.printf("Error %s occurred", e.getMessage());
413         }
414     }
415 
416 }
417 
```