```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

/**
 * This program counts word occurrences in a given text file and outputs an HTML
 * document with a table of the words and counts listed in alphabetical order.
 *
 * @author Hongda Lin, Net Zhang
 *
 */
public final class TagCloud {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private TagCloud() {
    }

    /**
     * Compare {@code String}s in Number of appearance order.
     */
    private static class CountComparator
            implements Comparator<Map.Entry<String, Integer>> {

        @Override
        public int compare(Map.Entry<String, Integer> o1,
                Map.Entry<String, Integer> o2) {
            int result = o1.getValue().compareTo(o2.getValue());
            if (result == 0) {
                result = o1.getKey().toLowerCase()
                        .compareTo(o1.getKey().toLowerCase());
            }
            return result;
        }
    }

    /**
     * Compare {@code String}s in Alphabetic order.
     */
    private static class AlphabetComparator
            implements Comparator<Map.Entry<String, Integer>> {

        @Override
        public int compare(Map.Entry<String, Integer> o1,
                Map.Entry<String, Integer> o2) {
            String s1 = o1.getKey().toLowerCase();
            String s2 = o2.getKey().toLowerCase();
```

```
63              int result = s1.compareTo(s2);
64              if (result == 0) {
65                  result = o1.getValue().compareTo(o2.getValue());
66              }
67              return result;
68          }
69      }
70
71      /**
72       * Generates the set of characters in the given {@code String} into the
73       * given {@code Set}.
74       *
75       * @param str
76       *            the given {@code String}
77       * @param strSet
78       *            the {@code Set} to be replaced
79       * @replaces strSet
80       * @ensures strSet = entries(str)
81       */
82      private static void generateElements(String str, Set<Character> strSet) {
83          assert str != null : "Violation of: str is not null";
84          assert strSet != null : "Violation of: strSet is not null";
85
86          for (int i = 0; i < str.length(); ++i) {
87              char temp = str.charAt(i);
88              if (!strSet.contains(temp)) {
89                  strSet.add(temp);
90              }
91          }
92
93      }
94
95      /**
96       * Returns the first "word" (maximal length string of characters not in
97       * {@code separators}) or "separator string" (maximal length string of
98       * characters in {@code separators}) in the given {@code text} starting at
99       * the given {@code position}.
100      *
101      * @param text
102      *            the {@code String} from which to get the word or separator
103      *            string
104      * @param position
105      *            the starting index
106      * @param separators
107      *            the {@code Set} of separator characters
108      * @return the first word or separator string found in {@code text} starting
109      *         at index {@code position}
110      * @requires 0 <= position < |text|
111      * @ensures <pre>
112      * nextWordOrSeparator =
113      *   text[position, position + |nextWordOrSeparator|)  and
114      * if entries(text[position, position + 1)) intersection separators = {}
115      * then
116      *   entries(nextWordOrSeparator) intersection separators = {}  and
117      *   (position + |nextWordOrSeparator| = |text|  or
118      *    entries(text[position, position + |nextWordOrSeparator| + 1))
119      *      intersection separators /= {})
120      * else
121      *   entries(nextWordOrSeparator) is subset of separators  and
122      *   (position + |nextWordOrSeparator| = |text|  or
123      *    entries(text[position, position + |nextWordOrSeparator| + 1))
124      *      is not subset of separators)
```

```
     * </pre>
     */
    private static String nextWordOrSeparator(String text, int position,
            Set<Character> separators) {
        assert text != null : "Violation of: text is not null";
        assert separators != null : "Violation of: separators is not null";
        assert 0 <= position : "Violation of: 0 <= position";
        assert position < text.length() : "Violation of: position < |text|";

        StringBuilder word = new StringBuilder();
        if (separators.contains(text.charAt(position))) {
            int stopPosition = position;
            while (stopPosition < text.length()
                    && separators.contains(text.charAt(stopPosition))) {
                word.append(text.charAt(stopPosition));
                stopPosition++;
            }
        } else {
            int stopPosition = position;
            while (stopPosition < text.length()
                    && !separators.contains(text.charAt(stopPosition))) {
                word.append(text.charAt(stopPosition));
                stopPosition++;
            }
        }
        return word.toString();
    }

    /**
     * Read all of the words from the file, count their appearances and store
     * them correspondingly in a map. Meanwhile store the words in a
     * alphabetical order in a queue.
     *
     * @param file
     *            the input stream
     * @param wordMap
     *            the map that stores the words and their number of appearances
     *            in the file
     * @updates file
     * @replaces words, wordMap
     * @requires file.is_open
     * @ensures file.is_open and file.content is null and wordMap contains all
     *          of the words as keys, and its corresponding number of
     *          appearances as value. The queue words contains the unique words
     *          and store them in a alphabetic order.
     */
    private static void getWords(BufferedReader file,
            Map<String, Integer> wordMap) {

        wordMap.clear();
        /*
         * Define separator characters
         */
        final String separatorStr = " \t\n\r,-.!?[]';:/()0123456789_\"*`";
        Set<Character> separatorSet = new HashSet<>();
        generateElements(separatorStr, separatorSet);
        /*
         * Read the file line by line
         */
        try {
            String tempLine = file.readLine();
            while (tempLine != null) {
```

```
187                 int position = 0;
188                 while (position < tempLine.length()) {
189                     String token = nextWordOrSeparator(tempLine, position,
190                             separatorSet);
191                     // All to lower case
192                     token = token.toLowerCase();
193                     // Check whether the token is a word or separator
194                     if (!separatorSet.contains(token.charAt(0))) {
195                         // If the word map already contains token
196                         if (wordMap.containsKey(token)) {
197                             Integer tempValue = wordMap.remove(token);
198                             wordMap.put(token, tempValue + 1);
199                         } else {
200                             wordMap.put(token, 1);
201                         }
202                     }
203                     position += token.length();
204                 }
205 
206                 tempLine = file.readLine();
207             }
208         } catch (IOException e) {
209             System.err.println("Error reading from file");
210         }
211 
212     }
213 
214     /**
215      * Sort the words by their number of appearance in the file.
216      *
217      * @param wordMap
218      *            A map that contains all of the words in a file and stores the
219      *            unique words as keys, their number of appearances as values
220      * @clears wordMap
221      * @return A sorting machine that sort all of the words by their number of
222      *         appearance
223      * @ensures The returned sorting machine is in Extraction mode
224      */
225     private static List<Map.Entry<String, Integer>> sortWordsCount(
226             Map<String, Integer> wordMap) {
227 
228         Comparator<Map.Entry<String, Integer>> cc = new CountComparator();
229         List<Map.Entry<String, Integer>> stCount = new LinkedList<>();
230         Set<Map.Entry<String, Integer>> wordMapView = wordMap.entrySet();
231         Iterator<Map.Entry<String, Integer>> iter = wordMapView.iterator();
232         while (iter.hasNext()) {
233             Map.Entry<String, Integer> tempEntry = iter.next();
234             stCount.add(tempEntry);
235         }
236         // Sort the List based on the number of appearance
237         Collections.sort(stCount, cc);
238 
239         return stCount;
240     }
241 
242     /**
243      * Calculate the scaled font size basing on the counts of the word.
244      *
245      * @param fontMax
246      *            The font size for the words that has the largest count
247      * @param fontMin
248      *            The font size for the words that has the minimum count
```

```
249      * @param count
250      *            The number of appearance of the word
251      * @param countMin
252      *            The minimum number of appearance in the selected words
253      * @param countMax
254      *            The maximum number of appearance in the selected words
255      * @return The scaled font size
256      * @ensures scaleSize is the scaled font size of the selected words
257      *
258      */
259     private static int scaleSize(int fontMax, int fontMin, int count,
260             int countMin, int countMax) {
261
262         int scaled;
263         if (countMin == countMax) {
264             scaled = 1;
265         } else {
266             scaled = (fontMax - fontMin) * (count - countMin)
267                     / (countMax - countMin) + fontMin;
268         }
269         return scaled;
270     }
271
272     /**
273      * Sort the top n words in a alphabetical order.
274      *
275      * @param fontSize
276      *            A map that used to store the words we want to display and
277      *            their scaled font size
278      * @param n
279      *            The top number of words that we want to display
280      * @param fontMax
281      *            The font size for the words that has the largest count
282      * @param fontMin
283      *            The font size for the words that has the minimum count
284      * @param stCount
285      *            A sorting machine that sorts all of the words in the file by
286      *            their number of appearance
287      * @replaces fontSize
288      * @updates stCount
289      * @requires {@code stCount} is in extraction mode {@code n} is in between 1
290      *           and the size of {@code stCount}
291      * @return A sorting machine that sorts the top {@code n} words through
292      *         alphabetical order
293      * @ensures {@code sortWordsAlphabet} sorts the top {@code n} words through
294      *          alphabetical order. {@code sortWordsAlphabet} is in extraction
295      *          mode. Meanwhile {@code fontSize} stores the top {@code n} words
296      *          as its keys and their scaled display font size as its
297      *          corresponding value
298      */
299     private static List<Map.Entry<String, Integer>> sortWordsAlphabet(
300             Map<String, Integer> fontSize, int n, int fontMax, int fontMin,
301             List<Map.Entry<String, Integer>> stCount) {
302         // replace fontSize
303         // check stCount is in extraction mode
304         fontSize.clear();
305
306         Comparator<Map.Entry<String, Integer>> ac = new AlphabetComparator();
307         List<Map.Entry<String, Integer>> stAlphabet = new LinkedList<>();
308
309         int countMin = 0;
310         int countMax = 0;
```

```
311         for (int i = 0; i < n; i++) {
312             int last = stCount.size() - 1;
313             Map.Entry<String, Integer> tempEntry = stCount.remove(last);
314             if (i == 0) {
315                 countMax = tempEntry.getValue();
316             } else if (i == n - 1) {
317                 countMin = tempEntry.getValue();
318             }
319             stAlphabet.add(tempEntry);
320         }
321         // Sort the List based on the alphabetical order
322         Collections.sort(stAlphabet, ac);
323 
324         Iterator<Map.Entry<String, Integer>> iter = stAlphabet.iterator();
325         while (iter.hasNext()) {
326             Map.Entry<String, Integer> tempEntry = iter.next();
327             int scaledSize = scaleSize(fontMax, fontMin, tempEntry.getValue(),
328                     countMin, countMax);
329             fontSize.put(tempEntry.getKey(), scaledSize);
330         }
331 
332         return stAlphabet;
333     }
334 
335     /**
336      * Output the HTML file that contains a tag cloud.
337      *
338      * @param out
339      *            the output stream
340      * @param fileName
341      *            the output file name
342      * @param stAlphabet
343      *            A sorting machine that sorts the top n words through
344      *            alphabetical order
345      * @param fontSize
346      *            A map that used to store the words we want to display and
347      *            their scaled font size
348      * @updates out.content, stAlphabet
349      * @requires out.is_open
350      * @ensures out.content has HTML lines that generate a tag cloud where words
351      *          are displayed in alphabetical orders, and their font size is
352      *          scaled basing on their number of appearances in the text file.
353      *          And the size {@code stAlphabet} is zero.
354      */
355     private static void outputTagCloud(PrintWriter out, String fileName,
356             List<Map.Entry<String, Integer>> stAlphabet,
357             Map<String, Integer> fontSize) {
358 
359         final String cssHref = "http://web.cse.ohio-state.edu/software/"
360                 + "2231/web-sw2/assignments/projects/"
361                 + "tag-cloud-generator/data/tagcloud.css";
362         int topN = stAlphabet.size();
363 
364         out.println("<html>");
365         // Header -----------------------------------------------------------
366         out.println("<head>");
367         out.println(
368                 "<title>Top " + topN + " words in " + fileName + "</title>");
369         out.println("<link href=\"" + cssHref + "\""
370                 + " rel=\"stylesheet\" type = \"text/css\">");
371         out.println("</head>");
372 
```

```
373         // Body ----------------------------------------------------------------
374 
375         out.println("<body>");
376         out.println("<h2>Top " + topN + " words in " + fileName + "</h2>");
377         out.println("<hr>");
378         out.println("<div class=\"cdiv\">");
379         out.println("<p class=\"cbox\">");
380 
381         while (stAlphabet.size() > 0) {
382             Map.Entry<String, Integer> tempEntry = stAlphabet.remove(0);
383             String word = tempEntry.getKey();
384             String size = "f" + fontSize.get(tempEntry.getKey());
385             int count = tempEntry.getValue();
386 
387             // Size
388             out.print("<span style=\"cursor:default\" class=\"" + size + "\" ");
389             // Count
390             out.print("title=\"count: " + count + "\">");
391             // Word
392             out.println(word + "</span>");
393 
394         }
395 
396         out.println("</p>");
397         out.println("</div>");
398         out.println("</body>");
399         out.println("</html>");
400     }
401 
402     /**
403      * Main method.
404      *
405      * @param args
406      *            the command line arguments
407      * @throws IOException
408      */
409     public static void main(String[] args) {
410 
411         final int fontMax = 48;
412         final int fontMin = 11;
413 
414         // Read the name of the text file ===================================
415         System.out.print("Enter the name of file location: ");
416         BufferedReader input = new BufferedReader(
417                 new InputStreamReader(System.in));
418         String inputLocation;
419         try {
420             inputLocation = input.readLine();
421         } catch (IOException e) {
422             System.err.println("Error reading stream from system input " + e);
423             return;
424         }
425 
426         // Read the name of the output tag cloud ============================
427         System.out.print("Enter the name of the output html file location: ");
428         String outputLocation;
429         try {
430             outputLocation = input.readLine();
431         } catch (IOException e) {
432             System.err.println("Error reading stream from system input " + e);
433             return;
434         }
```

```
435
436         // Open the text file ================================================
437         BufferedReader fileInput;
438         try {
439             fileInput = new BufferedReader(new FileReader(inputLocation));
440         } catch (IOException e) {
441             System.err.println("Error opening file from file location " + e);
442             return;
443         }
444
445         // Open an output file for the tag cloud ==============================
446         PrintWriter fileOutput;
447         try {
448             fileOutput = new PrintWriter(
449                     new BufferedWriter(new FileWriter(outputLocation)));
450         } catch (IOException e1) {
451             System.err.println("Error creating file in the location " + e1);
452             // close the opened text file
453             try {
454                 fileInput.close();
455             } catch (IOException e) {
456                 System.err.println("Error closing file" + e);
457                 return;
458             }
459             return;
460         }
461
462         // Read the text file to extract all the words ======================
463         Map<String, Integer> wordMap = new HashMap<String, Integer>();
464         Map<String, Integer> fontSize = new HashMap<String, Integer>();
465
466         getWords(fileInput, wordMap);
467
468         int maxNumber = wordMap.size();
469
470         // Ask the user for the number of words they want on the tag cloud ===
471         System.out.print(
472                 "Please enter the number of words to be included in tag cloud"
473                         + "[1, " + maxNumber + "]: ");
474
475         String inputNum;
476         try {
477             inputNum = input.readLine();
478         } catch (IOException e) {
479             inputNum = null;
480         }
481         int userNum;
482         try {
483             userNum = Integer.parseInt(inputNum);
484             while (userNum <= 1 || userNum > maxNumber) {
485                 System.out.print(
486                         "Please enter the number of words to be included in tag cloud"
487                                 + "[1, " + maxNumber + "]: ");
488                 try {
489                     inputNum = input.readLine();
490                 } catch (IOException e) {
491                     inputNum = null;
492                 }
493             }
494         } catch (NumberFormatException e) {
495             System.err.println("Error reading stream from system input " + e);
496             fileOutput.close();
```

```
497             return;
498         }
499 
500         // Sort the words ======================================================
501         List<Entry<String, Integer>> stCount = sortWordsCount(wordMap);
502         List<Entry<String, Integer>> stAlphabet = sortWordsAlphabet(fontSize,
503                 userNum, fontMax, fontMin, stCount);
504 
505         // Generate tag cloud HTML ============================================
506         outputTagCloud(fileOutput, inputLocation, stAlphabet, fontSize);
507 
508         // Don't forget to close ==============================================
509         try {
510             input.close();
511             fileInput.close();
512             fileOutput.close();
513         } catch (IOException e) {
514             System.err.println("Error closing file" + e);
515             return;
516         }
517     }
518 }
519 
```