

```

1 import java.util.Comparator;
2
3 import components.map.Map;
4 import components.map.Map1L;
5 import components.queue.Queue;
6 import components.queue.Queue1L;
7 import components.set.Set;
8 import components.set.Set1L;
9 import components.simplereader.SimpleReader;
10 import components.simplereader.SimpleReader1L;
11 import components.simplewriter.SimpleWriter;
12 import components.simplewriter.SimpleWriter1L;
13
14 /**
15  * Creates a well formatted HTML page for a glossary text file.
16  *
17  * @author Joe Fong
18  *
19  */
20 public final class Glossary {
21
22     /**
23      * Private constructor so this utility class cannot be instantiated.
24      */
25     private Glossary() {
26
27     }
28
29     /**
30      * Compare {@code String}s in lexicographic order.
31      */
32     private static class OrderAlpha implements Comparator<String> {
33         @Override
34         public int compare(String str1, String str2) {
35             return str1.compareTo(str2);
36         }
37     }
38
39     /**
40      * Searches text file and returns terms and definitions in a queue while
41      * updating the {@code map}
42      *
43      * @param in
44      *         the input stream
45      * @param termsAndDefinitions
46      *         Map connecting each term to its definition
47      * @updates termsAndDefinitions
48      * @return queue of terms read from {@code in}
49      * @requires input.is_open
50      * @ensures <pre>
51      * input.is_open and input.content = <> and
52      * termsAndDefinitions = #termsAndDefinitions *
53      * [term keys, definition values]
54      * </pre>
55      */
56     public static Queue<String> collectTerms(SimpleReader in,
57         Map<String, String> termsAndDefinitions) {
58
59         Queue<String> terms = new Queue1L<String>();

```

```

60     String str = "", definition = "", tempDefinition = "something";
61
62     /*
63     * while loop reads in each line
64     */
65     while (!in.atEOS()) {
66         /*
67         * term added to queue
68         */
69         str = in.nextLine();
70         terms.enqueue(str);
71         /*
72         * Definition added by line until blank space using tempDef. First
73         * line read outside of loop to avoid extra spaces when the
74         * definition has multiple lines
75         */
76         tempDefinition = in.nextLine();
77         definition = definition.concat(tempDefinition);
78         while (!tempDefinition.equals("") && !in.atEOS()) {
79             tempDefinition = in.nextLine();
80             if (!tempDefinition.equals("") && !in.atEOS()) {
81                 definition = definition.concat(" ");
82             }
83             definition = definition.concat(tempDefinition);
84
85         }
86         /*
87         * term and definition mapped together
88         */
89         termsAndDefinitions.add(str, definition);
90         /*
91         * resets variables for loop
92         */
93         definition = "";
94         tempDefinition = "something";
95     }
96     /*
97     * Returns the queue containing the terms
98     */
99     return terms;
100 }
101
102 /**
103  * Outputs the header and body for the index file containing the terms and
104  * their links.
105  *
106  * <html> <head> <title>index.html</title> </head> <body>
107  * <h1>Glossary</h1>
108  * <h3>Index</h3>
109  * <ul>
110  * list items
111  * </ul>
112  * </body> </html>
113  *
114  * @param out
115  *         the output stream
116  * @param terms
117  *         the terms of the glossary
118  * @updates out.content

```

```

119     * @requires out.is_open
120     * @ensures out.content = #out.content * [the HTML file]
121     */
122     public static void outputIndex(SimpleWriter out, Queue<String> terms) {
123         /*
124          * creates variables for length and the out put terms
125          */
126         int length = terms.length(), i = 0;
127         String term = "";
128         /*
129          * outputs html header
130          */
131         out.println("<html>");
132         out.println("<head>");
133         out.println("<title>Glossary</title>");
134         out.println("</head>");
135         out.println("<body>");
136         out.println("<h1>Glossary</h1>");
137         out.println("<h3>Index</h3>");
138         out.println("<ul>");
139         /*
140          * while loop used to print out list of linked terms
141          */
142         while (i < length) {
143             term = terms.front();
144             terms.rotate(1);
145             out.println("<li>");
146             out.println("<a href=\"\" + term + \".html\" + \">\" + term + "</a>");
147             out.println("</li>");
148             i++;
149         }
150         out.println("</ul>");
151         out.println("</body>");
152         out.println("</html>");
153     }
154
155     /**
156     * Outputs the header and body for term files containing the term and its
157     * definition.
158     *
159     * <html> <head> <title>index.html</title> </head> <body>
160     * <h1>Glossary</h1>
161     * <h3>Index</h3>
162     * <p>
163     * definition
164     * </p>
165     * <p>
166     * return to index
167     * </p>
168     * </body> </html>
169     *
170     * @param out
171     *         the output stream
172     * @param termsMap
173     *         the terms mapped to their definition
174     * @param term
175     *         the term of the glossary
176     * @param separators
177     *         the set of separating chars

```

```

178     * @param termSet
179     *         the set of terms
180     * @updates out.content
181     * @requires out.is_open
182     * @ensures out.content = #out.content * [the HTML file]
183     */
184     public static void outputTermHTML(SimpleWriter out,
185         Map<String, String> termsMap, String term,
186         Set<Character> separators, Set<String> termSet) {
187         /*
188          * outputs html header
189          */
190         out.println("<html>");
191         out.println("<head>");
192         out.println("<title>" + term + "</title>");
193         out.println("</head>");
194         out.println("<body>");
195         out.println("<h2><b><i><font color=\"red\">" + term
196             + "</font></i></b></h2>");
197         out.println("<p>");
198         /*
199          * sets variables for definition and the searched word
200          */
201         String definition = termsMap.value(term), word = "";
202         /*
203          * While loop runs while i < the length of the definition. Loop searches
204          * for words in the definition and prints out what is returned. However
205          * if a word is another term in the glossary it is linked.
206          */
207         int length = definition.length();
208         int i = 0;
209         while (i < length) {
210             word = nextWordOrSeparator(definition, i, separators);
211             i += word.length();
212             if (termSet.contains(word)) {
213                 out.print(
214                     "<a href=\"" + word + ".html" + "\">" + word + "</a>");
215             } else {
216                 out.print(word);
217             }
218         }
219         out.println();
220         out.println("</p>");
221         /*
222          * prints out return to index with link at the bottom of the page
223          */
224         out.println("<p>");
225         out.println("Return to <a href=\"index.html\">index</a>");
226         out.println("</p>");
227         out.println("</body>");
228         out.println("</html>");
229     }
230
231     /**
232      * Generates the set of terms in the given {@code Queue} into the given
233      * {@code Set}.
234      *
235      * @param terms
236      *         the given {@code Queue}

```

```
237     * @param termSet
238     *         the {@code Set} to be replaced
239     * @replaces termSet
240     * @ensures termSet = entries(queue)
241     */
242     public static void queueToSet(Queue<String> terms, Set<String> termSet) {
243         /*
244          * for loop adds terms from the queue to the set for the length of the
245          * queue
246          */
247         int length = terms.length();
248         String term = "";
249         for (int i = 0; i < length; i++) {
250             term = terms.front();
251             if (!termSet.contains(term)) {
252                 termSet.add(term);
253             }
254             terms.rotate(1);
255         }
256     }
257
258     /**
259     * Generates the set of characters in the given {@code String} into the
260     * given {@code Set}.
261     *
262     * @param str
263     *         the given {@code String}
264     * @param charSet
265     *         the {@code Set} to be replaced
266     * @replaces charSet
267     * @ensures charSet = entries(str)
268     */
269     public static void generateElements(String str, Set<Character> charSet) {
270         /*
271          * for loop adds characters to the set if they are not repeats through
272          * the length of the string
273          */
274         int length = str.length();
275         for (int i = 0; i < length; i++) {
276             if (!charSet.contains(str.charAt(i))) {
277                 charSet.add(str.charAt(i));
278             }
279         }
280     }
281
282     /**
283     * Returns the first "word" (maximal length string of characters not in
284     * {@code separators}) or "separator string" (maximal length string of
285     * characters in {@code separators}) in the given {@code text} starting at
286     * the given {@code position}.
287     *
288     * @param text
289     *         the {@code String} from which to get the word or separator
290     *         string
291     * @param position
292     *         the starting index
293     * @param separators
294     *         the {@code Set} of separator characters
295     * @return the first word or separator string found in {@code text} starting
```

```

296      *          at index {@code position}
297      * @requires 0 <= position < |text|
298      * @ensures <pre>
299      * nextWordOrSeparator =
300      *   text[position, position + |nextWordOrSeparator|) and
301      * if entries(text[position, position + 1)) intersection separators = {}
302      * then
303      *   entries(nextWordOrSeparator) intersection separators = {} and
304      *   (position + |nextWordOrSeparator| = |text| or
305      *   entries(text[position, position + |nextWordOrSeparator| + 1))
306      *   intersection separators /= {})
307      * else
308      *   entries(nextWordOrSeparator) is subset of separators and
309      *   (position + |nextWordOrSeparator| = |text| or
310      *   entries(text[position, position + |nextWordOrSeparator| + 1))
311      *   is not subset of separators)
312      * </pre>
313      */
314      public static String nextWordOrSeparator(String text, int position,
315          Set<Character> separators) {
316          String word = "";
317          int length = text.length();
318          /*
319           * if the char at position is in the set it returns the string of chars
320           * of consecutive separators else it does the opposite for chars not in
321           * the set by adding each char to a string
322           */
323          if (separators.contains(text.charAt(position))) {
324              while (position < length
325                  && separators.contains(text.charAt(position))) {
326                  word = word + text.charAt(position);
327                  position++;
328              }
329          } else {
330              while (position < length
331                  && !separators.contains(text.charAt(position))) {
332                  word = word + text.charAt(position);
333                  position++;
334              }
335          }
336          /*
337           * returns the word created of separators or non separators
338           */
339          return word;
340      }
341
342      /**
343       * Main method.
344       *
345       * @param args
346       *         the command line arguments
347       */
348      public static void main(String[] args) {
349          SimpleReader in = new SimpleReader1L();
350          SimpleWriter out = new SimpleWriter1L();
351          /*
352           * asks user for file name and folder to store in
353           */
354          out.println("Enter the name of the file for the terms: ");

```

```
355     String filename = in.nextLine();
356     out.println("Enter the name of the folder for the term html files: ");
357     String folderName = in.nextLine();
358     /*
359      * creates a reader for the text file and a writer for the index
360      */
361     SimpleReader inFile = new SimpleReader1L(filename);
362     SimpleWriter outIndex = new SimpleWriter1L(folderName + "/index.html");
363     /*
364      * creates the set of separators
365      */
366     String separators = " ,.";
367     Set<Character> separatorSet = new Set1L<Character>();
368     generateElements(separators, separatorSet);
369     /*
370      * creates the Map, Queue, and Set to store the terms and/or definitions
371      */
372     Map<String, String> termsAndDefs = new Map1L<>();
373     Queue<String> terms = collectTerms(inFile, termsAndDefs);
374     Set<String> termSet = new Set1L<String>();
375     queueToSet(terms, termSet);
376     /*
377      * creates comparator to sort the queue of terms alphabetically
378      */
379     Comparator<String> cs = new OrderAlpha();
380     terms.sort(cs);
381     /*
382      * prints to index.html
383      */
384     outputIndex(outIndex, terms);
385     /*
386      * for loop runs for each term of the glossary printing out their
387      * respective pages
388      */
389     int termNum = terms.length();
390     String term = "";
391     for (int i = 0; i < termNum; i++) {
392         term = terms.front();
393         SimpleWriter outTerm = new SimpleWriter1L(
394             folderName + "/" + term + ".html");
395         terms.rotate(1);
396         outputTermHTML(outTerm, termsAndDefs, term, separatorSet, termSet);
397         outTerm.close();
398     }
399     /*
400      * close input and output streams
401      */
402     in.close();
403     out.close();
404     outIndex.close();
405     inFile.close();
406 }
407
408 }
409
```