```
1import java.util.Comparator;
 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;
 9import components.simplereader.SimpleReader;
10 import components.simplereader.SimpleReader1L;
11import components.simplewriter.SimpleWriter;
12import components.simplewriter.SimpleWriter1L;
13
14/**
15 * Program that reads text input from files and turns the terms and definitions
16 * in the file into a glossary of a series of HTML webpages.
18 * @author Bashir Ali
19 *
20 */
21 public final class Glossary {
23
24
       * . Comparator class used to sort strings alphabetically.
25
       * @param o1
26
27
                    The first string to be compared
28
       * @param o2
29
                    the second string to be compared
30
       * @return A negative number if o1 is 'less than' o2, positive if o1 is
31
                 'greater than' o2, and zero if o1 and o2 are 'equal'
32
33
       */
34
      private static class Sort implements Comparator<String> {
35
          @Override
36
          public int compare(String o1, String o2) {
37
              return o1.compareTo(o2);
38
          }
39
      }
40
41
42
       * Returns the next word or separator from the text starting from the
43
       * specified position.
44
       * @param text
45
46
                    The text to get the next word or separator from
       * @param position
47
48
                    The starting position to search for the next word or separator
49
       * @param separators
50
                    A set of symbols considered as separators
51
       * @return The next word or separator found in the text starting from the
52
                 given position.
       */
53
54
      private static String nextWordOrSeparator(String text, int position,
55
              Set<Character> separators) {
56
          //avoids checkstyle error with incrementing position
57
          int num = position;
```

```
58
           String result = "";
59
           //boolean to track if separator is found
 60
           boolean hasSeparators = false;
 61
           while (num < text.length() && !hasSeparators) {</pre>
                //checks character at position index for separator
 62
 63
               char ch = text.charAt(num);
 64
               hasSeparators = separators.contains(ch);
 65
               if (hasSeparators) {
                   //add separator to string if found
 66
 67
                   //did not know how to complete with 1 return
 68
                   return result;
 69
               }
 70
               result += ch;
 71
               num++;
 72
 73
           return result;
 74
       }
 75
       /**
 76
 77
        * Reads terms and definitions and puts them into map and sorts the keys of
 78
        * map into a queue.
 79
 80
        * @param_input
 81
                      The SimpleReader to read input from.
        * map
 82
 83
                     The Map to populate with key-value pairs.
        * @requires format Term and previous definition to be separated by a single
 84
 85
 86
        * @return A sorted Queue containing the keys from the Map.
 87
 88
       public static Queue<String> inputTermsAndDefinitions(SimpleReader input,
 89
               Map<String, String> map) {
           //empty queue for keys to be stored in
 90
 91
           Queue<String> queue = new Queue1L<>();
           Comparator<String> comparator = new Sort();
 92
 93
 94
 95
            * while not at the end, store the first line as a key following line as
 96
            * a value if the next line is not empty, keep adding to the value
 97
 98
           while (!input.atEOS()) {
99
               String key = input.nextLine();
100
               String value = input.nextLine();
101
               String emptyLine = input.nextLine();
102
103
               while (!emptyLine.equals("")) {
104
                   value += emptyLine;
105
                   emptyLine = input.nextLine();
106
               }
107
               //add key and value to map, and sort the keys in the queue
108
109
               map.add(key, value);
110
               queue.enqueue(key);
111
               queue.sort(comparator);
112
113
           return queue;
114
       }
```

```
115
       /**
116
117
        * generates an index page in HTML. The index page includes a list of links
118
        * to individual pages, with each link being connected to a key in the
119
        * queue.
120
        * @param fileIn
121
122
                     SimpleReader to read input from.
123
          @param fileOut
124
                     SimpleWriter to write the HTML index page.
        * @param map
125
126
                     Map containing the terms as keys and definitions as values.
127
        * @param queue
128
                     Queue contains key from map in alphabetical order.
        */
129
130
       public static void generateIndexPage(SimpleReader fileIn,
131
               SimpleWriter fileOut, Map<String, String> map,
132
               Queue<String> queue) {
133
134
           //printing for <a href="html">html</a> page
135
           fileOut.println("<html>");
136
           fileOut.println("<title>Glossary</title>");
           fileOut.println("</head>");
137
           fileOut.println("<body>");
138
           fileOut.println("<h2>Glossary</h2>");
139
           fileOut.println("<hr/>");
140
141
           fileOut.println("<h3>Index</h3>");
142
143
           //creates term page for each item in the queue
144
           fileOut.println("");
145
           for (String str : queue) {
               fileOut.print("");
146
147
               fileOut.println(
                        "<a href=\"" + str + ".html\">" + str + "</a>");
148
149
150
           fileOut.println("");
151
152
           fileOut.println("</body>");
153
           fileOut.println("</html>");
154
       }
155
       /**
156
157
        * Outputs the definition (values in the map) to a HTML file which is
158
        * hyperlinked on the index page.
159
        * @param fileOut
160
161
                     SimpleWriter to write the HTML output.
        * @param definition
162
163
                     The definition of the term to be outputted
164
        * @param queue
                     Queue contains key from map in alphabetical order.
165
166
167
       public static void outputDefinitions(SimpleWriter fileOut,
168
               String definition, Queue<String> queue) {
169
           Set<Character> separators = new Set1L<>();
           separators.add(':');
170
171
           separators.add(';');
```

```
172
           separators.add(' ');
173
           separators.add('~');
           separators.add('!');
174
           separators.add(' ');
175
176
177
           fileOut.print("");
178
           int i = 0;
179
           while (i < definition.length()) {</pre>
               String str = nextWordOrSeparator(definition, i, separators);
180
181
               boolean contains = false;
182
               for (String s : queue) {
183
                   //if the term is contained in the queue, link to that other term page
184
                   if (s.equals(str)) {
185
                        contains = true;
                       fileOut.print("<a href= " + '"' + str + ".html" + '"' + ">"
186
                               + str + " " + "</a>");
187
188
                   }
189
               }
               //if its not contained, print as plain text
190
191
               if (!contains) {
                   fileOut.print(str + " ");
192
193
               }
194
               //adds 1 for the space between words and the word length to go to next word
195
               i = i + 1 + str.length();
196
197
           fileOut.print("");
198
       }
199
200
       /**
201
        * Generates individual HTML pages for each term in the glossary and stores
202
        * them in folder.
203
        * @param folder
204
205
                     The folder where the HTML pages will be stored.
        * @param map
206
207
                     Map containing the terms as keys and definitions as values.
        * @param queue
208
209
                     Queue contains key from map in alphabetical order.
210
211
       public static void generateTermPages(String folder, Map<String, String> map,
212
               Queue<String> queue) {
213
           // process terms in reverse order from the queue and remove from map
214
           for (int i = queue.length(); i != 0; i--) {
215
               Map.Pair<String, String> current = map.remove(queue.front());
216
               queue.rotate(-1);
217
218
               SimpleWriter fileOut = new SimpleWriter1L(
                       folder + "/" + current.key() + ".html");
219
220
               String term = current.key();
221
               String definition = current.value();
222
223
               //printing for html page
               fileOut.println("<html>");
224
               fileOut.println("<head>");
225
               fileOut.println("<title>" + term + "</title>");
226
227
               fileOut.println("</head>");
228
               fileOut.println("<body>");
```

```
Friday, April 14, 2023, 12:42 PM
Glossary.java
229
               fileOut.print("<h2>");
230
               fileOut.print("<b>");
231
               fileOut.print("<i>");
               fileOut.println("<font color=" + "\"red\">" + term
232
233
                        + "</font></i></b></h2>");
234
               fileOut.println("");
235
236
237
               //output definitions of words on the term page
238
               outputDefinitions(fileOut, definition, queue);
239
               fileOut.println("<hr />");
               fileOut.println("");
240
241
               fileOut.println("return to <a href=\"index.html\">index</a>");
               fileOut.println("");
fileOut.println("</body>");
242
243
               fileOut.println("</html>");
244
245
               fileOut.close();
246
           }
247
       }
248
249
250
        * Main method.
251
252
        * @param args
253
                     the command line arguments
        */
254
255
       public static void main(String[] args) {
256
           SimpleReader in = new SimpleReader1L();
257
           SimpleWriter out = new SimpleWriter1L();
258
259
           out.print("Enter a file: ");
260
           String userFile = in.nextLine();
           out.print("Enter the name of a folder: ");
261
262
           String userFolder = in.nextLine();
263
264
           SimpleReader fileIn = new SimpleReader1L(userFile);
265
           SimpleWriter fileOut = new SimpleWriter1L(userFolder + "/index.html");
266
267
           //creates queue of keys and updates empty map to fill with keys/values
268
           Map<String, String> map = new Map1L<>();
           Queue<String> queue = inputTermsAndDefinitions(fileIn, map);
269
270
271
           generateIndexPage(fileIn, fileOut, map, queue);
272
           generateTermPages(userFolder, map, queue);
273
274
           in.close();
           out.close();
275
276
           fileIn.close();
277
           fileOut.close();
278
       }
279
280}
281
```