```java
import java.util.Comparator;

import components.map.Map;
import components.map.Map1L;
import components.queue.Queue;
import components.queue.Queue1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Program accepts a file and generates a glossary in HTML terms.
 *
 * @ Vaishnavi Kasabwala
 *
 */

public final class Glossary {

    /**
     * Compare {@code String}s in lexicographic order.
     */
    private static class StringLT implements Comparator<String> {
        @Override
        public int compare(String o1, String o2) {
            return o1.compareTo(o2);
        }
    }

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

    /**
     * Reads the file and generates two Queues, one with words and then one with
     * both words and definition.
     *
     * @param file
     *            the input file from user
     * @param glossary
     *            the generated list of terms
     * @param terms
     *            the individual words given by the file
     * @updates glossary, keys
     * @requires <pre>
     * file != empty
     * </pre>
     * @ensures <pre>
     * glossary contains list of words and their definitions.
     * terms contains list of words.
     * </pre>
     */
    public static void reader(SimpleReader out,
            Queue<Map<String, String>> glossary, Queue<String> terms) {
```

```
58          assert glossary != null : "Violation of: glossary is not null";
59          assert terms != null : "Violation of: terms is not null";
60          assert out != null : "Violation of: file is not null";
61
62          while (!out.atEOS()) {
63              Map<String, String> line = new Map1L<>();
64
65              // intake for the word
66              String word = "";
67              if (!out.atEOS()) {
68                  word = out.nextLine();
69              }
70
71              //intake for the definition of the word
72              String definition = "";
73              if (!out.atEOS()) {
74                  definition = out.nextLine();
75              }
76
77              // if empty, moves onto text word
78              if (!word.isEmpty() && !definition.isEmpty()) {
79                  terms.enqueue(word);
80              }
81
82              //forms definition
83              String temp = "";
84              if (!out.atEOS()) {
85                  temp = out.nextLine();
86              }
87              while (!temp.isEmpty()) {
88                  definition = definition + " " + temp;
89                  if (!out.atEOS()) {
90                      temp = out.nextLine();
91                  } else {
92                      temp = "";
93                  }
94              }
95
96              if (!word.isEmpty() && !definition.isEmpty()) {
97                  // adds every word and definition to the map
98                  line.add(word, definition);
99
100                 // adds map to the glossary
101                 glossary.enqueue(line);
102             }
103         }
104     }
105
106     /**
107      * Sorts the words in the glossary into lexicographical order.
108      *
109      * @param glossary
110      *            the generated list of terms
111      * @param terms
112      *            the individual words given by the file
113      * @updates glossary
114      * @requires <pre>
```

```
115      * terms = first String in each Map in glossary
116      * </pre>
117      * @ensures <pre>
118      * glossary contains list of words and their definitions in lexicographical
119      *       order
120      * terms contains list of words in lexicographical order
121      * </pre>
122      */
123     public static void sort(Queue<Map<String, String>> glossary,
124             Queue<String> terms) {
125         assert glossary != null : "Violation of: glossary is not null";
126         assert terms != null : "Violation of: keys is not null";
127
128         Queue<Map<String, String>> temp = new Queue1L<>();
129         temp.transferFrom(glossary);
130
131         while (temp.length() != 0) {
132             String word = terms.dequeue();
133
134             for (int i = 0; i < temp.length(); i++) {
135                 Map<String, String> term = temp.dequeue();
136
137                 if (term.hasKey(word)) {
138                     glossary.enqueue(term);
139                 } else {
140                     temp.enqueue(term);
141                 }
142             }
143             terms.enqueue(word);
144         }
145     }
146
147     /**
148      * Generates the main index page, listing words in lexicographical order
149      * with a link to each word's page.
150      *
151      * @param terms
152      *            the individual words given by the file
153      * @param path
154      *            the file the HTML files will be saved to
155      * @requires <pre>
156      * terms = first String in each Map in glossary
157      * </pre>
158      * @ensures <pre>
159      * generates index with a lexicographical list of words with
160      * links to their own pages with a word and definition
161      * </pre>
162      */
163     public static void generateIndex(Queue<String> terms, String path) {
164         assert terms != null : "Violation of: terms is not null";
165
166         SimpleWriter out = new SimpleWriter1L(path + "/index.html");
167
168         // header
169         out.println("<html>");
170         out.println("<head>");
171         out.println("<title>Glossary</title>");
```

```
172         out.println("</head>");
173         out.println("<body>");
174         out.println("<h2>Glossary</h2>");
175         out.println("<hr>");
176         out.println("<h3>Index</h3>");
177         out.println("<ul>");
178
179         for (int i = 0; i < terms.length(); i++) {
180             out.println("<li>");
181             String name = terms.front();
182
183             out.println("<a href=\"" + name + ".html\">" + name + "</a>");
184             out.println("</li>");
185
186             terms.rotate(1);
187         }
188
189         //footer
190         out.println("</ul>");
191         out.println("</body>");
192         out.println("</html>");
193
194         out.close();
195     }
196
197     /**
198      * Outputs the "opening" tags in the generated HTML file. These are the
199      * expected elements generated by this method:
200      *
201      * @param terms
202      *            the individual words given by the file
203      * @param out
204      *            the output stream
205      * @updates out.content
206      * @requires out.is_open terms != empty
207      * @ensures out.content = #out.content * [the HTML "opening" tags]
208      */
209     private static void outputHeader(Queue<String> terms, SimpleWriter out) {
210         assert out != null : "Violation of: out is not null";
211         assert out.isOpen() : "Violation of: out.is_open";
212         assert terms != null : "Violation of: terms is not null";
213
214         String name = terms.front();
215
216         out.println("<html>");
217         out.println("<head>");
218         out.println("<title>" + name + "</title>");
219         out.println("</head>");
220         out.println("<body>");
221         out.println("<h2>");
222         out.println("<b>");
223         out.println("<i>");
224         out.println("<font color=\"red\">" + name + "</font>");
225         out.println("</i>");
226         out.println("</b>");
227         out.println("</h2>");
228         out.println("<blockquote>");
```

```
229     }
230
231     /**
232      * Outputs the "closing" tags in the generated HTML file. These are the
233      * expected elements generated by this method:
234      *
235      * @param out
236      *            the output stream
237      * @updates out.contents
238      * @requires out.is_open
239      * @ensures out.content = #out.content * [the HTML "closing" tags]
240      */
241     private static void outputFooter(SimpleWriter out) {
242         assert out != null : "Violation of: out is not null";
243         assert out.isOpen() : "Violation of: out.is_open";
244
245         out.println("</blockquote>");
246         out.println("<hr>");
247         out.println("<p>");
248         out.println("Return to ");
249         out.println("<a href=\"index.html\">index</a>");
250         out.println(".");
251         out.println("</p>");
252         out.println("</body>");
253         out.println("</html>");
254     }
255
256     /**
257      * Generates individual HTML pages containing a word, its definition, and a
258      * link back to the index.
259      *
260      * @param glossary
261      *            the generated list of terms
262      * @param terms
263      *            the individual words given by the file
264      * @param path
265      *            the file the HTML files will be saved to
266      * @requires <pre>
267      * terms != empty
268      * </pre>
269      * @ensures <pre>
270      * pages for different words will be formed containing the word, definition,
271      * and a link to the index page.
272      * </pre>
273      */
274     public static void generatePages(Queue<Map<String, String>> glossary,
275             Queue<String> terms, String path) {
276         assert glossary != null : "Violation of: glossary is not null";
277         assert terms != null : "Violation of: terms is not null";
278
279         for (int i = 0; i < terms.length(); i++) {
280             String name = terms.front();
281             SimpleWriter file = new SimpleWriter1L(path + "/" + name + ".html");
282
283             outputHeader(terms, file);
284             printDefinition(name, file, glossary, terms);
285             outputFooter(file);
```

```
286
287             terms.rotate(1);
288             glossary.rotate(1);
289
290             file.close();
291         }
292     }
293
294     /**
295      * Generates definitions for each word.
296      *
297      * @param name
298      *            name of the word
299      * @param out
300      *            the output stream
301      * @param glossary
302      *            the generated list of terms
303      * @param terms
304      *            the individual words given by the file
305      * @requires <pre>
306      * terms != empty, glossary != empty, name != empty, file is valid
307      * </pre>
308      * @ensures <pre>
309      * new definition is formed with links to other words in it
310      * </pre>
311      */
312     public static void printDefinition(String name, SimpleWriter out,
313             Queue<Map<String, String>> glossary, Queue<String> terms) {
314         assert glossary != null : "Violation of: glossary is not null";
315         assert terms != null : "Violation of: terms is not null";
316         assert name != null : "Violation of: name is not null";
317         assert out != null : "Violation of: file is not null";
318
319         String definition = glossary.front().value(name);
320         String concat = "";
321
322         for (int i = 0; i < terms.length(); i++) {
323             String word = terms.front();
324             concat = isTerm(word, definition);
325
326             if (!concat.isEmpty()) {
327                 definition = concat;
328             }
329             terms.rotate(1);
330         }
331
332         out.println(definition);
333     }
334
335     /**
336      * Alters definition to link to words found in definition.
337      *
338      * @param term
339      *            name of the word tested against each definition
340      * @param definition
341      *            definition of a particular unique word
342      * @requires <pre>
```

```
343      * term != empty, definition != valid
344      * </pre>
345      * @ensures <pre>
346      * new definition is formed with links to other words in it
347      * </pre>
348      */
349     public static String isTerm(String term, String definition) {
350         assert term != null : "Violation of: term is not null";
351         assert definition != null : "Violation of: definition is not null";
352 
353         String concat = "";
354 
355         for (int i = 0; i <= definition.length() - term.length(); i++) {
356             String temp = definition.substring(i, term.length() + i);
357 
358             if (temp.equals(term)) {
359                 String first = definition.substring(0, i - 1) + " <a href=\""
360                         + term + ".html\">" + term + "</a>";
361                 String last = definition.substring(i + term.length(),
362                         definition.length());
363 
364                 concat = first + last;
365             }
366         }
367         return concat;
368     }
369 
370     /**
371      * Main method.
372      *
373      * @param args
374      *            the command line arguments
375      */
376     public static void main(String[] args) {
377         SimpleReader in = new SimpleReader1L();
378         SimpleWriter out = new SimpleWriter1L();
379 
380         // output to folder in project folder
381         out.print(
382                 "Please a folder for the files to be stored (ex. data, doc, lib, etc.): ");
383         String path = in.nextLine();
384 
385         out.print("Enter a file to add to the glossary: ");
386         SimpleReader file = new SimpleReader1L(in.nextLine());
387 
388         Queue<Map<String, String>> glossary = new Queue1L<>();
389         Queue<String> terms = new Queue1L<>();
390 
391         reader(file, glossary, terms);
392         Comparator<String> comp = new StringLT();
393         terms.sort(comp);
394         sort(glossary, terms);
395         generateIndex(terms, path);
396         generatePages(glossary, terms, path);
397 
398         file.close();
399         in.close();
```

```
400         out.close();
401     }
402
403 }
```