```
 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.simplereader.SimpleReader;
 8 import components.simplereader.SimpleReader1L;
 9 import components.simplewriter.SimpleWriter;
10 import components.simplewriter.SimpleWriter1L;
11
12 /**
13  * Program to generate a glossary with a given input file.
14  *
15  * @author Ansh Pachauri
16  */
17 public final class Glossary {
18
19     /**
20      * Compare {@code String}s in Alphabetical order.
21      */
22     private static class StringLT implements Comparator<String>
   {
23         @Override
24         public int compare(String s1, String s2) {
25             return s1.compareTo(s2);
26         }
27     }
28
29     /**
30      * No argument constructor--private to prevent
   instantiation.
31      */
32     private Glossary() {
33     }
34
35     /**
36      * Extracts the terms and their definitions from an input
   data file. Assumes
```

```
37      * that each term starts on a new line and is terminated by
   an empty line.
38      * Stores the terms and their definitions in a dictionary-
   like data
39      * structure for easy access, where the terms are used as
   keys and the
40      * definitions are used as values.
41      *
42      * @param inputFile
43      *             The name of the input file containing the
   terms and their
44      *             definitions.
45      * @return A Map containing the extracted terms as keys and
   their
46      *         corresponding definitions as values.
47      * @ensures termMap contains term -> definition mappings
   from the input
48      *          file.
49      */
50     public static Map<String, String> termWithDef(String
   inputFile) {
51         Map<String, String> termMap = new Map1L<>();
52         SimpleReader in = new SimpleReader1L(inputFile);
53         String term = "";
54         String definition = "";
55
56         while (!in.atEOS()) {
57             String line = in.nextLine();
58
59             if (line.isEmpty()) {
60                 // Empty line indicates end of a term and its
   definition
61                 if (!term.isEmpty()) {
62                     termMap.add(term, definition);
63                     term = "";
64                     definition = "";
65                 }
66             } else {
67
```

```
 68                 // First non-empty line is assumed to be the
    term
 69                 if (term.isEmpty()) {
 70                     term = line;
 71                 }
 72                 // Accumulate lines as term definition
 73                 if (definition.isEmpty()) {
 74                     line = in.nextLine();
 75                     definition = line;
 76                 } else {
 77                     definition += "\n" + line;
 78                 }
 79
 80             }
 81         }
 82
 83         in.close();
 84         return termMap;
 85     }
 86
 87     /**
 88      * Sorts the terms in ascending order alphabetically after
    storing them in a
 89      * Queue.
 90      *
 91      * @param termMap
 92      *             The Map containing the terms as keys and
    their definitions as
 93      *             values.
 94      * @return A Queue with the terms sorted in ascending order
    alphabetically.
 95      * @ensures The terms in the returned Map are sorted in
    ascending order
 96      *          alphabetically in a queue.
 97      */
 98     public static Queue<String> sortTermsAlphabetically(
 99             Map<String, String> termMap) {
100         // Create a new Map to store the sorted terms
101         Map<String, String> temp = termMap.newInstance();
```

```
102         temp.transferFrom(termMap);
103
104         // Convert the keys of the input termMap to a Queue for
    sorting
105         Queue<String> termQueue = new Queue1L<>();
106         while (temp.size() > 0) {
107             Map.Pair<String, String> tempPair =
    temp.removeAny();
108             String key = tempPair.key();
109             String value = tempPair.value();
110             termQueue.enqueue(key);
111             termMap.add(key, value);
112         }
113
114         Comparator<String> order = new StringLT();
115         termQueue.sort(order);
116
117         return termQueue;
118     }
119
120     /**
121      * Creates the top-level index file and separate HTML files
    for each term in
122      * the given Map of terms and definitions.
123      *
124      * @param termMap
125      *             The Map containing the terms as keys and
    their definitions as
126      *             values.
127      * @param outputFolder
128      *             The output folder where the top-level index
    file and term HTML
129      *             files will be created.
130      * @param termQueue
131      *             Queue with sorted keys
132      * @requires termMap and outputFolder are not null.
133      * @ensures The top-level index file and term HTML files
    are created in the
134      *          specified output folder.
```

```
135     */
136    public static void createHTMLFiles(Map<String, String>
   termMap,
137            String outputFolder, Queue<String> termQueue) {
138        // Create the top-level index file
139        String indexFileName = outputFolder + "/index.html";
140        SimpleWriter indexWriter = new
   SimpleWriter1L(indexFileName);
141
142        // Write HTML code for the top-level index file
143        indexWriter.println("<html>");
144        indexWriter.println("<head><title>Glossary Index</
   title></head>");
145        indexWriter.println("<body>");
146        indexWriter.println("<h1>Glossary Index</h1>");
147        indexWriter.println("<hr>");
148        indexWriter.println("<h3>Index</h3>");
149        indexWriter.println("<ul>");
150
151        // Create a separate HTML file for each term
152
153        while (termQueue.length() > 0) {
154            String term = termQueue.dequeue();
155            String termFileName = outputFolder + "/" + term +
   ".html";
156            SimpleWriter termWriter = new
   SimpleWriter1L(termFileName);
157
158            // Write HTML code for the term file
159            termWriter.println("<html>");
160            termWriter.println("<head><title>" + term + "</
   title></head>");
161            termWriter.println("<body>");
162            termWriter.println("<h1><font color=\"red\"><b><i>"
   + term
163                    + "</i></b></font></h1>");
164            termWriter.println(
165                    "<p>" +
   "         "
```

```
166                             + termMap.value(term) + "</p>");
167             termWriter.println("<hr>");
168             // Prints a link to return to the index
169             termWriter.println(
170                     "<p>Return to <a
    href=\"index.html\">index</a>.</p>");
171             termWriter.println("</body>");
172             termWriter.println("</html>");
173 
174             termWriter.close();
175 
176             // Add a link to the term file in the top-level
    index file
177             indexWriter.println(
178                     "<li><a href=\"" + term + ".html\">" + term
    + "</a></li>");
179         }
180         indexWriter.println("</ul>");
181         indexWriter.println("</body>");
182         indexWriter.println("</html>");
183 
184         indexWriter.close();
185     }
186 
187     /**
188      * Check each definition for terms that appear in the
    glossary and replace
189      * them with hyperlinks to the corresponding term page.
190      *
191      * @param termMap
192      *             The Map containing the terms as keys and
    their definitions as
193      *             values.
194      * @requires termMap and outputFolder are not null.
195      * @ensures Definitions in termMap are updated to replace
    terms with
196      *          hyperlinks to the corresponding term pages.
197      */
198     public static void replaceTermsWithLinks(Map<String,
```

```
   String> termMap) {
199
200         Queue<String> termQueue =
   sortTermsAlphabetically(termMap);
201
202         for (String term : termQueue) {
203             String definition = termMap.value(term);
204             String[] words = definition.split(" ");
205             for (String word : words) {
206                 for (String terms : termQueue) {
207                     String termPageLink = "<a href=\"" + terms
   + ".html\">"
208                             + terms + "</a>";
209                     if (word.equals(terms) || word.equals(terms
   + ",")) {
210                         definition = definition.replace(terms,
   termPageLink);
211                     }
212                 }
213             }
214             termMap.replaceValue(term, definition);
215         }
216     }
217
218     /**
219      * Main method.
220      *
221      * @param args
222      *             the command line arguments; unused here
223      */
224     public static void main(String[] args) {
225         SimpleWriter out = new SimpleWriter1L();
226         SimpleReader in = new SimpleReader1L();
227
228         Queue<String> termQueue = new Queue1L<>();
229
230         // Get input file path from user
231         out.print("Enter input file path: ");
232         String inputFile = in.nextLine();
```

```
233
234         // Generate term map from input file
235         Map<String, String> termMap = termWithDef(inputFile);
236
237         // Sort terms alphabetically
238         termQueue = sortTermsAlphabetically(termMap);
239
240         // Create output folder
241         out.print("Enter output folder path: ");
242         String outputFolder = in.nextLine();
243         // Replace terms with links
244         replaceTermsWithLinks(termMap);
245         createHTMLFiles(termMap, outputFolder, termQueue);
246
247         // Display success message
248         out.println("Glossary generated successfully!");
249     }
250 }
251
```