```java
import components.map.Map;

/**
 *
 * This program processes a glossary text file to generate an HTML glossary. It
 * reads a list of terms and their definitions, sorts them, and creates an HTML
 * index along with individual pages for each term.
 *
 * @author David Park
 *
 */
public final class Glossary {

    /**
     * No argument constructor--private to prevent instantiation.
     */
    private Glossary() {
    }

    /**
     * Processes the input file and writes sorted glossary terms and their
     * definitions to HTML files.
     *
     * @param inputFilePath
     *            the path to the glossary input file
     * @param outputFolderPath
     *            the directory path where HTML files will be saved
     */
    private static void generateHTMLPages(String inputFilePath,
            String outputFolderPath) {
        SimpleReader inFile = new SimpleReader1L(inputFilePath);
        Map<String, String> dictionary = new Map1L<>();
        readTerms(inFile, dictionary);
        inFile.close();

        Queue<String> sortedTerms = sortTerms(dictionary);
        writeIndexHtml(sortedTerms, outputFolderPath);
        writeTermPages(dictionary, outputFolderPath);
    }

    /**
     * Reads terms and their definitions from a file and stores them in a map.
     *
     * @param inFile
     *            the SimpleReader to read from the input file
     * @param dictionary
     *            the map to store terms and definitions
     */
    private static void readTerms(SimpleReader inFile,
            Map<String, String> dictionary) {
        String key = null;
        StringBuilder value = new StringBuilder();
        while (!inFile.atEOS()) {
            String line = inFile.nextLine();
            if (line.trim().isEmpty()) {
                if (key != null) {
                    dictionary.add(key, value.toString().trim());
```

```
69                      key = null;
70                      value = new StringBuilder();
71                  }
72              } else if (line.indexOf(' ') == -1) {
73                  // Assuming single word (no spaces) is a term
74                  key = line.trim();
75              } else {
76                  value.append(line).append(" ");
77              }
78          }
79          if (key != null && value.length() > 0) {
80              dictionary.add(key, value.toString().trim());
81          }
82      }
83
84      /**
85       * Sorts the terms in alphabetical order using a sequence.
86       *
87       * @param dictionary
88       *            the map containing terms and their definitions
89       * @return a queue containing the sorted terms
90       */
91      private static Queue<String> sortTerms(Map<String, String> dictionary) {
92          Sequence<String> terms = new Sequence1L<>();
93          for (Map.Pair<String, String> term : dictionary) {
94              int position = 0;
95              // Find the correct position to insert in sorted order
96              while (position < terms.length()
97                      && terms.entry(position).compareTo(term.key()) < 0) {
98                  position++;
99              }
100             terms.add(position, term.key()); // Insert the term at the found position
101         }
102
103         // Now transfer the sorted terms from the Sequence to the Queue
104         Queue<String> sortedTerms = new Queue1L<>();
105         while (terms.length() > 0) {
106             sortedTerms.enqueue(terms.remove(0));
107             // Remove from the sequence and enqueue to the queue
108         }
109
110         return sortedTerms; // Return the queue with sorted terms
111     }
112
113     /**
114      * Writes the index HTML file containing links to each term's page.
115      *
116      * @param terms
117      *            the queue of sorted terms
118      * @param outputFolderPath
119      *            the path where the index.html file will be written
120      */
121     public static void writeIndexHtml(Queue<String> terms,
122             String outputFolderPath) {
123         SimpleWriter indexFile = new SimpleWriter1L(
124                 outputFolderPath + "/index.html");
125         indexFile.println(
```

```
126                 "<html><head><title>Glossary Index</title></head><body>");
127         indexFile.println("<h1>Glossary Index</h1><ul>");
128         for (String term : terms) {
129             indexFile.println(
130                     "<li><a href=\"" + term + ".html\">" + term + "</a></li>");
131         }
132         indexFile.println("</ul></body></html>");
133         indexFile.close();
134     }
135
136     /**
137      * Writes HTML pages for each term with their definitions formatted.
138      *
139      * @param dictionary
140      *            the map containing terms and their definitions
141      * @param outputFolderPath
142      *            the path where term HTML files will be written
143      */
144     public static void writeTermPages(Map<String, String> dictionary,
145             String outputFolderPath) {
146         for (Map.Pair<String, String> entry : dictionary) {
147             SimpleWriter termFile = new SimpleWriter1L(
148                     outputFolderPath + "/" + entry.key() + ".html");
149             termFile.println("<html><head><title>" + entry.key()
150                     + "</title></head><body>");
151             termFile.println(
152                     "<h1 style='color:red; font-weight:bold; font-style:italic;'>"
153                             + entry.key() + "</h1>");
154             termFile.println("<p>" + formatDefinition(entry.value(), dictionary)
155                     + "</p>");
156             termFile.println(
157                     "<p>Return to <a href=\"index.html\">Index</a>.</p>");
158             termFile.println("</body></html>");
159             termFile.close();
160         }
161     }
162
163     /**
164      * Formats the definition of a term by embedding hyperlinks to referenced
165      * terms within the glossary. This ensures that each term within the
166      * definition that matches a term in the glossary is clickable and links to
167      * the respective term page.
168      *
169      * @param definition
170      *            the definition to format
171      * @param dictionary
172      *            the map containing all terms and their definitions for lookup
173      * @return the formatted definition with HTML hyperlinks embedded
174      */
175     private static String formatDefinition(String definition,
176             Map<String, String> dictionary) {
177         Set<String> terms = new Set1L<>();
178         for (Map.Pair<String, String> entry : dictionary) {
179             terms.add(entry.key()); // Collect all terms for quick lookup
180         }
181
182         // Split definition into words considering punctuation and spaces
```

```
183         String[] words = definition.split("\\s+");
184         StringBuilder formatted = new StringBuilder();
185         for (String fragment : words) {
186             String cleanedFragment = fragment.replaceAll("[^a-zA-Z]", "");
187             // Clean word of punctuation
188             if (terms.contains(cleanedFragment)) {
189                 // Replace the term with a hyperlink only if it's a standalone term
190                 formatted.append(fragment.replace(cleanedFragment,
191                         "<a href=\"" + cleanedFragment + ".html\">"
192                                 + cleanedFragment + "</a>"));
193             } else {
194                 formatted.append(fragment);
195             }
196             formatted.append(' ');
197         }
198         return formatted.toString().trim();
199     }
200 
201     /**
202      * Main method.
203      *
204      * @param args
205      *            the command line arguments
206      */
207     public static void main(String[] args) {
208         SimpleReader in = new SimpleReader1L();
209         SimpleWriter out = new SimpleWriter1L();
210 
211         out.print("Enter the path to the glossary input file: ");
212         String inputFilePath = in.nextLine();
213         out.print("Enter the path to the output folder: ");
214         String outputFolderPath = in.nextLine();
215 
216         generateHTMLPages(inputFilePath, outputFolderPath);
217 
218         in.close();
219         out.close();
220     }
221 
222 }
223 
```