```java
import java.util.Comparator;

/**
 * The program is to prompt the user to enter the input file and generate an
 * index html page. In this way, several html pages would be generated with the
 * index html page.
 *
 * @author Yiming Cheng
 *
 */
public final class Glossary {

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

    /**
     * In order to sort the terms in an alphabetic order.
     *
     */
    private static class Order implements Comparator<String> {
        @Override
        public int compare(String s1, String s2) {
            return s1.compareTo(s2);
        }
    }

    /**
     * Returns the first "word" (maximal length string of characters not in
     * {@code separators}) or "separator string" (maximal length string of
     * characters in {@code separators}) in the given {@code text} starting at
     * the given {@code position}.
     *
     * @param text
     *            the {@code String} which are got from the word
     * @param position
     *            the starting position of index
     * @param separators
     *            the certain punctuation which is from the list.
     * @return the first word or separator string in the index position
     * @requires 0 <= position < |text|
     * @ensures <pre>
     * nextWordOrSeparator =
     *   text[position, position + |nextWordOrSeparator|)  and
     * if entries(text[position, position + 1)) intersection separators = {}
     * then
     *   entries(nextWordOrSeparator) intersection separators = {}  and
     *   (position + |nextWordOrSeparator| = |text|  or
     *    entries(text[position, position + |nextWordOrSeparator| + 1))
     *      intersection separators /= {})
     * else
     *   entries(nextWordOrSeparator) is subset of separators  and
     *   (position + |nextWordOrSeparator| = |text|  or
     *    entries(text[position, position + |nextWordOrSeparator| + 1))
     *      is not subset of separators)
     * </pre>
     */
    public static String nextWordOrSeparator(String text, int position,
            Set<Character> separators) {
        assert text != null : "Violation of: text is not null";
```

```
74          assert separators != null : "Violation of: separators is not null";
75          assert 0 <= position : "Violation of: 0 <= position";
76          assert position < text.length() : "Violation of: position < |text|";
77
78          int endPos = -1;
79          String word = "";
80          int i = position;
81          /*
82           * find the corresponding substrings by separator
83           */
84          while (i < text.length()) {
85              if (separators.contains(text.charAt(i)) && endPos == -1) {
86                  endPos = i;
87              }
88              if (endPos == -1) {
89                  word = text.substring(position, text.length());
90              } else if (endPos == position) {
91                  word = text.substring(position, position + 1);
92              } else {
93                  word = text.substring(position, endPos);
94              }
95              i++;
96          }
97
98          return word;
99
100     }
101
102     /**
103      * generate the list of the input and process it to output into the map.
104      *
105      * @param word
106      *            the map which are stored the data
107      * @param in
108      *            extract information from it
109      * @return a queue storing a list of terms from the input file
110      * @replaces word
111      * @requires SimpleReader in is open, in contains at least a term and a
112      *           definition, each pair of term and definition
113      * @ensures Map word has a series of terms and the related definitions,
114      *          Queue contains a list of terms
115      */
116     public static Queue<String> generateIn(Map<String, String> word,
117             SimpleReader in) {
118         assert word != null : "Violation of: word is not null";
119         assert in != null : "Violation of: in is not null";
120
121         Queue<String> summary = new Queue1L<>();
122         /*
123          * process the input into the map
124          */
125         while (!in.atEOS()) {
126             String term = in.nextLine();
127             String def = in.nextLine();
128             String moreDef = " ";
129             while (moreDef.length() > 0 && !in.atEOS()) {
130                 moreDef = in.nextLine();
131                 StringBuilder dDef = new StringBuilder(
132                         def.length() + moreDef.length());
133                 dDef.append(def);
134                 dDef.insert(def.length(), moreDef);
135                 def = dDef.toString();
```

```
136             }
137             word.add(term, def);
138             summary.enqueue(term);
139         }
140         return summary;
141     }
142 
143     /**
144      * distribute the string into the set.
145      *
146      * @param str
147      *            the given {@code String}
148      * @param charSet
149      *            the {@code Set} to be replaced
150      * @replaces charSet
151      * @ensures charSet = entries(str)
152      *
153      */
154     public static void generateElements(String str, Set<Character> charSet) {
155         assert str != null : "Violation of: str is not null";
156         assert charSet != null : "Violation of: charSet is not null";
157 
158         //find the right string for the set
159         for (int i = 0; i < str.length(); i++) {
160             char ch = str.charAt(i);
161             if (!charSet.contains(ch)) {
162                 charSet.add(ch);
163             }
164         }
165 
166     }
167 
168     /**
169      * generate several html pages to show the information.
170      *
171      * @param word
172      *            the map which stores information
173      * @param title
174      *            title include a list of terms
175      * @param outFolder
176      *            name of the output folder
177      * @restore title
178      * @requires word is not empty, termsQ is not empty
179      * @ensures create html pages corresponding to terms and definitions
180      */
181     public static void generateInfor(Map<String, String> word,
182             Queue<String> title, String outFolder) {
183         assert word != null : "Violation of: word is not null";
184         assert title != null : "Violation of: title is not null";
185         assert outFolder != null : "Violation of: outFolder is not null";
186         /*
187          * create the same queue as the title
188          */
189         Queue<String> tempTitle = title.newInstance();
190         /*
191          * create a loop to generate the children pages
192          */
193         while (title.length() > 0) {
194             /*
195              * find out whether the title includes terms
196              */
197             String term = title.dequeue();
```

```
198             String dfn = word.value(term);
199             tempTitle.enqueue(term);
200             /*
201              * generate html pages
202              */
203             SimpleWriter page = new SimpleWriter1L(
204                     outFolder + "/" + term + ".html");
205             page.println("<html>");
206             page.println("<head>");
207             page.println("<title>" + term + "</title>");
208             page.println("</head>");
209             page.println("<body>");
210             page.println("</head>");
211             page.println("<h2><b><i><font color=\"red\">" + term
212                     + "</font></i></b></h2>");
213             page.println("<blockquote>" + dfn + "</blockquote>");
214             page.println("<hr />");
215             page.println("<p>Return to <a href=\"index.html\">index</a>.</p>");
216             page.println("</body>");
217             page.println("</html>");
218             page.close();
219         }
220         title.transferFrom(tempTitle);
221     }
222
223     /**
224      * check if the the definition contains terms, and if the definition
225      * contains a term, then change the related html page to make the term link
226      * to the corresponding term page.
227      *
228      * @param word
229      *            a map which stores terms and the related definitions
230      * @param title
231      *            contains a list of terms
232      * @param strSet
233      *            a set contains a series of special separators
234      * @restore title
235      * @requires word is not empty, title is not empty
236      * @ensures change the definition format in the html page if the definition
237      *          contains terms
238      */
239     public static void changeTheTerms(Map<String, String> word,
240             Queue<String> title, Set<Character> strSet) {
241         assert word != null : "Violation of: word is not null";
242         assert title != null : "Violation of: title is not null";
243         assert strSet != null : "Violation of: strSet is not null";
244
245         Queue<String> temp = new Queue1L<>();
246         int position = 0;
247         while (title.length() > 0) {
248             /*
249              * extract terms and definitions into Strings
250              */
251             String term = title.dequeue();
252             temp.enqueue(term);
253             String dfn = word.value(term);
254             String key = "";
255             while (position < dfn.length()) {
256                 String str = nextWordOrSeparator(dfn, position, strSet);
257                 if (word.hasKey(str)) {
258                     key += "<a href=\"" + str + ".html\">" + str + "</a>";
259                 } else {
```

```
260                     StringBuilder kKey = new StringBuilder(
261                             key.length() + str.length());
262                     kKey.append(key);
263                     kKey.insert(key.length(), str);
264                     key = kKey.toString();
265                 }
266                 position = position + str.length();
267             }
268             /*
269              * update the definition
270              */
271             word.replaceValue(term, key);
272             position = 0;
273         }
274         title.transferFrom(temp);
275     }
276 
277     /**
278      * Main method.
279      *
280      * @param args
281      *            the command line arguments
282      */
283     public static void main(String[] args) {
284         SimpleReader in = new SimpleReader1L();
285         SimpleWriter out = new SimpleWriter1L();
286         /*
287          * prompt the user to enter the input file
288          */
289         out.print("Please enter an input file: ");
290         String file = in.nextLine();
291         SimpleReader inFile = new SimpleReader1L(file);
292         out.print("Please enter the folder to save files: ");
293         String folder = in.nextLine();
294         SimpleWriter outFile = new SimpleWriter1L(folder + "/index.html");
295         /*
296          * create a map to store terms and definitions create a queue to store
297          * terms to prepare for the next steps
298          */
299         Map<String, String> wordMap = new Map1L<>();
300         Queue<String> title = new Queue1L<>();
301         title.append(generateIn(wordMap, inFile));
302         /*
303          * sort the terms in an alphabetic order
304          */
305         Comparator<String> od = new Order();
306         title.sort(od);
307         /*
308          * generate an index page
309          */
310         Queue<String> temp = new Queue1L<>();
311         outFile.println("<html>");
312         outFile.println("<head>");
313         outFile.println("<title>Glossary</title>");
314         outFile.println("</head>");
315         outFile.println("<body>");
316         outFile.println("<h2>Glossary</h2>");
317         outFile.println("<hr />");
318         outFile.println("<h3>Index</h3>");
319         outFile.println("<ul>");
320         while (title.length() > 0) {
321             String term = title.dequeue();
```

```
322             temp.enqueue(term);
323             outFile.println(
324                     "<li><a href=\"" + term + ".html\">" + term + "</a></li>");
325         }
326         outFile.println("</ul>");
327         outFile.println("</body>");
328         outFile.println("</html>");
329         title.transferFrom(temp);
330         /*
331          * use the separators to divide terms
332          */
333         Set<Character> charSet = new Set1L<>();
334         final String separatorStr = " ,.:;!?";
335         generateElements(separatorStr, charSet);
336 
337         changeTheTerms(wordMap, title, charSet);
338 
339         /*
340          * generate several html pages for the user
341          */
342         generateInfor(wordMap, title, folder);
343 
344         /*
345          * Close input and output streams
346          */
347         inFile.close();
348         outFile.close();
349         in.close();
350         out.close();
351     }
352 
353 }
354 
```