```java
import components.map.Map;
import components.map.Map1L;
import components.sequence.Sequence;
import components.sequence.Sequence1L;
import components.set.Set;
import components.set.Set1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * This program will create an index page of many terms and will provide
 * separate pages for their definitions given an input file.
 *
 * @author Feras Akileh
 *
 */
public final class Glossary {

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

    /**
     * Adds the terms and definitions to two sequences
     *
     * @param in
     *            the input stream
     * @param termsMap
     *            the map that will contain the terms and their definitions
     * @param termsSet
     *            a set that contains just the terms
     */
    private static void deriveTermsAndDefs(SimpleReader in,
            Map<String, String> termsMap, Set<String> termsSet) {

        // creates a while loop that scans the input file
        while (!in.atEOS()) {

            // initializes variables for the term and definition
            String term = "";
            String definition = "";

            // creates a boolean variable that changes depending on if the
            // stream has an empty line
            boolean hasEmptyLine = true;

            // initializes a variable for the current line that the reader
            // is working on
            String current = in.nextLine();

            if (current.equals("")) {
                hasEmptyLine = false;
            } else {
                term = current;
            }
```

```
60
61              while (hasEmptyLine && !in.atEOS()) {
62
63                  current = in.nextLine();
64
65                  if (!current.equals("")) {
66                      definition = definition + " " + current;
67                  } else {
68                      hasEmptyLine = false;
69                  }
70              }
71
72              // adds terms and definitions to the map and terms set
73              termsMap.add(term, definition);
74              termsSet.add(term);
75
76          }
77
78      }
79
80      /**
81       * Takes the set of terms and puts them in alphabetical order
82       *
83       * @param termsSet
84       *            the set of terms
85       *
86       * @replaces termsSet
87       *
88       * @ensures terms = original terms set without word that is the earliest in
89       *          the alphabet
90       *
91       */
92      private static String alphabetize(Set<String> termsSet) {
93
94          // creates a duplicate set of termsSet
95          Set<String> termsSet2 = new Set1L<>();
96
97          // initializes the string that will be returned
98          String result = "";
99
100         // initializes a while loop that checks the terms in the set
101         while (termsSet.size() > 0 && result.equals("")) {
102
103             int index = 0;
104             String test = termsSet.removeAny();
105
106             for (String word : termsSet) {
107                 if (word.compareTo(test) < 0) {
108                     index++;
109                 }
110             }
111             if (index == 0) {
112                 result = test;
113             } else {
114                 termsSet2.add(test);
115             }
116         }
117
118         termsSet.add(termsSet2);
```

```
119         return result;
120
121     }
122
123     /**
124      * Generates the home page of the output HTML file.
125      *
126      * @param out
127      *            the output stream
128      * @param termsList
129      *            the sequence of terms in alphabetical order
130      * @param termsMap
131      *            the map of the terms with their definitions
132      * @param termArray
133      *            an array of the list of terms
134      * @param outPut
135      *            the output file location
136      */
137     private static void generateIndex(SimpleWriter out,
138             Sequence<String> termsList, Map<String, String> termsMap,
139             String[] termArray, String outPut) {
140
141         // creates new output file
142         String index = outPut + "/index.html";
143         SimpleWriter indexHome = new SimpleWriter1L(index);
144
145         // writes HTML code for the index page
146         indexHome.println("<html>");
147         indexHome.println("<head>");
148         indexHome.println("<title>" + "Glossary" + "</title>");
149         indexHome.println("</head>");
150
151         indexHome.println("<body>");
152         indexHome.println(
153                 "<p style=\"font-size:32pt;\"><Strong>Glossary</Strong></p>");
154         indexHome.println();
155         indexHome.println(
156                 "<p style=\"font-size:18pt;\"><Strong>Index</Strong></p>");
157
158         indexHome.println("<ul>");
159
160         // initializes a while loop that creates the html pages
161         while (termsList.length() > 0) {
162             String word = termsList.remove(0);
163             generateTermPage(word, out, termsMap, termArray, outPut);
164             indexHome.println(
165                     "<li><a href=\"" + word + ".html\">" + word + "</a></li>");
166         }
167
168         // writes html code to finish the home page
169         indexHome.println("</ul>");
170         indexHome.println("</body>");
171         indexHome.println("</html>");
172
173         // closes stream
174         indexHome.close();
175
176     }
177
```

```
178     /**
179      * Generates the page for the individual term
180      *
181      * @param word
182      *            the term the page is created around
183      * @param out
184      *            the output stream
185      * @param termsMap
186      *            the map of the terms with their definitions
187      * @param termArray
188      *            an array of the list of terms
189      * @param outPut
190      *            the output file location
191      */
192     private static void generateTermPage(String word, SimpleWriter out,
193             Map<String, String> termsMap, String[] termArray, String outPut) {
194
195         /*
196          * Generates string with location of the file as well as the html file's
197          * page
198          */
199         String termPage = outPut + "/" + word + ".html";
200         SimpleWriter webPage = new SimpleWriter1L(termPage);
201
202         webPage.println("<html>");
203         webPage.println("<head>");
204         webPage.println("<title>" + word + "</title>");
205         webPage.println("</head>");
206         webPage.println("<body>");
207         webPage.println("<p style=\"font-size:18pt;color:red;\"><b><i>" + word
208                 + "</b></i></p>");
209         webPage.println();
210
211         String definition = termsMap.value(word);
212         Set<Character> separatorSet = new Set1L<>();
213         String separators = " ,";
214
215         generateElements(separators, separatorSet);
216
217         String pageFeed = "";
218
219         int i = 0;
220         while (i < definition.length()) {
221             String item = nextWordOrSeparator(definition, i, separatorSet);
222             if (separatorSet.contains(item.charAt(0))) {
223                 pageFeed = pageFeed + item;
224             } else {
225                 int c = 0;
226                 int count = 0;
227                 while (c < termArray.length) {
228                     if (item.equals(termArray[c])) {
229                         pageFeed = pageFeed + "<a href=\"" + termArray[c]
230                                 + ".html\">" + item + "</a>";
231                         count++;
232                     }
233                     c++;
234                 }
235                 if (count == 0) {
236                     pageFeed = pageFeed + item;
```

```
237                 }
238             }
239             i += item.length();
240         }
241
242         webPage.println("<p style=\"text-align:left;\">" + pageFeed + "</p>");
243         webPage.println();
244         webPage.println("Return to <a href=\"index.html\">index</a>");
245         webPage.println("</body>");
246         webPage.println("</html>");
247
248         webPage.close();
249     }
250
251     /**
252      * Generates the set of characters in the given {@code String} into the
253      * given {@code Set}.
254      *
255      * @param str
256      *            the given {@code String}
257      * @param strSet
258      *            the {@code Set} to be replaced
259      * @replaces strSet
260      * @ensures strSet = entries(str)
261      */
262     private static void generateElements(String str, Set<Character> strSet) {
263         assert str != null : "Violation of: str is not null";
264         assert strSet != null : "Violation of: strSet is not null";
265
266         int i = str.length();
267
268         while (i > 0) {
269             char x = str.charAt(i - 1);
270             if (!strSet.contains(x)) {
271                 strSet.add(x);
272             }
273             i--;
274         }
275
276     }
277
278     /**
279      * Returns the first "word" (maximal length string of characters not in
280      * {@code separators}) or "separator string" (maximal length string of
281      * characters in {@code separators}) in the given {@code text} starting at
282      * the given {@code position}.
283      *
284      * @param text
285      *            the {@code String} from which to get the word or separator
286      *            string
287      * @param position
288      *            the starting index
289      * @param separators
290      *            the {@code Set} of separator characters
291      * @return the first word or separator string found in {@code text} starting
292      *         at index {@code position}
293      * @requires 0 <= position < |text|
294      * @ensures <pre>
295      * nextWordOrSeparator =
```

```
296      *   text[position, position + |nextWordOrSeparator|)  and
297      * if entries(text[position, position + 1)) intersection separators = {}
298      * then
299      *   entries(nextWordOrSeparator) intersection separators = {}  and
300      *   (position + |nextWordOrSeparator| = |text|  or
301      *    entries(text[position, position + |nextWordOrSeparator| + 1))
302      *      intersection separators /= {})
303      * else
304      *   entries(nextWordOrSeparator) is subset of separators  and
305      *   (position + |nextWordOrSeparator| = |text|  or
306      *    entries(text[position, position + |nextWordOrSeparator| + 1))
307      *      is not subset of separators)
308      * </pre>
309      */
310     private static String nextWordOrSeparator(String text, int position,
311             Set<Character> separators) {
312         assert text != null : "Violation of: text is not null";
313         assert separators != null : "Violation of: separators is not null";
314         assert 0 <= position : "Violation of: 0 <= position";
315         assert position < text.length() : "Violation of: position < |text|";
316
317         char charTest = text.charAt(position);
318         String result = "" + charTest;
319         boolean sepContain = separators.contains(charTest);
320         int endStr = position;
321
322         if (sepContain) {
323             endStr++;
324             if (endStr <= text.length()) {
325                 charTest = text.charAt(endStr);
326                 while (separators.contains(charTest)) {
327
328                     if (!separators.contains(charTest)) {
329                         sepContain = false;
330                     }
331                     endStr++;
332                     charTest = text.charAt(endStr);
333                 }
334             }
335
336         } else {
337             endStr++;
338             if (endStr < text.length()) {
339                 charTest = text.charAt(endStr);
340
341                 while (!separators.contains(charTest)
342                         && endStr != text.length()) {
343
344                     if (separators.contains(charTest)) {
345                         sepContain = true;
346                     }
347                     endStr++;
348                     if (endStr < text.length()) {
349                         charTest = text.charAt(endStr);
350                     }
351                 }
352             }
353         }
354
```

```
355         result = text.substring(position, endStr);
356
357         return result;
358
359     }
360
361     /**
362      * Main method.
363      *
364      * @param args
365      *            the command line arguments
366      */
367     public static void main(String[] args) {
368
369         // initializes input and output streams
370         SimpleWriter out = new SimpleWriter1L();
371         SimpleReader in = new SimpleReader1L();
372
373         // prompts the user for an input file
374         out.println("Please provide an input file: ");
375         String inPut = in.nextLine();
376         SimpleReader inFile = new SimpleReader1L(inPut);
377
378         // prompts the user for a folder in which to save the output pages
379         out.println("Please provide a folder: ");
380         String outPut = in.nextLine();
381
382         // creates a map that will have the terms and their definitions
383         Map<String, String> termsMap = new Map1L<>();
384         Set<String> termsSet = new Set1L<>();
385
386         // calls deriveTermsAndDefs
387         deriveTermsAndDefs(inFile, termsMap, termsSet);
388
389         Sequence<String> termsList = new Sequence1L<>();
390         String[] termArray = new String[termsSet.size()];
391
392         int i = 0;
393         while (0 < termsSet.size()) {
394             String nextTerm = alphabetize(termsSet);
395             termsList.add(termsList.length(), nextTerm);
396             termArray[i] = nextTerm;
397             i++;
398         }
399
400         // calls generateIndex
401         generateIndex(out, termsList, termsMap, termArray, outPut);
402
403         // closes streams
404         in.close();
405         out.close();
406
407     }
408
409 }
410
```