```java
import java.util.Comparator;

/**
 *
 * @author Micah Casey-Fusco
 *
 */
public class GlossarySearch {

    public static class Alpha implements Comparator<String> {
        @Override
        public int compare(String a, String b) {
            return a.compareTo(b);
        }
    }

    /**
     *
     * @param s
     * @return set<string>
     */
    public static Set<String> alphabetize(Map<String, String> s) {

        Comparator<String> compare = new Alpha();

        Queue<String> a = new Queue1L<String>();
        for (Map.Pair<String, String> i : s) {
            a.enqueue(i.key());
        }
        a.sort(compare);

        //make into set for Junit
        Set<String> termSet = new Set1L<>();

        while (a.length() != 0) {
            termSet.add(a.dequeue());
        }

        return termSet;

    }

    /**
     *
     * @param output
     * @param map
     * @param titleTerm
     * @param definition
     */
    public static void printHTMLfile(SimpleWriter output,
            Map<String, String> map, String titleTerm, String definition,
            Set<Character> separatorSet) {

        //create term.html file using variables
        output.println("<!DOCTYPE html>");
        output.println("<html lang=\"en\">");
        output.println("<head>");
```

```
 71         output.println("<meta charset=\"UTF-8\" />");
 72         output.println(
 73                 "<meta http-equiv=\"X-UA-Compatible\" content=\"IE=edge\" />");
 74         output.println(
 75                 "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\"
    />");
 76         output.println("<title>" + titleTerm + "</title>");
 77         output.println("<!-- css -->");
 78         output.println("<style>");
 79         output.println(".currentTerm {");
 80         output.println("font-weight: bold;");
 81         output.println("color: red;");
 82         output.println("font-style: italic;");
 83         output.println("display: block;");
 84         output.println("font-size: 1.5em;");
 85         output.println("margin-block-start: 0.83em;");
 86         output.println("margin-block-end: 0.83em;");
 87         output.println("margin-inline-start: 0px;");
 88         output.println("margin-inline-end: 0px;");
 89         output.println("}");
 90         output.println();
 91         output.println(".currentDefinition {");
 92         output.println("display: block;");
 93         output.println("margin-block-start: 0.83em;");
 94         output.println("margin-block-end: 0.83em;");
 95         output.println("margin-inline-start: 0px;");
 96         output.println("margin-inline-end: 0px;");
 97         output.println("}");
 98         output.println("</style>");
 99         output.println("</head>");
100         output.println("<body>");
101         output.println("<h2 class=\"currentTerm\">" + titleTerm + "</h2>");
102         output.println("<blockquote class=\"currentDefinition\">");
103 
104         /**
105          * currentIndex and currentWordLength were used to progress through the
106          * definition string, but their values were parallel so we combined them
107          */
108         int indexAndLength = 0;
109 
110         while (indexAndLength < definition.length()) {
111 
112             String currentWord = nextWord(definition, indexAndLength,
113                     separatorSet);
114             int cwLength = currentWord.length();
115 
116             //find substring of separator or non-separator characters
117             if (currentWord.charAt(0) != ' ' && currentWord.charAt(0) != ',') {
118                 if (map.hasKey(currentWord)) {
119                     output.print("<a href=\"" + currentWord + ".html\">"
120                             + currentWord + "</a>");
121                 } else {
122                     output.print(currentWord);
123                 }
124                 output.print(" ");
125             }
126 
```

```
127             indexAndLength = indexAndLength + cwLength;
128
129         }
130
131         output.println("</blockquote>");
132         output.println("<hr />");
133         output.println("<p>Return to <a href=\"index.html\">index</a>.</p>");
134         output.println("</body>");
135         output.println("</html>");
136     }
137
138     /**
139      *
140      * @param definition
141      * @param position
142      * @param separators
143      * @return string
144      */
145     public static String nextWord(String definition, int position,
146             Set<Character> separators) {
147
148         String result = "";
149         int i = position;
150
151         //compare char at position with separator set
152         if (!separators.contains(definition.charAt(position))) {
153             while (i < definition.length()
154                     && !separators.contains(definition.charAt(i))) {
155                 i++;
156             }
157             result = definition.substring(position, i);
158         } else {
159             while (i < definition.length()
160                     && separators.contains(definition.charAt(i))) {
161                 i++;
162             }
163             result = definition.substring(position, i);
164         }
165         return result;
166
167     }
168
169     /**
170      * Generates the set of characters in the given {@code String} into the
171      * given {@code Set}.
172      *
173      * @param str
174      *            the given {@code String}
175      * @param charSet
176      *            the {@code Set} to be replaced
177      * @replaces charSet
178      * @ensures charSet = entries(str)
179      */
180     public static void generateElements(String str, Set<Character> charSet) {
181
182         Set<Character> temp = new Set1L<>();
183
```

```
184         //check for empty case
185         if (str.equals("")) {
186             charSet.transferFrom(temp);
187         } else {
188             char[] strChars = str.toCharArray();
189             for (int i = 0; i < str.length(); i++) {
190                 if (!temp.contains(strChars[i])) {
191                     temp.add(strChars[i]);
192                 }
193 
194             }
195         }
196 
197         charSet.transferFrom(temp);
198     }
199 
200     /**
201      *
202      * @param output
203      * @param title
204      * @param map
205      */
206     public static void printIndexfile(SimpleWriter output, String title,
207             Map<String, String> map) {
208         Set<String> sortedTerms = alphabetize(map);
209         Queue<String> termsQ = new Queue1L<>();
210 
211         for (String term : sortedTerms) {
212             termsQ.enqueue(term);
213         }
214 
215         //create index.html text using variables
216         output.println("<html>");
217         output.println("<head>");
218         output.println("<title>" + title + "</title>");
219         output.println("</head>");
220         output.println("<body>");
221         output.println("<h2>" + title + "</h2>");
222         output.println("<hr />");
223         output.println("<h3>Index</h3>");
224         output.println("<ul>");
225         while (termsQ.length() > 0) {
226             String s = termsQ.dequeue();
227             output.println("<li><a href=" + s + ".html>" + s + "</a></li>");
228         }
229         output.println("</ul>");
230         output.println("</body>");
231         output.println("</html>");
232     }
233 
234     /**
235      *
236      * @param file
237      * @return map<String, String>
238      */
239     public static Map<String, String> getTerm(SimpleReader file) {
240 
```

```
241         //initialize variables
242         String term = "";
243         String definition = "";
244         Map<String, String> glossary = new Map1L<>();
245 
246         //loop through file and find terms + their definitions
247         while (!file.atEOS()) {
248             term = file.nextLine();
249             definition = file.nextLine();
250 
251             String holder = file.nextLine();
252             while (holder.length() != 0) {
253                 definition = definition.concat(holder);
254                 holder = file.nextLine();
255             }
256             glossary.add(term, definition);
257 
258         }
259 
260         return glossary;
261 
262     }
263 
264     /**
265      *
266      * @param args
267      */
268     public static void main(String[] args) {
269         //input file name and output folder
270         SimpleWriter out = new SimpleWriter1L();
271         SimpleReader in = new SimpleReader1L();
272 
273         //ask user for file and folder location
274         out.print("Enter input filename: ");
275         String inputFileName = in.nextLine();
276         out.print("Enter output folder name: ");
277         String outputFolderName = in.nextLine();
278 
279         //read glossary data from file
280         SimpleReader inputData = new SimpleReader1L(inputFileName);
281         Map<String, String> map = getTerm(inputData);
282 
283         //create output simplewriter for the index file, then run index method
284         SimpleWriter output = new SimpleWriter1L(
285                 outputFolderName + "/index.html");
286         printIndexfile(output, "Glossary", map);
287         output.close();
288 
289         //create separator set for later use
290         final String separatorStr = " \t, ";
291         Set<Character> separatorSet = new Set1L<Character>();
292         generateElements(separatorStr, separatorSet);
293 
294         //write html output files
295         Iterator<Map.Pair<String, String>> iter = map.iterator();
296         while (iter.hasNext()) {
297             Pair<String, String> p = iter.next();
```

```
298             SimpleWriter outputHTML = new SimpleWriter1L(
299                     outputFolderName + "/" + p.key() + ".html");
300             printHTMLfile(outputHTML, map, p.key(), p.value(), separatorSet);
301             outputHTML.close();
302         }
303
304         out.close();
305         in.close();
306         inputData.close();
307     }
308 }
309
```