```java
import java.io.Serializable;
import java.util.Comparator;

import components.map.Map;
import components.map.Map1L;
import components.queue.Queue;
import components.queue.Queue1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Generates a glossary of a group of HTML files.
 *
 * @author Ethan Jones
 */
public final class Glossary {

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

    /**
     * Comparator used to order the words queue.
     */
    public static final class StringOrder
            implements Comparator<String>, Serializable {
        private static final long serialVersionUID = 1L;

        @Override
        public int compare(String s1, String s2) {
            return s1.compareTo(s2);
        }
    }

    /**
     * Creates the header for the index file.
     *
     * @param fileOut
     *            printing to a specified file
     */
    public static void createIndexHeader(SimpleWriter fileOut) {
        fileOut.println("<html>");
        fileOut.println("<head>");
        fileOut.println("<title>Glossary</title>");
        fileOut.println("</head>");
        fileOut.println("<body>");
        fileOut.println("<h2>Glossary</h2>");
        fileOut.println("<hr>");
        fileOut.println("<h3>Index:</h3>");
        fileOut.println("<ul>");
    }

    /**
     * Creates the footer for the index file.
```

```
60      *
61      * @param fileOut
62      *            printing to a specified file
63      */
64     public static void createIndexFooter(SimpleWriter fileOut) {
65         fileOut.println("</ul>");
66         fileOut.println("</body>");
67         fileOut.println("</html>");
68     }
69
70     /**
71      * Creates the header for the index file.
72      *
73      * @param fileOut
74      *            printing to a file
75      * @param word
76      *            the word that is used as the title and is made bold and made
77      *            red in the file
78      * @param def
79      *            the definition of the word that is made bold and made red in
80      *            the file
81      */
82     public static void createHTMLWords(SimpleWriter fileOut, String word,
83             String def) {
84         fileOut.println("<html>");
85         fileOut.println("<head>");
86         fileOut.println("<title>" + word + "</title>");
87         fileOut.println("</head>");
88         fileOut.println("<body>");
89         fileOut.println("<h2><b><i><font color=\"red\">" + word
90                 + "</font></i></b></h2>");
91         fileOut.println("<blockquote>" + def + "</blockquote>");
92         fileOut.println("<hr />");
93         fileOut.println("<p>Return to <a href=\"index.html\">index</a>.</p>");
94         fileOut.println("</body>");
95         fileOut.println("</html>");
96         fileOut.println();
97     }
98
99     /**
100     * Creates a map that pairs the words to their definitions.
101     *
102     * @param in
103     *            grabs input from the specified file
104     * @param wordList
105     *            a given queue that has the words
106     * @return returns a sorted word definition list
107     */
108    public static Map<String, String> createMap(SimpleReader in,
109            Queue<String> wordList) {
110        Map<String, String> wordDefList = new Map1L<>();
111        String word = "";
112        String def = "";
113        while (!in.atEOS()) {
114            String text = in.nextLine();
115            if (text.contains(" ") && !text.isBlank()) {
116                def = text;
117            } else if (!text.contains(" ") && !text.isBlank()) {
118                word = text;
```

```
119                 wordList.enqueue(text);
120             } else {
121                 wordDefList.add(word, def);
122                 word = "";
123                 def = "";
124             }
125         }
126         Comparator<String> order = new StringOrder();
127         wordList.sort(order);
128         return wordDefList;
129     }
130 
131     /**
132      * Processes the words by creating files specific to each word.
133      *
134      * @param out
135      *             the file that is named after the word
136      * @param word
137      *             the word used to name the file
138      * @param def
139      *             the definition of the word that the file is named after
140      * @param folder
141      *             the folder for the path of the file that is created
142      */
143     public static void processWord(SimpleWriter out, String word, String def,
144             String folder) {
145         String fileName = word + ".html";
146         SimpleWriter fileOut = new SimpleWriter1L(folder + "/" + fileName);
147         out.println("<li><a href=\"" + fileName + "\">" + word + "</a></li>");
148         createHTMLWords(fileOut, word, def);
149     }
150 
151     /**
152      * Processes files by iterating through each word, capitalizing each word,
153      * then processing each word.
154      *
155      * @param file
156      *             The name of the file
157      * @param folder
158      *             the name of the folder for the path of the file
159      * @param out
160      *             the file that this is being printed to
161      */
162     public static void processFile(String file, String folder,
163             SimpleWriter out) {
164         SimpleReader inputFile = new SimpleReader1L(file);
165         Queue<String> wordList = new Queue1L<>();
166         Map<String, String> wordDefList = createMap(inputFile, wordList);
167         for (String word : wordList) {
168             String def = wordDefList.value(word);
169             word = word.substring(0, 1).toUpperCase() + word.substring(1);
170             processWord(out, word, def, folder);
171         }
172     }
173 
174     /**
175      * Main method.
176      *
177      * @param args
```

```
178      *            the command line arguments; unused here
179      */
180     public static void main(String[] args) {
181         SimpleWriter out = new SimpleWriter1L();
182         SimpleReader in = new SimpleReader1L();
183         SimpleWriter fileOut = new SimpleWriter1L("data/index.html");
184         out.print("Enter a file name: ");
185         String file = in.nextLine();
186         out.print("Enter a folder name: ");
187         String folder = in.nextLine();
188         createIndexHeader(fileOut);
189         processFile(file, folder, fileOut);
190         createIndexFooter(fileOut);
191         out.close();
192         in.close();
193         fileOut.close();
194     }
195 }
196
```