```java
import components.naturalnumber.NaturalNumber;

/**
 * Put a short phrase describing the program here.
 *
 * @author Jonathan Pater
 *
 */
public final class Glossary {

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

    }

    /**
     * Generates HTML file for the home-page of the glossary that
contains a
     * list of all of the featured terms that are also linked to
their.
     * respective HTML page
     *
     * @param out
     *            output stream
     * @param termArray
     *            array containing all of the terms featured in the
glossary in
     *            alphabetical order
     * @requires out.is_open
     * @ensures Output HTML file is valid and the HTML files for
the terms
     *          already exist
     *
     */
    public static void generateHomepage(SimpleWriter out, String[]
termArray) {
        out.println("<html> <head> <title>Glossary</title> </head>
<body> "
                + "<h1>Glossary</h1> <hr> <h2>Index</h2> <ul>");
        //generates a list of linked terms in alphabetical order
based off the
        //alphabetically ordered array provided
```

```
47          for (int i = 0; i < termArray.length; i++) {
48              String temp = termArray[i];
49              out.println("<li> <a href=\"" + temp + ".html\">" +
    temp
50                      + "</a> </li>");
51          }
52          out.println("</ul> </body> </html>");
53      }
54
55      /**
56       * Outputs an HTML file for a term and its definition, also
    linking terms to
57       * words in a definition that are also terms in the glossary.
58       *
59       * @param term
60       *              string that contains the term to be used
61       * @param def
62       *              String containing the definition
63       * @param outFolder
64       *              the output file in which the HTML file will be
    stored in
65       * @param termSet
66       *              set of all terms in the glossary
67       * @requires String outFolder contains an existing folder name
    for the HTML
68       *             file to be stored in
69       * @ensures output is a valid HTML file that properly gives the
    definition
70       *           of a term
71       */
72      public static void termHTML(String term, String def, String
    outFolder,
73              Set<String> termSet) {
74          String defi = getDef(termSet, def);
75          //^ gets definition string which is either the same as it
    was before the
76          //method call or will contain linked terms if terms in the
    glossary are
77          //contained in the definition string
78          //outFile = name/file destination for the html file
    generated by the method
79          String outFile = outFolder + "/" + term + ".html";
80          SimpleWriter output = new SimpleWriter1L(outFile);
81          output.println("<html> <head> <title>" + term + "</
    title>");
```

```
 82         output.println("<body> <h1> <b> <i> <font color = \"red\">"
    + term
 83                 + "</font> </i> </b> </h1>");
 84         output.println("<blockquote>" + defi + "</
    blockquote><hr>");
 85         output.println("<p> Return to ");
 86         output.println("<a href = \"index.html\">index</a></p>");
 87         output.println("</body> </html>");
 88         output.close();
 89     }
 90 
 91     /**
 92      * Returns a String containing HTML code for a definition,
    which will
 93      * contain linked terms to their respective HTML pages if the
    definition
 94      * contains terms in the glossary file.
 95      *
 96      * @param def
 97      *            definition to be used
 98      * @param termSet
 99      *            set containing the terms that appear in the
    glossary
100      * @requires termSet != null
101      * @return the HTML code for a definition
102      * @ensures String that is returned is valid to be used in an
    HTML file and
103      *          the returned definition will contained linked words
    if that word
104      *          featured in the definition is a term in the
    glossary
105      */
106     public static String getDef(Set<String> termSet, String def) {
107         final String separatorStr = " \t, \n";
108         //separators used for the next method call
109         String definition = def;
110         Set<Character> separatorSet = new Set1L<>();
111         generateElements(separatorStr, separatorSet); //makes set
    of separators
112         int position = 0;
113         while (position < def.length()) {
114             String temp = nextWordOrSeparator(def, position,
    separatorSet);
115             //checks if the string returned is a term, linking the
    term in the
```

```
116             //definition string with the term's own html page
    before it is returned
117             if (termSet.contains(temp)) {
118                 int tempIndex = definition.indexOf(temp);
119                 String linkedTerm = "<a href = \"" + temp +
    ".html\">" + temp
120                         + "</a>";
121                 String defSub1 = definition.substring(0,
    tempIndex);
122                 String defSub2 = definition.substring(tempIndex +
    temp.length(),
123                         definition.length());
124                 definition = defSub1 + linkedTerm + defSub2;
125                 //Splits original string up at the point of the
    term found within the
126                 //definition and concatenates it back together with
    the specific term
127                 //linked within the returned string
128             }
129             position += temp.length();
130             //makes sure the next word/separator is the one used in
    the next method call
131         }
132         return definition;
133     }
134
135     /**
136      * Returns the String of lines from an input file in the form
    of a string.
137      *
138      * @param input
139      *            source of strings, one per line
140      * @return String of lines read from {@code input}
141      * @requires input.is_open
142      * @ensures <pre>
143      * input.is_open  and  input.content = <>
144      * </pre>
145      */
146     public static Sequence<String> linesFromInput(SimpleReader
    input) {
147         assert input != null : "Violation of: input is not null";
148         assert input.isOpen() : "Violation of: input.is_open";
149         Sequence<String> returned = new Sequence1L<>();
150         String line = "";
151         int i = 0; //position for Sequence object to add strings to
```

```
    it
152         //draws new lines until end of stream is reached
153         while (!input.atEOS()) {
154             line = input.nextLine();
155             returned.add(i, line);
156             i++;
157         }
158 
159         return returned;
160     }
161 
162     /**
163      * Generates the set of characters in the given {@code String}
    into the
164      * given {@code Set}.
165      *
166      * @param str
167      *            the given {@code String}
168      * @param charSet
169      *            the {@code Set} to be replaced
170      * @requires charSet != null and str != null
171      * @ensures charSet = entries(str)
172      */
173     public static void generateElements(String str, Set<Character>
    charSet) {
174         assert str != null : "Violation of: str is not null";
175         assert charSet != null : "Violation of: charSet is not
    null";
176         Set<Character> temp1 = new Set1L<>();
177         for (int i = 0; i < str.length(); i++) {
178             char temp = str.charAt(i);
179             if (!temp1.contains(temp)) {
180                 temp1.add(temp);
181             }
182             //adds all characters of a given string to a set of
    characters
183         }
184         charSet.transferFrom(temp1); //restores the original
    charSet set object
185     }
186 
187     /**
188      * Returns the first "word" (maximal length string of
    characters not in
189      * {@code separators}) or "separator string" (maximal length
```

```
    string of
190      * characters in {@code separators}) in the given {@code text}
    starting at
191      * the given {@code position}.
192      *
193      * @param text
194      *            the {@code String} from which to get the word or
    separator
195      *            string
196      * @param position
197      *            the starting index
198      * @param separators
199      *            the {@code Set} of separator characters
200      * @return the first word or separator string found in {@code
    text} starting
201      *         at index {@code position}
202      * @requires 0 <= position < |text|
203      * @ensures <pre>
204      * nextWordOrSeparator =
205      *   text[position, position + |nextWordOrSeparator|)  and
206      * if entries(text[position, position + 1)) intersection
    separators = {}
207      * then
208      *   entries(nextWordOrSeparator) intersection separators = {}
    and
209      *   (position + |nextWordOrSeparator| = |text|  or
210      *    entries(text[position, position + |nextWordOrSeparator| +
    1))
211      *      intersection separators /= {})
212      * else
213      *   entries(nextWordOrSeparator) is subset of separators  and
214      *   (position + |nextWordOrSeparator| = |text|  or
215      *    entries(text[position, position + |nextWordOrSeparator| +
    1))
216      *      is not subset of separators)
217      * </pre>
218      */
219     public static String nextWordOrSeparator(String text, int
    position,
220             Set<Character> separators) {
221         assert text != null : "Violation of: text is not null";
222         assert separators != null : "Violation of: separators is
    not null";
223         assert 0 <= position : "Violation of: 0 <= position";
224         assert position < text.length() : "Violation of: position <
```

```
    |text|";
225         String returned = "";
226         char pos = text.charAt(position);
227         boolean contain = separators.contains(pos);
228         String temp = text.substring(position);
229         int i = 0;
230         //this indicates a word so it takes each non-separator
    character and
231         //concatenates them all into one string
232         if (!contain) {
233             while (!contain && i < temp.length()) {
234                 contain = separators.contains(temp.charAt(i));
235                 if (!contain) {
236                     returned = returned + temp.charAt(i);
237                 }
238                 i++;
239             }
240         } else { //the case where the String is a separator
241             i = 0;
242             while (contain && i < temp.length()) {
243                 contain = separators.contains(temp.charAt(i));
244                 if (contain) {
245                     returned = returned + temp.charAt(i);
246                 }
247                 i++;
248             }
249         }
250         return returned;
251     }
252 
253     /**
254      * Returns a set that contains all of the terms featured in the
    input file
255      * of the glossary.
256      *
257      * @param lines
258      *            The sequence object containing all of the lines
    drawn from the
259      *            input file
260      * @return A set containing all of the terms to be used in the
    glossary
261      * @requires lines != null and that the sequence being used was
    created from
262      *           a valid glossary input file as described in the
    project
```

```
263     *           description for Project 10
264     * @ensures The set returned contains all of the terms to be
    featured in the
265     *          glossary
266     */
267
268    public static Set<String> getTermSet(Sequence<String> lines) {
269        Set<String> terms = new Set1L<>();
270        int linesLength = lines.length();
271        for (int i = 0; i < linesLength; i++) {
272            String temp = lines.entry(i);
273            /*
274             * the lack of a space and empty string indicates that
    the string is
275             * a term because the project description indicates
    that a term from
276             * the input file is defined as a single word, meaning
    there are no
277             * spaces or empty string in a line from the input file
    that
278             * contains a term
279             */
280            if (!temp.contains(" ") && !temp.equals("")) {
281                terms.add(temp);
282            }
283        }
284        return terms;
285    }
286
287    /**
288     * Returns an array of strings that contains all of the terms
    from the input
289     * file in alphabetical order.
290     *
291     * @param terms
292     *            The set object containing all of the terms to be
    featured in
293     *            the glossary
294     * @return An array of strings (words) in alphabetical order
295     * @requires terms != null, that the set object contains all of
    the terms
296     *          featured in the input file, and that the length of
    the set of
297     *          terms is greater than 1
298     * @ensures The terms will be ordered in the array in
```

```
    alphabetical order
299      */
300     public static String[] alphabeticalTerms(Set<String> terms) {
301         int numTerms = terms.size();
302         Set<String> terms1 = new Set1L<>();
303         String[] orderedTerms = new String[numTerms];
304         for (int i = 0; i < numTerms; i++) {
305             orderedTerms[i] = terms.removeAny();
306             terms1.add(orderedTerms[i]);
307         }
308         // sorting loop that sorts all of the terms in the array in
    alphabetical
309         // order
310         for (int k = 1; k < orderedTerms.length; k++) {
311             for (int p = 0; p < orderedTerms.length - k; p++) {
312                 String s1 = orderedTerms[p];
313                 String s2 = orderedTerms[p + 1];
314                 int comp1 = s1.compareTo(s2);
315                 int comp2 = s2.compareTo(s1);
316                 //compares two strings in lexicographical order
317                 // v means the two words are out of alphabetical
    order so it
318                 //swaps the terms in the array
319                 if (comp2 < comp1) {
320                     orderedTerms[p] = s2;
321                     orderedTerms[p + 1] = s1;
322                 }
323             }
324         }
325         terms.transferFrom(terms1);
326         return orderedTerms;
327     }
328 
329     /**
330      * Returns a string containing the a term from the input file
    paired with
331      * its definition (term and definition are separated by a
    space) based on
332      * the position index value provided through arguments.
333      *
334      * @param lines
335      *            The sequence object containing all of the lines
    from the input
336      *            file
337      * @param position
```

```
338      *            The starting position index value from which the
    method will
339      *            retrieve sequence entries from
340      * @return A string containing a term paired with its
    definition separated
341      *         by a space
342      * @updates position
343      * @requires The value of position is not greater than the
    length of the
344      *           position object and that lines != null
345      * @ensures The string return contains the proper term-
    definition pair
346      *          separated by a space and that the pair returned is
    correct based
347      *          on the provided position value
348      */
349
350     public static String termDefPair(Sequence<String> lines,
351             NaturalNumber position) {
352         String pair = "";
353         String temp = "jp";
354         // jp is simply a filler for the string so the method runs
    as intended
355         NaturalNumber one = new NaturalNumber1L(1);
356         int pos = position.toInt();
357         //loop runs until an empty string is hit because a valid
    input file
358         //terminates the definition with an empty string
359         while (!temp.equals("") && pos < lines.length()) {
360             temp = lines.entry(pos);
361             position.add(one);
362             pos++;
363             pair = pair + temp + " ";
364             // makes sure that the term and definition are
    separated by a space
365         }
366         return pair;
367     }
368
369     /**
370      * Main method.
371      *
372      * @param args
373      *            the command line arguments
374      */
```

```
375     public static void main(String[] args) {
376         SimpleReader in = new SimpleReader1L();
377         SimpleWriter out = new SimpleWriter1L();
378         out.print("Enter an input file name for the glossary: ");
379         String fileInput = in.nextLine();
380         out.print("\n\nEnter an output folder name destination for
    the "
381                 + "output HTML files: ");
382         String folderOut = in.nextLine();
383         SimpleWriter output = new SimpleWriter1L(folderOut + "/
    index.html");
384         SimpleReader input = new SimpleReader1L(fileInput);
385         //creates a sequence of strings of all of the lines from
    the input file
386         Sequence<String> linesSeq = linesFromInput(input);
387         // Gets all of the terms from the sequence object created
    above
388         Set<String> terms = getTermSet(linesSeq);
389         // Creates an array that stores the terms in alphabetical
    order
390         String[] orderedTerms = alphabeticalTerms(terms);
391         // Natural Numbers used to be updated by the termDefPair
    method for
392         // position referencing within the lineSeq object
393         NaturalNumber position = new NaturalNumber1L(0);
394         int pos = position.toInt();
395         //used to eventually terminate loop when the end of the
    lineSeq object
396         //is reached
397         while (pos < linesSeq.length()) {
398             String pair = termDefPair(linesSeq, position);
399             int termIndex = pair.indexOf(" ");
400             //space separated word from definition
401             String term = pair.substring(0, termIndex);
402             String definition = pair.substring(termIndex + 1,
    pair.length());
403             termHTML(term, definition, folderOut, terms);
404             pos = position.toInt(); //updates value of position
    based on updated
405             //Natural number position value that is referenced in
    the object
406         }
407         generateHomepage(output, orderedTerms); //generates index
    HTML page
408         in.close();
```

```
409         out.close();
410         input.close();
411         output.close();
412     }
413
414 }
415
```