```java
import java.util.Comparator;

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

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

    /**
     *
     * @author Nicholas Cheong
     *
     */
    private static class StringLT implements Comparator<String> {
        @Override
        public int compare(String o1, String o2) {
            return o1.compareTo(o2);
        }
    }

    /**
     * Inputs a list of words and their definitions from the given file and
     * stores them in the given {@code Map}.
     *
     * @param fileName
     *            the name of the input file
     * @param wordDefinition
     *            the word and definition -> word and definition map
     * @replaces wordDefinition
     * @requires <pre>
     * [file named fileName exists but is not open, and has the
     *  format of one "word" (unique in the file) and one
```

```
definition, with
     *  word and definition separated by ' '
     * </pre>
     * @ensures [wordDefinition contains word and definition ->
mapping from
     *          file fileName]
     */
    public static void getMap(String fileName,
            Map<String, String> wordDefinition) {

        // reads every line in the file
        // reads the first line after the space to be the word
(key)
        // reads every line after to be the corresponding
definition (value)
        SimpleReader in = new SimpleReader1L(fileName);
        while (!in.atEOS()) {
            String thisLine = in.nextLine();
            String word = thisLine;
            String definition = "";
            String nextLine = in.nextLine();
            while (!nextLine.equals("")) {
                definition += nextLine + " ";
                nextLine = in.nextLine();
            }

            wordDefinition.add(word, definition);
        }

        // close output
        in.close();

    }

    /**
     *
     * @param wordDefinition
     *            the word and definition -> word and
definition map
     * @param wordBank
     *            the word -> word queue
     * @replaces wordBank
     * @ensures [wordBank contains all words from
wordDefinition map]
```

```java
     */

    public static void wordQueue(Map<String, String> wordDefinition,
            Queue<String> wordBank) {

        Comparator<String> cs = new StringLT();

        // enqueue each word in every pair of the map to the wordBank queue
        for (Map.Pair<String, String> p : wordDefinition) {
            wordBank.enqueue(p.key());
        }

        wordBank.sort(cs);

    }

    /**
     * Generates the set of characters in the given {@code String} into the
     * given {@code Set}.
     *
     * @param str
     *            the given {@code String}
     * @param charSet
     *            the {@code Set} to be replaced
     * @replaces charSet
     * @ensures charSet = entries(str)
     */
    public static void generateElements(String str, Set<Character> charSet) {
        assert str != null : "Violation of: str is not null";
        assert charSet != null : "Violation of: charSet is not null";

        Set<Character> tempSet = new Set1L<>();

        for (int i = 0; i < str.length(); i++) {
            if (!tempSet.contains(str.charAt(i))) {
                tempSet.add(str.charAt(i));
            }

        }
        charSet.transferFrom(tempSet);

    }

    /**
     * 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} from which to get the word or separator
     *            string
     * @param position
     *            the starting index
     * @param separators
     *            the {@code Set} of separator characters
     * @return the first word or separator string found in {@code text} starting
     *         at index {@code 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)
```

```java
     * </pre>
     */
    public static String nextWordOrSeparator(String text, int position,
            Set<Character> separators) {
        assert text != null : "Violation of: text is not null";
        assert separators != null : "Violation of: separators is not null";
        assert 0 <= position : "Violation of: 0 <= position";
        assert position < text.length() : "Violation of: position < |text|";

        String result = "";
        int positionCopy = position;

        if (!separators.contains(text.charAt(positionCopy))) {
            while (positionCopy < text.length()
                    && !separators.contains(text.charAt(positionCopy))) {

                result += text.charAt(positionCopy);
                positionCopy++;
            }
        } else {
            while (positionCopy < text.length()
                    && separators.contains(text.charAt(positionCopy))) {
                result += text.charAt(positionCopy);
                positionCopy++;
            }
        }

        return result;
    }

    /**
     * Outputs the "opening" tags in the generated HTML file. These are the
     * expected elements generated by this method:
     *
     * <html> <head> <title>Glossary</title> </head> <body>
     * <h2>Glossary</h2>
     * <hr>
     * <h3>Index</h3>
     * <ul>
     *
     * @param out
     *            the output stream
     * @updates out.content
     * @ensures out.content = #out.content * [the HTML "opening" tags]
     */
    public static void generateIndex(SimpleWriter out) {

        out.println("<html>");
        out.println("<head>");
        out.println("<title>");
        out.println("Glossary");
        out.println("</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>Glossary</h2>");
        out.println("<hr>");
        out.println("<h3>Index</h3>");
        out.println("<ul>");
    }

    /**
     * Outputs each word in the index HTML and outputs their definitions from
     * wordDefinition map to their own generated HTML file. Will also link to
     * other definitions if word in glossary is used in the definition of
     * another word
     *
     * @param wordDefinition
     *            the word and definition -> word and definition map
     * @param word
     *            the word imported from wordBank queue
     * @param definition
     *            the corresponding definition imported from wordDefinition map
     * @param out
     *            the output stream
     * @param outputFile
     *            the folder where HTML will be stored
     * @updates out.content
     * @ensures out.content = #out.content * word and HTMLS of each word with
     *          corresponding definitions
     *
     */
    public static void generateHTML(Map<String, String> wordDefinition,
            String word, String definition, SimpleWriter out,
            String outputFile) {

        // creating separator set
        String separatorStr = " \t,";
        Set<Character> separatorSet = new Set1L<>();
        generateElements(separatorStr, separatorSet);

        SimpleWriter outFile = new SimpleWriter1L(
                outputFile + "/" + word + ".html");
        String newDefinition = "";
        String linked = "";
        int i = 0;
        // prints word to the index html
        out.println("<a href = " + word + ".html>");
        out.println(word);
        out.println("</a>");

        // prints the definition to the word html
        outFile.println("<head>");
        outFile.println("<title>");
        outFile.println(word);
        outFile.println("</title>");
        outFile.println("</head>");
        outFile.println("<body>");
        outFile.println("<h2>");
        outFile.println("<b>");
        outFile.println("<i>");
        outFile.println("<font color = \"red\">" + word + "</font>");
        outFile.println("</i>");
        outFile.println("</b>");
        outFile.println("</h2>");
        outFile.println("<blockquote>");

        while (i < definition.length()) {
```

```java
            String link = nextWordOrSeparator(definition, i, separatorSet);

            if (wordDefinition.hasKey(link)) {

                linked = "<a href = " + link + ".html>" + link + "</a>";
                newDefinition += linked;

            } else {
                newDefinition += link;
            }
            i += link.length();
        }

        outFile.println(newDefinition);

        outFile.println("</blockquote>");
        outFile.println("<hr>");
        outFile.println("<p>");
        outFile.println("Return to");
        outFile.println("<a href = index.html>");
        outFile.println("index</a>.");
        outFile.println("</p>");
        outFile.println("</body>");
        outFile.println("</html>");

        // closing outputs
        outFile.close();

    }

    /**
     * Generates list and pulls each word from wordBank queue and matches with
     * corresponding definition before printing it. These expected elements
     * generated by this method:
     *
     * <li>word</li>
     *
     * @param wordBank
     *            the word -> word queue
     * @param wordDefinition
     *            the word and definition -> word and definition map
     * @param out
     *            the output stream
     * @param outputFile
     *            the folder where HTML will be stored
     * @ensures out.content = #out.content * list of words
     *
     */
    public static void generateWord(Queue<String> wordBank,
            Map<String, String> wordDefinition, SimpleWriter out,
            String outputFile) {

        String word = wordBank.dequeue();
        String definition = wordDefinition.value(word);
        out.println("<li>");
        generateHTML(wordDefinition, word, definition, out, outputFile);
        out.println("</li>");
        wordBank.enqueue(word);
    }

    /**
     * Outputs the "closing" tags in the generated HTML file. These are the
     * expected elements generated by this method:
     *
     * </ul>
     * </body> </html>
     *
     * @param out
     *            the output stream
     * @updates out.content
     * @ensures out.content = #out.content * [the HTML "closing" tags]
     */
    public static void generateCloser(SimpleWriter out) {
        out.println("</ul>");
        out.println("</body>");
        out.println("</html>");
    }

    /**
```

```
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        // prompts user for input and output files
        out.println("What is the name of the input file?");
        String newName = in.nextLine();

        out.println("Where do you want to store output
files?");
        String outputFile = in.nextLine();

        SimpleWriter outFile = new SimpleWriter1L(outputFile +
"/index.html");

        // initializes wordDefinition map and wordBank queue
        Map<String, String> wordDefinition = new Map1L<>();
        Queue<String> wordBank = new Queue1L<>();

        // gets map and queue
        // sorts queue in alphabetical
        getMap(newName, wordDefinition);
        wordQueue(wordDefinition, wordBank);

        // generating HTML files
        generateIndex(outFile);
        for (int i = 0; i < wordBank.length(); i++) {
            generateWord(wordBank, wordDefinition, outFile,
outputFile);
        }
        generateCloser(outFile);

        // closing outputs
        in.close();
        out.close();
        outFile.close();
    }

}
```