```java
import java.util.Comparator;

import components.map.Map;
import components.map.Map1L;
import components.queue.Queue;
import components.queue.Queue1L;
import components.set.Set;
import components.set.Set1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Creating a glossary facility for a client.
 *
 * @author Gabe Azzarita
 */
public final class Glossary {

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

    /**
     * Compare {@code String}s in lexicographic order.
     */
    private static class StringLT implements Comparator<String> {
        @Override
        /**
         * @param o1
         *            first string
         * @param o2
         *            second string
         * @ensures positive int, zero, or negative int if o1 is greater than,
         *          equal to, or less than o2
         * @return integer signaling which string is bigger
         */
        public int compare(String o1, String o2) {
            return o1.compareTo(o2);
        }
    }

    /**
     * Adds all elements in inputFile (words and definitions) into a map, and
     * adds all words in a queue.
     *
     * @param pairMap
     *            Map of words, definitions
     * @param wordQueue
     *            ArrayList of words
     * @param inputFile
     *            SimpleReader for file
     * @ensures wordQueue and pairMap are filled with words and words + def
     *
     */
    public static void getElements(Map<String, String> pairMap,
            Queue<String> wordQueue, SimpleReader inputFile) {

        String tempWord = "";
        String tempDef = "";
        String tempLine = "";

        /*
         * Add words and definitions into a map until end of file Add words to a
         * queue to sort later
         */
        while (!inputFile.atEOS()) {
            /*
             * File follows word + \n + term + \n + \n repeat, so we have to
             * account and ignore the empty line
             */
            tempLine = inputFile.nextLine();
            if (!tempLine.equals("")) {
                tempWord = tempLine;
                tempDef = inputFile.nextLine();
                tempLine = inputFile.nextLine();

                // This is needed in case definitions are multiple lines
                while (!tempLine.equals("")) {
                    tempDef += tempLine;
                    tempLine = inputFile.nextLine();
                }

                // Add word to queue and word + definition to map
                wordQueue.enqueue(tempWord);
                pairMap.add(tempWord, tempDef);
            }
        }
    }

    /**
     * Outputs the "opening" tags in the generated HTML file.
     *
     * @param wordQueue
     *            sorted wordQueue
     * @param out
     *            the output stream
     * @ensures out.content = #out.content * [the HTML "opening" tags]
```

```java
     */
    public static void outputHeader(Queue<String> wordQueue, SimpleWriter out) {
        out.println("<html>");
        out.println("  <head>");
        out.println("    <title> Sample Glossary </title>");
        out.println("  </head>");
        out.println("  <body>");
        out.println("    <h2> Sample Glossary </h2>");
        out.println("<hr>");
        out.println("    <h3> Index </h3>");
        out.println("    <ul>");

        Queue<String> wordQueueCopy = new Queue1L<>();
        // Print each word in wordQueue with link to its page
        while (wordQueue.length() != 0) {
            String tempWord = wordQueue.dequeue();
            wordQueueCopy.enqueue(tempWord);
            out.println("      <li> <a href=" + tempWord + ".html" + ">"
                    + tempWord + "</a> </li>");
        }
        wordQueue.transferFrom(wordQueueCopy);
    }

    /**
     * Processes one word and definition and outputs it to a corresponding HTML.
     *
     * @param word
     *
     * @param def
     *            definition associated with word
     * @param out
     *            output stream
     * @param separators
     *            the {@code Set} of separator characters
     * @param pairMap
     *            Map of words, definitions
     * @ensures out.content = #out.content * [an HTML page with the word, its
     *          definition, any linked words, and a return to index page option]
     *
     */
    public static void processItem(String word, String def, SimpleWriter out,
            Set<Character> separators, Map<String, String> pairMap) {
        out.println("<html>");
        out.println("  <head>");
        out.println("    <title>" + word + "</title>");
        out.println("  </head>");
        out.println("  <body>");
        out.println("    <h2><b><i><font color=\"#ff0000\">" + word
                + "</font></i></b></h2>");
        out.print("    <p>");
        // Printing text using nextWordOrSeparator
        int position = 0;
        while (position < def.length()) {
            String token = nextWordOrSeparator(def, position, separators);
            // Checking to see if word has a linked definition
            if (pairMap.hasKey(token)) {
                out.print("<a href=\"" + token + ".html" + "\">" + token
                        + "</a>");
            } else {
                out.print(token);
            }
            position += token.length();
        }

        out.println("    </p>");
        out.println("<hr>");
        out.println("    <p> Return to <a href=\"" + "index.html" + "\">"
                + "index" + "</a></p>");
        out.println("  </body>");
    }

    /**
     * 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 resultStr = "";
    String subStr = text.substring(position);
    char ch = text.charAt(position);
    boolean containsCh = separators.contains(ch);

    if (!containsCh) {
        // If first char is not separator, loop until separator is found
        for (int i = 0; i < subStr.length(); i++) {
            ch = text.charAt(position + i);
            if (!separators.contains(ch)) {
                resultStr += ch;
            } else { // As soon as next char is separator
                // Essentially a break
                i = text.substring(position).length();
            }
        }
    } else { // Separator found, return separator
        resultStr += ch;
    }
    return resultStr;
}

/**
 * Outputs the "closing" tags in the generated HTML file.
 *
 * @param out
 *           the output stream
 * @ensures out.content = #out.content * [the HTML "closing" tags]
 *
 */
public static void outputFooter(SimpleWriter out) {
    assert out != null : "Violation of: out is not null";
    assert out.isOpen() : "Violation of: out.is_open";

    out.println("    </ul>");
    out.println("  </body>");
    out.println("</html>");
}

/**
 * Main method.
 *
```

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

        // Grab input file and output folder and create reader/writer
        out.print("Name of input file: ");
        String fileName = in.nextLine();
        SimpleReader inputFile = new SimpleReader1L(fileName);
        out.print("Name of output folder: ");
        String outputFolder = in.nextLine();
        SimpleWriter outMain = new SimpleWriter1L(outputFolder + "/index.html");

        // Create map and queue and fill using getElements
        Map<String, String> pairMap = new Map1L<>();
        Queue<String> wordQueue = new Queue1L<>();
        getElements(pairMap, wordQueue, inputFile);

        // Create separator set and fill with necessary separators
        Set<Character> separators = new Set1L<>();
        separators.add(' ');
        separators.add(',');

        // Sort wordQueue in alphabetical order
        wordQueue.sort(new StringLT());

        outputHeader(wordQueue, outMain);
        // Process each item in the queue and create corresponding HTML page
        while (wordQueue.length() != 0) {
            String word = wordQueue.dequeue();
            String def = pairMap.value(word);
            SimpleWriter outWord = new SimpleWriter1L(
                    outputFolder + "/" + word + ".html");
            processItem(word, def, outWord, separators, pairMap);
            outWord.close();
        }
        outputFooter(outMain);

        // Close readers
        in.close();
        out.close();
    }
}
```