```java
import java.util.Comparator;

/**
 * Program create a glossary webpage from words and definitions from an input
 * file.
 *
 * @author Shyam Sai Bethina
 */
public final class Glossary {

    /**
     * Default constructor--private to prevent instantiation.
     */
    private Glossary() {
    }

    /**
     *
     * A comparator that orders the queue of words.
     *
     */

    private static class StringLT implements Comparator<String> {
        @Override
        /*
         * Compares two strings and returns them in alphabetical sequence, which
         * is used to order the words queue later on
         */
        public int compare(String one, String two) {
            return one.compareTo(two);
        }
    }

    /**
     * Gets the terms from the input file stream and returns a queue of the
     * terms.
     *
     * @param in
     *            The input file stream
     * @return A queue of terms from the input file
     * @ensures Returned Queue is filled with terms from the input
```

```java
file stream
     *
     */
    public static Queue<String> getTerms(SimpleReader in) {
        //Creates an empty queue to add in terms
        Queue<String> terms = new Queue1L<>();

        /*
         * While the input file stream is not at the end, it gets the next line
         * within the input, and if the is not empty and or does not contain a
         * space within the line, it is a term so it gets added into the queue
         */
        while (!in.atEOS()) {
            String term = in.nextLine();
            if (!(term.isEmpty() || term.contains(" "))) {
                terms.enqueue(term);
            }

        }

        return terms;
    }

    /**
     * Gets the definitions from the input file stream and returns a queue of
     * the definitions.
     *
     * @param in
     *            The input file stream
     * @return A queue of definitions from the input file
     * @ensures Returned Queue is filled with definitions from input file stream
     */
    public static Queue<String> getDefinitions(SimpleReader in) {
        //Creates an empty queue to add in definitions
        Queue<String> definitions = new Queue1L<>();

        /*
         * This loops keeps going if the input file stream is not at the end
         */
        while (!in.atEOS()) {
            /*
             * String definition will start out empty, and a test string of the
             * next line will be used
             */
            String definition = "";
            String testDefinition = in.nextLine();

            /*
             * If the test string contains a space, then it is part of a
             * definition, and then we set definition to testDefinition
             */
            if (testDefinition.contains(" ")) {
                definition = testDefinition;

                /*
                 * The while loop stops when the file stream is at the end or
                 * the next line is empty
                 */
                while (!testDefinition.equals("")) {
                    if (!in.atEOS()) {
                        testDefinition = in.nextLine();

                        /*
                         * If the next line is not empty and it is a sentence,
                         * it gets added to the definition string.
                         */
                        if (!testDefinition.isEmpty()
                                && testDefinition.contains(" ")) {
                            definition += " " + testDefinition;
                        } else {
                            /*
                             * If the next line is empty or it is not a
                             * sentence, then we go out of the loop
                             */
                            testDefinition = "";
```

```java
                        }
                    } else {
                        /*
                         * If we are at the end of the stream, then we go out of
                         * the loop
                         */
                        testDefinition = "";
                    }
                }
            }
            /*
             * If resulting definition value is not empty, then we add it to the
             * queue
             */
            if (!definition.isEmpty()) {
                definitions.enqueue(definition);
            }

        }

        return definitions;

    }

    /**
     * Creates an HTML page for each term.
     *
     * @param words
     *            The queue of words from input file
     * @param definitions
     *            The queue of definitions from input file
     * @param folderLocation
     *            The name of the folder location that user inputed
     * @requires <pre>
     * |words| != 0
     * |definitions| != 0
     * </pre>
     * @ensures The HTML page for each term as the term bold-faced and red, has
     *          the definition with linked terms, and has a link to return to
     *          the index page
     */
    public static void pageForWords(Queue<String> words,
            Queue<String> definitions, String folderLocation) {
        /*
         * For every word in the words queue, this loop creates a separate HTML
         * page
         */
        for (int i = 0; i < words.length(); i++) {
            /*
             * The words and its corresponding definition get dequeues from the
             * queues. Then a new output file stream gets created and creates a
             * new HTML file using the words and folder location which is based
             * on what the user inputted.
             */
            String word = words.dequeue();
            String definition = definitions.dequeue();
            SimpleWriter out = new SimpleWriter1L(
                    folderLocation + "/" + word + ".html");
            out.println("<html>");
            out.println("   <head>");

            //The title is the term
            out.println("       <title>" + word + "</title>");
            out.println("   </head>");
            out.println("   <body>");
            out.println("       <h2>");

            //The word is bold-faced and gets a red color
            out.println("           <b><i><font color='red'>" + word
                    + "</font></i></b>");
            out.println("       </h2>");

            /*
             * The String temp becomes the definition sentence, but each term
             * within the sentence is linked to it's respective page
             */
            String temp = linkWordsinDefinition(words, definition);

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

            //Closes the output file stream
            out.close();
            /*
             * Enqueues the word and definition to move onto the next words and
             * definition
             */
            words.enqueue(word);
            definitions.enqueue(definition);
        }

    }

    /**
     * Outputs the header for the base index HTML file.
     *
     * @param out
     *            The output file stream
     * @requires out.is_open
     * @ensures output file has the header for the index HTML file
     */
    public static void outputHeader(SimpleWriter out) {
        /*
         * Outputs the beginning code of the index HTML file to the output file
         * stream
         */
        out.println("<html>");
        out.println("   <head>");
        out.println("       <title>Glossary</title>");
        out.println("   </head>");
        out.println("   <body>");
        out.println("       <h2>Glossary</h2>");
        out.println("       <hr />");
        out.println("       <h3>Index</h3>");
        out.println("       <ul>");
    }

    /**
     * Outputs the footer for the base index HTML file.
     *
     * @param out
     *            The output file stream
     * @requires out.is_open
     * @ensures output file has the closing braces for the index HTML file
     */
    public static void outputFooter(SimpleWriter out) {
        /*
         * Outputs the closing code of the index HTML file to the output file
         * stream
         */
        out.println("       </ul>");
        out.println("   </body>");
        out.print("</html>");
    }

    /**
     * Creates an ordered, bullet-pointed list of the terms, and the terms are
     * linked to their respective HTML pages.
     *
     * @param out
     *            The output file stream
     * @param words
     *            The queue of words from the input file
     * @param definitions
     *            The queue of definitions from the input file
     * @requires <pre>
     * out.is_open
     * |words| != 0
     * |definitions| != 0
     * </pre>
     * @ensures An ordered list of terms in the index HTML file that is linked
     *           to their respective pages
     */
    public static void listForWords(SimpleWriter out, Queue<String> words,
            Queue<String> definitions) {
        /*
         * Orders the words in the queue by alphabetical order
         */
        Comparator<String> order = new StringLT();
        words.sort(order);

        /*
         * Each word in the queue gets linked to it's respective HTML page
         */
        for (int i = 0; i < words.length(); i++) {
            /*
             * Each word gets dequeued and enqueued after being linked to the
             * page
             */
            String word = words.dequeue();
            out.println("           <li><a href=" + word + ".html>" + word
                    + "</a></li>");
            words.enqueue(word);
        }
    }

    /**
     * Goes through each word in the definition and determines if the word is a
     * term, if it is, then the word is linked to the term HTML page. Then
     * returns the completed sentence.
     *
     * @param words
     *             The queue of words from the input file
     * @param definition
     *             A string representing the definition of a term
     * @return A string of the definition that links the terms within the
     *          sentence if there are any terms in it
     * @requires <pre>
     * |words| != 0
     * |definition| != 0
     * </pre>
     * @ensures A string of the definition where the terms are
linked to their
     *          pages
     */
    public static String linkWordsinDefinition(Queue<String>
words,
            String definition) {
        /*
         * Define all possible separator characters
         */
        final String separatorStr = " \t,!.?(){}[];:'";
        Set<Character> separatorSet = new Set1L<>();
        generateElements(separatorStr, separatorSet);

        //wordsAndSep is a queue with only words and only
separators
        Queue<String> wordsAndSep =
nextWordOrSeparator(definition,
                separatorSet);

        String result = "";
        for (int i = 0; i < wordsAndSep.length(); i++) {
            /*
             * For each element in wordsAndSep, temp becomes a
temporary string
             * of that element. Then it gets compared to each term
in the words
             * queue.
             */
            String temp = wordsAndSep.dequeue();
            for (String word : words) {
                /*
                 * If the word does equal a term, then temp
becomes a linked
                 * word
                 */
                if (temp.equals(word)) {
                    temp = "<a href=" + word + ".html>" + word +
"</a>";
                }

```

```java
            }
            //Restores wordsAndSep
            wordsAndSep.enqueue(temp);

            /*
             * Result becomes the previous sentence plus the temp string and a
             * whitespace so that the words don't become one whole word.
             */
            result = result + temp;

        }

        return result;
    }

    /**
     * Generates the set of characters in the given {@code String} into the
     * given {@code Set}.
     *
     * @param str
     *            the given String
     * @param charSet
     *            the Set to be replaced
     * @replaces charSet
     * @requires <pre>
     * |str| != 0
     * |charSet| = 0
     * </pre>
     * @ensures charSet = characters of str
     */
    public static void generateElements(String str, Set<Character> charSet) {

        /*
         * Goes through each character of the string and adds the non-duplicates
         * to the set
         */
        for (int i = 0; i < str.length(); i++) {
            char charTemp = str.charAt(i);
            if (!charSet.contains(charTemp)) {
                charSet.add(charTemp);
            }
        }

    }

    /**
     * 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 separators
     *            the {@code Set} of separator characters
     * @return Queue with only separators and only words
     * @requires 0 <= position < |text|
     * @ensures <pre>
     * The returned Queue will have separators and words, but not words with separators
     * </pre>
     */
    public static Queue<String> nextWordOrSeparator(String text,
            Set<Character> separators) {
        Queue<String> result = new Queue1L<>();

        //Indexes to get the substring of words or separators
        int firstIndex = 0;
        int secondIndex = 0;
        while (firstIndex < text.length()) {
            String subString;
            /*
             * If the character at firstIndex is a separator, then subString
             * will equal the string with only separators until the character is
             * a letter. If the character at firstIndex is a letter, then
```

```java
             * subString will equals the string with only letter until character
             * is a separator
             */
            if (separators.contains(text.charAt(firstIndex))) {
                while (secondIndex < text.length()
                        && separators.contains(text.charAt(secondIndex))) {
                    secondIndex++;
                }
            } else {
                while (secondIndex < text.length()
                        && ! separators.contains(text.charAt(secondIndex))) {
                    secondIndex++;
                }
            }

            /*
             * Enqueues the separator or word subString to Queue result, and
             * firstIndex will equal to secondIndex in order reset the count
             */
            subString = text.substring(firstIndex, secondIndex);
            result.enqueue(subString);
            firstIndex = secondIndex;
        }

        return result;

    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments; unused here
     */
    public static void main(String[] args) {
        /*
         * Creates input file stream for user input and output file stream to
         * ask questions
         */
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        /*
         * Gets the desired location and name from user, and inputName becomes
         * the answer
         */
        out.println("Enter location and name of input file: ");
        String inputName = in.nextLine();

        /*
         * This input file stream reads the input file using the name and
         * location the user inputed
         */
        SimpleReader inputFile = new SimpleReader1L(inputName);

        /*
         * Asks for the name of the output folder, and folderName becomes the
         * answer
         */
        out.println("Enter name of output folder: ");
        String folderName = in.nextLine();

        /*
         * This output file stream creates a new index HTML file in the folder
         * location the user wanted
         */
        SimpleWriter outLocation = new SimpleWriter1L(
                folderName + "/index.html");

        /*
         * Queue words is filled up with the terms from the input file, and
         * inputFile stream is closed because it is not needed anymore
         */
        Queue<String> words = getTerms(inputFile);
        inputFile.close();

        //CHECK THIS, do we need a second reader
        /*
         * Creates a new input file stream of the same input file to start at
         * the beginning of the file, and Queue definitions is filled up with
         * definitions from the input file
         */
        SimpleReader inputFile2 = new SimpleReader1L(inputName);
        Queue<String> definitions = getDefinitions(inputFile2);

        /*
         * Outputs the header for the index HTML file, creates the pages for the
         * terms, outputs ordered list of the terms to the index file, and
         * outputs the footer for the index HTML file all in the outLocation
         * output file stream
         */
        outputHeader(outLocation);
        pageForWords(words, definitions, folderName);
        listForWords(outLocation, words, definitions);
        outputFooter(outLocation);

        /*
         * Closes all the input and output file stream that was used
         */
        in.close();
        out.close();

        inputFile2.close();
        outLocation.close();

    }

}
```