```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;

/**
 * Creates a well formatted HTML page for a glossary text file.
 *
 * @author Joe Fong
 *
 */
public final class Glossary {

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

    }

    /**
     * Compare {@code String}s in lexicographic order.
     */
    private static class OrderAlpha implements Comparator<String> {
        @Override
        public int compare(String str1, String str2) {
            return str1.compareTo(str2);
        }
    }

    /**
     * Searches text file and returns terms and definitions in a queue while
     * updating the {@code map}
     *
     * @param in
     *            the input stream
     * @param termsAndDefinitions
     *            Map connecting each term to its definition
     * @updates termsAndDefinitions
     * @return queue of terms read from {@code in}
     * @requires input.is_open
     * @ensures <pre>
     * input.is_open  and  input.content = <>  and
     * termsAndDefinitions = #termsAndDefinitions *
     * [term keys, definition values]
     * </pre>
     */
    public static Queue<String> collectTerms(SimpleReader in,
            Map<String, String> termsAndDefinitions) {

        Queue<String> terms = new Queue1L<String>();
        String str = "", definition = "", tempDefinition = "something";

        /*
         * while loop reads in each line
         */
        while (!in.atEOS()) {
            /*
             * term added to queue
             */
            str = in.nextLine();
            terms.enqueue(str);
            /*
             * Definition added by line until blank space using tempDef. First
             * line read outside of loop to avoid extra spaces when the
             * definition has multiple lines
             */
            tempDefinition = in.nextLine();
            definition = definition.concat(tempDefinition);
            while (!tempDefinition.equals("") && !in.atEOS()) {
                tempDefinition = in.nextLine();
                if (!tempDefinition.equals("") && !in.atEOS()) {
                    definition = definition.concat(" ");
                }
                definition = definition.concat(tempDefinition);

            }
            /*
             * term and definition mapped together
             */
            termsAndDefinitions.add(str, definition);
            /*
             * resets variables for loop
             */
            definition = "";
            tempDefinition = "something";
        }
        /*
         * Returns the queue containing the terms
         */
        return terms;
    }

    /**
     * Outputs the header and body for the index file containing the terms and
     * their links.
     *
     * <html> <head> <title>index.html</title> </head> <body>
     * <h1>Glossary</h1>
     * <h3>Index</h3>
     * <ul>
     * list items
     * </ul>
     * </body> </html>
     *
     * @param out
     *            the output stream
     * @param terms
     *            the terms of the glossary
     * @updates out.content
     * @requires out.is_open
     * @ensures out.content = #out.content * [the HTML file]
     */
    public static void outputIndex(SimpleWriter out, Queue<String> terms) {
        /*
         * creates variables for length and the out put terms
         */
        int length = terms.length(), i = 0;
        String term = "";
        /*
         * outputs html header
         */
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Glossary</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Glossary</h1>");
        out.println("<h3>Index</h3>");
        out.println("<ul>");
        /*
         * while loop used to print out list of linked terms
         */
        while (i < length) {
            term = terms.front();
            terms.rotate(1);
            out.println("<li>");
            out.println("<a href=\"" + term + ".html" + "\">" + term + "</a>");
            out.println("</li>");
            i++;
        }
        out.println("</ul>");
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * Outputs the header and body for term files containing the term and its
     * definition.
     *
     * <html> <head> <title>index.html</title> </head> <body>
     * <h1>Glossary</h1>
     * <h3>Index</h3>
     * <p>
     * definition
     * </p>
     * <p>
     * return to index
     * </p>
     * </body> </html>
     *
     * @param out
     *            the output stream
     * @param termsMap
     *            the terms mapped to their definition
     * @param term
     *            the term of the glossary
     * @param separators
     *            the set of separating chars
     * @param termSet
     *            the set of terms
     * @updates out.content
     * @requires out.is_open
     * @ensures out.content = #out.content * [the HTML file]
     */
    public static void outputTermHTML(SimpleWriter out,
            Map<String, String> termsMap, String term,
            Set<Character> separators, Set<String> termSet) {
        /*
         * outputs html header
         */
        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + term + "</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2><b><i><font color=\"red\">" + term
                + "</font></i></b></h2>");
        out.println("<p>");
        /*
         * sets variables for definition and the searched word
         */
        String definition = termsMap.value(term), word = "";
        /*
         * While loop runs while i < the length of the definition. Loop searches
         * for words in the definition and prints out what is returned. However
         * if a word is another term in the glossary it is linked.
         */
        int length = definition.length();
        int i = 0;
        while (i < length) {
            word = nextWordOrSeparator(definition, i, separators);
            i += word.length();
            if (termSet.contains(word)) {
                out.print(
                        "<a href=\"" + word + ".html" + "\">" + word + "</a>");
            } else {
                out.print(word);
            }
        }
        out.println();
        out.println("</p>");
        /*
         * prints out return to index with link at the bottom of the page
         */
        out.println("<p>");
        out.println("Return to <a href=\"index.html\">index</a>");
        out.println("</p>");
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * Generates the set of terms in the given {@code Queue} into the given
     * {@code Set}.
     *
     * @param terms
     *            the given {@code Queue}
```

```java
     * @param termSet
     *            the {@code Set} to be replaced
     * @replaces termSet
     * @ensures termSet = entries(queue)
     */
    public static void queueToSet(Queue<String> terms, Set<String> termSet) {
        /*
         * for loop adds terms from the queue to the set for the length of the
         * queue
         */
        int length = terms.length();
        String term = "";
        for (int i = 0; i < length; i++) {
            term = terms.front();
            if (!termSet.contains(term)) {
                termSet.add(term);
            }
            terms.rotate(1);
        }
    }

    /**
     * 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) {
        /*
         * for loop adds characters to the set if they are not repeats through
         * the length of the string
         */
        int length = str.length();
        for (int i = 0; i < length; i++) {
            if (!charSet.contains(str.charAt(i))) {
                charSet.add(str.charAt(i));
            }
        }
    }

    /**
     * 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)
     * </pre>
     */
    public static String nextWordOrSeparator(String text, int position,
            Set<Character> separators) {
        String word = "";
        int length = text.length();
        /*
         * if the char at position is in the set it returns the string of chars
         * of consecutive separators else it does the opposite for chars not in
         * the set by adding each char to a string
         */
        if (separators.contains(text.charAt(position))) {
            while (position < length
                    && separators.contains(text.charAt(position))) {
                word = word + text.charAt(position);
                position++;
            }
        } else {
            while (position < length
                    && !separators.contains(text.charAt(position))) {
                word = word + text.charAt(position);
                position++;
            }
        }
        /*
         * returns the word created of separators or non separators
         */
        return word;
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();
        /*
         * asks user for file name and folder to store in
         */
        out.println("Enter the name of the file for the terms: ");
        String filename = in.nextLine();
        out.println("Enter the name of the folder for the term html files: ");
        String folderName = in.nextLine();
        /*
         * creates a reader for the text file and a writer for the index
         */
        SimpleReader inFile = new SimpleReader1L(filename);
        SimpleWriter outIndex = new SimpleWriter1L(folderName + "/index.html");
        /*
         * creates the set of separators
         */
        String separators = " ,.";
        Set<Character> separatorSet = new Set1L<Character>();
        generateElements(separators, separatorSet);
        /*
         * creates the Map, Queue, and Set to store the terms and/or definitions
         */
        Map<String, String> termsAndDefs = new Map1L<>();
        Queue<String> terms = collectTerms(inFile, termsAndDefs);
        Set<String> termSet = new Set1L<String>();
        queueToSet(terms, termSet);
        /*
         * creates comparator to sort the queue of terms alphabetically
         */
        Comparator<String> cs = new OrderAlpha();
        terms.sort(cs);
        /*
         * prints to index.html
         */
        outputIndex(outIndex, terms);
        /*
         * for loop runs for each term of the glossary printing out their
         * respective pages
         */
        int termNum = terms.length();
        String term = "";
        for (int i = 0; i < termNum; i++) {
            term = terms.front();
            SimpleWriter outTerm = new SimpleWriter1L(
                    folderName + "/" + term + ".html");
            terms.rotate(1);
            outputTermHTML(outTerm, termsAndDefs, term, separatorSet, termSet);
            outTerm.close();
        }
        /*
         * close input and output streams
         */
        in.close();
        out.close();
        outIndex.close();
        inFile.close();
    }

}
```