```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;

/**
 * Program that reads text input from files and turns the terms and definitions
 * in the file into a glossary of a series of HTML webpages.
 *
 * @author Bashir Ali
 *
 */
public final class Glossary {

    /**
     * . Comparator class used to sort strings alphabetically.
     *
     * @param o1
     *            The first string to be compared
     * @param o2
     *            the second string to be compared
     * @return A negative number if o1 is 'less than' o2, positive if o1 is
     *         'greater than' o2, and zero if o1 and o2 are 'equal'
     *
     */
    private static class Sort implements Comparator<String> {
        @Override
        public int compare(String o1, String o2) {
            return o1.compareTo(o2);
        }
    }

    /**
     * Returns the next word or separator from the text starting from the
     * specified position.
     *
     * @param text
     *            The text to get the next word or separator from
     * @param position
     *            The starting position to search for the next word or separator
     * @param separators
     *            A set of symbols considered as separators
     * @return The next word or separator found in the text starting from the
     *         given position.
     */
    private static String nextWordOrSeparator(String text, int position,
            Set<Character> separators) {
        //avoids checkstyle error with incrementing position
        int num = position;
        String result = "";
        //boolean to track if separator is found
        boolean hasSeparators = false;
        while (num < text.length() && !hasSeparators) {
            //checks character at position index for separator
            char ch = text.charAt(num);
            hasSeparators = separators.contains(ch);
            if (hasSeparators) {
                //add separator to string if found
                //did not know how to complete with 1 return
                return result;
            }
            result += ch;
            num++;
        }
        return result;
    }

    /**
     * Reads terms and definitions and puts them into map and sorts the keys of
     * map into a queue.
     *
     * @param input
     *            The SimpleReader to read input from.
     * @param map
     *            The Map to populate with key-value pairs.
     * @requires format Term and previous definition to be separated by a single
     *           line.
     * @return A sorted Queue containing the keys from the Map.
     */
    public static Queue<String> inputTermsAndDefinitions(SimpleReader input,
            Map<String, String> map) {
        //empty queue for keys to be stored in
        Queue<String> queue = new Queue1L<>();
        Comparator<String> comparator = new Sort();

        /*
         * while not at the end, store the first line as a key following line as
         * a value if the next line is not empty, keep adding to the value
         */
        while (!input.atEOS()) {
            String key = input.nextLine();
            String value = input.nextLine();
            String emptyLine = input.nextLine();

            while (!emptyLine.equals("")) {
                value += emptyLine;
                emptyLine = input.nextLine();
            }

            //add key and value to map, and sort the keys in the queue
            map.add(key, value);
            queue.enqueue(key);
            queue.sort(comparator);
        }
        return queue;
    }

    /**
     * generates an index page in HTML. The index page includes a list of links
     * to individual pages, with each link being connected to a key in the
     * queue.
     *
     * @param fileIn
     *            SimpleReader to read input from.
     * @param fileOut
     *            SimpleWriter to write the HTML index page.
     * @param map
     *            Map containing the terms as keys and definitions as values.
     * @param queue
     *            Queue contains key from map in alphabetical order.
     */
    public static void generateIndexPage(SimpleReader fileIn,
            SimpleWriter fileOut, Map<String, String> map,
            Queue<String> queue) {

        //printing for html page
        fileOut.println("<html>");
        fileOut.println("<title>Glossary</title>");
        fileOut.println("</head>");
        fileOut.println("<body>");
        fileOut.println("<h2>Glossary</h2>");
        fileOut.println("<hr/>");
        fileOut.println("<h3>Index</h3>");

        //creates term page for each item in the queue
        fileOut.println("<ul>");
        for (String str : queue) {
            fileOut.print("<li>");
            fileOut.println(
                    "<a href=\"" + str + ".html\">" + str + "</a></li>");
        }
        fileOut.println("</ul>");

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

    /**
     * Outputs the definition (values in the map) to a HTML file which is
     * hyperlinked on the index page.
     *
     * @param fileOut
     *            SimpleWriter to write the HTML output.
     * @param definition
     *            The definition of the term to be outputted
     * @param queue
     *            Queue contains key from map in alphabetical order.
     */
    public static void outputDefinitions(SimpleWriter fileOut,
            String definition, Queue<String> queue) {
        Set<Character> separators = new Set1L<>();
        separators.add(':');
        separators.add(';');
        separators.add('|');
        separators.add('~');
        separators.add('!');
        separators.add(' ');

        fileOut.print("<p>");
        int i = 0;
        while (i < definition.length()) {
            String str = nextWordOrSeparator(definition, i, separators);
            boolean contains = false;
            for (String s : queue) {
                //if the term is contained in the queue, link to that other term page
                if (s.equals(str)) {
                    contains = true;
                    fileOut.print("<a href= " + '"' + str + ".html" + '"' + ">"
                            + str + " " + "</a>");
                }
            }
            //if its not contained, print as plain text
            if (!contains) {
                fileOut.print(str + " ");
            }
            //adds 1 for the space between words and the word length to go to next word
            i = i + 1 + str.length();
        }
        fileOut.print("</p>");
    }

    /**
     * Generates individual HTML pages for each term in the glossary and stores
     * them in folder.
     *
     * @param folder
     *            The folder where the HTML pages will be stored.
     * @param map
     *            Map containing the terms as keys and definitions as values.
     * @param queue
     *            Queue contains key from map in alphabetical order.
     */
    public static void generateTermPages(String folder, Map<String, String> map,
            Queue<String> queue) {
        // process terms in reverse order from the queue and remove from map
        for (int i = queue.length(); i != 0; i--) {
            Map.Pair<String, String> current = map.remove(queue.front());
            queue.rotate(-1);

            SimpleWriter fileOut = new SimpleWriter1L(
                    folder + "/" + current.key() + ".html");
            String term = current.key();
            String definition = current.value();

            //printing for html page
            fileOut.println("<html>");
            fileOut.println("<head>");
            fileOut.println("<title>" + term + "</title>");
            fileOut.println("</head>");
            fileOut.println("<body>");
            fileOut.print("<h2>");
            fileOut.print("<b>");
            fileOut.print("<i>");
            fileOut.println("<font color=" + "\"red\">" + term
                    + "</font></i></b></h2>");

            fileOut.println("<p>");

            //output definitions of words on the term page
            outputDefinitions(fileOut, definition, queue);
            fileOut.println("<hr />");
            fileOut.println("<p>");
            fileOut.println("return to <a href=\"index.html\">index</a>");
            fileOut.println("</p>");
            fileOut.println("</body>");
            fileOut.println("</html>");
            fileOut.close();
        }
    }

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

        out.print("Enter a file: ");
        String userFile = in.nextLine();
        out.print("Enter the name of a folder: ");
        String userFolder = in.nextLine();

        SimpleReader fileIn = new SimpleReader1L(userFile);
        SimpleWriter fileOut = new SimpleWriter1L(userFolder + "/index.html");

        //creates queue of keys and updates empty map to fill with keys/values
        Map<String, String> map = new Map1L<>();
        Queue<String> queue = inputTermsAndDefinitions(fileIn, map);

        generateIndexPage(fileIn, fileOut, map, queue);
        generateTermPages(userFolder, map, queue);

        in.close();
        out.close();
        fileIn.close();
        fileOut.close();
    }

}

```