```java
import components.queue.Queue;
import components.queue.Queue2;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Reads an input file and outputs an html file glossary with specifications
 * provided by customer.
 *
 * @author Isaac Frank
 *
 * @customer Cy Burnett
 */
public final class Glossary {

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

    /**
     * Reads a String and creates an html file for one term and its definition.
     *
     * @param input
     *            the name of the file with the term and definition to read
     *
     * @param outFolder
     *            the folder to output files to
     *
     * @requires input is formatted with a one word term, followed by a space
     *           and the definition of the term
     *
     * @ensures the creation of an html file titled with the name of the term,
     *          the html file's contents are the term in red boldfaced italics
     *          followed by its definition AND generateTerm = term
     *
     * @return the name of the term
     */
    public static String generateTerm(String input, String outFolder) {
        int firstSpace = input.indexOf(' ');
        String term = input.substring(0, firstSpace);
        String definition = input.substring(firstSpace + 1);

        SimpleWriter outFile = new SimpleWriter1L(
                outFolder + '/' + term + ".html");
        outFile.println("<html> <body> <p style=\"color:#FF0000\">");
        outFile.println("<strong> <em>" + term + "</em> </strong> </p>");
        outFile.println("<p>" + definition);
        outFile.println("</p> </body> </html>");

        outFile.close();

        return term;
    }


    /**
     * Takes an input file and sorts the terms A to Z, printing the sorted list
     * in a new text file (uses insertion sort technique).
     *
     * @param sort
     *            all terms and definitions (term + definition is one element)
     *
     * @requires sort is formatted such that the term is the first word of each
     *           element
     *
     * @ensures sort is sorted alphabetically by terms
     */
    public static void sortAToZ(String[] sort) {
        int len = sort.length;
        for (int i = 0; i < len - 1; i++) {
            String temp = sort[i + 1];
            int j = i;
            for (; j >= 0 && sort[j].compareTo(temp) > 0; j--) {
                sort[j + 1] = sort[j];
            }
            sort[j + 1] = temp;
        }
    }

    /**
     * Generates the glossary top level index with terms in alphabetical order.
     *
     * @param inFileName
     *            the input file name with terms and definitions
     *
     * @param outFolder
     *            the name of the folder to output files to
     *
     * @requires the file that inFileName leads to is formatted such that a
     *           single term on the first line is followed on the next line by
     *           its definition on one or more lines, and after each definition
     *           is a new, empty line AND outFolder must exist AND the input
     *           file is not empty
     *
     *
     * @ensures an html file is generated that fits all of Cy's glossary
     *          requirements
     */
    public static void generateList(String inFileName, String outFolder) {
        SimpleWriter outFile = new SimpleWriter1L(
                outFolder + '/' + "index.html");
        SimpleReader inFile = new SimpleReader1L(inFileName);

        // Storing each term and def into an unsorted queue (can't do an array
        // because there is an unknown number of terms)
        Queue<String> unsortedQ = new Queue2<>();
        while (!inFile.atEOS()) {

            // Takes out the term and copies to termAndDef and then adds a space
            String termAndDef = inFile.nextLine();
            if (termAndDef.length() == 0) {
                termAndDef = inFile.nextLine();
            }
            termAndDef += ' ';

            // Add definition to termAndDef here until empty line
            while (!inFile.atEOS() && inFile.peek() != '\n') {
                termAndDef += inFile.nextLine();
            }
            // Adds each term and definition onto unsorted
            unsortedQ.enqueue(termAndDef);
        }

        // Moving all elements from unsortedQ to an unsorted String array
        String[] sorted = new String[unsortedQ.length()];
        while (unsortedQ.length() > 0) {
            sorted[unsortedQ.length() - 1] = unsortedQ.dequeue();
        }

        // Stores the sorted array of terms and definitions
        sortAToZ(sorted);

        // Opening tags and heading
        outFile.println("<html> <body> <h1>" + "Glossary" + "</h1>");
        outFile.println("<ul>");

        for (int i = 0; i < sorted.length; i++) {
            String term = generateTerm(sorted[i], outFolder);
            // Adding term to a list and making it a link
            outFile.print("<li>");
            outFile.print("<a href = \"" + term + ".html" + "\">");
            outFile.print(term + "</a>");
            outFile.println("</li>");
        }
        // Closing tags
        outFile.println("</ul>");
        outFile.print("</body> </html>");

        // Closing streams
        outFile.close();
        inFile.close();
    }

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

        // Get user input for input file
        out.print("Enter the input file name: ");
        String inFile = in.nextLine();

        // Get user input for output folder name (must already exist)
        out.print("Enter an existing folder's name to output glossary to: ");
        String outFolder = in.nextLine();

        // Output index.html with a list of items
        generateList(inFile, outFolder);

        // Closing streams
        in.close();
        out.close();
    }

}

```