```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;

/**
 * A componenent based glossary application that outputs a group of HTML files.
 * each term in this glossary consists of a single word.
 *
 * @author VishalKumar
 *
 */

public final class Glossary {

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

    /**
     * Reads the input file and grabs the terms and their respective definitions
     * and loads them into a map. This method also puts all of the terms
     * (without their definitions into a sequence;
     *
     * @param inFile:
     *            the given file
     * @param map:
     *            the map to be populated
     *
     * @replaces map
     * @replaces terms
     * @requires inFile is reading a properly formatted text file
     */
    private static void getTerms(SimpleReader inFile, Map<String, String> map) {

        String term = "term";
        String definition = "";

        while (!(inFile.atEOS()) && !term.isEmpty()) {
            term = inFile.nextLine();

            definition = inFile.nextLine();
            if (!(inFile.atEOS())) {
                String temp = inFile.nextLine();

                while (!temp.isEmpty()) {
                    definition += temp;
                    temp = inFile.nextLine();
                }
            }
            if (!map.hasKey(term)) {
                map.add(term, definition);
            }
        }
    }

    /**
     * A class that defines a compare method for alphabetically sorting strings
     *
     * @author VishalKumar
     *
     */
    private static class AlphabeticalSort implements Comparator<String> {
        /**
         * compares two strings and determines which one comes first
         * alphabetically
         *
         * @param s1
         *            the first String
         * @param s2
         *            the second String
         */
        @Override
        public int compare(String s1, String s2) {
            return s1.compareTo(s2);

        }
    }

    /**
     * grabs terms from a Map and puts them into a sorted Queue.
     *
     * @param map:
     *            the given Map that stores all terms with definition
     * @return A sorted queue that stores all terms
     */
    public static Queue<String> createSortedQueue(Map<String, String> map) {
        Queue<String> terms = new Queue1L<String>();
        // loop through map and add keys to a queue
        for (Map.Pair<String, String> pair : map) {
            terms.enqueue(pair.key());
        }
        // sort the queue
        Comparator<String> strCompare = new AlphabeticalSort();
        terms.sort(strCompare);
        return terms;
    }

    /**
     * outputs the index.html page
     *
     * @param terms
     *            all glossary terms that need to be printed
     * @param folderName
```

```java
     *            where to output the file
     */
    public static void printIndexPage(Queue<String> terms, String folderName) {
        SimpleWriter termWriter = new SimpleWriter1L(
                folderName + "/" + "index.html");
        termWriter.print(
                "<html>\n<head>\n<title>Glossary Index</title>\n</head>\n");
        termWriter.print(
                "<body>\n<h2>Glossary Index</h2>\n<hr />\n<h3>Index</h3>\n<ul>\n");
        for (String term : terms) {
            termWriter.println(
                    "<li><a href=\"" + term + ".html\">" + term + "</a></li>");
        }
        termWriter.print("</ul>\n</body>\n</html>\n");
        termWriter.close();
    }

    /**
     * outputs all terms, each with its respective definitions page
     *
     * @param map
     *            a Map<String, String> that stores all terms with definitions
     * @param folderName
     *            where to output the files
     */
    public static void printTermPages(Map<String, String> map,
            String folderName) {
        // a set of terms
        Set<String> terms = new Set1L<String>();
        // loop through map and create term pages
        for (Map.Pair<String, String> term : map) {
            SimpleWriter out = new SimpleWriter1L(
                    folderName + "/" + term.key() + ".html");
            out.print("<html>\n<head>\n<title>" + term.key()
                    + "</title>\n</head>\n");
            out.print("<body>\n<h2><b><i><font color=\"red\">" + term.key()
                    + "</font></i></b></h2>\n");
            String definition = term.value();
            // some definitions may have other terms in the glossary nested in them
            terms = GetAllTermsInSentence(term.value(), map);
            for (String words : terms) {
                definition = definition.substring(0, definition.indexOf(words))
                        + "<a href=\"" + words + ".html\">" + words + "</a>"
                        + definition.substring(
                                definition.indexOf(words) + words.length());
            }
            out.print("<blockquote>" + definition + "</blockquote>");
            out.println("<hr />");
            out.println("<p>Return to <a href=\"index.html\">index</a>.</p>");
            out.print("</body>\n</html>");
        }
    }

    /**
     * check to see if there are any additional terms in a definition sentence
     *
     * @param str
     *            the given definition
     * @param map
     *            a Map<String, String> that stores all terms
     * @return a Set<String> that stores all terms exist in the given definition
     */
    private static Set<String> GetAllTermsInSentence(String str,
            Map<String, String> map) {
        // loop through map and grab all terms that appear in str
        Set<String> terms = new Set1L<String>();
        for (Map.Pair<String, String> term : map) {
            if (str.contains(term.key()) && !terms.contains(term.key())) {
                terms.add(term.key());
            }
        }
        return terms;
    }

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

        // get input file name from user and output folder name
        out.println("Please enter an input file name: ");
        String fname = in.nextLine();
        out.println("Awesome, please also enter an output folder name: ");
        String folderName = in.nextLine();

        // construct inReader and folder output objects
        SimpleReader inFile = new SimpleReader1L(fname);

        // make a map to hold glossary terms and their definitions
        Map<String, String> map = new Map1L<String, String>();

        // populate the map
        getTerms(inFile, map);

        // create a queue containing all the glossary terms (sorted)
        // this will be important for html output
        Queue<String> terms = new Queue1L<String>();
        terms = createSortedQueue(map);

        // create the glossary using the map and queue and output it to
        // a user-provided folder
        printIndexPage(terms, folderName);
        printTermPages(map, folderName);

        /*
         * Close input and output streams
         */
        in.close();
        out.close();
    }

```

```
}

```