```
import java.io.PrintStream;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;


/**
 *
 * Console Output is a Output word interface that outputs the words to the conso
le and determines how
 * the words are to be outputed.
 * @author Tomas Barreto
 *
 */
public class ConsoleOutput implements IOutputWords {

        public void PrintWords(PrintStream out, Map words) {
                Set keys = words.keySet();
                Iterator it = keys.iterator();
                while (it.hasNext()){
                        String word = (String) it.next();
                        WordLocation wl = (WordLocation) words.get(word);

                        out.print(word+":"+wl.toString());
                        out.println();
                }
        }

}
```



```
/**
 * @author Tomas Barreto
 * IMakeWord is the word format interface
 */
public interface IMakeWord {
        public String makeWord(String word);


}
```

Sep 06, 05 11:13 **IOutputWords.java** Page 1/1

```
import java.io.PrintStream;
import java.util.Map;

/**
 * @author Tomas Barreto
 * How to output the words interface
 */
public interface IOutputWords {

        public void PrintWords(PrintStream out, Map words);

}
```

Sep 06, 05 11:13 **IWordFilter.java** Page 1/1

```
/**
 * @author Tomas Barreto
 * IWordFilter is the word filter interface
 */
public interface IWordFilter {

        public boolean filter (String word);

}
```

```
/**
 * @author Tomas Barreto
 * simplefilter is a IWordFilter that filters strings that are null
 */
public class SimpleFilter implements IWordFilter{

        public static void main(String[] args) {
        }

        /* (non-Javadoc)
         * @see IWordFilter#filter(java.lang.String)
         */
        public boolean filter(String word) {
                // TODO Auto-generated method stub
                if(word!=null)
                return true;
                return false;
        }
}
```



```
/**
 * @author Tomas Barreto
 * SimpleWorder is a IMakeWord class that formats Strings to lowercase and remov
es unnecessary characters
 * from the beginning and end
 */
public class SimpleWorder implements IMakeWord {

        /* (non-Javadoc)
         * @see IMakeWord#makeWord(java.lang.String)
         */
        public String makeWord(String word) {
                // TODO Auto-generated method stub

                word = word.toLowerCase();

                try{
                        while(!Character.isLetter(word.charAt(0))){
                                word = word.substring(1);
                        }
                        while(!Character.isLetter(word.charAt(word.length()-1)))
{
                                word = word.substring(0,word.length()-1);
                        }
                }
                catch(Exception e){
                        return null;
                }

                return word;
        }

        public static void main(String[] args) {
        }
}
```

```
/**
 * Class that contains the main method to launch the code
 * @author Tomas Barreto
 *
 */
public class Starter {
        public static void main(String[] args){
                WordProcessor wt = new WordProcessor();
                wt.readAll();

                IOutputWords console = new ConsoleOutput();
                console.PrintWords(System.out,wt.getMap());

                System.exit(0);
        }
}
```



```
import java.util.ArrayList;



/**
 *
 * WordLocation is a class that contains the fileName, title and line numbers fo
r any given word
 *
 * @author Tomas Barreto
 *
 */
public class WordLocation {
        private String fileName;
        private String title;
        private ArrayList lineNum = new ArrayList();

        public String getFileName(){
                return fileName;
        }
        public String getTitle(){
                return title;
        }
        public ArrayList getLineNum(){
                return lineNum;
        }
        public void addLineNum(Integer line){
                lineNum.add(line);
        }
        public void setFileName(String fName){
                fileName = fName;
        }
        public void setTitle(String t){
                title = t;
        }

        //printlines outputs the arraylist of line numbers with commas after eac
h one
        private String printLines(ArrayList l){
                String lines = "";
                for(int x = 0; x<l.size(); x++){
                        lines+=l.get(x)+",";
                }
                lines = lines.substring(0,lines.length()-2);
                return lines;
        }

        //the toString method determines how the class contents will be displaye
d when printed
        public String toString(){
                return fileName+":"+title+":"+printLines(lineNum);
        }

        public static void main(String[] args) {
        }
}
```

```
/**
 *
 * WordProcessor allows the user to select a directory.  Once the directory is chosen all the
 * files within the directory are read and processed into a TreeMap.
 *
 *
 * @author Tomas Barreto
 * @version 0.8
 */

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;

import javax.swing.JFileChooser;

public class WordProcessor {

        /**
         *  wordMap is a TreeMap that maps a word to a WordLocation object
         *  curTitle keeps stored the Title of the file
         *  curFile keeps stored the current file which is being processed
         *      worder is a word formatter interface that formats the word according to the word specifications
         *  filter is a word filter that every word must pass before it is placed in the map
         *
         */

        Map wordMap = new TreeMap();
        String curTitle = "<no title>";
        String curFile ="";
        IMakeWord worder;
        IWordFilter filter;


        private static JFileChooser ourChooser = new JFileChooser(".");
        static {
                ourChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        }
        /**
         * Read all files in a director chosen by the user
         * and process the files one-at-a-time.
         * @see readOne
         */
        public void readAll(){

                int retval = ourChooser.showOpenDialog(null);
                if (retval != JFileChooser.APPROVE_OPTION){
                        return;
                }
                File dir = ourChooser.getSelectedFile();
                File[] allFiles = dir.listFiles();
                for(int k=0; k < allFiles.length; k++){
                        readOne(allFiles[k]);
                }
        }

        /**
         * Process one file by processing all words in
         * the file.
         * @param f is file processed
         * @see processOneWord
         */
        protected void readOne(File f){
                BufferedReader reader = null;
                try {
                        reader = new BufferedReader(new FileReader(f));
```



```
                } catch (FileNotFoundException e) {
                        showMessage("error opening "+f.getName());
                        e.printStackTrace();
                }

                /*
                 * set the name of the current File
                 * and of the current title
                 */
                curFile = f.getName();
                curTitle = getCurTitle(reader);

                int lineCount = 0;
                String line;
                try {
                        while ((line = reader.readLine()) != null){
                                lineCount++;
                                String[] allWords = line.split("\\s+");
                                for(int k=0; k < allWords.length; k++){
                                        processOneWord(allWords[k],lineCount,f);
                                }
                        }
                } catch (IOException e) {
                        showMessage("error reading line "+lineCount+" of "+f.getName());
                        e.printStackTrace();
                }
        }

        /**
         * This method extracts the title from the file
         *
         * @param reader is the buffered reader that was initialized before this method is called
         * @return
         */
        private String getCurTitle(BufferedReader reader) {
                /*
                 * x keeps track of how many lines are read before *** START is reached.  If it is not found after
                 * 40 lines it is assumed that the file does not contain it and is returned to the readOne method.
                 *
                 * title is the string returned for the current title
                 *
                 */
                int x = 0;
                String title = "<no title>";
                String line;
                try {
                        while ((line = reader.readLine()) != null && x < 40){
                                if(line.indexOf("*** START")>=0)
                                        break;
                                int start;
                                if((start = line.indexOf("Title: ")) >= 0){
                                        title = line.substring(start+7);
                                }

                                x++;
                        }
                } catch (IOException e) {
                        System.out.println("problem getting title");
                        e.printStackTrace();
                }
                return title;
        }

        /**
         * Process one word, occuring on specified line, in specified file.
         * @param word is word to process
         * @param lineCount is line on which word occurs
         * @param f is the File from which word was read
         */
        protected void processOneWord(String word, int lineCount, File f) {
```

```
                    /*
                     * WordLocation object is the value that every word is mapped to
    .  It has the title, filename
                     * and linenumbers on which a word appears
                     */
                    WordLocation wl = new WordLocation();

                    /*
                     * worder is a SimpleWorder that formats a string into a real wo
    rd.  It does this by removing
                     * the punctuation from the beginning and end of a string and al
    so makes the word lowercase.
                     */
                    worder = new SimpleWorder();
                    word = worder.makeWord(word);

                    /*
                     * Simple filter only filters out words that are null at this po
    int.
                     */
                    filter = new SimpleFilter();

                    if(filter.filter(word)){
                            if(wordMap.containsKey(word)){
                                    wl = (WordLocation) wordMap.get(word);
                                    wl.addLineNum(new Integer(lineCount));
                                    wordMap.remove(word);
                                    wordMap.put(word, wl);
                            }
                            else {
                                    wl.setFileName(curFile);
                                    wl.setTitle(curTitle);
                                    wl.addLineNum(new Integer(lineCount));
                                    wordMap.put(word, wl);
                            }
                    }
            }

            /**
             * Show user a message.
             * @param s is message shown
             */
            protected void showMessage(String s){
                    System.out.println(s);
            }

            /**
             * Returns the map stored in WordProcessor
             *
             * @return wordMap is the word map that is contained in word processor
             */
            public Map getMap(){
                    return wordMap;
            }

    }
```