```java
import java.io.BufferedReader;

/**
 * Creates a tag cloud from a given file input text.
 *
 * @author Chloe Feller
 * @author Krish Patel
 *
 */
public final class TagCloudGenerator {

    /**
     * No argument constructor--private to prevent instantiation.
     */
    private TagCloudGenerator() {
    }

    /**
     * This is a numerical ordering system which orders the largest numbers over
     * the smaller numbers.
     */
    @SuppressWarnings("serial")
    private static class Sort
            implements Serializable, Comparator<Map.Entry<String, Integer>> {
        @Override
        public int compare(Map.Entry<String, Integer> one,
                Map.Entry<String, Integer> two) {
            int compared = 0;
            if (one.getValue().equals(two.getValue())) {
                compared = one.getKey().compareToIgnoreCase(two.getKey());
            } else {
                compared = two.getValue().compareTo(one.getValue());
            }
            return compared;
        }

    }

    /**
     * This is an alphabetical ordering system which orders words starting from
     * a all the way to z.
     */
    @SuppressWarnings("serial")
    private static class SortTwo
            implements Serializable, Comparator<Map.Entry<String, Integer>> {
        @Override
        public int compare(Map.Entry<String, Integer> one,
                Map.Entry<String, Integer> two) {
            return one.getKey().compareToIgnoreCase(two.getKey());
        }
    }

    /**
     * Reads words from the input file and adds them to a {@code Map}. Words are
     * not alphabetized yet.
     *
     * @param words
     *            the {@code Map} of words
     * @param file
     *            file input by user
     *
     * @requires file.isOpen
     * @requires words != null
     * @replaces words
     *
     */
    private static void readFile(Map<String, Integer> words,
            BufferedReader file) throws IOException {
        assert words != null : "Violation of : words is not null";
        assert file.ready() : "Violation of : file is open";

        String separator = " \t,.-;'/\"@#$%&()*`";
        Set<Character> charSet = new HashSet<>();

        generateElements(separator, charSet);

        /*
         * Read through the file until all lines are read, while adding words to
         * the Map
         */
        String line = file.readLine();

        while (file.ready()) {
            int i = 0;

            while (i < line.length()) {
                String text = nextWordOrSeparator(line, i, charSet);
                if (!charSet.contains(text.charAt(0))) {
                    /*
                     * Sees if words contains the word. If it does not, the word
                     * is added. If it does, the number of times it has appeared
                     * is increased.
                     */
                    if (words.containsKey(text)) {
                        int numberAppear = words.get(text);
                        numberAppear++;
                        words.replace(text, numberAppear);
                    } else {
                        words.put(text, 1);
                    }
                }
                // Skip to the next word/separator
                i += text.length();
            }

            line = file.readLine();
        }

    }

    /**
     * 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)
     */
    private static void generateElements(String str, Set<Character> charSet) {
        for (int i = 0; i < str.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>
     */
    private static String nextWordOrSeparator(String text, int position,
            Set<Character> separators) {
        assert text != null : "Violation of: text is not null";
        assert position >= 0 : "Violation of: position is not >= 0";
        assert position < text
                .length() : "Violation of: position is not < |text|";
        assert separators != null : "Violation of: separators is not null";

        String str = "";
        char returnedChar = 'a';

        if (separators.contains(text.charAt(position))) {
            for (int i = 0; i < text.substring(position, text.length())
                    .length(); i++) {
                returnedChar = text.charAt(position + i);
                if (separators.contains(returnedChar)) {
                    str = str + returnedChar;
                } else {
                    i = text.substring(position, text.length()).length();
                }
            }
        } else {
            for (int i = 0; i < text.substring(position, text.length())
                    .length(); i++) {
                returnedChar = text.charAt(position + i);
                if (!separators.contains(returnedChar)) {
                    str = str + returnedChar;
                } else {
                    i = text.substring(position, text.length()).length();
                }
            }
        }

        return str;
    }

    /**
     * Outputs the opening tags for the output HTML file.
     *
     * @param out
     *            output stream
     * @param file
     *            input file given by user
     * @param x
     *            number of words given by user
     * @updates {@code out}
     * @requires <pre>
     * {@code file} is open and not null and {@code out} is open
     * </pre>
     * @ensures <pre>
     * {@code out = #out * tags}
     * </pre>
     */
    private static void outputHeader(PrintWriter out, String file, int x) {
        assert out != null : "Violation of : out is not null";
        assert file != null : "Violation of : file is not null";

        /*
         * Print out beginning of HTML file
         */
        out.println("<html>");
        out.println("<head>");

        /*
         * Print out title
         */
        out.println("<title>Top " + x + " words in " + file + "</title>");
        out.println("<link href=\"http://web.cse.ohio-state.edu/software/2231/"
                + "web-sw2/assignments/projects/tag-cloud-generator/data/"
                + "tagcloud.css\" rel=\"stylesheet\" type=\"text/css\">");
        out.println("<link href=\"doc/tagcloud.css\" "
                + "rel=\"stylesheet\" type=\"text/css\">");
        out.println("</head>");

        /*
```

```
         * Print out body
         */
        out.println("<body>");
        out.println("<h2>Top " + x + " Words Counted in " + file + "</h2>");
        out.println("<hr>");
        out.println("<div class=\"cdiv\">");
        out.println("<p class=\"cbox\">");

    }

    /**
     * Prints footer for the output HTML file.
     *
     * @param out
     *            output stream
     * @updates {@code out}
     * @requires <pre>
     * {@code out} is open
     * </pre>
     * @ensures <pre>
     * {@code out = #out * tags}
     * </pre>
     */
    private static void outputFooter(PrintWriter out) {
        out.println("</p>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * This sorts the words into two different groups. It starts sorting through
     * a comparator in numerical order and then a second ordering method being
     * alphabetically. It would then print to the output file in HTML format
     * with the appropriate fonts.
     *
     * @param mapCount
     *            This is the map of words and numbers that show up
     * @param out
     *            output file stream
     * @param words
     *            number of words given by user
     */
    private static void sortingAndFonts(Map<String, Integer> mapCount,
            PrintWriter out, int words) {

        Iterator<Map.Entry<String, Integer>> sorting = mapCount.entrySet()
                .iterator();
        Map.Entry<String, Integer> pair;

        //numerical order
        Comparator<Map.Entry<String, Integer>> nums = new Sort();
        List<Map.Entry<String, Integer>> numberOrder;
        numberOrder = new LinkedList<Map.Entry<String, Integer>>();

        while (mapCount.size() > 0) {
            pair = sorting.next();
            sorting.remove();

```

```java
            numberOrder.add(pair);
        }

        numberOrder.sort(nums);

        Iterator<Map.Entry<String, Integer>> sort2 = numberOrder.iterator();

        //alphabetical ordering
        Comparator<Map.Entry<String, Integer>> numsTwo = new SortTwo();
        List<Map.Entry<String, Integer>> sortTwo;
        sortTwo = new LinkedList<Map.Entry<String, Integer>>();

        int min = 0;
        int max = 0;

        //loop ordering
        for (int i = 0; (i < words) && (1 < numberOrder.size()); i++) {
            Map.Entry<String, Integer> wording = sort2.next();
            sort2.remove();

            int neg = words - 1;

            if (i == 0) {
                max = wording.getValue();
            } else if (i == neg) {
                min = wording.getValue();
            }
            sortTwo.add(wording);
        }

        sortTwo.sort(numsTwo);

        //alphabetical + printing to the output stream
        while (sortTwo.size() > 0) {
            Map.Entry<String, Integer> removed = sortTwo.remove(0);

            final int eleven = 11;
            final int fortyeight = 48;
            int sizeFont = 0;
            if (removed.getValue() == min) {
                sizeFont = eleven;
            } else if (removed.getValue() == max) {
                sizeFont = fortyeight;
            } else {
                sizeFont = eleven
                        + ((removed.getValue() * (fortyeight - eleven))
                                / (max));
            }

            String f = "f" + sizeFont;

            out.println("<span style=\"cursor:default\" class=\"" + f
                    + "\" title=\"count: " + removed.getValue() + "\">"
                    + removed.getKey().toLowerCase() + "</span>");
        }

    }

    /**
```

```
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        /*
         * Open input file.
         */
        BufferedReader in = new BufferedReader(
                new InputStreamReader(System.in));
        BufferedReader input = null;
        String inRead = "";

        try {
            System.out.print("Enter an input file: ");
            inRead = in.readLine();
            input = new BufferedReader(new FileReader(inRead));
        } catch (IOException e) {
            System.err.println("Unable to open input file");
        }

        /*
         * Open output file.
         */
        PrintWriter output = null;
        String outRead;

        try {
            System.out.print("Enter an output file: ");
            outRead = in.readLine();
            output = new PrintWriter(
                    new BufferedWriter(new FileWriter(outRead)));
        } catch (IOException e) {
            System.err.println("Unable to open output file");
        }

        /*
         * Gather the number of words in the tag cloud.
         */
        System.out.print("Enter number of words: ");

        int words = -1;

        try {
            while (words < 0) {
                words = Integer.parseInt(in.readLine());
                if (words < 0) {
                    System.out.println("Must enter a positive number");
                }
            }
        } catch (IOException e) {
            System.err.println("Error reading number");
        } catch (NumberFormatException e) {
            System.err.println("Number is not in the correct format");
        }

        /*
         * Output the header of the HTML file.
```

```java
         */
        Map<String, Integer> tmpMap = new HashMap<>();

        /*
         * Generate tags and sort them.
         */
        try {
            readFile(tmpMap, input);
        } catch (IOException e) {
            System.err.println("Error reading lines in the input file");
        }
        outputHeader(output, inRead, words);
        sortingAndFonts(tmpMap, output, words);

        /*
         * Output footer of HTML file and close output stream.
         */
        if (output != null) {
            outputFooter(output);
            output.close();
        }

        /*
         * Close input streams
         */
        try {
            in.close();
        } catch (IOException e) {
            System.err.println("Error closing files");
        }

        if (input != null) {
            try {
                input.close();
            } catch (IOException e) {
                System.err.println("Error closing files");
            }
        }
    }

}
```