```java
import java.io.BufferedReader;

/**
 * Read txt and find the amount words that appear most frequently. The amount is
 * an integer entered by the user.
 *
 * @author Qinuo Shi & Yiming Cheng
 */
public final class TagCloudStandard {

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

    /**
     * there separator are used in countingWords method.
     */
    public static final String Separator = "\\ \t\n\r,-.!?[]';:/()@&~`\"";

    /**
     * countingWords is for inserting the separated word and its number of uses
     * into map.
     *
     * @param contentIn
     *             BufferedReader
     * @param wordNum
     *             the map include all words and their occurrences
     * @requires BufferedReader should not be empty
     * @update wordwordNum
     * @ensures wordNum should not be empty
     *
     */
    public static void countingWords(BufferedReader contentIn,
            Map<String, Integer> wordNum) {
        assert contentIn != null : "Violation of: txt file is not null";

        /*
         * Store all possible non-alphabetic symbols in a set.
         */
        Set<Character> sepSet = new HashSet<Character>();
        for (int i = 0; i < Separator.length(); i++) {
            sepSet.add(Separator.charAt(i));
        }

        /*
         * Read content in the file and write them into map.
         */
        String contentLine;
        try {
            contentLine = contentIn.readLine();
            while (contentLine != null) {
                int pos = 0;
                while (pos < contentLine.length()) {
                    String word = nextWordOrSeparator(contentLine, pos, sepSet);
                    if (!sepSet.contains(word.charAt(0))) {
                        if (!(wordNum.containsKey(word))) {
                            wordNum.put(word, 1);
                        } else {
                            /*
                             * If a word is already recorded, change the map
                             * value, and leaving the set unchanged.
                             */
                            int num = wordNum.get(word) + 1;
                            wordNum.put(word, num);
                        }
                    }
                    pos += word.length();
                }
                contentLine = contentIn.readLine();
            }
        } catch (IOException e1) {
            e1.printStackTrace();
            System.out.print("There are some errors with recording content");
            return;
        }
    }

    /**
     * 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 str
     *            the {@code String} from which to get the word or separator
     *            string
     * @param pos
     *            the starting index
     * @param sepSet
     *            the {@code Set} of separator characters
     * @return the first word or separator string found in {@code str} starting
     *         at index {@code pos}
     * @requires 0 <= pos < |str|
     * @ensures <pre>
     * nextWordOrSeparator =
     *   str[pos, pos + |nextWordOrSeparator|)  and
     * if entries(str[pos, pos + 1)) intersection separators = {}
     * then
     *   entries(nextWordOrSeparator) intersection separators = {}  and
     *   (pos + |nextWordOrSeparator| = |str|  or
     *    entries(str[pos, pos + |nextWordOrSeparator| + 1))
     *      intersection separators /= {})
     * else
     *   entries(nextWordOrSeparator) is subset of separators  and
     *   (pos + |nextWordOrSeparator| = |str|  or
     *    entries(str[pos, pos + |nextWordOrSeparator| + 1))
     *      is not subset of separators)
     * </pre>
     */
    public static String nextWordOrSeparator(String str, int pos,
            Set<Character> sepSet) {
        assert str != null : "Violation of: str is not null";
        assert sepSet != null : "Violation of: separators is not null";
        assert 0 <= pos : "Violation of: 0 <= pos";
        assert pos < str.length() : "Violation of: pos < |str|";

        int endPos = -1;
        String word = "";
        /*
         * Use for loop to find the term's position.
         */
        for (int i = pos; i < str.length(); i++) {
            if (sepSet.contains(str.charAt(i)) && endPos == -1) {
                endPos = i;
            }
        }
        /*
         * Depending on the case, intercepts the corresponding substring.
         */
        if (endPos == pos) {
            word = str.substring(pos, endPos + 1);
        } else if (endPos == -1) {
            word = str.substring(pos);
        } else {
            word = str.substring(pos, endPos);
        }

        return word;
    }

    /**
     * compare two integers and return a value.
     */
    private static class CompareNum
            implements Comparator<Map.Entry<String, Integer>> {
        @Override
        public int compare(Entry<String, Integer> p1,
                Entry<String, Integer> p2) {
            int cmp = p2.getValue().compareTo(p1.getValue());
            if (cmp != 0) {
                return cmp;
            }
            return p2.getValue().compareTo(p1.getValue());
        }

    }

    /**
     * compare two strings and return a value.
     */
    private static class CompareString implements Comparator<String> {

        @Override
        public int compare(String p1, String p2) {
            int cmp = p1.toLowerCase().compareTo(p2.toLowerCase());
            if (cmp != 0) {
                return cmp;
            }

            return p2.compareTo(p1);
        }

    }

    /**
     * Sorting the {@code sortByStr} with top {@code n} counts in the
     * {@code sortByInt} and returns the String that contains the maximum and
     * mininum number of count in {@code sortByStr} separated by character ';'.
     *
     * @param wordNum
     *            the map include all words and their occurrences
     * @param orderedInt
     *            a PriorityQueue to sort all int value
     * @param orderedString
     *            a SortedMap for recording all ordered things for html output
     *            in other method
     * @param number
     *            the amount of the most frequent words
     * @return a String that record the maximum and the minimum occurrences
     * @requires the sortingMap and the number should not be empty
     * @ensures MaxMin need record "maximum value ; minimum value"
     */
    private static String sortingMap(Map<String, Integer> wordNum,
            PriorityQueue<Map.Entry<String, Integer>> orderedInt,
            SortedMap<String, Integer> orderedString, int number) {
        assert wordNum != null : "Violation of: the map is not null";
        assert number > 0 : "Violation of: Number must be positive";

        String maxAndMin = "";

        /*
         * Put each pair into a PriorityQueue, then the int value would be
         * ordered.
         */
        for (Map.Entry<String, Integer> pairToPQueue : wordNum.entrySet()) {
            orderedInt.add(pairToPQueue);
        }

        for (int i = 0; i < number; i++) {
            Map.Entry<String, Integer> pair = orderedInt.remove();
            int occurrence = pair.getValue();
            String word = pair.getKey();

            /*
             * After ordering, the first one is the most frequent one, and last
             * one is the least one.
             */
            if (i == 0) {
                maxAndMin = pair.getValue().toString() + ";";
            }
            if (i == number - 1) {
                maxAndMin += pair.getValue().toString();
            }

            /*
             * Put them together again for output html file.
             */
            orderedString.put(word, occurrence);
        }

        return maxAndMin;
    }

    /**
     * read the txt file and enter all the words into set and map.
     *
     * @param orderedString
     *            its contents are used to identify the most frequently used
     *            words, as well as the number of occurrences.
     * @param out
     *            the output
     * @param htmlName
     *            the name of output file
     * @param maxMin
     *            a String that record the maximum and the minimum occurrences
     * @requires orderedString should not be empty, htmlName should not be
```

```java
     *           empty, maxMin should not be empty
     */
    public static void creatHtml(SortedMap<String, Integer> orderedString,
            PrintWriter out, String htmlName, String maxMin) {
        assert orderedString != null : "Violation of: the orderedString is not null";
        assert htmlName != null : "Violation of: the htmlName is not null";
        assert maxMin != null : "Violation of: the maxMin is not null";

        /*
         * Output front part of html.
         */
        out.println("<html>");
        out.println("  <head>");
        out.println("    <title>" + "Top " + orderedString.size() + " words in "
                + htmlName + "</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("  </head>");
        out.println("  <body>");
        out.println("    <h2> Top " + orderedString.size() + " words in "
                + htmlName + "</h2>");
        out.println("    <hr>");
        out.println("    <div class = \"cdiv\">");
        out.println("      <p class=\"cbox\">");

        /*
         * Subtract the maximum and minimum value from maxMin.
         */
        int symbolpos = maxMin.indexOf(';');
        int max = Integer.parseInt(maxMin.substring(0, symbolpos));
        int min = Integer.parseInt(maxMin.substring(symbolpos + 1));

        /*
         * Output each line of middle part of html.
         */
        final int maxTypeSize = 48;
        final int minTypeSize = 11;
        for (Map.Entry<String, Integer> wordList : orderedString.entrySet()) {
            String word = wordList.getKey();
            int numOfWords = orderedString.get(word);
            int typeSize;

            /*
             * Use a formula to calculate the type size of each word.
             */
            if (max - min != 0) {
                typeSize = ((maxTypeSize - minTypeSize) * (numOfWords - min)
                        / (max - min)) + minTypeSize;
            } else {
                typeSize = maxTypeSize;
            }
            out.println("        <span style=\"cursor:default\" class=\"" + "f"
                    + typeSize + "\" title=\"count: " + numOfWords + "\">"
                    + word + "</span>");
        }

        /*
         * Output last part of html.
         */
        out.println("      </p>");
        out.println("    </div>");
        out.println("  </body>");
        out.println("</html>");

    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {

        BufferedReader in = new BufferedReader(
                new InputStreamReader(System.in));

        /*
         * Ask the users to enter the txt file name they want to check.
         */
        String fileName;
        try {
            System.out.println("Enter a txt file name: ");
            fileName = in.readLine();
        } catch (IOException e) {
            System.err
                    .println("There are some errors with entering file name..");
            return;
        }

        /*
         * Ask the users to enter the html file name they want to write in.
         */
        String htmlName;
        try {
            System.out.println("Enter a html file name: ");
            htmlName = in.readLine();
        } catch (IOException e) {
            System.err
                    .println("There are some errors with entering html name..");
            return;
        }

        /*
         * Ask the users to enter a number for the amount of the most frequent
         * words they want to check.
         */
        int num;
        try {
            System.out.println(
                    "Enter a positive number for the count of the most frequent words: ");
            num = Integer.parseInt(in.readLine());
            while (!(num > 0)) {
                System.out.println(num
                        + " is not a positive number, enter another number: ");
                num = Integer.parseInt(in.readLine());
            }
        } catch (IOException e) {
            System.err.println(
                    "There are some errors with enter the number for the count of the most frequent words.");
            return;
        }

        /*
         * Build a BufferedReader type to read the txt file.
         */
        BufferedReader fileContent;
        try {
            fileContent = new BufferedReader(new FileReader(fileName));
        } catch (IOException e) {
            System.err.println(
                    "There are some errors with reading the txt file.");
            return;
        }

        /*
         * Build a map which will store the contents in txt files.
         */
        Map<String, Integer> wordNum = new HashMap<String, Integer>();
        countingWords(fileContent, wordNum);

        /*
         * If users' value is lager the amount of all word, just arrange all the
         * words in the txt file.
         */
        if (num > wordNum.size()) {
            num = wordNum.size();
        }

        /*
         * Build a PrintWriter type to write html file.
         */
        PrintWriter htmlContent;
        try {
            htmlContent = new PrintWriter(
                    new BufferedWriter(new FileWriter(htmlName)));
        } catch (IOException e) {
            System.err.println(
                    "There are some errors with writing the html file.");
            return;
        }

        /*
         * Build a map to store the contents of the file. Build a comparator to
         * arrange. Build a PriorityQueue to order int value.
         */
        Comparator<String> orderString = new CompareString();
        Comparator<Map.Entry<String, Integer>> orderInt = new CompareNum();
        PriorityQueue<Map.Entry<String, Integer>> orderedInt = new 
PriorityQueue<Map.Entry<String, Integer>>(
                orderInt);
        SortedMap<String, Integer> orderedString = new TreeMap<String, Integer>(
                orderString);

        /*
         * Return a String that record the maximum and the minimum occurrences
         * int the map which called wordWithOccurrence.
         */
        String maxMin = sortingMap(wordNum, orderedInt, orderedString, num);

        /*
         * Output the content in html format to a html file.
         */
        creatHtml(orderedString, htmlContent, htmlName, maxMin);

        /*
         * Close all things.
         */
        try {
            in.close();
            fileContent.close();
            htmlContent.close();
        } catch (IOException e) {
            System.err.println("There are some errors with closing things");
        }
    }

}

```