```java
import java.util.Comparator;

/**
 * This Java program generates a HTML file with tag cloud from a given input
 * text.
 *
 * @author Zhuoyang Li + Xinci Ma
 *
 */
public final class TagCouldGenerator {

    /**
     * Set up the frequency for each word.
     */
    private static int minFrequency = 0;
    /**
     * Set up the frequency for each word.
     */
    private static int maxFrequency = 100;

    /**
     * Set up the font size for each word.
     */
    private static int minFontSize = 10;
    /**
     * Set up the font size for each word.
     */
    private static int maxFontSize = 50;

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

    /**
     * Sort the words with alphabetical order.
     */

    private static Comparator<Map.Pair<String, Integer>> alphaOrder = new Comparator<Map.Pair<String, Integer>>() {
        @Override
        public int compare(Pair<String, Integer> o1, Pair<String, Integer> o2) {
            // ignore case a = A
            return o1.key().compareToIgnoreCase(o2.key());
        }
    };

    /**
     * Sort the words with frequency order Positive for descending order.
     * Negative for ascending order
     */

    private static Comparator<Map.Pair<String, Integer>> frequencyOrder = new Comparator<Map.Pair<String, Integer>>() {
        @Override
        public int compare(Pair<String, Integer> o1, Pair<String, Integer> o2) {
            return o2.value() - o1.value();
        }
    };

    /**
     * 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 <pre>
     * {@code 0 <= position < |text|}
     * </pre>
     * @ensures <pre>
     * {@code nextWordOrSeparator =
     * text[ position .. position + |nextWordOrSeparator| ) and
     * if elements(text[ position .. position + 1 )) intersection separators = {}
     * then
     * elements(nextWordOrSeparator) intersection separators = {} and
     * (position + |nextWordOrSeparator| = |text| or
     * elements(text[ position .. position + |nextWordOrSeparator| + 1 ))
     * intersection separators /= {})
     * else
     * elements(nextWordOrSeparator) is subset of separators and
     * (position + |nextWordOrSeparator| = |text| or
     * elements(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 separators != null : "Violation of: separators is not null";
        assert 0 <= position : "Violation of: 0 <= position";
        assert position < text.length() : "Violation of: position < |text|";

        int i = position;

        if (!separators.contains(text.charAt(position))) {
            while (i < text.length() && !separators.contains(text.charAt(i))) {
                i++;
            }
        } else {

            while (i < text.length() && separators.contains(text.charAt(i))) {
                i++;
            }


        }
        return text.substring(position, i);
    }

    /**
     * Generate the HTML file with tag cloud.
     *
     * @param file
     *            the input file
     * @param m
     *            the map to store the frequency of each word
     */
    private static void countFrequency(SimpleReader file,
            Map<String, Integer> m) {
        assert file != null : "Violation of: file is not null";
        assert file.isOpen() : "Violation of: file is open";
        assert m.size() == 0 : "Violation of: m.size() = 0 (m is empty)";

        String separators = "\\t/()?!.,<>;:|[]{}~@#$%-\\";
        Set<Character> notIn = new Set1L<>();
        for (int i = 0; i < separators.length(); i++) {
            notIn.add(separators.charAt(i));
        }
        while (!file.atEOS()) {
            String line = file.nextLine();
            int i = 0;
            while (i < line.length()) {
                String word = nextWordOrSeparator(line, i, notIn);
                if (!m.hasKey(word)) {
                    m.add(word, 1);
                } else {
                    m.replaceValue(word, m.value(word) + 1);
                }
                i += word.length();
            }
        }
    }

    /**
     * calculate the font size of the word.
     *
     * @param frequency
     * @return the font size of the word
     */
    private static int wordSizs(int frequency) {
        int size = 0;
        if (frequency == minFrequency) {
            return minFontSize;
        }
        if (frequency == maxFrequency) {
            return maxFontSize;
        }

        // Precompute these values to avoid recalculating them for each word
        final int frequencyRange = maxFrequency - minFrequency;
        final int fontSizeRange = maxFontSize - minFontSize;

        // Linear interpolation between minFontSize and maxFontSize
        return (int) Math
                .floor(minFontSize + (double) (frequency - minFrequency)
                        / frequencyRange * fontSizeRange);
    }

    /**
     * Sort the map with frequency order.
     *
     * @param m
     *            the map to store the frequency of each word
     * @param cloudSize
     *            the size of the cloud
     * @return the sorted map
     */

    public static Map<String, Integer> sortFrequency(Map<String, Integer> m,
            int cloudSize) {

        SortingMachine<Map.Pair<String, Integer>> sm = new SortingMachine1L<>(
                frequencyOrder);
        for (Map.Pair<String, Integer> pair : m) {
            sm.add(pair);
        }
        sm.changeToExtractionMode();
        Map<String, Integer> sorted = new Map1L<>();
        int i = 0;
        while (i < cloudSize && sm.size() > 0) {
            Pair<String, Integer> pair = sm.removeFirst();
            sorted.add(pair.key(), pair.value());
            i++;
        }
        return sorted;
    }

    /**
     * Sort the map with alphabetical order.
     *
     * @param m
     *            the map to store the frequency of each word
     * @return the sorted map
     */
    public static SortingMachine<Map.Pair<String, Integer>> generateAlphabeticSortedMap(
            Map<String, Integer> m) {
        SortingMachine<Map.Pair<String, Integer>> sm = new SortingMachine1L<>(
                alphaOrder);
        for (Map.Pair<String, Integer> pair : m) {
            sm.add(pair);
        }
        sm.changeToExtractionMode();
        return sm;
    }

    /**
     * Generate the HTML file with tag cloud.
     *
     * @param file
     *            the input file
     * @param sm
     *            the sorting machine
     * @param cloudSize
     *            the size of the cloud
     * @param outName
     *            the output file name
     */

    public static void printHTML(SimpleWriter outName, String file,
            int cloudSize, SortingMachine<Map.Pair<String, Integer>> sm) {
        // Generate the Title
        outName.println("<html>");
        outName.println("<head>");
        outName.println("<title>Tag Cloud Generator</title>");
        outName.println(
                "<link href=\"tagcloud.css\" rel=\"stylesheet\" type=\"text/css\">");
        outName.println("</head>");

        // Generate the body
        outName.println("<body>");
        outName.println("<h2>Top " + cloudSize + " Words in " + file + "</h2>");
        outName.println("<hr>");
        outName.println("<div class=\"cdiv\">");
        outName.println("<p class=\"cbox\">");
        int i = 0;
        while (i < cloudSize && sm.size() > 0) {
            Map.Pair<String, Integer> pair = sm.removeFirst();
            outName.println("<span style=\"cursor:default\" class=\"f"
                    + wordSizs(pair.value()) + "\" title=\"frequency: "
                    + pair.value() + "\">" + pair.key() + "</span>");
            i++;
        }
        outName.println("</p>");
        outName.println("</div>");
        outName.println("</body>");
        outName.println("</html>");

    }


    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();
        out.println("Enter the file name done with txt: ");
        String fileName = in.nextLine();
        SimpleReader input = new SimpleReader1L(fileName);
        out.println("Please enter the name of output file: ");
        SimpleWriter output = new SimpleWriter1L("data/" + in.nextLine());
        out.println("Please enter the size for the tag cloud: ");
        int cloudSize = in.nextInteger();
        Map<String, Integer> str = new Map1L<String, Integer>();
        countFrequency(input, str);
        Map<String, Integer> topWords = sortFrequency(str, cloudSize);
        SortingMachine<Map.Pair<String, Integer>> sort = generateAlphabeticSortedMap(
                topWords);
        printHTML(output, fileName, cloudSize, sort);

        in.close();
        out.close();
        input.close();
        output.close();
    }

}
```