```java
import java.util.Comparator;

/**
 * Program to take a file of text, count each instance of every word, and
 * generate a tag cloud with each word and corresponding size
 *
 * @author Robert Frenken
 * @author Bennett Palmer
 *
 *         NOTE: This code handles all words to lower case
 */
public final class TagCloudGenerator {

    /**
     * Default constructor--private to prevent instantiation.
     */
    private TagCloudGenerator() {
        // no code needed here
    }

    /**
     * Compare {@code String}s in alphabetical order.
     */
    private static class StringLT
            implements Comparator<Map.Pair<String, Integer>> {
        @Override
        public int compare(Map.Pair<String, Integer> o1,
                Map.Pair<String, Integer> o2) {
            // use to lower case to group both capitalized and non capitalized
            return o1.key().toLowerCase().compareTo(o2.key().toLowerCase());
        }
    }

    /**
     * Compare {@code Integer}i in decreasing order.
     */
    private static class IntegerLT
            implements Comparator<Map.Pair<String, Integer>> {
        @Override
        public int compare(Map.Pair<String, Integer> o1,
                Map.Pair<String, Integer> o2) {

            return o2.value().compareTo(o1.value());
        }
    }

    /**
     * Generates the set of characters in the given {@code String} into the
     * given {@code Set}.
     *
     * @param str
     *            the given {@code String}
     * @param strSet
     *            the {@code Set} to be replaced
     * @replaces strSet
     * @ensures strSet = entries(str)
     */
    private static void generateElements(String str, Set<Character> strSet) {
        assert str != null : "Violations of: str is not null";
        assert strSet != null : "Violation of: strSet is not null";

        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
            if (!strSet.contains(c)) {
                strSet.add(c);
            }
        }
    }

    /**
     * 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 separators != null : "Violation of: separators is not null";
        assert 0 <= position : "Violation of: 0 <= position";
        assert position < text.length() : "Violation of: position < |text|";

        int endPosition = position;
        if (!separators.contains(text.charAt(position))) {
            // find the length of the word
            while (endPosition < text.length()
                    && !separators.contains(text.charAt(endPosition))) {
                endPosition++;
```

```java
            }
        } else {
            // find the length of the separator
            while (endPosition < text.length()
                    && separators.contains(text.charAt(endPosition))) {
                endPosition++;
            }
        }

        return text.substring(position, endPosition);
    }

    /**
     * Take file given and build a map, with each word to lower case as the map
     * key and the occurrence of each word as the map value.
     *
     * @param in
     *            the SimpleReader file
     * @ensures all words from inFile will be in map, with count of each word
     * @return Map<String, Integer> of words of the file and their counts
     */
    private static Map<String, Integer> readFileToMap(SimpleReader inFile) {
        assert inFile != null : "Violation of: inFile is not null";

        Map<String, Integer> map = new Map1L<>();

        /*
         * Define separator characters for test
         */
        final String separatorStr = " \t, .-!?_@#$%&*[]()";
        Set<Character> separatorSet = new Set1L<>();
        generateElements(separatorStr, separatorSet);

        // Read file, and compile all the words into map
        while (!inFile.atEOS()) {
            String oneLine = inFile.nextLine();
            int i = 0;
            while (i < oneLine.length()) {
                String word = nextWordOrSeparator(oneLine, i, separatorSet)
                        .toLowerCase();
                boolean isWord = true;
                for (int j = 0; j < word.length(); j++) {
                    char c = word.charAt(j);
                    if (separatorSet.contains(c)) {
                        isWord = false;
                    }
                }
                if (isWord) {
                    // add word to map or add one to count
                    if (map.hasKey(word)) {
                        int count = map.value(word);
                        map.replaceValue(word, count + 1);
                    } else {
                        map.add(word, 1);
                    }
                }
                i += word.length();
            }
        }

        return map;

    }

    /**
     * Takes map and integer value, and first sorts the map with sortingMachine
     * by occurrence of each word in decreasing order. Then makes a second
     * sortingMachine, and sorts alphabetically. numberDisplay determines the
     * number of words to be put in second sortingMachine.
     *
     * @param map
     *            map of all of the words and their counts
     * @param numberDisplay
     *            the amount of words that will be in the first sortingMachine
     * @requires map is not null
     * @ensures all Map.Pairs is sorted in decreasing order by their values
     * @return SortingMachine<Map.Pair<String, Integer>> of words and their
     *         counts in alphabetical order
     *
     */
    public static SortingMachine<Map.Pair<String, Integer>> mapToSortingMachineAlphabet(
            Map<String, Integer> map, Integer numberDisplay) {
        assert map != null : "Violation of: words is not null";

        Comparator<Map.Pair<String, Integer>> sortByCount = new IntegerLT();
        Comparator<Map.Pair<String, Integer>> sortByAlpha = new StringLT();
        SortingMachine<Map.Pair<String, Integer>> sortCount = new SortingMachine1L<>(
                sortByCount);
        SortingMachine<Map.Pair<String, Integer>> sortAlpha = new SortingMachine1L<>(
                sortByAlpha);

        // build the sorting machine with the map
        while (map.size() > 0) {
            Map.Pair<String, Integer> temp = map.removeAny();
            sortCount.add(temp);

        }

        sortCount.changeToExtractionMode();

        // build sorting machine in alphabetical order
        for (int i = 0; i < numberDisplay; i++) {
            sortAlpha.add(sortCount.removeFirst());
        }

        sortAlpha.changeToExtractionMode();

        return sortAlpha;
    }

    /**
     * Outputs the tag cloud given the words in SortingMachie
     *
     * @param sortAlpha
     *            the map sorted alphabetically
     * @param out
     *            the output stream
     * @param title
     *            the string of the file name
     * @updates out.content
     * @requires out.is_open
     * @ensures out.content = #out.content * [the HTML tags]
     */
    private static void outputhtml(
            SortingMachine<Map.Pair<String, Integer>> sortAlpha,
            SimpleWriter out, String title) {
        assert out.isOpen() : "Violation of: out.is_open";
        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + title
                + "</title><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\">"
                + "</head><body><h2>" + "Top " + sortAlpha.size() + " words in "
                + title + "</h2><hr>");

        out.println("<div class=\"cdiv\">");
        out.println("<p class =" + '"' + "cbox" + '"' + ">");
        int maxCount = 0;
        for (Map.Pair<String, Integer> p : sortAlpha) {
            if (p.value() > maxCount) {
                maxCount = p.value();
            }
        }

        while (sortAlpha.size() > 0) {
            Map.Pair<String, Integer> temp = sortAlpha.removeFirst();

            int fontSize = 37 * temp.value() / maxCount + 11;
            out.println("<span style=\"cursor:default\" class=\"f" + fontSize
                    + "\" title=\"count:" + temp.value() + "\">" + temp.key()
                    + "</span>");
        }
        out.println("</p>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments; unused here
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        out.print("Enter file that will be used to obtain the words: ");
        String textFile = in.nextLine();
```

```java
        SimpleReader inFile = new SimpleReader1L(textFile);

        out.print("Enter the name of a of the html page to write to: ");
        String htmlpage = in.nextLine();
        SimpleWriter outputhtml = new SimpleWriter1L(htmlpage);

        out.print(
                "Enter the number of words that will be displayed (integer): ");
        String numberToDisplay = in.nextLine();
        // checks if number is positive integer
        while (!FormatChecker.canParseInt(numberToDisplay)
                || !(Integer.parseInt(numberToDisplay) > 0)) {
            out.print(
                    "ERROR: Not Positive Integer \nPlease enter the number of words that will be displayed (integer): ");
            numberToDisplay = in.nextLine();
        }
        int numberDisplay = Integer.parseInt(numberToDisplay);

        Map<String, Integer> map = readFileToMap(inFile);

        SortingMachine<Map.Pair<String, Integer>> sortAlpha = mapToSortingMachineAlphabet(
                map, numberDisplay);

        outputhtml(sortAlpha, outputhtml, textFile);

        inFile.close();
        out.close();
        in.close();
    }

}

```