CSE 2231 – Software 2: Software Development and Design

Professor: Rob LaTour

Project #1: Word Counter

The Ohio State University

College of Engineering

Columbus, Ohio

```
import java.util.Comparator;

import components.map.Map;
import components.map.Map1L;
import components.queue.Queue;
import components.queue.Queue1L;
import components.set.Set;
import components.set.Set1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 *
 * This Java program counts word occurrences in a user-specified text (.txt)
 * file. It outputs a single well-formed HTML (.html) document displaying the
 * name of the input file in the heading, followed by a table listing the words
 * and their corresponding counts in lexicographic (alphabetical) order (clear
 * of Checkstyle and FindBugs warnings).
 *
 * @author Danny Kan (kan.74@osu.edu)
 *
 * @version January 20th, 2023
 *
 */
public final class WordCounter {

    /**
     * Compare {@code String}s in lexicographic (alphabetical) order.
     */
    private static class StringLT implements Comparator<String> {
```

```
    @Override
    public int compare(String o1, String o2) {
        return o1.compareTo(o2);
    }
}

/**
 * Default no argument constructor. It is private to prevent the
 * instantiation of this utility class.
 */
private WordCounter() {
    // no code needed here.
}

/**
 * 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) {
    assert str != null : "Violation of: str is not null";
    assert charSet != null : "Violation of: charSet is not null";

    for (char x : str.toCharArray()) {
        if (!charSet.contains(x)) {
            charSet.add(x);
```

```
        }
    }
}

/**
 * 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 index = position;
    if (separators.contains(text.charAt(position))) {
        while (index < text.length()
                && separators.contains(text.charAt(index))) {
            index++;
        }
    } else {
        while (index < text.length()
                && !separators.contains(text.charAt(index))) {
            index++;
        }
    }

    return text.substring(position, index);
}

/**
 * Generates a single well-formed HTML (.html) document displaying the name
 * of the input file in the heading, followed by a table listing the words
 * and their corresponding counts in lexicographic (alphabetical) order.
 *
```

```
 * @param wordsAndCounts
 *            the {@code String} word -> {@code Integer} count map
 * @param words
 *            a {@code Queue<String>} lexicographically (alphabetically)
 *            sorted queue of words
 * @param inputFileName
 *            the {@code String} name of the input file
 * @param outputFile
 *            the {@code SimpleWriter} output stream
 * @clears wordsAndCounts
 * @updates words
 * @requires <pre>
 * [the outputFile is open. wordsAndCounts, words, and
 *          outputFile are not null.]
 * </pre>
 */
private static void generateHTML(Map<String, Integer> wordsAndCounts,
        Queue<String> words, String inputFileName,
        SimpleWriter outputFile) {
    assert wordsAndCounts != null : "Violation of: wordsAndCounts is not null";
    assert words != null : "Violation of: words is not null";
    assert inputFileName != null : "Violation of: inputFileName is not null";
    assert outputFile != null : "Violation of: outputFile is not null";
    assert outputFile.isOpen() : "Violation of: outputFile.is_open";

    outputFile.println("<html>");

    outputFile.println("<head>");
    outputFile.println("<title>");
    outputFile.println("Words Counted in " + inputFileName);
    outputFile.println("</title>");
    outputFile.println("</head>");
```

```
outputFile.println("<body>");
outputFile.println("<h2>");
outputFile.println("Words Counted in " + inputFileName);
outputFile.println("</h2>");
outputFile.println("<hr>");
outputFile.println("<table border=\"1\">");
outputFile.println("<tbody>");
outputFile.println("<tr>");
outputFile.println("<th>");
outputFile.println("Words");
outputFile.println("</th>");
outputFile.println("<th>");
outputFile.println("Counts");
outputFile.println("</th>");
outputFile.println("</tr>");

Map<String, Integer> temp = wordsAndCounts.newInstance();
temp.transferFrom(wordsAndCounts);

while (temp.size() > 0) {
   String word = words.dequeue();
   Map.Pair<String, Integer> pair = temp.remove(word);
   outputFile.println("<tr>");
   outputFile.println("<td>");
   outputFile.println(pair.key());
   outputFile.println("</td>");
   outputFile.println("<td>");
   outputFile.println(pair.value());
   outputFile.println("</td>");
   outputFile.println("</tr>");
}
```

```
    outputFile.println("</tbody>");
    outputFile.println("</table>");
    outputFile.println("</body>");

    outputFile.println("</html>");
}

/**
 * Generates a {@code Map<String, Integer>} of words and corresponding word
 * occurrence counts from a user-specified text (.txt) file.
 *
 * @param wordsAndCounts
 *            the {@code String} word -> {@code Integer} count map
 * @param inputFile
 *            the {@code SimpleReader} input stream
 * @updates wordsAndCounts
 * @requires <pre>
 * [the inputFile is open and not null]
 * </pre>
 * @ensures [wordsAndCounts contains word -> count mapping from the
 *          inputFile]
 */
private static void generateMap(Map<String, Integer> wordsAndCounts,
        SimpleReader inputFile) {
    assert wordsAndCounts != null : "Violation of: wordsAndCounts is not null";
    assert inputFile != null : "Violation of: inputFile is not null";
    assert inputFile.isOpen() : "Violation of: inputFile.is_open";

    /*
     * This section is almost identical to the skeleton code from CSE 2221:
     * Software Components, Laboratory #23: Words and Separators.
```

```
 */
final String separatorStr = " `~!@#$%^&*()-_=+[{]}|;:',<.>/?";
Set<Character> separatorSet = new Set1L<>();
generateElements(separatorStr, separatorSet);

String str = inputFile.nextLine();

int position = 0;
while (position < str.length()) {
   String token = nextWordOrSeparator(str, position, separatorSet)
         .toLowerCase();
   /*
    * If there is no intersection between {@code Set<Character>}
    * separatorSet and {@code String} token, then it is the first word.
    */
   if (!separatorSet.contains(token.charAt(0))) {
      /*
       * If {@code Map<String, Integer>} wordsAndCounts does not
       * contain the {@code String} key, then add it to the map.
       * Otherwise, it is a duplicate: increment the occurrence count
       * and replace the existing value.
       */
      if (!wordsAndCounts.hasKey(token)) {
         wordsAndCounts.add(token, 1);
      } else {
         int count = wordsAndCounts.value(token);
         count++;
         wordsAndCounts.replaceValue(token, count);
      }
   }
   position += token.length();
}
```

```
}

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

    /*
     * Prompt the user to enter the input file name, including the folder
     * name / directory and the .txt extension.
     */
    out.print("Enter the input file name: ");
    String inputFileName = in.nextLine();
    SimpleReader inputFile = new SimpleReader1L(inputFileName);

    /*
     * Prompt the user to enter the output file name, including the folder
     * name / directory and the .html extension.
     */
    out.print("Enter the output file name: ");
    String outputFileName = in.nextLine();
    SimpleWriter outputFile = new SimpleWriter1L(outputFileName);

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

```
        while (!inputFile.atEOS()) {
            generateMap(wordsAndCounts, inputFile);
        }

        Queue<String> words = new Queue1L<>();
        for (Map.Pair<String, Integer> pair : wordsAndCounts) {
            words.enqueue(pair.key());
        }

        Comparator<String> order = new StringLT();
        words.sort(order);
        generateHTML(wordsAndCounts, words, inputFileName, outputFile);

        /*
         * Close input and output streams.
         */
        in.close();
        out.close();
        inputFile.close();
        outputFile.close();
    }

}
```