CSC 110: Fundamentals of Programming I
Assignment #5: 1D arrays, file input/output, and exceptions

Due date

Due date

Friday, November 4th, 2016 at 5:00 pm via submission to conneX.

How to hand in your work

Submit the requested files for Part (b) (see below) through the Assignment #4 link on the CSC 110 conneX site. Please make sure you follow all the required steps for submission (including confirming your submission). Part (a) is not marked.

Learning outcomes

When you have completed this assignment, you should understand:

- How to create a *one-dimensional (i.e., 1D) array*s.
- How to *read from and write to array elements*, using both *explicit and computed index values*.
- How to perform *simple file input and output* using *streams*.
- How to use *exceptions*.

Part (a): Problems from the Textbook

Complete Chapter 7 self-check problems 1 to 26 and compare your answers with those available at:

`http://www.buildingjavaprograms.com/self-check-solutions-4ed.html`

Part (b): Text Analysis

Problem description

A common task performed by computers is the analysis of text. For example, word processors often have a menu item to compute the number of characters, words and lines in the current document. Another example is where some text-compression algorithms first calculate the frequency with which individual words appear in a text file and then use that information to guide the algorithm used to compress the file. In this assignment your finished program will compute simple statistics for text files.

The program will open a file (with name of the file specified at run time) and read in its contents word by word. Specific words may appear once, twice, or many times in the text, and the program will keep track of how frequently a word does appear. Once file input is complete some additional statistics will be computed: the number of total words in the file, the number of words in the wordlist for the file (i.e., unique words), and the frequency of word lengths.

Consider the following small text file named `in01.txt`:

```
Ask not what your country can do for you
ask what you can do for your country
```

Note the text uses no punctuation. (The text files given to you for this assignment will not have punctuation at the start or end of words, but there may be punctuation within a word, as in "`it's`".) The text-analysis program is run using the command below:

```
java TextAnalysis in01.txt
```

and here is the output produced by the program:

```
TEXT FILE STATISTICS
--------------------
Length of longest word: 7 ("country")
Number of words in file wordlist: 9
Number of words in file: 17

Word-frequency statistics
  Word-length 1: 0
  Word-length 2: 2
  Word-length 3: 9
  Word-length 4: 4
  Word-length 5: 0
  Word-length 6: 0
  Word-length 7: 2
  Word-length 8: 0
  Word-length 9: 0
```

```
  Words-length >= 10: 0

Wordlist dump:
ask:2
not:1
what:2
your:2
country:2
can:2
do:2
for:2
you:2
```

The order of the words in "Complete word-frequencies" produced by your program may differ from what is shown above.

Lettercase is ignored when computing word frequencies (i.e., "Ask" ,"ask", "ASK", "aSK", etc. are considered the same word). Also, the longest word length is reported along with an example of the longest word (i.e., the text file may have multiple different words that are all of the longest length and so you need print only one of them).

A few more points:

- The maximum size of any file's wordlist is 10,000 words. Note that the actual number of words in the file itself will be larger. For example, the number of words in `in01.txt`'s wordlist is 9, but the number of words in the file is 17.

- Punctuation will not appear at the start of ends of words within test files; this is done in order to simply your construction and use of a wordlist. For example, the text shown above is normally quoted as "Ask not what your country can do for you, ask what you can do for your country.", but a Java scanner would treat "you," as a word when "next()" is called and this is not be the same as the word "you" (i.e., difference of a comma).

- Each text file has its own wordlist, that is, you will compute the wordlist for a file from scratch. This wordlist is, of course, discarded once the program finishes execution.

- The wordlist of words must be stored as an array of Strings. The frequency of each word must be stored as an array of integers. The same index used to access an element in either array would therefore correspond to the same wordlist word (e.g., if words[10] is "breakfast", then frequency[10] would be used to keep track of the number of times "breakfast" has appeared in the file).

- Take advantage of the fact that Java initializes String array elements to null and integer array elements to 0.

- Break down your solution into several methods and write small little test programs that try out smaller methods. For example, you can write a method called findWord() which takes two parameters (array of Strings and a String) and returns an integer indicating the index location in which string parameter is located in the array parameter. If the word was not in the array, then the method can add the word to the next available element in the string array. (The first String having the value `null` can be considered the end of the wordlist.)

- First read the words of the file and construct the wordlist plus word frequencies. After that is done, then go ahead and compute some of the other statistics required for output by using the String and integer arrays you constructed during file reading (i.e., you must read the file only once).

File to submit: TextAnalysis.java

Grading scheme

- "A" grade: An exceptional submission demonstrating creativity and initiative going above and beyond the assignment requirements. The program runs without any problems and uses arrays, streams, exceptions and breaks down the program into several methods. Any extra work appears in a file named *TextAnalysisExtra.java*, and identified within the file (i.e., Class comment) is how you have extended the assignment to demonstrate creativity and initiative.

- "B" grade: A submission completing the requirements of the assignment. The program runs without any problems and uses arrays, streams, exceptions breaks down the program into several methods.

- "C" grade: A submission completing most of the requirements of the assignment. The program runs with some problems but uses arrays, streams, exceptions and is broken down into several methods, and yet might not have the expected output.

- "D" grade: A serious attempt at completing requirements for the assignment. The program runs with major problems, or does not use arrays, streams, exceptions or several methods.

- "F" grade: Either no submission given, the submission does not compile, or submission represents very little work.