Homework Assignment 1

Posted: 1/19/2010
Due: 1/26/2010

The assignment is due in class; please follow the instructions on the homework submission form.

Objectives

Apply features of object-orientation (specialization via inheritance, interfaces) in Java to achieve code reuse when solving closely related problems.

Develop a basic framework for the next assignment.

Familiarize yourself with the programming style requirements and the homework submission process.

Related Files

Homework submission form:
http://www.cs.unc.edu/Courses/comp524-s10/hw/submission-form.pdf

Text of Herman Melville's novel "**Moby Dick, or, the whale**" (Project Gutenberg version):
http://www.cs.unc.edu/Courses/comp524-s10/hw/1/mobydick.txt

Part 1

Implement a Java program, named `Frequency`, that lists the most-frequently used words in a text file (or, alternatively, the standard input stream) and the number of times that they occur.

`Frequency` must accept one or two command line arguments. The first argument specifies a limit on the number of words that should be displayed, e.g., if the first argument is "20", then only the 20 most-frequently occurring words should be displayed. The second argument specifies the name of a text file. If the second argument is omitted, then the program should process the standard input stream (`System.in`).

Your implementation of `Frequency` should handle special cases (file empty) and error cases (invalid numbers, file not readable) gracefully (e.g., display an error message and exit).

For the purpose of this assignment, a word is simply a sequence of non-whitespace, non-punctation, non-bracket characters, i.e., a word contains neither a space, a newline separator, nor one of the following characters: `.:;,?!-"'()[]{}`. Consecutive words are separated by at least one non-word character.

Part 2

Reusing the classes developed for Part 1, implement a Java program, named `Count`, that determines how often a given set of words is used in a text file (or, alternatively, the standard input stream).

`Count` must accept one or two command line arguments. The first argument specifies the name of a file that contains a list of words (one word per line). The second argument specifies the name of a text file. If the second argument is omitted, then the program should process the standard input stream (`System.in`).

Given a word list (specified by the first argument), `Count` must output the number of times that each word appears in the specified text file (second argument, or `System.in` if omitted).

Your implementation of `Count` should handle special cases (either file empty) and error cases (invalid filenames, either file not readable) gracefully (e.g., display an error message and exit).

Part 3

Use your implementations of `Count` and `Frequency` to examine Herman Melville's novel "Moby Dick, or, the whale". Specifically, determine

a) the 30 most-frequently used words, and

b) how many times the words "harpoon", "whale", "blood", "love", "peace", and "flowers" occur.

Example

Suppose we are given the following two files, `wordlist.txt` and `story.txt`.

Content of `wordlist.txt`

`is`
`short`
`moon`

Content of `story.txt`
`This is a rather short story in a short file.`

Invoking `Count` on these two files should result in output similar to the following.

Output of `java Count wordlist.txt story.txt`

`1 is`
`2 short`
`0 moon`

Invoking `Frequency` with a limit of 5 on the file `story.txt` should result in output similar to the following.

Output of `java Frequency 5 story.txt`

`2 short`
`2 a`
`1 story`
`1 is`
`1 file`

Guidelines

You may discuss possible approaches with other students, but you **cannot share source code**.

Your solution may use any class that is part of the Java standard library (J2SE 6, see [3]).

Hints:
- Make use of the Java Collections Framework and the `Iterator`/`Iterable` interfaces.
- Have a look at the `StringTokenizer` class.

Deliverables

The Java source code files that implement `Count` and `Frequency`. Make sure that your code can be compiled manually with `javac`.

A text file (plain text or PDF) that contains the answers to Part 3.

Grading

15 points — Functionality. Do `Count` and `Frequency` work as specified? (This covers Part 3).

05 points — Is the source code documented?

05 points — Style. Does the code look like a clean Java program?

05 points — Is the design properly object-oriented? Is code reused among the two programs?

Style Guide

> *"Programs must be written for people to read, and only incidentally for machines to execute."*
> — Abelson & Sussman, The Structure and Interpretation of Computer Programs

> *"How do we convince people that in programming simplicity and clarity —in short: what mathematicians call "elegance"— are not a dispensable luxury, but a crucial matter that decides between success and failure?"*
> — Edsger W. Dijkstra

Use a clean and consistent coding style. The standard Eclipse style is ok. Spaces are preferred over tabulators. See [1] or [2] for detailed style guides that are acceptable.

Make sure you structure classes consistently (i.e., either first attributes, then constructors, followed by methods, or the other way around, but the order should be consistent across classes).

Do not use unstructured control flow constructs (`break`, `continue`). However, you may use `System.exit()` to simplify argument checking in the `main` method.

Implement proper exception handling (the `main` method should not throw any exceptions).

Write the program in an object-oriented style. The problem should be solved by collaborating objects. Make only sparse use of static methods; ideally, `main` should be the only static method.

Write brief and concise comments. Just get the idea across; do not spend too much time on documentation.

References

[1] http://developers.sun.com/sunstudio/products/archive/whitepapers/java-style.pdf

[2] http://incubator.apache.org/ace/java-coding-style-guide.html

[3] http://java.sun.com/javase/6/docs/api/index.html