CIS*2430 (Fall 2010) Assignment One

Instructor: F. Song

Due Time: October 4, 2010 by midnight.

In this assignment, you will implement a simple "LibrarySearch" program which allows the user to enter and search for library information. You will have a chance to become familiar with `Java` and `NetBeans` environments and establish good habits for testing and documentation.

More specificially, you need to implement the following components in the "`main`" method of your "`LibrarySearch`" class:

(1) A command loop that accepts one of the three commands: add, search, and quit. The add command allows the user to enter a book or journal record; the search command helps the user find all matched records for a set of keywords; and the quit command terminates the command loop and the program.

(2) Each record is entered on a single line and must contain the call number, the title, the publisher (for a book), and the organization (for a journal), with all fields separated by commas. For example, we can have a book "QA76.64.c62, Object Oriented Programming, Yourdon Press" and a journal "QA76.A772, Communications of the ACM, Association for Computing Machinery". Inside your program, you should store all records in an array of Strings up to a certain number. If there is no room in the array, you should let the user know and no new records can be added to the array. You can also assume that the call numbers are unique for each record. As a result, you should check if a record already exists before adding it to the array. If the new record is a duplicate, you should report an error and let the user know.

(3) The search command takes a set of keywords as a query. To find all matched records, you can scan the array sequentially from the start and the end. You should tokenize both the query and the internal records so that words can be matched properly. Only the records that contain all the keywords in a given query should be returned as the result. In addition, since the words in the query and the records may be stored in a different mix of upper- and lower-case letters, you should ignore the cases when matching two words. All the matched records will be shown on the screen, one on a separate line.

You can easily find some book and journal information from the University library or you can make some data yourself in order to test your program. Note that you may need to use some predefined methods for String, Scanner, and StringTokenizer. So make sure that you read the related API descriptions or the textbook for further information and examples.

Deliverables:

All implementations should be done individually using Java. Your program will be marked for both correctness and style. By correctness, we mean that (1) your programs are free of syntactic errors and can be compiled successfully; (2) your programs are logically correct; and (3) your

programs should give appropriate runtime messages (i.e., prompts) and are reasonably roboust in handling user input. To make sure your programs are logically correct, you need to prepare a test plan along with a set of test cases. The test plan describes the major steps involved in testing your programs and whether you have considered all possible conditions. The test cases provided concrete examples that can be used for testing. To ensure that your programs are reasonably roboust in handling user input, you need to exercise defensive programming. Although you have clearly show a prompt asking for a certain kind of input (e.g., "quit"), the user may enter a shorter input (such as "q" or "Q") or something quite different (e.g, "bye" or "leave"). Your program should accept most of the reasonable values (such as "q" or "Q"), but reject all the irrelevant values (such as "bye" or "leave"). Furthermore, for the irrelevant values, you should give users feedback messages and ask them to re-enter the required values.

A good style in programming allows people to understand and maintain (modify or extend) your programs easily. This often includes (1) meaningful names and well-indented layout for control structures (branches, loops, and blocks); (2) internal documentation (the various comments within your programs); and (3) external documentation (additional description outside your programs, usually the README file). In Java, different naming conventions are used for representing classes, variables and methods, and symbolic constants. There is no limit about the length of an identifier; so try to use meaningful names for the sake of readability. For internal documentation, Java introduces "Javadoc", which will automatically turn comments in /** some comments here */ or /* some comments here */ before public classes, public instance/class variables, and public instance/class methods into a set of web documents. In addition to Javadoc comments, the traditional comments can still be used to explain the meanings of local variables and highlight the major steps within a particular method. For external documentation or the README file, you can describe the general problem you are trying to solve; what are the assumptions and limitations of your solution; how can a user build and test your program; how is the program tested for correctness (i.e., the test plan should be part of the README file); and what possible improvements could be done if you were to do it again or have extra time available.

In summary, a complete submission should include: (i) a README file; (ii) all the Java source files; (iii) JavaDoc files; and (iv) all the testing files if needed.

Organize your program and related files into appropriate directories, and tar and gzip all of your files/directories into one compressed file (name it in the form of <userid>_a<#>.tar.gz, e.g., fsong_a1.tar.gz) and drop it to the course website on moodle.cis.uoguelph.ca by the due time. Please verify afterwards that your submission is indeed uploaded successfully to the moodle server.