CIS*2430 (Fall 2010) Assignment Two

Instructor: F. Song

Due Time: October 25, 2010 by midnight.

In Assignment One, you implemented a simple "Library Search" program that stores all records in an array of Strings and performs "add" and "search" functions for the records. For the remaining assignments, you will re-design and extend this project by using the object-oriented concepts in general and some specific features of the Java language.

General Description

A library typically holds different kinds of references and allows you to add and search for relevant references. A reference is better modeled by an object so that you can distinguish different fields and apply suitable methods for accessing and processing these fields. For this project, we limit ourselves to two kinds of objects: Book and Journal. A book object should have a call number, a set of authors, a title, a publisher, and a year. A journal object should have a call number, a title, an organization, and a year. Here are two examples illustrating the relevant fields for book and journal objects, respectively:

```
QA76.73.J38S265                    P98.C6116
Walter Savitch,  Kenrich Mock      Computational Linguistics
Absolute Java                      Association for Computational Linguistics
Addision-Wesley                    2008
2009
```

One limitation with an array is that you have to declare a fixed size, which is not convenient, since a library frequently adds new references (but rarely deletes existing references). To avoid such a problem, you should use ArrayList in Java, which is dynamic and can grow its size as needed. In addition, a call number is not quite unique for each reference. For example, a book may have multiple editions, and all of them may share the same call number. Similarly, a journal is typically organized into volumes and all of them will share the same call number. One possible solution is to use the call number and the year together to identify each reference, which are generally unique for most libraries. As a result, when adding a new book or journal, you need to search the corresponding ArrayList and check the values of call numbers and years to make sure that the same object does not already exist in the system. If it already exists in the system, you should reject the new addition.

To search for relevant references, the user can specify a search request and only the objects that match the entire search request will be returned as the result. In Assignment One, a search request is simply a set of keywords, but generally, a search request may involve more fields such as a call number, a set of title keywords, a time period (with starting and ending years), or any combinations of these fields. For example, a search request may just contain a call number "P98.C6116" and in this case, all objects with the same call number will be returned. If a search request contains more fields, e.g., "P98.C6116" as the call number and "2000-2009" as the time period, then all objects with the same call number during years 2000 and 2009 will be returned.

There are several special cases for the time period. For example, "2008-2008" means year 2008 only; "2008-" means year 2008 and later; and "-2009" means year 2009 and before. For any of these fields, if the input is an empty string, it will match any value in the corresponding field of an object. In one extreme case, if you will leave all the fields empty for a search request, then all the objects will be matched and returned as the search result.

Specific Requirements for Assignment Two

(1) You need to extend the command loop in Assignment One to accept these commands: *add*, *search*, and *quit*. For the *add* command, the user needs to enter the type of an object (either book or journal), followed by the other required fields for the corresponding object type. Note that some fields can be empty, but the call numbers, titles, and years are always required for each object. For the *search* command, the user needs to provide values for three components: a call number, title keywords, and a time period (with starting and ending years). Note that the user can enter an empty string (i.e., no value) for any or all of the components. The *quit* command simply terminates the command loop and then the program.

(2) As described earlier, you should use two ArrayLists for storing objects: one for books and the other for journals. When adding a new object to the corresponding list, you still need to check for duplicates, but the unique key is a combination of call number and year, since the call number alone may correspond to multiple objects in the system. If the key of an object already exists, you should reject the new addition. Note that you no longer need to check for the maximum size of a list, since ArrayList is dynamic and can grow as long as needed.

(3) Based on the description so far, your implementation should include at least three classes: Book, Journal, and LibrarySearch. The Book class should contain a call number, a set of authors, a title, a publisher, and a year. The Journal class should contain a call number, a title, an organization, and a year. For both book and journal objects, we should always have a call number, a title, and a year. The other fields can be empty if not available. In addition, a year should be a four digit number between 1000 and 9999 in order to reject some irrelevant values. The LibrarySearch class is the most challenging, since it needs to maintain two ArrayLists for books and journals, search the lists sequentially for the matched references, and display the result on the screen. Based on the reference type, you can decide which list to search; for the time period, we need to make sure that it covers the year of the matched object; and for the title keywords, we need to make sure that they all appear in the title field of an object, although they can be in a different order. For all the class definitions, you need to follow the conventions by making all the instance variables private and providing suitable accessor and mutator methods. You should also provide suitable constructors and any other commonly used methods for a class, including the "equals" and "toString" methods. Any methods that work on an individual book or journal should be defined in its corresponding class, but any methods that work on a list of books or journals should be defined in the LibrarySearch class.

(4) You should organize all of your classes in one package and use Javadoc to create a set of external documents so that the TA can examine the contents of your package easily for the marking purpose.

Deliverables:

All implementations should be done individually in 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 the user some feedback message and ask the user 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 benefit 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_a2.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.