

Library Catalog Management

This project gives you more practice programming from a design. The design uses both aggregation and association.

The program implements the management of a simple library catalog system. This system will allow the registration of patrons, adding books, checking out and checking in books, and a number of searches to print information about overdue books, books checked out to patrons, and the availability of books. The program is driven by a command interface that reads from a text file. The commands and the appropriate action are described below.

Commands and Input Format:

The program will read input from a file named `library.data`. Each line of the file will be one of the following commands. All items in each command are tab delimited. If the command is cancelled because of an error, a useful message should be printed. Unless stated otherwise command parameters are all strings that are guaranteed not to contain a tab character. The first line in the file is guaranteed to be a `setdate` command.

`setdate` `<date>`

Set the system date. The “date” is an integer in the range 0 to 30,000.

`add` `<name>` `<title>` `<call #>`

Add the book with the given author, title and call number to the catalog of books. The author name, title, and call number are all strings of arbitrary length. The book should initially be available, and the call number should be unique. If the call number is already in use, the book should not be added, and the addition should be cancelled.

`register` `<name>` `<address>` `<patron-id>`

Add the named library patron to the list of patrons. Name and address should be treated as strings. The patron id is a string of six digits, and is unique to each patron. The command should check that no other patrons exist in the patron list with the same id. If there is a conflict, an error message should be printed, and the registration cancelled.

`checkout` `<patron-id>` `<call #>`

Check out the book with the given call number to the patron with the given identification number. The book should become unavailable, and the due date should be one week from the current date (current numeric date plus 7). If the patron has any overdue books (books whose due date is strictly before the current date), then the book should not be checked out.

`checkin` `<call #>`

Check in the book with the given call number. If the book is overdue (due date strictly before the current date), print a message listing the patron’s id, name, and address and the number of days overdue. If the book is not checked out, or is not in the catalog, the check in is cancelled.

`list` `all`

List all books in the catalog. Available books should be listed first, sorted by call number. Unavailable items should be listed after all available items, and should also be printed in sorted order by call number. The information for unavailable books should be preceded by a “*”. For each book, print the call number, title, and author name. For books that are checked out, also print the patron id, patron name, and due date.

`list` `overdue`

List all books that are checked out, and are overdue (meaning that the due date is strictly before the current date). For each book, list the call number, title, patron id, and patron name. Print the information sorted by call number. If no books are overdue, print a useful message.

`list` `<patron-id>`

List all books checked out to the particular patron. For each book print the call number, title, author name, and due date. Print the information sorted by call number.

`availability` `<call #>`

If the book with the given call number is available, print “Available:” followed by the call number, author name, and title. If the book is checked out, print “Out:” followed by the call number, author name, title and due date. If a checked out book is overdue, then print “Overdue:” before the information for the book. If the call number doesn’t exist in the catalog, print a useful error message.

```
find <call #>
    Find a book by its call number, and print the call number, author, and title. If no book with the call number exists
    in the catalog, print a useful error message.

find <patron-id>
    Find the information on the patron. Print the id, name, and address of the patron with the given id. If no patron
    exists with the given id, print a useful error message.
```

Sample Input:

```
setdate 0
add Doctor Roe, E. L. Swimming Upstream in Spring F123.4 v5
add Pain, Major Bits and Bytes, and Everything Mites Q124 45.3a
add Dog, Chubby Proper Weaning for Weimeraners V86.4 DOG
list all
register I. Reed 2000 Book Worm Way 000000
register U. Right 1 Penn St. 100000
checkout 000000 F123.4 v5
checkout 100000 F123.4 v5
setdate 1
checkout 100000 Q124 45.3a
list overdue
setdate 2
checkout 000000 V86.4 DOG
list 000000
list 100000
setdate 7
list overdue
setdate 8
list overdue
checkin F123.4 v5
```

Output description and sample:

The output of the program should go to a text file named `library.out`. The output for the input given above should look something like.

```
Current date: 0
All books in catalog, Date: 0
F123.4 v5 Doctor Roe, E. L. Swimming Upstream in Spring
Q124 45.3a Pain, Major Bits and Bytes, and Everything Mites
V86.4 DOG Dog, Chubby Proper Weaning for Weimeraners

Error: item with call number F123.4 v5 is not available
Current date: 1
No books currently overdue.
Current date: 2
Items checked out to: I. Reed, Patron ID: 000000
F123.4 v5 Doctor Roe, E. L. Swimming Upstream in Spring
V86.4 DOG Dog, Chubby Proper Weaning for Weimeraners

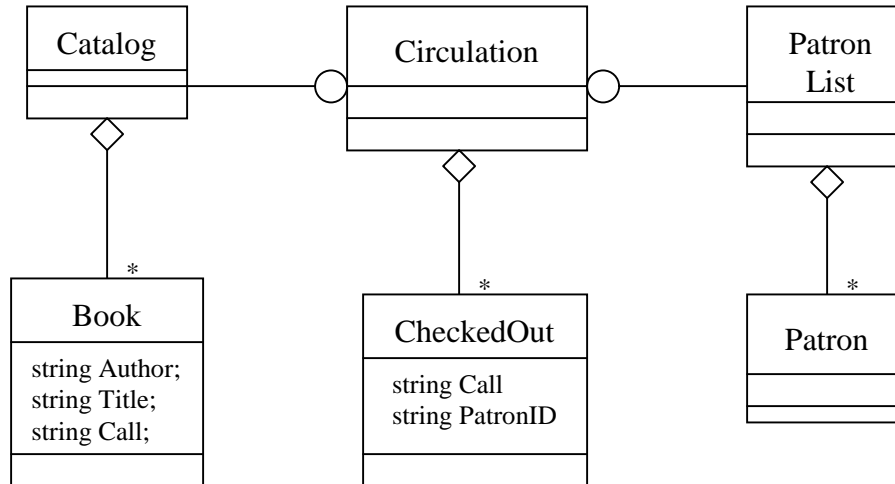
Items check out to: U. Right, Patron ID: 100000
Q124 45.3a Pain, Major Bits and Bytes, and Everything Mites

Current date: 7
No books currently overdue.
Current date: 8
Overdue books:
F123.4 v5 Swimming Upstream in Spring 000000 I. Reed

Book checked in is overdue: Patron ID: 000000, Name: I. Reed, Days overdue: 1
```

Design:

You must use the following classes, and class diagram when implementing your program. You may use other classes as you feel are necessary (parser, list nodes, etc.), but you may not change the relationships between the classes shown in the diagram.

**Classes:**

Book – information about a book, including author name, book title, and call number.

Patron – information about a library patron, including name, address, and patron id.

Catalog – catalog of library books, implement as list sorted by call number

Patron List – list of people able to checkout books, list sorted by patron id.

CheckedOut – record of checked out book. Stores call number, patron id, and due date.

Circulation – list of currently checked out books

Deliverables:

Your final project submission must include the following:

- All source code (`*.cpp` and `*.h` files) comprising your project.
- MS Visual C++ project files (`dsp` and `dsw`) or Unix `makefile`, as appropriate. (VC++ users: do not submit unnecessary files, such as `ncb`, `opt`, `ilk`, `obj`, `pch`, or `pdb` files).
- One set of input/output files from your testing.
- A brief ASCII text readme file, named `readme.txt`, containing:
 - Your name and e-mail address.
 - Your instructor's name, and the section of the course you're enrolled in (example: 8:00 MWF).
 - The project name.
 - The platform and compiler you're using (example: MSVC++ 6.0 under NT).
 - Any special execution instructions. (If your program is not fully functional, be sure to mention that here and to follow the directions in the Evaluation section below.)
- An ASCII text file, named `pledge.txt`, containing the Honor Code Pledge listed at the end of this file.

Your project submission will consist of a zipped archive file containing all of the items specified above.

You are allowed to submit your solution up to five times, in case you detect (or solve) problems after your first submission. Your last submission will be the only one tested and graded. Note that, due to late penalties, it is possible that a fixed but late submission may receive a lower score than a faulty but on-time submission.

You will submit this assignment to the Curator System (as for Homework 1), and you will demonstrate its operation to the GTA of your section. In order to receive any credit for this assignment, you are required to meet with the GTA, even if your program does not compile and run.

The submission client can be found at: <http://spasm.cs.vt.edu/eags/>

You may need to download the Java 1.2 runtime environment, and the curator client to properly submit.

Evaluation:

Your score on this project may be based on several factors:

- Runtime testing of your program, and correctness and completeness of output.
- Adherence of implementation to design, and adherence to good software engineering practice.
- Quality of internal documentation.

The relative weighting of these factors will be decided later. To receive partial credit for programs that are non-working, or are not fully functional, a brief one or two paragraph description of the problem(s) must be included in the assignment submission, in an ASCII text file named `problems.txt`. The location of the problem, minimally identified to a specific function or functions, must also be specified along with possible corrections that need to be made.

Programming Standards:

We will be evaluating your source code on this assignment for the quality of internal documentation and programming style, so you should observe good practice. Here's a brief description of some of the things we'd normally expect:

Documentation:

- You must include the honor pledge (below) in the header comment of the file containing `main()`.
- Your header comment must describe what your program does.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose.
- Precede every function you write with a header comment. This should explain in one sentence what the function does, then describe the logical purpose of each parameter (if any), describe the return value (if any), and state reasonable pre- and post-conditions.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable.

Coding:

- Use named constants instead of variables where appropriate.
- Use an struct or class variables to organize related heterogeneous data.
- Make good use of user-defined functions in your design and implementation. The body of `main()` should contain no more than 20 executable statements and the bodies of the other functions you write should each contain no more than 40 executable statements. An executable statement is any statement other than a constant or variable declaration, function prototype or comment. Blank lines do not count.
- The definition of `main()` must be the first function definition in your source file. You may use file-scoped function prototypes and you may use file-scoped constants. You may also make any type definition statements file-scoped.
- You may not use file-scoped variables of any kind.
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter. Pass any large objects, such as array parameters, by constant reference (using `const`) when pass-by-reference is not needed.
- Use string objects to store character data, not `char` arrays.
- Separate the parsing of input from the execution of input commands.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header preceding your `main()` function:

```
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Automated Grader.
```

Failure to include this pledge in a submission is a violation of the Honor Code.