Capstone Project for C# Programming Certificate

The application you will build is a Library Information Storage and Retrieval System to be used by a small library. The program will keep track of book titles, authors, copies, and allow you to retrieve that information in a variety of ways. This is a program that might be running on a machine within the library for visitors (patrons) to use.

A test plan is included at the end of this document. You should be able to run through this test plan at a minimum and mark each scenario as a pass. You should then expand your testing to other scenarios and alternatives to make sure your application is as bullet proof as possible.

## Requirements:

### Main Interface

As the program starts, the user will be shown the following three options:

1. Find a book by Title, Author, Subject, or ISBN
2. Librarian Login
3. Exit the program (save)

If the user selects option 2 (and provides the appropriate user ID and password), they get to the librarian menu:

1. Find a book by Title, Author, Subject, or ISBN
2. Check out a book
3. Check in a book
4. Add book(s) to the system (make sure the book doesn’t already exist)
5. Update an existing book in the system (requires a look up, editing, and then an overwrite of the existing recording in the data source. Also, you should allow the user to abandon changes made before they are finalized).
6. Remove book(s) from the system (requires a look up and a verification that the book can be deleted (not currently checked out)).
7. Display the following lists:
8. Librarians – id, first name, last name, phone (sorted by last name, first name). \*1
9. Cardholders – id, first name, last name, library card id, phone, books currently checked out (book id, ISBN, title, check out date, indicate if overdue) (sorted by last name, first name, book title). \*1
10. Authors – id, first name, last name, books authored (book id, ISBN, title, publisher) (sorted by last name, first name, book title). \*1
11. Books currently overdue (more than 30 days past the check-out date and no check-in date exists) – book id, ISBN, title, check out date, card holder info (id, first name, last name, library card id, phone) (sorted by check out date, book title).

\*1 NOTE: Polymorphism is to be applied when displaying librarians, cardholders, and authors. No duplicate code should exist for formatting and displaying the ID, first name, and last name of these three types.

Below are the classes with the information that needs to be available:

* Book
  + Book ID (int) (must be unique)
  + ISBN (string) (must be unique)
  + Title (string)
  + Author ID (int) (link to Author class)
  + NumPages (int)
  + Subject (string)
  + Description (string)
  + Publisher (String)
  + Year Published (int) (Range allowed is 1400 – current year)
  + Language (one string listing all languages is sufficient here)
  + Number of copies (int)
* Author \*2
  + ID (int) \* (ID for author, card holder, and librarian must be unique across all three classes)
  + First name (string)
  + Last name (string)
  + Bio (string)
* Cardholder \*2
  + ID (int) \* (see comment for ID in Author class)
  + First name (string)
  + Last name (string)
  + Phone (string)
  + Library Card ID (string) (must be unique)
* Librarian \*2
  + ID (int) \* (see comment for ID in Author class)
  + First name (string)
  + Last name (string)
  + Phone (string)
  + User ID (string) (combined values for user id and password must be unique)
  + Password (string)
* CheckOutLog
  + ID (int) (must be unique)
  + Cardholder ID (int) (link to Cardholder class)
  + Book ID (int) (link to Book class)
  + Date / Time checked out
  + Date / Time checked in (must be >= to Date / Time checked out). Note: This field may not be needed, if you choose to remove the CheckoutLog entry when the book is checked in.

\*2 NOTE: The hierarchical structure for these 3 class types needs to be setup so that the three common fields, ID, first name, and last name, are in a separate base class that is the base class for all of these three class types. It should not be allowed for the base class to be directly instantiated. Only the three class types, librarian, author, and cardholder, can be directly instantiated.

### Lookup Menu Items

Here are the requirements for the user lookup menu:

* The program starts in Lookup / Patron mode. If the user selects the first option (Find), they are asked to enter a search string.
* Once the search string is validated the program lists out books where the title, author, subject, or ISBN has the search string within it.
  + For example, searching for “Tom” might display “Tom Sawyer” as well as books by Tom Clancy.
  + If no match is found, then indicate that.
* This portion of the user interface (UI) should simply display the key elements about the book (book ID, title, author (first name, last name), publisher, year published), one line per book. (You will need to retrieve the author’s name from the instance of the author where the ID matches the author ID of the book being displayed.)
* There should be some visual indicator next to each book showing if all the copies of the book are checked out.
* The user should be able to select any one of these books in the list and go to a detailed screen showing all the information about the book (title, author, number of pages, subject, description, publisher, year published, languages supported, number of copies available, total number of copies). NOTE: You will need to retrieve information from the collection containing the CheckoutLog instances, counting the number of matching book IDs with a check-out date and no check-in date. This count will need to be subtracted from the total number of copies of the book to get the number of copies available.
* From the book details screen, the user should be able to go back to the previous list of books (to select another one or to exit to the main menu).
* For the summary list of results (after the user does a search), show which mode the user is in (patron or librarian) because you will reuse this screen for the librarian search option.

### Librarian Menu

The following are the requirements for the Librarian Menu:

* The Librarian menu is accessed from the main menu via option #2. Before the Librarian menu is shown, the user’s credentials should be validated. (Matching user ID and password).
* The librarians can perform several functions from the Librarian Login option.
* The first option of the Librarian menu allows the employee to search for a book just as patrons do (this allows them to search without leaving the Librarian menu). The code for this option in the librarians and the code in the patron’s menu option should not be duplicated. The search screen should show what mode the user is in (Library Patron vs. Librarian).
* The next two options allow the librarian to check out or check back in a book. The options should ask the user for the patron’s card number, and then one or more ISBN numbers of books to check out/check in.
* As books are checked out, a CheckoutLog entry must be added with the current date & time in the check-out date field. (NOTE: A card holder with overdue books will not be allowed to check-out any books until the overdue book(s) are returned.)
* For checking in, the CheckoutLog entry for the specific book must be located and updated to show the check in date & time, or remove entry, if you choose that option.
* At any given time, the CheckoutLog collection should minimally contain all books that are currently checked out with their check out dates and time and no entry in the check in date. It is your option as to whether you choose to remove the CheckoutLog entry at the time the book is being checked in. If you choose this option, then the check-in date field is not needed. Note that several options will require you to calculate how many copies of a specific book are available to check out. Make sure this code is not duplicated.
* The other 3 options of the Librarian menu allow the librarian to add/update/remove a book from the system.
  + Start by asking for the ISBN number.
  + For add/remove, ask for the number of copies.
  + If you are adding a book and the ISBN number they entered is already in the system, simply add the number of copies to the existing record (there should only be one record for each book).
  + If you are removing the book, you can either make the count 0 (it should not show up in patron-based searches), or remove the record completely (your choice). Allow the librarian to cancel the delete before it’s committed.
  + If you are adding new books:
    - Continue by asking for the information about the book. As you ask for the Author’s name, if the author’s name doesn’t already exist in the Author’s table, get the information necessary and create a new Author record (not needed if the author already exists). Then get the author’s ID. It will be needed as part of the information for creating a book record below.
    - At this point, ask for the rest of the information in the Book class and create a book record.
    - Be sure to provide a way for the librarian to abandon the addition.
  + For updating an existing book, you should use the searching functionality that both the users and librarians use, and then, once a book is selected, provide an edit screen for the librarian to modify the data. Once the librarian is done, be sure to ask if they want to commit the changes. Also provide a way to abandon the changes.

### Data Storage

You are being given a SQL Server database, LibraryInformation2. Your application must set up a connection to this SQL Server database. The tables for People, Books, Authors, Cardholders, and CheckOutLog are preloaded with data.

The table schemas in the database do NOT need to match to the structure of your collections in your application. Just make sure all data is persisted.

As your application is running, keep all data persisted in memory. Therefore, as the program exits, this information should be saved in a method of your choice. Since you have attended the Data Access Using C# course, you should consider using the database your application is connected to. Some other options are serialization or text document(s) or XML.

The program should load this information automatically into the collections you created in your application as the program starts.

*As a stretch goal, support data source writes in the event your program crashes (for example, the user just checked out a book and the computer restarts because Windows Update ran. So, you would want to update your data source at the point the user completed the book checkout).*

### Additional Rules

1. The program can be written as a console application, a Windows Forms application, a WPF application, or a Web application.
2. Each of the collections of data (People, Books, and CheckoutLog) should be a generic collection type (e.g. List<T>) contained in a SEPARATE class. The three class types (Author, Librarian, and Cardholder) are to be contained in the same collection together. Remember that the data contained in all three collections must be saved to the disk somehow as the program exits and read in as the program starts. The program must be designed to hold a minimum of 1000 books, 1000 authors, 20 librarians, and 1000 checkout transactions.
3. Polymorphism should be implemented for the following actions for the Author, Librarian, and Cardholder classes to eliminate duplicate code:
4. Display of information as indicated under the Librarian menu options.
5. Exporting of data to disk when the program ends as indicated in the Data Storage section earlier in this document.
6. All three classes containing a collection in the application should allow for any code in the application to iterate through the class using a “for each” loop coding structure.
7. All collections should be kept sorted at all times as follows:
8. Books: Title, ISBN
9. People: Last name, first name
10. CheckoutLog: Check-out date, title
11. All expected errors are to be handled via exception handling. Whenever possible, the user should be notified of the error and provided instructions on how to correct the issue.
12. Code files should be laid out according to standard C# conventions (i.e. each class in its own source file).
13. The code should be well commented. If a beginning programmer can’t understand what is going on in your code, you need more comments!
14. All data fields in classes should be private with supporting properties. This includes collections contained in a separate class of their own.

### Notes

1. Remember to consider this as an example project to show potential employers. This means:
   1. There should be some sort of design that you can talk through.
   2. Your project should be modularized – use simple classes and structures. For example, you might want to follow the rule of thumb that “a class has a single job in the application”.
   3. Code should be simple and readable, and the flow should be easy to understand.
   4. Feel free to use this project as a vehicle to show more advanced concepts (e.g. interfaces, operator overloading, delegates, TPL (Task Parallel Library), lambda expressions, LINQ, etc.), but not at the expense of missing deadlines. **Meet the requirements of this project first, then embellish.** You can always iterate on it after the first version.

## Test Plan:

As I evaluate the program at the end of the project, I will go through the following steps:

* Overall structure and flow:
  + Verify that the basic display and functionality requirements in the above document were met, and that all required components are complete and functional.
  + Verify that the flow of the user interface is intuitive and makes sense to both users and librarians.
* Functionality (golden path)
  + Enter several new book records
  + Verify that the system can retrieve and display those records successfully:
    - Search for books by Title
    - Search for books by ISBN
    - Search for books by Author
    - Search for books by Subject
    - Search in a way that yields multiple results, and verify that all the appropriate results are displayed (search for a single string that appears in Author names as well as Titles)
    - Select one or more of the displayed results to verify that the details of the book are properly displayed
  + Check out and check in several books, verifying that the appropriate counts are calculated properly.
  + Log in as a librarian and verify that a proper login process takes place and that only proper ID/Password combinations are allowed access.
  + Search for books using the Librarian menu, verifying that appropriate results are displayed (see above for details).
  + Add and remove several books, making sure that – when a book already exists -- existing book records are modified (rather than new ones created). Also checking that – when a book doesn’t exist – a new book record is created.
  + Verify that author records are linked to and added (as appropriate) properly.
  + Exit the program and restart it, verifying that the data has not been lost or duplicated.
  + Remove one or more books from the system, and verify that they no longer show up in the searches and that this state is saved as the program exits.
  + Update existing books and verify through a search as well as exit and restart that the book data has been updated and persisted and not duplicated.
* I will evaluate the program based on unexpected scenarios:
  + Attempt to look up items that do not exist. An appropriate error message should be displayed (Depending on implementation, attempt to checkout a book that (a) doesn’t exist or (b) has no more copies available.)
  + Attempt to add books that already exist, and verify that the program recovers and a proper error message is displayed.
  + Attempt to log into the Librarian menu with invalid credentials to make sure I cannot access the menu.
  + Attempt to update a book that isn’t in the system.
  + Display lists for as indicated in the librarian options menu.
* Exit the program and look at the code, and verify that:
  + The classes are defined according to the above instructions (properties, access modifiers, class hierarchical structure for author, librarian, and cardholder, etc.)
  + Verify that the code is laid out properly in appropriate files and is well commented.
  + Verify that exception handling is implemented.
  + Verify that polymorphism is applied when displaying information for authors, librarians and cardholders.
  + Verify that polymorphism is applied when exporting data from the collection containing the authors, librarians, and cardholders, when the session ends.
  + Verify that the three major collections are generic and contained in a separate class of their own, and that the containing class needs to allow for the application using the class to use a “for each” loop structure against the collection within the class.
  + Verify that the three major collections can be sorted according to the requirements specified above.
  + Provide any feedback that would enhance the project in the next version.
  + Verify that the class containing the ID, first name, and last name for author, librarian, and cardholder cannot be directly instantiated.
  + Verify that there is code to block a cardholder from checking out a book if they have any overdue books.
  + No duplicate code when searching for a book either by a patron or librarian.
  + No duplicate for determining the number of books available.

If the program passes the above tests and evaluations, the student will pass the evaluation.