|  |  |
| --- | --- |
| Library-aisle-with-wooden-shelves-and-hundreds-of-books-539673956-hd | 響林社  CourseWORK 1  By Robert Iarinka | **CST2550**  Student ID M00862854 |

Project Introduction

In this project, I have created a system for librarians which includes the following functionalities:

* Add a member,
* Issue a book to a member,
* Display a list of members,
* Return a book from a member.

This software will be used exclusively by the Librarian. It is designed to simplify the librarian's workflow by allowing them to easily see which members can borrow books. The feature to display a list of current members also facilitates bug testing and fixing, as it provides a clear view of whether the members' vector is being updated properly when testing the 'add members' function or any other feature that may be implemented according to the provided UML diagram.

\*\*Basic Flow of a Use Case Interaction between a Librarian and

a Member Borrowing a Book:\*\*

1. The Librarian inquires if the user wishes to register. Upon affirmation, the Librarian registers the member's information, such as their name, address, and email, and then the system allocates an ID to them. This ID is used later for issuing and returning books.
2. The Librarian can issue a book to a member by selecting a member via their ID and allocating a book of the member's choice using the book's ID.
3. The Librarian has the option to display a list of all current members, along with the books they have borrowed and those that are still outstanding.
4. The Librarian can return the books borrowed by a member. This is possible because, when a book is issued, the program records the current time from the Librarian's system and sets the due date for three days later. Upon return, the system checks the current time again, and if it is not past the book's due date, the member can return the book without incurring a fine.

**UML Implementation:**

In the implemented system, I utilized the provided UML diagram for both the design and functionality, referring to how the different classes interact. The current setup of the design, as well as the UML, indicates that the Person class is a base class containing common attributes, such as the person's name. The Member and Librarian classes inherit from the Person class and have their unique attributes. The Book class does not inherit from the Person class; instead, it possesses attributes related to the books that members will borrow, such as the book's ID, name, etc. There are different relationships between the classes. For instance, the arrows with filled heads pointing from the Librarian and Member classes to the Person class indicate that Librarian and Member are subclasses; this is known as an "is-a" relationship. There is also a relationship between Member and Book, indicating that a member can have zero or many books loaned, while a book can be borrowed only once.

A screenshot of a computer

Description automatically generated

A diagram of a company

Description automatically generated

**Program Test**

The test cases are designed to simulate a user going through the processes of adding a member, adding a book, issuing the book to the member, returning the book, and calculating fines. These test cases ensure that the Librarian class methods function as expected when called in a sequence that reflects typical usage of the program. I utilized the Catch2 header file, which I downloaded from GitHub, to facilitate testing of my program.

1. Full Program Flow

This comprehensive test case examines the complete workflow of the library system. It ensures that the librarian can successfully add a member and a book to their respective lists, issue the book to the member, and then process its return. It also includes validation steps for each action, such as checking that the member list and book list are initially empty, confirming the addition of a member and a book, and verifying the successful issuance and return of the book. Finally, it simulates the display of borrowed books and the calculation of fines, followed by cleanup actions to clear the member and book lists.

2. Adding Multiple Members

This test case tests the librarian's ability to add multiple members to the system. It starts with an assertion that the member list is empty, then adds two members with distinct names, and verifies that the member list correctly reflects these additions. A final check confirms the name of the second member added, ensuring the addMember function's accuracy.

3. Issuing Book Not In Catalogue

In this scenario, the system's behavior is tested when attempting to issue a book that is not present in the catalogue. After setting up the environment and adding a member for testing, the librarian tries to issue.

a non-existent book, identified by an arbitrary book ID that is not in the library's catalogue. The test confirms that the member's borrowed book list remains empty, indicating that the system correctly prevents issuing books that do not exist in the catalogue.

4. Returning Book Not Borrowed

This test case addresses the situation where a member attempts to return a book they have not borrowed. The system setup involves adding a member to an initially empty member list. The librarian then processes the return of a book using a non-existent book ID. The test ensures that the member's borrowed book list is still empty after the attempted return, confirming that the system correctly handles such cases.

5. Calculating Fine for Late Returns

The last test case focuses on the fine calculation process for late book returns. After the librarian adds a member and a book, the system mocks the current date to be after the due date of the issued book. The librarian issues and then returns the book, which should trigger the fine calculation mechanism. This test is designed to verify that the system accurately assesses fines for late returns.

Function Mocking for Input Handling

To facilitate these tests, input functions are mocked to return predetermined values. The getStringInput function responds with appropriate test inputs based on the prompt's content, while the getIntInput function consistently returns a specific integer. These mocked functions are crucial for simulating user interactions and validating the system's response without actual user input.

**Software** **Design** **Explanation**

When designing the software, I paid real close attention to following the specifications of the given UML and the limitations, as some classes only took a certain parameter in, which meant the way I had to code the different functions with the classes had to be very specific and I had to ensure that the header files of each class weren’t different from the UML. To successfully create the classes and the required functionality I had to also create some extra functions and objects, such as the Date class which allows me to set the due dates and create a Date Object as well as help me get the current day through a get current day function which is used in the Librarian class to be able to return an issued book, as well as the difference in days function which calculates the difference between two days this is used in calculating the fine by subtracting the due date and the current date. To receive correct user input and provide user feedback upon inputting some text I created InputHandle.cpp and InputHandle.h to ensure user inputs are handled in correctly. I also had to make a glob.cpp to ensure the global vectors were defined and saved.

**The Approaches I tried to Implement.**

The approach I used -> Throughout my project I tried to follow the OOP approach with using object data structures because it made sense and was easier to implement than other approaches. Another approach I used was frequent testing of the code, as well as frequent version control through git and GitHub and clear understandable code and comments.

**How I tried to implement that in my development ->** The usage and implementation of those approaches varied as the usage OOP was in a way necessary as we had to use classes and objects to implement the needed functionality as well as frequent git commits was also necessary. I implemented frequent testing by making the make file and the needed Main cpp file early on so I can run and compile my program to test if the implemented code worked.

**Make File Usage**

The make file was used in this project because we are working with a lot of different files so compiling then using Cygwin or command Prompt one by one wouldn't be sensible idea, instead I created a make file that automatically compiles the cpp files and creates . o files. After the make file configuration is done and the make file is written working and there are no issues with the coding, it creates an exe file which I named library\_system so the file I run is library\_system.exe. This also works for the test file that works the same way except there is no executable I must run.

**Usage of Version Control**

Usage of version control was needed to ensue regular updates were pushed to my GitHub repository, so not only I have a save of the code so I can go back in case something breaks, or something gets deleted but also to keep track of where I am at and what I should do next if I get confused or don’t work on the program for some time. This was achieved through regular push and commits with commit messuages of what the commit is about.

A screenshot of a computer

Description automatically generated

**Conclusion**

To conclude upon everything, throughout the creation of the project I managed to create the library system with the required specifications given to me, I followed the given UML and implemented the required features and classes to make the program work as intend.   
The limitations my work have only been the test cases and the test.cpp could have had better more effective tests and the Dates and Fines system and logic be a bit different and could have made the program more effective.  
In the future I would approach this project differently, the main part is I would have allowed more time for the project as I started quite late and had to rush things and couldn't make them exactly how I wanted, the way it worked and the logic. I would have focused more on making much proper test cases that test section by section more in-depth.