Library Management System

Shanell Pereira - M00912746

Introduction

In this project, I have created a library management system, which allows librarians to interact with a system to perform various tasks to maintain the system. The tasks include adding new members, issuing books to members, handling returned books, calculating fines for overdue returns and tracking books borrowed by members.

This presentation is structured to provide an overview, starting with the design phase where UML diagrams have been utilised. These diagrams serve as graphical tools for representing the structure and functionality of software systems, providing a clear visual framework for understanding the design. Next, the implementation phase, where I will describe how I translated the design into the software. Explaining the purpose of the Makefile and the significance of using version control. I will provide a glimpse into my GitHub repository to showcase the commit history. Following this, I'll outline the testing approach, explaining the approach used and detailing various test cases. Then the software demonstration phase will showcase the program and explain the implementation in a video format.

Design

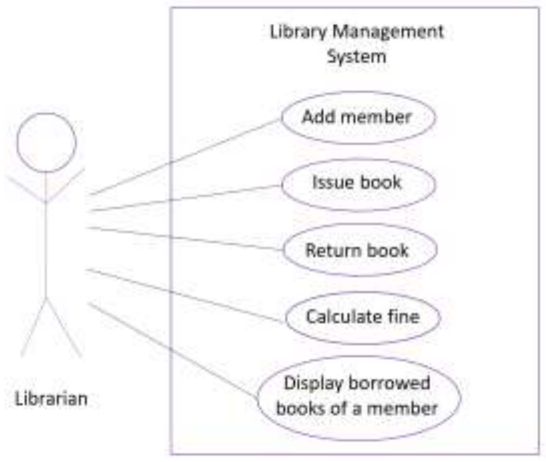


Figure 1.01 – UML Use Case Diagram

The diagram is a use case diagram for a Library Management System with one actor, "Librarian", who can perform five actions: add member, issue book, return book, calculate fine and display borrowed books of a member.

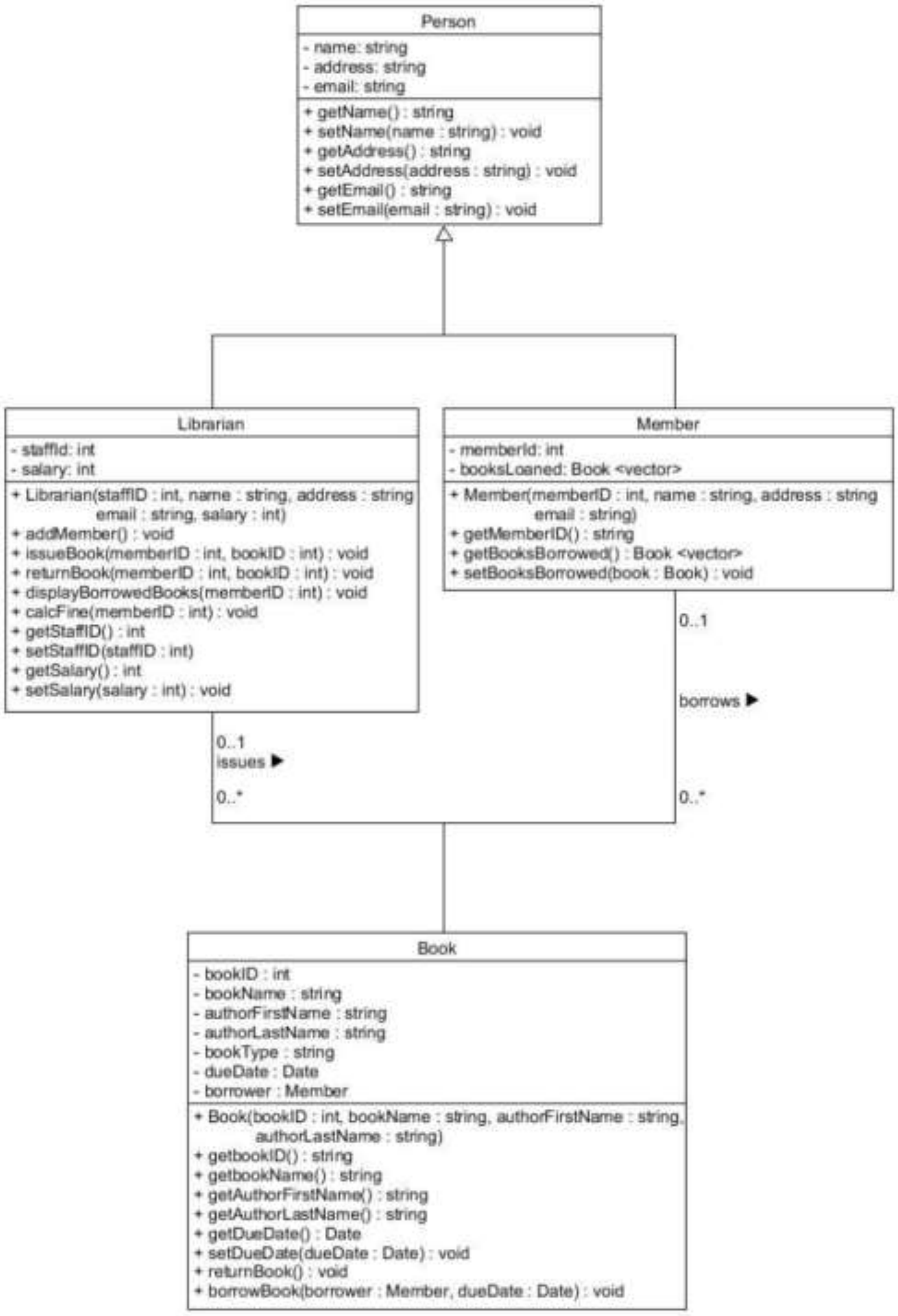

Person
- name: string
- address: string
- email: string
+ getName() : string
+ setName(name : string) : void
+ getAddress() : string
+ setAddress(address : string) : void
+ getEmail() : string
+ setEmail(email : string) : void
Librarian
- staffId: int
- salary: int
+ Librarian(staffID : int, name : string, address : string email : string, salary : int)
+ addMember() : void
+ issueBook(memberID : int, bookID : int) : void
+ returnBook(memberID : int, bookID : int) : void
+ displayBorrowedBooks(memberID : int) : void
+ calcFine(memberID : int) : void
+ getStaffID() : int
+ setStaffID(staffID : int)
+ getSalary() : int
+ setSalary(salary : int) : void
Member
- memberId: int
- booksLoaned: Book <vector>
+ Member(memberID : int, name : string, address : string email : string)
+ getMemberID() : string
+ getBooksBorrowed() : Book <vector>
+ setBooksBorrowed(book : Book) : void
0..1
borrows ▶
0..*
0..1
issues ▶
0..*
Book
- bookID : int
- bookName : string
- authorFirstName : string
- authorLastName : string
- bookType : string
- dueDate : Date
- borrower : Member
+ Book(bookID : int, bookName : string, authorFirstName : string, authorLastName : string)
+ getbookID() : string
+ getbookName() : string
+ getAuthorFirstName() : string
+ getAuthorLastName() : string
+ getDueDate() : Date
+ setDueDate(dueDate : Date) : void
+ returnBook() : void
+ borrowBook(borrower : Member, dueDate : Date) : void

Figure 1.02 – UML Class Diagram

This Class Diagram are blueprints of the system. It describes the attributes and methods. It displays the relationships between objects. The Person class is the parent class to both Librarian and Member as they inherit from Person. This means they both share the common attributes and methods of Person, but they also have additional function specific to their classes. There are also relationships of associations. This is where other classes interact with another. As the Librarian class is associated with Book objects because they can issue books. The Member class is associated with Book objects because they can borrow books.

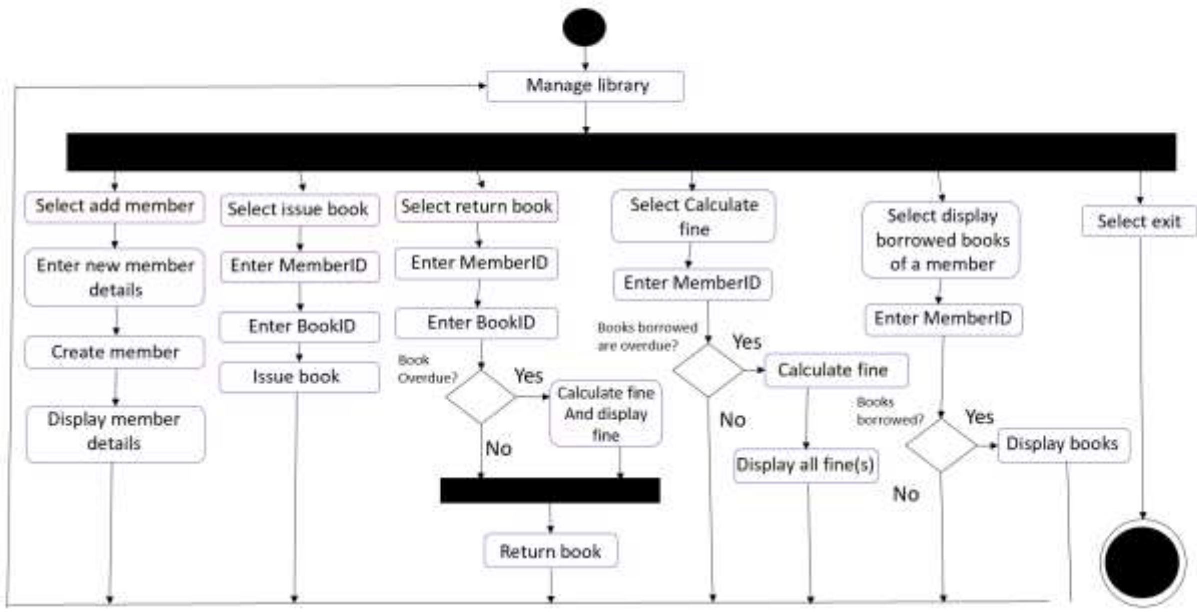


Figure 1.03 – UML Activity Diagram

The Activity Diagram is used to explain the flow of a system. The process begins by allowing the operator to manage the system by selecting activities.

By selecting to add a member, the operator is prompted to enter in details of the new member. Those details are used to create a new member. The new member's details are then displayed to the operator.

If the operator chooses to issue a book, they will be asked for the ID of the member to issue to, the ID of the book to be issued and then the book will be issued.

Upon choosing to return a book, the ID of the member will be required as well as the ID of the book. First the system will check if the book is overdue, if it is it will calculate a fine and then the book will be returned, otherwise the book will be returned with no problem.

The operator can also calculate the fine of a member. The ID of the member will be required. It will check when the members due date was and calculate the fine by adding the fine rate. Once the total fine has been calculated, it will be displayed.

By selecting to display the borrowed books of a member. The operator enters the ID of the member. Then the system checks if the member has borrowed any books. If yes, the books will be displayed. If no, the process ends.

Lastly, the operator has the option to exit, which will end the program.

Implementation

Following the class diagram, we can start to implement some code. The Person class served as a base with attributes like name, address, and email, with corresponding accessor and mutator methods. Librarian and Member are subclasses of Person, they would inherit all the attributes and methods of Person, but will also incorporate unique attributes and methods of their own. The book class will also be implemented with the blueprint provided by the class diagram regarding its attributes and methods.

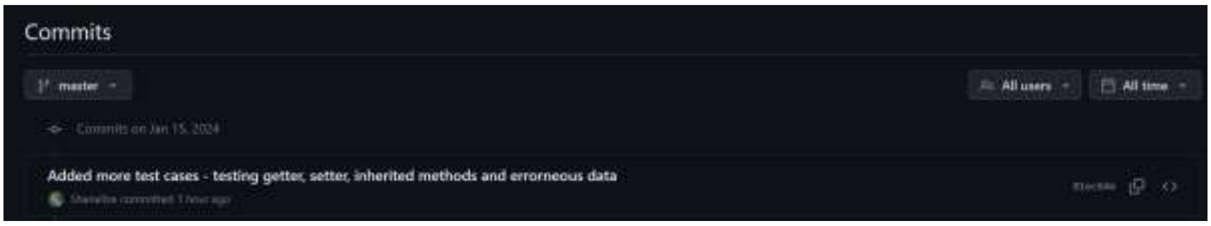
While examining the use case diagram, it is used to describe the functionality of the system from a user's perspective. It showcases how the librarian should interact with the system.

The activity diagram is an essential tool for understanding and communicating the complex processes within our library management system. It allows us to see the process of each action and how it should run.

A Makefile is a special file used for automating the compilation and building process. It specifies how to compile and link the program, often with a set of rules and dependencies. This ensures that only the necessary parts of the project are recompiled when changes are made, saving time and resources.

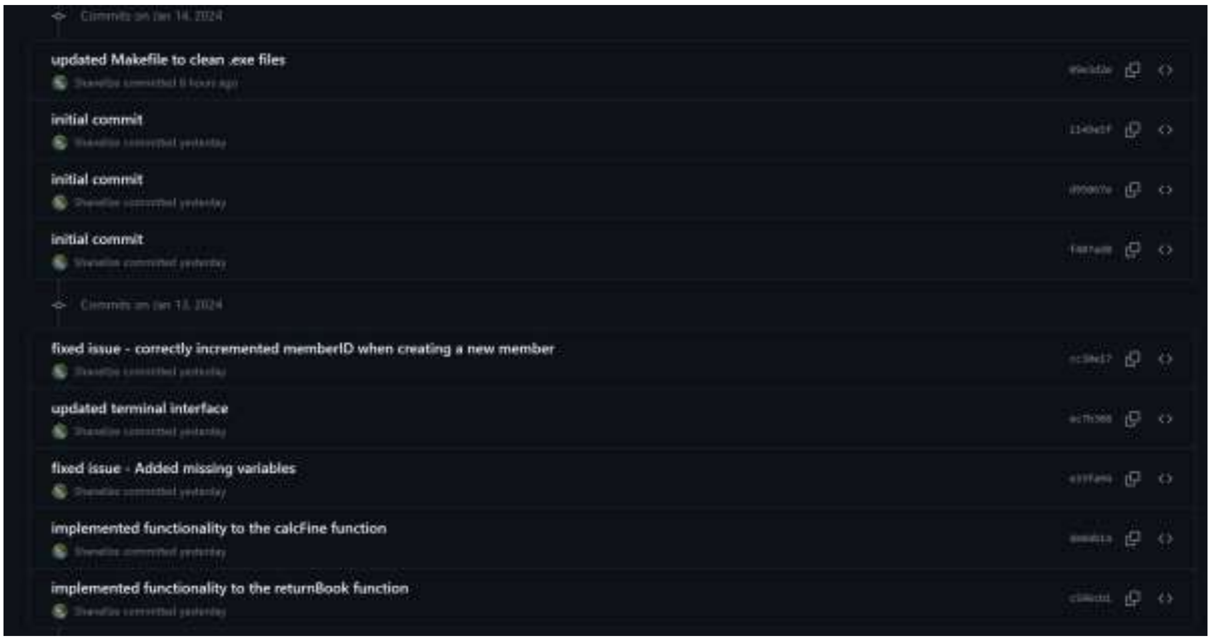
Version control is used for tracking and managing changes to the software code. This significantly improves the overall quality and maintainability of the code. If a mistake is made It also offers the ability to revert to previous versions of the code. By maintaining a history of all changes, version tracking enables transparency into the development process.

This is the commit history of the project:



Commits on Jan 15, 2024

Added more test cases - testing getter, setter, inherited methods and errorneous data
[illegible] committed 1 hour ago



Commits on Jan 14, 2024

updated Makefile to clean .exe files
[illegible] committed 8 hours ago

initial commit
[illegible] committed yesterday

initial commit
[illegible] committed yesterday

initial commit
[illegible] committed yesterday

Commits on Jan 13, 2024

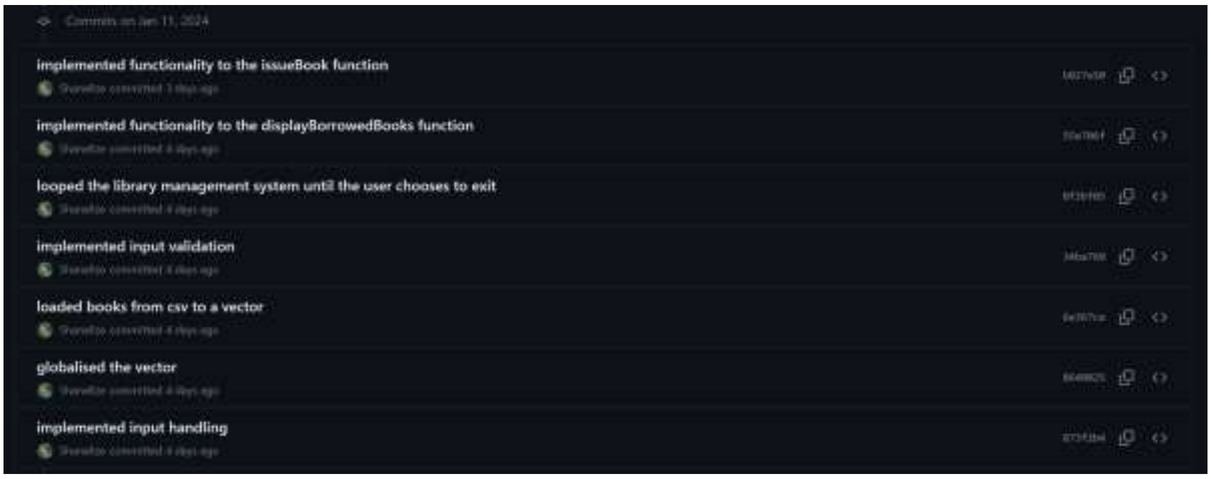
fixed issue - correctly incremented memberID when creating a new member
[illegible] committed yesterday

updated terminal interface
[illegible] committed yesterday

fixed issue - Added missing variables
[illegible] committed yesterday

implemented functionality to the calcFine function
[illegible] committed yesterday

implemented functionality to the returnBook function
[illegible] committed yesterday



Commits on Jan 11, 2024

implemented functionality to the issueBook function
[illegible] committed 3 days ago

implemented functionality to the displayBorrowedBooks function
[illegible] committed 4 days ago

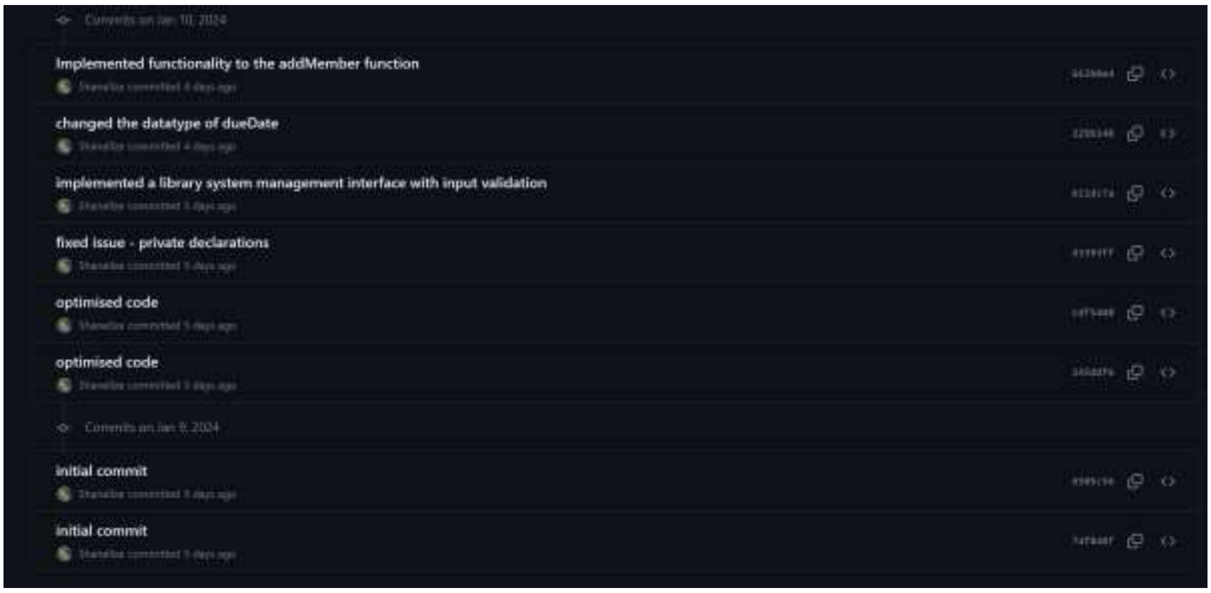
looped the library management system until the user chooses to exit
[illegible] committed 4 days ago

implemented input validation
[illegible] committed 4 days ago

loaded books from csv to a vector
[illegible] committed 4 days ago

globalised the vector
[illegible] committed 4 days ago

implemented input handling
[illegible] committed 4 days ago



Commits on Jan 10, 2024

Implemented functionality to the addMember function
[illegible] committed 4 days ago

changed the datatype of dueDate
[illegible] committed 4 days ago

implemented a library system management interface with input validation
[illegible] committed 5 days ago

fixed issue - private declarations
[illegible] committed 5 days ago

optimised code
[illegible] committed 5 days ago

optimised code
[illegible] committed 5 days ago

Commits on Jan 9, 2024

initial commit
[illegible] committed 5 days ago

initial commit
[illegible] committed 5 days ago

Testing approach

For my library management system, I adopted a unit testing approach, focusing on ensuring that the classes are working correctly. This method allows each function of a class to be tested before integrating it to the main system.

I created test cases for each class, testing each function. This included testing both the correct functionality and the handling of erroneous data. By doing so, I ensured that each unit of the system behaved as expected under various scenarios.

Test case details:

- Person class: Test were designed to test the getter and setter methods for attributes like name, address, and email. This includes both verifying correct data retrieval and updating attributes, followed by checking erroneous data to ensure robustness.
- Librarian class: The tests for this class focused on specific librarian attributes such as staff ID and salary, along with inherited attributes from the Person class. Both the inherited getter and setter methods and those unique to the Librarian class were tested. As well as data that would be erroneous to evaluate how the functions will handle them.

- Member class: Similarly to the Librarian class, Member inherits from Person. The tests developed for the Member class were targeted at evaluating both unique and inherited functionalities. This includes the getter and setter methods for the member-specific attribute, such as member ID, along with the inherited attributes from the Person class like name, address, and email. These tests involved verifying the correct functioning of these methods in terms of accurate data retrieval and updating. In addition to these, the tests also checked the class's handling of erroneous data.
- Book class: Testing in the Book class were aimed to evaluate the functionality of the getter methods for attributes like dueDate, bookID, BookName, BookAuthorFirstName and BookAuthorlastName. It also included testing the setter method for the duedate attribute. This not only involved retrieving the correct data and updating attributes, but also the class's capability to handle erroneous data effectively, confirming its overall robustness.

Conclusion

To conclude, I have created a library management system designed to assist librarians in managing multiple tasks. The librarian can add members, issue books, return books, calculate fines and display borrowed books of a member. In this project, I started by creating a plan. Planning the use case and activity diagram to provide a clear structure and functionality representation. The implementation involves developing classes and methods based on the diagrams, using a Makefile to automate the compilation of files, and version control to manage and track the updates in the code. I have also included detailed test cases for evaluating the classes and provided a software demonstration in video format to showcase the functionality of the code.

One limitation to my work was the difficulty in implementing Catch2. While I was able to use it to test my classes, I couldn't get it to run at the same time as my main code. This led to me having to modify the int main() by removing it from the code to be able to test, which is not practical for a fully operational application. Due to the time constraint, I was not able to tackle this issue.

In the future, I would try to use an alternative testing approach and devote more time to fully understanding the testing approach.