

Student Name: Daniah Alhashim

Student ID Number: M00911465

Software Engineering Management and Development

Course Work 1

Library Management System Report

This report will discuss the library management system that I've created. The overview of this report will outline key sections, including design, implementation, testing, software demonstration, and conclusion. Also, I will briefly explain the significance of the UML diagrams. Moving on to implementation, I'll discuss how I turned the design into functional software, and the purpose of the Makefile. Explaining my testing approach, I'll cover the statement of approach, its application, and the test cases without delving into the code. During the software demonstration, I'll emphasize understanding the implementation beyond program execution. Finally, in the conclusion, I'll provide a concise summary of the project, highlight any limitations, and discuss how I'd approach a similar project in the future while addressing identified limitations.

The objective for this coursework is to create a library management system which is designed to assist librarians in organizing their book inventory and member records efficiently. This will allow librarians to use the system to add new members, lend books with due dates, accept book returns, and calculate fines for overdue returns. It simplifies the administrative tasks of a librarian, making it easier to manage the library's resources and ensure a smooth borrowing and returning process for members.

UML Diagram

The UML diagram for the library system is like a family tree for the different parts of the program. At the top, there's the "Library" class which is considered as the head of the family. This class manages everything in the library, including books and members. Now, you can think of the "Book" and "Member" classes as children of the "Library" class. They have their unique information but are part of the library family. The "Book" class holds details about each book, like its name and author, while the "Member" class keeps track of members with their names and IDs. These classes work together, creating a well-organized family where the library manages books and members smoothly.

Adding on, I will be explaining how they interact. It's like a librarian handing books and members. The "Library" class can add new members or books and issue them to members. So, if a new member joins, the "Library" knows and updates its family. When a book is borrowed, the "Library" marks it as borrowed in the "Book" family, keeping things in order. The "Member" class can also return books, and the "Library" updates its records accordingly. It's a teamwork scenario, therefore everyone has a role, and they communicate to make sure the library runs smoothly.

In conclusion, the UML diagram is like a family portrait, showing the relationships between the "Library," "Book," and "Member" classes. They work together, making sure the library functions like a well-coordinated family where each member has its role.

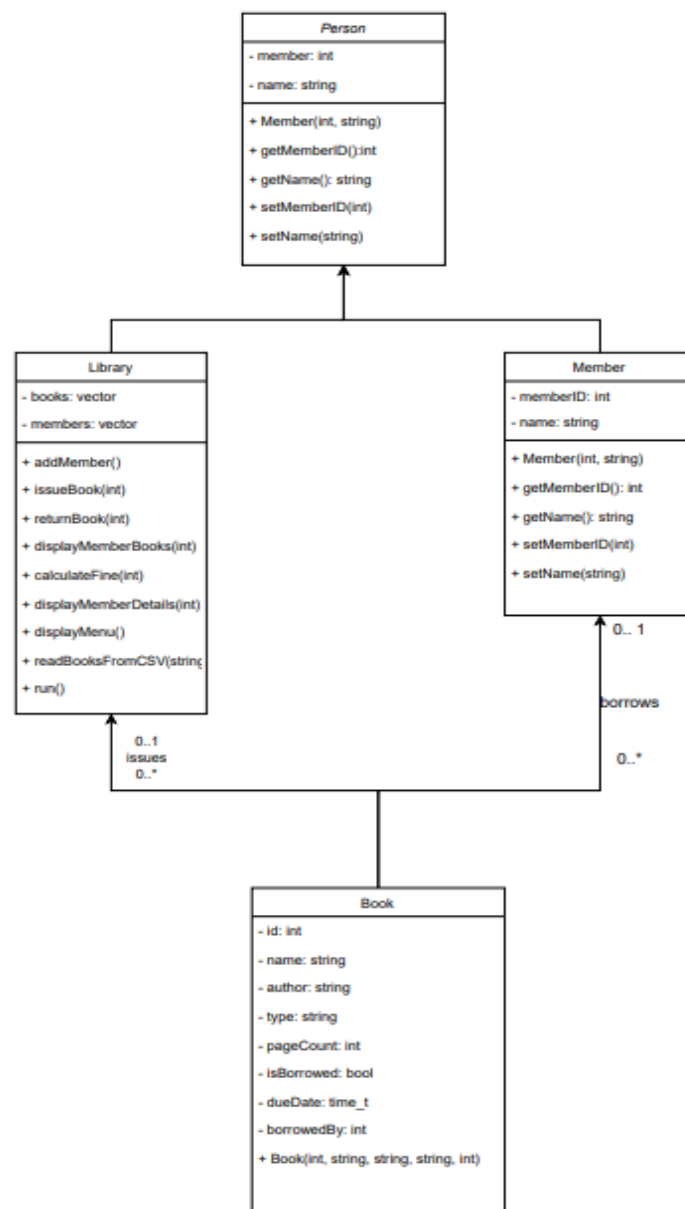


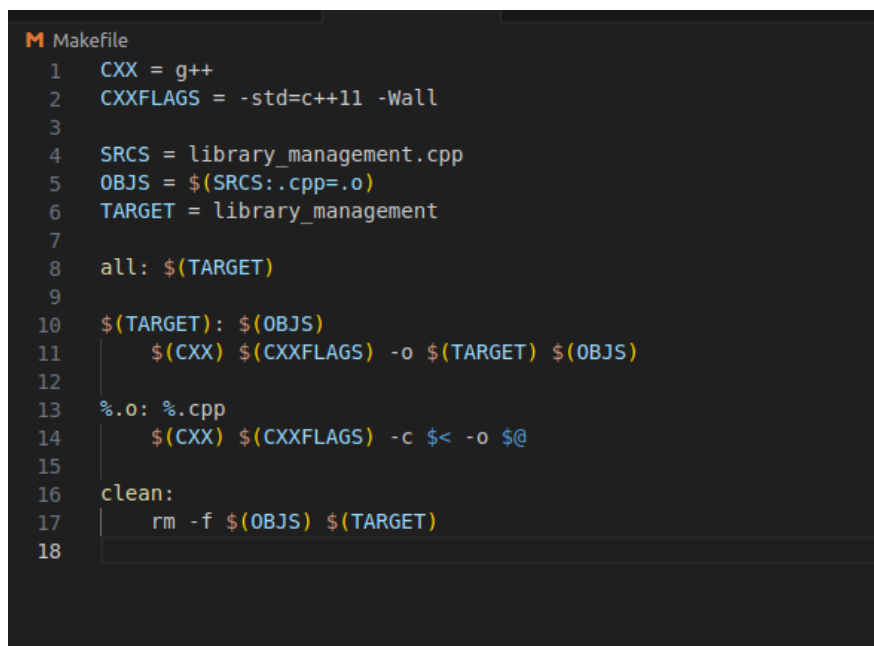
Figure 1. UML Diagram

Project Implementation – Approach

Firstly, I used the family structure from the UML diagram – the “Library,” “Book,” and “Member” classes. The “Library” became the main manager, and I made sure it could do things like adding new members, lending books, and keeping track of everything. Then, I focused on the “Book” and “Member” classes, giving them the ability to store information and connect with the “Library” when needed.

Implementation – Makefile

The Makefile is extremely significant and is a helpful guide. It tells the computer how to put together all the pieces of the library program so that it can run smoothly. When we type “make” in the computer, the Makefile follows its instructions, compiles our code, and creates the final program we can use.



```
M Makefile
1  CXX = g++
2  CXXFLAGS = -std=c++11 -Wall
3
4  SRCS = library_management.cpp
5  OBJS = $(SRCS:.cpp=.o)
6  TARGET = library_management
7
8  all: $(TARGET)
9
10 $(TARGET): $(OBJS)
11     $(CXX) $(CXXFLAGS) -o $(TARGET) $(OBJS)
12
13 %.o: %.cpp
14     $(CXX) $(CXXFLAGS) -c $< -o $@
15
16 clean:
17     rm -f $(OBJS) $(TARGET)
18
```

Figure 2. Makefile

Testing Approach

For testing my library project, I followed a careful approach. First, I made a plan, deciding what I wanted to test – like making sure new members could join and borrow books correctly. Then, went through each part of the program to see if it behaved the way I expected. For instance, I checked if the library fined members for returning books late. I

wrote down these checks as my test cases, sort of like a to-do list, making sure I didn't miss anything.

Conclusion

To conclude this report, I've created a digital library system that helps librarians manage books and members efficiently. The "Library," "Book," and "Member" classes work together like a well-organized team, allowing librarians to add new members, lend and return books, and calculate fines for overdue returns. The program utilizes a Makefile for easy compilation and version control for tracking changes, ensuring a smooth development process. However, like any project, there are some limitations. The system currently lacks a graphical user interface (GUI), making it more suitable for command-line interactions. In the future, to enhance user experience, I would consider incorporating a simple GUI for a more intuitive interface. Additionally, error handling could be strengthened to provide clearer messages in case of incorrect inputs. By refining these aspects, the system could become even more user-friendly and robust for librarians to efficiently manage their libraries.