**CST2550 COURSEWORK -1**

**ANAKHA JAYAKUMAR**

**M00895343**

**INTRODUCTION**

The project is on a library management system and the system is designed for librarians without requiring any prior training but can operate effectively In this system librarians will be able to manage the members and the collection of books of different genres available in the library which is provided in the CSV file and The file contains details of all books that are available for members.

The books are available to borrow and return within three days. the system allows them to issue books and also asses the returns. The system also automatically calculates the fines for overdue books based on a rate of 1 pound per day.

**DISCRIPTION OF THE PROVIDED UML**

**BOOK CLASS**

**Attributes:**

**bookID:** An integer representing the unique identifier of the book.

**bookName:** A string representing the name of the book.

**pageCount:** An integer representing the number of pages in the book.

**authorFirstName:** A string representing the first name of the author.

**authorLastName**: A string representing the last name of the author.

**bookType:** A string representing the type or genre of the book.

**Methods:**

**getBookID():** Returns the book's ID.

**getBookName():** Returns the book's name.

**getPageCount():** Returns the number of pages in the book.

**getAuthorFirstName**(): Returns the first name of the author.

**getAuthorLastName():** Returns the last name of the author.

**getBookType():** Returns the type or genre of the book.

**MEMBER CLASS**

**Attributes:**

**memberID:** An integer representing the unique identifier of the member.

**firstName:** A string representing the first name of the member.

**lastName:** A string representing the last name of the member.

**borrowedBook**s: A vector of integers representing the IDs of books borrowed by the member.

**Methods:**

**getMemberID():** Returns the member's ID.

**getFirstName():** Returns the first name of the member.

**getLastName():** Returns the last name of the member.

**getBorrowedBooks():** Returns the vector of book IDs borrowed by the member.

**borrowBook(bookID):** Adds a book ID to the list of borrowed books.

**returnBook(bookID):** Removes a book ID from the list of borrowed books.

**LIBRARY CLASS**

**Attributes:**

**books:** A vector of Book objects representing the books in the library.

**members**: A vector of Member objects representing the members of the library.

**Methods:**

**addBook(book):** Adds a Book object to the library's collection.

**addMember(member):** Adds a Member object to the library's collection.

**issueBook(bookID, memberID):** Issues a book to a member.

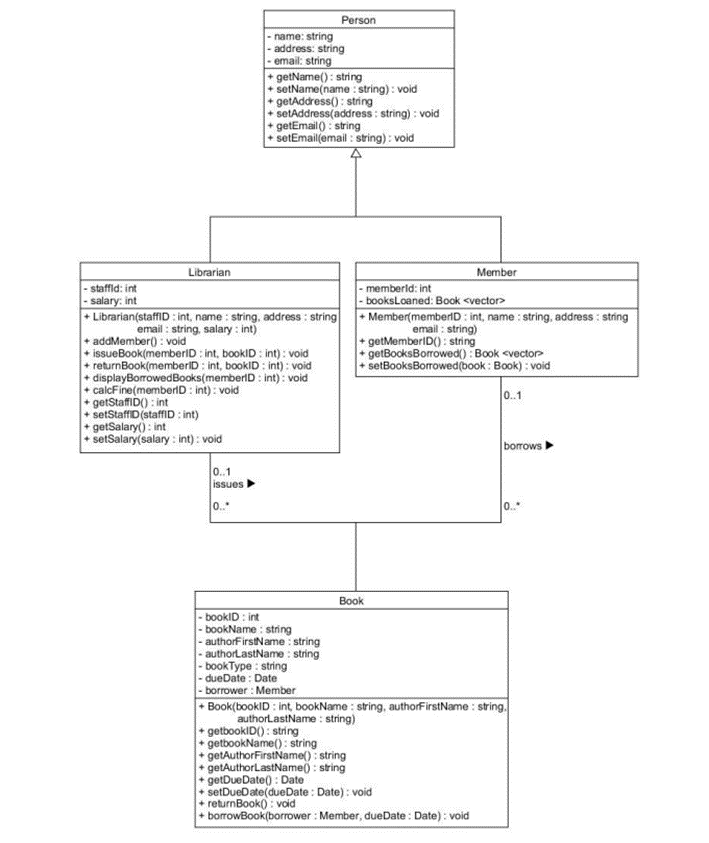
**returnBook(bookID, memberID):** Returns a book from a member.

**displayBorrowedBooks(memberID):** Displays the books borrowed by a member.

**calculateFine(memberID):** Calculates and displays the fine for overdue books for a member.

**displayBooks():** Displays information about all books in the library.

**displayMembers():** Displays information about all members in the library.



**IMPLEMENTATION**

UML is used to define requirements, design solutions, and track progress, making them a valuable tool in software development. the UML diagram provided for this library management system has four classes Person, Librarian, Member, Book.

The attributes of each class are specified in the UML diagram and used in the code example the member class in the diagram had attributes like member ID, name, address, and email these are in the MEMBER class in the code. In the same way, all other classes are written in the code.

UML diagram illustrates how the data flows between different components and how certain functionalities are intended to work. the functionalities like adding members, issuing and returning books, and displaying the information were designed with the librarian in mind, following the principles outlined in the UML diagram.

In summary, the UML diagram played a pivotal role in guiding the translation of the systems design into working software.

A make file is typically used to automate the building process of the project by a set of rules. It’s a text file that contains a set of rules for compiling and linking the code related to the project.

For the library system the makefile of the C++ project. it includes all the rules and settings that help in compilation and linking.

**CC** = G++ ( compiler)

**CXXFLAGS**: Compiler flags for C++ compilation. It includes -std=c++11 to use the C++11 standard and -Wall to enable most warning messages.

**LDFLAGS**: Linker flags. It can be customized if needed.

**APPNAME**: Specifies the name of the executable.

**EXT**: Extension of source files (.cpp in this case).

**SRCDIR**: Directory where source files are located.

**OBJDIR**: Directory where object files will be stored.

**SRC**: A list of all source files with the specified extension in the source directory.

**OBJ**: Corresponding object files that will be generated in the object directory.

**DEP**: Dependency files generated during the compilation process.

**all**: The default target. It depends on the $(APPNAME) target.

$(APPNAME): The target to build the executable. It depends on the object files ($(OBJ)). The $@ variable represents the target name, and $^ represents the dependencies.

**Usage:**

make: Builds the executable.

make clean: Cleans the project.

make cleandep: Cleans only dependency files.

make cleanw: Additional cleaning for Windows.

make cleandepw: Cleans only dependency files for Windows

UseCase and Activity diagram

A diagram of a library management system

Description automatically generated

A diagram of a flowchart

Description automatically generated

**Summary**

This coding of the project is implemented in C++ This program is for a librarian to track the details of their books and members. the data source is provided as CSV files. All books have an id, name, author, type, and page count. The program will only be used by the librarian at the library (not by customers/members) and will not

take payment details.

**Limitations of work**

The code reads information of books from a CSV file.

Minimal validation of user

**Avoid limitations**

Implement thorough input validation at all user interaction points. Validate user input for adding members, issuing/returning books, and any other user prompts.

Make file paths, such as the one for the CSV file, configurable. Allow users to specify file paths or provide the file path as a command-line argument to enhance flexibility.