CST2550 SOFTWARE ENGINEERING MANAGEMENT AND DEVELOPMENT COURSEWORK 1

Anikhil Patel M00861390

INTRODUCTION

My project is made for librarians to simply add members, issue and return books from these members, and see which books are borrowed by which member

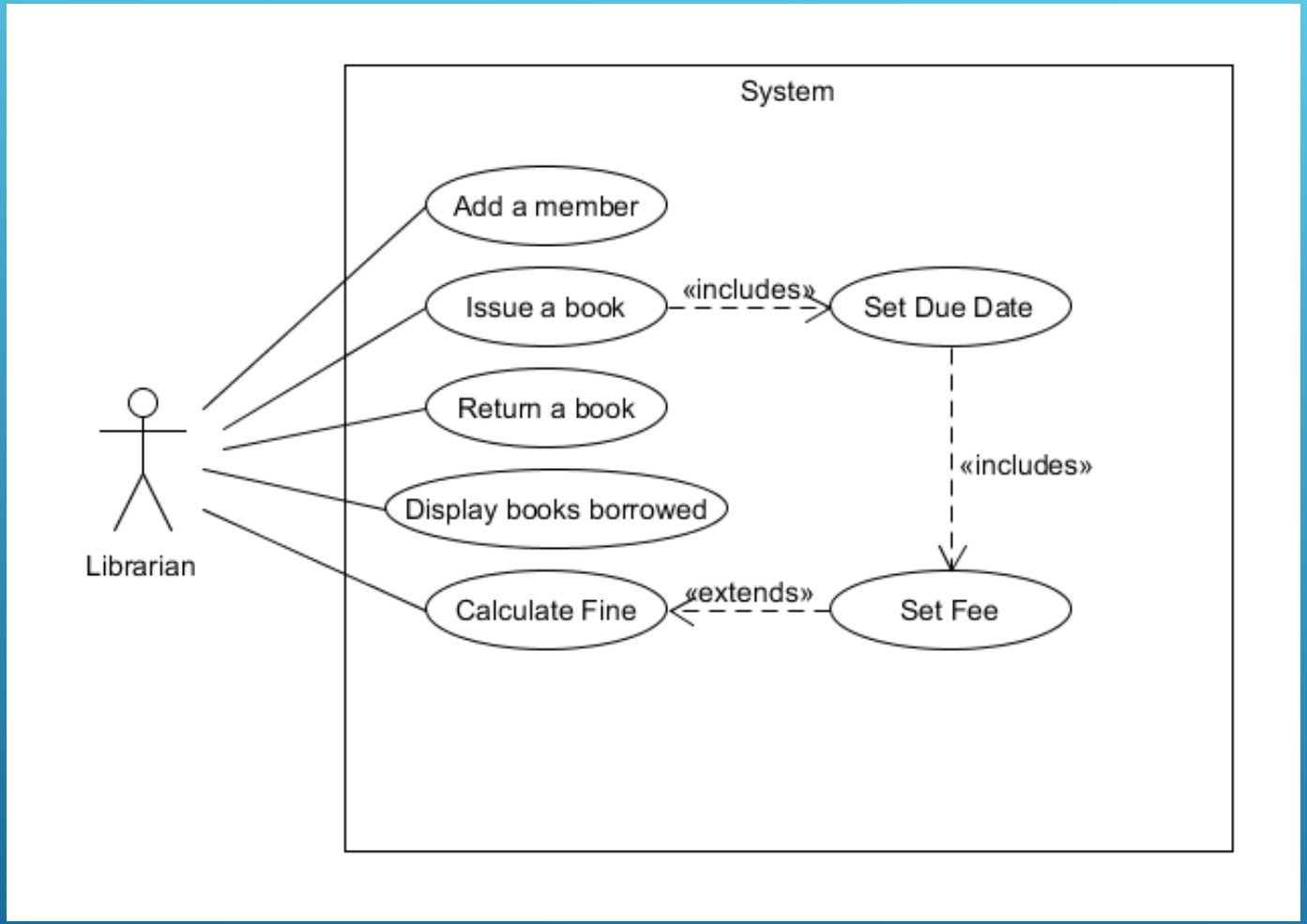
The librarian logs into the system inputting their ID, name, address, email and salary. Then the librarian has the option to add a member issue and return books, display the books borrowed by a member and calculate the fine depending on the due date of the book borrowed.

The program has a simple and easy to understand UI for the librarian to be able to carry out these following tasks as easy as possible

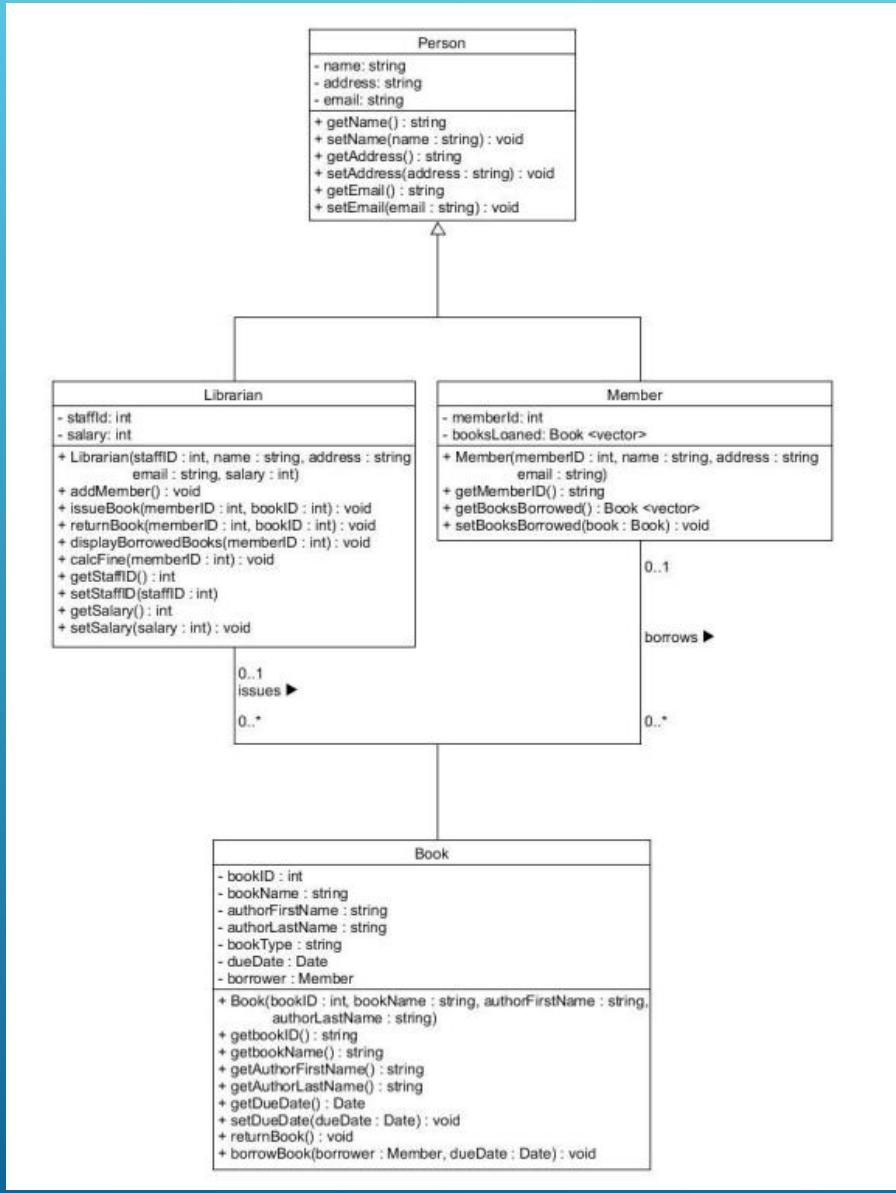
Overview:

- Design – Use cases
- Approach – Implementation and Makefile
- Test cases
- Conclusion – Summary and Limitations

USE CASE DIAGRAM



UML CLASS DIAGRAM



USE CASE SPECIFICATIONS

Use case: Add a Member

ID: 1

Brief Description:
Add a member to the program

Primary actors: Librarian

Secondary actors: None

Preconditions:
Librarian is Logged in to the program

Main flow:
1. Librarian enters Member Name
2. Librarian enters Member Address
3. Librarian enters Member email
4. System creates and stores Member details

Postconditions:
Member is successfully added

Alternate flows:
None

Use case: Issue a book

ID: 2

Brief Description:
Book is issued to the Member

Primary actors: Librarian

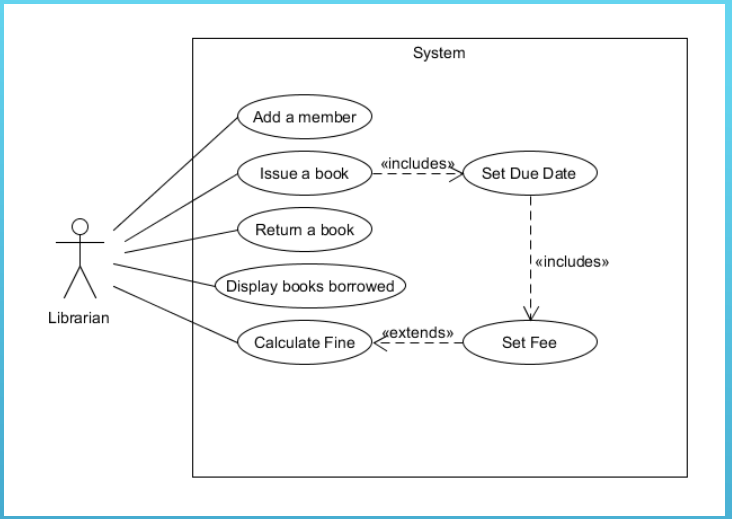
Secondary actors: None

Preconditions: Librarian is logged in to the program and a member has been created

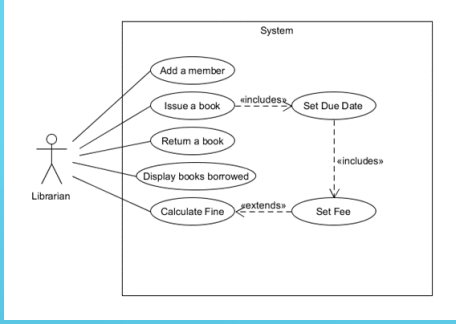
Main flow:
1. Librarian enters Member ID
2. Librarian enters Book ID
3. System issues the Book with the Book ID entered to the Member ID

Postconditions:
Book is issued to the corresponding Member

Alternate flows:
None



USE CASES SPECIFICATIONS CONTINUED



Use case: Return a book

ID: 3

Brief Description: Return an issued book from a member

Primary actors: Librarian

Secondary actors: None

Preconditions: Librarian is logged in, a member is created, and a book has been issued to the Member

Main flow:
1. Librarian enters Member ID
2. Librarian enters Book ID of book to be returned
3. System returns the book borrowed by the Member

Postconditions:
Book is returned and can be issued again

Alternate flows:
None

Use case: Display books borrowed

ID: 4

Brief Description:
Show all the books borrowed by a Member

Primary actors: Librarian

Secondary actors: None

Preconditions: Librarian is logged in, a Member has been created and been issued a book

Main flow:
1. Librarian enters Member ID
2. System checks and displays all books borrowed by the given Member ID

Postconditions:
Books borrowed by the Member ID given are displayed to the Librarian

Alternate flows:
None

Use case: Calculate Fine

ID: 5

Brief Description:
Calculate the fine depending on the due date of the borrowed book

Primary actors: Librarian

Secondary actors: None

Preconditions: Librarian is logged in, a Member is added, and the Member has borrowed a book, the book is overdue by 1 day

Main flow:
1. Librarian enters the Member ID
2. System checks according to the date the book was issued if there is a fine
3. System issues a fine for the number of days late the book is £1 per late day

Postconditions:
Displays the fine the Member is dues depending on how many days past the due date is

Alternate flows:
None

APPROACH AND IMPLEMENTATION

Given the UML Class Diagram, I used this to help implement the design into a working software

I declared the classes and the functions that would be needed in separate C++ header (.h) files

The .cpp files, corresponding to the .h files, contained the implementation of these functions and the uses of these declared classes

It helped to develop an understandable project and I made use of actual time within C++ using <ctime>

APPROACH AND IMPLEMENTATION

Makefile:

The Makefile was used to construct and compile all the .cpp files and .h files together into an executable file

The compiler was set to g++ and the compiler flags –g -Wall -Wextra –Wpedantic for enabling additional warning messages

The Makefile was used to make an executable file to run the program and the executable file relied on several object files

For each source file Librarian.cpp, Member.cpp, Book.cpp, Date.cpp, main.cpp, there is a compilation rule that specifies how to compile it into an object file

The Makefile also defines what make clean does in this case it removes the .o (objects created for the executable) and the executable itself

The ‘make’ command itself compiles and links all the .cpp files together tø get an executable program

TESTING APPROACH

Unit testing:

The approach I used to test the system is unit testing as I used it to test classes and functions in the program

I conducted unit tests for individual classes such as Librarian, Member, Book, and Date, this is to make sure that they worked correctly

Details of Test Cases:

Test Case: Librarian Class:

Section: Checking if a member can be added

Section: Issue a valid book to the member

Section: Return the valid book from the member

Test Case: Member Class:

Section: Returning a book from the member

Test Case: Book Class:

Section: Borrowing the defined book

Section: Returning the defined book

Test Case: Date Class:

Section: Printing the date that is defined

TESTING APPROACH CONTINUED

Test Case: Add Member Function:
Section: Adding a member and checking to see if the name was added
Section: Add multiple members and check if the size of the members vector is 2

Test Case: Issue book Function:
Section: Issuing a valid book to a member
Section: Issuing an already borrowed book
Section: Issuing a book with invalid member ID
Section: Issuing a book with invalid book ID

Test Case: Return book Function:
Section: Return a valid book
Section: Returning an already returned book

Test Case: Display Borrowed Books Function:
Section: Displaying borrowed books for a valid member
Section: Displaying borrowed books for an invalid member

Test Case: Calculate Fine Function:
Section: Calculating fine for a member with overdue books
Section: Calculating fine for a member with no overdue books
Section: Calculating fine for an invalid member

TESTING APPROACH CONTINUED

Catch2 Tests

```
===============================================================================
test cases:  9 |  7 passed | 2 failed
assertions: 30 | 28 passed | 2 failed
```

2 Tests will always fail this as they are dependent on the date:

```
system_tests.cpp:164: FAILED:
  REQUIRE( oss.str() == "Books borrowed by Dave (ID: 1):\nBook ID: 1\nBook Name:
 The Great Gatsby\nPage Count: 208\nAuthor: F. Scott Fitzgerald\nType: Fiction\n
Due Date: 13/01/2024\n\n\n" )
with expansion:
  "Books borrowed by Dave (ID: 1):
  Book ID: 1
  Book Name: The Great Gatsby
  Page Count: 208
  Author: F. Scott Fitzgerald
  Type: Fiction
  Due Date: 16/01/2024


  "
  ==
  "Books borrowed by Dave (ID: 1):
  Book ID: 1
  Book Name: The Great Gatsby
  Page Count: 208
  Author: F. Scott Fitzgerald
  Type: Fiction
  Due Date: 13/01/2024


  "
```

```
system_tests.cpp:188: FAILED:
  REQUIRE( oss.str() == "Fine for Member ID 1: £9\n\n" )
with expansion:
  "Fine for Member ID 1: £12

"
  ==
  "Fine for Member ID 1: £9

"
```

If I change the date to the correct date they do pass successfully

CONCLUSION

The project successfully works for the librarian to add members to issue books, see the books that the members are borrowing and return the borrowed books. The librarian is also successfully able to see the fine due for overdue books, however the only way I managed to test this is to manually change the date on my system to a few days into the future while the program is running.

Limitations:

There were many limitations to this project, the main limitation being time, if more time were allocated, perhaps the librarian login details would have been saved in a separate file for ease of use, same for the member details, the details could have been saved elsewhere so that its easier instead of adding a member each time the program is run

I also had issues with calculating the fine, the function worked, but I had to change the date on my computer to check while the program was still running

In future projects, I would approach them in a similar way, designing and creating UML class diagram to start with and then implement them

I would change the testing approach and include more testing approaches to ensure the project is tested more, but as time was a limitation on this project, I could only test it with unit testing