

CST2555 COURSEWORK 1

Implementation of a library management system

Done by: Teon
Morgan

Table of Contents

Introduction	2
Design.....	3
Implementation	6
Testing Approach.....	13
Conclusion.....	14

Introduction

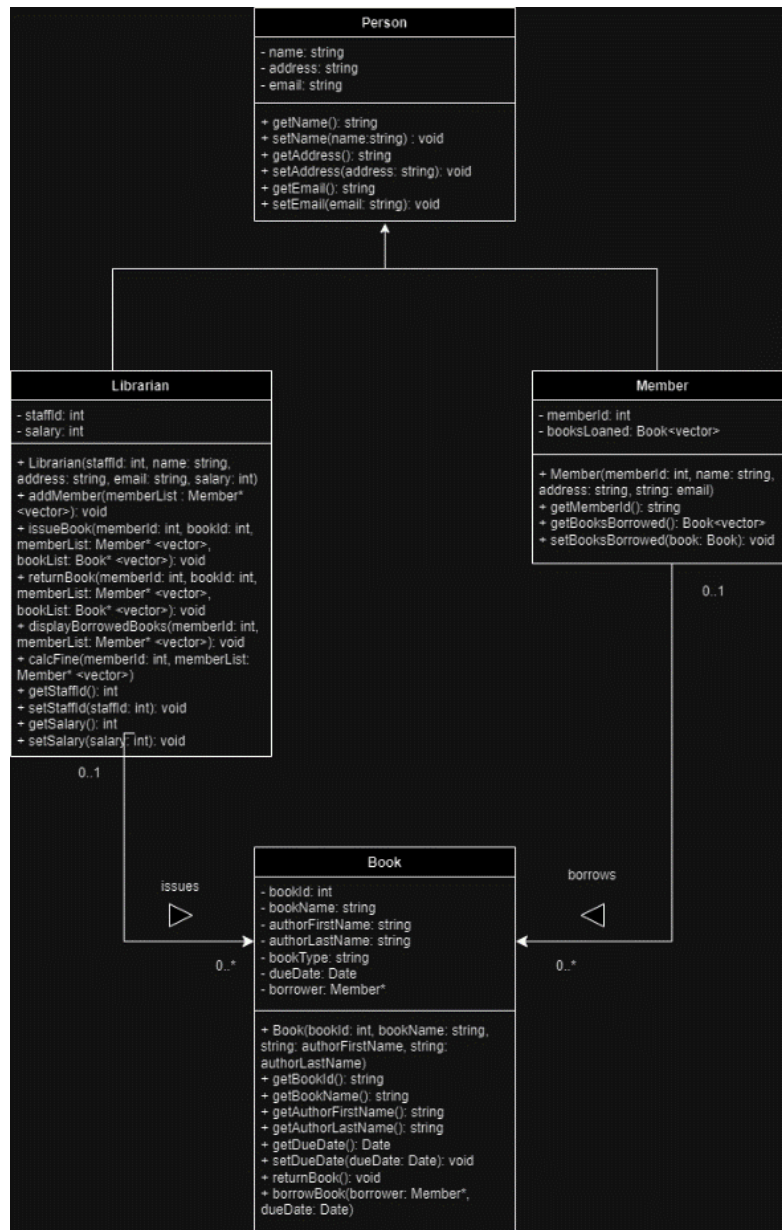
Student Number: M00829986

The program is meant to simulate a simplified version of a library system. The system's main classes consist of the following: Person, Member, Librarian and Book. The system is operated assuming the user is a librarian and is therefore in charge of the adding of members, as well as the issuing and returning of books. The system provides an interface in which the assumed librarian can perform these actions.

In this presentation, I will outline the UML diagrams that the system was based on, providing a clear description as to what each diagram is and how it is reflected in the actual implementation. Next, I will explain in detail the implementation as well as the tools used to achieve it and make workflow more efficient. After, the approach taken towards testing will be explored in depth. Finally, there will be a demonstration of the implementation.

Design

Class Diagram



This class diagram depicts the properties and methods assigned to each of the classes used by the system. There is a differentiation between private and public properties of each class.

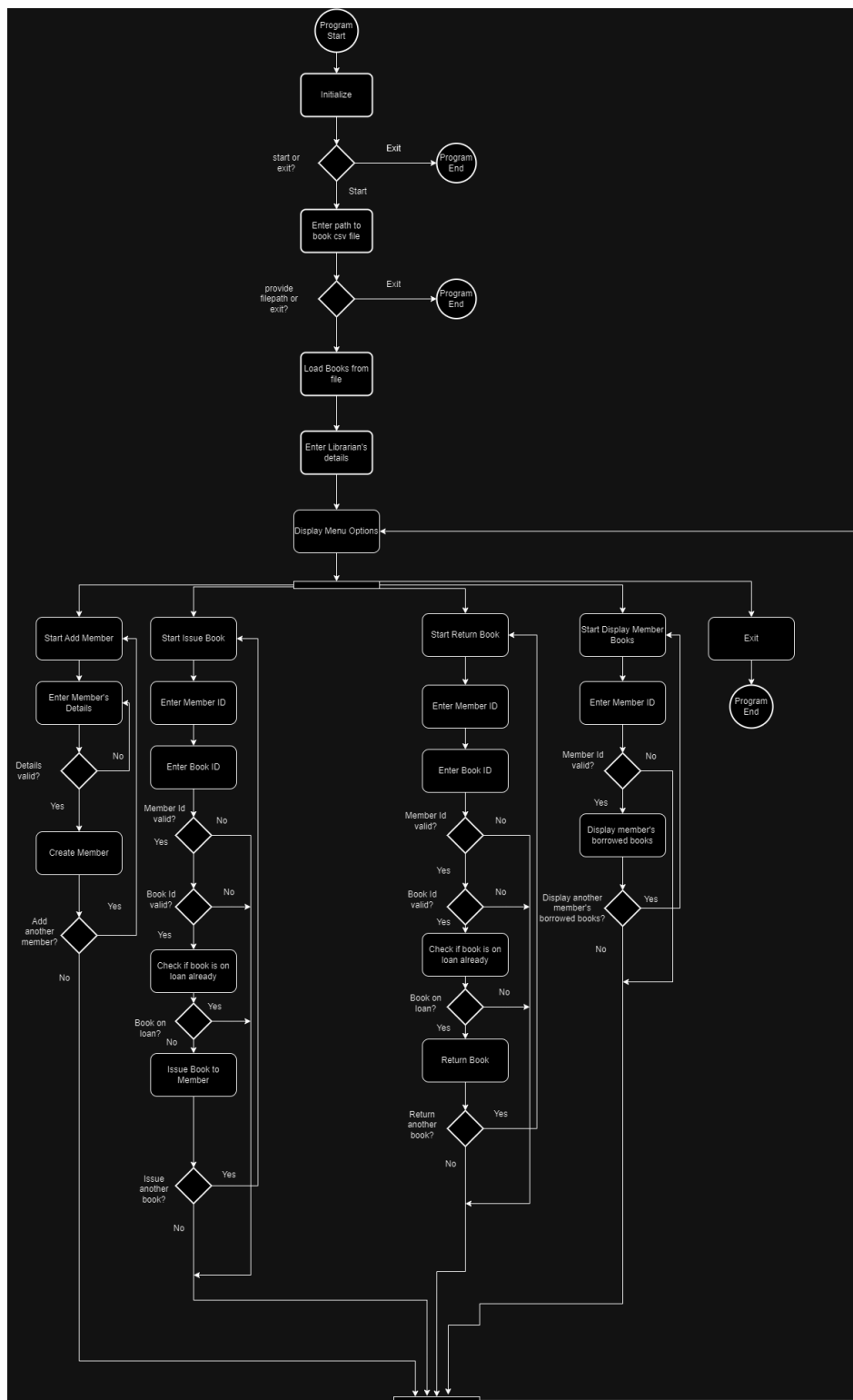
The relationships amongst classes are also outlined.

For instance, it is shown that both Librarian and Member inherit from the same parent class, that being Person. Other relationships as well as the multiplicities of the relationships are also depicted.

- 0..1 Librarian can issue 0..* number of books
- 0..1 Member can borrow 0..* number of books

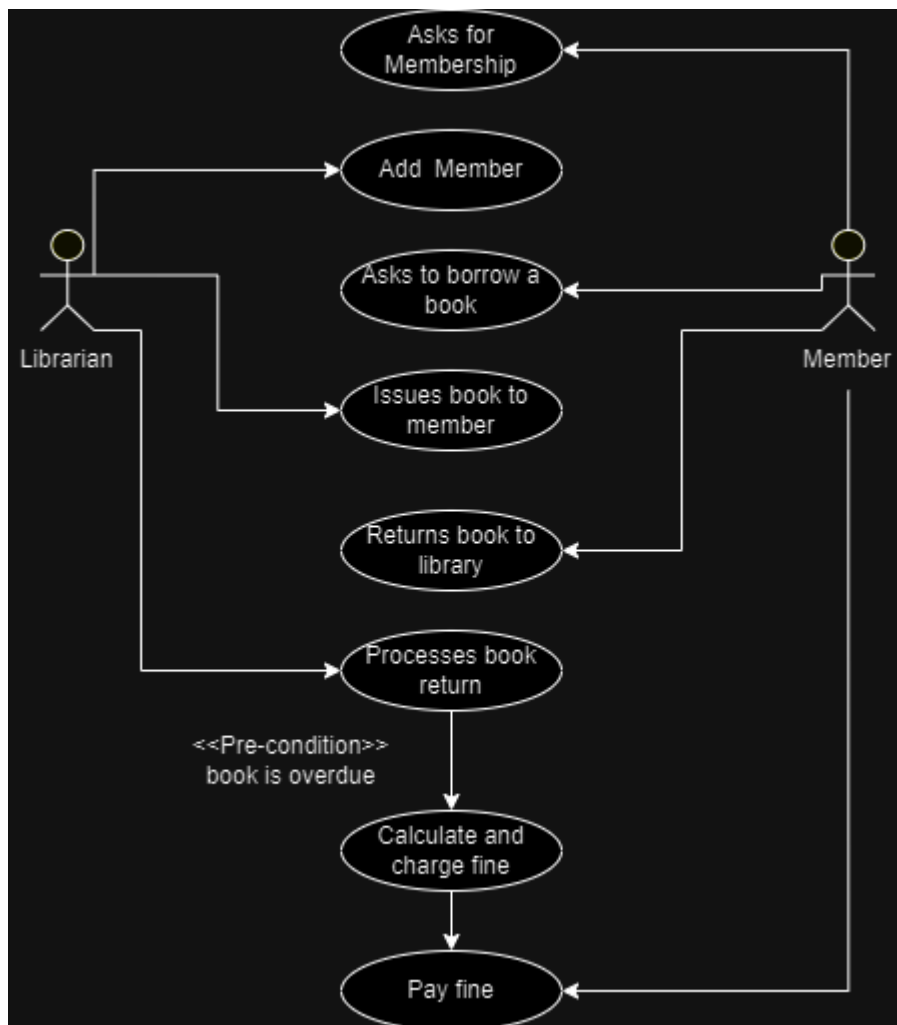
Overall, the class diagram is a way to clearly depict all the properties and methods needed for classes as well as providing an indication as to how the classes should interact in a proper implementation.

Activity Diagram



The activity diagram above represents the general design structure for the system. It outlines all the major functionalities provided by the system as well as the results when certain conditions are met. It extrapolates all possible outcomes from utilizing the system and simplifies it, allowing for easier implementation as the intended flow of the system is already outlined.

Use Case Diagram



The use case diagram is used to represent a common scenario in which the system may be utilized to solve. In the above diagram, the use case depicts an interaction between a librarian and a person seeking a membership to borrow a book. This is a likely scenario in which the library management system could be utilized.

The construction of these use cases allows for an easier implementation of a design as we are actively thinking about how the system would work in an actual scenario.

Implementation

When implementing the finalized design for the system I first went about creating as close as possible replications of the classes outlined in the design. Where necessary class properties or methods were adjusted to accommodate the coding environment (C++) being used.

Class Implementation

Person

```
class Person{
private:
    std::string name;
    std::string email;
    std::string address;
public:
    std::string getName();
    void setName(std::string name);
    std::string getAddress();
    void setAddress(std::string address);
    std::string getEmail();
    void setEmail(std::string email);
};
```

```
std::string Person::getName(){
    return name;
}
void Person::setName(std::string name){
    this->name = name;
}
std::string Person::getEmail(){
    return email;
}
void Person::setEmail(std::string email){
    this->email = email;
}
std::string Person::getAddress(){
    return address;
}
void Person::setAddress(std::string address){
    this->address = address;
}
```

The Person class is not directly used in the library management system but rather is used as a blueprint for which other classes can inherit from (Member and Librarian). As such the class consists only of important properties (name, address and email) as well as functions to change the values of the properties (setName, setAddress and setEmail) and a set of functions to get the value of these properties (getName, getAddress and getEmail).

Librarian

```
class Librarian : public Person{
private:
    int staffId;
    int salary;
public:
    Librarian(int staffId, std::string name, std::string address, std::string email, int salary);
    // Edited from UML
    void addMember(std::vector<Member *> *memberList);
    void issueBook(int memberId, int bookId, std::vector<Member *> *memberList, std::vector<Book *> *bookList);
    void returnBook(int memberId, int bookId, std::vector<Member *> *memberList, std::vector<Book *> *bookList);
    void displayBorrowedBooks(int memberId, std::vector<Member *> *memberList);
    void calcFine(int memberId, std::vector<Member *> *memberList);
    int getStaffId();
    void setStaffId(int staffId);
    int getSalary();
    void setSalary(int salary);
};
```

```
Librarian::Librarian(int staffId, std::string name, std::string address, std::string email, int salary){
    this->setStaffId(staffId);
    this->setName(name);
    this->setAddress(address);
    this->setEmail(email);
    this->setSalary(salary);
}
// Edited from UML
void Librarian::addMember(std::vector<Member *> *memberList){
}
void Librarian::issueBook(int memberId, int bookId, std::vector<Member *> *memberList, std::vector<Book *> *bookList){
}
void Librarian::returnBook(int memberId, int bookId, std::vector<Member *> *memberList, std::vector<Book *> *bookList){
}
void Librarian::displayBorrowedBooks(int memberId, std::vector<Member *> *memberList){
}
void Librarian::calcFine(int memberId, std::vector<Member *> *memberList){
}
int Librarian::getStaffId(){
    return staffId;
}
void Librarian::setStaffId(int staffId){
    this->staffId = staffId;
}
int Librarian::getSalary(){
    return salary;
}
void Librarian::setSalary(int salary){
    this->salary = salary;
}
```

The Librarian class acts as the core of the system, this is because the system was made assuming that the librarian will be acting as the user. The class inherits from the Person class, meaning it shares the properties and methods of the Person class. Most of the operations that the program provides are directly made by this class. Operations that the librarian class allows for:

- Adding of members
- Issuing of books to members
- Processing the return of books from members
- Displaying the books, a member has out on loan
- Calculating fines for overdue books

Member

```
✓ class Member : public Person{
    private:
        int memberId;
        std::vector<Book> booksLoaned;
    public:
        Member(int memberId, std::string name, std::string address, std::string email);
        std::string getMemberId();
        std::vector<Book> getBooksBorrowed();
        void setBooksBorrowed(Book book);
};

✓ Member::Member(int memberId, std::string name, std::string address, std::string email){
    this->memberId = memberId;
    this->setName(name);
    this->setAddress(address);
    this->setEmail(email);
}

std::string Member::getMemberId(){
    return std::to_string(memberId);
}

std::vector<Book> Member::getBooksBorrowed(){
    return booksLoaned;
}

void Member::setBooksBorrowed(Book book){
    this->booksLoaned.push_back(book);
}
```

The Member class inherits from the Person class, as such it possesses all the properties and methods of the Person class as well as having its own unique methods and properties. The class is used to represent the data stored about a Member, as such most of its methods and properties are used externally (by the Librarian class mainly), rather than having methods to operate upon itself.

Book

```
class Book{
private:
    int bookId;
    std::string bookName;
    std::string authorFirstName;
    std::string authorLastName;
    std::string bookType;
    Date dueDate;
    Member* borrower;
public:
    Book(int bookId, std::string bookName, std::string authorFirstName, std::string authorLastName);
    std::string getBookId();
    std::string getBookName();
    std::string getAuthorFirstName();
    std::string getAuthorLastName();
    Date getDueDate();
    void setDueDate(Date dueDate);
    void returnBook();
    void borrowBook(Member* borrower, Date dueDate);
};
```

```
Book::Book(int bookId, std::string bookName, std::string authorFirstName, std::string authorLastName){
    this->bookId = bookId;
    this->bookName = bookName;
    this->authorFirstName = authorFirstName;
    this->authorLastName = authorLastName;
}

std::string Book::getBookId(){
    return std::to_string(bookId);
}

std::string Book::getBookName(){
    return bookName;
}

std::string Book::getAuthorFirstName(){
    return authorFirstName;
}

std::string Book::getAuthorLastName(){
    return authorLastName;
}

Date Book::getDueDate(){
    return dueDate;
}

void Book::setDueDate(Date dueDate){
    this->dueDate = dueDate;
}

void Book::returnBook(){
    this->borrower = nullptr;
}

void Book::borrowBook(Member* borrower, Date dueDate){
    this->borrower = borrower;
    this->setDueDate(dueDate);
}
```

The Book class is used to store all information regarding the library system's listing of books. The list of books used by the system is loaded from a csv file. The following operations can be performed to a book:

- Issuing – A book can be issued out on loan to a Member class
- Returning – A book that was previously issued out on loan to a Member class can be reset to being not out on loan

Main Function Implementation

All the classes were made to interface with one another via the main function of the program. This main function made use of various helper functions, which performed a variety of tasks.

- `loadBooks(filepath : string) : void` – reads books from a csv file and loads them into memory as a list of Book classes
- `createLibrarian() : Librarian` – asks user to input data relevant to the librarian (name, email, address, salary and staffId) before creating an instance of the Librarian class with this information. The Librarian instance created in this function is the instance that will be used to perform various functions during the program's runtime
- `displayMenu() : int` – displays the operations the user is able to perform. Returns the selection made by the user
- `initialize() : string` – starts the program, provides the user to terminate the program early. This is where the user is asked to provide the file to be loaded by the system, this filepath is returned by the function once it is determined the provided input is an actual file
- `addMember(librarian : Librarian*, *memberList : Member* <vector>) : void` – essentially a wrapper function for the librarian class's `addMember` method. Allows for quicker, repeated execution of the adding member's functionality should the user desire
- `returnBook(librarian : Librarian*, *memberList : Member* <vector>, *bookList : Book* <vector>)` – a wrapper function for the librarian class's `returnBook` method. Ensures the data entered by the user is valid before passing to the librarian class's `returnBook` method. Also allows for quicker, repeated execution of the function should the user desire
- `bookIssuing(librarian : Librarian*, *memberList : Member* <vector>, *bookList : Book* <vector>)` – a wrapper function for the librarian class's `issueBook` method. Ensures the entered data is valid before passing to the librarian class's `issueBook` method. Also allows for quicker, repeated execution of the function should the user desire
- `displayMemberBooks(librarian : Librarian*, *memberList : Member* <vector>)` – a wrapper function for the librarian class's `displayMemberBooks` method. Ensures the entered data is valid before passing to the librarian class's `displayMemberBooks`. Also allows for quicker, repeated execution of the function should the user desire
- `main` – the main function ties all the classes and functions together and executes operations in a sensible order

Makefile

The makefile was used to compile all the headers and their associated .cpp files that made up classes. Once these files were compiled, they were linked together with the main file to generate the final program. The makefile also provided the additional functionality of cleaning the working directory by removing files leftover from compilation.

```
CC = g++

main: main.o classes.o
    $(CC) -o main main.o classes.o
    $(CC) test.cpp -o test classes.o

main.o: main.cpp classes.h
    $(CC) -c main.cpp



















classes.o: classes.h


















clean:
    -rm *.o
    -rm main
    -rm test
```

Version Control

Github was the site used to manage the program's version control. Version control was utilized as it allows one not only to keep track of progress being made towards project completion, but also allows for older versions of the program to be used instead of the most current. It typically allowed for a more streamlined workflow, being able to access and work on the project regardless of what device I had available was also very useful.

Github Commits

Commits on Jan 12, 2024		
Video presentation draft	Parrot Live user committed 22 minutes ago	b4cdb28  
Finalized changes	Parrot Live user committed 52 minutes ago	2e63396  
Added comments	Parrot Live user committed 1 hour ago	885f1bd  
Adjusted input validation for certain functions to prevent unnecessary repetition	Parrot Live user committed 10 hours ago	bfc49be  
Changes made to adhere with coding guidelines	Parrot Live user committed 12 hours ago	eb1bd8e  
Fixed return book functionality	Parrot Live user committed 12 hours ago	8778d20  
Commits on Jan 10, 2024		
Edited test cases and implemented returning of books	Parrot Live user committed 2 days ago	7afaf6e  
Commits on Jan 9, 2024		
Added test cases for creation of classes	Parrot Live user committed 3 days ago	7663abf  
Now reads in book names with commas properly	Parrot Live user committed 3 days ago	1f73001  

Commits on Dec 29, 2023		
Added fine calculation, needs to be integrated with book returning	Parrot Live user committed 2 weeks ago	adcdb72  
-Fully finished issuing of books function 	Parrot Live user committed 2 weeks ago	98a6dc4  
-Implemented the ability to issue books 	Parrot Live user committed 2 weeks ago	47eead1  
Commits on Dec 28, 2023		
-Implemented the addMember function of Librarian class 	Parrot Live user committed 2 weeks ago	42cddb3  
Commits on Dec 26, 2023		
Finished implementation of classes. Implemented loading data from CSV file	Parrot Live user committed 2 weeks ago	23e8518  
Commits on Dec 25, 2023		
Removed submission of .o files	Parrot Live user committed 3 weeks ago	df2214c  
Implementation of Person and Librarian classes as well as creation of makefile	Parrot Live user committed 3 weeks ago	d8b57e1  

Testing Approach

A reactive test approach was utilised for this project, meaning that the test design was formulated after the software was produced. An analytical approach was used to ensure that sections of the software critical to overall functionality were in proper working order.

The system being reliant on the classes outlined in the UML, they were the centre of most testing.

Test Cases	Description
Person Construction	Ensured that the Person class was functional, meaning all methods performed the correct operation and returned the correct results
Librarian Construction	Ensured that the Librarian class was functional. The class's constructor was tested to be functional (correct values being assigned to correct class properties) as well as the various other methods of the class.
Book Construction	Ensured that the Book class was functional. The class's constructor was tested to be functional (correct values being assigned to correct class properties) as well as the various other methods of the class.
Member Construction	Ensured that the Member class was functional. The class's constructor was tested to be functional (correct values being assigned to correct class properties) as well as the various other methods of the class

Conclusion

The project was an implementation of a library management system. It assumed the user was a librarian and as such provided the user with operations useful to a librarian. The system was split into various classes to organize data and allow for easier manipulation of the values using various functions and methods.

The design of the system was thought of first. The use of class diagrams, activity diagrams and use case diagrams allowed for a simpler representation of the way the various parts of system would behave and interface with one another.

For the actual programming, version control was utilized to provide a safety net when implementing the different sections of the system.

Test cases were designed post-implementation and were made to test the critical components that make up the system.

Various limitations and challenges arose throughout the development of the system. The most noteworthy relating to some of the initial designs for classes utilized by the system. The lack of certain methods and properties meant the need for long workarounds to be implemented.

Creating a similar system in the future I would ensure that the designs the system is based on, is able to accommodate all required functionality in a simple manner.