CHUFAN XUE

B00928273

OOP

Design and Development Document

Design and Development Document CHUFAN XUE B00928273

Software Background

In this assessment (PSA2-Com102), I have chosen to simulate a scenario where Luton City Borough Council serves as my hypothetical client and I'm tasked to programme a Book Lending Checkout System(BLCS) for their local library. This system is designed to modernized the library's checkout operations, making sure a smooth interaction between their clients and the collection of resources that are available in the library.

User Requirements

After several meeting with my client, I have noted down their key requirements:

1. The application must be able to handle two types of lendable items held by the library, i.e. Book and Multimedia, which have been predefined (read only) as given in the file ITEMS.csv

2. For each loan, their Barcode, User ID, Issue Date, Due Date and Number of Renews must be recorded.

3. The set of library users is predefined (read only) as given in file USERS.csv

4. The program should allow the librarian to issue an item (i.e. to create a loan). To do this, the librarian supplies the item barcode and the user id. The system should check that both the barcode and the user id exist. For books, loans are for a four week period. For multimedia items, loans are for a one week period. When an item is issued a loan object is created and added to the list / array of loans.

5. The program should allow the librarian to renew an existing loan. On supply of the item barcode, the loan's return date is increased by two weeks from the current date for books and one week for multimedia items and the number of renews is increased by one. A book cannot be renewed more than three times and the maximum number of renews for a multimedia item is two.

6. The program should allow the librarian to record the return of an item on loan. To do this, the librarian supplies the barcode. Items are always returned on or before their due date. When an item on loan is returned the corresponding loan object is removed from the list of loans.

7. The program should allow the librarian to view all the items currently on loan.

8. It must be possible to report (i.e. print to the screen) on the loan. This report should indicate the library name which could be hard-coded, the total number of each type of loans held by the library and the percentage of loan items having been renewed more than once.

9. When the program exits, the list of current loans should be written to LOANS.csv.

10. When the program is started, it should read from these csv files to re-populate the application with any previously stored data as a starting point.

11. The program user must also be able to search for a loan item by the barcode, causing the sought book details to be displayed if present.

12. The program must employ a console interface only

System Requirements

After analyzing the User Requirements, I have synthesized the essential system requirements necessary to achieve the desired functionalities of the Book Lending Checkout System, these requirements are:

System performance: the system must be lightweight and handle user inputs and file access as efficiently as possible. The programme can only be run within the console.

User interface: user interface must be minimalistic, easy to read and easy to navigate around.

Validation when loaning: ensuring to implement validation to check book barcode and user id are provided and valid before initiating the loan. Also the system must be able to calculate the due date and date of issue automatically without human intervention before initiating the loan.

Error handling: The system must have a robust error handling system to manage and respond to input errors, file access issues, and data validation failures effectively.

Validation of Loan return: validation on returning the loan before its due date and be removed from the LOAN.csv after the return has been authorized must be implemented into the system.

Loan Renew: the system must have the ability to extent a loan up to a limited amount of times.

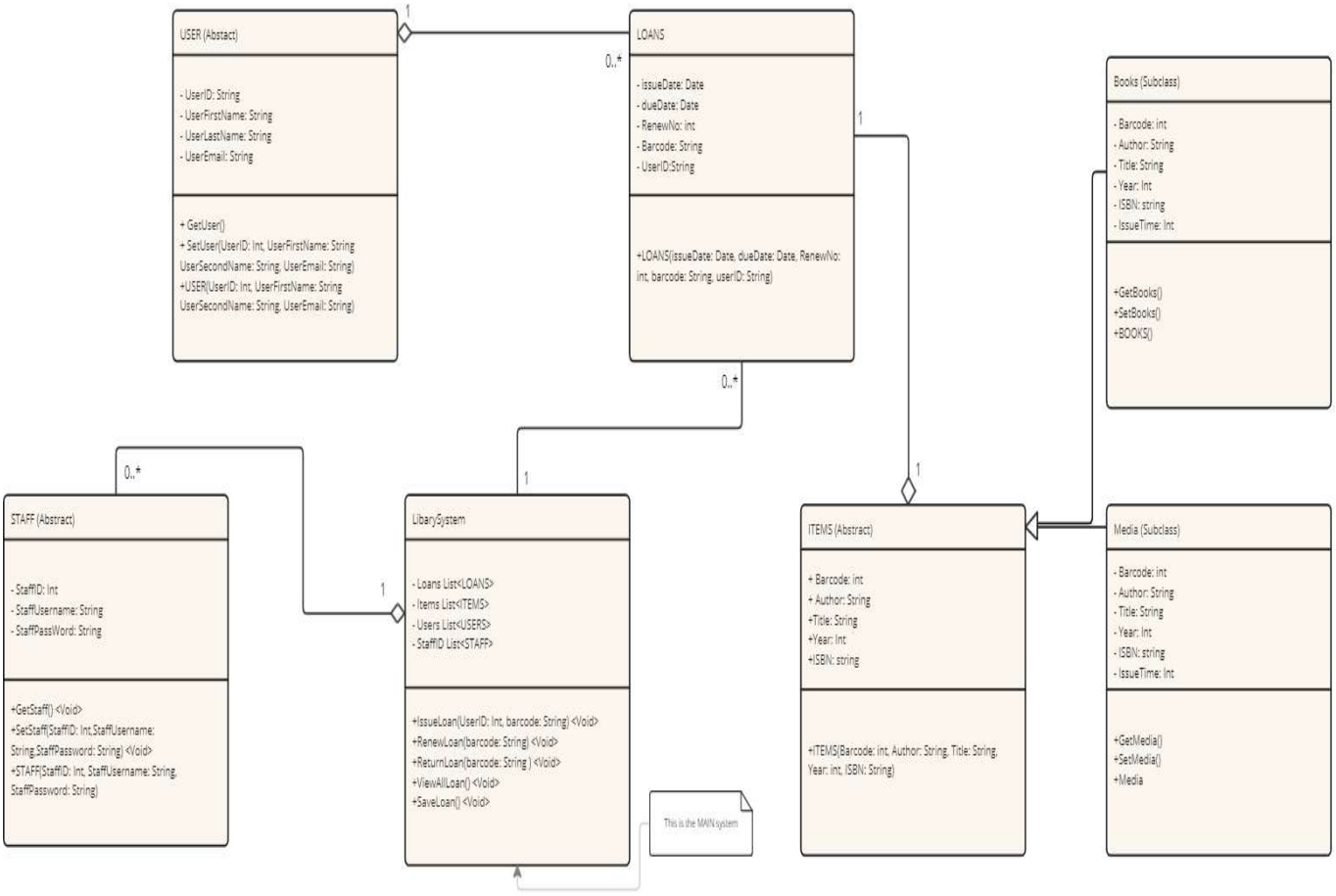
Loan report system: ensuring that a report of all loan is accessible by the staffs, the system must show a breakdown of all the loans with a easy to read layout.

data/File management: The system must be able to store and interact with .CVS files especially “ITEMS.cvs”, “USERS.cvs” and “LOAN.cvs”, the system must be able to read and write data into these files. Also ensuring data’s integrity and consistency when writing into these files.

The Design

In this section, I'll display the basic UML and data relationships for the Book Lending Checkout System, explaining why I have chosen them, and illustrate how they link to the User/System Requirements.

UML Diagram



Object orientated programming Justifications

UML designs:

Class	Comment
USER	The User class is going to be a subclass that's connected to the Loans class with a 1 to many connection. -one user can apply for many loans with different items. -for a loan to be valid there must be a user. -the user class has the ability to read from the user file
Items	The Items class is going to be a abstract class that's connected to the loan class using a 1 to 1 connection. It has 2 subclasses which are Books and Media . -there can only be one item per loan. -there can be 2 different types of items, each item has its own loan out time.
Loans	The loan class is going to collect the necessary information such as userID barcode ect... to make a loan. This class is connected by the user and items class using a 1 to many(User) and a 1 to 1(items) connection. -multiple loans can be made by one user. -each loan can only have 1 item - the user can only submit one loan at a time to the library system. - information about the user and the item can be access by the library system.
LibrarySystem	This is the main operation of the system as it validates and save the loan. This class is connected by the loan and staff class. As a (staff)many to 1 and a 1 to many(loan) connection. -the library system can save/mod multiple loans at a time -a staff can authorize multiple loan transition within the library system. -library system has the ability to read/write into the Loan and item file.
Staff	The Staff class has a 1 to 0..* (One to Many) connection to the library system. The staff class plays a big part in the validation of saving/authorizing a loan for a user. -each staff can authorize multiple loan or loan renews in the libary system.

Books	The book class is used as a subclass for the items class. The book class will holds all the books in the items files. And set an apocopate loan time for that item
Media	The media class is very similar to the books class. The media class will hold all the media in the items file and set a loan time for that item.

Data Sheet

Data Name	Data Type	Comments
UserID	String	This serves as a unique identifier for each user that's registered to the system.
FirstName	String	This holds the registered user's first name
SecondName	String	This holds the registered user's last name
Email	String	This holds the registered user's email address
Barcode	Int	This serves as a unique identifier for the Books, as type of books will only have one barcode.
Author	String	This holds the registered book's author
Title	String	This holds the registered book's Title
Type	String	This holds the registered book's type. There will only be 2 types Books/Multimedia as its written in the (1) user requirements
Year	DateTime	This holds the registered book's year of published in a format of (YYYY)
ISBN	String	This holds the registered book's ISBN. Which is a mixture of numbers and symbols.
IssueDate	DateTime	This holds the registered Loan's issue date. In the format of (YYYY/MM/DD)
dueDate	DateTime	This holds the registered Loan's Return date. In the format of (YYYY/MM/DD)
RenewsNum	int	This holds the number of renews on that loan.

Method definition

Method Name	Method Type	Comments
User Class		
+GetUser()	Void	This method allows me to access all the private attributes within the User class
+ SetUser(UserID: Int, UserFirstName: String UserSecondName: String, UserEmail: String)	Void	Similar to a constructor this method allow me to change the information of the private properties within the class after the user is already loaded in.
+USER(UserID: Int, UserFirstName: String UserSecondName: String, UserEmail: String)	Constructor	This is the constructor of the user class.
Books		
+GetBooks()	Void	This method allow me to access all the private attributes within the books class.
+SetBooks()	Void	This method allow me to change all the private variables within the Books class
+BOOKS()	Void	This is the constructor of the books class.
Media		
+GetMedia()	Void	This method allow me to access all the private attributes within the Media class.

+SetMedia()	Void	This method allow me to change all the private variables within the subclass
+Media()	Void	This is the constructor of the books class.
Items		
+ITEMS(Barcode: int, Author: String, Title: String, Year: int, ISBN: String)	Void	This is the constructor of the items class. It allows me to gather information of the Book or media item that the client picked.
Loan		
+LOANS(issueDate: Date, dueDate: Date, RenewNo: int, barcode: String, userID: String	Void	This is the constructor of the Loans class. This allows me to gather all the information that's required to make a loan and set the infomration within the loan class.
Library System		
+IssueLoan(UserID: Int, barcode: String) <Void>	Void	This method will be programmed to manage the loan issue process. A user and a item will need to be picked. After validating item's quantity and user input and obtaining staff authorization, the loan will be issued to that user. With output of a receipt congaing return date, issue time, client id and book barcode.
+RenewLoan(barcode: String) <Void>	Void	This method will be programmed to manage the loan renewal process. The user will need to enter the item's barcode. After validating how many times the loan has been renewed, and obtaining staff authorization, the loan will be renewed.
+ReturnLoan(barcode: String) <Void>	Void	This method will be programmed to run the return loan process. Where the staff/user must input the book's barcode and after staffs authorization the loan will be remove from LOAN.cvs
+ViewAllLoan() <Void>	Void	This method will read ,organized a output all the information within the LOAN.cvs file
+SaveLoan() <Void>	Void	This method will allow me to save the loan that validated and authorized to the LOAN.cvs file.
Staff		
+GetStaff() <Void>	Void	This class will allow me to access all the private attroubutes within the staff class
+SetStaff(StaffID: Int,StaffUsername: String,StaffPassword: String) <Void>	Void	This method will take the process of setting/modifying staff's private variables.
+STAFF(StaffID: Int, StaffUsername: String, StaffPassword: String)	Void	This is the constructor of the class. Thus allowing me to populate that class with staff's information from the staff file.

The use of Inheritance

Inheritance has been used multiple times, this improves the software's code reusability and organization. by allowing classes to inherit properties and methods from existing old classes, this not just improves the software's efficiency it also reduces the size of the software making it very lighweight.

The use of polymorphism

Polymorphism has been used in this project to allow me to handle different types of Loans submitting and user interactions seamlessly through a unified code structure. This approach enables the system to adapt its functionality based on the specific needs of each transaction type without complicating the overall codebase.

The use of abstraction

Abstraction is used in this program to simplify complex operations by hiding the details within. For example, data such as user details, staff information, and items are accessed and managed through a high level get methods, ensuring a safe and simple interactions with the elements.

The use of Encapsulation

Encapsulation, such as Public, Private, and Protected properties, has been applied many times within the program. The use of encapsulation ensures that the internal state of an object is safely hidden from outside interference, which enhances modularity and prevents data from being exposed or modified unexpectedly. This approach also simplifies maintenance and upgrades, as changes to the encapsulated code can be made with minimal impact on other parts of the program.

File handling

File handling measures such as read and write operators are commonly used within the checkout system to manage data flow. The file reading operator is essential for accessing existing data files, such as USER.cvs and LOAN.cvs. This ensurs that the system can fetch and display this information. Additionally , file writing is also crucial for updating and adding new information to data files, such as processing loan renews or new loan applications ensuring that all data are updated and accurate. Additionally, the system validates these information during these operations to handle any issues that might occour from the file access, such as corruption or inaccurate data format.

Error Handling

Error handling measures such as try-catch, finally blocks, and Error handling design patterns will be used within the library checkout system. Implementing these measures is important as they help to manage system errors, ensuring the system remains robust and reliable. The try-catch block will allow the system to catch an exception that has occured during the execution of the program(the try block), effectively avoiding these errors so the program doesn't crash. Additionally, the finally block will be used to ensure that cleanup procedures, such as closing file streams and scanners are performed regardless of an exception occurs, thus maintaining the integrity of system resources. Furthermore, by adopting error handling design patterns, the system can be structured in a way that isolates error handling from other sections of the software, making the code cleaner, easier to read and maintain. Having these measures in place significantly lower the risk of critical errors disrupting the user experience and enhance the system's ability to recover from unexpected situations.

Loops and data control measures

Control measures such as: while loops, switches, for loops, and foreach loops will be used within the library checkout program. The use of loops is important as it's utilized for validation, file handling, and data collection. The use of while loops in this progam can ensure that the input received from the user meets certain criteria before proceeding to the next step, which is essential for maintaining data integrity within the system. The use of For loops and foreach loops are used in iterating over the collections of ITEMS, such as the VIEW ALL LOAN option and categorizing all the ITEMS. Additionally, switch statements offer a structured way to handle multiple conditions without the need for complex if-else chains, enhancing the readability

and over all efficiency of the code. These loops and control measures are very important in creating a robust and efficient Libary checkout system thats able to handle various user inputs and complex validations.

Assumptions made

In this planning phase and UML diagram I've assumed the following points:

1. Each item can only be lend out to 1 client at a time
2. Only 1 client will be using this system at a time
3. A staff member needed to verify each loan.
4. I'm only creating a lending system. But have left methods and varables that can be used for other system.
5. I don't need to add or modify a clients, staff or items information.

Has the User requirement been satisfied.

Appendix

(UML diagram)

https://miro.com/app/dashboard/

(This document)

Microsoft Word

(IDE used)

https://www.jetbrains.com/idea/download/?fromIDE=&section=windows

(Programming language used)

https://www.oracle.com/java/technologies/downloads/

(other research resource used when programming)

https://www.w3schools.com/java/

https://www.geeksforgeeks.org/inheritance-in-java/

https://www.programiz.com/java-programming/class-objects

https://creately.com/blog/diagrams/uml-diagram-types-examples/

https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

https://junit.org/junit5/docs/current/user-guide/

https://www.freecodecamp.org/news/java-unit-testing/

https://blog.jetbrains.com/idea/2020/09/writing-tests-with-junit-5/