BookWare LS

CS4125: System Analysis and Design Project

SEMESTER 1 – 2015

Group Members:
Michal Wrzosek – 13165909
Raymond Lee – 13132436
James Licup - 13147013
Conor Stephens – 13123947

Table of Contents

Table of Contents

Project Plan & Allocation of Roles

Deliverable	Description	Week	Assigned
Presentation			
	Cover Page, Logo, Design	4	RL, MW
Narrative			
	Overview	5	RL
Software Lifecycle			
	Discussion on Software Lifecycle	5	CS
Requirements			
	Use Case Diagram	6	Group
	Use Case Description	6	Group
	Non-Functional	6	RL, JL
	Discussion on Tactics to Support Quality Attribute	7	RL
	GUI Prototypes	7	MW, CS
System Architecture			
	Discussion on Package Diagram	8	RL, CS
Analysis			
	Data Driven Design	9	CS
	Class Diagram	9	RL, MW
	Sequence Diagram	9	CS
	E-R Diagram	9	JL
Code			
	Code	10	Group
Design			
	Architectural Diagram	11	JL
	Class Diagram	11	JL, CS
	Interaction Diagram	11	JL, MW
	State Diagram	11	CS
	Description of Design Pattern Used	11	RL, CS
Critique			
	Critique on Analysis/Design	12	Group
Reference			
	Sources Used	12	Group

Narrative

Overview

Over the last number of years, the number of university applicants has reached a record high as demand for higher education courses continues to rise. In a recent

study carried out in 2015, it is estimated that applications from the EU have risen by 7 per cent, while those from outside the EU have also gone up by 3 per cent overall.

Overall, this has led to an increase in the amount of students accepted into university and thus has placed a bigger demand from universities all around the world. With the tuition fees rising every year, these results with higher quality in teaching and in order for students to reach their maximum potential, a stable, secure and reliable library system needs to be placed in universities in order to supply their demands.

Also with an increase in the varieties of course available to students, more books are required for different topics in university libraries. Not only in universities are library systems under pressure but also public libraries. In this year alone the amount of new titles published as this is written is 2,237,688.

Libraries need a management system where they keep track of all the books members loaned out, the amount they owe and to prevent them loaning more books when they reach their limit. This is where BookWare LS come in.

What is BookWare LS?

BookWare LS is a library management system for libraries. This system will then be sold onto a number of BookWare LS clients. It allows librarians to add new books/titles, delete any unnecessary books, allow for updates and browse/search through books in the database.

Members can access books by loans, check their history and balance they owe, they are unable to loan any books out if they already have more than 3 books currently loaned and make reservations on books they wish to loan in the near future. The aim is to encourage more students/members to use the library to loan out books with the system's easy accessible user interface

Software Lifecycle

After discussing three potential lifecycle models: Agile, Waterfall, V-model we came to the decision that Agile was the most suitable. We felt the alternatives introduced characteristics to the project that would hinder the development of the software.

Waterfall model:

This model reduces our ability to adapt to changes that might occur in the business domain we are targeting. Working code is a later priority in this solution, and we

came to the decision that it was of the up-most importance to have a more incremental development of the product.

Testing performed through the waterfall model is the last step before publishing. Our project is not suited for this type of test as test harness is necessary for every programmed class to ensure integrity of the database. Carrying out such tests at the end of development would compromise the effectiveness of such executable test cases.

V-Model:

Very inflexible model for software development focusing on two streams for verification and testing, with implementation happening once verification is complete but before testing. Being a small group we felt like we didn't need to take such lengths to ensure such lengths for integrity of design.

Since software is only developed at the implementation phase of the V-Model there are no prototypes or proof of concepts to show stakeholders. We believe we should have something to show them at the earliest possible stage of the project.

Agile Software development

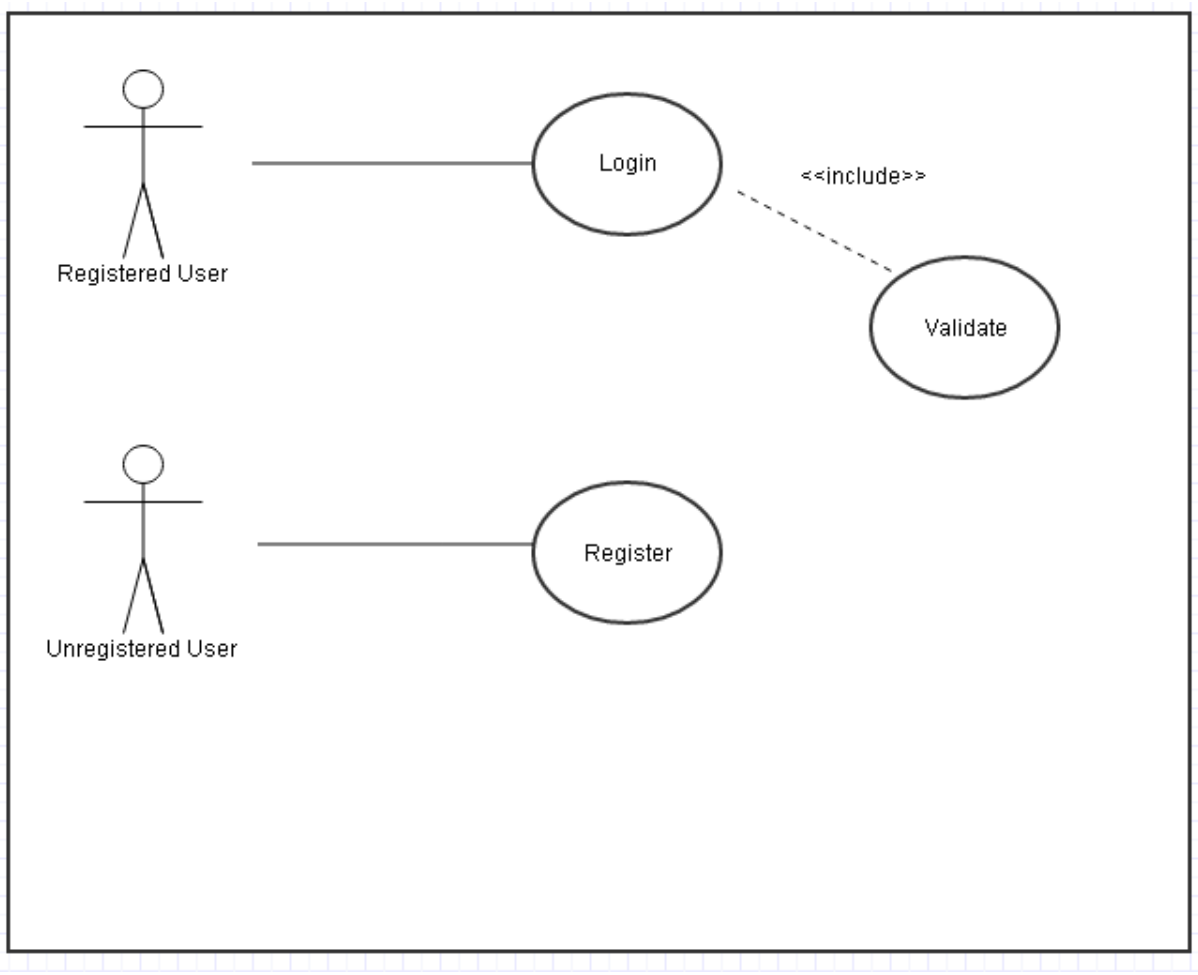
Agile methods break tasks into small increments each progressively approaching the final solution. Every aspect of the project is continually revisited throughout the lifecycle of the projects life. We believe this 'inspect-and-adapt' approach will significantly reduce our development cost, time to market and help meet our stakeholders' expectations of the project sooner.

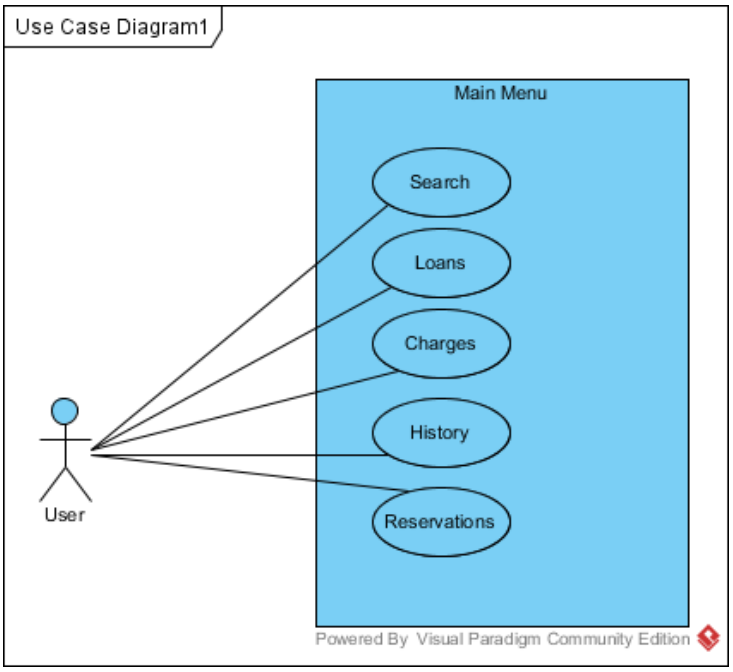
Waterfall and V-model are sequential process to develop software. Compared to these set-in-stone approaches, agile software development focusses on adaptability and agility. To do so the agile model involves multiple iterative development schedules that we feel would help our programmers by constantly having them create better and more engaging code.

Requirements

Functional Requirements

Use Case Diagram





Detailed Use Case Description

Search Book

Name: Search Book

Description: Allows user to search the book from a database by inserting title or author. As a result program displays list of books.

Actors: User

Pre-Conditions:

1. User is logged in

Flow:

1. User selects “Search”.
2. System displays an input Dialog with list options between “Title” and “Author” to search by.
3. User selects an option and input the keywords.
4. System searches for the book in the database.
5. System displays list of books.

Alternate Flow:

2. System displays a input Dialog with list options between “Title” and “Author” to search by.
   a) User selects “Back”. System returns to main menu.
4. System searches for the book in the database.
   a) Could not find a book and lets user to try search another book.

Post-Condition:
1. User gets list of books in a format Title ,Author, Reserved, Loaned
2. System returns to main menu

Loan Book

Name: Loan Book

Description: Allows user to loan a book from library for a period of time. If user did not return already loaned book on time, he won’t be allowed to loan another one.

Pre-Condition:
1. User is logged in
2. User returned all books on time
3. User has less than three books loaned at the same time

Flow:
1. User selects “Loan”
2. System displays list of loaned books by the user and button to “Loan a Book”
3. User selects “Loan a Book”
4. System displays Input Dialog with option to loan by “Title” or “Author”
5. User selects option and inputs the keywords.
6. System checks does the book exist in the database.
7. Displays Book and Confirm Dialog “do you want to loan this book”
8. User selects “Yes”
9. System sets Date and Time when the book has been loaned and when it needs to be returned to user account

Alternative Flow:
2. System displays list of loaned books by the user and button to “Loan a Book”
   a) System will display clean list if the user didn’t loan any of the books from library
4. System displays Input Dialog with option to loan by “Title” or “Author”
   a) System won’t allow user to input any data further if he did not return a book on time
   b) System won’t allow user to input any data further if he has already three books loaned
7. Displays Book and Confirm Dialog “do you want to loan this book”
   a) Displays Message “The book wasn’t found” and asks user to try loan a book again

Post-Condition:

1. User loans a book. All data is saved into users account.
2. User could not loan a book because it wasn't returned on time.
3. User could not loan a book because he already took three books at the same time

Delete Book

Name: Loan Book

Description: Librarian deletes a book from a database, given that the book selected is not currently on loan.

Pre-conditions:

1. User is a librarian
2. Book to be deleted is not currently on loan

Flow:

Alternative Flow:

Post-Condition:

Add Book

Name: Add Book.

Description: Librarian adds a book to the database.

Actor: User.

Pre-conditions:
1. User is a librarian.
Flow:
1. Librarian selects "Browse Catalogue".
2. Librarian selects "Add Book".
3. Librarian enters Book details.
4. Librarian clicks Finish.
5. Book is added.
Alternate Flow:
6. Librarian clicks Finish.
    a. Empty field.
    b. Book already exists in the database.

Post-conditions:

1. Book is added to the database.

Browse

Name: Browse

Description:

Actor: User

Pre-condition:
1.

Flow:
1. User selects "Browse Catalogue".
2. User can then sort the books based on specified filters.
3. When the user is finished browsing, click "Finish".

Post-conditions:
1. Once finished browsing user is returned to the main menu.

Structured Use Case Description

Login

Actor Action	System Response
1. Selects “Login”	2. Login form displays
3. Enter details and click submit	4. Login details are verified, enters main menu
Alternative course: 3a: Invalid information entered: • System displays error regarding incorrect information.	

Non-Functional Requirement: **Performance**

System should respond within 5 seconds to the login request. The change between GUI displays should be fast and responsive.

Logout

Actor Action	System Response
1. Selects “Logout”	2. System updates and saves their session, user is set to offline.

Create Account

Actor Action	System Response
1. Selects “Create Account”	2. Register form displays
3. Enter details and click submit	4. Registration details are verified, user verification email.
Alternative course: 3a: Invalid information entered: • System displays error regarding incorrect information.	

Non-Functional Requirement: **Security**

System should encrypt the user’s personal information and passwords and store them securely.

Check for Balance due On User Account

Actor Action	System Response
1. Selects “Balance Due”	2. Displays the amount of money due in euros

Check User History

Actor Action	System Response
1. Selects “History”	2. Displays list of books previously or currently loaned
3. Selects a book	4. Displays book information, date loaned, return or due returned date
Alternative course:/ 2a: No History: • System displays no history message.	

Search Book

Use Case: 001	Search Book by title
Actor Action	System Response
1. User selects “Search”.	2. System displays a input Dialog with list options between “Title” and “Author” to search by.
3.User selects a option “Title” and enters title	4. System searches for the book by given title in the database.
	5. System displays list of books.
Alternative Route: 2a: User selects “Back” •System returns to main menu. 4a: Could not find a book in database. •System allows user to try finding a book again.	

Use Case: 002	Search Book by title
Actor Action	System Response
1. User selects “Search”.	2. System displays a input Dialog with list options between “Title” and “Author” to search by.
3.User selects a option “Author” and enters title	4. System searches for the book by given author in the database.
	5. System displays list of books.
Alternative Route: 2a: User selects “Back” •System returns to main menu. 4a: Could not find a book in database. •System allows user to try find a book again.	

Loan Book

Use Case:001	Loan a Book
Actor Action	System Response
1.User selects “Loan”	2.System displays list of loaned books by the user and button to “Loan a Book”
3.User selects “Loan a Book”	4.System displays Input Dialog with option to loan by “Title” or “Author”
5. User selects option and enters the keywords	6. System checks does the book exist in the database.
	7. Displays Book and Confirm Dialog “do you want to loan this book”
8.User selects “Yes”	8.System sets Date and Time when the book has been loaned and when it needs to
Alternative route 2a: System will display clean list if the user didn’t loan any of the books from library 4a: System won’t allow user to input any data further if he did not return a book on time 4b:System won’t allow user to input any data further if he has already three books loaned 7a: Displays Message “The book wasn’t found” and asks user to try loan a book again.	

Non-Functional Requirement: **Usability**

The UI for browsing through the history should be user friendly and easy to use.

Make a Book Reservation

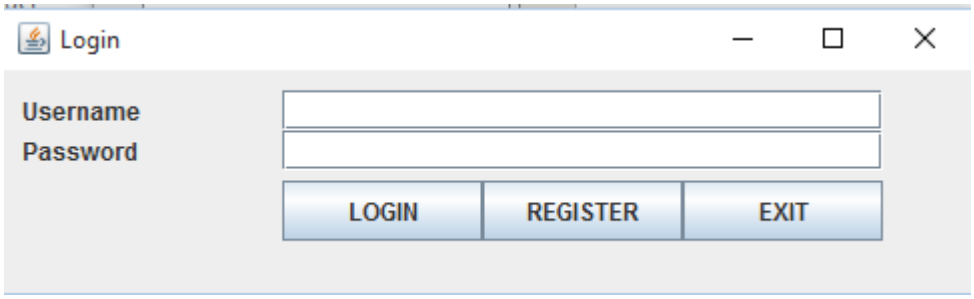
Actor Action	System Response
1. Selects “Reservation”	2. Asks user to enter book name or author
3. Selects a book	4. No action.
5. Can select how long they want to reserve book for	6. Book is reserved for user until date chosen.
Alternative course: 2a: No specified book name or author in database: • System displays error 6a: User reserves book for too long • Cannot reserve book for user. System displays error 6b: User is already has a book reserved • Cannot reserve book for user. System displays error	

Non-Functional Requirement: **Extensibility**

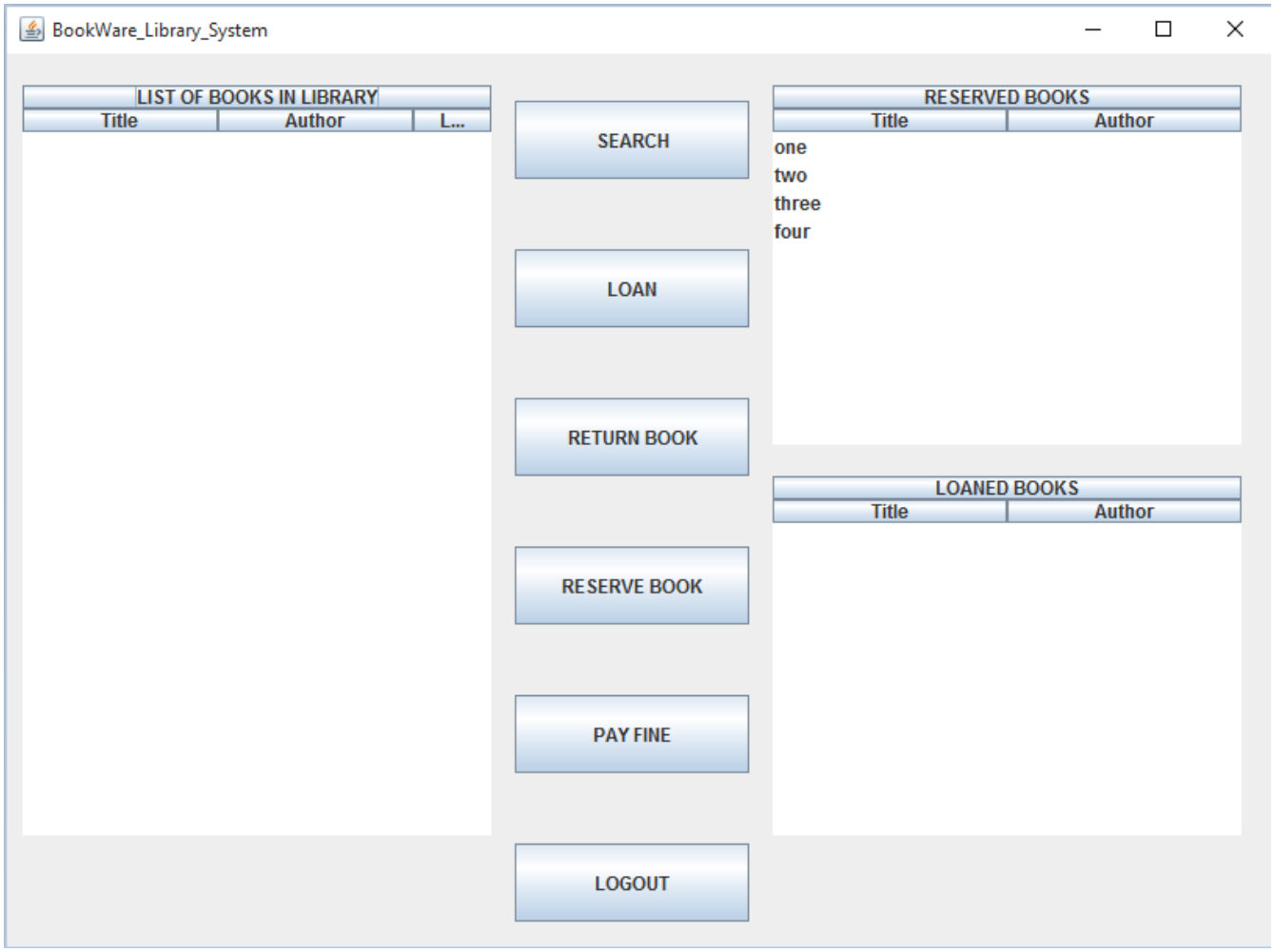
The algorithm for selecting which book should be reserved for the user should be modifiable by the customer.

GUI Prototypes

Log In Screen



Library Interface



Discussion on Tactics to Support Quality Attributes

As a team, we decided to program with the use of Java Programming language after much discussion. Java is very portable among many systems and architecture which is a massive benefit to us. Java is the most popular and commonly used programming language used among software developers and software engineers worldwide. This will enable us to recruit new members to work on our system easier and also to maintain and update our system in the future.

Our target is to deliver the highest of standards to supply to libraries worldwide. In order to achieve this we are taking the agile development approach and constantly improve our system. With this approach it would assist us to help maintain our system's reliability modifiability

One of our priority is to ensure our users are confident when using our system and have them reassured that any private information is stored efficiently and safely. To

support this ability we are limiting access to the server functions to users with valid credentials and have the required access level.

For our system to be successful and unique, we wanted to make our UI user friendly and easily accessible. New users should be able to browse through our menus and windows effortlessly. To support this, we have clear straight forward layout and design for our UI. Also we developed our own algorithms to ensure accuracy to all information and service we deliver to our users.

Analysis Artefacts

Data Driven Design

Candidate Objects

In order to list any possible objects we used 'noun identification'. After building the initial list we then filtered it by using Heuristics in order to specify more meaningful objects.

The initial List:

Login Interface	Library Interface	Librarian	User	DataBase
Login	Menu	Remove Book	Loan Book	
Register	Reserve Book	Remove user	Reserve Book	List of Reserved Books
Option	Loan Book	List of Users	Return Book	List of Loaned Books
Validation	Filter	Add Book	Pay Fine	Update
Browser	Return Book	BookManager	UserManager	Journal
Display	UserName		List of Books	List of users
NewUserScreen	Help			List of Librarians

Heuristic applied:

1. Too vague or specific – **RED**
2. Out of scope – **BLUE**
3. Attribute – **GREEN**
4. Operation – **YELLOW**
5. Equivalent(other similar objects) – **PURPLE**

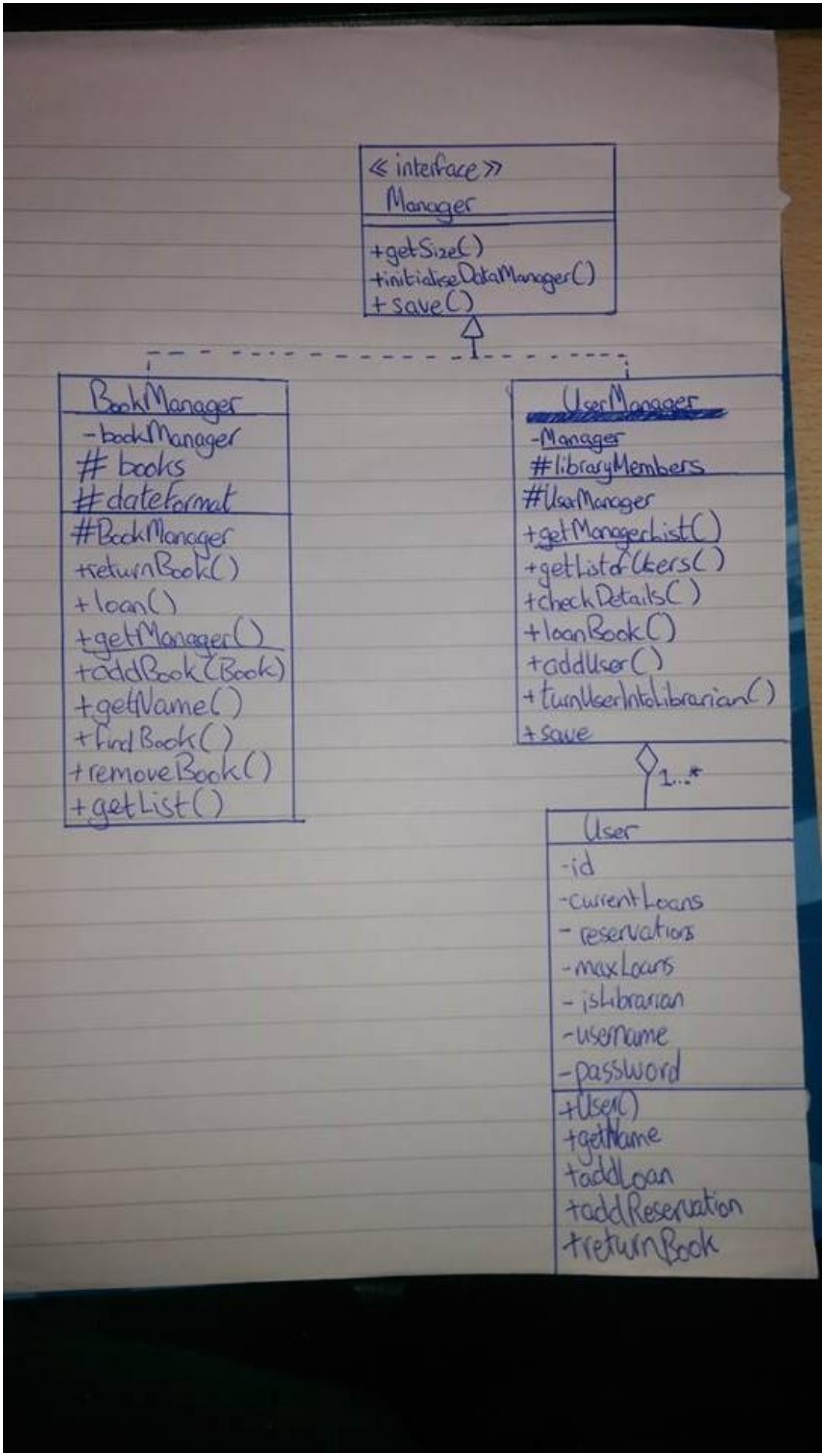
The Filtered List:

LoginScreen	User	Librarian	ShowDetailsInterface

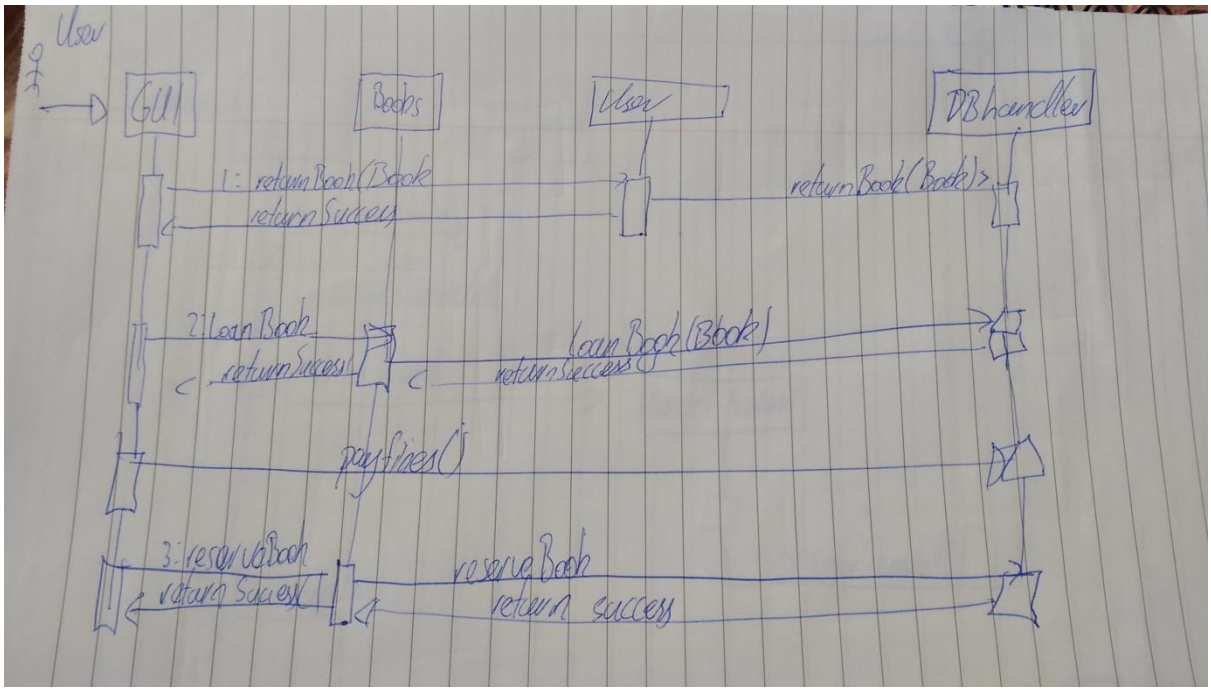
NewUserScreen	Book	UserManager	Journal
LibraryIterface	BookManager	AddBookIterface	AudioBook

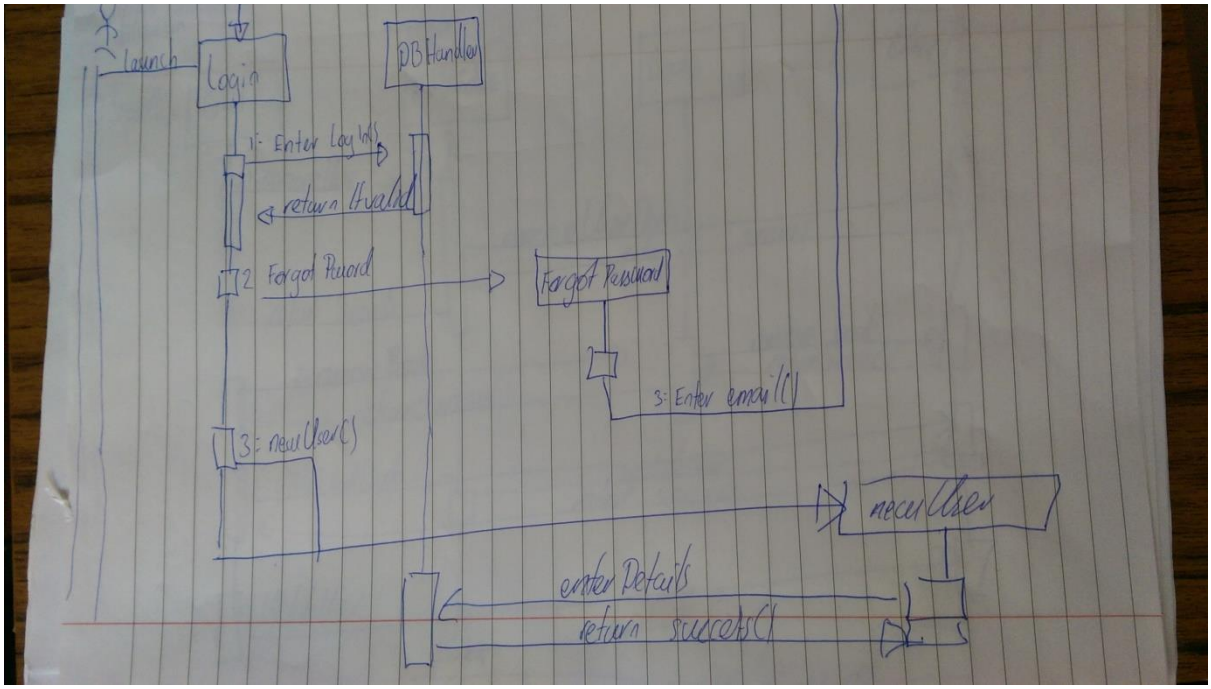
LoginScreen GUI to log in dependent on UserManager.
NewUserScreen GUI to log in dependent on UserManager.
LibraryIterface GUI to log in dependent on UserManager and BookManager.
User Entity class.
Book Entity class.
BookManager Controller class that manages Books.
UserManager Controller class that manages Users.
AddBookIterface GUI to log in dependent on BookManager.
ShowDetailsInterface GUI to log in dependent on BookManager.
Journal Inherits from Book features different print() function.
AudioBook Inherits from Book features different showDetails() function.

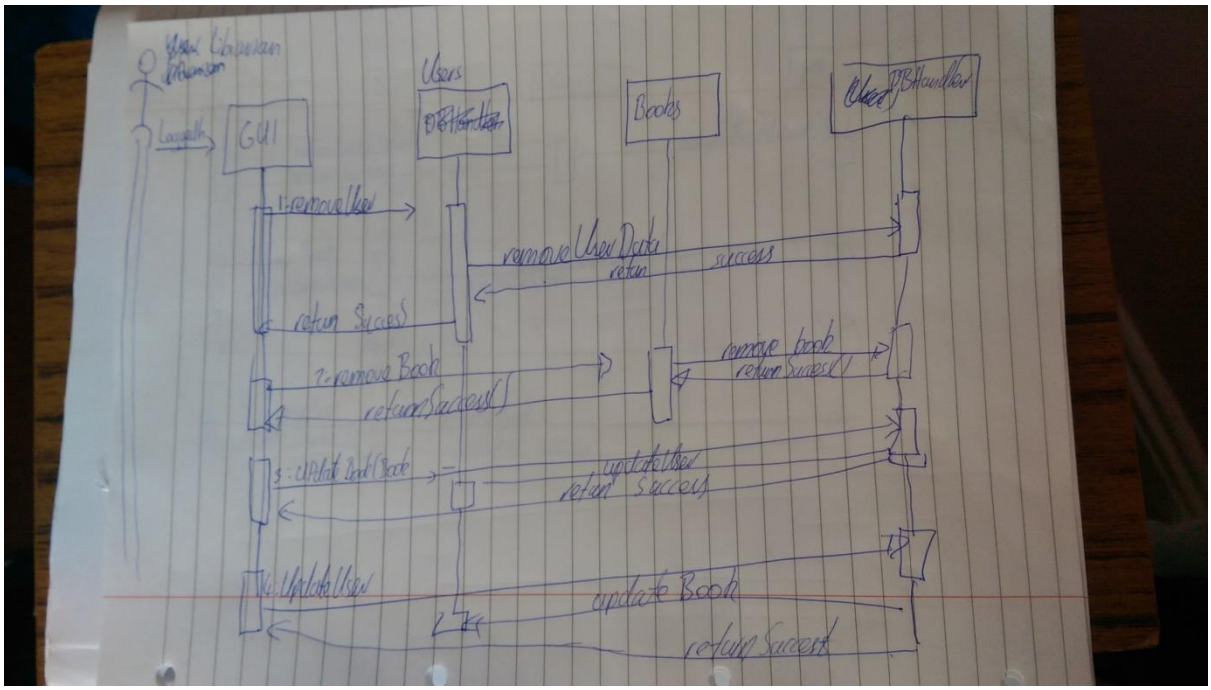
Class Diagram


«interface»
Manager
+getSize()
+initialiseDataManager()
+save()
BookManager
-bookManager
#books
#dateformat
#BookManager
+returnBook()
+loan()
+getManager()
+addBook(Book)
+getName()
+findBook()
+removeBook()
+getList()
UserManager
-Manager
#libraryMembers
#UserManager
+getManagerList()
+getListofUsers()
+checkDetails()
+loanBook()
+addUser()
+turnUserIntoLibrarian()
+save
1..*
User
-id
-currentLoans
-reservations
-maxLoans
-isLibrarian
-username
-password
+User()
+getName
+addLoan
+addReservation
+returnBook

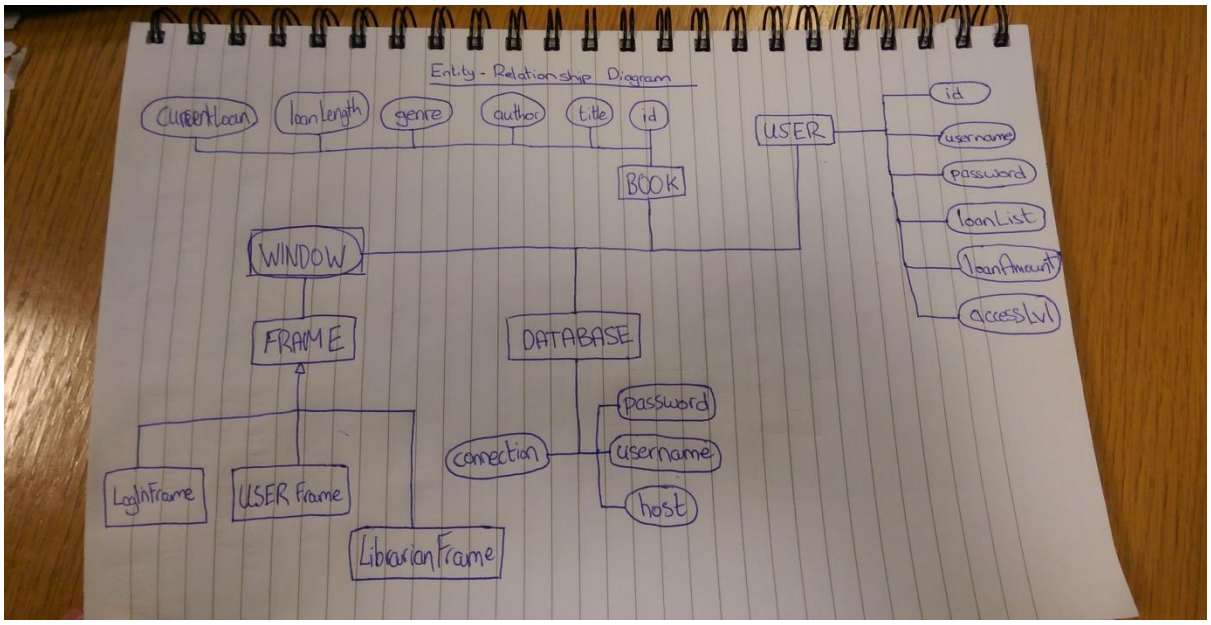
Sequence Diagram







Entity-Relationship Diagram



System Architecture

System Architectural Decisions

The system for this project is divided into two separate sub-systems. Each sub-system, the GUI (main interface) and file system runs independently from each other and strict rules are in place to ensure that only the main system can only access the file system.

The User interacts directly with the interface which contains most of the logic for the application. The program responds to events (button clicks) and the required data is retrieved and the interface is updated. The User does not have direct access to the file system that is used. The basic principle of this approach is that it enhances security and usability of the program.

The file sub-system contains all the data the interface needs to perform every operation. It is responsible for updating the user interface and contains user-specific information that is used to authenticate and validate actions made by the user.

Implementation Language

After a short discussion, we decided to use the Java programming language to implement our systems. Using java allows our software system to be run and available on any system or architecture that supports the Java Virtual Machine (JVM) as it is very portable. Also, every team member had experience working with java so we thought it this was best suited for optimum productivity during the coding stage. If we had more time and experience with C++, we would have choose this language to develop this system.

Although C++ would increase speed and performance of our software, we decided that it would make our system more complicated than it needs to be and that our experience with C++ was minimal. Speed is a big factor on our systems as the number of users that could be using it at any time could potentially be costly so minimizing processing time as much as possible is important. We did this by distributing our system into components efficiently to spread the work across multiple instances.

UML Workbench

We decided to use Visual Paradigm (Community Edition) as our UML workbench. Although we didn't experiment with any other workbenches, this workbench was available at no cost and provided sufficient feature set and provides easy to use tools to form professional UML diagrams. We are using the free community edition, which gives us limited access to code generation tools from diagrams to gain more familiarity on the concept. We also used Gliffy as our UML workbench due to the expiration of our Visual Paradigm subscription. We did not expect this problem.

Design Patterns Descriptions

At this phase in the project we have considered numerous design patterns to improve extensibility and maintainability of our program and to enforce key design principles of the system. The user interface windowing is implemented using a combination of singleton and observer patterns.

The singleton design pattern is used for the window object. It is a creational pattern which ensures that only one instance of a class is ever created and provides a global point of access to the object. This means that only one window will ever be instantiated. It can also be used to implement classes that encapsulate all the data related to the user. Only one instance of the user is needed on the system as only one person can be logged into the system at any given time. The BookManager and UserManager classes use the singleton design pattern to ensure consistency in the data by limiting the instances of each object to 1 by having a protected constructor and a static method that returns the instance.

We chose to use the MVC architectural pattern for the main interface separating the system with a ‘Database Adapter’ as the model and the ‘User Interface’ as the view. This separates the UI from the file system allowing upgradability and maintainability.

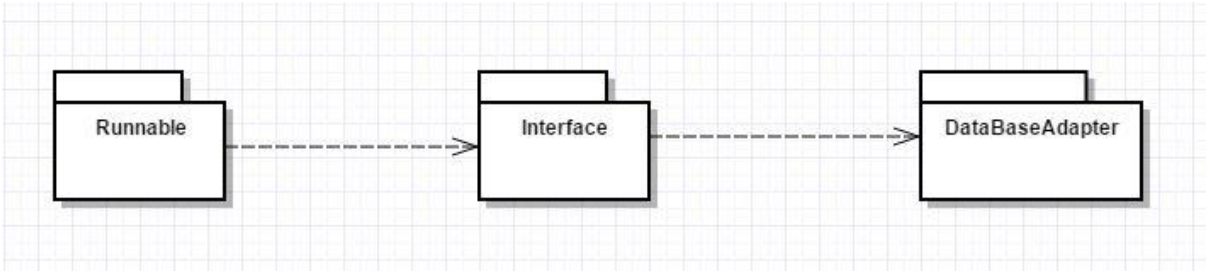
The Factory pattern is used to compartmentalize the creation of the different Concrete classes that implement the LibraryItem interface. It provides one of the best ways to create an object and deals with the problem of creating objects without having to specify the exact class of the object that is created. This creates objects without exposing the client to the logic used by the system and can refer to newly created objects through a common interface.

We used the Decorator pattern when adding LibraryItem (Books and Audio Books) decorating them with an image. We use the Decorator design pattern to create a wrapper class that encapsulates both the image and the Library item. This also implements the LibraryItem interface overriding the methods the image would be required in. This pattern allows for new functionality to be added to an existing object without altering and affecting its structure.

Concurrency support

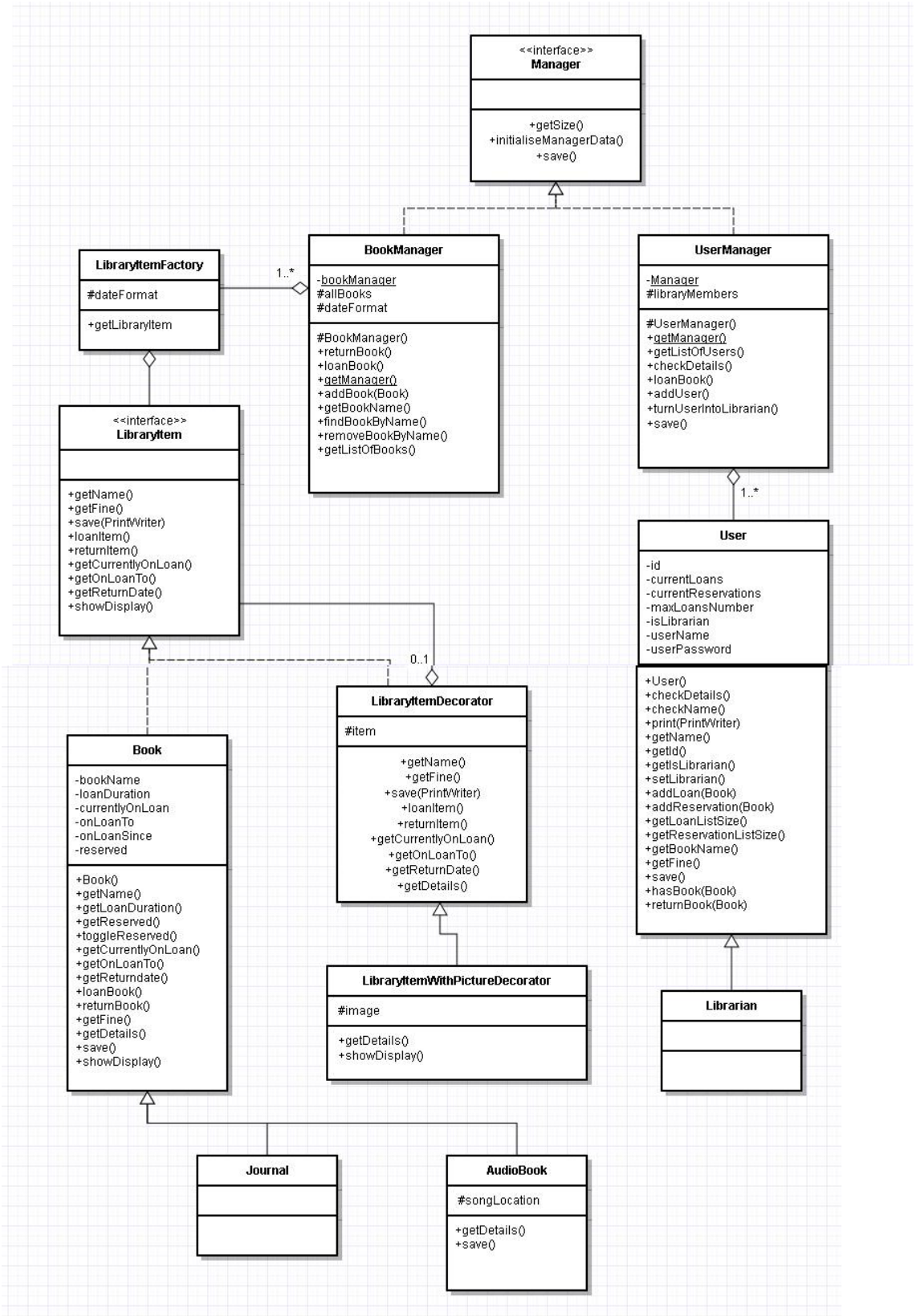
The project contains two manager classes; BookManager and UserManager. In order to achieve efficient searching and saving of the data stored in each file we created an Executor pool (A capped ArrayList of different threads, size set to 5). When save or initialiseData is called it adds another thread to the classes respected pool which is executed asynchronously.

Package Diagram

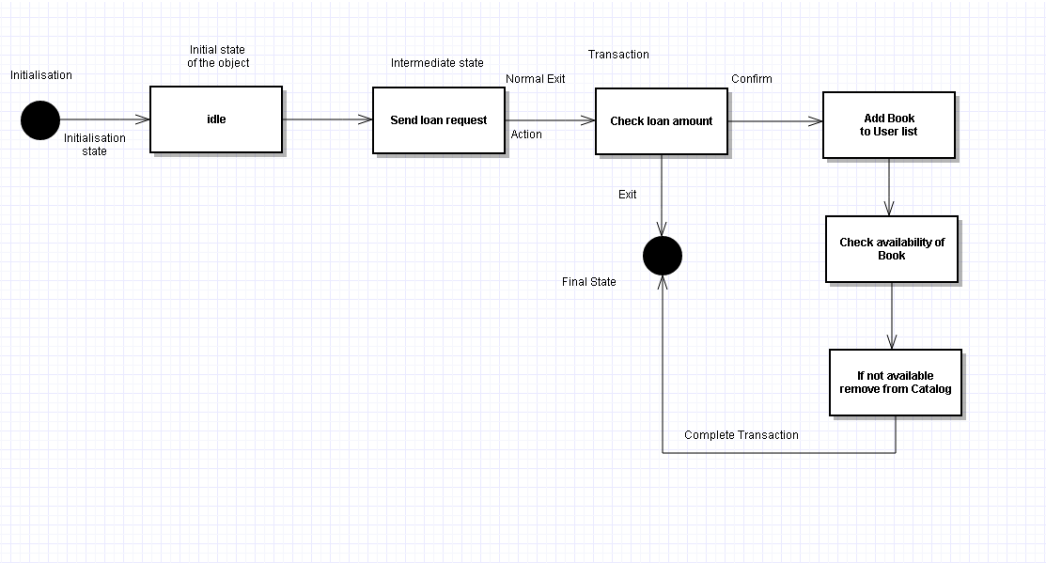

Runnable
Interface
DataBaseAdapter

Design

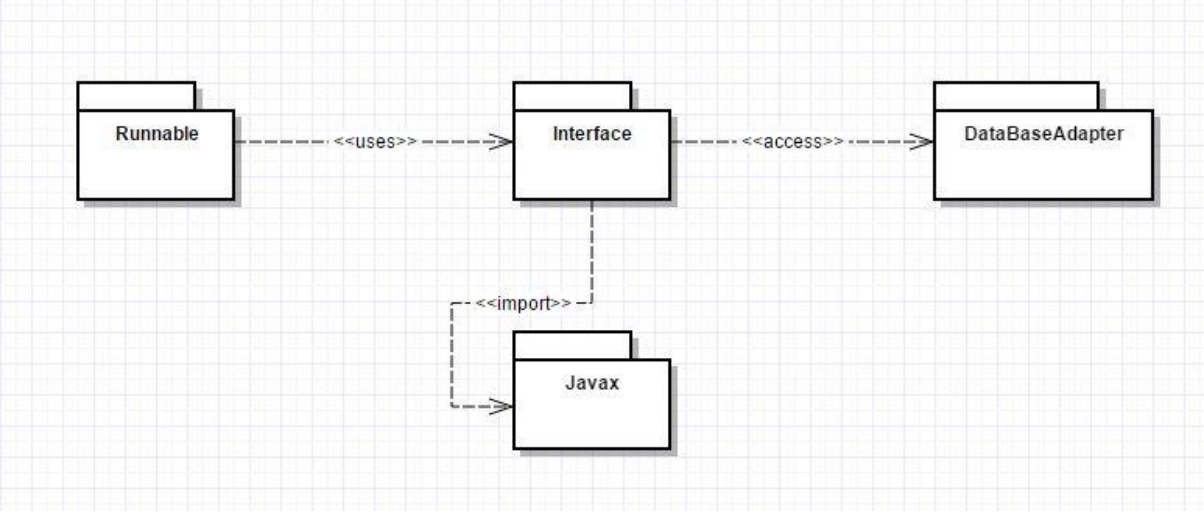
Class Diagram


<<interface>>
Manager
+getSize()
+initialiseManagerData()
+save()
BookManager
-bookManager
#allBooks
#dateFormat
#BookManager()
+returnBook()
+loanBook()
+getManager()
+addBook(Book)
+getBookName()
+findBookByName()
+removeBookByName()
+getListOfBooks()
UserManager
-Manager
#libraryMembers
#UserManager()
+getManager()
+getListOfUsers()
+checkDetails()
+loanBook()
+addUser()
+turnUserIntoLibrarian()
+save()
LibraryItemFactory
#dateFormat
+getLibraryItem
1..*
<<interface>>
LibraryItem
+getName()
+getFine()
+save(PrintWriter)
+loanItem()
+returnItem()
+getCurrentlyOnLoan()
+getOnLoanTo()
+getReturnDate()
+showDisplay()
1..*
User
-id
-currentLoans
-currentReservations
-maxLoansNumber
-isLibrarian
-userName
-userPassword
+User()
+checkDetails()
+checkName()
+print(PrintWriter)
+getName()
+getId()
+getIsLibrarian()
+setLibrarian()
+addLoan(Book)
+addReservation(Book)
+getLoanListSize()
+getReservationListSize()
+getBookName()
+getFine()
+save()
+hasBook(Book)
+returnBook(Book)
0..1
LibraryItemDecorator
#item
+getName()
+getFine()
+save(PrintWriter)
+loanItem()
+returnItem()
+getCurrentlyOnLoan()
+getOnLoanTo()
+getReturnDate()
+getDetails()
Book
-bookName
-loanDuration
-currentlyOnLoan
-onLoanTo
-onLoanSince
-reserved
+Book()
+getName()
+getLoanDuration()
+getReserved()
+toggleReserved()
+getCurrentlyOnLoan()
+getOnLoanTo()
+getReturndate()
+loanBook()
+returnBook()
+getFine()
+getDetails()
+save()
+showDisplay()
LibraryItemWithPictureDecorator
#image
+getDetails()
+showDisplay()
Librarian
Journal
AudioBook
#songLocation
+getDetails()
+save()

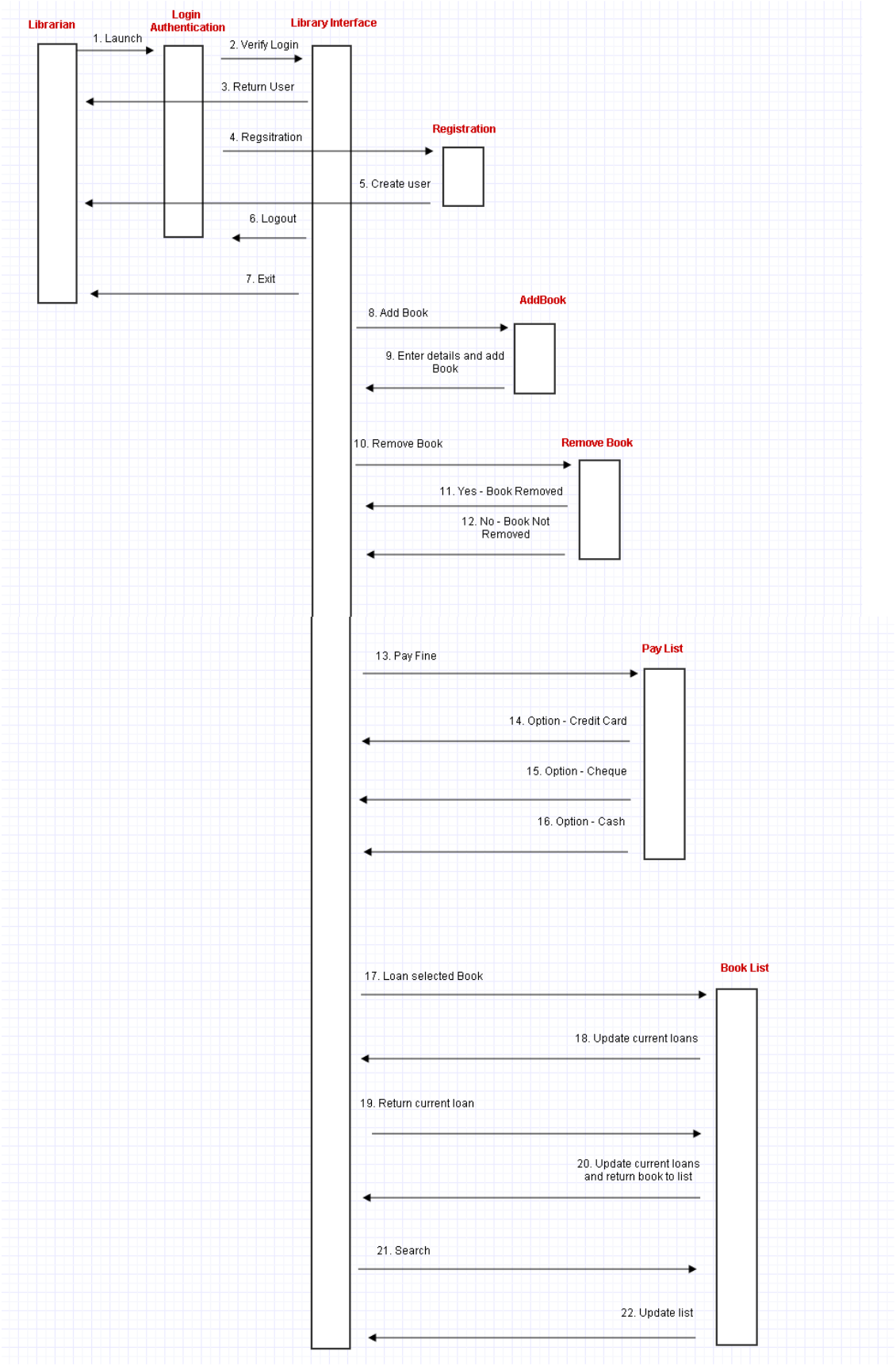
State Diagram

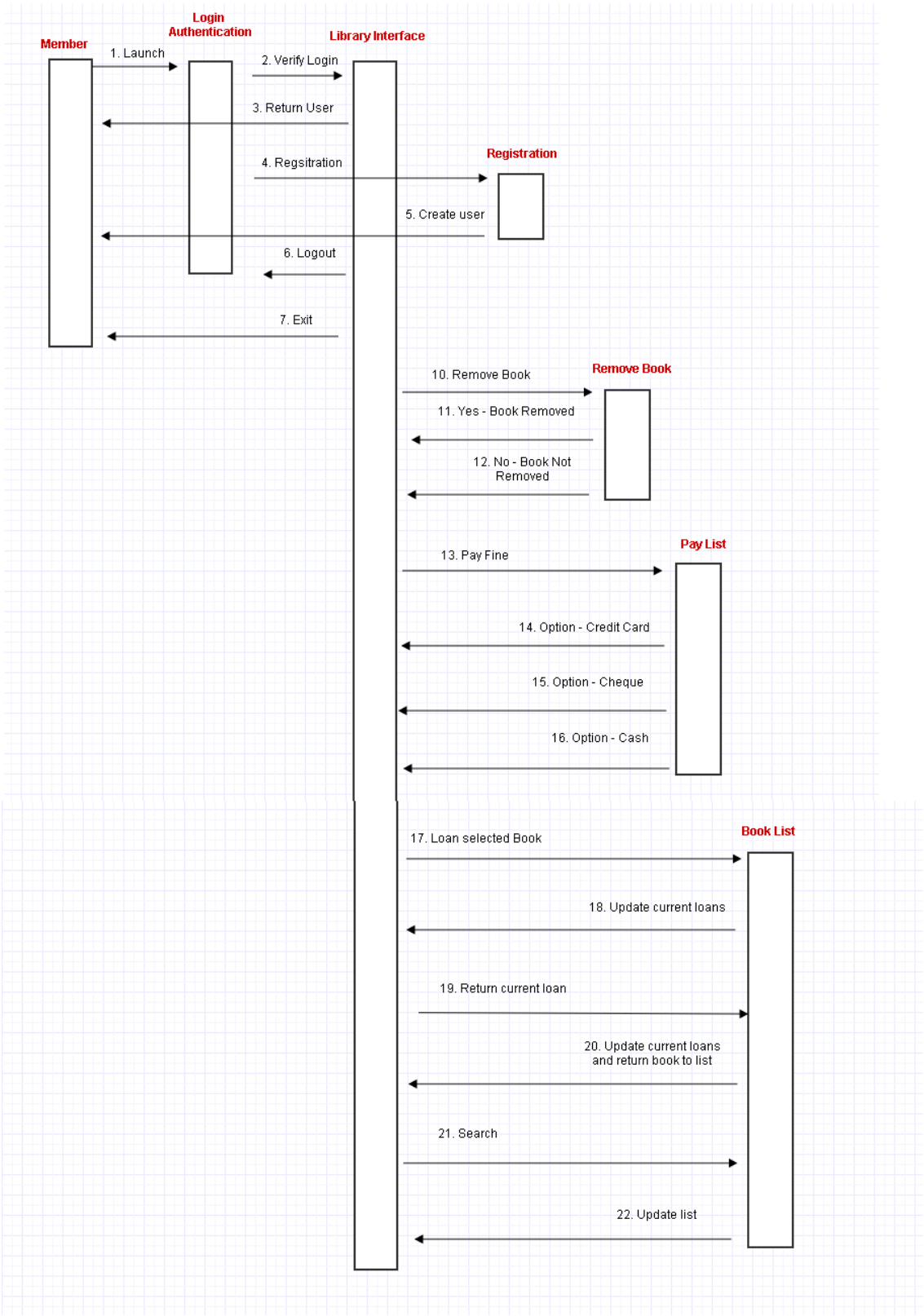


Architectural Diagram



Interaction Diagram




Member
Login Authentication
Library Interface
Registration
Remove Book
Pay List
Book List
1. Launch
2. Verify Login
3. Return User
4. Regsitration
5. Create user
6. Logout
7. Exit
10. Remove Book
11. Yes - Book Removed
12. No - Book Not Removed
13. Pay Fine
14. Option - Credit Card
15. Option - Cheque
16. Option - Cash
17. Loan selected Book
18. Update current loans
19. Return current loan
20. Update current loans and return book to list
21. Search
22. Update list

Critique

Analysis Artefacts and Design – Compare and Contrast

When comparing and contrasting the analysis artefacts to the design artefacts, we noticed that the overall structure of the system remained somewhat similar. Due to our approach during the analysis phase where we went into too much detail on what the UI and finished product would look like and not following the analysis guideline correctly. By diving straight into details of the finished product/system without breaking down the problems into pieces or fragments caused a lot problems and issues for the group. This resulted in Paralysis by analysis which is an anti-pattern, the state of over-thinking a situation so that a decision or action is never taken, in effect paralyzing the outcome. We learned that this highlights the importance of following the analysis process correctly and not to think ahead too much.

What Was Designed

The noun verification technique (And the subsequent filtering of these candidate objects) introduced a variety of classes, However this list did not contain the object names of interfaces required and the concrete classes needed to implement the Factory and Decorator design pattern. We did not account for the need of such classes until implementation and we believe this could have been avoided.

Initially we had quite high coupling between classes, however this was difficult to remove to create more maintainable coupling.

What Should Have Been Designed?

We should have used use case realisation to match the design patterns to where they would be most effective. Afterwards we should have used the noun verification technique to choose the best and most effective nouns necessary to realise the implementation we were designing.

Coupling should have been managed earlier in the development lifecycle, it is of the upmost importance to designate an objects responsibilities and data and we believe that some of the implementation could have been compartmentalised better earlier on.

What Was Implemented?

We have implemented the Singleton, Decorator and Factory design pattern to manage the coupling of Books and AudioBooks, We did this be providing a common interface to effectively group the responsibilities in an easily adaptable way.

Implementation Criticisms

Firstly, we should have compartmentalized GUI elements within the wrapper classes for concrete classes. This would mean we could add books and users to the screen more easily.

It would have been very effective using the composite design pattern to connect Books. This would mean we could have recommended Books based off of a previous loan.

A remote database would have been more effective than text files as this would have allowed concurrency and multiple instances of the program to communicate with each other.

GUI Classes have too many dependencies with each other. A lot of time was taken to remove circular dependencies. One circular connection exists between classes. However, we achieved to remove all dependencies by only using it in a 'destructor' (This is dispose(); for JFrame in Java).

Communication is very poorly executed in the program. It would have been more effective to use the observer pattern coupled with encapsulating communication into message objects. This would have made the Storage more adaptable when changing storage technique.

GUI Communication Controller - This class should have implemented the singleton design pattern. Currently it is possible to create a number of instances of the object. It would have been better to limit the instances so only a single user can log on at any one time.

Additional Information

GitHub

We used GitHub during the implementation phase of our project. GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git.

Microsoft Azure

At the beginning of the implementation phase we attempted to use a remote server hosted by Microsoft Azure to store all the information in an SQL Database.

References/Resources

- Library System Logo adapted from: https://upload.wikimedia.org/wikipedia/commons/0/0f/Frank_Twin_Lakes_Library_System.png
- Analysis paralysis:

  https://en.wikipedia.org/wiki/Analysis_paralysis

- TutorialsPoint UML:

  http://www.tutorialspoint.com/uml/

- TutorialsPoint Design Patterns:

  http://www.tutorialspoint.com/design_pattern/

- Research for narrative (Students going to Uni):

  http://www.theguardian.com/education/2014/aug/15/record-number-students-university-women-deprived-areas

  http://www.telegraph.co.uk/education/universityeducation/11377594/Record-dents-applying-to-university.html

- Visual Paradigm (UML):

  http://www.visual-paradigm.com/

- Gliffy (UML):

  https://www.gliffy.com/

- NetBeans (IDE):

  https://netbeans.org/