***Design Report – Library Management Software***

***Introduction***

I have created a console based Library Management software which has got all the functionality of managing the library resources and it is also capable of processing the members and admin requests such as borrow item, return item, search item, access member details, remove and add members etc. I designed this software in a way in which admin and member have an account and they both are connected with the Library Management System class. Library Management system is responsible to handle all the requests from the members and admin.

***Design Process***

The concept of **Abstraction** in OOP helped me to come up with roles and responsibilities what librarian/admin and member class must have. So, I consider a scenario in which librarian is sitting in the library and he has got the record of all the registered members. Every member has a unique id which is helpful to distinguish two members with same names. After thinking about the roles and responsibilities of both admin and member, the list of responsibilities of both member and admin I come up with are shown below –

Diagram

Description automatically generated

*Figure 1 – Responsibilities of Member and Librarian class*

My next step was to think about the how I want my software to work and what other classes do I need in my software to achieve the desired functionality. I created a Library Database class which has a list of all the registered members and library resources. After Library Database has been created, I need a class which will manage my Library Database and handle all the requests from the members and admin. To manage the library database and handling the requests from the user and admin, I created a Library Management System Class. The whole software is centred around the library management system class. After that I created an abstract class called Library Resource which is the parent class of my library resources such as laptop, book and newspaper. I found similar roles and responsibilities in member and admin, so to avoid the code repetition, I created an Account class which is basically the parent class of my librarian and member class.

***How Library Management System Works?***

Below figures are representing how library management system is handling and responding to the requests of the member and librarian.

Diagram

Description automatically generated

*Figure 2 - Figure is showing how librarian can search items*

Diagram

Description automatically generated

*Figure 3 – Figure is showing how member can borrow items.*

***Sequence Diagram***

Diagram

Description automatically generated

*Figure 4 – Sequence diagram showing how member can search items*

***How I designed this Software using Object Oriented Principles?***

*Abstraction*

Abstraction is taking the real-world objects and mapping it into software constructs. It basically helps us to identify classifications, roles, responsibilities and collaborations. As we know that, the objects know and do things and the abstraction helps us to design the objects by figuring out what things that object should know which we declare in the form of fields in the class and the second thing is its functionality which we declare in the form of methods. While designing this software, abstraction helps me figuring out the roles and responsibilities of the classes. For e.g. what a librarian must know things and what things it must be able to do. In figure 4 , it is clearly seen that the things that librarian/admin must know his name, username and password and the things that librarian can do are add members, remove members, add resources , remove existing resources, can access member details etc. In conclusion, the idea of abstraction helps me to figure out how to design the objects from scratch.

Diagram, schematic

Description automatically generated

*Figure 5 – Librarian Class*

*Encapsulation-*

The general meaning of encapsulation is hiding the data and functionality into a single unit called class. In other words, we can say that encapsulation is a type of abstraction that ensures information is hidden and well contained. For e.g. as you can see in figure 5, my librarian class has got mutliple functionalities and it also has its information about its name, username and password. So, this is an example of encapsulation as fields declared within the librarian are not shared with any other class. That would be an example of poor encapsulation if the fields declared within the librarian class can easily be manipulated by some other class. An object is said to be highly cohesive if methods and data are closely related.

Diagram, schematic

Description automatically generated

*Figure 6 - UML representing how each object/class encapsulates its data and functionality*

*Inheritance and Polymorphism*

Inheritance is the process of creating a new class called Derived Class or Child Class, from an existing class called Base Class or Parent Class. My Book, Laptop and Newspaper class are all derived from the Library Resource class and they are inheriting all the fields declared within the Library Resource class for e.g. they all have Name and Status field and property declared within the Library Resource class. So, inheritance is basically used if the objects follow “is a” relationship e.g. “ A Book is a Library Resource”, “A Laptop is a Library Resource” etc. Polymorphism means ability to take many forms or reference different specialisations using single date type. As you can see in figure 5, the Library resource is an abstract class and the laptop and book objects are derived from the Library resource class. This thing represents polymorphism which means a reference variable can reference objects of classes that are derived from the variable’s class.

|  |
| --- |
| Libraryresource resource1 = new Laptop(name,modelNumber,os,id); Libraryresource resource1 = new Book(name,author,isbn,format); |

Here, you can see resource1 variable can be used to reference a Laptop object and Book object. This basically represents the idea of polymorphism in which a reference variable can store the reference of objects of types different from its own as long as these types are subclasses of its type.

Diagram

Description automatically generated

*Figure 7 – UML representing Book, Laptop and Newspaper are Library Resources*

***Screenshot of the program is Working***

Table

Description automatically generated

Text, letter

Description automatically generated

***ScreenShot of File Saving is Working***

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

***How Flexible Saving Feature Works in my Software?***

I have added the flexible saving feature in my software. The way how flexible saving works is when librarian adds a new member or new resource into the library database, he can also save the details of the member and resource, he added in the database. Next time, when he start the software, by pressing just one key, he can load all the details of the member and resources, he added into the database last time. Without having flexbile saving feature, the software is not capable to run in real world.

# ***Required Roles***

Table 1: Librarian or Admin details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Must be able to add new members in the library database | AddNewMember(Member m) | Take new member object as an argument to add in the database |
| Must be able to remove existing members form the library database | RemoveExistingMember(string memberID): string | Takes existing memberID to remove from the database |
| Must be able to add new library resource in the library database | AddNewItem(LibraryResource resource) | Take new library resource object as an argument to add in the database |
| Must be able to remove the resource from the library database | RemoveItem(string name) : string | Take item name as an argument to remove from the database |
| Must be able to search any resource existing in the library database | SearchItem(string name) : string | Take item name as an argument to search an in the library database |
| Librarian must be able to update his details | UpdateDetails(string name, string username) | Takes new username and new name as an argument to update details |
| Must be able to see his details | MyDetails() : string |  |
| Must be able to access any registered member details | AccessMemberDetails(int memberId) :string | Takes member Id as an argument to access member details |

Table 2: Member details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Member must be able to borrow any item from the library | BorrowItem(string itemName , int memberId) : string |  |
| It must be able to return any item to the library | ReturnItem(string itemName, int memberID) : string |  |
| It must be able to see the number of items it has borrowed from the library | NumberOfItemsOnLoan() : string |  |
| Member must be able to update his details | UpdateDetails(string name, string username, string address, string phoneNumber) |  |
| Must be able to search any resource existing in the library database | SearchItem(string name) : string |  |
| Must be able to see his details | ShowMyDetails(): string |  |
| Must be able to reset his password | ResetPassword(string password) |  |

Table 3 Library Managemet System details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Must be able to add new members in the library database | AddMember(Member m) |  |
| Must be able to remove existing members form the library database | RemoveMember(int member ID):string |  |
| Must be able to add new library resource in the library database | AddNewResource(LibraryResource resource) |  |
| Must be able to remove the resource from the library database | RemoveResource(string name) : string |  |
| Must be able to retrieve any item details | ShowItemDetails(string name) : string |  |
| Must be able to check is member registerd or not | VerifyMember(int memberId):bool |  |
| Must be able to retrieve any member details | ShowMemberDetails(int memberId) : string |  |
| Must be able to issue any item to the member | IssueItem(string itemName, int memberID) |  |
| Must be able to put back item to the library database | ReturnItem(string name, int member Id) |  |

Table 4 Library Database details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Must be able to search members in the library database | LocateMember(Member m) |  |
| Must be able to search resources in the library database | LocateResource(Resource r) |  |

Table 5 Library Resource details

Library Resource is an abstract class!

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| Must be able to display its details by itself | DisplayDetails() : string <<abstract>> |  |
| Must be able to update its status by itself | UpdateStatus(Status s) <<abstract>> |  |

Table 6 Format details

|  |  |
| --- | --- |
| Value | Notes |
| Audiobook |  |
| Paperback |  |
| Hardcover |  |
| Ebook |  |

Table 7 Status details

|  |  |
| --- | --- |
| Value | Notes |
| Available |  |
| Loaned |  |
| Lost |  |
| Reserved |  |

***Conclusion***

I have applied all of the four Object Oriented Principles and good object oriented design principles as well ( high cohesion and low coupling) while designing the software. Overall, it was my first experience to design something from scratch. It was fun and I really enjoyed the process.