UNIVERSITY COLLEGE OF ENGINEERING NAGERCOIL

(ANNA UNIVERSITY CONSTITUENT COLLEGE)

KONAM, NAGERCOIL – 629 004

RECORD NOTE BOOK

CCS356-OBJECT ORIENTED SOFTWARE ENGINEERING

REGISTER NO :

NAME :

YEAR/SEMESTER :

DEPARTMENT :

UNIVERSITY COLLEGE OF ENGINEERINGNAGERCOIL

(ANNA UNIVERSITY CONSTITUENT COLLEGE)

KONAM, NAGERCOIL – 629 004

Register No:

Certified that, this is the bonafide record of work done by ***Mr./Ms.****…… of VI Semester in Computer Science and Engineering of this college, in the* ***CCS356 – OBJECT ORIENTED SOFTWARE ENGINEERING*** *during academic year 2023-2024 in partial fulfillment of the requirements ofthe B.E Degree course of the Anna University Chennai.*

Staff-in-charge **Head of the Department**

This record is submitted for the University Practical Examination held on .....

Internal Examiner **External Examiner**

INDEX

Exp No	Date	Title	Page	Sign
1.		Identify a software system that needs to be developed		
2.		Document the Software Requirements Specification (SRS) for the Library Management system		
3.		Identify the use cases and the develop the Use Case model		
4.		Identify the conceptual classes and develop a Domain Model and also derive a Class diagram for the Library Management System		
5.		Using the identified scenarios, find the interaction between objects and represent them using UML Sequence and Collaboration diagram		
6.		Draw relevant State Chart and Activity Diagram for the Library Management System		
7.		Implement the system as per the detailed design		
8.		Test the software system for all the scenarios identified as per the Use Case diagram		
9.		Improve the reusability and maintainability of the software system by applying appropriate design patterns		

EX NO:1 DATE:	To identify a software system that needs to be developed - Library Management System

Aim:

To identify a software system that needs to be developed – Library Management System.

Introduction:

In today's information age, libraries play a crucial role in facilitating access to knowledge and resources. A Library Management System is a comprehensive software solution designed to streamline and automate the various functions and processes within a library to efficiently manage their resources while enhancing user satisfaction. This introduction highlights the significance of such a system and outlines its key features.

Features of Library Management System:

1. **Cataloging and Classification:**

   Allows librarians to catalog and classify library materials according to standard classification systems such as Dewey Decimal Classification or Library of Congress Classification. It enables efficient organization and retrieval of library resources.

2. **Circulation Management:**

   Facilitates the borrowing and returning of library items by patrons. This includes features such as check-in, check-out, renewal of items, holds, reservations, and managing fines for late returns.

3. **User Management:**

   Enables librarians to manage patron accounts, including registration, authentication, and defining user privileges. It also tracks user activity and preferences for personalized services.

4. **Inventory Management:**

   Tracks the acquisition, disposition, and availability of library materials. It includes features like real-time updates on stock levels, item locations, and status, as well as managing lost or damaged items.

5. **Online Public Access Catalog (OPAC):**

Provides users with a web-based interface to search, browse, and access library resources remotely. It typically includes advanced search options, filters, and integration with other library services.

6. **Resource Management:**

Facilitates the procurement, selection, and evaluation of library materials. It includes features for managing subscriptions, acquisitions, donations, and deselection of outdated materials.

7. **Reporting and Analytics:**

Generates reports and insights on library usage, circulation patterns, popular items, and other key metrics. It helps librarians make data-driven decisions regarding collection development, resource allocation, and user services.

8. **Interlibrary Loan:**

Facilitates borrowing and lending of materials between libraries to expand access to resources beyond the library's own collection. It includes features for managing requests, tracking loaned items, and coordinating with other libraries.

9. **Integration and Customization:**

Integrates with other library systems and third-party services, such as electronic resource management platforms, authentication systems, and discovery tools. It also allows for customization to adapt to the unique needs and workflows of different libraries.

10. **Security and Access Control:**

Implements security measures to protect sensitive library data and resources. It includes features such as role-based access control, encryption, and audit trails to ensure data integrity and compliance with privacy regulations.

Result:

Thus, the software system that is need to be developed for Library management system was executed successfully.

Ex. No:02	Document the Software Requirements System (SRS) for Library Management System

AIM:

To document the Software Requirements System (SRS) for Library Management System.

PROBLEM STATEMENT:

Conventional libraries are having difficulty integrating various formats, including multimedia and e-resources, because of outdated management systems. Inefficient cataloguing, resource tracking bottlenecks, and a lack of analytics tools hinder librarians from optimizing collections and improving user experiences. To close the gap, librariesrequire a modern library management system with an intuitive interface, effective cataloguing, and analytics capabilities to resurrect libraries as vibrant centers of knowledge and community involvement in the digital era.

1. INTRODUCTION:

1.1. PURPOSE:

The main objective of this document is to illustrate the requirements of the project Library Management system. The document gives the detailed description of the both functional and non-Functional requirements proposed by the client. The document is developed after a number of consultations with the client and considering the complete requirement specifications of the given Project. The final product of the team will be meeting the requirements of this document.

1.2 PRODUCT SCOPE:

The product is designed for both the Users and Library Admin. It will be a helpfulproduct in a very effective way as it will reduce the tiresome workload from both Users and Library Admin.

2. OVERALL DESCRIPTION:

2.1 PRODUCT PERSPECTIVE:

The proposed Library Management System is an on-line Library ManagementSystem. This System will provide a search functionality to facilitate the search ofresources. This search will be based on various categories viz. book name or the ISBN. Also Advanced Search feature is provided in order to search various categoriessimultaneously. Further the library staff personnel can add/update/remove the resourcesand the resource users from the system.

2.2 PRODUCT FUNCTIONALITY

Functionality of this system is:

LIBRARIAN:

• A librarian can issue a book to the student

• Can view the different categories of books available in the Library

• Can view the List of books available in each category

• Can take the book returned from students

• Add books and their information of the books to the database

• Edit the information of the existing books.

• Can check the report of the issued Books.

• Can access all the accounts of the students.

USERS:

• Can view the different categories of books available in the Library.

• Can view the List of books available in each category

• Can own an account in the library

• Can view the books issued to him

• Can put a request for a new book

• Can view the history of books issued to him previously

• Can search for a particular book

2.3 USERS CLASSES AND CHARACTERISTICS:

There are various kinds of users for the product. Usually, web products are visited by various users for different reasons. The users include:

- Students who will be using the above features by accessing the Library online.
- Librarian who will be acting as the controller and he will have all the privileges ofan administrator.

2.4 OPERATING ENVIRONMENT

This system will be operated on any computer with the following minimum specifications:

• Windows XP Service Pack-3 OR higher versions of windows.

• Computer hardware should be build on INTEL chipset.

• Minimum free RAM of 128 MB.

• Internet connectivity required.

2.5 DESIGN AND IMPLEMENTATION CONSTRAINTS

The design & implementation constraints are:

• The system database used should be an open-source technology.

• The system should be implemented in Java Technology

• The downtime of the system should be less than 10 min.

• RAM usage should not exceed 1024MB

• This system software size should not exceed 1GB.

2.6 USER DOCUMENTATION

The user manual and help will be available online and can be accessed any time by any user. The manual will be updated on a regular basis so as to keep the contents up todate.

2.7 ASSUMPTIONS AND DEPENDENCIES

It is assumed that the optimum internet connectivity speed will be more than 512Kbps.If the bandwidth is less than this then the transaction completion will take more time to processed and to be complete.

3. Specific Requirements:

3.1 EXTERNAL INTERFACE REQUIREMENTS

3.1.1 User Interface

Various GUI elements like forms, images and standard buttons will be included inthe User Interface.

3.1.2 Software Interface

Software will work on Windows OS. The Database used will be an open-sourcedatabase like MySql. And the system will run on Java Virtual Machine.

3.1.3 Communication Interface

This system will require web browser, internet connection which supports HTTPand server.

3.2 FUNCTIONAL REQUIREMENTS

• Issuing the books

• Return the books

• Search the stock of books

• Record of books issued and returned

• Calculate fine if required

4. OTHER NON-FUNCTIONAL REQUIREMENTS

4.1 PERFORMANCE REQUIREMENTS

1. Login/Registration will not take more than 10 seconds.
2. Any financial transactions will not take more than 15 seconds.

4.2 SAFETY AND SECURITY REQUIREMENTS

1. Database will be secured by authentication process.
2. Unauthorized access will be avoided and will be tracked.
3. Database backup will be maintained.

4.3 SOFTWARE QUALITY ATTRIBUTES

1. System will be reliable.
2. System can be maintained easily.

RESULT:

Thus, the Software Requirements Specification (SRS) for the Library Management System was documented successfully.

Ex. No:03	Identify the use cases and develop the Use Case model

AIM:

To identify use cases and develop the use case model for library management system.

INTRODUCTION OF USE CASE DIAGRAM:

Use case diagrams are graphical representations that depict the interactions between users and system to achieve specific goals or tasks. They are an essential tool in software development for capturing and communicating system requirements in a clear and visual manner.

Purpose:

- **Requirement Analysis:** Use case diagrams help stakeholders, including developersand clients, understand the system's functionality and behavior from a user'sperspective. They provide a high-level overview of the system's features andinteractions.
- **Communications:** Use case diagrams facilitate communication between stakeholders by providing a common visual language to discuss system requirements and functionalities. They help ensure that everyone involved in the project has a shared understanding of the system's scope and objectives.
- **Design Validation:** Use case diagrams assist in validating the system design by identifying potential gaps, inconsistencies, or missing functionalities. They allow stakeholders to review and refine the system requirements before proceeding with the implementation phase.

Components:

- Actors: Actors represent external entities that interact with the system to accomplish specific goals. Actors are depicted as stick figures and are connected to use cases by association relationships.
- **Use Cases:** Use cases represent the specific functionalities or tasks that the system performs to fulfil user goals. Each use case describes a sequence of interactions between the system and actors to achieve a particular outcome. Use cases are depicted as ovals within the diagram.
- **Relationships:** Relationships between actors and use cases are depicted using association lines. These relationships indicate the interactions between actors and the

system to accomplish various tasks. There are different types of relationships, including include, extend, and generalization.

Benefits:

- **Clarity:** Use case diagrams provide a visual representation of system requirements, making it easier for stakeholders to understand and visualize the system's behavior and functionality.
- **Requirements Prioritization:** Use case diagrams allow stakeholders to prioritize system requirements based on user goals and objectives. By identifying critical use cases, stakeholders can focus on implementing essential functionalities first.
- **Change Management:** Use case diagrams facilitate change management by providing a structured framework for evaluating and incorporating new requirements or modifications to existing functionalities. They help assess the impact of changes on the overall system.

Use cases diagram for Library Management System:

In a Library Management System, the use case diagram could include actors like "Borrower", "Librarian". Some potential use cases like "make reservation","remove reservation" , " lend item" , " return of idea" , "Add title" , " remove title" , "add item" ," add browser" and "remove browser"

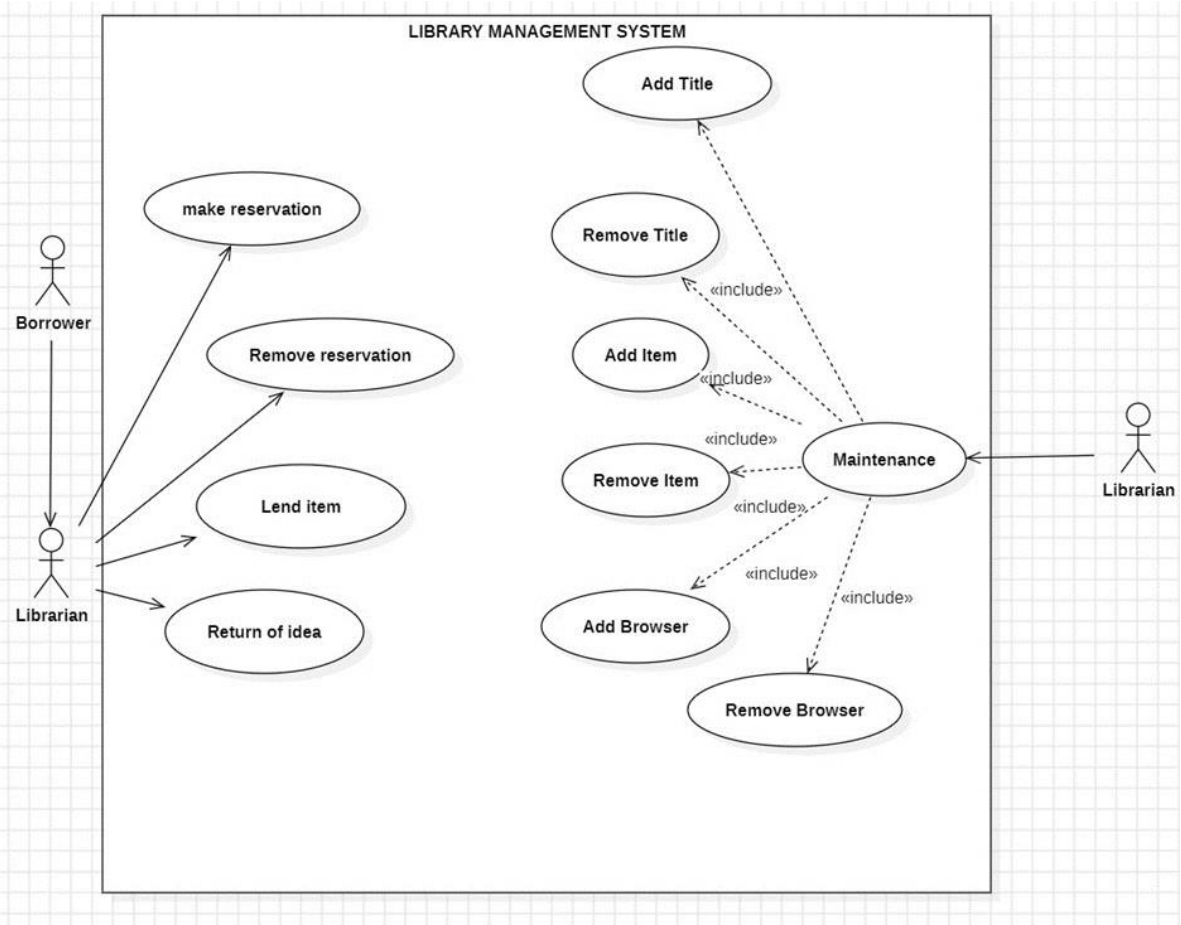
Actors:

- Borrower
- Librarian

Use cases:

- Make Reservation
- Remove Reservation
- Lend Item
- Return of idea
- Add Title
- Remove Title
- Add Item
- Add Browser
- Remove Browser

Use Case Diagram:



Borrower:

Borrower is an individual or entity that temporarily receives or uses an item, resource, or service that belongs to someone else. Borrowers typically have a specific period during which they can use the item, after which they are expected to return it to the owner.

Borrowers may be required to adhere to certain terms and conditions, such as paying a fee for borrowing or returning the item in its original condition.

Librarian:

Represents a staff member responsible for library operations with attributes like name and role.

Make Reservation:

Enables users to reserve or book a service, resource, or item.

Remove Reservation:

Allows users to cancel or remove a reservation.

Lend Item:

Allows users to lend an item to someone else.

Return Item:

Enables users to return a borrowed or lent item to its original owner or location.

Add Title:

Allows users to add a new title to a collection or library.

Remove Title:

Enables users to remove a title from a collection or library.

Add Item:

Permits users to add a new item to a list or inventory.

Remove Item:

Allows users to remove an item from a list or inventory.

Add Browser:

Let’s users add a new web browser to their system or application.

Remove Browser:

Allows users to remove a web browser from their system or application.

Result:

Thus, the Use Case Model was identified and developed successfully.

EX. NO:4	Identify the Conceptual classes and develop a domain model and also derive a Class Diagram

AIM:

To identify the conceptual classes and develop a Domain Model and also derive aClass diagram from the Library Management System.

CONCEPTUAL CLASSES:

In Library Management System, several conceptual classes can be identified torepresent the various entities and functionalities of the system. Here are some key conceptual classes for the Library Management System:

1. **Book:**

Represents a library book with attributes like title, author, and availability status.

2. **Librarian:**

Represents a staff member responsible for library operations with attributes like nameand role.

3. **Transaction:**

Represents borrowing or returning transactions between members and the library.

4. **MemberRecord:**

Represents a member's information and borrowing history.

5. **Bill:**

Represents bills generated for fines or charges incurred by members.

6. **Faculty:**

Represents academic staff with attributes like department and borrowing privileges.

7. **Student:**

Represents students with attributes like program and borrowing privileges.

8. **Journals:**

Represents academic journals with attributes like title and publisher.

9. StudyBooks:

Represents textbooks or reference books with attributes like edition and subject.

10. Magazines:

Represents magazines or periodicals with attributes like publisher and publicationfrequency.

DOMAIN MODEL FOR LIBRARY MANAGEMENT SYSTEM:

Book:

❖ **Attributes:**

- bookId
- author
- name
- price
- rackNo
- status
- edition
- dateOfPurchase

❖ **Operations:**

- DisplayBookDetails()
- updateStatus()

Librarian:

❖ **Attributes:**

- name
- password

❖ **Operations:**

- searchBook()
- verifyMember()
- issueBook()

- calculateFine()
- createBill()
- returnBook()

Transaction:

❖ **Attributes:**

- transId
- memberId
- bookId
- dateOfIssue
- dueDate

❖ **Operations:**

- createTransaction()
- deleteTransaction()
- retrieveTransaction()

MemberRecord:

❖ **Attributes:**

- memberId
- type
- dateOfMembership
- noBookIssued
- maxBookLimit
- name
- address
- phoneNo
- billCreate()

❖ **Operations:**

- retrieveMember()
- increaseBookIssued()
- decreaseBookIssued()
- payBill()

Bill:

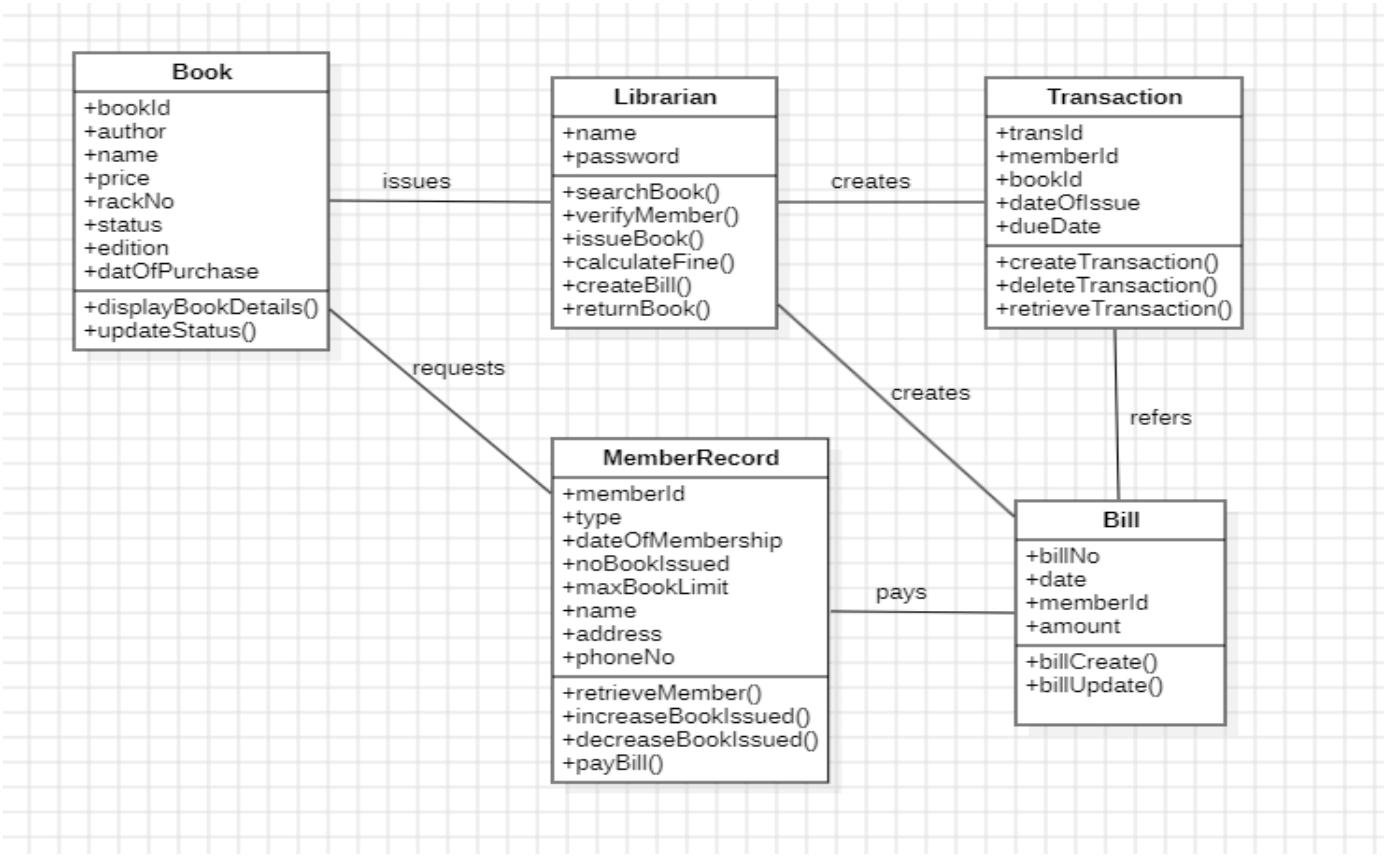
❖ **Attributes:**

- billNo
- date
- memberId
- amount

❖ **Operations:**

- billCreate()
- billUpdate()

CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM:



RESULT:

Thus, the conceptual classes, Domain Model and Class Diagram for LibraryManagement system was identified and developed successfully.

Ex.No:05	Using the identified scenarios, find the interaction between objects and represent them using UML Sequence and Collaboration Diagram

AIM:

To find the interaction between objects and represent them using UML Sequence and Collaboration Diagram.

INTRODUCTION OF INTERACTION DIAGRAM:

An interaction diagram is a visual representation that illustrates how objects in a system interact and communicate with each other over time, typically used in software design and modeling to depict the flow of messages or method calls between objects duringthe execution of a use case or scenario.

There are two main types of interaction diagrams commonly used in software development:

- Sequence Diagram
- Communication (or) Collaboration Diagram

SEQUENCE DIAGRAM:

A sequence diagram is a type of interaction diagram in UML (Unified Modeling Language) that shows how objects interact in a particular scenario or use case over time. Itillustrates the sequence of messages exchanged between objects, along with their lifelines,to depict the flow of control and communication in a system or software application.

A sequence diagram is a type of interaction diagram that illustrates the flow of messages or method calls between objects in a system over time. Its main components include:

Objects:

Represented by rectangles with the object name at the top, indicating the entitiesor instances participating in the interaction.

Lifelines:

Vertical dashed lines extending from each object's rectangle, representing the object'sexistence and lifespan during the interaction.

Messages:

Arrows indicating communication or interaction between objects, with labelsspecifying the type of message. There are different types of messages,

- Synchronous Message
- Asynchronous Message
- Return Message

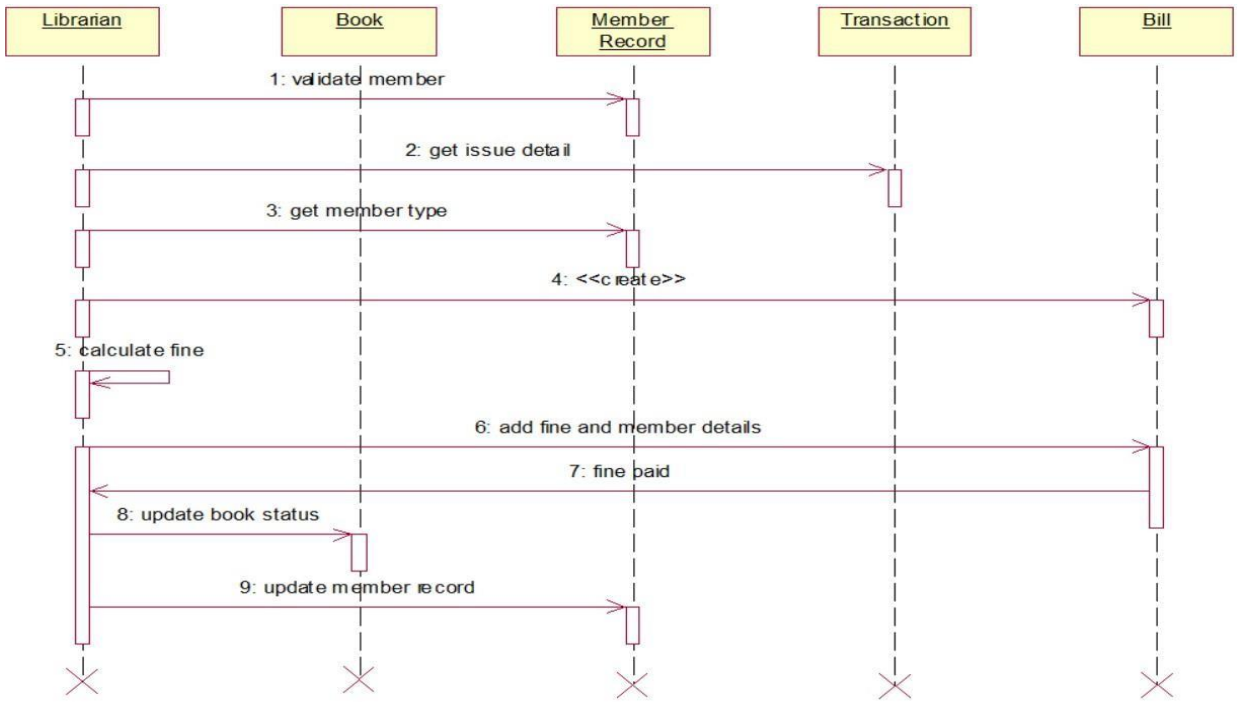
Activation Boxes:

Horizontal bars on lifelines representing the period when an object is activelyprocessing or executing a message.

Focus of Control:

Indicates the flow of control between objects during the interaction, showing whichobject is actively performing actions or responding to messages at any given time.

Sequence Diagram for Library Management System:



COLLABORATION DIAGRAM:

A collaboration diagram, also known as a communication diagram in UML (Unified Modeling Language), is a type of interaction diagram that visualizes the interactions and relationships between objects or actors in a system or software application. It emphasizes the communication flow and messages exchanged between objects to achieve specific functionalities or scenarios within the system.

Collaboration diagrams are used to visualize and model the interactions,collaborations, and dependencies between objects in a system, helping to understand the dynamic behavior and communication patterns within the software or system architecture. A collaboration diagram is an interaction diagram in UML (Unified Modeling Language) that illustrates the interactions and relationships between objects in a system or software application. Its main components include:

Objects:

Represented by rectangles with the object's name, showing the entities or instances participating in the collaboration.

Links:

Lines connecting objects to represent associations or relationships between them,indicating how objects interact or communicate.

Messages:

Arrows indicating the flow of communication or interactions between objects, withlabels specifying the type of message (e.g., synchronous, asynchronous).

Roles:

Descriptions or labels attached to links or arrows indicating the role or purpose of theinteraction between objects.

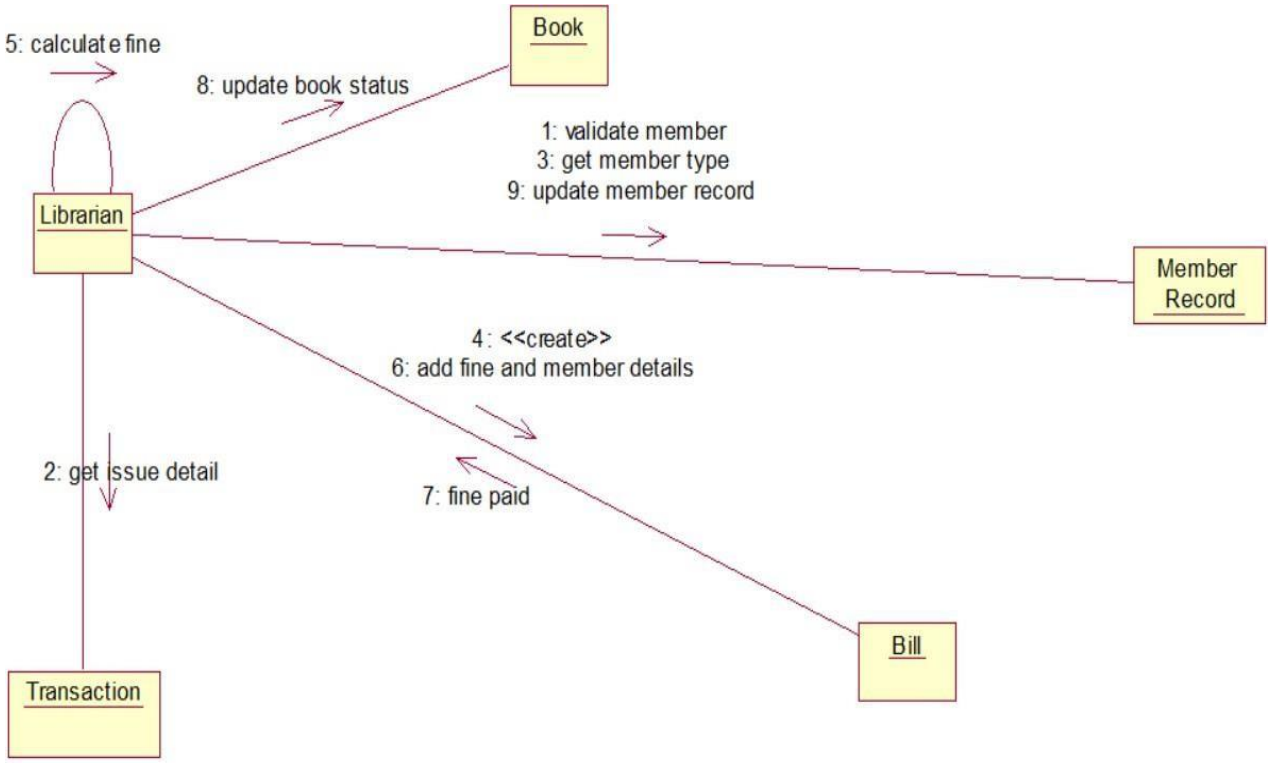
Multiplicity:

Optional notation on links or arrows indicating the cardinality or multiplicity of the relationship between objects (e.g., one-to-one, one-to-many).

Notes:

Additional information or comments placed in rectangular boxes attached to relevantparts of the diagram, providing explanations or details about the collaboration.

Collaboration Diagram for Library Management System:



RESULT:

Thus, the interaction between objects and represent them using UML Sequence andCollaboration Diagrams were identified successfully.

Ex. No:06	Draw relevant State Chart and Activity Diagrams for the Library Management System

AIM:

To draw the relevant State chart diagram and Activity diagram for the Library Management System.

INTRODUCTION TO STATE CHART DIAGRAM:

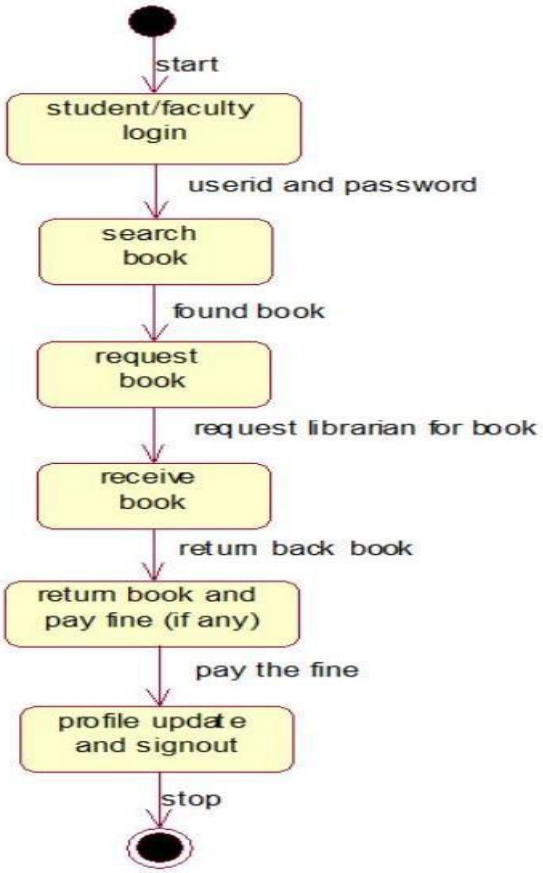
A state chart, also known as a state machine diagram, is a visual representation used in software engineering to model the behavior of a system or object over time. It depicts the various states that an object can be in and the transitions between these states based onevents or conditions. State charts are particularly useful for modeling complex systems with multiple states and defining the sequence of actions or operations that occur as the system transitions between states.

COMPONENTS:

- **States:** States represent the different conditions or situations that an object or system can be in at any given time. Each state is typically depicted as a rounded rectangle with a name inside, describing the state (e.g., "Idle," "Active," "Stopped").
- **Transitions:** Transitions indicate the movement or change of state that occurs in response to events or conditions. They are represented by arrows between states, with labels describing the triggering event or condition that causes the transition (e.g., "Start," "Stop," "Timeout").
- **Events:** Events are external stimuli or triggers that initiate state transitions. They can be user actions, system inputs, or timed events that cause the system to changefrom one state to another (e.g., "Button Clicked," "Data Received," "Timer Expired").
- **Actions:** Actions represent the operations or behaviors that occur when a state transition takes place. They are often associated with transitions and can includetasks, computations, or changes in system variables (e.g., "Initialize System," "Update Data," "Send Notification").

- **Initial State:** The initial state is the starting point of the state chart diagram, indicating the initial condition or state of the system when it first begins execution.It is typically denoted by a filled circle connected to the initial state with a transitionarrow.
- **Final State:** The final state represents the end or termination point of the state chart,indicating that the system has completed its process or reached a final state where no further transitions occur. It is usually denoted by a filled circle with a dot insideor a rounded rectangle with the word "End."
- **Composite State:** Composite states are hierarchical states that contain sub-states within them. They allow for the organization and nesting of states to represent complex behavior more effectively. Composite states are depicted as a larger state box containing nested states.

STATE CHART DIAGRAM:



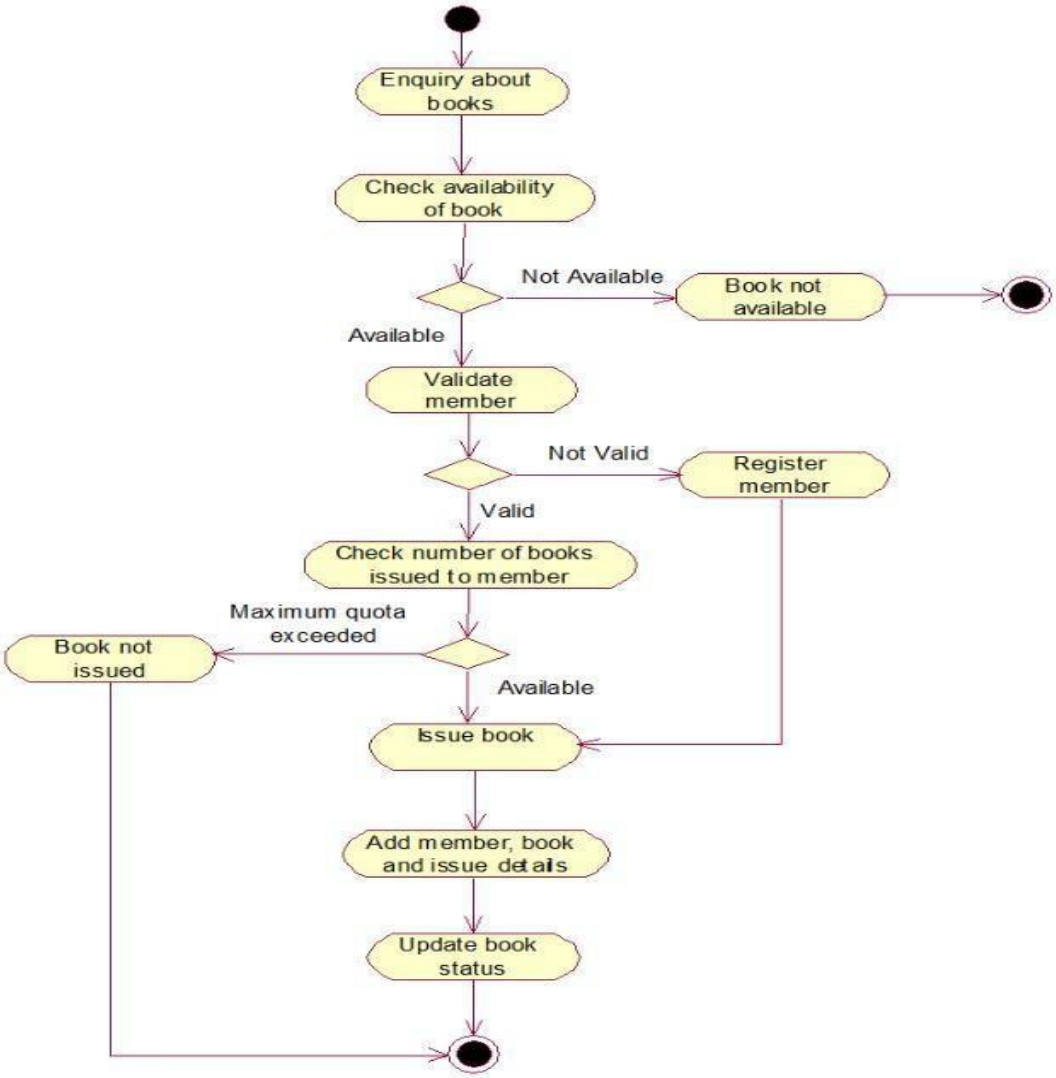
INTRODUCTION TO ACTIVITY DIAGRAM:

Activity Diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We can depict both sequential processing and concurrent processing of activities using an activity diagram i.e an activity diagram focuses on the condition of flow and the sequence in which it happens.

COMPONENTS:

- **Initial State:** The initial state is the starting point of the state chart diagram, indicating the initial condition or state of the system when it first begins execution. It is typically denoted by a solid circle or a filled circle.
- **Action or Activity State:** An activity represents execution of an action on objects or by objects. It is represented by a rectangular shape with rounded corners.
- **Action Flow or Control flows**: Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another activity state. They are represented by solid lines with arrows indicating the direction of flow.
- **Decision node and Branching:** Also known as a decision point or branch point, this node represents a decision to be made within the process flow. It typically involves evaluating a condition and branching the flow based on the result. It is represented by a diamond shape.
- **Fork:** This node is used to split the flow of control into multiple parallel paths, allowing activities to be performed concurrently. It is represented by a solid line with a single incoming flow and multiple outgoing flows.
- **Join:** The join node is used to synchronize multiple parallel paths back into a single path. It is represented by a solid line with multiple incoming flows and a single outgoing flow.
- **Final State:** This represents the end point of the activity diagram. It is typically depicted as a solid circle with a border or an unfilled circle.
- **Object Flow:** Object flows represent the flow of objects or data between activities or nodes in the activity diagram. They are represented by dashed lines with arrows indicating the direction of flow.

ACTIVITY DIAGRAM:



RESULT:

Thus, the relevant State chart diagram and Activity diagram for the LibraryManagement system was drawn successfully.

Ex. No:07	Implement the System as per the detailed design

AIM:

To implement the library management system as per the detailed design.

INTRODUCTION:

The Library Management System project aims to develop a comprehensive software solution that automates and streamlines various functions and processes within a library. By leveraging technology, the LMS project seeks to enhance the management of library resources, improve user experience, and facilitate the day-to-day operations of librarians and administrators.

PROBLEM STATEMENTS:

1. **Limited Accessibility:**

   Lack of a user-friendly online interface restricts access to library resources, especially for remote users who cannot physically visit the library.

2. **Poor User Management:**

   Libraries face challenges in managing user accounts, including registration, authentication, and tracking user activity, leading to ineffective communication and personalized services.

3. **Inadequate Reporting and Analytics:**

   Libraries lack tools for generating insights on usage patterns, resource popularity, and user preferences, hindering informed decision-making and resource allocation.

4. **Integration Complexity:**

   Existing library systems are not seamlessly integrated with each other or with external services, leading to inefficiencies and duplication of efforts.

5. **Security Vulnerabilities:**

Libraries face security risks due to inadequate data protection measures, leaving sensitive patron information and library resources vulnerable to unauthorized access and breaches.

6. Scalability Challenges:

Libraries struggle to scale their operations and services to meet growing demand, resulting in overcrowded facilities, resource shortages, and decreased service quality.

7. Lack of Customization:

Existing library systems lack flexibility and customization options to adapt to the unique needs and workflows of different libraries, leading to inefficiencies and suboptimaluser experiences.

IMPLEMENTATION:

Bill:

```
import java.util.*; public
class Bill { public Bill() {  }
public void billNo; public
void date; public void
memberId;public void
amount;
public void billCreate() { } public
void billUpdate() {  }
}
```

Book:

```
import java.util.*; public
class Book {public
Book() { } public void
bookId;
```

```
public void author; public
void name; public void
price; public void rackNo;
public void status; public
void edition;
public void datOfPurchase;
public void displayBookDetails() {  }public
void updateStatus() { }
}
```

Faculty:

```
import java.util.*;
public class Faculty extends MemberRecord {public
Faculty() { }
}
```

Journals:

```
import java.util.*;
public class Journals extends Book {public
Journals() { }
}
```

Librarian:

```
import java.util.*; public
class Librarian {public
Librarian() { } public void
name; public void password;
public void searchBook() { } public void
verifyMember() { }public void
issueBook() { }
```

```
public void calculateFine() {  }public
void createBill() { } public void
returnBook() { }
}
```

Magazines:

```
import java.util.*;
public class Magazines extends Book {public
Magazines() { }
}
```

MemberRecord:

```
import java.util.*;
public class MemberRecord {public
MemberRecord() { public void
memberId;
public void type;
public void dateOfMembership;public
void noBookIssued; public void
maxBookLimit; public void name;
public void address; public
void phoneNo;
public void retrieveMember() { } public
void increaseBookIssued() { }public void
decreaseBookIssued() {  }public void
payBill() { }
}
```

Student:

```
import java.util.*;
public class Student extends MemberRecord {public
Student() { }
}
```

StudyBooks:

```
import java.util.*;
public class StudyBooks extends Book {public
StudyBooks() { }
}
```

Transaction:

```
import java.util.*;
public class Transaction {public
Transaction() { } public void
transId; public void memberId;
public void bookId; public void
dateOfIssue; public void
dueDate;
public void createTransaction() { } public
void deleteTransaction() { } public void
retrieveTransaction() {}
}
```

RESULT:

Thus, the detailed design of Library Management System was implemented successfully.

Ex. No:08	Test the Software System for all the scenarios identified as perthe use case diagram

AIM:

To test the software system for all the scenarios identified as per the use case diagram.

Library Management System:

A library management system (LMS) is a software tool that automates and streamlines library operations, including cataloging, circulation, member management, reporting, search and discovery, reservations, and integration with other library systems. Its primary goal is to facilitate efficient management of library collections and services, providing librarians with tools to organize and track items, manage memberships, and generate reports, while offering users a user-friendly interface for accessing and discovering library resources.

Review Use Case Diagram:

- Librarians are responsible for tasks such as managing the library catalog, processing book transactions, and generating reports.
- Library members, on the other hand, interact with the system to search for and borrow books, as well as manage their accounts.
- The use case diagram provides a clear representation of the system's functionalities and user roles, ensuring alignment with business requirements.

Identify Test Scenarios:

- Test scenarios are identified by exploring different user roles and perspectives within the library management system.
- These scenarios encompass a diverse range of functionalities, including basic operations and edge cases.

- Examples of test scenarios include logging in as a librarian to add a new book to the system, searching for a book by title and checking it out as a library member, and returning a book. By considering various user roles and scenarios, comprehensive test coverage is ensured.

Create Test Cases:

- Test cases are created by translating identified test scenarios into detailed steps, inputs, expected outputs, and test conditions.
- Both positive and negative test cases, boundary tests, and error-handling scenarios are included to thoroughly validate the system's functionality and robustness.
- For instance, a test case for adding a new book includes steps such as logging in as a librarian, navigating to the "Add Book" page, entering valid book details, and verifying that the book is successfully added to the system.

Prepare Test Environment:

- The test environment is prepared by establishing a dedicated setup that isolates testing activities from the production system, minimizing risks of interference or disruption.
- The test environment is validated to ensure it replicates the production environment's configuration and data accurately, enabling accurate testing of system behavior and performance under realistic conditions.

Execute Test Cases:

- Test cases are executed diligently by adhering to predefined steps and documenting both the actions performed and the observed outcomes.

- Diverse test data is utilized to cover a range of scenarios, including boundary values and invalid inputs to comprehensively validate the system's functionality and behavior.

Report Bugs:

- Any discrepancies or defects encountered during testing are documented in a dedicated bug tracking system.
- Detailed information such as steps to reproduce, system configurations, and screenshots is included to assist developers in diagnosing and resolving issues efficiently.

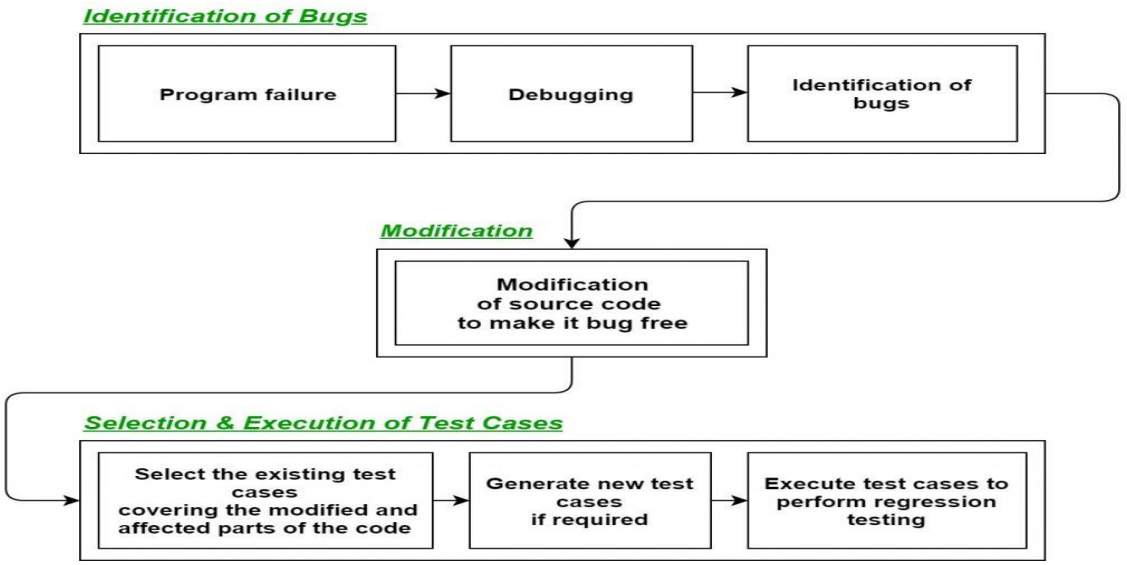
Regression Testing:

- Regression testing ensures that modifications or enhancements to a Library Management System don't introduce new defects or adversely affect existing functionalities.

Example:

Adding a new feature for tracking book reviews.

Test Case: After implementing book reviews, re-run test cases involving book checkout, return, and user management to confirm they still function correctly.



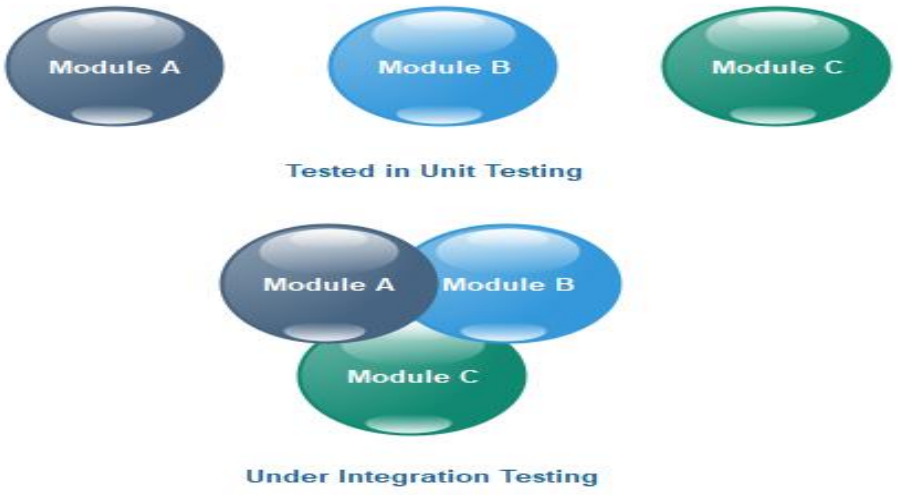
Integration Testing:

- Integration testing ensures that different components and modules of a Library Management System work seamlessly together, validating data flow and system interoperability.

Example:

Integration of the book cataloging module with the user management module.

Test Case: Add a new book to the catalog and ensure that it is visible to library users, who can then check it out through the user management module.



System Testing:

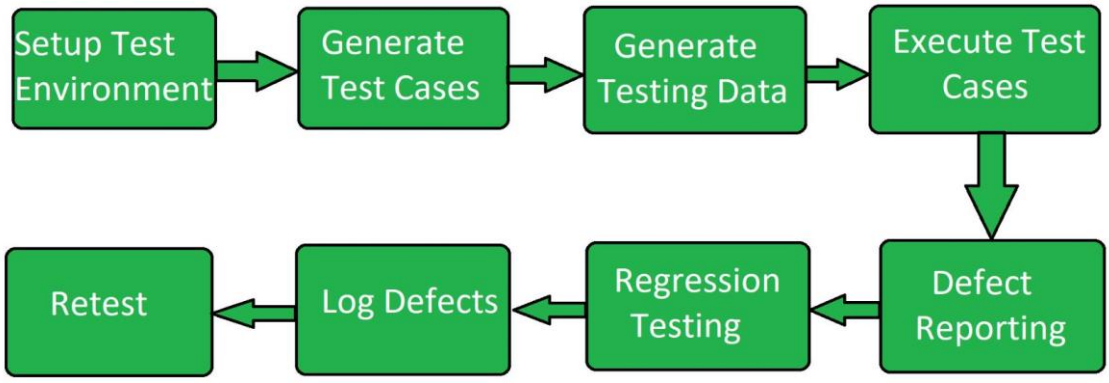
- System testing validates the entire Library Management System to ensure it meets functional and non-functional requirements.

Example:

End-to-End Workflow in a Library Management System.

Test Case: Simulate a user checking out a book, returning it, and generating a report of overdue books. Verify that all system components (catalog, user management, book circulation, reporting)

function correctly throughout the process.



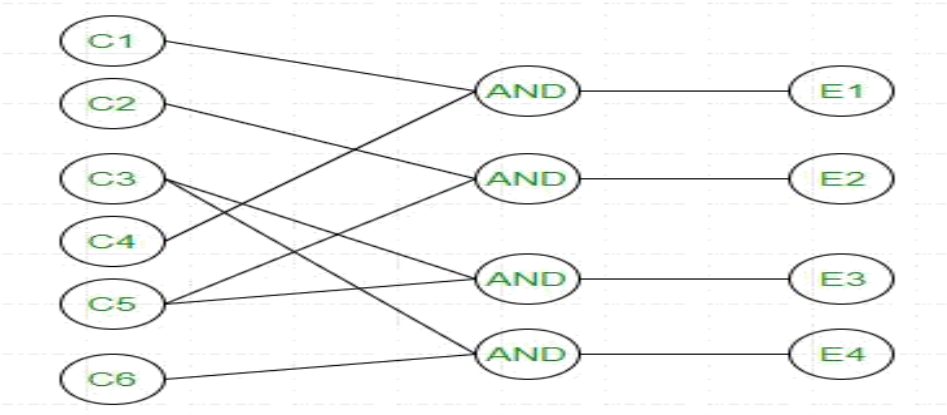
Black Box Testing:

- Black box testing focuses on validating the functionality of the Library Management System without examining its internal implementation details.

Example:

Functionality of Book Checkout.

Test Case: A user attempts to check out a book. Verify that the system correctly processes the checkout, updates the book's status, and reflects the change in the user's borrowed book list.



		1	2	3	4
CAUSES	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	1
EFFECTS	E1	X	-	-	-
	E2	-	X	-	-
	E3	-	-	X	-
	E4	-	-	-	X

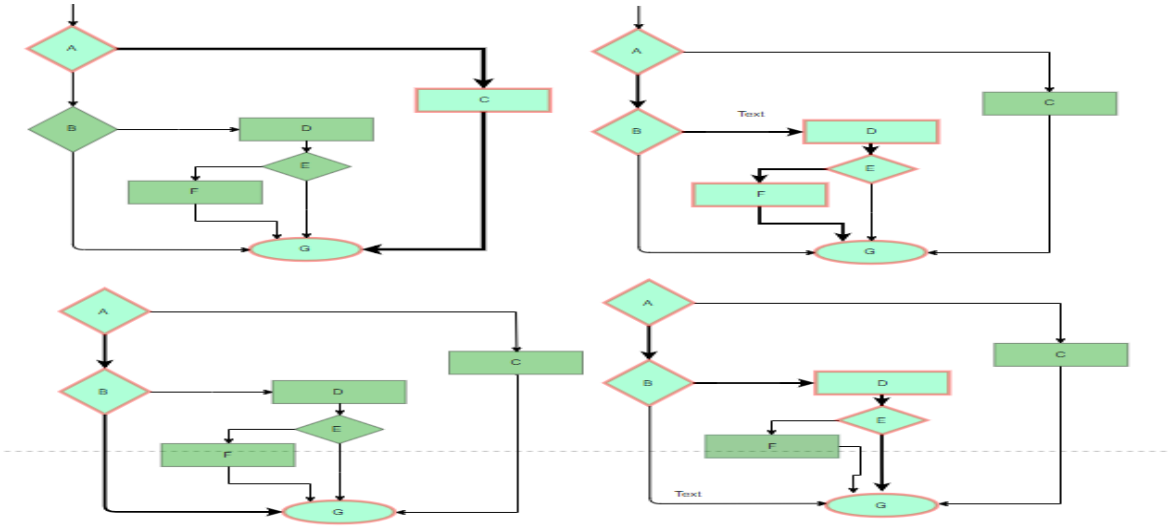
White Box Testing:

- White box testing examines the internal structure, code paths, and logic of the Library Management System.

Example:

Check-in/Check-out algorithm.

Test Case: Verify that the algorithm correctly handles book check-ins and check-outs, including edge cases like multiple users trying to reserve the same book or books being returned past their due dates.



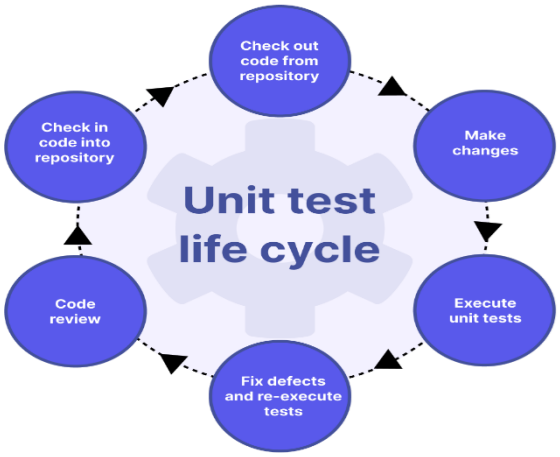
Unit Testing:

- Unit testing ensures that individual components or units of a Library Management System work as expected in isolation.

Example:

User Management Module.

Test Case: Test the `add User ()` function to ensure it correctly adds new users to the system, verifies their data, and assigns the appropriate permissions.



RESULT:

Thus, the software system for all the scenarios identified as per the use case diagram for Library Management System and was tested successfully.

Ex.No:09	Improve the Reusability and Maintainability of the Software System by Applying Appropriate Design Patterns

Aim:

To improve the reusability and maintainability of the software system by applying appropriate design patterns.

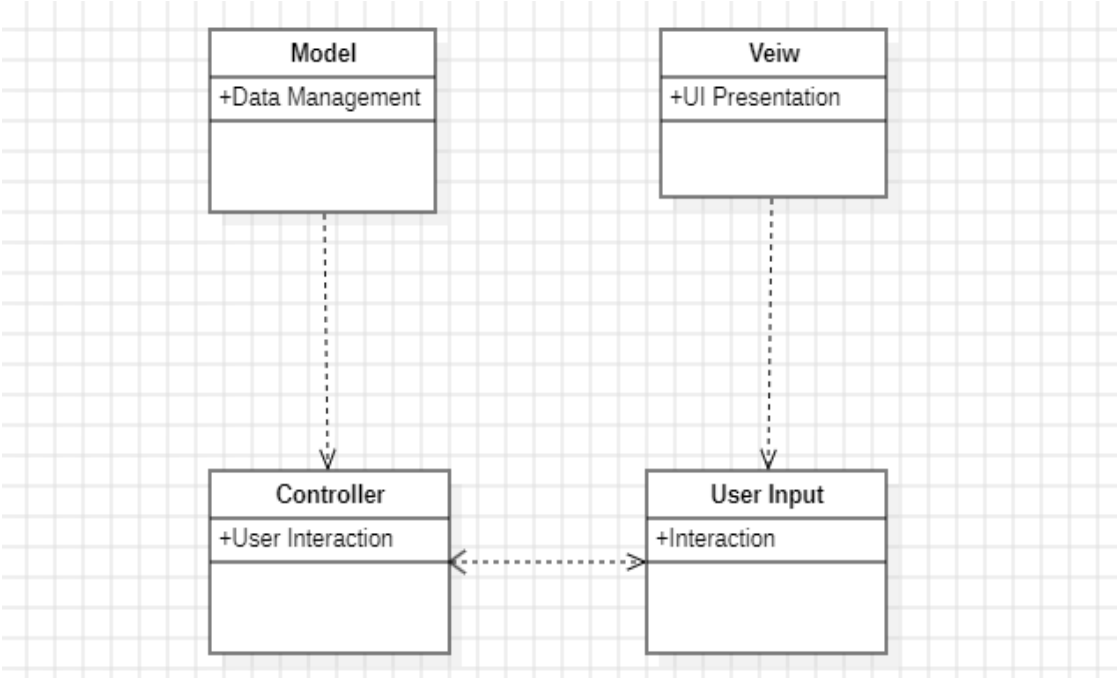
Library Management System:

A library management system (LMS) is a software tool that automates and streamlines library operations, including cataloging, circulation, member management, reporting, search and discovery, reservations, and integration with other library systems. Its primary goal is to facilitate efficient management of library collections and services, providing librarians with tools to organize and track items, manage memberships, and generate reports, while offering users a user-friendly interface for accessing and discovering library resources.

Applying these design patterns to a library management system can greatly enhance its reusability and maintainability. Let's adapt each pattern to fit the context of a library system:

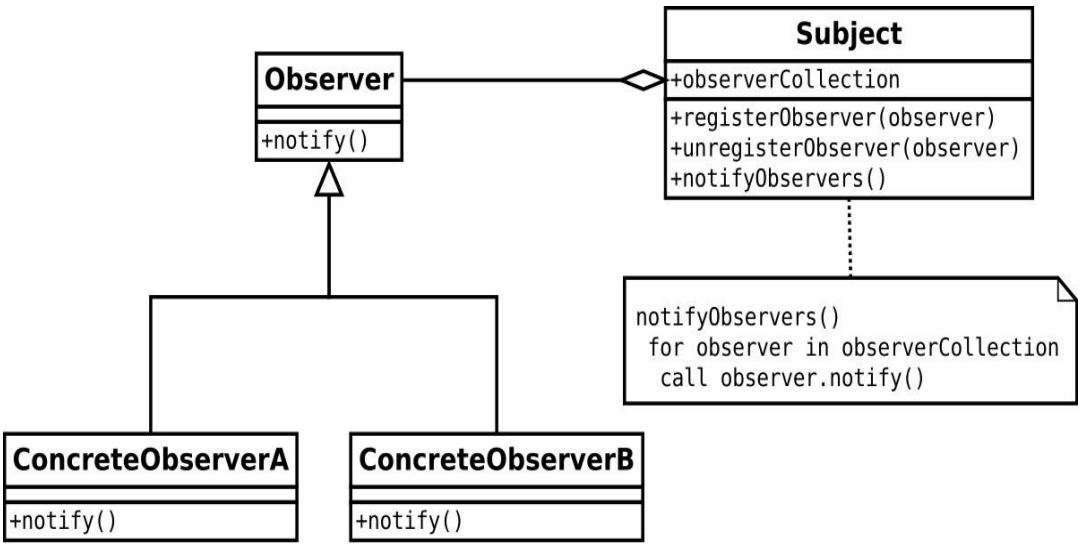
1. Model-View-Controller (MVC) Pattern:

- Separates data, presentation, and user interaction.
- Manages library data, displays information, handles user actions.
  - ❖ Context: Separates data management (Model), UI presentation (View), and user interaction (Controller).
  - ❖ Problem: Without MVC, code complexity increases, making maintenance difficult.
  - ❖ Solution: Divides the system into Model, View, and Controller, simplifying development and improving code organization.



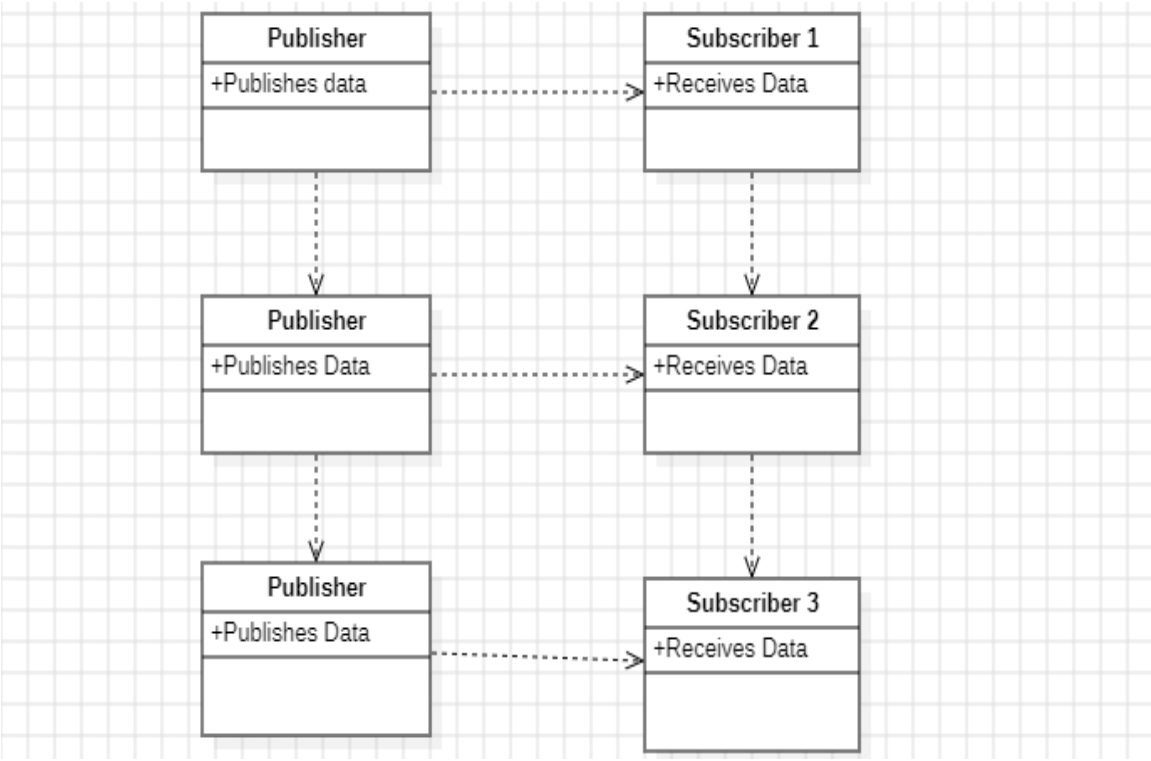
2. Observer Pattern:

- Notifies components of state changes.
- Updates librarians/patrons about new books or overdue notices.
  - ❖ Context: Notifies components about system changes, enhancing flexibility and scalability.
  - ❖ Problem: Establishes a one-to-many relationship without tightly coupling objects, ensuringautomatic updates.
  - ❖ Solution: Define a subject interface with methods for attaching, detaching, and notifyingobservers. Observers register with subjects to receive updates on state changes.



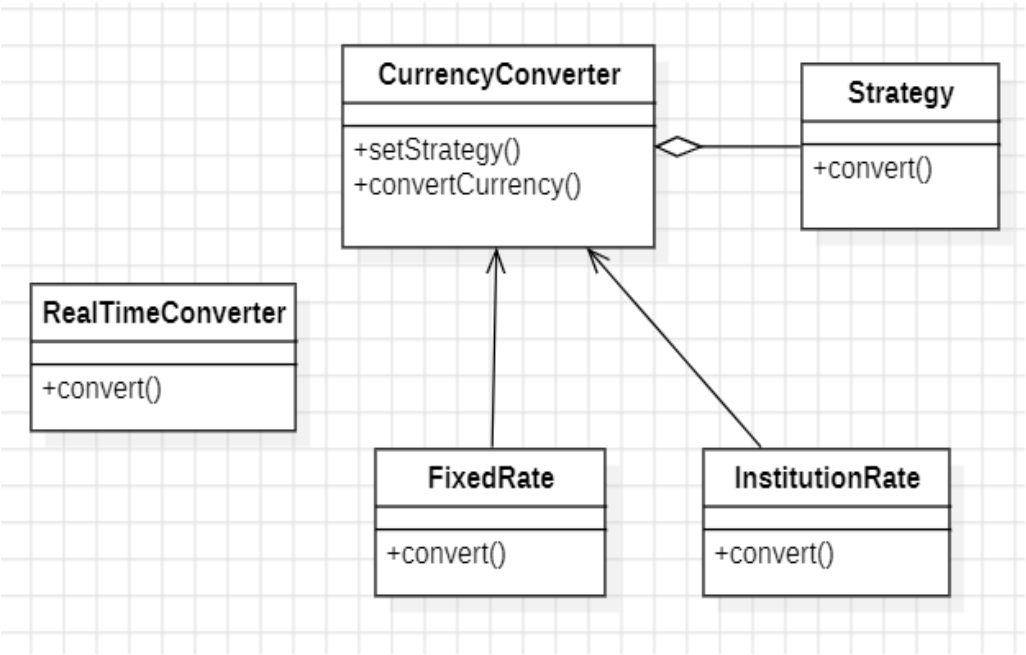
3. Publish-Subscribe Pattern:

- Enables real-time updates across components.
- Distributes updates like new book arrivals efficiently.
  - ❖ Context: Enables real-time updates and notifications across multiple components.
  - ❖ Problem: Distributing real-time updates without this pattern leads to data latency and inefficiencies.
  - ❖ Solution: Provides efficient data distribution, letting providers publish to specific topics, ensuring relevant updates reach subscribing modules, enhancing system scalability and responsiveness.



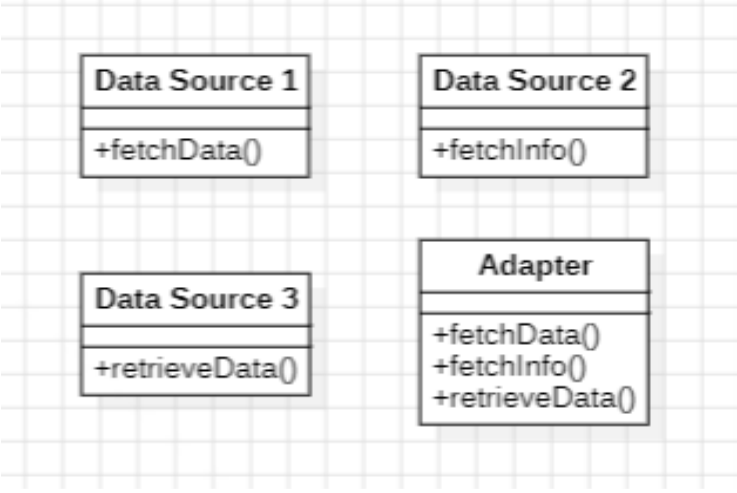
4. Strategy Pattern:

- Defines interchangeable algorithms.
- Manages various tasks like fine calculation or book search.
  - ❖ Context: Handles different algorithms for tasks like international currency conversion.
  - ❖ Problem: Facilitates dynamic algorithm selection and switching, promoting code flexibility and reusability.
  - ❖ Solution: Define a strategy interface with algorithm methods. Implement concrete strategy classes for each algorithm and allow runtime switching between them.



5. Adapter Pattern:

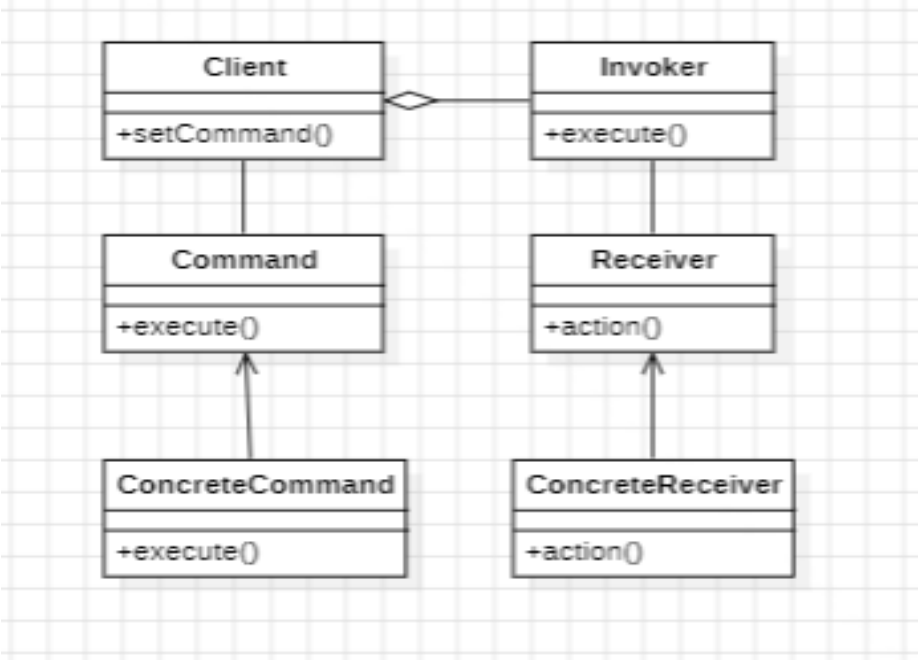
- Converts incompatible interfaces.
- Integrates different data sources seamlessly.
  - ❖ Context: Integrates different data sources seamlessly.
  - ❖ Problem: Incompatibility between data sources and the system leads to communication issues.
  - ❖ Solution: Provides a wrapper translating data formats and protocols for seamless integration without compromising data integrity or communication.



6. Command Pattern:

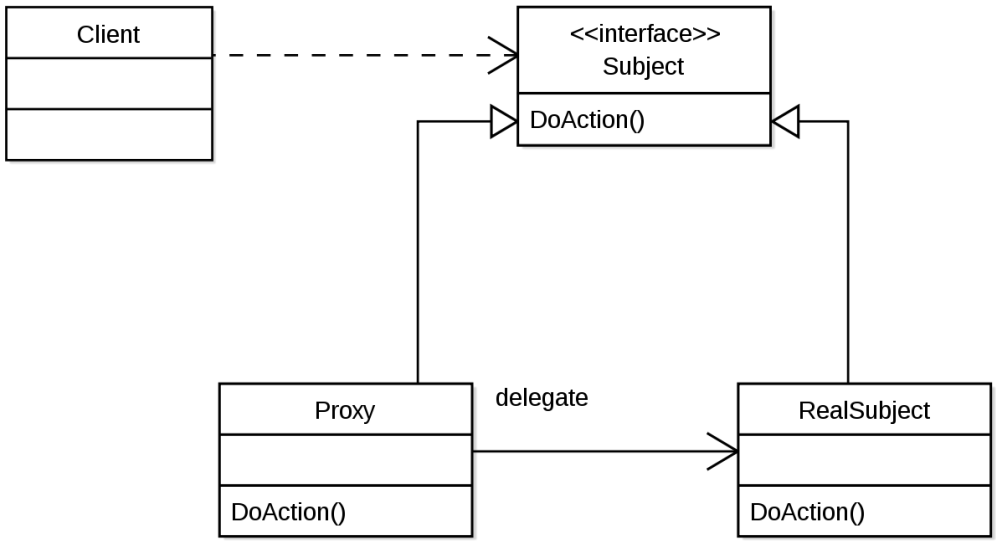
- Encapsulates requests as objects.
- Represents actions like borrowing or returning books.

- ❖ Context: Encapsulates requests as objects, enabling parameterization, queuing, logging, and supporting undoable operations.
- ❖ Problem: Decouples sender and receiver, supports undoable operations, and provides structured request handling.
- ❖ Solution: Define command objects that encapsulate requests and parameters. Clients create and queue commands, and invokers execute them when needed.



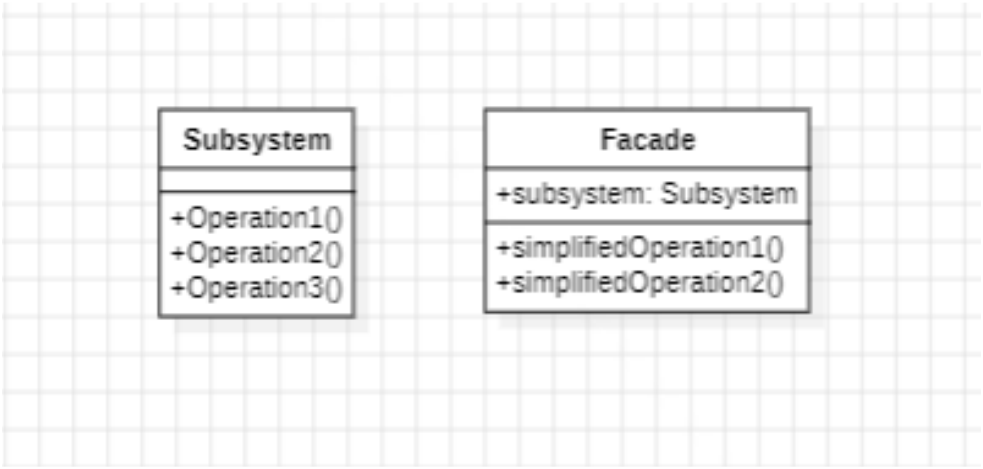
7. Proxy Pattern:

- Controls access to sensitive data.
- Manages access to books, enforcing borrowing rules.
  - ❖ Context: Controls access to sensitive data or expensive operations.
  - ❖ Problem: Direct access to resources can lead to security risks or performance issues.
  - ❖ Solution: Provides a surrogate object managing access, adding functionalities like caching or logging, enhancing security, performance, and resource management.



7. Facade Pattern:

- Provides a simplified interface to a complex system.
- Simplifies interactions for managing books and borrowers.
  - ❖ Context: Simplifies interactions with complex subsystems.
  - ❖ Problem: Managing multiple subsystem interactions leads to complex code.
  - ❖ Solution: Facade Pattern simplifies interactions, hiding subsystem complexities for easier system management and maintenance.



Result:

Thus the reusability and maintainability of the software system by applying appropriate design pattern was improved successfully.