EX.NO: 1 DATE:	**Identify a Software System that needs to be Developed** **Book Bank Management System**

AIM:

To identify a software system that needs to be developed - Book Bank Management System.

INTRODUCTION:

In the realm of educational institutions, the availability of textbooks and reference materials is indispensable for fostering academic excellence. However, the traditional method of purchasing individual textbooks can be cost-prohibitive for many students. Enter the Book Bank Management System – a revolutionary solution designed to democratize access to educational resources. The Book Bank Management System streamlines the process of borrowing and returning textbooks, ensuring equitable distribution and maximizing the utility of available resources. By centralizing the management of textbooks, institutions can reduce financial barriers for students while promoting sustainability through the efficient use of materials.

FEATURES OF BOOK BANK MANAGEMENT SYSTEM:

1. USER REGISTRATION:

- **Description:** User registration is the process by which individuals create accounts within the Book Bank Management System to access its services and resources.
- **Functionality:** Users typically fill out a registration form with information such as name, student/faculty ID, email address, contact number, and any other required details. The system validates the provided information to ensure accuracy and completeness. Once validated, the user's account is created, and they receive confirmation of registration.
- **Purpose:**User registration allows individuals to access the book bank's resources,borrow books, reserve materials, and manage their account preferences. It facilitates personalized interactions and services tailored to the user's needs.

2. AUTHENTICATION:

- **Description:** Authentication verifies the identity of users attempting to access the Book Bank Management System, ensuring that only authorized individuals can log in and utilize its features.

- **Functionality:** Upon accessing the system's login interface, users provide their credentials, typically consisting of a username and password. The system compares these credentials with stored user data. If the provided credentials match those in the system, access is granted, and the user is logged into their account.
- **Purpose:** Authentication protects the system from unauthorized access, safeguarding user data, borrowing history, and other sensitive information stored within the Book Bank Management System. It ensures the integrity and security of the system by verifying the identity of users before granting access.

3. BOOK SEARCH:

- **Description:** Book search functionality allows users to explore the book bank's inventory and find specific titles, authors, subjects, or keywords of interest.
- **Functionality:** Users enter search queries into the system's search interface, which retrieves relevant results from the catalog database. The system may offer advanced search options, filters, and sorting capabilities to refine search results. Users can view detailed information about each book, including availability status, location, and borrowing history.
- **Purpose:** Book search empowers users to locate and access the books they need efficiently, enhancing their overall experience and satisfaction with the Book Bank Management System. It facilitates the discovery of relevant resources and promotes self-service access to library materials.

4. GET BOOKS (BORROWING):

- **Description:** The "get books" feature enables users to borrow books from the book bank's collection for personal use or academic purposes.
- **Functionality:** After finding the desired books through the search functionality, users can select the titles they wish to borrow and initiate the borrowing process. The system checks the availability of the selected books and updates the user's borrowing record accordingly. It assigns a due date for return and provides users with confirmation of the successful borrowing transaction.
- **Purpose:** Borrowing functionality facilitates seamless access to library resources, allowing users to borrow books for study, research, or leisure purposes. It promotes knowledge acquisition, supports academic pursuits, and fosters a culture of lifelong learning within the user community.

5. MEMBERSHIP STATUS:

- **Description:** Membership status indicates the current standing of users within the Book Bank Management System, reflecting their borrowing privileges, account status, and any associated membership tiers or categories.
- **Functionality:** The system tracks users' membership statuses based on predefined criteria such as enrollment status, academic affiliation, or library membership type. Users may have different levels of access or borrowing privileges based on their membership status. The system displays users' membership statuses within their account profiles, allowing them to view their current privileges and account status.
- **Purpose:** Membership status management ensures fair and equitable access to book bank resources, aligning borrowing privileges with users' eligibility criteria and institutional affiliations. It provides transparency regarding users' access rights and helps administrators monitor user accounts, enforce borrowing policies, and administer membership benefits effectively.

6. FINE CALCULATION:

- **Description:** Fine calculation determines the fines incurred by users for overdue book returns, encouraging timely return of borrowed materials and ensuring accountability in the use of library resources.
- **Functionality:** The system tracks due dates for borrowed books and calculates fines based on predefined rules established by the library, such as daily late fees or fixed fines per overdue period. When a book becomes overdue, the system calculates the fine accrued based on the duration of the overdue period and the applicable fine rules. It updates the user's account with the fine amount and notifies the user of the overdue status and fines incurred.
- **Purpose:** Fine calculation promotes responsible borrowing behavior and helps maintain the availability and integrity of library resources. It incentivizes users to return borrowed materials promptly, ensuring equitable access to library resources for all users. Fine calculation also generates revenue for the library and helps offset costs associated with managing overdue materials.

RESULT:

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

EX.NO: DATE:	**Document The Software Requirements Specification(SRS) For The Identified System**

AIM:

To implement a Software Requirement Specification (SRS) for Book Bank management system.

1.PROBLEM STATEMENT:

1. Books are gathered at the start of each semester and must be given back by the semester's end.This keeps track of borrowed books and ensures they're available for everyone.
2. People who want to join need to pay a deposit. This helps protect the books and gives members borrowing rights.
3. Members can take out either six or seven books every semester. This makes sure everyone has a fair chance to borrow books while keeping track of how many books each person has.
4. Members can renew their membership using a special number they're given. This makes it easy for them to keep borrowing books if they want to.
5. There's a system that keeps track of who's borrowing what and when. And there's a search tool that helps members find out if a specific book is available.

SOFTWARE REQUIREMENT SPECIFICATION:

2.INTRODUCTION:

The book bank is like a library where members can borrow books for a certain period, usually a semester. They have to return the books by the end of the semester. The book bank keeps track of who borrowed which book, when they borrowed it, and when they need to return it. It also keeps track of any fees or penalties if someone returns a book late or damages it. This helps the book bank manage borrowing and make sure everyone follows the rules. By keeping an eye on what members are doing and what books are available, the book bank aims to make borrowing easy and fair for everyone.

2.1 PURPOSE:

- The document provides a full picture of the Book Bank System, explaining its purpose, functions, and features.
- It describes how users interact with the system, such as signing up, searching for books, and

managing administrative tasks.

- The document outlines what the system does, including book lending, member management, inventory tracking, and fine calculation.
- It lists any limitations the system must work within, like budget, technology, regulations, or business rules.
- Lastly, it explains how the system responds to unexpected events like system failures, security breaches, user behavior changes, or updates to regulations.

2.2 DOCUMENT CONVENTION:

	Font	Style	Size
Heading	Times New Roman	Bold	16
Sub-Heading	Times New Roman	Bold	14
Other's	Times New Roman	Regular	12

2.3 INTENDED AUDIENCE AND READING SUGGESTIONS:

This SRS is mainly developed for the project development team. In this team there are the project manager, developer, coder, tester and documentation writer and the user of the project also.

User (Customer):

This document is intended to user and customer to make ensure that the document satisfies the needs of the customer.

Project Manager:

This SRS document is also very important for the project manager as it helps in cost estimation which can be performed by referring to the SRS document and it contains all the information that is required for the project plan.

Project Developer:

The project developer will refer to the SRS document to ensure that the product developed is as per the needs of the customer.

Tester:

The tester reads the SRS document to ensure that the requirements are understandable based on the functionality specified so that he can test the software and validates its working.

Document Writer:

The document writer reads the SRS document to ensure that they understand the document well enough and write user manuals based on the SRS document.

Maintenance:

The SRS document helps the maintenance engineers to understand functionality of the system. A clear knowledge of the functionality is needed to design and code

2.4 PRODUCT SCOPE:

The Book Bank Management System is designed to automate and streamline book lending operations, member management, and inventory tracking within the book bank organization. It provides a user-friendly interface for members to access and manage their borrowing activities, while offering administrative features for staff to monitor transactions and generate reports.

2.5 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS:

- **Director:** The highest authority in the book bank organization responsible for overseeing system operations.
- **Member:** Any individual registered with the book bank organization.
- **HTML:** HyperText Markup Language used for web page creation.

3. OVERALL DESCRIPTION:

3.1 PRODUCT PERSPECTIVE:

The Book Bank Management System is a standalone software application designed to integrate seamlessly with the existing infrastructure of the book bank organization. It operates as a centralized platform for managing book lending activities, member information, and inventory tracking.

3.2 PRODUCT FUNCTIONS:

The key functions of the system include:

- Member registration and authentication
- Book browsing and searching
- Book borrowing and return management
- Fine calculation and penalty enforcement
- Inventory management and tracking
- Reporting and analytics

3.3 USER CLASSES AND CHARACTERISTICS:

The system caters to two primary user classes:

Members: Individuals registered with the book bank organization who borrow books.

Administrators: Staff members responsible for managing the system, handling book transactions, and generating reports.

3.4 OPERATING ENVIRONMENT:

Particulars	Client System	Server System
Operating Systems	Windows/Linux/Android/OS	Linux
Processor	Intel or AMD	Intel or AMD
Hard Disk	1 GB	1 TB
RAM	256 MB	8 GB

3.5 DESIGN AND IMPLEMENTATION CONSTRAINTS:

The system must adhere to industry-standard design and development practices, utilizing technologies such as HTML, CSS, JavaScript, and PHP for web-based functionality. It should also comply with relevant security and data protection regulations.

3.6 ASSUMPTIONS AND DEPENDENCIES:

The successful implementation of the Book Bank Management System assumes the availability of adequate hardware infrastructure, including servers and network components. It also depends on the cooperation and support of stakeholders and end-users throughout the development and implementation process.

4. EXTERNAL INTERFACE REQUIREMENTS:

4.1 USER INTERFACES:

The system will feature a user-friendly web-based interface accessible to both members and administrators. It will include intuitive navigation menus, search functionality, and interactive forms for book borrowing and management.

4.2 HARDWARE INTERFACES:

The system requires standard hardware components including servers, computers, and network infrastructure to host and operate the software.

4.3 COMMUNICATION INTERFACES:

The system will integrate with email and SMS notification services for member communication, providing alerts for overdue books, pending requests, and other relevant updates.

5. SYSTEM FUNCTION:

The system will perform various functions to facilitate efficient book lending operations, including member management, book browsing, borrowing, return management, fine calculation, inventory tracking, and reporting.

6. OTHER NON-FUNCTIONAL REQUIREMENTS:

6.1 PERFORMANCE REQUIREMENTS:

- The system should support concurrent access by multiple users without performance degradation.
- Response times for user interactions (e.g., book search, borrowing) should be minimal.

6.2 SECURITY REQUIREMENTS:

- The system must ensure data confidentiality and integrity, implementing appropriate encryption measures.
- Access control mechanisms should be in place to restrict unauthorized access to sensitive information.

6.3 SOFTWARE QUALITY ATTRIBUTES:

- The system should be reliable, scalable, and maintainable, with minimal downtime and easy maintenance procedures.
- User interfaces should be intuitive, accessible, and responsive across different devices and screen sizes.

6.4 BUSINESS RULES:

- Members must be registered with the book bank organization to borrow books.
- Books must be returned by the specified due date to avoid fines.
- Administrators are responsible for approving book requests, managing member accounts, and maintaining inventory records.

RESULT:

Thus the Software Requirement Specification (SRS) for Book Bank management system was implemented successfully.

EX NO:3 DATE:	Identify use cases and develop the Use Case model

AIM:

To identify the use cases and develop use case model

Introduction of Use Case Diagram:

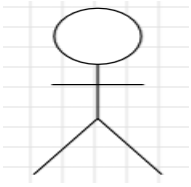
A use case diagram is a type of behavioural diagram in the Unified Modelling Language (UML) that represents the interactions between a system and its actors in terms of the system's behaviour. It provides a graphical overview of the functionalities provided by a system and the actors involved in those functionalities. Use case diagrams are widely used in software engineering to capture the requirements of a system and to visualize the high-level system structure.

Purpose of Use Case Diagram:

The purpose of a use case diagram is to visually represent the functional requirements of a system from the perspective of its users or external systems. It helps in understanding the system's behaviour by illustrating how users interact with the system to accomplish specific tasks or goals. Use case diagrams facilitate communication between stakeholders, including developers, designers, and clients, by providing a clear and concise representation of the system's functionality.

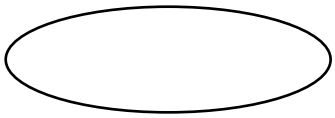
Notation:

Actors: Actors represent the users or external systems interacting with the system being moduled . In the Book bank management system, the actors include the students, administrator, database.



Use Cases: Use cases represent the functionalities or services provided by the system. Each use case describes a specific interaction between an actor and the system to achieve a particular

goal. In the Book bank management system, examples of use cases include "Registration," "Search books," "Get books," "Membership status,".



Relationships: Relationships, such as associations and dependencies, depict the connections between actors and use cases. Associations represent the interaction between actors and use cases, while dependencies represent the reliance of one use case on another

Benefits of Use Case Diagram:

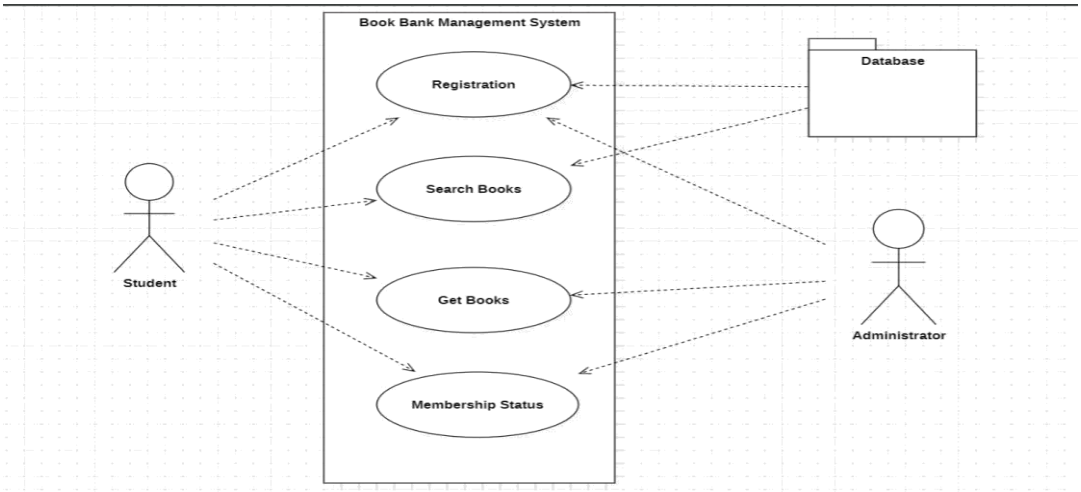
Requirements Elicitation: Use case diagrams help in eliciting and defining the functional requirements of a system by identifying the interactions between users and the system.

Visualization: Use case diagrams provide a visual representation of the system's functionality, making it easier for stakeholders to understand and analyze the system's behaviour.

Communication: Use case diagrams serve as a communication tool between stakeholders, including developers, designers, and clients, by providing a common understanding of the system's requirements and behaviour.

Requirements Verification: Use case diagrams can be used to verify whether all the necessary functionalities and interactions have been captured and addressed in the system design.

Use Cases of Book Bank Management System



User:

Registration:

- This use case represents the process through which students register themselves with the book bank system. During registration, students provide necessary information such as their name, student ID, and contact details. Once registered, students become authorized users of the book bank system.

Get books:

- This use case describes the process where students borrow books from the book bank system. Students can request specific books they want to borrow, and the system facilitates the borrowing process by checking the availability of the requested books and managing the borrowing transactions.

Search books:

- This use case allows students to search for books available in the book bank system. Students can search for books based on various criteria such as title, author, genre, or ISBN. The system provides search results based on the entered criteria, helping students find the desired books efficiently.

Membership Status:

- This use case enables students to check their membership status within the book bank system. By accessing this functionality, students can verify whether their membership is active, view the expiration date of their membership, or check any outstanding dues or penalties associated with their membership.

Actors:

- Actors represent the users or external systems interacting with the system being moduled. In the Book bank management system, the actors include the students, administrator, database.

Student:

- Represents individuals who interact with the book bank system as users. Students utilize various functionalities of the system, including registration, borrowing books, searching for books, and checking their membership status.

Administrator:

- Represents the personnel responsible for managing the book bank system. Administrators oversee the system's operations, handle administrative tasks such as user registration and book management, and assist students in accessing the system's functionalities.

Database:

- Represents the database system that stores and manages the data related to users, books, transactions, and other relevant information within the book bank system. The database interacts with the system to retrieve and update data as needed during various operations

Result

Thus to identify the use cases and develop use case model was done successfully.

EX NO:4 DATE:	**Identify the conceptual classes and develop a Domain Model and also derive a Class Diagram from that**.

AIM:

To identify the conceptual classes and develop a domain model and derive class diagram for Book bank management system.

Conceptual Classes:

1. **Book:** Represents individual books that are stored in the book bank. This class could include attributes such as title, author, ISBN, genre, publication year, and condition

2. **User:** Represents individuals or organizations who interact with the book bank. This class could include attributes such as name, contact information, and membership status.

3. **Transaction:** Represents the borrowing and returning of books from the book bank. This class could include attributes such as transaction ID, book ID, user ID, transaction type (borrow or return), and transaction date.

4. **Inventory:** Represents the collection of books available in the book bank. This class could include attributes such as book ID, quantity available, location within the book bank, and availability status.

5. **Membership:** Represents the membership status of users, whether they are registered members or not. This class could include attributes such as membership ID, membership type (individual or organizational), and membership duration.

Domain Model for Book Bank Management System:

This domain model represents the core entities and their relationships within a book bank management system. Depending on the specific requirements of the system, you may need to further refine and expand upon these entities.

Registration:

- Attributes:
- Name: String
- Age: Integer
- Address: String
- Year of Study: Integer
- Contact No: integer
- Member id: Integer
- User name: String
- Pass word: String
- Operations:
- Register()

Authentication:

- Attributes:
- Username
- Pass Word :
- Operations:
- Login()

Book Search:

- Attributes:
- Title
- Author
- Publications
- ISBN

- Operations:
- Search()

get Book:

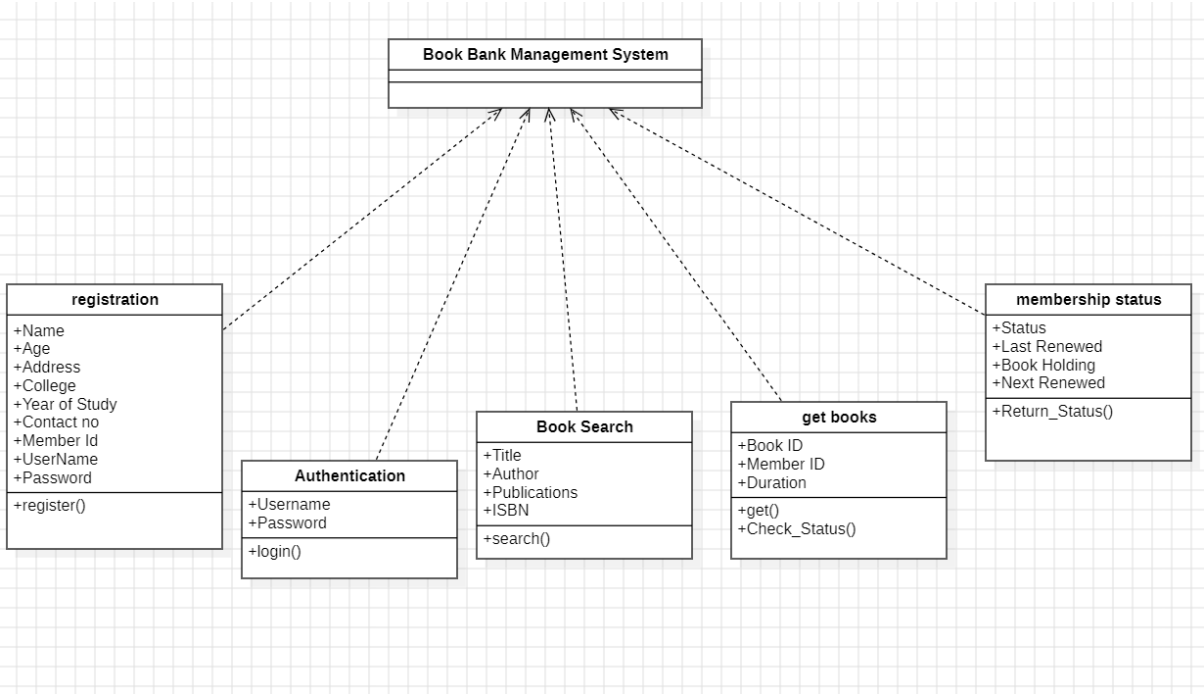
- Attributes:
- Book Id
- Member Id
- Duration
- Operation:
- get()
- Check_ Status()

Membership Status:

- Attributes:
- Status
- Last Renewed
- Book Holding
- Next Renewal
- Operation:
- Return_ Status()

This domain model captures the key entities and their attributes within the Book Bank Management System, Such as Book bank management, registration, Authentication, Book Search, Get books, Membership Status. The operations associated with each entity represent the functionalities performed within the system, including register, login, search, get ,check the status and return status

Class diagram for BPO management system:



RESULT

Thus the class diagram for Book Bank management system is implemented and executed Successfully.

EX NO:5 DATE:	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.

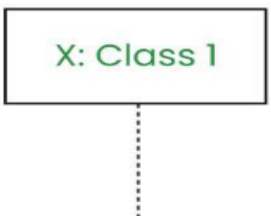
SEQUENCE DIAGRAM:

- A sequence diagram is a type of interaction diagram in UML that represents the interactions between objects over time.
- It illustrates the sequence of messages exchanged between objects within a system to accomplish a specific task or scenario.
- **Sequence Diagram Notation:**

1.Actors:

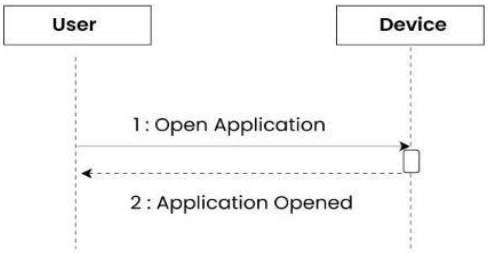
An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

2.Lifelines:



A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.

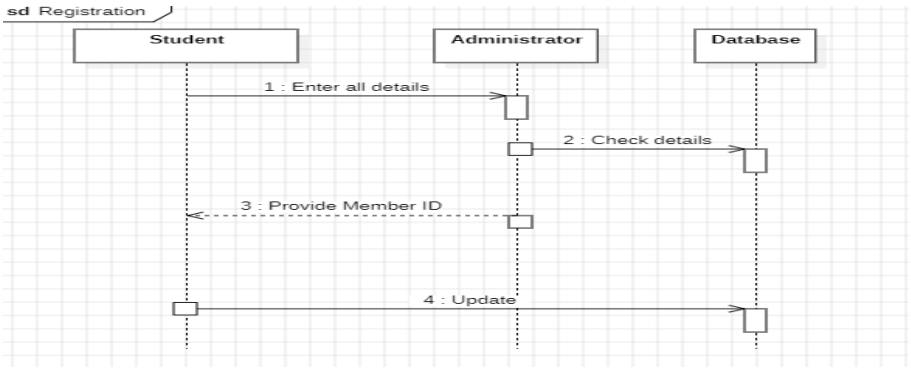
3.Messages:



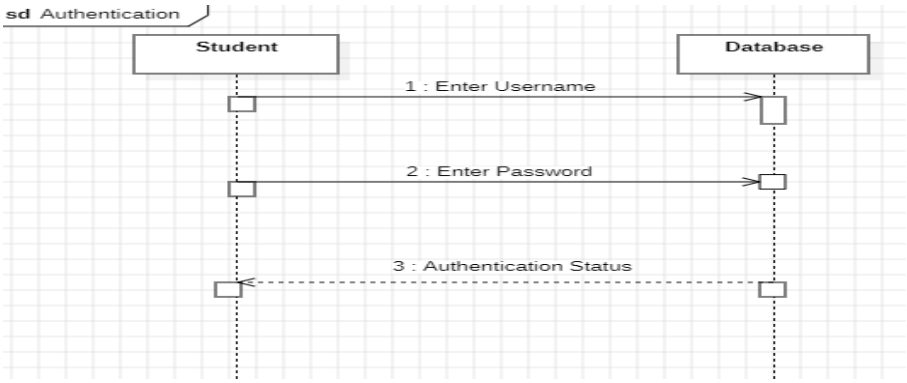
- Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline.
- We represent messages using arrows.
- Lifelines and messages form the core of a sequence diagram.

SEQUENCE DIAGRAM FOR BOOK BANK SYSTEM:

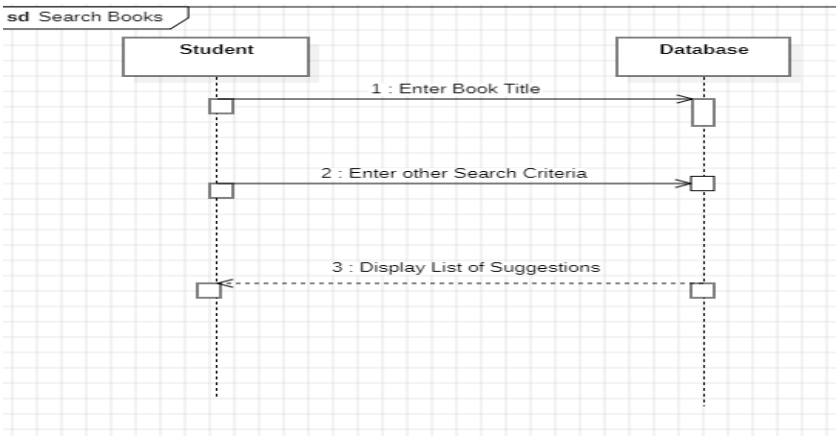
REGISTRATION



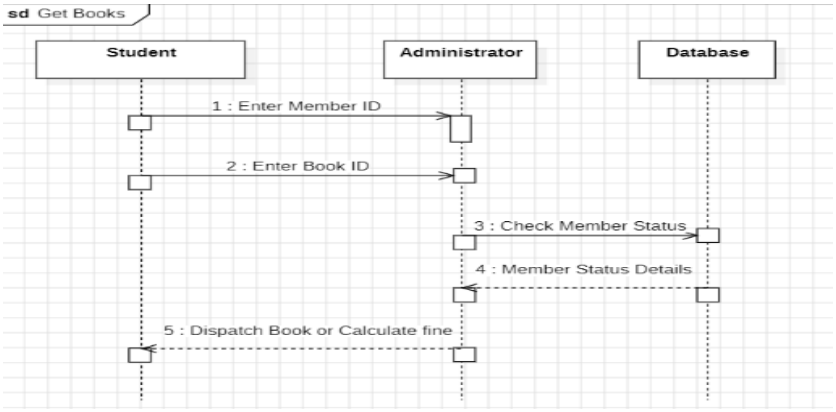
AUTHENTICATION



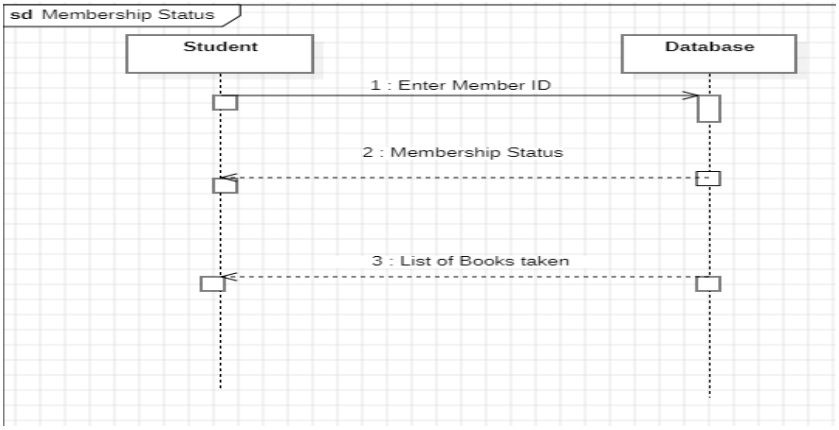
SEARCH BOOKS



GET BOOKS



MEMBERSHIP STATUS



COMMUNICATION DIAGRAM:

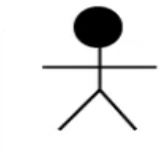
A communication diagram in UML (Unified Modelling Language) is a type of interaction diagram that visualizes the interactions between objects or parts in terms of sequenced messages. It emphasizes the structural organization of the objects and how they collaborate to achieve a specific task or behaviour. Communication diagrams are particularly useful for understanding the dynamic aspects of a system's behaviour during runtime.

Collaboration Diagram Notation:

1. **Objects/Participants:**

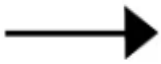
Objects are represented by rectangles with the object's name at the top. The diagram shows each object participating in the interaction as a separate rectangle. Objects are connected by lines to indicate messages being passed between them.

2. **Actors:**



They are usually depicted at the top or side of the diagram, indicating their involvement in the interactions with the system's objects or components. They are connected to objects through messages, showing the communication with the system.

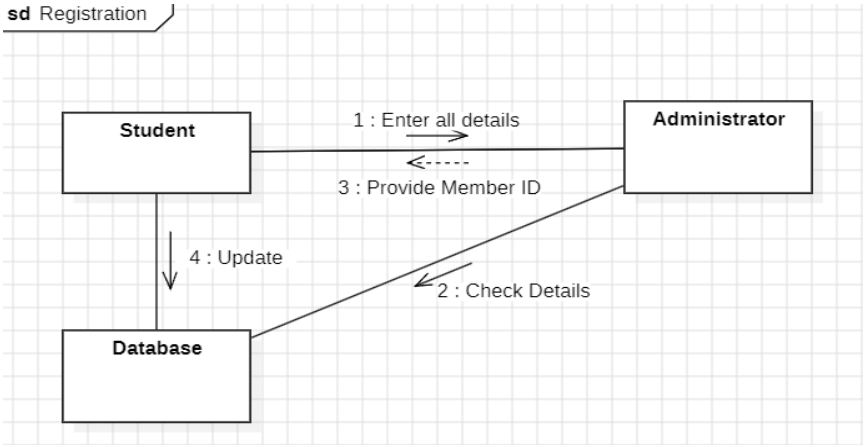
3. **Messages:**



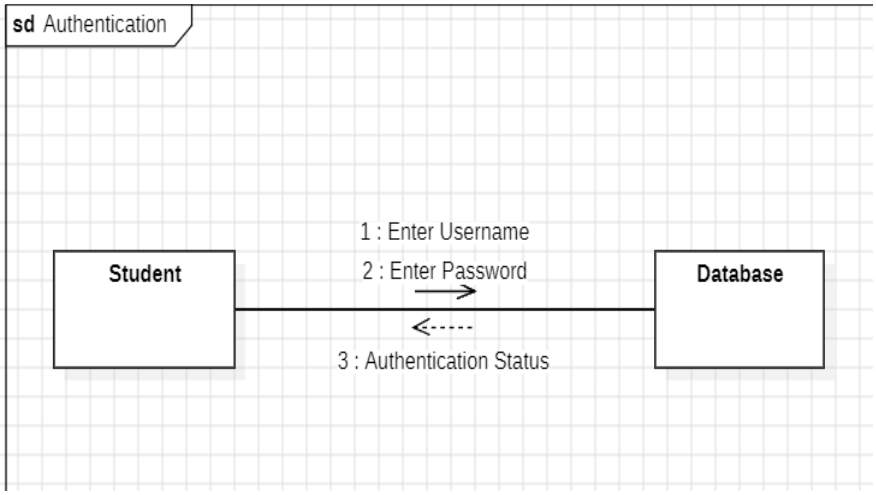
Messages represent communication between objects. Messages are shown as arrows between objects, indicating the flow of communication. Each message may include a label indicating the type of message (e.g., method call, signal). Messages can be asynchronous (indicated by a dashed arrow) or synchronous (solid arrow).

COMMUNICATION DIAGRAM FOR BOOK BANK SYSTEM:

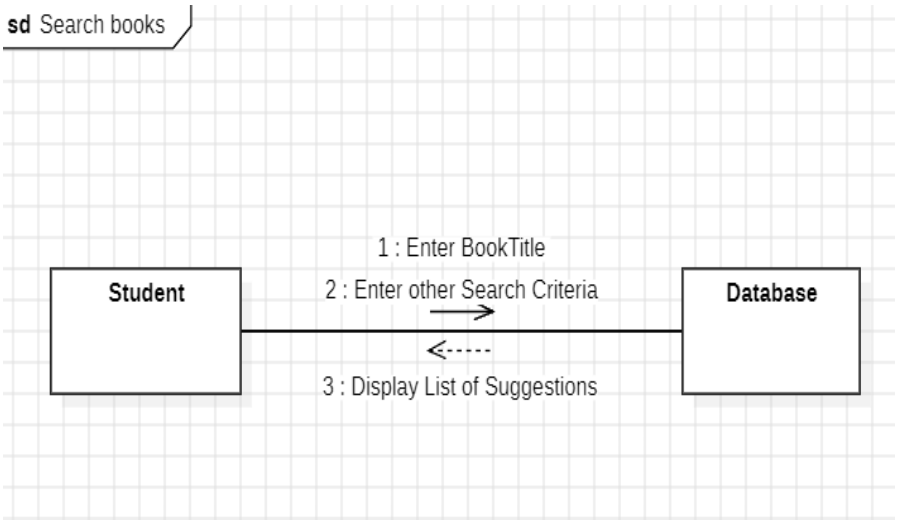
REGISTRATION



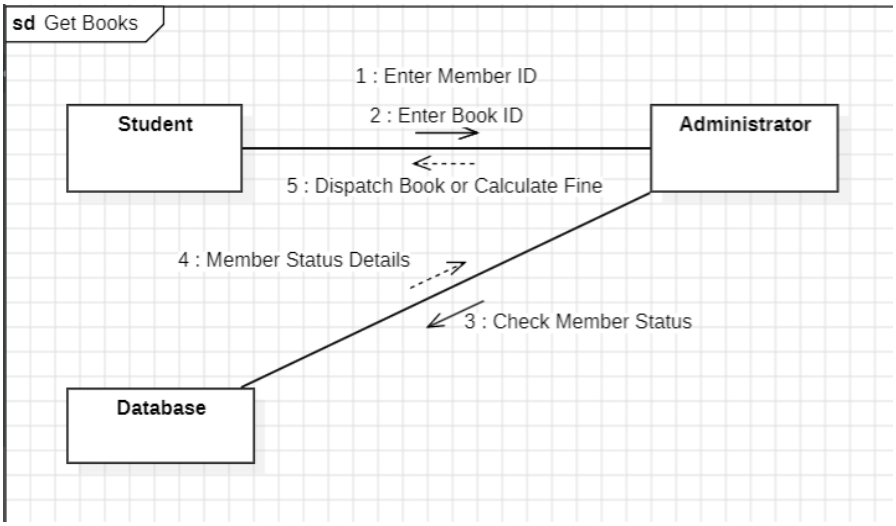
AUTHENTICATION



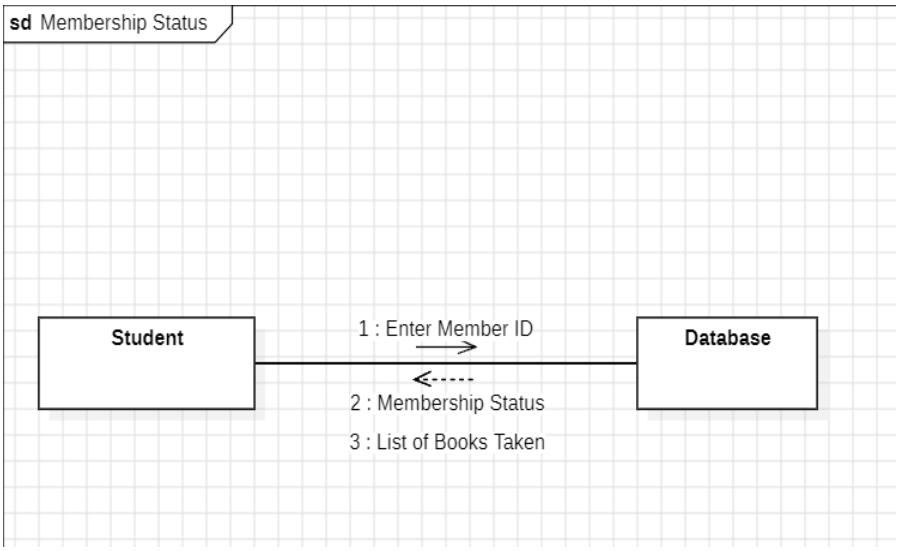
SEARCH BOOKS



GET BOOKS



MEMBERSHIP STATUS



RESULT:

Thus to find the interaction between objects and represent them using UML Sequence and Collaboration diagram was done successfully.

EX NO:6 DATE:	**Draw Relevant State Chart And Activity Diagram For The Same System**

AIM:

To draw State Chart diagram and Activity diagram for Book Bank System.

STATE-CHART DIAGRAM:

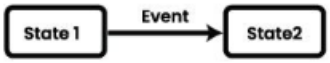
A State Machine Diagram represents the system's condition or part of the system at finite instances of time. It's a behavioural diagram and it represents the behaviour using finite state transitions.

State Chart Diagram Notation:

1.Initial State:

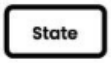
We use a black-filled circle to represent the initial state of a System or a Class.

2. Transition:



We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

3.State:



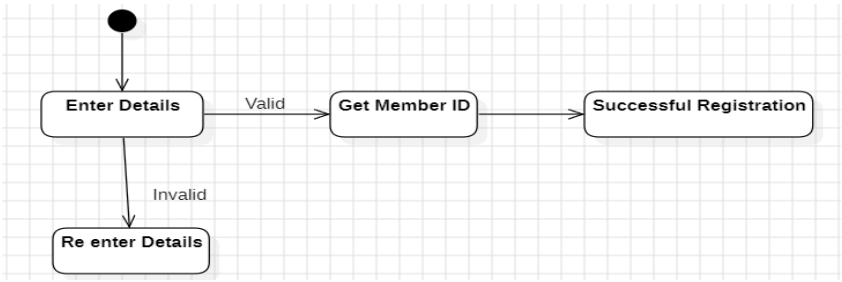
We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.

4.Final State:

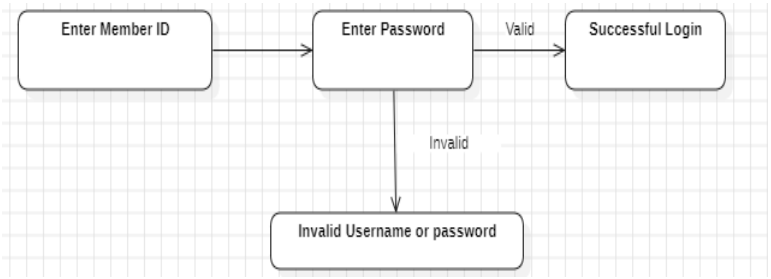
We use a filled circle within a circle notation to represent the final state in a state machine diagram.

STATE-CHART DIAGRAM FOR BOOK BANK SYSTEM:

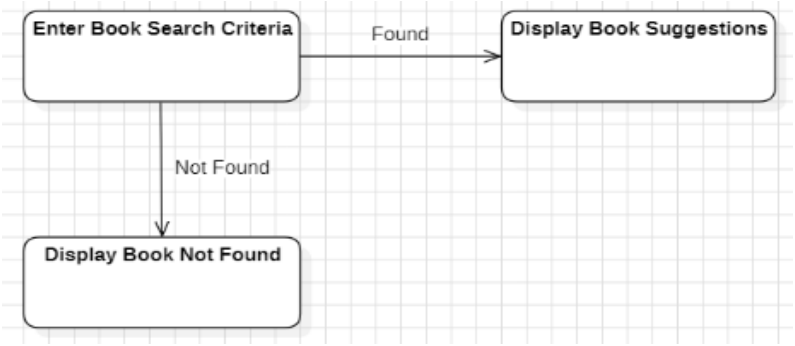
REGISTRATION



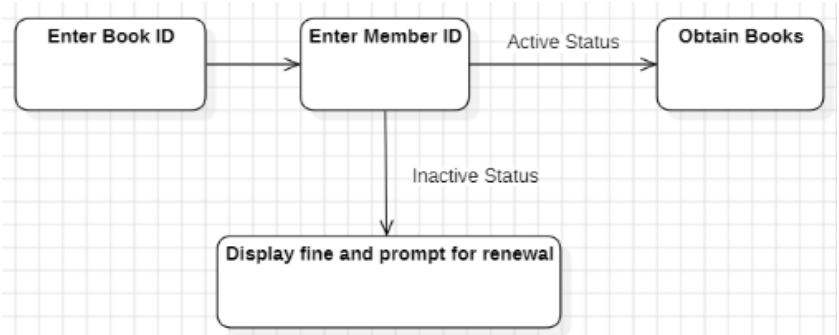
AUTHENTICATION



SEARCH BOOKS



GET BOOKS



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.

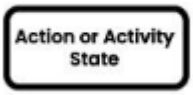
- We describe what causes a particular event using an activity diagram.
- An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.
- They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system.

Activity Diagram Notation:

1. **Initial State:**

A process can have only one initial state unless we are depicting nested activities. We use a black-filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

2. **Action or Activity State:**



An activity represents the execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically, any action or event that takes place is represented using an activity.

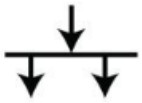
3. **Action Flow or Control Flow:**

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.

4. **Decision Node or Branching:**

When we need to make a decision before deciding the flow of control, we use the decision node. The outgoing arrows from the decision node can be labelled with conditions or guard expressions. It always includes two or more output arrows.

5. **Fork:**



Fork nodes are used to support concurrent activities. When we use a fork node both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts.

6. **Join:**

Join nodes are used to support concurrent activities converging into one. We have two or more incoming edges and one outgoing edge for join notations.

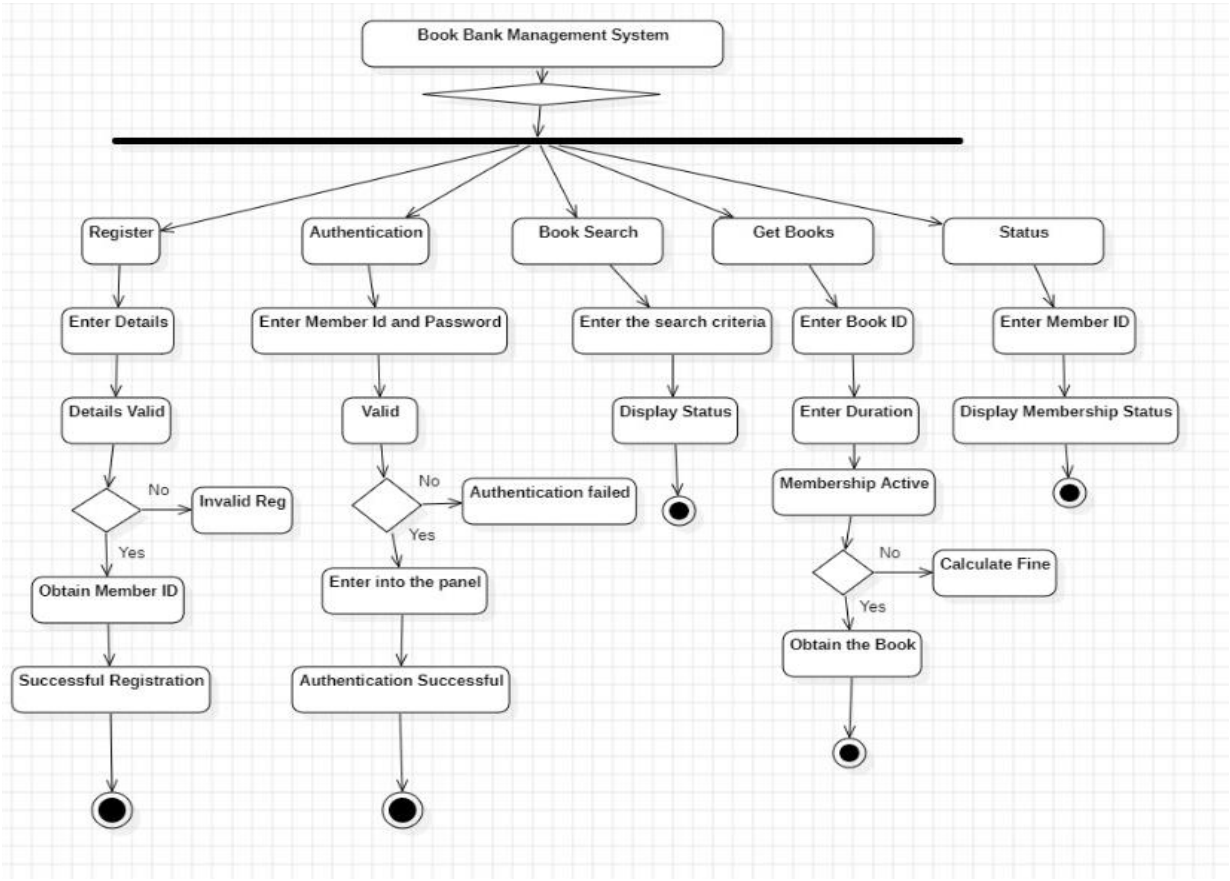
7. **Final State or End State:**

The state that the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

8. **Time Event:**

This refers to an event that stops the flow for a time; an hourglass depicts it. We can have a scenario where an event takes some time to complete.

ACTIVITY DIAGRAM FOR BOOK BANK SYSTEM:



RESULT:

Thus to draw State Chart diagram and Activity diagram for Book Bank System was done successfully.

EX NO:7 DATE:	Implement the system as per the detailed design

AIM:

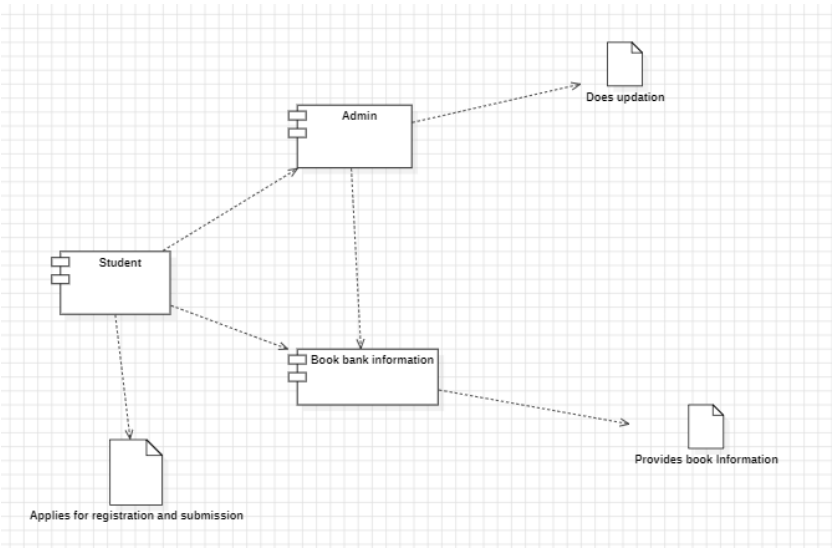
To implement Book Bank System as per the detailed design.

INTRODUCTION:

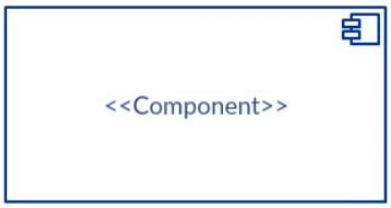
The Book Bank System is a software application designed to manage the borrowing and lending of books in a library setting. This document outlines the detailed implementation plan for developing the Book Bank System. The implementation plan covers various aspects including technology stack, development steps, coding guidelines, documentation, version control, security, CI/CD, and performance optimization.

COMPONENT DIAGRAM:

A component diagram is a type of UML (Unified Modeling Language) diagram used to illustrate the structure of a software system by showing the system's components and their relationships. It helps in visualizing the modular structure of a system and the interactions between its components.



COMPONENT DIAGRAM NOTATION:

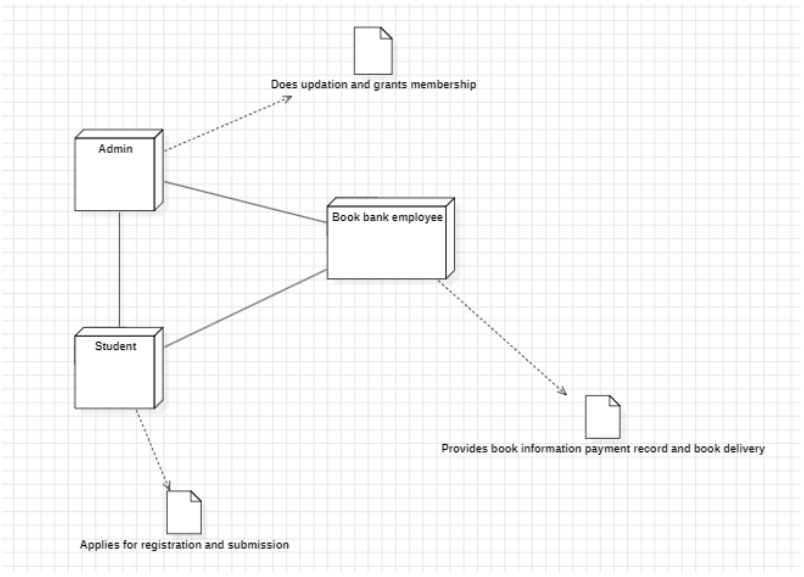


1. **Component:**
   Rectangle with the component icon in the top right corner and the name of the component.
2. **Dependencies:**

Although you can show more detail about the relationship between two components using the ball-and-socket notation (provided interface and required interface), you can just as well use a dependency arrow to show the relationship between two components.

DEPLOYMENT DIAGRAM:

A deployment diagram is a type of UML (Unified Modeling Language) diagram that illustrates the physical deployment of software components in a system, showing how software artifacts are deployed onto hardware nodes and interconnected through networks. It depicts the configuration and arrangement of system elements such as servers, computers, devices, and communication channels, providing insight into the system's deployment architecture and infrastructure.



DEPLOYMENT DIAGRAM NOTATION:

1. **Component:**

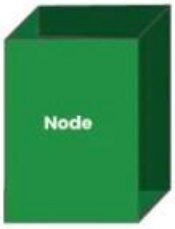
A component represents a modular and reusable part of a system, typically implemented as a software module, class, or package. It encapsulates its behaviour and data and can be deployed independently.

2. **Artifact:**

An artifact represents a physical piece of information or data that is used or produced in the software development process. It can include source code files, executables, documents, libraries, configuration files, or any other tangible item.

3. **Node:**

A node represents a physical or computational resource, such as a hardware device, server, workstation, or computing resource, on which software components can be deployed or executed.



Code generation from class diagram:

Registration:

```
import java.util.*;
public class registration {
  public registration() {
  }
  public void Attribute1;
  public void Name;
  public void Age;
  public void Address;
  public void College;
  public void Year of Study;
  public void Contact no;
  public void Member Id;
  public void UserName;
  public void Password;
  public void register() {
  }}
```

Authentication:

```
import java.util.*;
public class Authentication {
    public Authentication() {
      }
```

```
        public void Username;
        public void Password;
        public void login() {
        }}
```

Book Search:

```
    import java.util.*;
    public class Book Search {
      public Book Search() {
      }
      public void Title;
      public void Author;
      public void Publications;
      public void ISBN;
      public void search() {
}}
```

Get Books:

```
    import java.util.*;
    public class get books {
      public get books() {
      }
      public void Book ID;
      public void Member ID;
      public void Duration;
```

```
public void get() {
}
public void Check_Status() {
}}
```

Membership Status:

```
import java.util.*;
public class membership status {
  public membership status() {
  }
  public void Status;
  public void Last Renewed;
  public void Book Holding;
  public void Next Renewed;
  public void Return_Status() {
  } }
```

RESULT:

Thus the book bank management system was implemented successfully.

EX.NO:8 DATE:	**Test the software system for all scenarios identified as per the usecase diagram**

AIM:

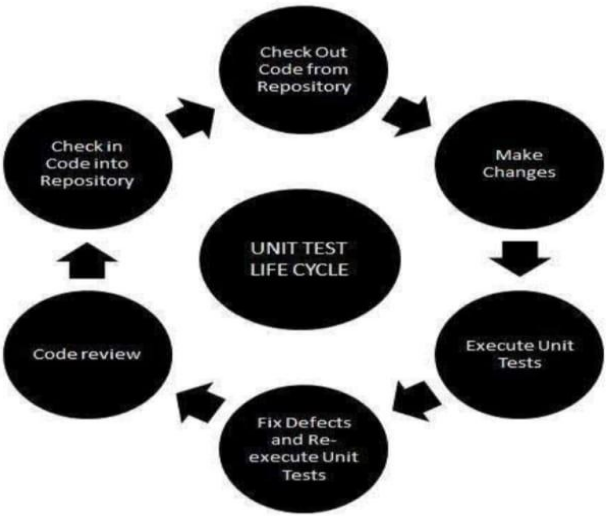
To test the software system for all scenarios identified in the use case diagram.

BPO MANAGEMENT SYSTEM:

In the realm of outsourcing, effective management of business processes is critical for operational efficiency and client satisfaction. A BPO management system streamlines the process of handling client requests, assigning tasks to employees, ensuring quality assurance, and facilitating client communication.

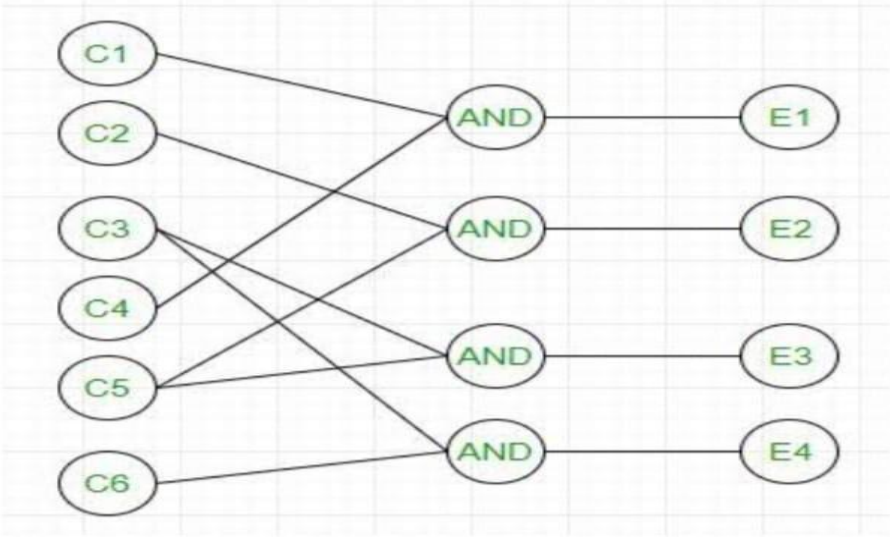
Unit Testing:

- Unit testing involves testing individual components or modules of the software to ensurethey function correctly in isolation.
- Each module, such as client management, project management, employee management, etc.,would undergo unit testing.
- **Example:** In the book bank management system, unit tests could be written to validate functions like adding a new book, updating user information, or calculating overdue fees.



Black box Testing:

- Black box testing is a technique where the internal workings of the system are not known tothe tester. The tester only tests the system's functionality based on its specifications.
- Testers would input various sets of data into the system and verify that the expected output isproduced.
- **Example:** In the book bank management system, black box testing could involve testing the borrowing process by providing inputs like member credentials and book selections and verifying that the correct books are borrowed.

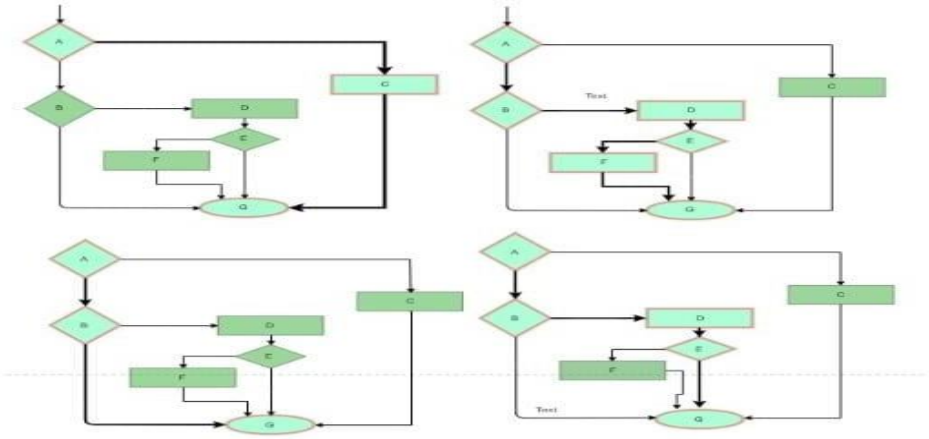


		1	2	3	4
CAUSES	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	1
EFFECTS	E1	X	-	-	-
	E2	-	X	-	-
	E3	-	-	X	-
	E4	-	-	-	X

White Box Testing:

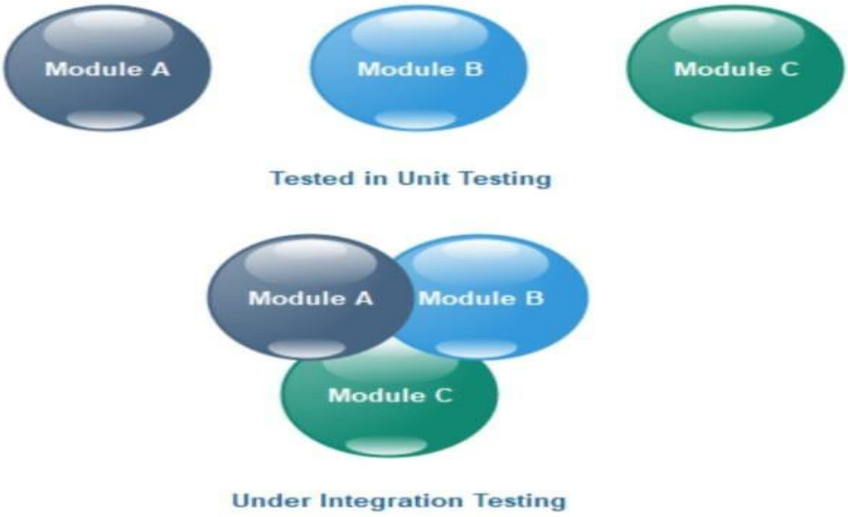
- White box testing involves testing the internal logic and structure of the software code.

- Testers would examine the code of individual modules to ensure that all code paths aretested.
- **Example:** White box testing for the book bank management system could involve testing different branches and conditions within functions responsible for book management, such as checking for valid inputs and error handling.



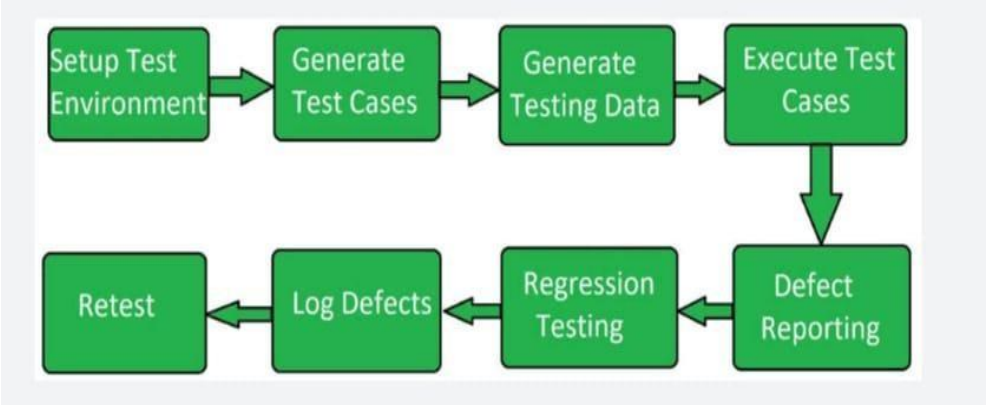
Integration Testing:

- Integration testing verifies that different modules of the software work together as expected.
- Testers would test the interaction between different modules, such as client management,project management, and employee management.
- **Example:** In the book bank management system, integration testing could involve testing the interaction between the user interface, database, and backend logic, while system testing could involve testing scenarios such as borrowing and returning books, searching for books, and updating user information.



System Testing:

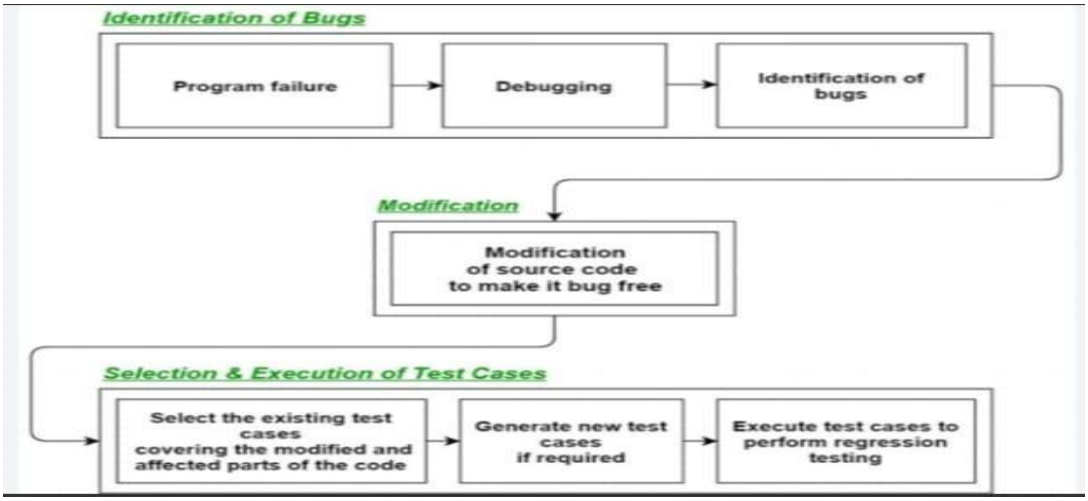
- System testing involves testing the entire system as a whole to verify that it meets thespecified requirements.
- Testers would perform end-to-end testing of the entire system, including all modules andtheir interactions.
- Example: Testing the entire process from client negotiation to project delivery and paymentto ensure that it functions as expected.



Regression Testing:

- Regression testing ensures that new changes or additions to the system do not adverselyaffect existing functionalities.
- Testers would re-run previously conducted tests after new changes or additions are made tothe system to ensure that existing functionalities are not affected.

- **Example:** Regression testing for the book bank management system could involve re-testing functionalities like borrowing, returning, and searching for books after implementing new features or bug fixes.



RESULT:

Thus the software system for all scenarios identified in the use case diagram was tested successfully.

EX NO:9 DATE:	**Improve the reusability and maintainability of the software system by applying appropriate design patterns**

AIM:

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

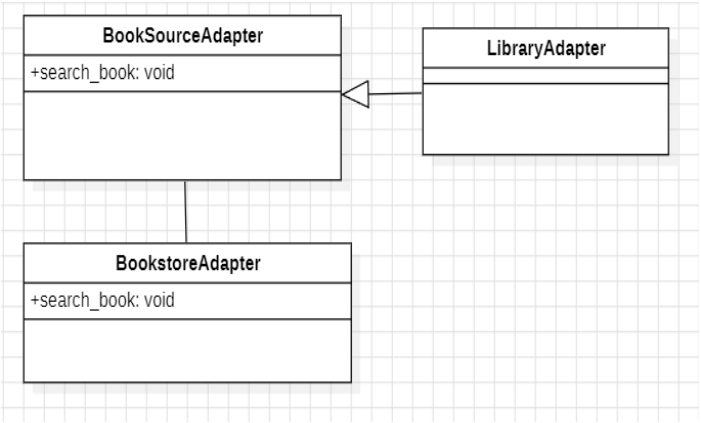
DESIGN PATTERNS:

1. Adapter Pattern:

Context/Problem: When you have two incompatible interfaces that need to work together.

Solution: Create an adapter that acts as a bridge between the two interfaces, allowing them to collaborate without modifying their existing code.

Visual:

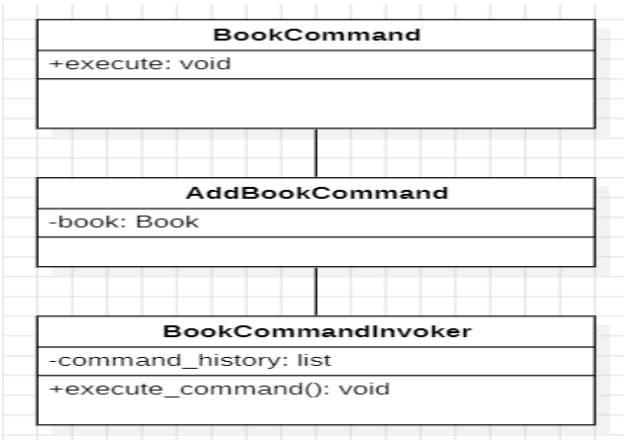


2.Command Pattern:

Context/Problem: When you need to decouple the sender of a request from its receiver, or when you want to parameterize objects with requests.

Solution: Encapsulate a request as an object, allowing clients to parameterize objects with queues, requests, and operations. This enables the sender to be decoupled from the receiver and promotes the ability to execute requests in a configurable manner.

Visual:

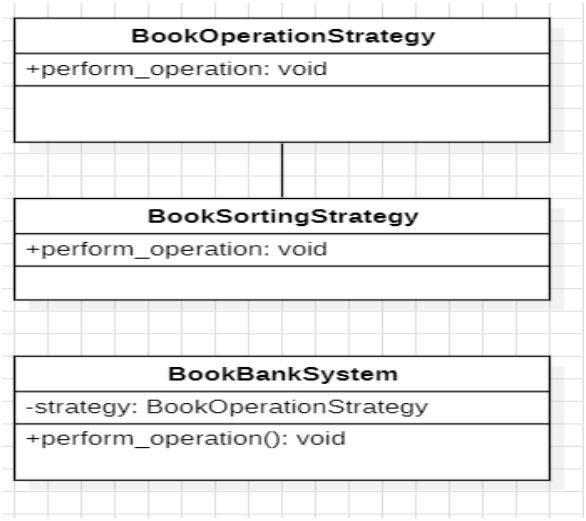


3.Strategy Pattern:

Context/Problem: When you have a family of algorithms and want to make them interchangeable.

Solution: Define each algorithm as an object and encapsulate them within their own classes. Then, make these algorithms interchangeable by having a context class that can switch between them dynamically.

Visual:

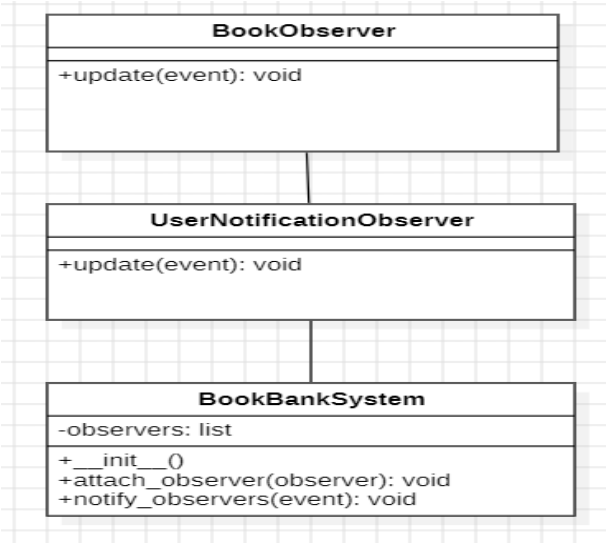


4.Observer Pattern:

Context/Problem: When you need to establish a one-to-many dependency between objects, ensuring that when one object changes state, all its dependents are notified and updated automatically.

Solution: Define a subject interface for objects that need to be observed, and observer interfaces for objects that need to observe changes. Whenever the subject's state changes, it notifies all its observers.

Visual:

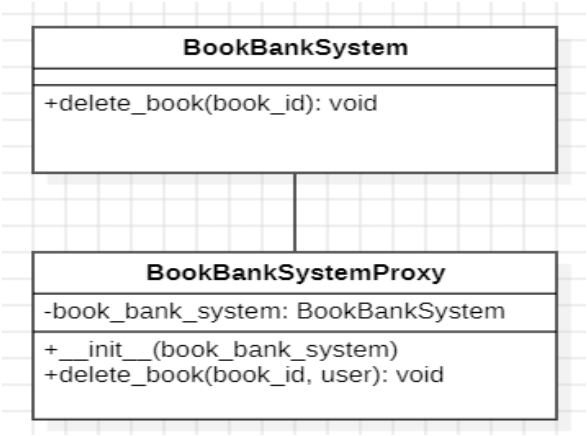


5.Proxy Pattern:

Context/Problem: When you want to control access to an object or add additional functionality to it without changing its code.

Solution: Create a proxy object that acts as a placeholder for the original object. The proxy controls access to the original object and can add additional behavior before or after forwarding requests to it.

Visual:

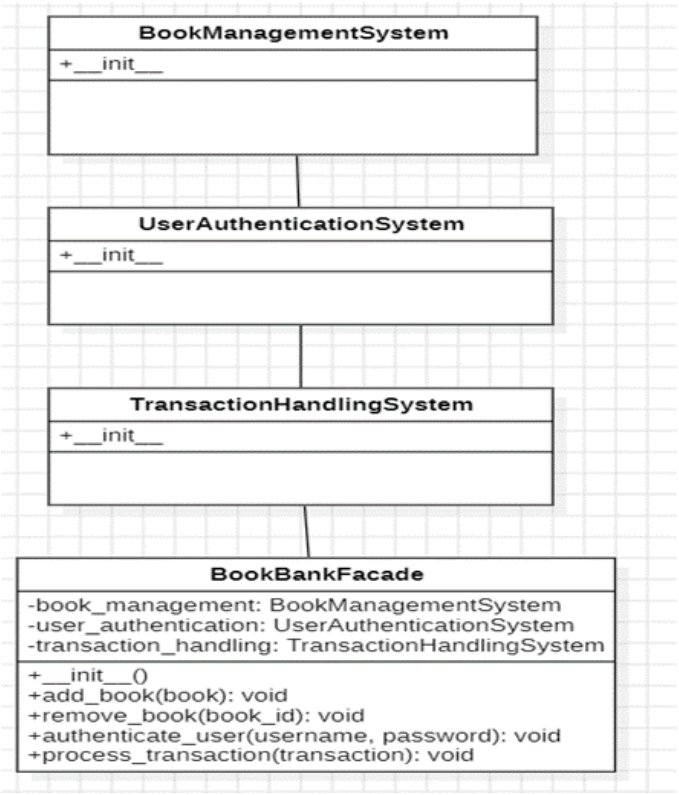


6.Facade Pattern:

Context/Problem: When you want to provide a simplified interface to a complex subsystem or set of interfaces.

Solution: Create a facade class that provides a unified interface to the subsystem. Clients interact with the facade, which in turn delegates requests to the appropriate objects within the subsystem. This hides the complexity of the subsystem and makes it easier to use.

Visual:



RESULT:

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