LIBRARY MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

Pragnay Komakula -22BCA30006

Submitted to

Mrs. Suman Acharya

in partial fulfillment for the award of the degree of

BACHELOR OF COMPUTER APPLICATIONS (BCA)

IN

UNIVERSITY INISTITUE OF COMUTING

Chandigarh University

May 2025

TABLE CONTENTS

ABSTRACT **3**

Introduction **4**

Class Diagram **5**

Interface **6**

Actors **7**

Design Flow / Process **12**

Results / Validation **14**

Conclusion **16**

ABSTRACT

This project presents a **Library Management System** developed using the principles of *Object Oriented Analysis and Design (OOAD)*, as part of the CS309 course. The system emphasizes clean architectural design, entity decoupling, and minimal GUI code, featuring a **console-based interface**. Designed to streamline the management of library resources and user interactions, the system supports core functionalities required by different user roles such as **Borrower**, **Checkout Clerk**, **Librarian**, and **Administrator**.

The use case-driven approach ensures that the system meets real-world requirements effectively. Borrowers can search for books, place hold requests, and view their borrowed items, while checkout clerks and librarians manage circulation, user data, and item inventory. Administrators are empowered with account management and reporting capabilities.
A significant architectural enhancement involves the introduction of the **HoldRequestOperations** class, which removes the bidirectional dependency between HoldRequest and Book, thereby improving modularity and maintainability.

The project includes a detailed **Class Diagram** viewable via **StarUML** and a defined **Database Schema** based on Java DB (Derby). The development environment relies on **Java SE Development Kit 8 (JDK 8)** and **NetBeans IDE**, ensuring smooth integration of logic and database components. This system provides a robust foundation for managing libraries efficiently, demonstrating a strong application of OOAD principles such as encapsulation, modularity, and responsibility-driven design.

Introduction

The effective management of library operations is essential for ensuring seamless access to books and information resources. Traditional library systems often face challenges such as manual tracking, limited accessibility, and inefficient handling of borrower data. To address these issues, this project presents a **Library Management System** developed using the principles of **Object Oriented Analysis and Design (OOAD)**.

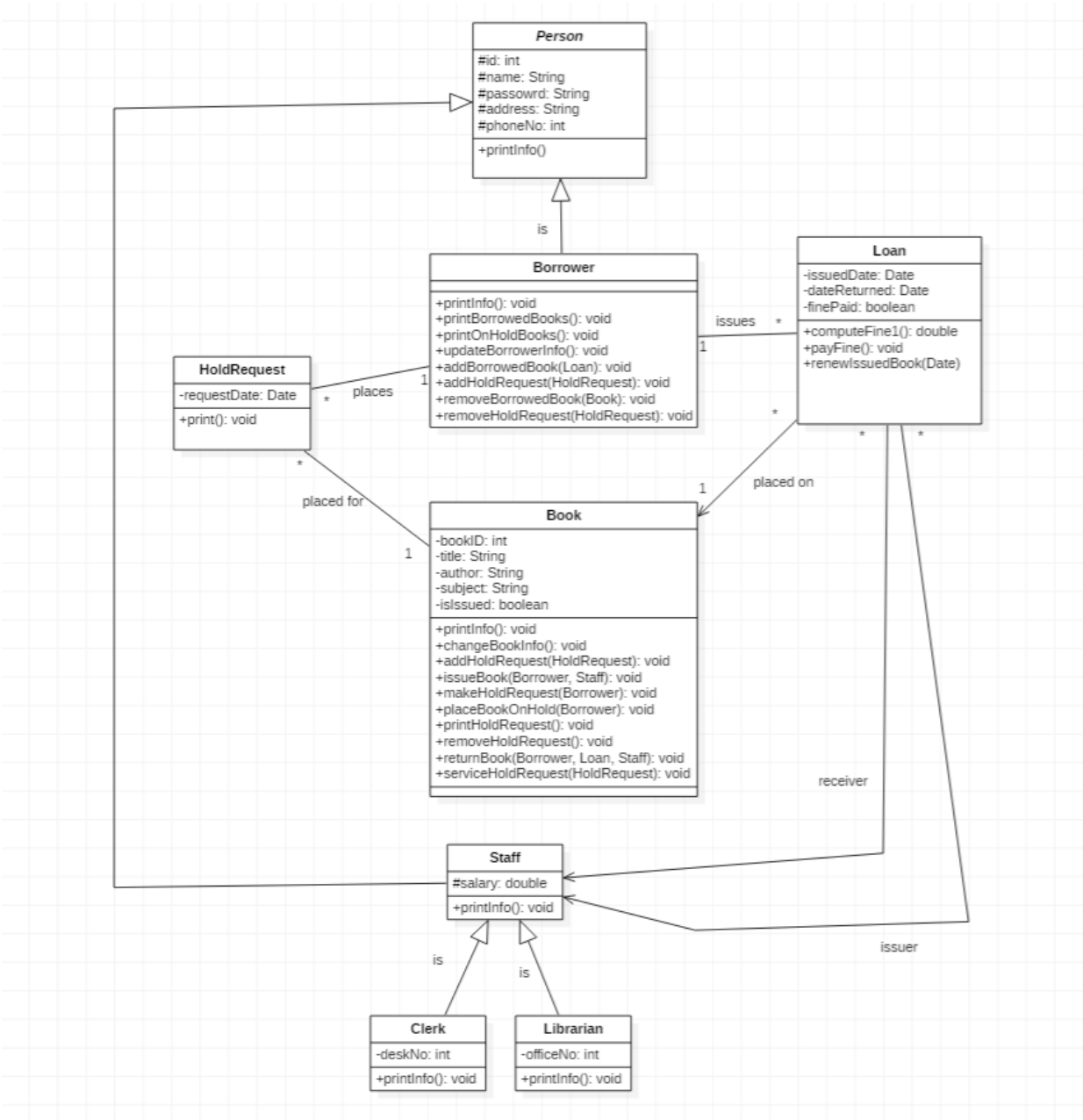
The system is designed with a **console-based interface** and focuses on clean architecture, modularity, and low coupling between components. By applying object-oriented principles, the system models real-world entities such as **Borrowers**, **Books**, **Clerks**, and **Librarians** into classes and interfaces that interact cohesively. A notable design decision includes the addition of the HoldRequestOperations class, which effectively decouples the HoldRequest and Book classes, improving the maintainability and scalability of the system.

Each user role in the system is associated with specific **use cases**, from book search and borrowing to administrative functions such as user account and inventory management. The backend of the system is supported by a **Java DB (Derby)** database, integrated through **NetBeans IDE** and **JDK 8**, which stores all necessary records securely.

This project not only addresses the practical needs of a library environment but also demonstrates how OOAD methodologies can be applied to create structured, reusable, and efficient software systems.

A Library Management System made using the concepts of Object Oriented Analysis and Design. Minimal Code is written in the GUI and the entities are decoupled as well. The interface is console based. This project was designed during the course "Object Oriented Analysis and Design CS309". The **Class Diagram** of the project is also provided along with the **Database Schema** file.

Class Diagram



Note: After Refactoring, new Class "HoldRequestOperations" is added to the above structure which lies in between the HoldRequest class and Book class. This class removes the bidirectional dependency between HoldRequest and Book.

Interface

```
------------------------------------------------------------
        Welcome to Library Management System
------------------------------------------------------------
Following Functionalities are available:

1- Login
2- Exit
3- Admininstrative Functions
------------------------------------------


Enter Choice:
3

Enter Password:
lib
```

```
------------------------------------------------------------
        Welcome to Admin's Portal
------------------------------------------------------------
Following Functionalities are available:

1- Add Clerk
2- Add Librarian
3- View Issued Books History
4- View All Books in Library
5- Logout
-------------------------------------------

Enter Choice:
1

Enter Name:
Haris
Enter Address:
Lahore
Enter Phone Number:
991
Enter Salary:
25000

Clerk with name Haris created successfully.

Your ID is : 1
Your Password is : 1

Press any key to continue..
```

Actors

The actors include the following:

- Librarian
- Checkout Clerk
- Borrower
- Administrator

Use Cases:

After determining the actors, the second step in use case analysis is to determine the tasks that each actor will need to do with the system. Each task is called a use case because it represents one particular way the system will be used.

In other words, only those use cases are listed that actors will need to do when they are using the system to solve the customer's problem.

Borrower:

- Search for items by title.
- by author.
- by subject.
- Place a book on hold if it is on loan to somebody else.
- Check the borrower's personal information and list of books currently borrowed.

Checkout Clerk:

- All the Borrower use cases, plus
- Check out an item for a borrower.
- Check in an item that has been returned.
- Renew an item.
- Record that a fine has been paid.
- Add a new borrower.
- Update a borrower's personal information (address, telephone number etc.).

Librarian:

- All of the Borrower and Checkout Clerk use cases, plus
- Add a new item to the collection.
- Delete an item from the collection.
- Change the information the system has recorded about an item.

Administrator:

- Add Clerk.
- Add Librarian.
- View Issued Books History.
- View All Books in Library.

How to Run

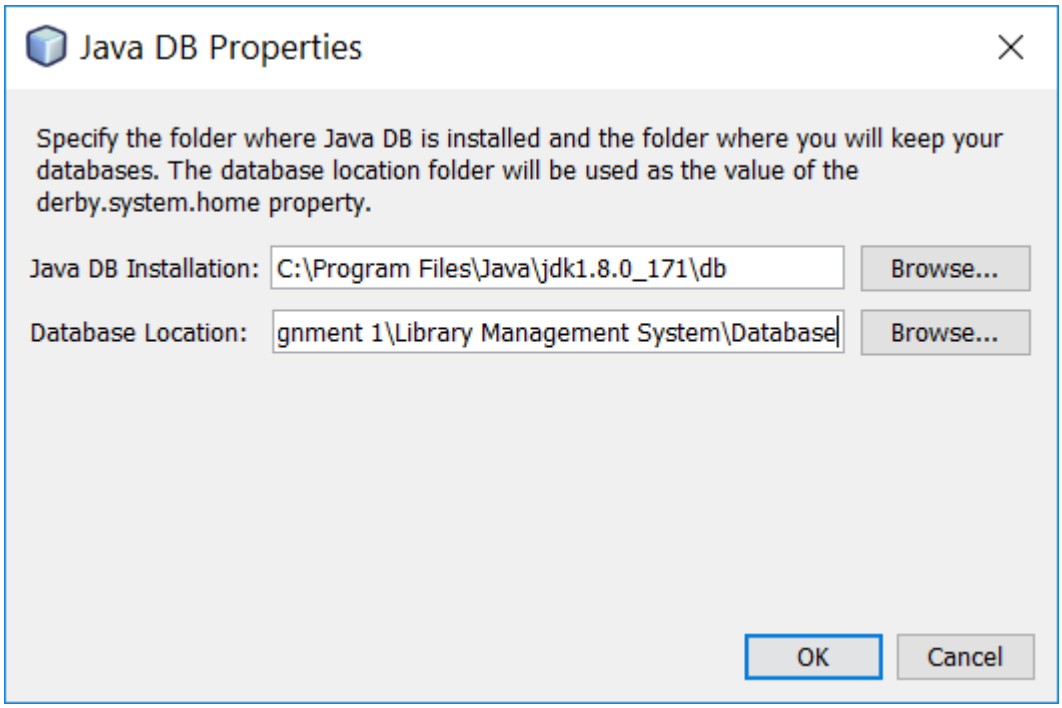
1- Install these:

- Java SE Development Kit 8 (JDK 8)
- After installing JDK 8, install NetBeans IDE

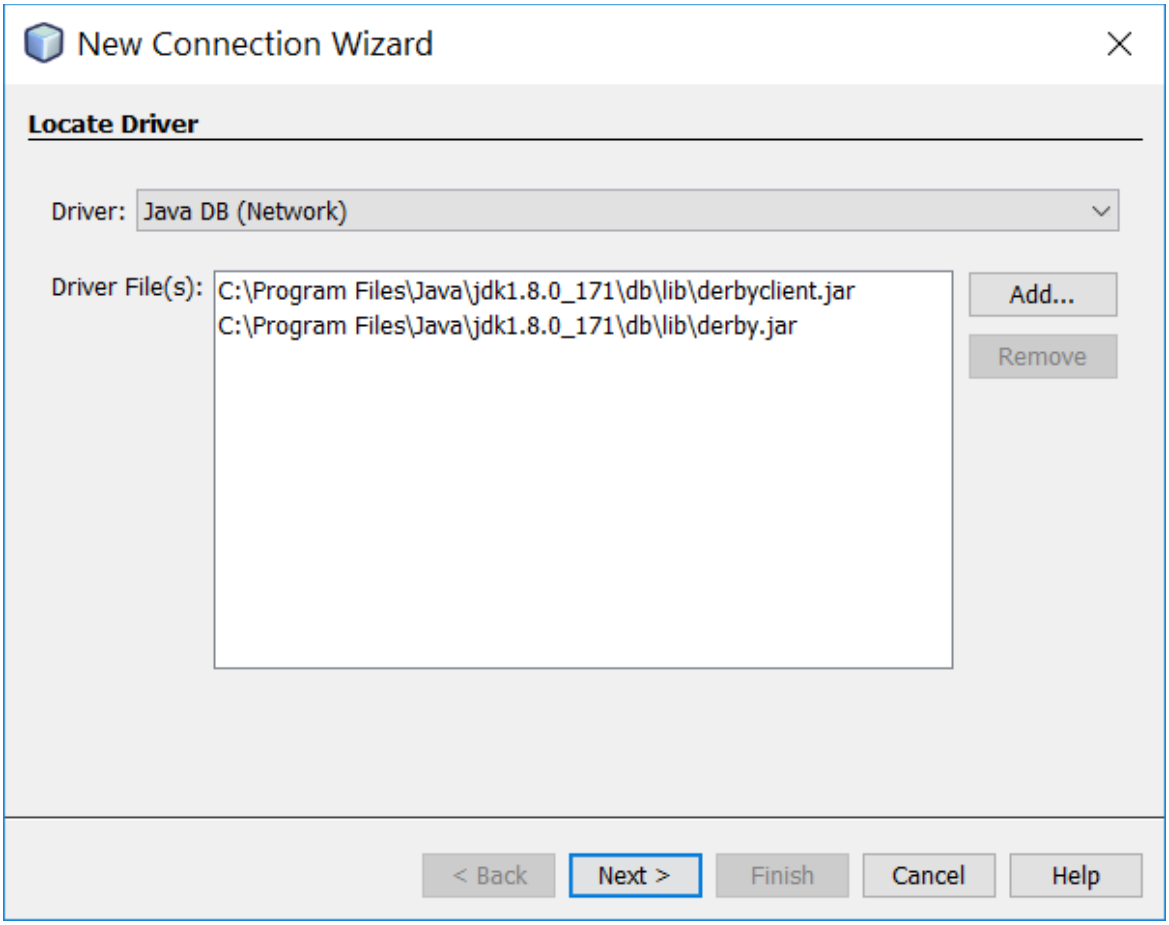
2- Open NetBeans IDE. Click on File -> Open Project and browse to the downloaded folder named "Project" and select it. It will load the NetBeans project.

3- Now everything is setup except the Java DB (Derby) Database of NetBeans. So, follow these steps to setup the database:

Step 1: In the Netbeans Window, there is a tab named "Services" on the left. Select it. Then right click on JavaDB > Properties and change database location to "Database" folder downloaded with this repository (its placed besides the "Project" folder).



Step 2: After that a database named LMS will show up under JavaDB tab. Now Right Click Databases > New Connection and select Java DB Network and click Next.



Step 3: Provide the following database crendentials in the next popup and click Next.

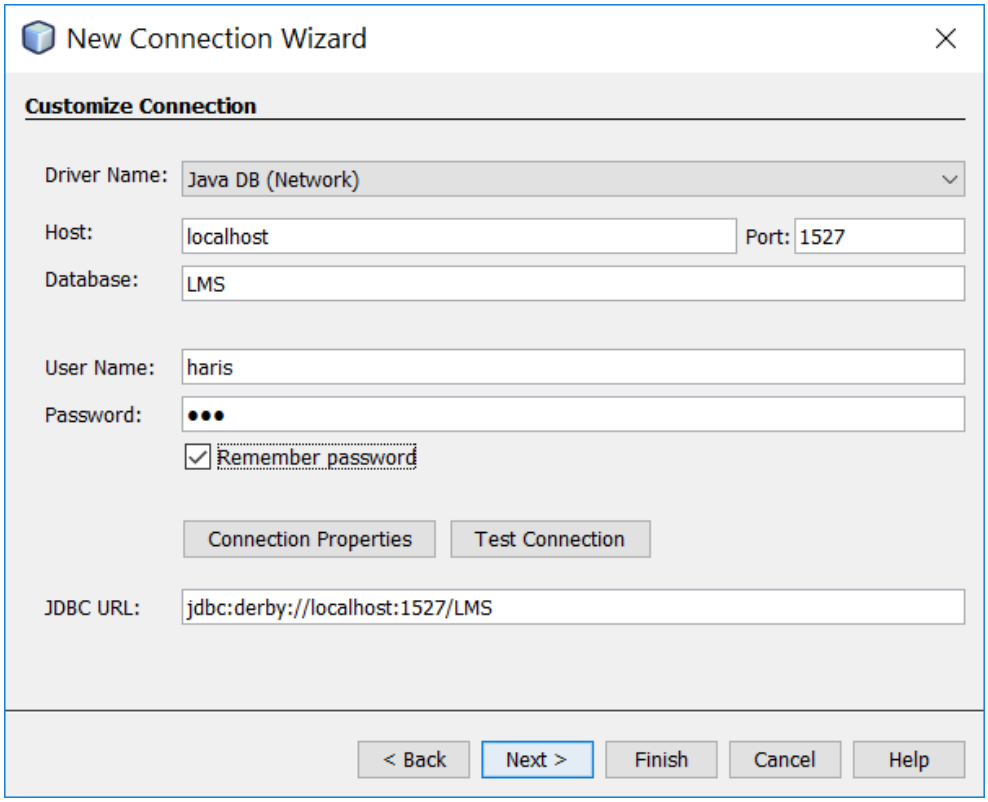
Host: localhost

Port: 1527

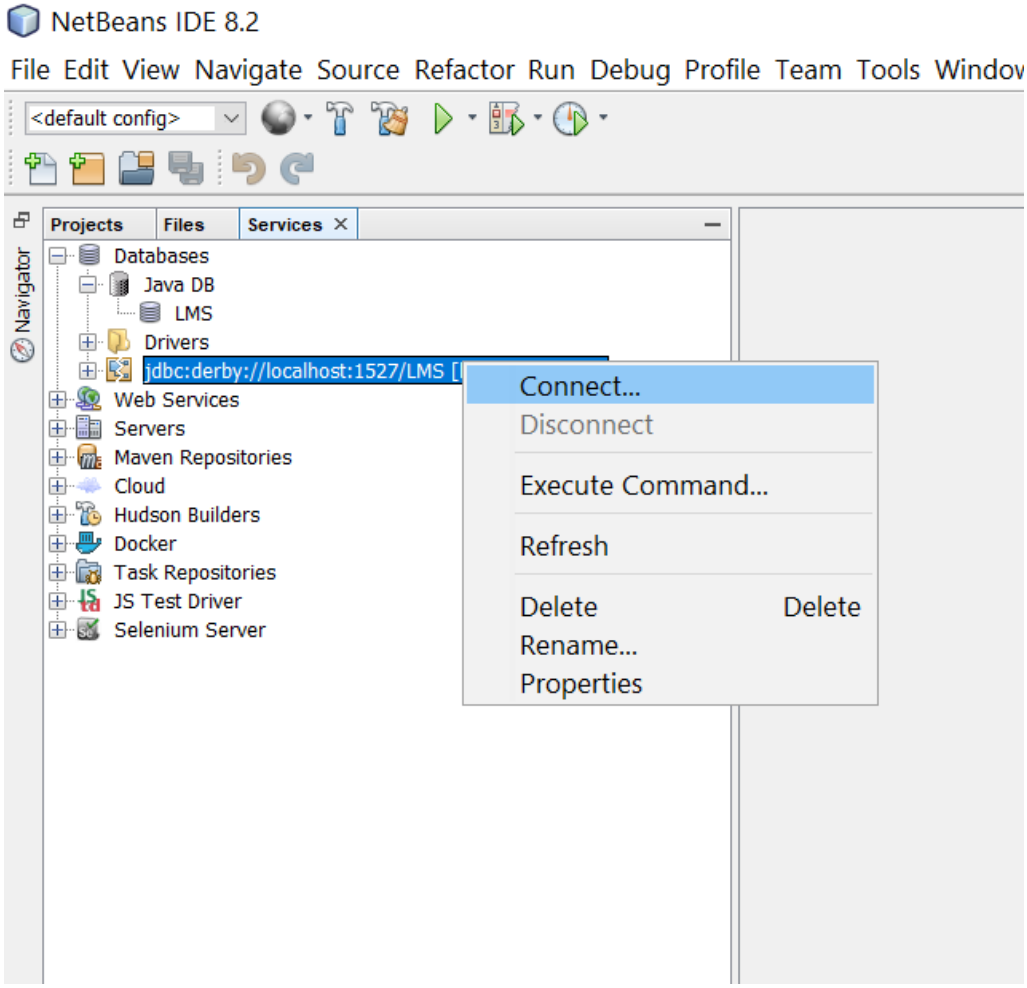
Database: LMS

User Name: haris

Password: 123



Step 4: Now just click Next for the rest of the windows. After all this the database connection is made. Make sure that you connect with the database before running the project by right clicking on the connection and selecting connect. Now you are ready to run the project.



Note

The password for Administrative Functions is *lib*. The admin adds new clerks and librarian, then they both do the rest of the functions.

Design Flow / Process

The development of the **Library Management System** followed a structured and iterative approach grounded in **Object Oriented Analysis and Design (OOAD)** principles. The design process can be outlined in the following key stages:

1. Requirement Analysis

The first step involved identifying system requirements and user needs. The main stakeholders (actors) such as **Borrower, Checkout Clerk, Librarian**, and **Administrator** were analyzed to understand their interactions with the system. Each actor's responsibilities were mapped into distinct **use cases** to define system functionality.

2. Use Case Modeling

Each actor's interactions were translated into use cases such as:

- Borrower: Search books, place hold requests, view personal data.
- Clerk: Check in/out items, renew, add borrower, update info.
- Librarian: Manage books, view borrower/clerk info.
- Administrator: Add staff, view reports/history.

These use cases helped in identifying required operations and relationships between system components.

3. Class Identification and Responsibility Assignment

Core entities were identified as **classes**, including:

- Person, Borrower, Clerk, Librarian
- Book, Loan, HoldRequest
- Library, Database, etc.

Responsibilities were distributed among these classes using the **Responsibility-Driven Design** approach, ensuring single responsibility per class and low coupling between modules.

4. Class Diagram Design

A detailed **class diagram** was created (using **StarUML**) to visually represent relationships such as inheritance, associations, and dependencies between classes. This diagram served as the backbone of the system architecture.

Additionally, a new class HoldRequestOperations was introduced during refactoring to **decouple** Book and HoldRequest, resolving bidirectional dependency and improving the system's modularity.

5. Database Schema Design

A relational database was modeled using **Java DB (Derby)**. The schema consisted of tables for users, books, loans, and hold requests. Primary and foreign keys were carefully assigned to preserve data integrity and support efficient queries.

6. Implementation

The project was implemented in **Java**, using **NetBeans IDE**. All core logic was embedded in backend classes with **minimal GUI code**, maintaining a **console-based interface** to keep the design clean and focused on core logic.

7. Testing and Debugging

Each functionality was tested using sample data. Edge cases such as invalid logins, duplicate librarian entries, and overdue returns were considered. Errors were logged and resolved to ensure system robustness.

8. Refinement and Finalization

After testing, the codebase was refactored to improve readability and structure. Unused dependencies were removed, input validations were strengthened, and the interface flow was polished for usability.

This systematic design flow allowed for the development of a robust, maintainable, and extensible Library Management System that meets both functional and design expectations.

Results / Validation

The functionality of the **Library Management System** was validated through rigorous testing against the system's use cases and requirements. The goal was to ensure that each feature works correctly for all defined actors (Borrower, Clerk, Librarian, Administrator) and handles both normal and exceptional scenarios gracefully.

Functional Results

Use Case	Expected Behavior	Validation Result
Book Search	Search by title, author, or subject returns matching results.	Successfully returns accurate results.
Place Hold Request	Only allowed if the book is currently checked out. Request is saved with timestamp.	Works as intended; recorded in system with correct association.
Check Out / Check In	Books can be issued to and returned by borrowers.	Transaction logged and borrower’s list updated.
Renew Book	Book issue is extended if not on hold by someone else.	Validated, extension granted appropriately.
Add / Update Borrower Info	New borrowers are saved; existing data can be updated.	Fully functional; reflects in database and output.
Book Management (Librarian)	Librarians can add, remove, or update book records.	All operations performed successfully.
Admin Functions	Admin can add staff and view reports. Password-protected.	Correct access control and functionality.
Login & Role Handling	Users are logged in based on credentials and directed to relevant portals.	Accurate role-based access confirmed.

Exception Handling & Edge Case Validation

- **Invalid Input Handling**: All menu choices and data entries are validated to avoid crashes or logic failures.
- **Duplicate Librarian Check**: Ensures only one librarian can exist in the system at a time.
- **Hold Request Expiry**: Validated by assigning expiry duration via system settings.
- **Database Connectivity**: Verified successful connection and data persistence through Derby DB.

Final Outcome

- All primary and secondary use cases were successfully implemented and tested.
- Console output confirmed correct flow of data and system response.
- The system supports extensibility and clean separation of concerns due to OOAD-based design.
- Data persistence and retrieval through the **Java DB (Derby)** database performed reliably.

The system met all of its functional requirements and design goals. Its performance and correctness validate the strength of the object-oriented design approach used during development.

Conclusion

The **Library Management System** developed as part of the Object Oriented Analysis and Design course demonstrates the practical application of OOAD principles to solve real-world problems efficiently. Through the use of object-oriented design techniques, the system achieves modularity, reusability, and scalability, which are essential for long-term software maintenance and evolution.

The clear separation of responsibilities among classes, minimal GUI logic, and proper use of inheritance and encapsulation reflect a robust design approach. The console-based interface ensures simplicity and easy interaction for all user roles, while the integration with a Java DB (Derby) database provides persistent and reliable data storage.

The inclusion of multiple actors such as Borrowers, Clerks, Librarians, and Administrators, each with distinct use cases, showcases how complex workflows can be broken down and handled using role-based access and structured logic.
Moreover, the addition of the HoldRequestOperations class during the refactoring phase further improves the architecture by reducing bidirectional dependencies and enhancing decoupling.

Overall, the project not only fulfills the functional requirements of a library management system but also serves as a strong example of how effective software engineering practices can lead to clean, maintainable, and extensible applications.