Object-Oriented Programming

Project Report

Library Management System

Student Name: Mukund Kumra

Lecturer Name: Dr. Adrielle Moraes

Project: Library Management System (Console Application)

Module: Objection Oriented Programming - I

Table of Contents

Introduction

This project implements a console-based Library Management System application is written in Java 21. The application demonstrates both fundamental and advanced OOP concepts required by the Objection Oriented Programming I Project. It allows different types of library members to list, search, borrow, and return books, while admins can add or remove items. The system provides a clear separation between user interface and core logic, with emphasis on maintainability, immutability, defensive programming, and modern Java features such as records, lambdas, pattern matching, and custom exceptions.

The project focuses not only on building a functional system but demonstrating clear understanding of modern Java design principles, good encapsulation practices, and correct use of inheritance and polymorphism.

The application aims on creating a clean separation between the user interface (LibraryApp) and core logic (Library), ensuring the design is modular, maintainable, and fully aligned with the requirements of the assignment brief.

Reason for choosing This Domain

I chose this domain (Library Management System) as libraries are familiar, intuitive, and easy to map to object-oriented structures, making ideal for demonstrating the full range of Java fundamentals and advanced techniques. The domain includes:

Entities: Books, Members, Admins.

Behaviours: Borrow, Return, Search, List.

Rules/Constraints: Due dates, Availability, Borrow Limits.

Hierarchies: Student -> Staff -> Admin.

Immutable identifiers: ISBN numbers.

State transitions: Available -> Borrowed -> Returned.

This domain is simple to understand, easy to test through console input, and rich enough to demonstrate features like records, pattern matching, lambdas (Predicate filtering), custom exception design, and defensive copying. Due to being relatable and frequently used in the computer science curriculum, this domain avoids forced complexity, allowing focus on clean, realistic software design.

User stories Implemented

To guide the design of the Library Management System, a set of user stories was created so that the functionality would reflect realistic user goals rather than just technical features. These stories shaped the structure of the menus and ensured each class existed for a clear purpose.

- **US1 - List All Items:** As a user, I want to view all books so that I can explore the available catalogue.

- **US2 - Search Items by Title:** As a user, I want to search for books by title so that I can quickly find specific items.

- **US3 - Borrow a Book:** As a user, I want to borrow a book so that I can take it home to read.

- **US4 - Return a book:** As a user, I want to return a book so that it becomes available for others.

- **US5 - List active loans**: As a user or admin, I want to view active loan records so that I can track borrowed items.

- **US6 - Admin add book**: As an admin, I want to add a new book to the library so that the catalogue stays up-to-date.

- **US7 - Admin remove book:** As an admin, I want to delete a book so that unavailable or outdated items can be removed.

System Overview & Architecture

Upon starting the application, the user is presented with a textual menu allowing them to choose from different option based on them being a staff/student or an admin. Based on the selected option, different functionalities are unlocked. For example:

- A StudentMember or StaffMember can borrow and return books.
- An AdminMember can additionally add or remove items.

The system internally manages lists of users, book items, and active loan records. When a member borrows a book, the program creates a new LoanRecord, updates the book's availability, and stores the transaction. Returning a book reverses this process, calculates whether the item is overdue, and removes the corresponding record.

The system follows a layered architecture, keeping UI and logic separate, improving maintainability and making the system easier to extend:

1. UI Layer

LibraryApp handles:

- Console I/O
- Menu system
- Input validation
- Calls to Library methods

It contains no domain logic, following separation of concerns.

2. Domain / Logic Layer

Library manages:

- Collections of LibraryItem, LibraryUser, and LoanRecord
- Borrow/return logic
- Searching and filtering
- Admin-only operations
- Defensive copying of provided lists

3. Domain Model

Key classes:

- **LibraryItem (abstract)**
    - Common properties: title, genre, availability
    - Parent for specific types of items
- **BookItem extends LibraryItem**
    - Contains Isbn immutable type
    - Represents borrowable items
- **LibraryUser (sealed)**
    - Parent for all users
    - Allows polymorphic storage and behaviour
- **AdminMember, StudentMember, StaffMember**
    - Different access levels
    - Demonstrates inheritance and polymorphism
- **LoanRecord (Java record)**
    - Immutable representation of an active loan
    - Stores user, item, borrow date, calculated due date
- **Isbn (custom immutable type)**
    - Validates and stores ISBN
    - Ensures identity consistency

UML / Class relationships

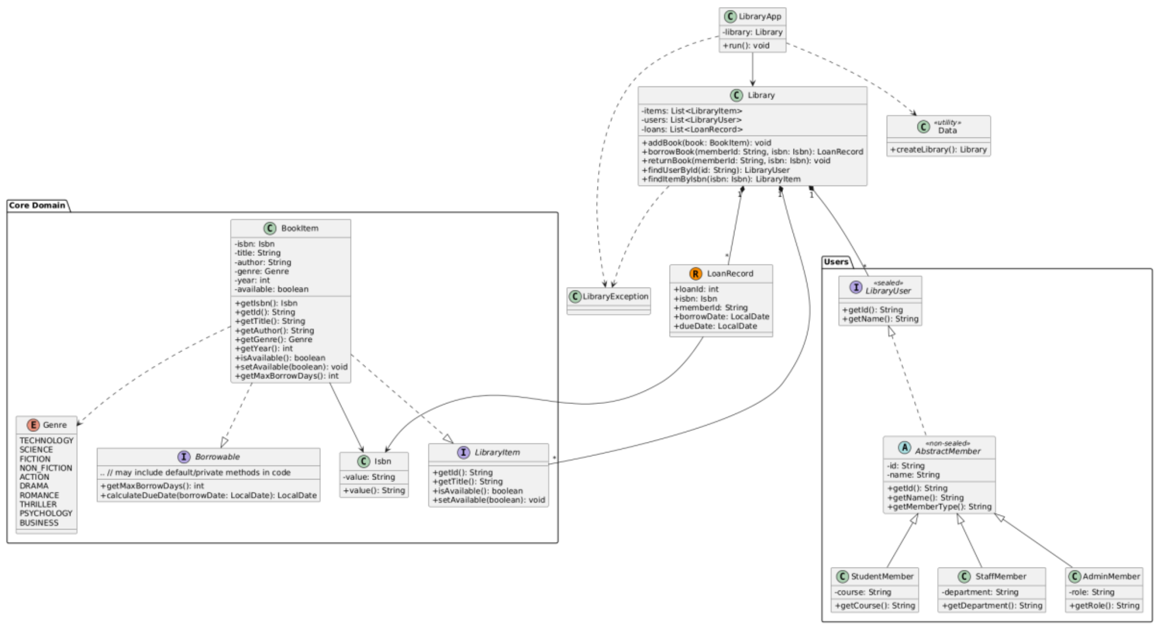

Figure 1: UML Diagram of The Library Management System

This diagram shows how all classes, interfaces, enums, and records in the Library Management System relate to one another, illustrating inheritance, associations, and implemented behaviours. On the user side of the system, the hierarchy is centred around the LibraryUser interface, which defines the shared attributes and behaviours required for all users, such as retrieving a user's ID and name. This interface is implemented by the abstract class AbstractMember, which provides the common fields (id and name) and shared functionality for different types of members. The three concrete subclasses: StudentMember, StaffMember, and AdminMember extend this abstraction to represent increasingly specialised roles within the system. AdminMember in particular provides elevated authority for tasks such as adding or removing books, demonstrating how the system supports distinct permission levels through inheritance and polymorphism.

The item hierarchy is structured around the LibraryItem interface, which defines the essential operations that all library items must support, including retrieving an item's title or availability state. BookItem is the concrete implementation of this interface, and it also implements the Borrowable interface to provide borrowing-specific behaviour such as due-date calculation and availability management. BookItem composes an Isbn object, a custom immutable type that ensures consistent identification of books, and it uses the Genre enum to classify items into fixed literary categories. Active loans are represented through LoanRecord, a Java record that stores immutable loan details such as the ISBN, the borrowing member's ID, the borrow date, and the calculated due date.

All core library operations including borrowing, returning, searching, adding books, and managing users are coordinated by the Library class. This class maintains associations with collections of BookItem, LibraryUser, and LoanRecord objects, and it provides the central business logic needed to enforce borrowing rules, locate items or users, validate input, and update item availability. It also interacts with custom exceptions to handle invalid commands or borrowing attempts when items are unavailable.

Finally, the primary entry point of the system is LibraryApp, which acts as the console-based interface. It is responsible for handling user input, interacting with the Data class to initialise a pre-populated library, and delegating all menu-driven operations to the Library class. Through LibraryApp, every workflow from item browsing and searching to borrowing processes, admin actions, and loan summaries is orchestrated, making it the central connection between users and the underlying system logic.

Data Flow Chart

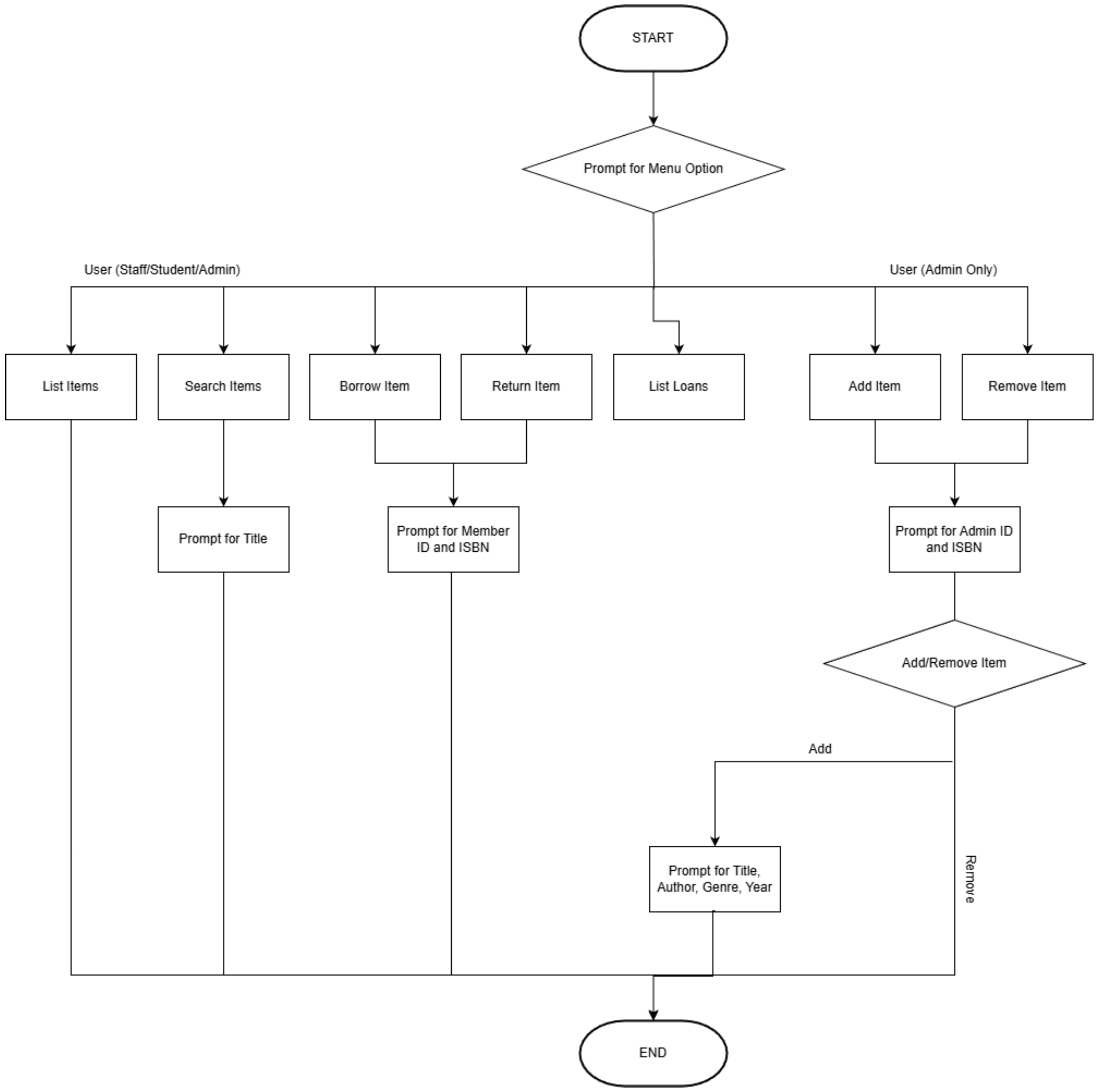

Figure 2: Data Flow Chart of The Library Management System from draw.io

This flowchart illustrates the main operational flow of the Library Management System, detailing how user input moves through the program and how the system responds to different menu selections. When the application starts, the user is presented with a menu of available actions. The first decision point determines which option the user has selected this may be a general operation available to all users (StudentMember, StaffMember, or AdminMember) or an administrative operation restricted exclusively to AdminMember accounts.

For general users, the system provides several core functions: listing all items, searching for items, borrowing a book, returning a book, or listing all loan records. Depending on the chosen option, the system prompts the user for additional information. For instance, selecting Search Items leads to an input prompt for a book title, whereas borrowing or returning a book requires the user to enter both their member ID and the ISBN of the item involved. The interactions shown in the flowchart highlight the structured sequence the program follows: first collecting input, then performing lookups and validations in the Library class, and finally displaying the outcome to the user.

Administrative functions follow a slightly different path. If the user selects either Add Item or Remove Item, the system first prompts for an admin ID and the ISBN of the item to be modified. A decision is then made based on whether the goal is to add or remove an item. When adding a new book, the program requests additional details such as title, author, genre, and publication year. When removing an item, the system attempts to delete the specified book based on the ISBN, handling exceptions if the item does not exist or cannot be removed due to active loans.

Once the selected action has been processed whether it involves listing, searching, borrowing, returning, adding, or removing the flowchart concludes at the END node. This visual layout provides a clear representation of how input moves through the system, how decisions guide the program’s behaviour, and how the LibraryApp class coordinates all interactions with the Library logic. It accurately reflects the procedural flow implemented in the console-based interface and demonstrates how the system handles both standard user and administrator operations.

Key Java / OOP features used

Fundamentals

The Library Management System demonstrates strong adherence to object orientation. Items, users, and loan records all map cleanly to classes or records. The AbstractMember class acts as a parent for StudentMember, StaffMember, and AdminMember, allowing common attributes (ID and name) to be reused through inheritance. Each subclass uses super() to initialize inherited fields properly.

Encapsulation is maintained through private fields and controlled access via getters. Method overloading appears in search functionality, and constructor chaining simplifies initialisation. The system uses clear class responsibilities: LibraryApp for UI, Library for logic, and BookItem for item behaviour, creating a clean, maintainable structure.

Interfaces and Polymorphism

The project includes multiple interfaces, notably LibraryItem, Borrowable, and LibraryUser. These interfaces allow different classes to be treated polymorphically, for example, Library can store all users as LibraryUser types regardless of their concrete subclass.

Borrowable demonstrates behaviour sharing by allowing any implementing class to define its own due-date calculation. This showcases how polymorphism allows loosely coupled, extensible designs across item types.

Enums, Arrays, Collections, and API Usage

The Genre enum provides meaningful categorisation for books, giving each item a clearly defined type. Java Collections are used extensively throughout the program: ArrayList stores items, users, and loan records, enabling dynamic resizing and efficient lookups.

The Date and Time API (LocalDate) supports borrow/return operations, due-date calculation, and comparison for overdue items. String manipulation uses Java’s core API, and defensive copying is used in the Library constructor to avoid shared-mutable-state issues from seeded data.

Advanced Java Features

The system incorporates several of the advanced features required by the project assignment:

- Records (LoanRecord) to represent immutable loan data
- Custom immutable type (Isbn) for safe book identification
- Pattern matching with instanceof in user validation
- Lambdas and Predicates in search functions
- Method references for streamlined iteration
- Defensive copying to protect internal collections
- Custom checked and unchecked exceptions for error handling

Collectively, these features demonstrate modern Java design practices and align strongly with the module learning outcomes.

Evaluation & Challenges

Developing the Library Management System was a valuable learning experience that allowed me to apply the module’s concepts in a real-world style scenario. The program meets the assignment requirements, but the development included several challenges that deepened my understanding of Java.

One early issue involved maintaining consistent behaviour among LibraryUser subclasses. Ensuring that StudentMember, StaffMember, and AdminMember all correctly inherited shared fields required refining constructors and confirming that overridden behaviours functioned correctly.

Search functionality presented another challenge. Implementing filtering using Predicates and lambdas required careful handling of null input and edge cases. Through these difficulties, I developed a stronger intuition for functional programming concepts in Java.

Validation and exception handling were also a source of complexity. The system needed to respond safely to invalid menu selections, missing ISBNs, unavailable books, and improper returns. Implementing custom exceptions significantly improved robustness and taught me the importance of graceful error handling in user-facing applications.

Overall, the system is stable, well-structured, and demonstrates strong use of OOP principles, Java APIs, and advanced language features.

Strengths and Weaknesses

Strengths

- Strong design separation: UI vs domain
- Use of modern Java (records, pattern matching)
- Well-structured OOP hierarchy
- Clean and readable code
- Consistent use of immutability where appropriate

Challenges

- **No persistent storage**
  Data resets with each run.
- **User validation currently depends on seeded data**
  Requires future dynamic user creation.
- **Console input handling limitations**
  Try/catch blocks mitigate issues but UI could be more robust.
- **Time constraints**
  Reduced ability to implement GUI or more advanced features.

Version Control

The project was tracked using Git and kept in a public GitHub repository to facilitate development and keep an accurate record of progress. I was able to maintain a clear history of modifications, create backups of the codebase, and make frequent contributions as features were added by using Git. The repository shows regular updates throughout development and includes the complete source code, diagrams, and project report. This directly satisfies the assignment's requirements for additional marks for using a public repository. The repository's link is:

https://github.com/mukundkumra/librarymanagementsystem

Future improvements

Although the library management system satisfies project requirements, it can be improved in numerous ways in the future. Some possible improvements may look like:

- Persistent storage using JSON, JDBC or text-based saving
- Graphical UI (JavaFX)
- Better validation and input sanitisation
- Additional item types (Journals, Magazines, DVDs)
- Unit testing (JUnit)
- Reporting features such as overdue lists

These additions would further extend the system and make it more realistic for real-world deployment.

Presentation Video Link

The video presentation for the project can be viewed here:

Java Project Presentation Video

Conclusion

In conclusion, the Library Management System effectively demonstrates the full range of object-oriented and advanced Java features required for this module. It incorporates inheritance, interfaces, polymorphism, enums, encapsulation, records, lambdas, custom exceptions, and robust API usage. Throughout development, challenges in input handling, inheritance design, and functional programming deepened my understanding of Java and improved the program's structure and reliability.

The project meets the assignment requirements, provides a solid architectural foundation, and leaves room for future improvements such as persistent storage and graphical user interfaces.