## 

## **PROJECT DEFINITION — LIBRARY MANAGEMENT SYSTEM**

### **🎯 Objective:**

Design a clean, modular, and simulation-ready **Library Management System** that manages **physical books**, **user accounts**, **borrowing/returning of books**, and **fine calculation**. The system supports **Admin**, **Librarian**, and **Registered User** roles, and is intended to be simulated via a **console-based C++ application** using object-oriented design.

### **💼 User Roles:**

* **Registered User**: Can borrow and return books, pay fines, and view their history.
* **Librarian**: Can issue/accept returns and manage book records.
* **Admin**: Can manage users and configure system settings.

### **📌 Core Functional Features:**

#### **1. User Management**

* User registration and login
* Role-based access control
* Update/view user profile

#### **2. Book Management**

* Search and view book information
* Admin/Librarian can add, update, and delete book records
* Track availability status of books (Available, Borrowed)

#### **3. Borrowing & Returning**

* Registered users can borrow/return books
* System sets due dates and checks for overdue status
* Maximum borrow limit is enforced
* Borrowing restricted if user has unpaid fines

#### **4. Fine Management**

* Overdue fines are auto-calculated
* Users must pay fines before borrowing again

#### **5. System Configuration**

* Admin can set borrow limit and fine rate

## **USE CASE DIAGRAM**

### **👥 Actors:**

1. **Registered User**
2. **Librarian**
3. **Admin**

### **🎯 Use Cases:**

#### **📘 Common to All Users:**

* **Login**
* **Search Book**
* **View Book Details**
* **Logout**

#### **👤 Registered User:**

* **Borrow Book**
  + <<include>> Check Borrowing Eligibility
  + <<include>> Generate Borrow Transaction
  + <<include>> Update Book Availability
  + <<include>> Set Due Date
* **Return Book**
  + <<include>> Generate Return Transaction
  + <<include>> Check for Overdue
  + <<extend>> Generate Fine
* **Pay Fine**
  + <<include>> Check Outstanding Fines
  + <<include>> Update Fine Status
* **View Borrowing History**

#### **🧑‍🏫 Librarian:**

* **Issue Book**
  + <<include>> Verify User
  + <<include>> Update Book Status
* **Accept Return**
  + <<include>> Update Book Status
  + <<include>> Update Transaction Record
* **Add Book**
* **Update Book**
* **Remove Book**

#### **🛠️ Admin:**

* **Add User**
  + <<include>> Validate User Input
* **Remove User**
* **Modify User Role**
* **Add Book**
* **Update Book**
* **Remove Book**
* **Configure System Settings**
  + <<include>> Set Borrow Limit
  + <<include>> Set Fine Rate

### **🔄 <<include>> Relationships:**

| **Use Case** | **<<includes>>** |
| --- | --- |
| Borrow Book | Check Borrowing Eligibility, Generate Borrow Transaction, Update Book Availability, Set Due Date |
| Return Book | Generate Return Transaction, Check for Overdue |
| Pay Fine | Check Outstanding Fines, Update Fine Status |
| Issue Book | Verify User, Update Book Status |
| Accept Return | Update Book Status, Update Transaction Record |
| Add User | Validate User Input |
| Configure System Settings | Set Borrow Limit, Set Fine Rate |

### **🔁 <<extend>> Relationships:**

| **Use Case** | **<<extends>>** |
| --- | --- |
| Return Book | Generate Fine (only if overdue) |

### **📝 Notes for Drawing:**

* Group common use cases centrally.
* Place actors on the outer sides: Registered User (left), Librarian (middle), Admin (right).
* Use solid lines to link actors to use cases.
* Use dashed arrows labeled <<include>> or <<extend>> between use cases.

## **ADVANCED CLASS DIAGRAM — LIBRARY MANAGEMENT SYSTEM**

## **I. Classes, Attributes, and Methods**

### **1. Class: User (Abstract)**

**Attributes (private):** • userID: string (unique)  
 • name: string  
 • email: string (unique)  
 • password: string

**Methods (public):** • login(email: string, password: string): bool  
 • logout(): void  
 • updateProfile(data: map<string, string>): void  
 • viewProfile(): void  
 • searchBook(query: string): vector<Book\*>  
 • resetPassword(): void

### **2. Class: RegisteredUser (extends User)**

**Attributes (private):** • currentBorrowedBooks: vector<Book\*>  
 • borrowingHistory: vector<Transaction\*>  
 • outstandingFine: double

**Methods (public):** • borrowBook(book: Book\*): bool  
 • returnBook(book: Book\*): bool  
 • payFine(amount: double): bool  
 • viewBorrowingHistory(): vector<Transaction\*>  
 • getFineAmount(): double

### **3. Class: Librarian (extends User)**

**Attributes (private):** • employeeID: string (unique)

**Methods (public):** • issueBook(book: Book\*, user: RegisteredUser\*): Transaction\*  
 • acceptReturn(book: Book\*, user: RegisteredUser\*): Transaction\*  
 • addBook(book: Book\*): void  
 • updateBook(book: Book\*): void  
 • removeBook(book: Book\*): void

### **4. Class: Admin (extends User)**

**Attributes (private):** • adminID: string (unique)  
 • rolePermissions: vector<string>

**Methods (public):** • addUser(user: User\*): void  
 • removeUser(user: User\*): void  
 • modifyUserRole(user: User\*, newRole: string): void  
 • addBook(book: Book\*): void  
 • updateBook(book: Book\*): void  
 • removeBook(book: Book\*): void  
 • configureSystem(settings: map<string, string>): void

### **5. Class: Book**

**Attributes (private):** • bookID: string (unique)  
 • title: string  
 • author: string  
 • category: string  
 • isbn: string (unique)  
 • status: enum {Available, Borrowed}

**Methods (public):** • isAvailable(): bool  
 • getDetails(): string  
 • updateStatus(newStatus: enum): void

### **6. Class: Transaction**

**Attributes (private):** • transactionID: string (unique)  
 • user: RegisteredUser\*  
 • book: Book\*  
 • transactionType: enum {Borrow, Return}  
 • transactionDate: date  
 • dueDate: date  
 • returnDate: date  
 • fineAmount: double

**Methods (public):** • calculateFine(currentDate: date): double  
 • isOverdue(currentDate: date): bool  
 • closeTransaction(returnDate: date): void

### **7. Class: Fine**

**Attributes (private):** • fineID: string (unique)  
 • transaction: Transaction\*  
 • amount: double  
 • paid: bool

**Methods (public):** • pay(): bool  
 • isPaid(): bool  
 • getAmount(): double

### **8. Class: LibrarySystem (Singleton)**

**Attributes (private):** • users: vector<User\*>  
 • books: vector<Book\*>  
 • transactions: vector<Transaction\*>  
 • fines: vector<Fine\*>  
 • maxBorrowLimit: int  
 • fineRatePerDay: double

**Methods (public):** • getInstance(): LibrarySystem&  
 • searchBooks(query: string): vector<Book\*>  
 • getUserByID(userID: string): User\*  
 • recordTransaction(transaction: Transaction\*): void  
 • calculateFine(user: RegisteredUser\*): double  
 • setFineRate(rate: double): void  
 • setBorrowLimit(limit: int): void

## **II. Relationships & Cardinalities (Pairwise)**

| **Class A** | **Relationship Type** | **Class B** | **Cardinality** |
| --- | --- | --- | --- |
| User | Inheritance | RegisteredUser | 1 → 1 |
| User | Inheritance | Librarian | 1 → 1 |
| User | Inheritance | Admin | 1 → 1 |
| RegisteredUser | Association | Book | 1 → 0..\* (currentBorrowedBooks) |
| RegisteredUser | Association | Transaction | 1 → 0..\* (borrowingHistory) |
| Transaction | Association | Book | 1 → 1 |
| Transaction | Association | RegisteredUser | 1 → 1 |
| Fine | Association | Transaction | 1 → 1 |
| Admin | Aggregation | User | 1 → 0..\* |
| Librarian/Admin | Association | Book | 1 → 0..\* (for management) |
| LibrarySystem | Aggregation | All entities | 1 → 0..\* |

## **III. Constraints and Business Rules**

1. **Unique Identifiers:** All IDs (userID, bookID, transactionID, fineID) must be unique.
2. **Role-Based Access:**
   * **RegisteredUser**: Can borrow/return books and pay fines
   * **Librarian**: Can issue/accept books and manage inventory
   * **Admin**: Can manage users, configure system, and manage books
3. **Borrowing Rules:**
   * Max books: defined by maxBorrowLimit
   * Due date = Borrow date + 14 days (default)
   * Cannot borrow if:  
     + Max books borrowed
     + Outstanding fine not paid
4. **Fine Rules:**
   * fineAmount = overdueDays × fineRatePerDay
   * Must pay fine to continue borrowing
5. **Security:**
   * Passwords encrypted (simulated)
   * Role-based function access enforced by LibrarySystem

## **IV. UML Notation (Text Format Conventions)**

* + = public
* - = private
* # = protected
* **Inheritance**: Solid line with hollow triangle
* **Aggregation**: Hollow diamond
* **Association**: Solid line with multiplicity (e.g., 1 → 0..\*)
* **Singleton**: getInstance() method in LibrarySystem

## **ADVANCED STATE MODELING DIAGRAM — LIBRARY MANAGEMENT SYSTEM**

## **I. Entities & States**

### **1. State Model: Book**

**States:**

1. **Available** *(Initial)*
2. **Borrowed**
3. **Overdue**
4. **Returned**
5. **Removed** *(Final)*

**Transitions (pairwise):**

* **Available → Borrowed** • Event: User borrows book  
   • Guard: Book is available AND user eligible  
   • Action: Record transaction, update availability
* **Borrowed → Overdue** • Event: Due date passed  
   • Guard: Book not yet returned  
   • Action: Mark as overdue, apply fine
* **Borrowed/Overdue → Returned** • Event: Book returned  
   • Action: Close transaction, update status
* **Returned → Available** • Event: Book re-shelved  
   • Action: Mark as available
* **Any State → Removed** • Event: Admin or Librarian removes book  
   • Action: Mark as removed, delete from catalog

### **2. State Model: Transaction**

**States:**

1. **Initiated** *(Initial)*
2. **Active**
3. **Overdue**
4. **Closed** *(Final)*

**Transitions (pairwise):**

* **Initiated → Active** • Event: Book issued  
   • Action: Set borrow & due date
* **Active → Overdue** • Event: Due date passed  
   • Action: Calculate fine
* **Active/Overdue → Closed** • Event: Book returned  
   • Action: Record return date, finalize fine

### **3. State Model: Fine**

**States:**

1. **Unpaid** *(Initial)*
2. **Partially Paid**
3. **Paid** *(Final)*

**Transitions (pairwise):**

* **Unpaid → Partially Paid** • Event: User pays partial amount  
   • Action: Deduct paid amount, update status
* **Unpaid/Partially Paid → Paid** • Event: Full fine amount paid  
   • Guard: Amount due == 0  
   • Action: Mark fine as paid

### **4. State Model: RegisteredUser Session**

**States:**

1. **Logged Out** *(Initial)*
2. **Logging In**
3. **Logged In**
4. **Session Expired / Logged Out (Final)**

**Transitions (pairwise):**

* **Logged Out → Logging In** • Event: User submits credentials  
   • Action: Authenticate
* **Logging In → Logged In** • Event: Successful login  
   • Action: Create session
* **Logging In → Logged Out** • Event: Login failed  
   • Action: Reject access
* **Logged In → Session Expired** • Event: Inactivity timeout or logout  
   • Action: Clear session

### **5. State Model: User Fine Access Eligibility**

**States:**

1. **Eligible** *(Initial)*
2. **Blocked**
3. **Eligible Again**

**Transitions (pairwise):**

* **Eligible → Blocked** • Event: Outstanding fine exists  
   • Guard: fineAmount > 0  
   • Action: Restrict borrowing
* **Blocked → Eligible Again** • Event: Fine fully paid  
   • Guard: fineAmount == 0  
   • Action: Restore borrowing privileges

## **II. Entry / Exit / Do Actions**

| **Entity** | **State** | **Entry Action** | **Exit Action** | **Do (During) Action** |
| --- | --- | --- | --- | --- |
| Book | Borrowed | Set status to Borrowed | N/A | Countdown to due date |
| Book | Overdue | Set overdue flag | Clear overdue flag (on return) | Accumulate fine |
| Transaction | Overdue | Calculate fine | Close transaction | Monitor late days |
| Fine | Partially Paid | Update payment record | N/A | Await next payment |
| User | Logged In | Create session | Invalidate session | Accept commands |
| User | Blocked | Restrict borrow access | Enable access | Display fine reminder (UI) |

## **III. Guard Conditions and Triggers**

| **Trigger** | **Condition (Guard)** | **Target State** |
| --- | --- | --- |
| Borrow Request | User eligible AND Book is Available | Book → Borrowed |
| Return Request | Valid transaction exists | Transaction → Closed |
| Due Date Passed | CurrentDate > DueDate | Transaction → Overdue |
| Fine Payment | fineAmount == 0 | Fine → Paid |
| Session Timeout | inactivityPeriod > allowedThreshold | User → Logged Out |
| Login Attempt | credentials are valid | User → Logged In |

## **IV. Notes on Diagram Layout (for drawing)**

* Group state diagrams for **Book**, **Transaction**, **Fine**, and **User** on separate swimlanes
* Use rounded rectangles for states
* Use arrows with labels for transitions
* Add [guard] and /action to transition arrows
* Mark initial (●) and final (◉) states clearly

## **COMPLETE SYSTEM ARCHITECTURE — LIBRARY MANAGEMENT SYSTEM (For State Modeling)**

This structure is:

* 🔗 **Loosely Coupled**
* 🧩 **Modular**
* ⚙️ **Easy to simulate in C++**
* 📐 **Supports all UML diagrams**
* 🧼 **Clean and reservation/report-free**

## **📦 I. MODULE OVERVIEW (Packages)**

| **Module Name** | **Responsibility** | **Main Classes** |
| --- | --- | --- |
| UserModule | Manages all users and sessions | User, RegisteredUser, Librarian, Admin |
| BookModule | Manages book records and search | Book |
| TransactionModule | Manages borrow/return transactions | Transaction |
| FineModule | Calculates and manages overdue fines | Fine |
| SystemModule | Central coordinator, enforces rules and connections | LibrarySystem |

## **🔗 II. MODULE INTERACTIONS (Connectivity)**

lua

CopyEdit

UserModule ---> TransactionModule ---> FineModule

| | |

v v v

BookModule <-------- LibrarySystem <------- Admin Actions

### **✅ Connection Details:**

#### **🔹 UserModule**

* Handles registration, login, session
* Calls LibrarySystem.searchBook()
* Calls LibrarySystem.borrowBook() → handled by TransactionModule
* Calls LibrarySystem.returnBook() → handled by TransactionModule

#### **🔹 BookModule**

* Maintains inventory (add, update, delete, getAvailability)
* Controlled by Librarian and Admin
* Queried by all users through LibrarySystem.searchBook()

#### **🔹 TransactionModule**

* Creates a Transaction when a book is borrowed or returned
* Checks due dates, sets status (borrowed, overdue, returned)
* Communicates with BookModule to update status
* Calls FineModule to apply overdue fine if needed

#### **🔹 FineModule**

* Calculates fine based on return delays
* Keeps track of paid/unpaid fines
* Enforces fine block via LibrarySystem.checkEligibility()

#### **🔹 SystemModule (LibrarySystem)**

* Singleton controller
* Central interface to:  
  + Search books
  + Process borrow/return
  + Track user sessions
  + Check fine and borrow limits
  + Manage users and books via Admin
* Aggregates access to all sub-modules (Books, Users, Transactions, Fines)

## **🧩 III. CLASS TO MODULE MAPPING**

| **Class** | **Belongs To Module** |
| --- | --- |
| User | UserModule |
| RegisteredUser | UserModule |
| Librarian | UserModule |
| Admin | UserModule |
| Book | BookModule |
| Transaction | TransactionModule |
| Fine | FineModule |
| LibrarySystem | SystemModule |

## **🧠 IV. WORKFLOW EXAMPLES**

### **🔸 Borrow Book (RegisteredUser)**

pgsql

CopyEdit

RegisteredUser.login() → UserModule

LibrarySystem.borrowBook(bookID) → SystemModule

→ Check user eligibility → TransactionModule + FineModule

→ Check book availability → BookModule

→ Create Transaction → TransactionModule

→ Update book status → BookModule

### **🔸 Return Book (RegisteredUser)**

pgsql

CopyEdit

LibrarySystem.returnBook(bookID) → SystemModule

→ Lookup transaction → TransactionModule

→ Check for overdue → TransactionModule

→ Calculate fine if needed → FineModule

→ Update book status → BookModule

→ Close transaction → TransactionModule

### **🔸 Admin Adds User**

pgsql

CopyEdit

Admin.login() → UserModule

LibrarySystem.addUser(userObj) → SystemModule

→ Store user → UserModule

### **🔸 Librarian Adds Book**

scss

CopyEdit

Librarian.login() → UserModule

LibrarySystem.addBook(bookObj) → SystemModule

→ Store book → BookModule

## **🖥️ V. C++ Implementation Notes**

* Each **module = header + cpp file pair** (e.g., UserModule.h/cpp)
* Use **LibrarySystem** as a singleton coordinator
* Use std::vector, std::map, or fstream for in-memory/file-based storage
* Optional: use an enum for roles, transaction types, statuses

## **📐 VI. UML Diagram Mapping**

| **UML Diagram** | **Mapped From This Structure** |
| --- | --- |
| Use Case Diagram | From use case text list |
| Class Diagram | From previous structured class diagram |
| Package Diagram | From the module table above |
| Sequence Diagram | From the workflow examples |
| State Diagram | From previous state machine model |
| Activity Diagram | Can follow each workflow |
| Deployment Diagram | LibrarySystem as core service, files as storage |

## **DEPLOYMENT DIAGRAM — LIBRARY MANAGEMENT SYSTEM**

### **I. Nodes (Execution Environments)**

1. **Client Machine (Terminal/Console User)**
2. **Application Layer (C++ Executable)**
3. **Local Storage (File System / Database Simulation)**

### **II. Artifacts (Code Modules Deployed)**

* **user\_module.o** (User management logic)
* **book\_module.o** (Book inventory logic)
* **transaction\_module.o** (Borrow/return logic)
* **fine\_module.o** (Fine calculation logic)
* **system\_module.o** (Main controller – LibrarySystem)
* **main.exe** (C++ compiled program)

### **III. Data Files (as local DB substitutes)**

* users.dat – stores user info
* books.dat – stores book records
* transactions.dat – stores borrowing history
* fines.dat – stores fine records

### **IV. Deployment Diagram (Text Layout)**

plaintext

CopyEdit

+----------------------+

| Client Machine | ← Console I/O

|----------------------|

| User Interface (CLI)|

| main.exe |

+----------|-----------+

|

v

+----------------------+

| Application Layer | ← C++ App Logic

|----------------------|

| system\_module.o | (Singleton controller)

| user\_module.o |

| book\_module.o |

| transaction\_module.o|

| fine\_module.o |

+----------|-----------+

|

v

+----------------------+

| Local Storage | ← Simulated Data Store

|----------------------|

| users.dat |

| books.dat |

| transactions.dat |

| fines.dat |

+----------------------+

### **V. Communication (Connectors)**

* **main.exe → system\_module.o**: executes core system logic
* **system\_module.o → other modules (.o)**: uses functions and methods via direct function calls
* **Application Layer → Local Storage**: file I/O to simulate persistence (e.g., fstream)
* **User → CLI → main.exe**: direct command-line interaction (menus, prompts)

### **VI. UML Notation Mapping**

| **Component** | **UML Symbol** |
| --- | --- |
| Client Machine | Node |
| CLI App (main.exe) | Artifact on Node |
| .o modules | Artifacts inside a Node |
| .dat files | File artifacts (external resource) |
| Connectors | Solid lines with arrows |

### **✅ Summary:**

* **Single-node simulation architecture**
* All modules compiled together into one application
* Data is persisted using local .dat files
* Easily drawn in UML using 3 nodes and artifacts

## **PACKAGE DIAGRAM — LIBRARY MANAGEMENT SYSTEM**

### **I. Packages (Modules)**

Each package groups related classes and their responsibilities. These can map to C++ headers/namespaces/folders.

### **📦 1. user**

Handles user login, registration, role-specific actions.

**Classes:**

* User (abstract)
* RegisteredUser
* Librarian
* Admin

### **📦 2. book**

Manages the book catalog and search functionality.

**Classes:**

* Book

### **📦 3. transaction**

Handles borrow and return operations.

**Classes:**

* Transaction

### **📦 4. fine**

Calculates and manages overdue fines.

**Classes:**

* Fine

### **📦 5. system**

Central coordinator and facade between all modules.

**Classes:**

* LibrarySystem (Singleton)

### **📦 6. storage**

Simulated local data persistence (files, I/O functions).

**Artifacts / Files:**

* users.dat, books.dat, transactions.dat, fines.dat
* Optional: StorageHandler class or DataUtils module for file operations

## **II. Package Dependencies**

plaintext

CopyEdit

+-------------+ +--------------+ +--------------+

| user | <------+ system +-------> | book |

+-------------+ +--------------+ +--------------+

^ | ^ |

| | | |

| v | v

+-------------+ +--------------+ +-------------+

| transaction |<------+ +--------->| fine |

+-------------+ +-------->+-------------+

\ /

\ /

---------> storage <---------

### **Dependency Explanations:**

* system depends on all other packages: it orchestrates requests and rules.
* transaction depends on user and book to associate users with borrowed books.
* fine depends on transaction to calculate overdue amounts.
* storage is used by all packages for file-based data persistence.
* user, book, transaction, and fine are independent of each other **by design**, to keep modules loosely coupled.

## **✅ UML Notes for Drawing**

* Represent each package as a **tabbed folder symbol**
* Connect packages with **dashed arrows labeled «import» or «use»**
* Place system at the center; others radiating outward
* Show storage below, with imports from all other packages

## **✅ Summary**

| **Package** | **Responsibility** | **Depends On** |
| --- | --- | --- |
| user | Manage users and roles | system, storage |
| book | Catalog and book info | system, storage |
| transaction | Borrowing/returning logic | user, book, storage |
| fine | Fine tracking and payment | transaction, storage |
| system | Core coordinator and access manager | all other packages |
| storage | File I/O and data simulation | used by all other packages |

✅ This **Package Diagram is clean, modular, and aligns with your full system architecture**, ideal for:

* UML modeling
* Folder/file layout in C++
* Clear separation of responsibilities

## **ACTIVITY DIAGRAM — LIBRARY MANAGEMENT SYSTEM**

### **🎯 Main Flow: Borrowing and Returning Books**

This is the **core activity flow** most users follow, connecting:

* User Login
* Book Search
* Borrow Book
* Return Book
* Fine Handling
* Logout

### **I. Activity States**

1. Start
2. Enter Credentials
3. Validate Login
4. Show Menu
5. [Decision] User Type
6. Search Book
7. View Book Details
8. [Decision] Borrow Book?
9. Check Eligibility
10. Book Available?
11. Borrow Book → Create Transaction
12. Set Due Date
13. Show Borrow Success
14. [Decision] Return Book?
15. Select Book to Return
16. Match Transaction
17. Calculate Fine
18. [Decision] Is Fine Due?
19. Pay Fine
20. Mark Book Returned
21. Show Return Success
22. Logout
23. End

### **II. Activity Flow (with decisions/actions)**

plaintext

CopyEdit

(START)

|

[Enter Credentials]

|

[Validate Login]

|--> [Invalid?] --> [Show Error] --> (END)

|

[Show Menu]

|

[Decision: User Type]

|

[Search Book] → [View Book Details]

|

[Decision: Borrow Book?]

|--> No → [Decision: Return Book?]

|

[Check Eligibility (fines, max books)]

|

[Book Available?]

|--> No → [Show Error] → [Decision: Return Book?]

|

[Create Transaction]

|

[Set Due Date]

|

[Show Borrow Success]

|

[Decision: Return Book?]

|--> No → [Logout]

|

[Select Book to Return]

|

[Match Transaction]

|

[Calculate Fine]

|

[Decision: Is Fine Due?]

|--> Yes → [Pay Fine] → [Mark Book Returned]

|--> No → [Mark Book Returned]

|

[Show Return Success]

|

[Logout]

|

(END)

### **III. Swimlanes (Optional for UML Layout)**

| **Lane** | **Responsibilities** |
| --- | --- |
| **User** | Login, select options, borrow, return, pay fine |
| **LibrarySystem** | Validate, check eligibility, calculate fines |
| **BookModule** | Check availability, update status |
| **TransactionModule** | Create/close transaction |
| **FineModule** | Apply and resolve fines |

### **IV. UML Notation (for drawing)**

* **Rounded rectangles** → Activity nodes (actions/tasks)
* **Diamonds** → Decision nodes (if/else, yes/no)
* **Arrows** → Control flow
* **Black circle** → Initial state
* **Bullseye** → Final state
* **Vertical partitions (optional)** → Swimlanes for User/System separation

## **COMPLETE SEQUENCE DIAGRAM — LIBRARY MANAGEMENT SYSTEM**

### **🎯 Scenario**

A registered user logs in, searches for a book, borrows it (if eligible), returns it later, pays fine (if overdue), and logs out.

### **I. Participants (Objects/Actors)**

1. :RegisteredUser
2. :LibrarySystem *(Singleton)*
3. :UserModule
4. :BookModule
5. :TransactionModule
6. :FineModule

### **II. Interactions in Sequence**

plaintext

CopyEdit

=== [Login Phase] ===

1. :RegisteredUser → :LibrarySystem : login(email, password)

2. :LibrarySystem → :UserModule : validateUser(email, password)

3. :UserModule → :LibrarySystem : return success

4. :LibrarySystem → :RegisteredUser : loginSuccess()

=== [Search Book Phase] ===

5. :RegisteredUser → :LibrarySystem : searchBook(query)

6. :LibrarySystem → :BookModule : search(query)

7. :BookModule → :LibrarySystem : return list of Book\*

8. :LibrarySystem → :RegisteredUser : displaySearchResults()

=== [Borrow Book Phase] ===

9. :RegisteredUser → :LibrarySystem : borrowBook(bookID)

10. :LibrarySystem → :FineModule : checkUserFine(userID)

11. :FineModule → :LibrarySystem : return fineAmount

12. :LibrarySystem → :UserModule : getBorrowCount(userID)

13. :UserModule → :LibrarySystem : return borrowCount

14. :LibrarySystem → :BookModule : isAvailable(bookID)

15. :BookModule → :LibrarySystem : return true

16. :LibrarySystem → :TransactionModule : createTransaction(userID, bookID)

17. :TransactionModule → :LibrarySystem : return Transaction object

18. :LibrarySystem → :BookModule : updateStatus(bookID, "Borrowed")

19. :BookModule → :LibrarySystem : confirm update

20. :LibrarySystem → :RegisteredUser : confirmBorrow(transactionDetails)

=== [Return Book Phase] ===

21. :RegisteredUser → :LibrarySystem : returnBook(bookID)

22. :LibrarySystem → :TransactionModule : findTransaction(userID, bookID)

23. :TransactionModule → :LibrarySystem : return transactionDetails

24. :LibrarySystem → :TransactionModule : closeTransaction(returnDate)

25. :TransactionModule → :LibrarySystem : return fineAmount

26. :LibrarySystem → :BookModule : updateStatus(bookID, "Available")

27. :BookModule → :LibrarySystem : confirm update

28. :LibrarySystem → :RegisteredUser : returnProcessed()

=== [Pay Fine Phase (If Needed)] ===

29. :LibrarySystem → :FineModule : isFineDue(userID)

30. :FineModule → :LibrarySystem : return true / false

31. [If true] :RegisteredUser → :LibrarySystem : payFine(amount)

32. :LibrarySystem → :FineModule : processPayment(userID, amount)

33. :FineModule → :LibrarySystem : paymentSuccess()

34. :LibrarySystem → :RegisteredUser : fineCleared()

=== [Logout Phase] ===

35. :RegisteredUser → :LibrarySystem : logout()

36. :LibrarySystem → :UserModule : clearSession(userID)

37. :UserModule → :LibrarySystem : sessionTerminated

38. :LibrarySystem → :RegisteredUser : logoutSuccess()

### **III. UML Drawing Guidelines**

| **Element** | **UML Representation** |
| --- | --- |
| Participants | Vertically aligned lifelines |
| Method calls | Solid arrows (left to right, top-down) |
| Returns | Dashed arrows |
| Activation boxes | Narrow rectangles on lifelines |
| Optional logic | Use alt or opt fragments if drawing |
| Loops (e.g., retry) | Use loop fragment if needed |

### **IV. Diagram Coverage Summary**

| **Module/Phase** | **Covered** | **Participants Involved** |
| --- | --- | --- |
| User Authentication | ✅ | UserModule |
| Book Search | ✅ | BookModule |
| Borrow Book | ✅ | UserModule, BookModule, TransactionModule, FineModule |
| Return Book | ✅ | TransactionModule, BookModule |
| Fine Management | ✅ | FineModule |
| Logout | ✅ | UserModule |

### **✅ FINAL NOTES:**

* **System logic is fully covered**
* **All class-level and package-level interfaces are exercised**
* **Directly implementable in C++**
* **Easily drawn in UML tools like draw.io, Lucidchart, StarUML**

## **COMMUNICATION DIAGRAM — FULL USER WORKFLOW**

### **🎯 Use Case Flow**

A RegisteredUser logs in, searches for a book, borrows it, returns it later, pays fine if any, and logs out.

### **I. Objects / Participants**

1. user : RegisteredUser
2. system : LibrarySystem *(singleton controller)*
3. userModule : UserModule
4. bookModule : BookModule
5. transactionModule : TransactionModule
6. fineModule : FineModule

### **II. Communication Links & Numbered Messages**

plaintext

CopyEdit

[1] user → system : login(email, password)

[2] system → userModule : validateUser(email, password)

[3] userModule → system : return success

[4] system → user : loginSuccess()

[5] user → system : searchBook(query)

[6] system → bookModule : search(query)

[7] bookModule → system : return list of Book\*

[8] system → user : displaySearchResults()

[9] user → system : borrowBook(bookID)

[10] system → fineModule : checkUserFine(userID)

[11] fineModule → system : fineAmount

[12] system → userModule : getBorrowCount(userID)

[13] userModule → system : return borrowCount

[14] system → bookModule : isAvailable(bookID)

[15] bookModule → system : return true

[16] system → transactionModule : createTransaction(userID, bookID)

[17] transactionModule → system : return Transaction

[18] system → bookModule : updateStatus(bookID, "Borrowed")

[19] bookModule → system : confirm

[20] system → user : confirmBorrow(transaction)

[21] user → system : returnBook(bookID)

[22] system → transactionModule : findTransaction(userID, bookID)

[23] transactionModule → system : return transaction

[24] system → transactionModule : closeTransaction(returnDate)

[25] transactionModule → system : return fineAmount

[26] system → bookModule : updateStatus(bookID, "Available")

[27] bookModule → system : confirm

[28] system → user : returnProcessed()

[29] system → fineModule : isFineDue(userID)

[30] fineModule → system : return true/false

[31] user → system : payFine(amount)

[32] system → fineModule : processPayment(userID, amount)

[33] fineModule → system : paymentSuccess()

[34] system → user : fineCleared()

[35] user → system : logout()

[36] system → userModule : clearSession(userID)

[37] userModule → system : sessionTerminated

[38] system → user : logoutSuccess()

### **III. UML Layout Instructions (for Drawing)**

* **Each participant**: box labeled objectName : ClassName
* **Links**: solid lines between objects (show who communicates with whom)
* **Messages**:  
  + Numbered labels (e.g., [5])
  + Arrows with method calls and parameters (e.g., searchBook(query))

#### **Suggested Layout (Left to Right)**

plaintext

CopyEdit

user : RegisteredUser

|

system : LibrarySystem

|

userModule : UserModule

|

bookModule : BookModule

|

transactionModule : TransactionModule

|

fineModule : FineModule

### **✅ FULL INTERACTION MAP (Who Talks to Whom)**

| **From** | **To** | **Purpose** |
| --- | --- | --- |
| user | system | Initiates all actions |
| system | userModule | Authentication, borrow count, session |
| system | bookModule | Search, availability, update status |
| system | transactionModule | Borrow, return, close transaction |
| system | fineModule | Fine checks and processing |
| fineModule | system | Returns fine info/payment result |
| transactionModule | system | Returns transaction and fine info |

### **✅ Diagram Scope Coverage**

| **Phase** | **✅ Included** |
| --- | --- |
| Login | ✅ |
| Book Search | ✅ |
| Borrow Book | ✅ |
| Return Book | ✅ |
| Fine Handling | ✅ |
| Logout | ✅ |

### **✅ Diagram Usefulness**

* 🔎 Helps **trace complete workflow logic**
* 🧩 Highlights **module-level communication**
* 📐 Great for **UML tools** like StarUML, Lucidchart, draw.io
* 💻 Matches **C++ module calls** exactly