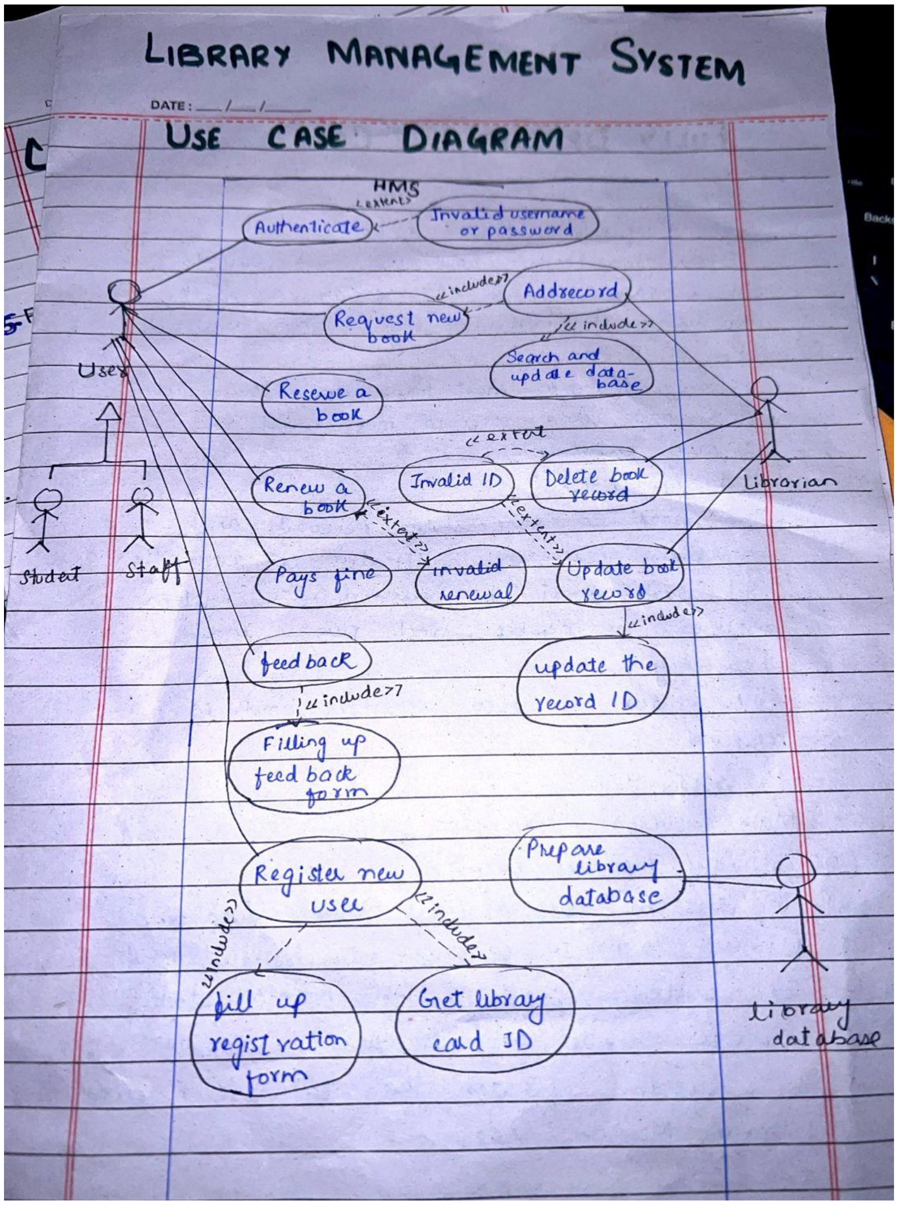
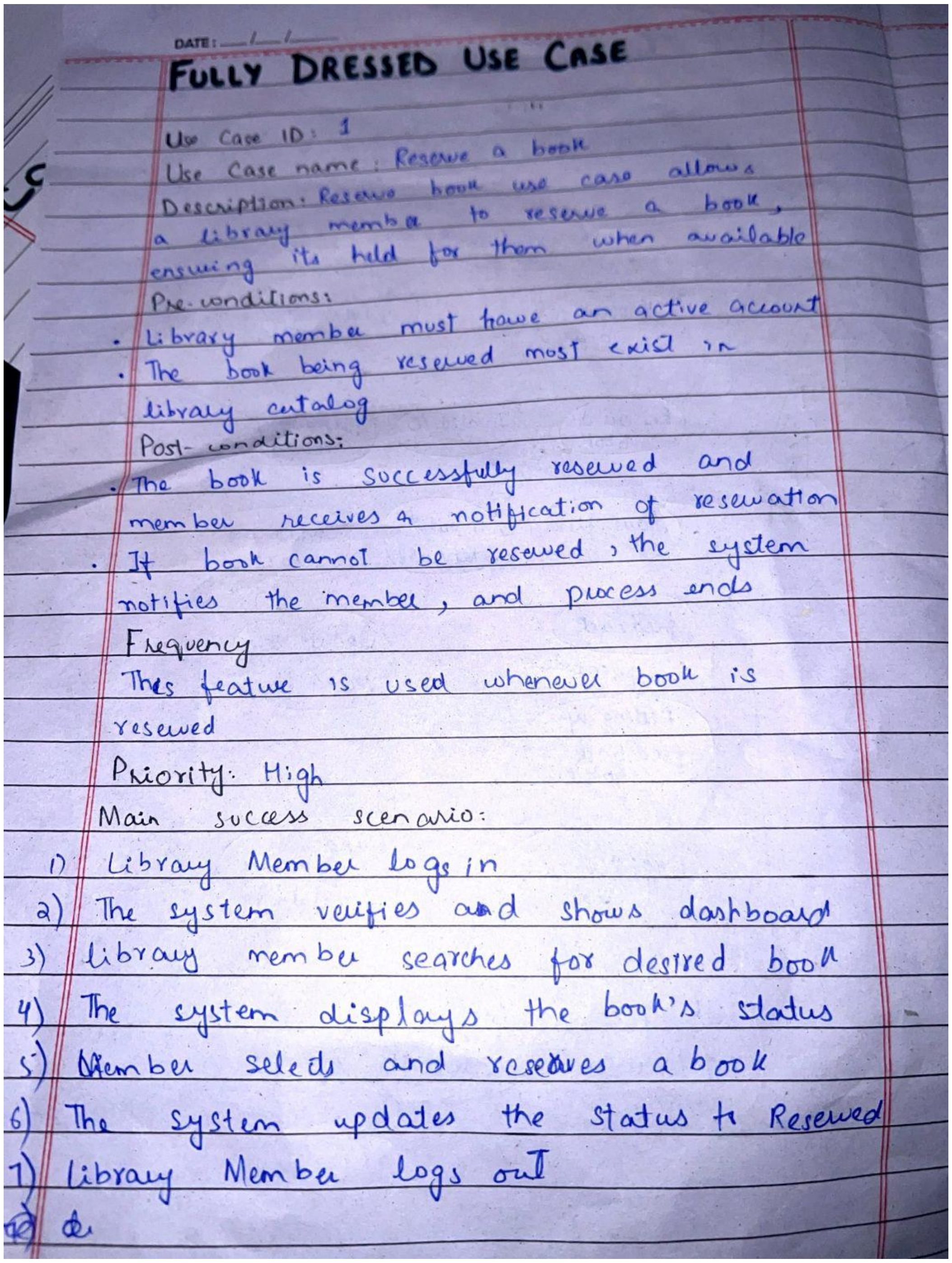
LIBRARY MANAGEMENT SYSTEM

Use case diagram



RESERVE A BOOK

Fully dressed use case



DATE: __/__/__

FULLY DRESSED USE CASE

Use Case ID: 1

Use Case name: Reserve a book

Description: Reserve book use case allows a library member to reserve a book, ensuring its held for them when available

Pre-conditions:

- Library member must have an active account
- The book being reserved must exist in library catalog

Post-conditions:

- The book is successfully reserved and member receives a notification of reservation
- If book cannot be reserved, the system notifies the member, and process ends

Frequency

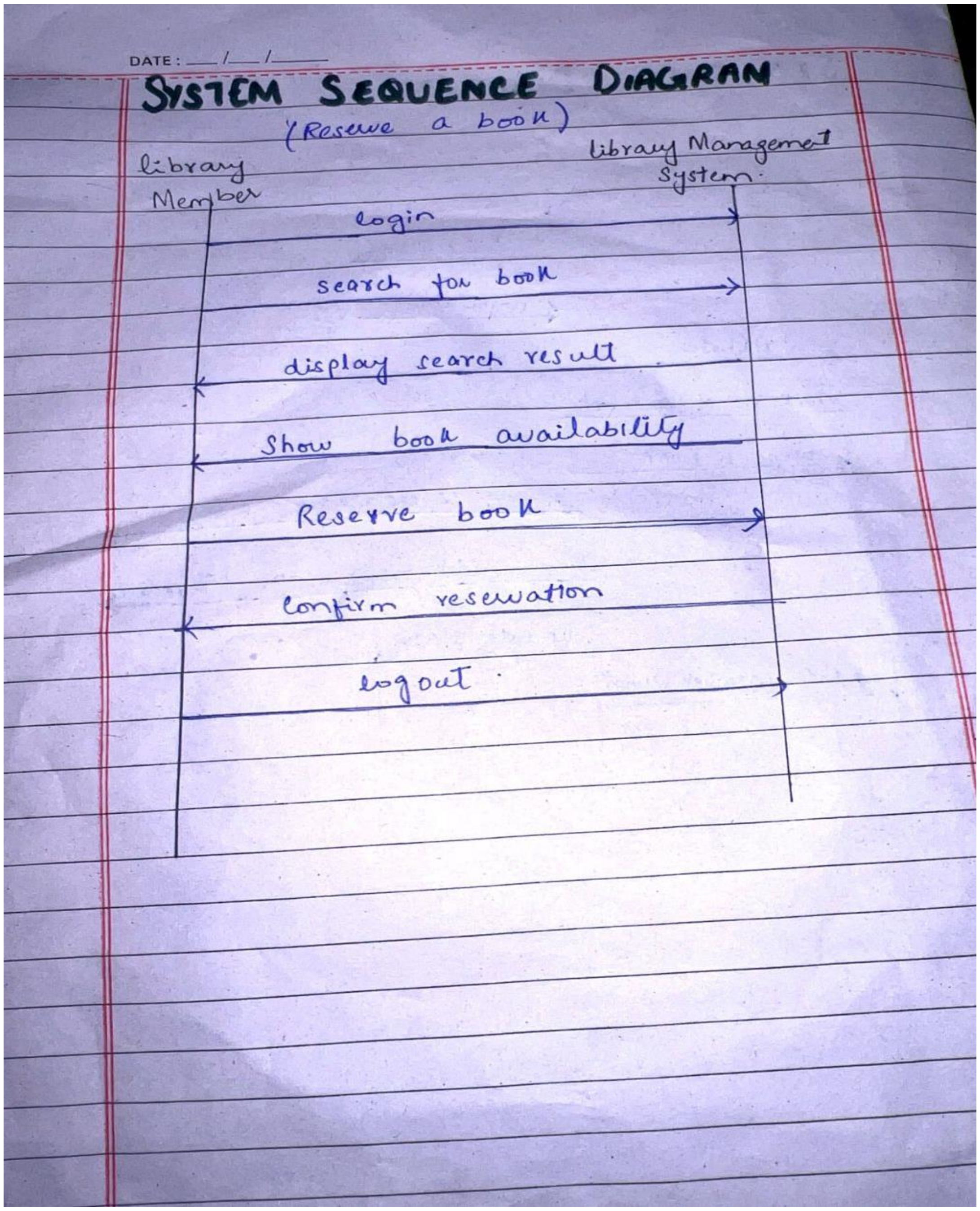
This feature is used whenever book is reserved

Priority: High

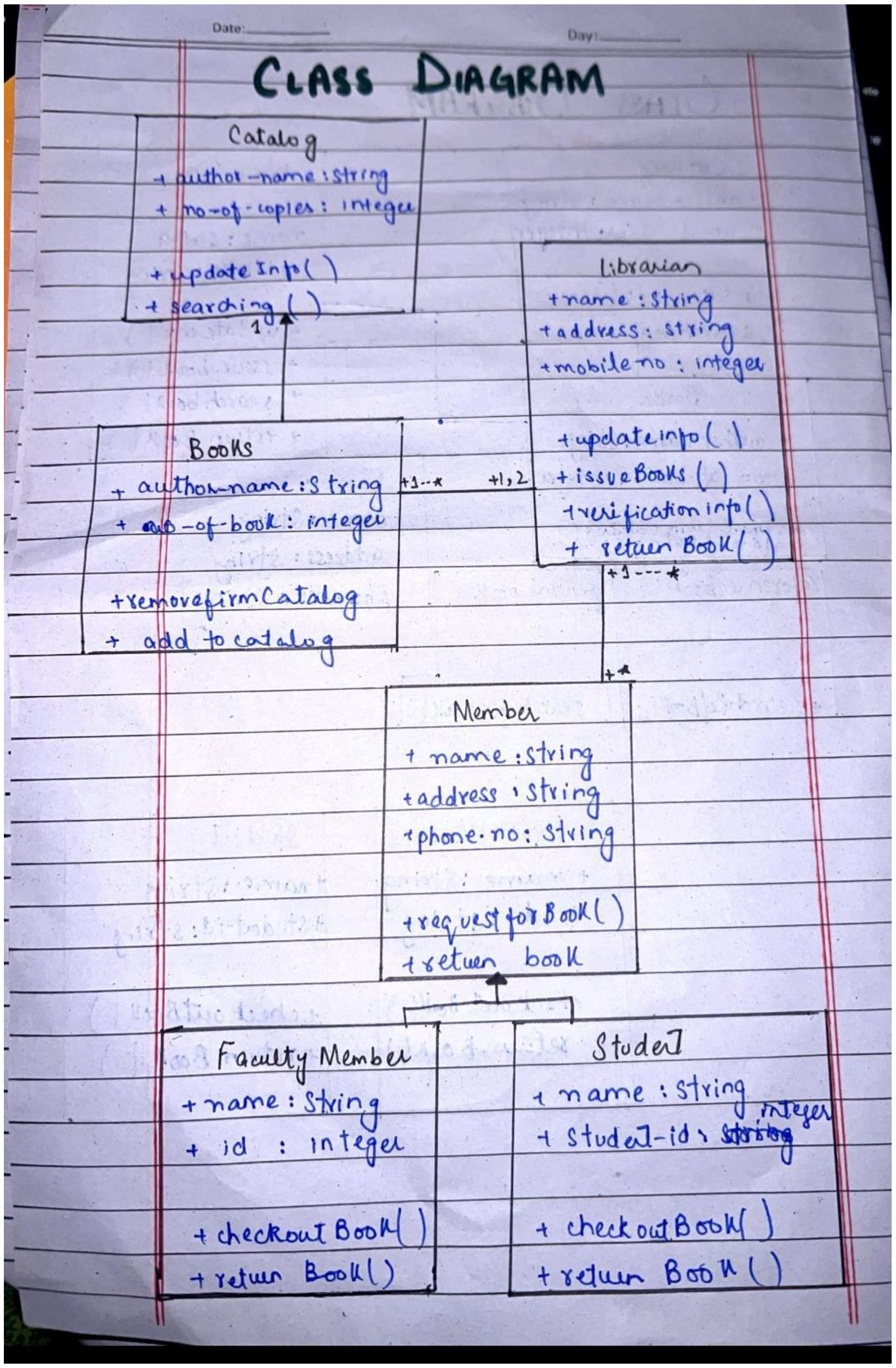
Main success scenario:

1) Library Member logs in
2) The system verifies and shows dashboard
3) library member searches for desired book
4) The system displays the book's status
5) Member selects and reserves a book
6) The system updates the status to Reserved
7) library Member logs out
~~8) a~~

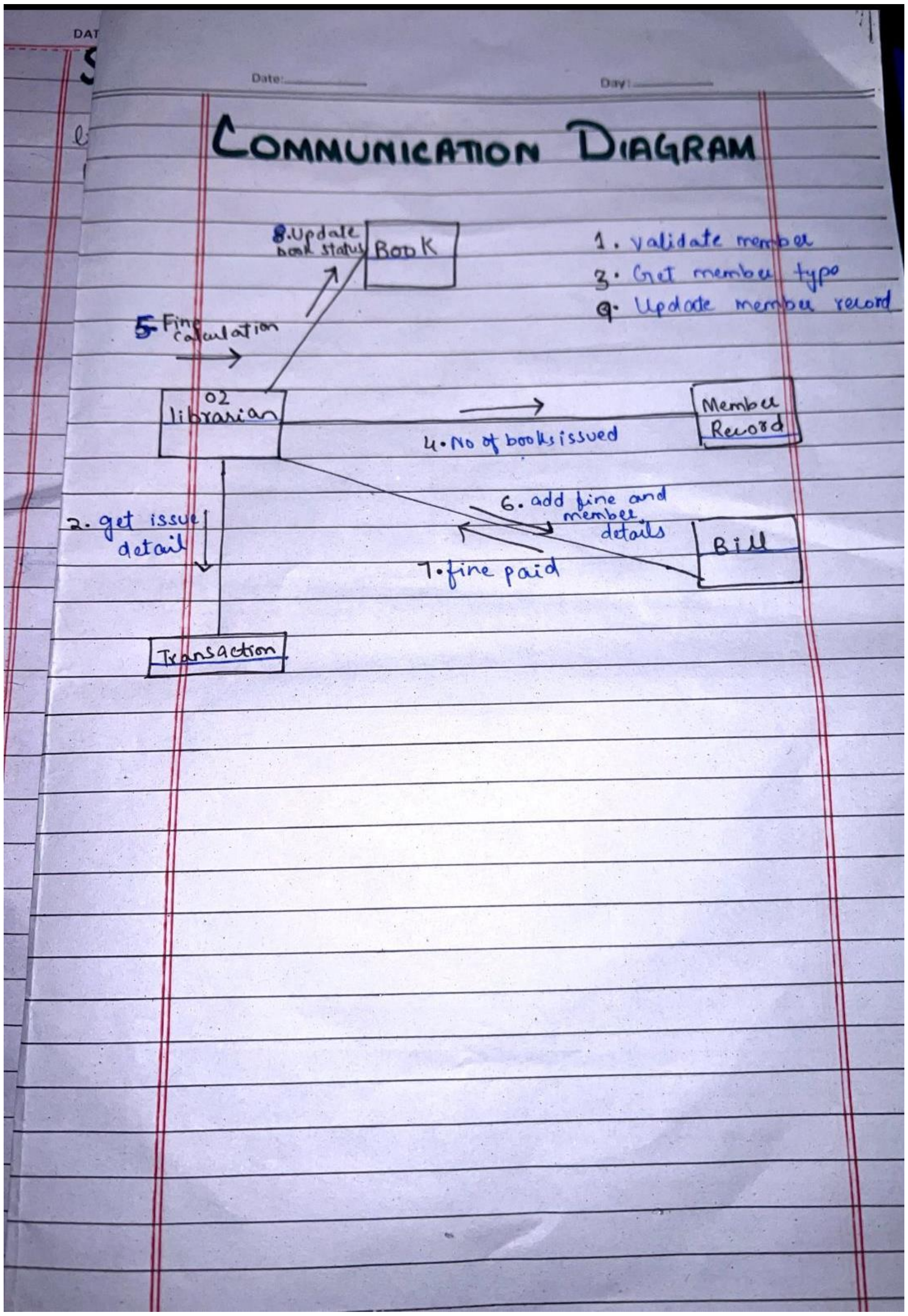
System Sequence diagram



Class diagram



Communication diagram



Principles used in Communication diagram

1. Polymorphism:

- **Definition**: Polymorphism allows objects to be treated as instances of their parent class, enabling flexibility and extensibility.
- **In the Diagram**: Although the diagram doesn't explicitly show polymorphism, it can be assumed in the interactions. For instance, different types of members (e.g., student, faculty) might be handled in the **Member Record** system through polymorphic behaviors, where different types of users are validated using the same interface.

2. Pure Fabrication:

- **Definition**: Pure fabrication refers to creating classes or objects that don't map directly to the real-world problem but are necessary for maintaining high cohesion and low coupling.
- **In the Diagram**: The **Bill** object could be an example of pure fabrication. It's not a real-world actor but a construct created to manage transactions such as fine payments, separate from the **Librarian** and **Transaction** components.

3. Controller:

- **Definition**: The controller is responsible for handling system events and directing messages between different objects.
- **In the Diagram**: The **Librarian** acts as the controller, initiating interactions such as book validation, issuing books, and updating records. The librarian controls the flow of data between different system components (Book, Member Record, etc.).

Code Implementation

Book.java

The `Book` class represents a book entity, managing its reservation status.

```
java
Copy code
public class Book {
    private String title;
    private String author;
    private boolean isReserved;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isReserved = false;
    }
    public boolean isReserved() {
        return isReserved;
    }

    public void reserve() {
        this.isReserved = true;
    }

    public void unreserve() {
        this.isReserved = false;
    }

    @Override
    public String toString() {
        return title + " by " + author;
    }
}
```

Member.java

The `Member` class represents a library member and is used to identify different types of users. **Polymorphism** can be applied here if different member types (e.g., student, faculty) require specific behaviors.

```
java
Copy code
public class Member {
    private String memberId;
    private String name;

    public Member(String memberId, String name) {
        this.memberId = memberId;
        this.name = name;
    }

    public String getMemberId() {
        return memberId;
    }

    public String getName() {
        return name;
    }
}
```

Reservation.java

The `Reservation` class manages the reservation data. This is an example of **Pure Fabrication** — an additional class created to maintain high cohesion and low coupling by handling reservations separately from `Librarian` and `Book`.

```
java
Copy code
```

```
public class Reservation {
    private Book book;
    private Member member;
    private String reservationDate;

    public Reservation(Book book, Member member, String reservationDate) {
        this.book = book;
        this.member = member;
        this.reservationDate = reservationDate;
    }

    public Book getBook() {
        return book;
    }

    public Member getMember() {
        return member;
    }

    @Override
    public String toString() {
        return "Reservation for: " + book.toString() + " by " +
member.getName();
    }
}
```

Librarian.java

The `Librarian` class acts as the **Controller**, managing the reservation process and serving as the central class that interacts with other components.

```
java
Copy code
import java.util.ArrayList;
import java.util.List;

public class Librarian {
    private List<Reservation> reservations;

    public Librarian() {
        this.reservations = new ArrayList<>();
    }

    // Reserve a book
    public String reserveBook(Book book, Member member, String
reservationDate) {
        if (book.isReserved()) {
            return "The book is already reserved.";
        }

        // Create a new reservation
        book.reserve();
        Reservation reservation = new Reservation(book, member,
reservationDate);
        reservations.add(reservation);
        return "Book reserved successfully for " + member.getName() + ".";
    }

    // Cancel a reservation
    public String cancelReservation(Book book, Member member) {
        for (Reservation reservation : reservations) {
            if (reservation.getBook().equals(book) &&
reservation.getMember().equals(member)) {
                reservations.remove(reservation);
                book.unreserve();
                return "Reservation canceled successfully for " +
member.getName() + ".";
            }
        }
        return "Reservation not found.";
    }

    public List<Reservation> getReservations() {
        return reservations;
    }
}
```

Test Library Management System

This class demonstrates the functionality of the `reserveBook` use case.

```
java
Copy code
public class LibraryManagementSystemTest {
    public static void main(String[] args) {
        // Create Librarian (controller)
        Librarian librarian = new Librarian();

        // Create Book and Member objects
        Book book1 = new Book("Effective Java", "Joshua Bloch");
        Member member1 = new Member("M01", "John Doe");

        // Reserve a book
        String result = librarian.reserveBook(book1, member1, "2024-11-05");
        System.out.println(result);  // Output: Book reserved successfully
for John Doe.

        // Attempt to reserve the same book again
        String result2 = librarian.reserveBook(book1, member1, "2024-11-05");
        System.out.println(result2); // Output: The book is already reserved.

        // Cancel the reservation
        String result3 = librarian.cancelReservation(book1, member1);
        System.out.println(result3); // Output: Reservation canceled
successfully for John Doe.
    }
}
```