Class Design (OOP Approach)

1. **Abstract Class: `User` (Abstraction)**
   - Common properties: `id`, `name`, `email`
   - Abstract method: `showMenu()`
2. **Derived Classes (Inheritance)**
   - `Student` (can borrow/return books)
   - `Librarian` (can add/remove books)
3. **Class: `Book` (Encapsulation)**
   - Fields: `bookId`, `title`, `author`, `available`
   - Private fields with getters/setters
4. **Class: `Library` (Encapsulation + Composition)**
   - Maintains a `List<Book>`
   - Methods: `addBook`, `removeBook`, `searchBook`, `borrowBook`, `returnBook`
5. **Main Application: `LibraryApp`**
   - Console-driven menu for login (student/librarian)
   - Uses **polymorphism**: once logged in, calls `user.showMenu()`

User.java

```
import java.util.*;

// Abstraction
abstract class User {
    protected int id;
    protected String name;
    protected String email;

    public User(int id, String name, String email) {
        this.id = id;
        this.name = name;
        this.email = email;
    }

    public abstract void showMenu(Library library, Scanner sc);
}
```

```
// Inheritance: Student
class Student extends User {
    public Student(int id, String name, String email) {
        super(id, name, email);
    }

    @Override
    public void showMenu(Library library, Scanner sc) {
        while (true) {
            System.out.println("\n--- Student Menu ---");
            System.out.println("1. View Books");
            System.out.println("2. Borrow Book");
            System.out.println("3. Return Book");
            System.out.println("4. Logout");
            System.out.print("Choose: ");
            int choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1 -> library.viewBooks();
                case 2 -> {
                    System.out.print("Enter Book ID to borrow: ");
                    int id = sc.nextInt();
                    library.borrowBook(id);
                }
                case 3 -> {
                    System.out.print("Enter Book ID to return: ");
                    int id = sc.nextInt();
                    library.returnBook(id);
                }
                case 4 -> { return; }
                default -> System.out.println("Invalid choice!");
            }
        }
    }
}
```

Librarian.java

```
// Inheritance: Librarian
class Librarian extends User {
    public Librarian(int id, String name, String email) {
        super(id, name, email);
    }

    @Override
    public void showMenu(Library library, Scanner sc) {
        while (true) {
            System.out.println("\n--- Librarian Menu ---");
            System.out.println("1. Add Book");
            System.out.println("2. Remove Book");
            System.out.println("3. View Books");
            System.out.println("4. Logout");
            System.out.print("Choose: ");
            int choice = sc.nextInt();
            sc.nextLine();

            switch (choice) {
                case 1 -> {
                    System.out.print("Enter Title: ");
                    String title = sc.nextLine();
                    System.out.print("Enter Author: ");
                    String author = sc.nextLine();
                    library.addBook(new Book(title, author));
                }
                case 2 -> {
                    System.out.print("Enter Book ID to remove: ");
                    int id = sc.nextInt();
                    library.removeBook(id);
                }
                case 3 -> library.viewBooks();
                case 4 -> { return; }
                default -> System.out.println("Invalid choice!");
            }
        }
    }
}
```

// Encapsulation: Book

```
class Book {
    private static int counter = 1;
    private int bookId;
    private String title;
    private String author;
    private boolean available;

    public Book(String title, String author) {
        this.bookId = counter++;
        this.title = title;
        this.author = author;
        this.available = true;
    }

// Getters and Setters
    public int getBookId() { return bookId; }
    public String getTitle() { return title; }
    public String getAuthor() { return author; }
    public boolean isAvailable() { return available; }
    public void setAvailable(boolean available) { this.available = available; }

    @Override
    public String toString() {
        return bookId + " | " + title + " | " + author + " | " + (available ? "Available" : "Not
Available");
    }
```

// Encapsulation + Composition: Library

```
class Library {
    private List<Book> books = new ArrayList<>();

    public void addBook(Book b) {
        books.add(b);
        System.out.println("Book added: " + b.getTitle());
    }

    public void removeBook(int bookId) {
        books.removeIf(b -> b.getBookId() == bookId);
        System.out.println("Book removed with ID: " + bookId);
    }

    public void viewBooks() {
        if (books.isEmpty()) {
            System.out.println("No books available!");
            return;
        }
        books.forEach(System.out::println);
    }

    public void borrowBook(int bookId) {
        for (Book b : books) {
            if (b.getBookId() == bookId && b.isAvailable()) {
                b.setAvailable(false);
                System.out.println("You borrowed: " + b.getTitle());
                return;
            }
        }
        System.out.println("Book not available!");
    }

    public void returnBook(int bookId) {
        for (Book b : books) {
            if (b.getBookId() == bookId && !b.isAvailable()) {
                b.setAvailable(true);
                System.out.println("You returned: " + b.getTitle());
                return;
            }
        }
        System.out.println("Invalid return request!");
    }
}
```

// Main App

```
public class LibraryApp {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Library library = new Library();

        while (true) {
            System.out.println("\n==== Library System ====");
            System.out.println("1. Student Login");
            System.out.println("2. Librarian Login");
            System.out.println("3. Exit");
            System.out.print("Choose: ");
            int choice = sc.nextInt();
            sc.nextLine();

            User user = null;
            if (choice == 1) {
                user = new Student(1, "John", "john@mail.com");
            } else if (choice == 2) {
                user = new Librarian(100, "Admin", "admin@mail.com");
            } else if (choice == 3) {
                System.out.println("Exiting...");
                break;
            } else {
                System.out.println("Invalid choice!");
                continue;
            }
            user.showMenu(library, sc);
        }
    }
}
```

Features Demonstrated

✓ **Encapsulation** → `Book` & `Library` keep data private, exposed with getters/setters.
✓ **Abstraction** → `User` is abstract, forces subclasses to implement `showMenu()`.
✓ **Inheritance** → `Student` and `Librarian` extend `User`.
✓ **Polymorphism** → At runtime, `user.showMenu()` behaves differently depending on object type.
✓ **Modern approach** → Clean OOP design, `List<Book>` (no arrays), menu-driven.