

MAHATMA EDUCATION SOCIETY'S  
PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE  
(Autonomous)  
NEW PANVEL

PROJECT REPORT ON

**“ LibraryManagement System”**

IN PARTIAL FULFILLMENT OF  
BACHELOR OF COMPUTER SCIENCE

SEMESTER V – 2024-25

PROJECT GUIDE

NAME: Mrs. SMITA HADAWALE MA'AM

SUBMITTED BY: PawanKumar Jaiswal

ROLL NO: 9127

# Library Management System

## Introduction

The **Library Management System** is a simple yet efficient Java-based application designed to manage a library's essential operations such as managing books and members. It utilizes **Hibernate** as the ORM (Object-Relational Mapping) framework to connect and interact with a MySQL database, making the operations like adding, updating, and deleting book or member information smooth and reliable. The application features a user-friendly command-line interface to handle core library functionalities, making it an ideal solution for small-scale libraries looking to digitize their record management.

## Key Features

1. **Book Management:**
  - Add, update, retrieve, and delete book information.
  - Maintain details such as book title, author, genre, and year of publication.
  - Ensures proper management and organization of book data.
2. **Member Management:**
  - Add, update, retrieve, and delete member information.
  - Stores critical details like member name, email, and address.
  - Enables easy tracking of registered members.
3. **Hibernate ORM Integration:**
  - **Efficient Database Interactions:** All operations like saving, updating, and deleting records are handled through Hibernate ORM, ensuring minimal boilerplate code and fast data access.
  - **SessionFactory and Transaction Management:** Hibernate manages database connections and transactions automatically, providing robust error handling and rollback features.
4. **Database Connection Pooling:**
  - Uses **C3P0 connection pooling** to efficiently manage database connections, improving performance, especially in multi-user environments.
5. **MySQL Database Support:**
  - Fully compatible with MySQL, a widely used relational database. The system can be easily reconfigured for other relational databases with minor changes.
6. **Entity Mapping:**
  - Clearly defined **entity classes** (LibraryBook and LibraryMember) map Java objects directly to corresponding database tables.
  - Provides a seamless bridge between the application logic and the database schema.
7. **CRUD Operations:**
  - Full **Create, Read, Update, Delete** (CRUD) operations are supported for both books and members, making it easy to manage records.
  - Supports user-friendly input validation for adding and updating records.
8. **Modular DAO Pattern:**

- The system employs a **Data Access Object (DAO)** pattern for managing database transactions in a clean and maintainable way.
- Separation of concerns makes the system scalable and easy to maintain.

#### 9. Error Handling and Rollback:

- Comprehensive error handling ensures smooth user experience and ensures database consistency by rolling back in case of failed transactions.

LibraryApp.java

```
package com.library.app;
```

```
import com.library.dao.LibraryBookDAO;  
import com.library.dao.LibraryMemberDAO;  
import com.library.model.LibraryBook;  
import com.library.model.LibraryMember;
```

```
import java.util.InputMismatchException;  
import java.util.List;  
import java.util.Scanner;
```

```
public class LibraryApp {  
    private static Scanner scanner = new Scanner(System.in);  
    private static LibraryBookDAO bookDAO = new LibraryBookDAO();  
    private static LibraryMemberDAO memberDAO = new LibraryMemberDAO();
```

```
    public static void main(String[] args) {  
        while (true) {  
            System.out.println("Library Management Menu:");  
            System.out.println("1. Add Book");  
            System.out.println("2. View Book");  
            System.out.println("3. Update Book");  
            System.out.println("4. Delete Book");  
            System.out.println("5. View All Books");  
            System.out.println("6. Add Member");  
            System.out.println("7. View Member");  
            System.out.println("8. Update Member");  
            System.out.println("9. Delete Member");  
            System.out.println("10. View All Members");  
            System.out.println("11. Exit");
```

```
            int choice = getIntInput("Choose an option: ");  
            switch (choice) {  
                case 1:  
                    addBook();  
                    break;  
                case 2:  
                    viewBook();  
                    break;  
                case 3:  
                    updateBook();
```

```
        break;
    case 4:
        deleteBook();
        break;
    case 5:
        viewAllBooks();
        break;
    case 6:
        addMember();
        break;
    case 7:
        viewMember();
        break;
    case 8:
        updateMember();
        break;
    case 9:
        deleteMember();
        break;
    case 10:
        viewAllMembers();
        break;
    case 11:
        System.out.println("Exiting...");
        scanner.close();
        return;
    default:
        System.out.println("Invalid choice. Try again.");
    }
}
}
```

```
private static void addBook() {
    System.out.println("Enter Title:");
    String title = scanner.nextLine();

    System.out.println("Enter Author:");
    String author = scanner.nextLine();

    System.out.println("Enter Genre:");
    String genre = scanner.nextLine();

    int year = getIntInput("Enter Year:");

    LibraryBook book = new LibraryBook();
    book.setTitle(title);
    book.setAuthor(author);
    book.setGenre(genre);
    book.setYear(year);

    bookDAO.addBook(book);
}
```

```
        System.out.println("Book added successfully!");
    }

    private static void viewBook() {
        int id = getIntInput("Enter Book ID to view: ");
        LibraryBook book = bookDAO.getBook(id);

        if (book != null) {
            System.out.println("Book ID: " + book.getId());
            System.out.println("Title: " + book.getTitle());
            System.out.println("Author: " + book.getAuthor());
            System.out.println("Genre: " + book.getGenre());
            System.out.println("Year: " + book.getYear());
        } else {
            System.out.println("Book not found.");
        }
    }

    private static void updateBook() {
        int id = getIntInput("Enter Book ID to update: ");
        LibraryBook book = bookDAO.getBook(id);

        if (book != null) {
            System.out.println("Enter new Title (leave empty to keep current):");
            String title = scanner.nextLine();
            if (!title.isEmpty()) {
                book.setTitle(title);
            }

            System.out.println("Enter new Author (leave empty to keep current):");
            String author = scanner.nextLine();
            if (!author.isEmpty()) {
                book.setAuthor(author);
            }

            System.out.println("Enter new Genre (leave empty to keep current):");
            String genre = scanner.nextLine();
            if (!genre.isEmpty()) {
                book.setGenre(genre);
            }

            int year = getIntInput("Enter new Year (leave empty to keep current):");
            if (year > 0) {
                book.setYear(year);
            } else {
                System.out.println("Year must be positive.");
                return;
            }

            bookDAO.updateBook(book);
            System.out.println("Book updated successfully!");
        }
    }
}
```

```
} else {
    System.out.println("Book not found.");
}
}

private static void deleteBook() {
    int id = getIntInput("Enter Book ID to delete: ");
    LibraryBook book = bookDAO.getBook(id);

    if (book != null) {
        bookDAO.deleteBook(id);
        System.out.println("Book deleted successfully!");
    } else {
        System.out.println("Book not found.");
    }
}

private static void viewAllBooks() {
    List<LibraryBook> books = bookDAO.getAllBooks();

    if (books.isEmpty()) {
        System.out.println("No books found.");
    } else {
        for (LibraryBook book : books) {
            System.out.println("Book ID: " + book.getId());
            System.out.println("Title: " + book.getTitle());
            System.out.println("Author: " + book.getAuthor());
            System.out.println("Genre: " + book.getGenre());
            System.out.println("Year: " + book.getYear());
            System.out.println("-----");
        }
    }
}

private static void addMember() {
    System.out.println("Enter Name:");
    String name = scanner.nextLine();

    System.out.println("Enter Email:");
    String email = scanner.nextLine();

    System.out.println("Enter Address:");
    String address = scanner.nextLine();

    LibraryMember member = new LibraryMember();
    member.setName(name);
    member.setEmail(email);
    member.setAddress(address);

    memberDAO.addMember(member);
    System.out.println("Member added successfully!");
}
```

```
}
```

```
private static void viewMember() {  
    int id = getIntInput("Enter Member ID to view: ");  
    LibraryMember member = memberDAO.getMember(id);  
  
    if (member != null) {  
        System.out.println("Member ID: " + member.getId());  
        System.out.println("Name: " + member.getName());  
        System.out.println("Email: " + member.getEmail());  
        System.out.println("Address: " + member.getAddress());  
    } else {  
        System.out.println("Member not found.");  
    }  
}
```

```
private static void updateMember() {  
    int id = getIntInput("Enter Member ID to update: ");  
    LibraryMember member = memberDAO.getMember(id);  
  
    if (member != null) {  
        System.out.println("Enter new Name (leave empty to keep current):");  
        String name = scanner.nextLine();  
        if (!name.isEmpty()) {  
            member.setName(name);  
        }  
  
        System.out.println("Enter new Email (leave empty to keep current):");  
        String email = scanner.nextLine();  
        if (!email.isEmpty()) {  
            member.setEmail(email);  
        }  
  
        System.out.println("Enter new Address (leave empty to keep current):");  
        String address = scanner.nextLine();  
        if (!address.isEmpty()) {  
            member.setAddress(address);  
        }  
  
        memberDAO.updateMember(member);  
        System.out.println("Member updated successfully!");  
    } else {  
        System.out.println("Member not found.");  
    }  
}
```

```
private static void deleteMember() {  
    int id = getIntInput("Enter Member ID to delete: ");  
    LibraryMember member = memberDAO.getMember(id);  
  
    if (member != null) {
```

```
        memberDAO.deleteMember(id);
        System.out.println("Member deleted successfully!");
    } else {
        System.out.println("Member not found.");
    }
}

private static void viewAllMembers() {
    List<LibraryMember> members = memberDAO.getAllMembers();

    if (members.isEmpty()) {
        System.out.println("No members found.");
    } else {
        for (LibraryMember member : members) {
            System.out.println("Member ID: " + member.getId());
            System.out.println("Name: " + member.getName());
            System.out.println("Email: " + member.getEmail());
            System.out.println("Address: " + member.getAddress());
            System.out.println("-----");
        }
    }
}

private static int getIntInput(String prompt) {
    int input = 0;
    boolean valid = false;

    while (!valid) {
        System.out.print(prompt);
        try {
            input = scanner.nextInt();
            valid = true;
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter an integer.");
            scanner.next(); // Clear invalid input
        }
    }
    scanner.nextLine(); // Clear the newline character
    return input;
}
}

LibraryBookDAO.java
package com.library.dao;

import com.library.model.LibraryBook;
import com.library.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.util.List;
```



```
public class LibraryBookDAO {
    public void addBook(LibraryBook book) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.save(book);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) transaction.rollback();
            e.printStackTrace();
        }
    }

    public LibraryBook getBook(int id) {
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            return session.get(LibraryBook.class, id);
        }
    }

    public void updateBook(LibraryBook book) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.update(book);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) transaction.rollback();
            e.printStackTrace();
        }
    }

    public void deleteBook(int id) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            LibraryBook book = session.get(LibraryBook.class, id);
            if (book != null) {
                session.delete(book);
            }
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) transaction.rollback();
            e.printStackTrace();
        }
    }

    public List<LibraryBook> getAllBooks() {
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            return session.createQuery("from LibraryBook", LibraryBook.class).list();
        }
    }
}
```

```
}
LibraryMemberDAO.java
package com.library.dao;

import com.library.model.LibraryMember;
import com.library.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.util.List;

public class LibraryMemberDAO {
    public void addMember(LibraryMember member) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.save(member);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) transaction.rollback();
            e.printStackTrace();
        }
    }

    public LibraryMember getMember(int id) {
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            return session.get(LibraryMember.class, id);
        }
    }

    public void updateMember(LibraryMember member) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.update(member);
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) transaction.rollback();
            e.printStackTrace();
        }
    }

    public void deleteMember(int id) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            LibraryMember member = session.get(LibraryMember.class, id);
            if (member != null) {
                session.delete(member);
            }
            transaction.commit();
        }
    }
}
```

```
    } catch (Exception e) {  
        if (transaction != null) transaction.rollback();  
        e.printStackTrace();  
    }  
}  
  
public List<LibraryMember> getAllMembers() {  
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {  
        return session.createQuery("from LibraryMember", LibraryMember.class).list();  
    }  
}  
}
```

LibraryBook.java

package com.library.model;

import javax.persistence.\*;

@Entity

@Table(name = "library\_books")

public class LibraryBook {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String title;

private String author;

private String genre;

private int year;

// Getters and setters

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getTitle() { return title; }

public void setTitle(String title) { this.title = title; }

public String getAuthor() { return author; }

public void setAuthor(String author) { this.author = author; }

public String getGenre() { return genre; }

public void setGenre(String genre) { this.genre = genre; }

public int getYear() { return year; }

public void setYear(int year) { this.year = year; }

}

LibraryMember.java

package com.library.model;

import javax.persistence.\*;

@Entity

```
@Table(name = "library_members")
public class LibraryMember {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;
    private String email;
    private String address;

    // Getters and setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getAddress() { return address; }
    public void setAddress(String address) { this.address = address; }
}
HibernateUtil.java
package com.library.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            return new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

**Hibernate.cfg.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
    <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/clg1</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root75</property>

    <!-- Specify dialect -->
    <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>

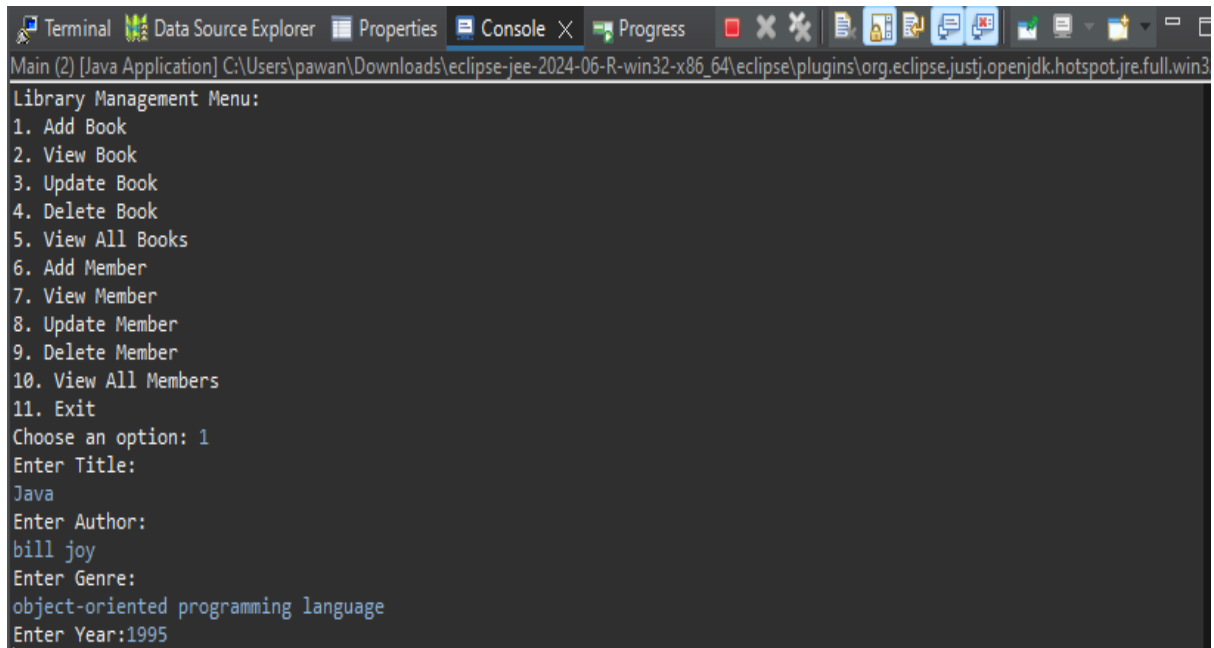
    <!-- Enable Hibernate's automatic session context management -->
    <property name="current_session_context_class">thread</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">true</property>

    <!-- Drop and re-create the database schema on startup -->
    <property name="hbm2ddl.auto">update</property>

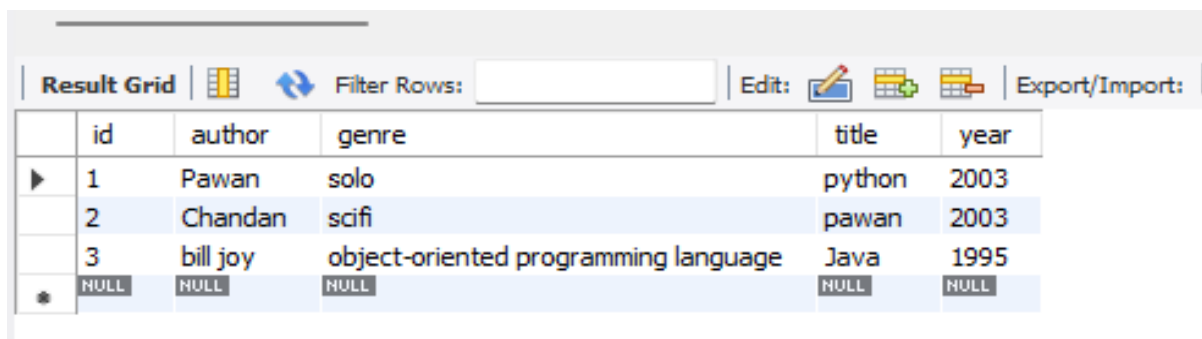
    <!-- Mention annotated classes -->
    <mapping class="com.library.model.LibraryBook"/>
    <mapping class="com.library.model.Member"/>

  </session-factory>
</hibernate-configuration>
```

**Output:****Create**

```
Terminal Data Source Explorer Properties Console X Progress
Main (2) [Java Application] C:\Users\pawan\Downloads\eclipse-jee-2024-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
Library Management Menu:
1. Add Book
2. View Book
3. Update Book
4. Delete Book
5. View All Books
6. Add Member
7. View Member
8. Update Member
9. Delete Member
10. View All Members
11. Exit
Choose an option: 1
Enter Title:
Java
Enter Author:
bill joy
Enter Genre:
object-oriented programming language
Enter Year:1995
```

```
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironment]
Hibernate: insert into library_books (author, genre, title, year) values (?, ?, ?, ?)
Book added successfully!
Library Management Menu:
1. Add Book
2. View Book
3. Update Book
4. Delete Book
5. View All Books
6. Add Member
7. View Member
8. Update Member
9. Delete Member
10. View All Members
11. Exit
```



	id	author	genre	title	year
▶	1	Pawan	solo	python	2003
	2	Chandan	scifi	pawan	2003
	3	bill joy	object-oriented programming language	Java	1995
✱	NULL	NULL	NULL	NULL	NULL

**Read**

```

Library Management Menu:
1. Add Book
2. View Book
3. Update Book
4. Delete Book
5. View All Books
6. Add Member
7. View Member
8. Update Member
9. Delete Member
10. View All Members
11. Exit
Choose an option: 2
Enter Book ID to view: 3
Hibernate: select libraryboo0_.id as id1_0_0_, libraryboo0_.author as author2_0_0_, libraryboo0_.genre as genre3_0_0_ from library_books libraryboo0_ where libraryboo0_.id=3
Book ID: 3
Title: Java
Author: bill joy
Genre: object-oriented programming language
Year: 1995

```

**Update**

```

Library Management Menu:
1. Add Book
2. View Book
3. Update Book
4. Delete Book
5. View All Books
6. Add Member
7. View Member
8. Update Member
9. Delete Member
10. View All Members
11. Exit
Choose an option: 3
Enter Book ID to update: 1
Hibernate: select libraryboo0_.id as id1_0_0_, libraryboo0_.author as author2_0_0_, libraryboo0_.genre as genre3_0_0_ from library_books libraryboo0_ where libraryboo0_.id=1
Enter new Title (leave empty to keep current):
python
Enter new Author (leave empty to keep current):
Guido van Rossum
Enter new Genre (leave empty to keep current):
object oriented
Enter new Year (leave empty to keep current):1947
Hibernate: update library_books set author=?, genre=?, title=?, year=? where id=?
Book updated successfully!

```

**Delete**

```

Library Management Menu:
1. Add Book
2. View Book
3. Update Book
4. Delete Book
5. View All Books
6. Add Member
7. View Member
8. Update Member
9. Delete Member
10. View All Members
11. Exit
Choose an option: 4
Enter Book ID to delete: 2
Hibernate: select libraryboo0_.id as id1_0_0_, libraryboo0_.author as author2_0_0_, libraryboo0_.genre as genre3_0_0_ from library_books libraryboo0_ where libraryboo0_.id=2
Hibernate: delete from library_books where id=?
Book deleted successfully!

```

## SWOT

### Strengths:

1. **Ease of Use:**
  - The system features a simple command-line interface, making it easy for users with basic technical skills to operate and manage library data.
2. **Efficient Database Management with Hibernate:**
  - Hibernate ORM abstracts the database layer, providing efficient database operations with minimal code and reducing errors.
3. **Modularity:**
  - The application follows a modular approach with a well-structured DAO pattern, making it easy to maintain and expand the system.
4. **Scalability:**
  - By using Hibernate and MySQL, the system is scalable to larger databases and can handle a growing number of books, members, and transactions.
5. **Cross-Platform Compatibility:**
  - Written in Java, the system is platform-independent and can run on any operating system that supports the JVM (Java Virtual Machine).

### Weaknesses:

1. **Basic User Interface:**
  - The system currently only provides a command-line interface, which may not be intuitive or visually appealing for all users.
2. **Limited Features:**
  - The system focuses primarily on basic book and member management. Advanced features like loan tracking, fine calculation, or catalog search are not included.
3. **Single-User Mode:**
  - The application is primarily designed for single-user interactions. Multi-user concurrency, session handling, and role-based access control are not fully implemented.
4. **No Real-Time Data Sync:**
  - The system does not support real-time updates or synchronization, making it less suitable for libraries requiring real-time transaction logging.

### Opportunities:

1. **Web Interface Integration:**
  - Developing a web-based UI would significantly enhance user experience, making the system more accessible and attractive to a wider audience.
2. **Multi-User Support:**
  - Adding support for multiple users, roles (e.g., admin, librarian), and access control can make the system suitable for larger libraries.



**3. Advanced Features:**

- Features such as loan tracking, catalog search, and fine management can add more value to the system and make it a complete library management solution.

**4. Integration with External Systems:**

- Integrating the system with other library systems or third-party services (e.g., book recommendation systems, library membership databases) can increase its functionality.

**Threats:**

**1. Competition with Established Systems:**

- Many mature and feature-rich library management systems already exist, such as Koha, which may overshadow this system's basic feature set.

**2. Technological Obsolescence:**

- With rapid technological advancements, systems based on simple CLI (Command-Line Interface) and traditional Java may become obsolete unless they are continually updated.

**3. Security and Data Privacy Concerns:**

- The current version lacks advanced security measures such as user authentication, encryption, or role-based access, which can pose risks for libraries dealing with sensitive data.

**4. Dependency on Hibernate:**

- If Hibernate or the underlying database is not properly configured or maintained, it could lead to performance issues, limiting the system's efficiency in high-demand scenarios.