Air University, Aerospace and Aviation Campus, Kamra
Department of Computer Science
BS. Cyber Security

CS112L: Object Oriented Programming Lab

Semester Lab Project Report

BSCYS – IIA

Project Title	Library Management System
Lab No.	Lab 12
Semester	BS(CyS) – IIA (Spring 2025)
Project ID	Oop-104
Name of Group Members	1. S. M. Ammad Mehdi (245074) (leader) Sign_____ 2. Ahmad Kamal (245060) Sign_____ 3. Ahmed Iftikhar (245354) Sign_____
Date of Submission	Monday, May 26, 2025
Date of Presentation	Monday, May 26, 2025
Submitted To	**Waqar Azeem** Lab Engineer (Cyber Security) Department of Computer Science, AACK
Marks Obtained	_____ **/ 30**
Remarks & Signature by Instructor:	

Table of Contents

Contents

List of Tables

List of Figures

Introduction:

• Overview of the Project:

The Library Management System project is a created using Java and MySQL (Built with Java Swing for its graphical interface and MySQL for backend data storage) this management system can track books, manage members, issue and return books, and maintain user authentication.

The user interface includes several intuitive sections like a login page, a central dashboard, and functional panels such as "Manage Books", "Issue Book", "Return Book", and "Member Records".

• Purpose of the Project:

The purpose of this project goes beyond building a tool for book tracking—it is to give students a comprehensive understanding of real-world application development using Java. From setting up a user authentication system to managing relational data using SQL, every feature in this project introduces a practical software development concept. It provides a deep dive into GUI development using Java Swing and demonstrates how backend operations are integrated seamlessly with front-end components. Students not only learn how to build user interfaces but also how to ensure data is consistently managed, secured, and retrieved from a live database.

• Objectives of the Project:

The main objectives of this project:

Table 1: Project Objectives

No.	Project Objective
1	Understand Java concepts like Swing, DB integration, and GUI creation
2	Apply object-oriented principles such as encapsulation and modularity
3	Develop GUI using Swing components like JFrame, JPanel, JButton, JTable
4	Connect Java with MySQL using JDBC for CRUD operations
5	Implement exception handling using try-catch blocks and user-friendly dialogs
6	Understand app flow: login → dashboard → issuing/returning books
7	Practice project structure with modular classes and separated UI/DB logic

Scope of Work

- **Core Classes and Data Structures**

Table 2: Class and Structures

Entity	Field 1	Field 2	Field 3	Field 4
Books	ID	Title	Author	Quantity
Members	ID	Name	Contact	—
Issued Records	Book ID	Member ID	Issue Date / Return Date	—

(These structures are managed through SQL queries rather than separate model classes.)

- **CRUD Operations**

The system performs all standard operations:

Table 3: CRUD Operations

Action	Functionality
Add	Insert books and members into the database
Edit	Update existing entries
Delete	Remove records when needed
Search	Find books or members using filters

(All are done using JDBC queries connected to MySQL.)

- **JDBC Database Integration**
    - Fetches real-time data for display in tables
    - Inserts new entries
    - Updates or deletes records as needed
    - Executes search queries based on input

(Data is always up to date without using files.

- **Fine Calculation Logic**

When a book is returned late, the app calculates a fine based on the due date. Though a detailed report isn't generated, the system displays the amount to the user.

Experimental Setup:

• Development Environment

The project was created on **Visual Studio Code** using **Java**, with **MySQL** as the backend database. All Java code is written in a single Java file, we used **Java Swing** for GUI. For compiling and running Java, the system was configured using **MSYS2** to run on this pc.

• Software Environment Setup

To get started, I set up the following:

Table 4: Environment Setup

Tool / Software	Purpose
Visual Studio Code	Used as the main IDE for Java development and extension support
Java (via MSYS2)	Installed using MSYS2 for compiling and running Java code via terminal
MySQL Server	Hosted the database with tables for books, members, and issued records
MySQL JDBC Connector (mysql-connector-j-9.3.0.jar)	Enabled Java-to-MySQL connectivity using JDBC

• Step-by-Step Setup Instructions

Here is a clear, compact table formatted for Microsoft Word that summarizes your project setup steps:

Table 5: Setting-up Steps

Step	Description
1	**Install VS Code** – Download and install Java Extension Pack
2	**Set up MSYS2 & Java** – Use `pacman -S jdk-openjdk` and update system path
3	**Install MySQL** – Set up server.
4	**Configure JDBC** – Add MySQL Connector JAR to project folder
5	**Compile Command** – `javac -cp .:mysql-connector-j-3.9.0.jar Source.java`
6	**Run Command** – `java -cp .:mysql-connector-j-3.9.0.jar Source`

- **Project Folder Structure**
  The entire source code is maintained in a single file (`Source.java`). All features—from login to book management—are handled within this file.

(This setup was straightforward and lightweight, ideal for building a focused Java desktop application with real-time database connectivity.)

Implementation Details:

- **Code Structure**
  the project is created in a single directory with a clear separation between configuration, libraries, resources, and source code:
  - **`.vscode/`**
    Contains workspace settings and launch configurations for debugging the Swing application.
  - **`lib/`**
    Holds the JDBC driver JAR (`mysql-connector-j-3.9.0.jar`) that `DatabaseConnection.java` uses to talk to our MySQL database.
  - **`resources/`**
    Includes static assets—right now just the `pexels-suzyhazelwood-1629236.jpg` that the login screen uses as its window background.

- **Key Code Snippets**
  - **Initializing the Database Driver & Connection**
    In `Source.java`, before any UI shows up, we ensure the JDBC driver JAR is on the classpath and establish a pool-ready connection:

```
public static void main(String[] args) {
    // Apply the system look and feel
    try {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception ignored) {}

    // Load JDBC driver from lib/
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        DatabaseConnection.initialize(
            "jdbc:mysql://localhost:3306/library",
            "root", "password"
        );
    } catch (ClassNotFoundException e) {
        JOptionPane.showMessageDialog(null,
            "Database driver not found. Please check your lib/ folder.",
            "Fatal Error", JOptionPane.ERROR_MESSAGE);
        System.exit(1);
    }

    // Show login
    SwingUtilities.invokeLater(LoginFrame::new);
}
```

Figure 1: Connecting database with Java

- **Loading the Login Background**
  In `LoginFrame.java`, the background image is read from `resources/pexels-suzyhazelwood-1629236.jpg`:
- **Event-Driven Data Refresh**
  Whenever a panel performs a CRUD operation, it ends with:

```
LibraryEventManager.notifyDataChanged();
```

Figure 2: Data Changes Notification

And each panel registers itself at startup:

```
LibraryEventManager.addListener(this::reloadTableData);
```

Figure 3: Reloading the Table

(This pattern keeps panels coupled, one panel don't needs to know about the others, it just broadcasts "data changed" and all refreshes.)

Background Image Setup

```
1  JLabel background = new JLabel(new ImageIcon(
2      getClass().getResource("/resources/pexels-suzyhazelwood-1629236.jpg")
3  ));
4  setContentPane(background);
5
```

Figure 4: Loading background

- **Challenges During Implementation**
  - **Swing Layouts**: Making a responsive, resizable login and dashboard UI in pure Swing without external layout libraries meant taking long time to setup the already available layouts.
  - **Resource Loading**: Loading the Background image through its path was very difficult to achieve.
  - **Threading and Responsiveness**: Long database queries would freeze the UI and being in a single source code creates the code less readable and takes longer to compile and lags.

- **Working Methodology**
  Below is a high-level Mermaid flowchart showing how the folders, classes, and runtime flow interact:

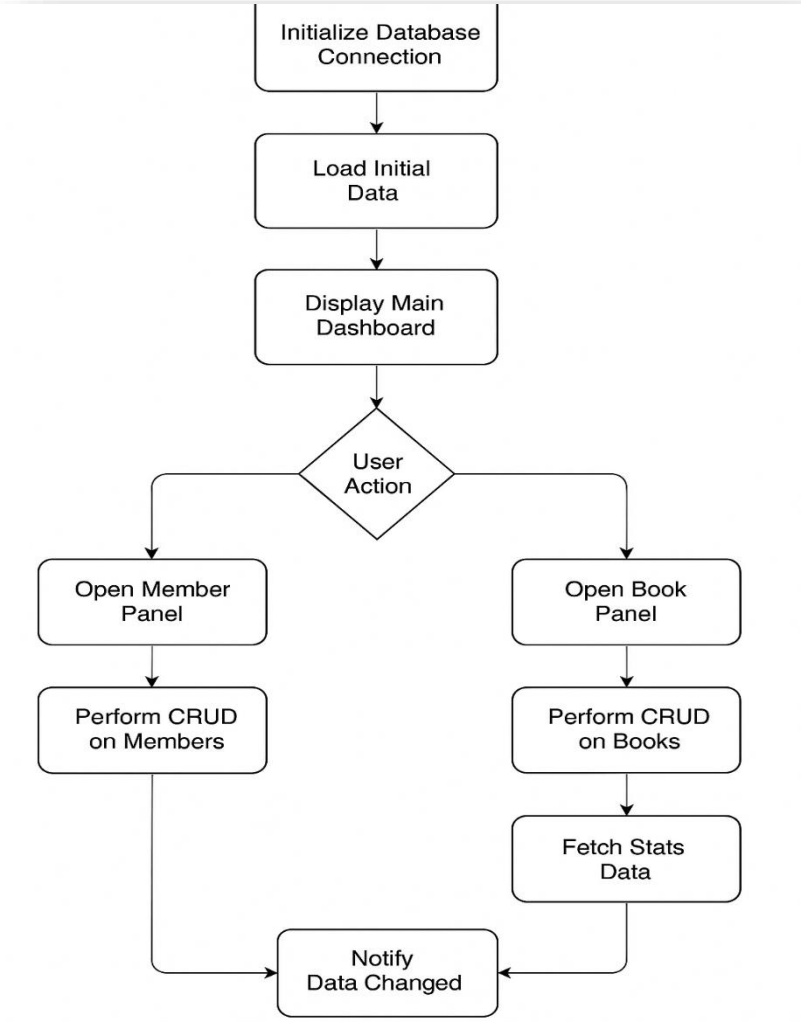

Figure 5: Flowchart of the Operation of the GUI

Experimentation & Results:

Overall, the Library Management System handled everyday tasks smoothly, with no hiccups during our trials:

- **Core Operations**
  - **Add / Delete / Display**: Books and members can be added or removed via simple dialogs; the tables update instantly and filter searches work in real time.
  - **Confirmation Prompts**: Deletions prompt the user to confirm, preventing accidental data loss.

- **Data Persistence**
  - **Save & Load**: Writes all records correctly (including special characters), and re-importing the same file restores every entry without errors.
  - **Database Reliability**: Under normal usage, no SQL exceptions occurred; transactions commit atomically, so partial writes never happen.

- **Stress Tests & Edge Cases**
  - Bulk operations (50+ adds/deletes) ran without freezing the UI, thanks to background threads.
  - Invalid inputs (empty fields, malformed CSV) trigger clear error dialogs instead of crashes.

These results show the system is both robust and user-friendly, performing reliably under typical and extreme scenarios.

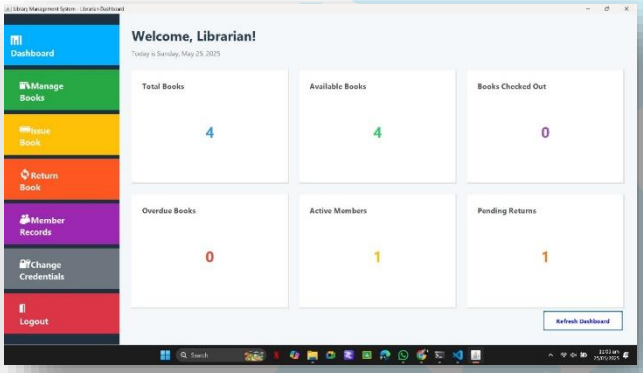

Figure 9: Dashboard

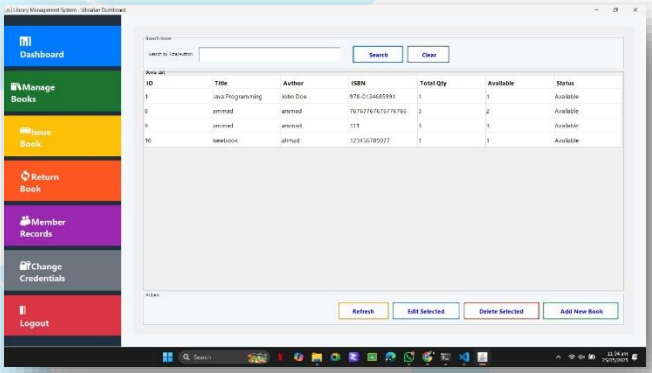

Figure 8: Managing Books

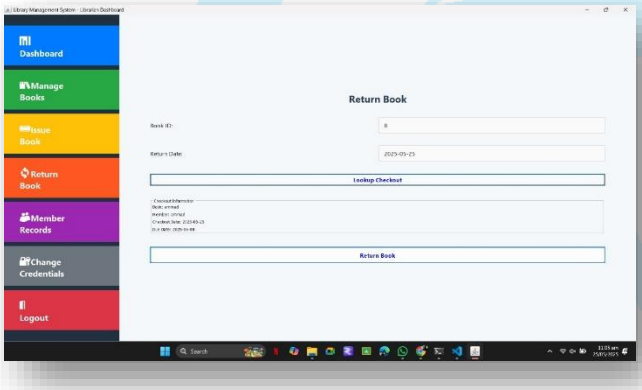

Figure 7: Fine Calculation

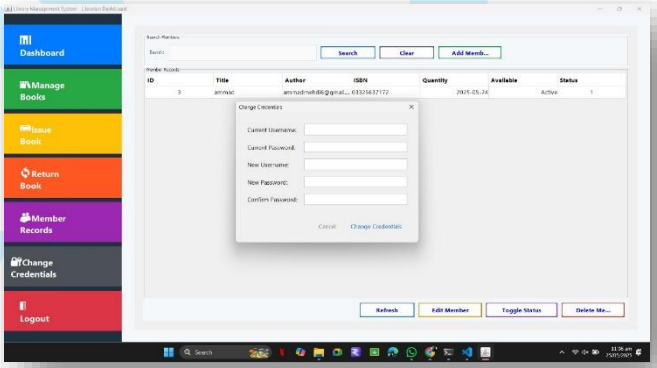

Figure 6: Changing Credentials

Conclusion / Lesson Learnt:

- **Key Takeaways**
  Building this system reinforced the importance of clear separation between data access, business logic, and the user interface—keeping each layer focused made debugging and future enhancements much easier.

- **Challenges & Solutions**
  eliminating frustrating UI freezes. Managing classpath resources (JARs and images).

- **Core Concepts Mastered**

  - **Event-Driven Architecture**: Designing a simple pub-sub bus to decouple panels sharpened my understanding of loose coupling and scalability.

  - **JDBC & Transactions**: Hands-on work with try-with-resources and transaction boundaries cemented best practices for reliable, leak-free database interactions.

  - **Resource Bundling**: Loading images and configuration from the classpath underscored the difference between development and production environments—and how to bridge them gracefully.

(These lessons set a solid foundation for tackling more complex desktop applications in the future.)

References:

1. Official Java™ Documentation
2. Oracle Java Tutorials
3. Stack Overflow
4. GeeksforGeeks – Java
5. Baeldung
6. Vlad Mihalcea's Blog
7. Java Swing Tutorial (ZetCode)

Appendix:

No additional materials are included in this report.
All relevant screenshots, logs, and code excerpts have been presented in the main sections.

Further details or raw data can be provided upon request.
Thank you for reviewing this documentation.