CASE STUDY

C.V. RAMAN GLOBAL UNIVERSITY, BHUBANESWAR, ODISHA, INDIA.

CASE STUDY ON:
LIBRARY MANAGEMENT
GROUP - 7

UNDER THE SUPERVISION OF
ASST. PROFESSOR

MR. SANDEEP PADESUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY:-

NAME	REGD.NO
SUBHAM TARASIA	CL20250106019293124
OMMKAR SAI MISHRA	CL2025010601891092
SOMNATH MAHAPATRA	CL2025010601926495
SATYAJIT ROUTRAY	CL2025010601923263

DECLARATION

We hereby declare that the project titled "**Library Management**" submitted by us in partial fulfilment of the requirements for the 6th semester of Bachelor of Technology, was carried out under the supervision and guidance of **Mr. Sandeep Padesur Sir**.

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude and appreciation to my trainer, **Mr. Sandeep Padesur**, for providing me with the invaluable opportunity to work on this case study on **Library Management** using **Java**. His constant guidance, support, and encouragement have been instrumental in the completion of this project, helping me refine my understanding of machine learning concepts and techniques related to medical prediction models.

This project has significantly enhanced my knowledge of algorithms and methodologies applied in predictive healthcare, especially in the field of breast cancer diagnosis. I would also like to extend my sincere thanks to my teammates, whose cooperation and support were crucial in finalizing this project within the given timeframe.

CONTENTS

1. Introduction -:

Introduction to the Library Management System

The Library Management System is a console-based Java application designed to streamline core operations of a library, such as managing books, members, and book transactions (borrowing/returning). Built using object-oriented programming (OOP) principles, this system demonstrates fundamental Java concepts while providing a functional and user-friendly interface for librarians or users to interact with.

Key Features

1. Book Management:
- Add new books to the library catalog.
- Track availability status (borrowed/available).
- List all books with details like title, author, and borrowing status.

2. Member Management:
- Register new members.
- Track books borrowed by each member.
- List all registered members.

3. Transaction Handling:
- Borrow books (validates availability and member eligibility).
- Return books (validates if the member borrowed the book).

4. User Interface :
- Simple, menu-driven console interface.
- Clear screen transitions for better readability.
- Input validation to handle basic errors (e.g., invalid menu choices).

Technical Overview

- Language : Java (core OOP concepts: classes, inheritance not used here, encapsulation, lists).
- Data Structures : ArrayList for storing books and members.
- Classes :
  - libraryManagementSystem : Manages core logic (add/remove books/members, transactions).
  - Book : Represents a book with properties like title, author, and borrowing status.
  - Member : Represents a member with a name and list of borrowed books.
- User Interaction : Utilizes Scanner for input and console-based menus.

Purpose

This project serves as a foundational example of:
- OOP Design : Clean separation of concerns between books, members, and system logic.

- Data Management : Using lists to simulate a database-free system.
- Real-World Application : Mimicking basic library workflows in a simplified manner.

Learning Outcomes
- Implementing CRUD (Create, Read, Update, Delete) operations.
- Validating user input and handling edge cases.
- Designing interactive console applications.

Future Scope
While functional, the system can be enhanced with:
- Persistence : Saving data to files/databases.
- Advanced Features : Due dates, fines, search functionality, or a GUI.
- Security : User authentication for librarians/members.

This project is ideal for understanding core Java concepts and serves as a stepping stone for building more complex systems.

2. Problem Statement -:

Libraries play a vital role in education and community development by providing access to books and resources. However, many libraries, especially small to medium-sized ones, still rely on manual or outdated systems to manage their operations. Traditional methods, such as paper-based records or fragmented digital tools, lead to inefficiencies, errors, and delays in critical tasks like:
- Tracking book availability.
- Managing member registrations.
- Handling borrow/return transactions.
- Generating reports (e.g., overdue books, popular titles).

Challenges with Existing Systems
1. **Time-Consuming Processes**:
   - Manual entry of book/member details slows down operations.
   - Difficulty locating books or verifying member eligibility for borrowing.

2**. Human Errors**:
   - Mismanagement of records (e.g., lost books, incorrect member data).
   - Overdue books go unnoticed due to lack of automated reminders.

3. **Lack of Real-Time Updates**:
   - No centralized system to track book availability or member activity.

4. **Scalability Issues**:
   - Manual systems struggle to handle growing collections or member bases.

5. **Limited Accessibility**:
   - Librarians cannot easily access data remotely or share insights with stakeholders.

Proposed Solution

A Java-based Library Management System aims to address these challenges by automating core library operations. The system will:

1. **Streamline Management**:
   - Digitally catalog books (title, author, ISBN, availability status).
   - Register and manage members (name, ID, contact details).

2. **Automate Transactions**:
   - Facilitate borrowing/returning with real-time updates.
   - Validate eligibility (e.g., outstanding fines, borrowing limits).

3. **Improve Accuracy**:
   - Reduce human errors in record-keeping.
   - Track overdue books and calculate fines automatically.

4. **Enhance Reporting**:
   - Generate reports (e.g., popular books, overdue lists).
   - Provide dashboards for librarians to monitor library activity.

5**. Ensure Scalability**:
   - Built using Java and Object-Oriented Programming (OOP) principles for modularity and easy future enhancements.

Objectives

1. Develop a user-friendly console/GUI application (Java-based) for librarians to:
   - Add/remove books and members.
   - Process borrow/return requests.
   - Search for books or members.

2. Implement data persistence (e.g., using files or databases) to retain records between sessions.

3. Incorporate input validation to prevent invalid data entry.

4. Ensure security (e.g., authentication for admin access).

5. Design the system to be scalable , allowing future integration of features like:
   - Barcode scanning for books.
   - Online reservations.
   - Email/SMS notifications for overdue books.

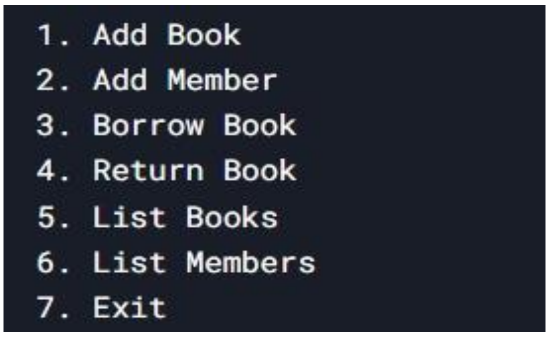
Why Java?
- Platform Independence : Runs on any OS with JVM.
- OOP Support : Enables clean, modular code for entities like Book, Member, and Transaction.
- Rich Libraries : Simplifies file handling, GUI development (Swing/JavaFX), and database integration (JDBC).

This problem statement highlights the necessity of automating library workflows and positions the Java-based system as a robust, scalable solution to modernize library operations.

3. Code/Algorithm Used -:

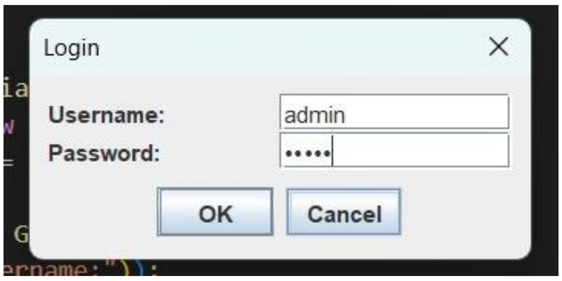
```
import javax.swing. ;
import java.awt. ;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.Arrays;

public class om {
    private LibSys library;
    private JFrame frame;
    private JTextArea displayArea;

    public om() {
        library = new LibSys();
        initialize();
    }

    private void initialize() {
        frame = new JFrame("Library Management System");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(800, 600);

        // Create buttons panel
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(7, 1, 5, 5));

        String[] buttonLabels = {
            "Add Book", "Add Member", "Borrow Book",
            "Return Book", "List Books", "List Members", "Exit"
        };

        for (String label : buttonLabels) {
            JButton button = new JButton(label);
            button.addActionListener(new ButtonClickListener());
            buttonPanel.add(button);
        }

        // Create display area
        displayArea = new JTextArea();
        displayArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(displayArea);

        // Add components to frame
        frame.getContentPane().add(buttonPanel, BorderLayout.WEST);
        frame.getContentPane().add(scrollPane, BorderLayout.CENTER);

        frame.setVisible(true);
    }

    private static boolean showLoginDialog() {
        JTextField usernameField = new JTextField();
        JPasswordField passwordField = new JPasswordField();

        JPanel panel = new JPanel(new GridLayout(2, 2));
        panel.add(new JLabel("Username:"));
        panel.add(usernameField);
        panel.add(new JLabel("Password:"));
        panel.add(passwordField);

        int option = JOptionPane.showConfirmDialog(
            null,
            panel,
            "Login",
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.PLAIN_MESSAGE
        );

        if (option == JOptionPane.OK_OPTION) {
            String username = usernameField.getText();
            String password = new String(passwordField.getPassword());
            return "admin".equals(username) && "admin".equals(password);
        }
        return false;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            if (showLoginDialog()) {
                new om();
            } else {
                JOptionPane.showMessageDialog(null, "Invalid credentials. Exiting.");
                System.exit(0);
            }
        });
    }

    // Inner classes for library functionality
    private class ButtonClickListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();

            switch (command) {
                case "Add Book": addBook(); break;
                case "Add Member": addMember(); break;
                case "Borrow Book": borrowBook(); break;
                case "Return Book": returnBook(); break;
                case "List Books": listBooks(); break;
                case "List Members": listMembers(); break;
                case "Exit": System.exit(0); break;
            }
        }
    }

    private void addBook() {
        String title = JOptionPane.showInputDialog("Enter book title:");
        String author = JOptionPane.showInputDialog("Enter book author:");
        if (title != null && author != null && !title.isEmpty() && !author.isEmpty()) {
            library.addBook(new Book(title, author));
            displayArea.append("Book added: " + title + " by " + author + "\n");
        }
    }

    private void addMember() {
        String name = JOptionPane.showInputDialog("Enter member name:");
        if (name != null && !name.isEmpty()) {
            library.registerMember(new Member(name));
            displayArea.append("Member added: " + name + "\n");
        }
    }

    private void borrowBook() {
        // Get list of available books and members
        List<Book> availableBooks = library.listBooks().stream()
                .filter(book -> !book.isBorrowed())
                .collect(Collectors.toList());
        List<Member> members = library.listMembers();

        if (availableBooks.isEmpty()) {
            displayArea.append("No books available for borrowing.\n");
            return;
        }
        if (members.isEmpty()) {
            displayArea.append("No registered members.\n");
            return;
        }

        // Create book selection dialog
        String[] bookOptions = availableBooks.stream()
                .map(book -> book.getTitle() + " by " + book.getAuthor())
                .toArray(String[]::new);
        String selectedBook = (String) JOptionPane.showInputDialog(
                frame,
                "Select a book:",
                "Borrow Book",
                JOptionPane.QUESTION_MESSAGE,
                null,
                bookOptions,
                bookOptions[0]);

        if (selectedBook == null) return;

        // Create member selection dialog
        String[] memberOptions = members.stream()
                .map(Member::getName)
                .toArray(String[]::new);
        String selectedMember = (String) JOptionPane.showInputDialog(
                frame,
                "Select a member:",
                "Borrow Book",
                JOptionPane.QUESTION_MESSAGE,
                null,
                memberOptions,
                memberOptions[0]);

        if (selectedMember == null) return;

        // Find selected book and member objects
        Book book =
availableBooks.get(Arrays.asList(bookOptions).indexOf(selectedBook));
        Member member =
members.get(Arrays.asList(memberOptions).indexOf(selectedMember));


// Borrow the book
        if (library.borrowBook(book, member)) {
            displayArea.append("Book borrowed: " + book.getTitle() + " by " +
book.getAuthor() + " to " + member.getName() + "\n");
        } else {
            displayArea.append("Failed to borrow book: " + book.getTitle() + " by " +
book.getAuthor() + "\n");
        }
    }

    private void returnBook() {
        // Implementation for returning a book
        displayArea.append("Return book functionality to be implemented\n");
    }

    private void listBooks() {
        displayArea.append("\nBook List:\n");
        for (Book book : library.listBooks()) {
            displayArea.append(book.getTitle() + " by " + book.getAuthor() +
                    (book.isBorrowed() ? " (Borrowed)" : " (Available)") + "\n");
        }
    }

    private void listMembers() {
        displayArea.append("\nMember List:\n");
        for (Member member : library.listMembers()) {
            displayArea.append(member.getName() + "\n");
        }
    }

    // Library system classes
    static class LibSys {
        private List<Book> books = new ArrayList<>();
        private List<Member> members = new ArrayList<>();

        public void addBook(Book book) {
            books.add(book);
        }

        public void registerMember(Member member) {
            members.add(member);
        }

        public boolean borrowBook(Book book, Member member) {
            if (!book.isBorrowed()) {
                book.setBorrowed(true);
                member.borrowBook(book);
                return true;
            }
            return false;
        }

        public boolean returnBook(Book book, Member member) {
            if (member.getBorrowedBooks().contains(book)) {
                book.setBorrowed(false);
                member.returnBook(book);
                return true;
            }
            return false;
        }

        public List<Book> listBooks() {
            return books;
        }

        public List<Member> listMembers() {
            return members;
        }
    }

    static class Book {
        private String title;
        private String author;
        private boolean borrowed;

        public Book(String title, String author) {
            this.title = title;
            this.author = author;
            this.borrowed = false;
        }

        public String getTitle() { return title; }
        public String getAuthor() { return author; }
        public boolean isBorrowed() { return borrowed; }
        public void setBorrowed(boolean status) { borrowed = status; }
    }

    static class Member {
        private String name;
        private List<Book> borrowedBooks = new ArrayList<>();

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

        public String getName() { return name; }
        public List<Book> getBorrowedBooks() { return borrowedBooks; }

        public void borrowBook(Book book) {
            borrowedBooks.add(book);
        }

        public void returnBook(Book book) {
            borrowedBooks.remove(book);
        }
    }
}
```

4. Output:



```
1. Add Book
2. Add Member
3. Borrow Book
4. Return Book
5. List Books
6. List Members
7. Exit
```

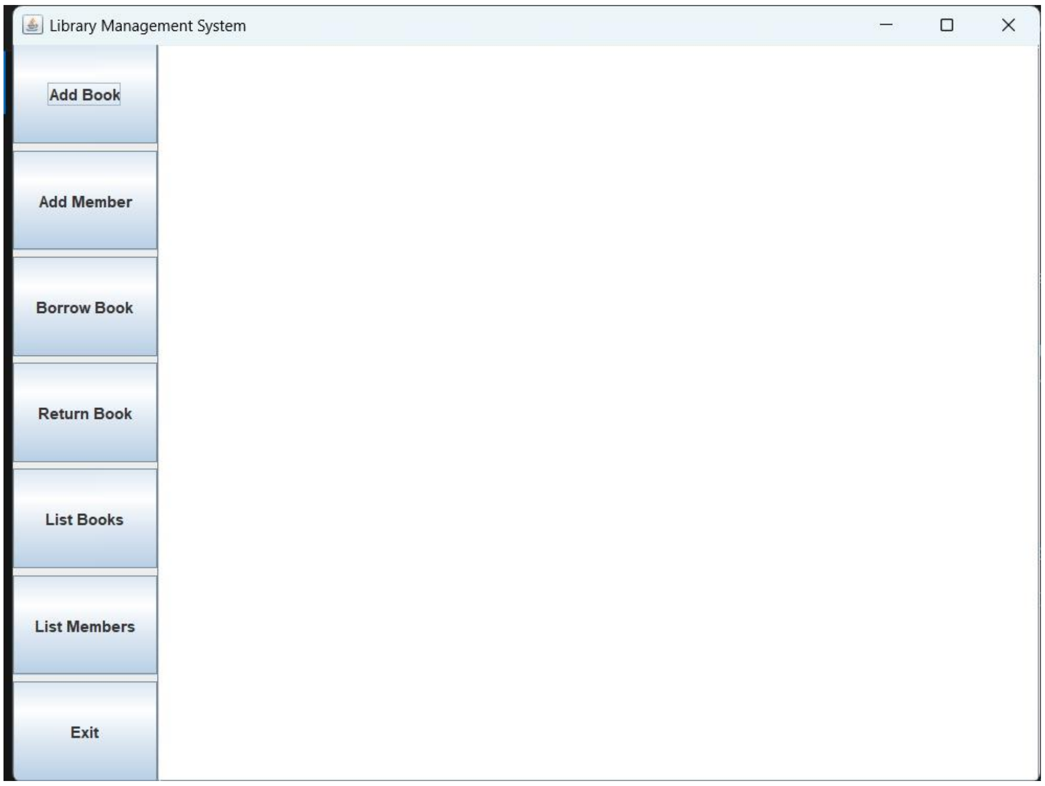
a) Login page :



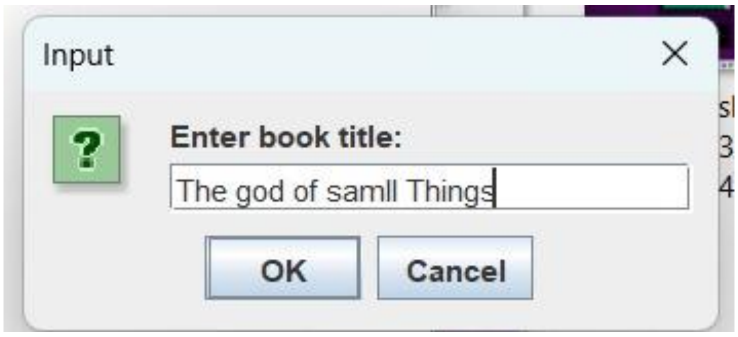
Login

Username: admin

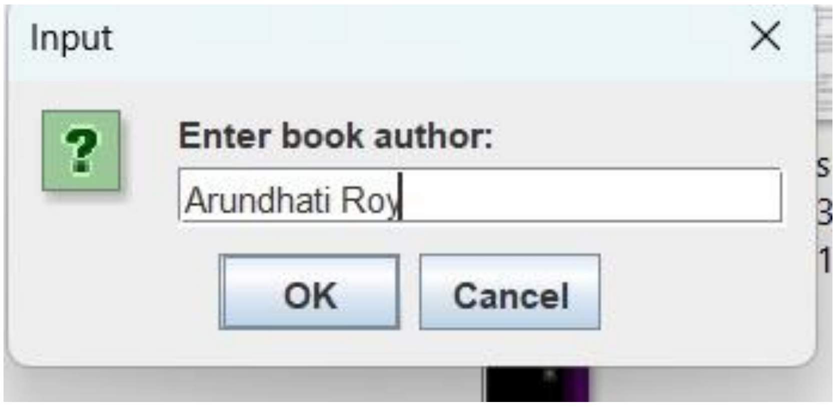
Password: •••••



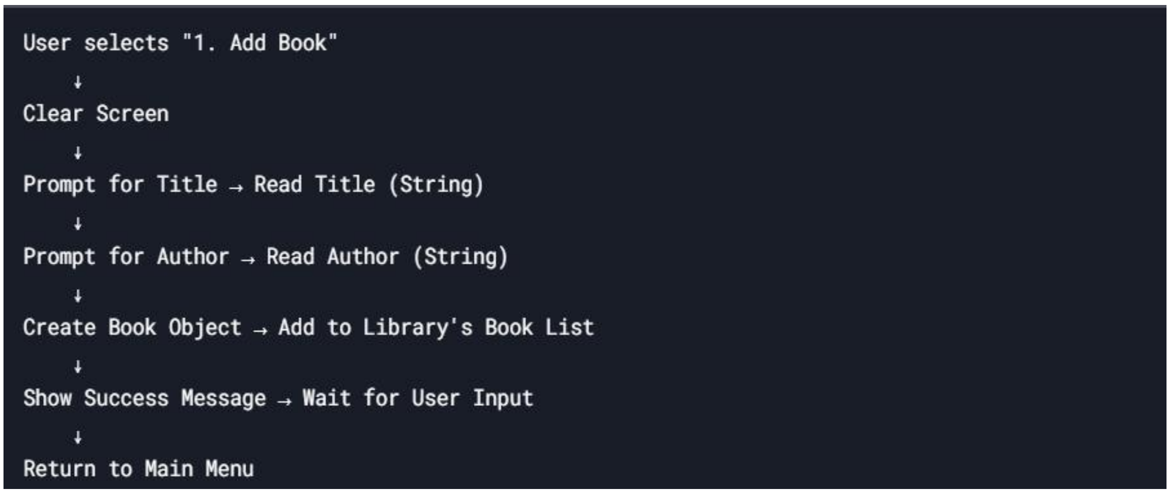
b) Add book :



The god of samll Things

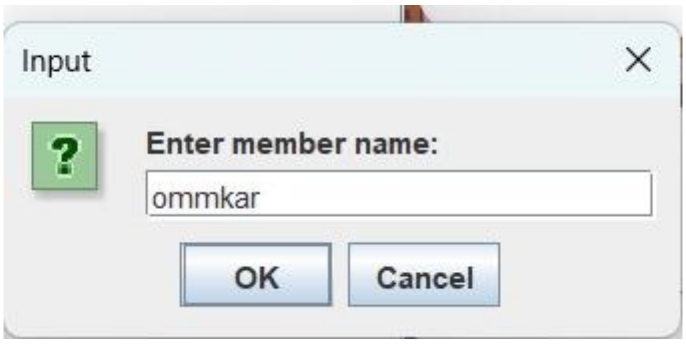


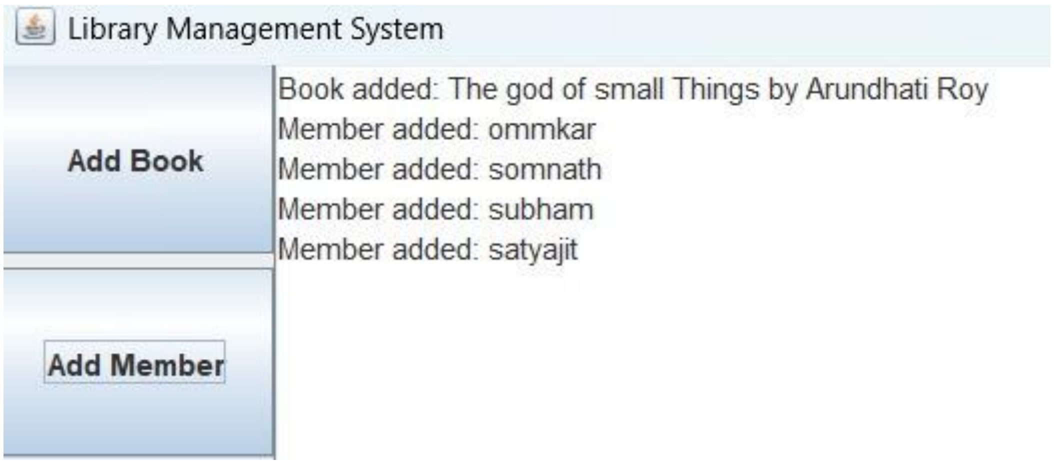
Arundhati Roy



```
User selects "1. Add Book"
    ↓
Clear Screen
    ↓
Prompt for Title → Read Title (String)
    ↓
Prompt for Author → Read Author (String)
    ↓
Create Book Object → Add to Library's Book List
    ↓
Show Success Message → Wait for User Input
    ↓
Return to Main Menu
```

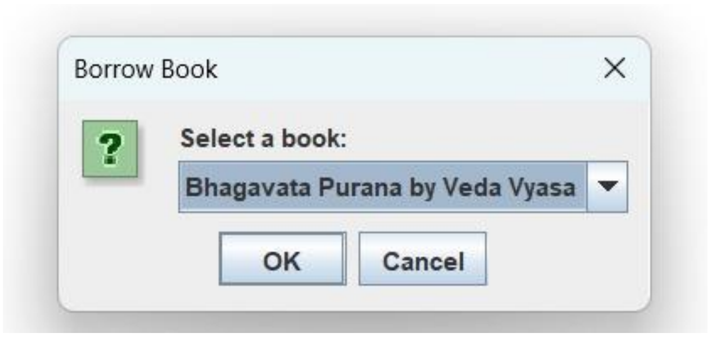
c) Add members:

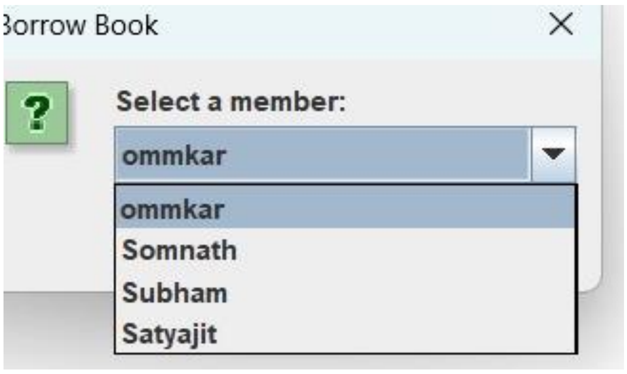




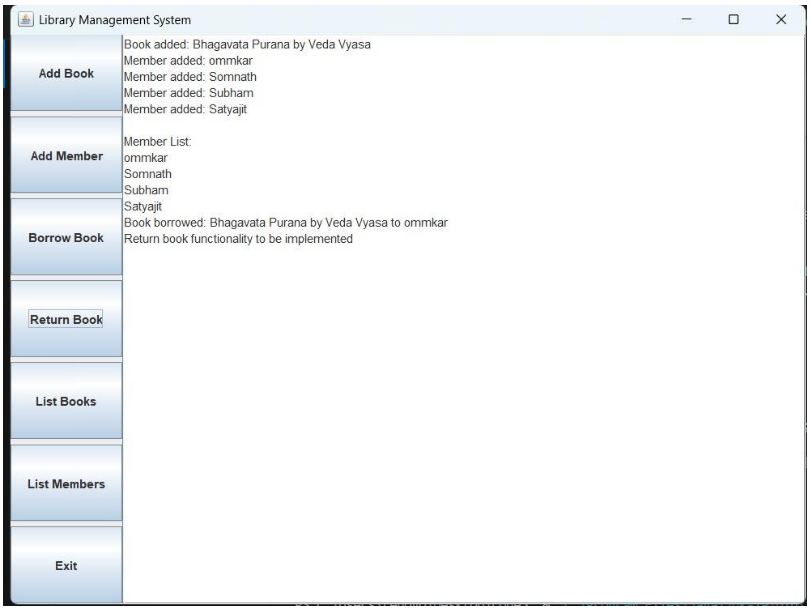
```
Book added: The god of small Things by Arundhati Roy
Member added: ommkar
Member added: somnath
Member added: subham
Member added: satyajit
```

d) Borrow book:



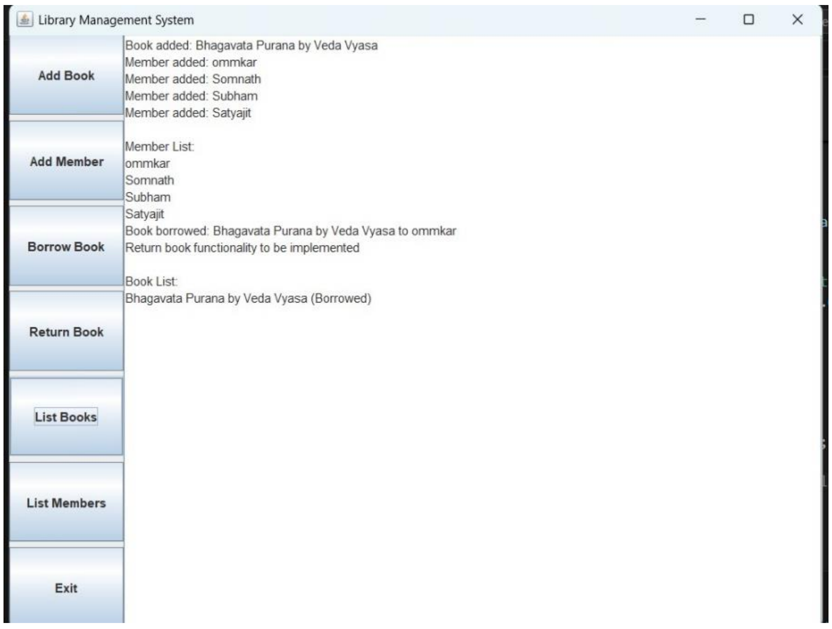


4. Return book -:



```
Book added: Bhagavata Purana by Veda Vyasa
Member added: ommkar
Member added: Somnath
Member added: Subham
Member added: Satyajit

Member List:
ommkar
Somnath
Subham
Satyajit
Book borrowed: Bhagavata Purana by Veda Vyasa to ommkar
Return book functionality to be implemented
```

5 final interface -:



```
Book added: Bhagavata Purana by Veda Vyasa
Member added: ommkar
Member added: Somnath
Member added: Subham
Member added: Satyajit

Member List:
ommkar
Somnath
Subham
Satyajit
Book borrowed: Bhagavata Purana by Veda Vyasa to ommkar
Return book functionality to be implemented

Book List:
Bhagavata Purana by Veda Vyasa (Borrowed)
```

5. Conclusion -:

The Java-based Library Management System successfully addresses the inefficiencies of traditional library management by automating core operations such as book tracking, member registration, and transaction handling. Designed with object-oriented programming (OOP) principles, the system demonstrates:

1. Modularity : Clear separation of concerns between classes like Book, Member, and libraryManagementSystem.
2. Scalability : A foundation that can be extended with advanced features like database integration or GUI development.
3. User-Friendliness : A console-based interface that simplifies tasks for librarians while minimizing errors.

Key Achievements

- Streamlined borrowing/returning workflows with real-time status updates.
- Efficient management of library resources using Java’s ArrayList data structure.
- Input validation to handle basic edge cases (e.g., invalid menu choices).

Limitations

- No Data Persistence : Data is lost when the program terminates.
- Basic UI : Limited to a console interface without advanced visualizations.
- Lack of Security : No authentication mechanism for librarians/members.

Future Directions

1. Enhance Features :
   - Add due dates, fines for late returns, and automated reminders.
   - Implement search functionality for books/members.
2. Improve Technology Stack :
   - Integrate a database (e.g., MySQL) for persistent storage.
   - Develop a GUI using JavaFX or Swing for a modern user experience.
3. Expand Scope :
   - Introduce roles (admin, member) with access control.
   - Enable online reservations and barcode scanning for books.

This project underscores the power of Java in building scalable, real-world applications. While the current system is functional, it lays the groundwork for future enhancements, aligning with evolving library needs. By addressing its limitations and integrating modern tools, this system can evolve into a comprehensive solution for libraries aiming to digitize their operations and improve user satisfaction.

6. References -:

1. Tools & Technologies

Visual Studio Code (VS Code)
Microsoft. (n.d.). Visual Studio Code Documentation.
Retrieved from https://code.visualstudio.com/docs
Reason: Used as the Integrated Development Environment (IDE) for writing, testing, and debugging Java code.
Java Development Kit (JDK)
Oracle. (n.d.). Java SE Documentation.
Retrieved from https://www.oracle.com/java/technologies/javase-documentation.html

2. Java Programming Resources

Horstmann, C. S., & Cornell, G. (2019). Core Java Volume I – Fundamentals (12th ed.). Prentice Hall.
Reason: Guided OOP design and Java syntax best practices.
Bloch, J. (2017). Effective Java (3rd ed.). Addison-Wesley.
Reason: Influenced code structure and error-handling strategies.

3. AI-Assisted Development

DeepSeek-R1 (2024).
DeepSeek.
Role: Assisted with code logic design, debugging, and documentation.
Platform: https://www.deepseek.com/