LIBRARY MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

ARUNA SHIVANI A S(2303811710422013)

in partial fulfillment of requirements for the award of the course

CGB1201 - JAVA PROGRAMMING

In

COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

NOVEMBER- 2024

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report on **"LIBRARY MANAGEMENT SYSTEM"** is the bonafide work of **ARUNA SHIVANI A S (2303811710422013)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

CGB1201-JAVA PROGRAMMING Dr.A.DELPHIN CAROLINA RANI, M.E.,Ph.D., HEAD OF THE DEPARTMENT PROFESSOR [illegible]	CGB1201-JAVA PROGRAMMING Mr. M. SARAVANAN, M.E., INTERNAL EXAMINER ASSISTANT PROFESSOR [illegible]
SIGNATURE	**SIGNATURE**
Dr.A.Delphin Carolina Rani, M.E.,Ph.D.,	Mr. M. Saravanan, M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
PROFESSOR	ASSISTANT PROFESSOR
Department of CSE	Department of CSE
K.Ramakrishnan College of Technology (Autonomous)	K.Ramakrishnan College of Technology (Autonomous)
Samayapuram–621112.	Samayapuram–621112.

Submitted for the viva-voce examination held on 02.12.2024

CGB1201-JAVA PROGRAMMING Mr.MADARMANNAN A, M.E., INTERNAL EXAMINER ASSISTANT PROFESSOR [illegible]	CGB1201-JAVA PROGRAMMIN Dr.R.SENTHAMILSELVI, M.E., Ph. EXTERNAL EXAMINER PROFESSOR 8138-SCE, TRICHY. [illegible]
INTERNAL EXAMINER	EXTERNAL EXAMINER

DECLARATION

I declare that the project report on **"LIBRARY MANAGEMENT SYSTEM"** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of **BACHELOR OF ENGINEERING**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1201 - JAVA PROGRAMMING.**

.

Signature

[illegible]

ARUNA SHIVANI A S

Place: Samayapuram

Date: 02/12/2024

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.,** Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. A. DELPHIN CAROLINA RANI, M.E.,Ph.D.,** Head of the department, **COMPUTER SCIENCE AND ENGINEERING** for providing her encourage pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **Mr. M. SARAVANAN, M.E.,** Department of **COMPUTER SCIENCE AND ENGINEERING,** for his incalculable suggestions,creativity, assistance and patience which motivated us to carry out this project.

I render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To serve the society by offering top-notch technical education on par with global standards

MISSION OF THE INSTITUTION

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of the industry and society.
- Be an institute with world class research facilities
- Be an institute nurturing talent and enhancing the competency of students to transform them as all-round personality respecting moral and ethical values

VISION OF DEPARTMENT

To be a center of eminence in creating competent software professionals with research and innovative skills.

MISSION OF DEPARTMENT

M1: Industry Specific: To nurture students in working with various hardware and software platforms inclined with the best practices of industry.

M2: Research: To prepare students for research-oriented activities.

M3: Society: To empower students with the required skills to solve complex technological problems of society.

PROGRAM EDUCATIONAL OBJECTIVES

1. PEO1: Domain Knowledge

To produce graduates who have strong foundation of knowledge and skills in the field of Computer Science and Engineering.

2. PEO2: Employability Skills and Research

To produce graduates who are employable in industries/public sector/research organizations or work as an entrepreneur.

3. PEO3: Ethics and Values

To develop leadership skills and ethically collaborate with society to tackle real-world challenges.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The **Library Management System** is a desktop application developed using Java's AWT (Abstract Window Toolkit) to manage users, books, and borrowed books in a library setting. The system allows administrators to perform various tasks such as adding and deleting users, adding and removing books, borrowing and returning books, viewing lists of books and users, and managing overdue fines. The program relies on three key classes: Book, which contains information about each book including its ID, title, and availability status; User, representing a library user with a unique ID and name; and BorrowedBook, which keeps track of borrowed books along with their due dates. The graphical interface uses buttons to trigger specific actions, with the results displayed in a text area for user feedback. While the system provides basic functionality for library management, it can be expanded by adding features like persistent data storage, improved error handling, and fine calculations based on overdue days. This system serves as a foundational tool for library management tasks and can be further refined to accommodate larger-scale applications.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD JAVA APPLICATIONS FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
The **Library Management System** is designed to automate and streamline library operations, such as adding books, borrowing, returning, and managing users. It provides an intuitive interface for easy management of books and users. Key features include tracking due dates, viewing books, and handling user details. The system aims to improve efficiency and reduce manual workload in libraries. It enhances the user experience by providing quick access to books and managing library transactions seamlessly.	**PO1 -3** **PO2 -3** **PO3 -3** **PO4 -3** **PO5 -3** **PO6 -3** **PO7 -3** **PO8 -3** **PO9 -3** **PO10 -3** **PO11-3** **PO12 -3**	**PSO1 -3** **PSO2 -3** **PSO3 -3**

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of the Library Management System is to provide an efficient and user-friendly solution for managing library operations, including user and book management, borrowing and returning books, and handling overdue fines. The system aims to automate key library tasks such as adding and deleting users, managing books, and tracking their availability. It allows for borrowing books with due dates and returning them while updating their availability status. Additionally, the system manages fines for overdue books, ensuring that users arenotified of any penalties.

1.2 Overview

The Library Management System is a software application designed to automate and streamline the core operations of a library. Built using Java and the Abstract Window Toolkit (AWT), it provides a user-friendly graphical interface for managing users, books, and borrowed books. The system allows administrators to add or remove users and books, track the availability of books, and manage borrowing and returns. When a user borrows a book, the system assigns a due date, and when the book is returned, its availability is updated. Additionally, the system includes functionality to manage overdue fines, ensuring users are informed of any fees for late returns. The application is designed for simplicity and efficiency, providing immediate feedback for each action performed, such as confirming successful book borrowing or returning.

1.3 Java Programming Concepts

Encapsulation: Encapsulation is the bundling of data (variables) and methods (functions) that operate on that data into a single unit (class). It restricts direct access to certain components of an object, making the code more secure and modular.

Example: The Book, User, and BorrowedBook classes encapsulate the attributes (id, title, etc.) and provide controlled access through getter methods like getId() and getTitle().

Inheritance: Inheritance allows one class (child class) to inherit properties and behaviors (methods) from another class (parent class). This promotes code reuse and establishes a hierarchy between classes.

Example:

While inheritance is not directly implemented in the provided code, if there were a parent class like LibraryEntity and Book or User inherited from it, that would demonstrate inheritance.

Polymorphism: Polymorphism allows methods to have the same name but behave differently based on the context. It is achieved via method overloading (compile time polymorphism) and method overriding (runtime polymorphism).

Example:

If there were multiple display() methods in a class, one for books and another for users, the behavior of display() would change depending on the object.

Abstraction: Abstraction hides the implementation details and only exposes the essential features of an object. It simplifies complexity and increases readability and usability.

Example:

The User and Book classes abstract the concept of a library system entity, focusing only on key attributes like id and title instead of exposing how the library manages or stores these entities internally.

Java AWT and Swing

The code uses AWT (java.awt.*) and Swing (javax.swing.*) for building a GUI:

Components:

Frame: The main window of the application.

Label: Displays static text like the title.

Button: Performs actions when clicked (e.g., "ADD BOOK").

Panel: Groups and organizes components.

JOptionPane: Provides dialog boxes for user input and messages.

Layouts:

BorderLayout and GridLayout are used for arranging components within the GUI.

Event Handling:

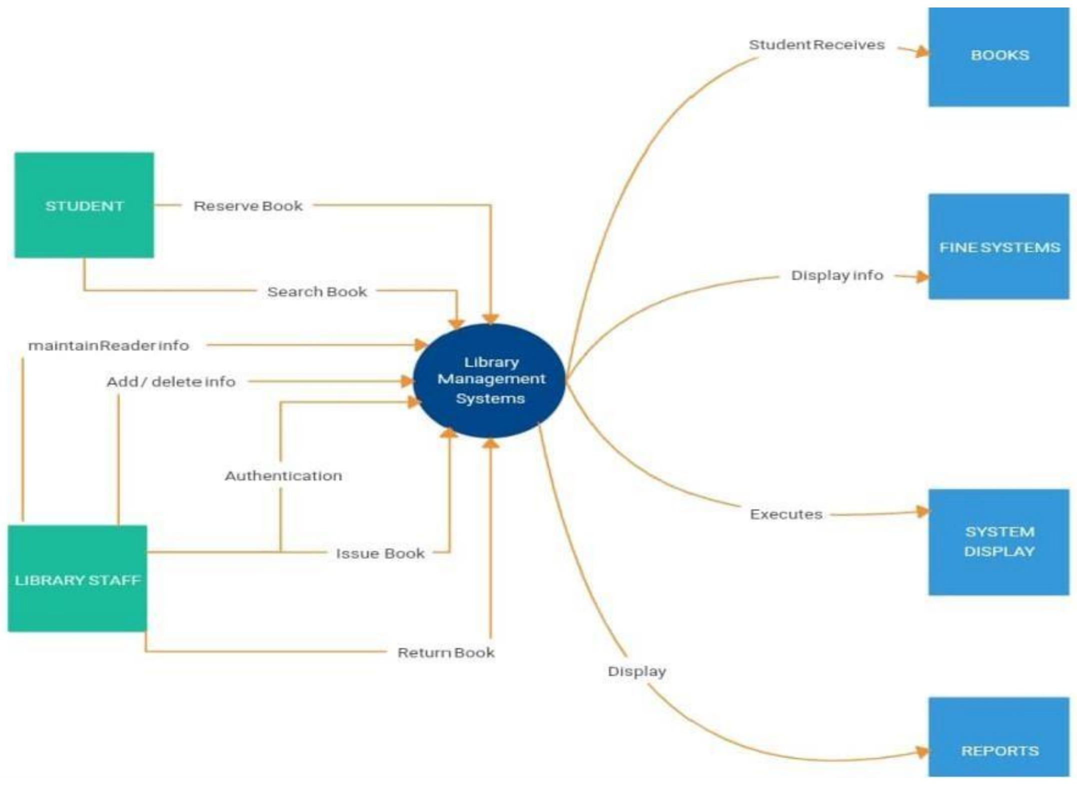
Action Listener: Implements ActionListener to define actions triggered by button clicks (e.g., addBookBtn.addActionListener).

CHAPTER 2
PROJECT METHODOLOGY

2.1 Proposed Work

The proposed work focuses on developing a robust Library Management System with an interactive graphical user interface (GUI) to streamline library operations. The system will enable users to perform essential tasks such as adding books, viewing available books, borrowing and returning books, managing users, and checking due dates. It leverages the principles of Object-Oriented Programming (OOP) for modular and scalable design, with entities like Book, User, and BorrowedBook representing core components. The implementation utilizes Java AWT and Swing for GUI development, creating a visually appealing and user- friendly interface. Dynamic data handling is achieved through the Collections Framework (ArrayList) to manage books, users, and transactions efficiently.

2.2 Block Diagram



CHAPTER 3
MODULE DESCRIPTION

3.1 User Management Module

This module manages all operations related to library users.

Features:

- Add new users (addUser method).
- View all registered users (viewUsers method).
- Remove a user from the system (removeUser method).

Classes:

- User: Represents a library user with attributes such as userId and userName.

Methods:

- findUserById: Utility method to search for a user by their ID.

3.2 Book Management Module

This module handles the management of books in the library.

Features:

- Add new books (addBook method).
- View the list of all books (viewBooks method).

Classes:

- Book: Represents a book with attributes such as id, title, and author.

Methods:

- findBookById: Utility method to search for a book by its ID.

3.3 Borrow and Return Module

This module manages book borrowing and returning functionalities.

Features:

- Borrow books (borrowBook method), including due date calculation.
- Return borrowed books (returnBook method).
- Check due dates for borrowed books (checkDue method).

Classes:

- BorrowedBook: Represents a borrowed book with details like userId, book, and dueDate.

Methods:

- findBorrowedBookById: Utility method to locate borrowed books by user and book IDs.
- Due date calculation is implemented using Calendar and Date.

3.4 Graphical User Interface (GUI) Module

This module creates and manages the application's user interface.

Features:

- Provides a main window with buttons for all system operations.
- Displays input and message dialogs for interaction.

Components:

- AWT Components: Frame, Label, Panel, Button, GridLayout, BorderLayout.

- Swing Components: JOptionPane for user input and alerts.

Methods:

- Action listeners for each button to trigger respective functionality.

3.5 Utility Module

This module provides utility functions used across the system.

Features:

- ID generation for users and books (userIdCounter, bookIdCounter).
- Dynamic data handling with ArrayList for users, books, and borrowed books.

Classes:

- Reusable utility classes and helper methods to simplify operations.

CHAPTER 4
CONCLUSION & FUTURE SCOPE

4.1 CONCLUSION

In conclusion, the Library Management System provides a comprehensive solution for automating key library functions, such as user management, book management, borrowing and returning books, and fine management. With its simple yet effective graphical interface, the system ensures that library operations are streamlined, efficient, and easy to use. By allowing for the seamless tracking of book availability and overdue fines, it enhances the overall user experience and reduces administrative workload.While the current version is suited for small-scale libraries, the system has the potential to be expanded and refined with additional features like persistent data storage, advanced fine calculations, and more robust error handling.

4.2 FUTURE SCOPE

The Library Management System is a foundational tool for managing books, users, and borrowing processes. Future enhancements include integrating a database for persistent storage and scalability. It can be transformed into a web or mobile application for broader accessibility. Features like user authentication, late fee tracking, automated notifications, and book renewals will enhance usability. Advanced search options and e-book support can improve resource management, while analytics and reporting will provide insights into usage trends. Deploying the system on cloud platforms will support large-scale operations, and multilingual support will ensure inclusivity. AI integration can add features like book recommendations and chatbots for assistance. This system can be used in schools, libraries, corporate spaces, and online book rental platforms. With these improvements, it could become a versatile tool for modern library management.

APPENDIX A

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.ArrayList;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Calendar;

public class Main {

    // List to simulate the in-memory database for books and users
    private static ArrayList<Book> books = new ArrayList<>();
    private static ArrayList<User> users = new ArrayList<>();

    // List to track borrowed books
    private static ArrayList<BorrowedBook> borrowedBooks = new ArrayList<>();

    // User ID tracker
    private static int userIdCounter = 1; // Starting User ID
    private static int bookIdCounter = 1;  // Starting Book ID

    // Main method to launch the GUI
    public static void main(String[] args) {
        // Create the main window (Frame)
        Frame mainFrame = new Frame("Library Management System");
        mainFrame.setLayout(new BorderLayout());
```

```
   // Set background color for the main frame
        mainFrame.setBackground(new Color(240, 248, 255)); // Alice Blue
background for a soft look

        // Create a large title at the top
        Label titleLabel = new Label("LIBRARY MANAGEMENT SYSTEM",
Label.CENTER);
        titleLabel.setFont(new Font("Courier", Font.BOLD, 24));
        titleLabel.setForeground(new Color(0, 0, 139)); // Steel Blue color for the title
        mainFrame.add(titleLabel, BorderLayout.NORTH);

        // Create a panel for the buttons
        Panel buttonPanel = new Panel();
        buttonPanel.setLayout(new GridLayout(6, 2, 10, 10));  // 6 rows, 2 columns,
with spacing between buttons
        buttonPanel.setBackground(Color.WHITE); // White Background for buttons
panel

        // Create buttons for each operation
        Button addBookBtn = new Button("ADD BOOK");
        Button viewBooksBtn = new Button("VIEW BOOKS");
        Button borrowBookBtn = new Button("BORROW BOOK");
        Button returnBookBtn = new Button("RETURN BOOK");
        Button checkDueBtn = new Button("CHECK DUE");
        Button removeUserBtn = new Button("REMOVE USER");
        Button viewUsersBtn = new Button("VIEW USERS");
        Button addUserBtn = new Button("ADD USER");
```

```
// Set button background color and foreground color with soft mild colors
    addBookBtn.setBackground(new Color(173, 216, 230));
    viewBooksBtn.setBackground(new Color(240, 230, 140));
    borrowBookBtn.setBackground(new Color(144, 238, 144));
    returnBookBtn.setBackground(new Color(255, 228, 181));
    checkDueBtn.setBackground(new Color(255, 239, 213));
    removeUserBtn.setBackground(new Color(255, 182, 193));
    viewUsersBtn.setBackground(new Color(221, 160, 221));
    addUserBtn.setBackground(new Color(216, 191, 216));

    // Set larger size for buttons and increased font size with Courier font
    Font largeFont = new Font("Courier", Font.BOLD, 16);

    addBookBtn.setFont(largeFont);
    viewBooksBtn.setFont(largeFont);
    borrowBookBtn.setFont(largeFont);
    returnBookBtn.setFont(largeFont);
    checkDueBtn.setFont(largeFont);
    removeUserBtn.setFont(largeFont);
    viewUsersBtn.setFont(largeFont);
    addUserBtn.setFont(largeFont);

    // Set button size
    addBookBtn.setPreferredSize(new Dimension(200, 50));
    viewBooksBtn.setPreferredSize(new Dimension(200, 50));
    borrowBookBtn.setPreferredSize(new Dimension(200, 50));
    returnBookBtn.setPreferredSize(new Dimension(200, 50));
```

```
checkDueBtn.setPreferredSize(new Dimension(200, 50));
    removeUserBtn.setPreferredSize(new Dimension(200, 50));
    viewUsersBtn.setPreferredSize(new Dimension(200, 50));
    addUserBtn.setPreferredSize(new Dimension(200, 50));

    // Add action listeners to buttons
    addBookBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            addBook();  // Trigger add book functionality
        }
    });

    viewBooksBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            viewBooks();  // View books
        }
    });

    borrowBookBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            borrowBook();  // Borrow a book
        }
    });

    returnBookBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            returnBook();  // Return a book
}
    });

    checkDueBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            checkDue();  // Check due date
        }
    });

    removeUserBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            removeUser();  // Remove a user
        }
    });

    viewUsersBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            viewUsers();  // View all users
        }
    });

    addUserBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            addUser();  // Add a user
        }
    });
```

```
     // Add buttons to the panel
        buttonPanel.add(addBookBtn);
        buttonPanel.add(viewBooksBtn);
        buttonPanel.add(borrowBookBtn);
        buttonPanel.add(returnBookBtn);
        buttonPanel.add(checkDueBtn);
        buttonPanel.add(removeUserBtn);
        buttonPanel.add(viewUsersBtn);
        buttonPanel.add(addUserBtn);

        // Create a simple border effect with a background color
        Panel borderPanel = new Panel();
        borderPanel.setLayout(new BorderLayout());
        borderPanel.setBackground(Color.WHITE);
        borderPanel.add(buttonPanel, BorderLayout.CENTER);

        // Add the button panel to the main frame
        mainFrame.add(borderPanel, BorderLayout.CENTER);

        // Set frame properties
        mainFrame.setSize(500, 650); // Increased the size of the main frame to
accommodate the new button
        mainFrame.setVisible(true);

        // Handle window close event
        mainFrame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
```

```
System.exit(0);  // Close the application when the window is closed
            }
        });
    }

    // Add book functionality
    private static void addBook() {
        String title = JOptionPane.showInputDialog("Enter book title:");
        String author = JOptionPane.showInputDialog("Enter book author:");

        if (title != null && author != null && !title.trim().isEmpty() &&
!author.trim().isEmpty()) {
            Book newBook = new Book(bookIdCounter++, title, author);
            books.add(newBook);
            JOptionPane.showMessageDialog(null, "Book added: " +
newBook.getTitle() + " by " + newBook.getAuthor());
        } else {
            JOptionPane.showMessageDialog(null, "Book title and author cannot be
empty.");
        }
    }

    // View all books
    private static void viewBooks() {
        if (books.isEmpty()) {
            JOptionPane.showMessageDialog(null, "No books available.");
        } else {
```

```
 StringBuilder bookList = new StringBuilder();
            for (Book book : books) {
                bookList.append("ID:          ").append(book.getId()).append(",          Title:
").append(book.getTitle()).append(",                                                        Author:
").append(book.getAuthor()).append("\n");
            }
            JOptionPane.showMessageDialog(null, bookList.toString());
        }
    }

    // Borrow book functionality
    private static void borrowBook() {
        String userIdInput = JOptionPane.showInputDialog("Enter User ID:");
        if (userIdInput != null && !userIdInput.trim().isEmpty()) {
            int userId = Integer.parseInt(userIdInput.trim());
            User user = findUserById(userId);
            if (user != null) {
                String bookIdInput = JOptionPane.showInputDialog("Enter Book ID:");
                if (bookIdInput != null && !bookIdInput.trim().isEmpty()) {
                    int bookId = Integer.parseInt(bookIdInput.trim());
                    Book book = findBookById(bookId);
                    if (book != null) {
                        // Record the borrowed book and due date
                        Calendar cal = Calendar.getInstance();
                        cal.add(Calendar.DAY_OF_YEAR, 14);  // 2 weeks due date
                        Date dueDate = cal.getTime();
```

```
   BorrowedBook borrowedBook = new BorrowedBook(userId, book, dueDate);
                        borrowedBooks.add(borrowedBook);
                        JOptionPane.showMessageDialog(null,           "THANKS           FOR
BORROWING. Due Date: " + dueDate);
                    } else {
                        JOptionPane.showMessageDialog(null, "Book not found.");
                    }
                }
            } else {
                JOptionPane.showMessageDialog(null, "User not found.");
            }
        }
    }

    // Return borrowed book
    private static void returnBook() {
        String userIdInput = JOptionPane.showInputDialog("Enter User ID:");
        if (userIdInput != null && !userIdInput.trim().isEmpty()) {
            int userId = Integer.parseInt(userIdInput.trim());
            User user = findUserById(userId);
            if (user != null) {
                String bookIdInput = JOptionPane.showInputDialog("Enter Book ID:");
                if (bookIdInput != null && !bookIdInput.trim().isEmpty()) {
                    int bookId = Integer.parseInt(bookIdInput.trim());
                    BorrowedBook    borrowedBook    =    findBorrowedBookById(userId,
bookId);
                    if (borrowedBook != null) {

borrowedBooks.remove(borrowedBook);
```

```
                    JOptionPane.showMessageDialog(null,          "Book          returned
successfully.");
                } else {
                    JOptionPane.showMessageDialog(null,    "This    book    was    not
borrowed by the user.");
                }
            }
        } else {
            JOptionPane.showMessageDialog(null, "User not found.");
        }
    }
}

// Check due date for borrowed books
private static void checkDue() {
    String userIdInput = JOptionPane.showInputDialog("Enter User ID:");
    if (userIdInput != null && !userIdInput.trim().isEmpty()) {
        int userId = Integer.parseInt(userIdInput.trim());
        User user = findUserById(userId);
        if (user != null) {
            StringBuilder dueList = new StringBuilder();
            for (BorrowedBook borrowedBook : borrowedBooks) {
                if (borrowedBook.getUserId() == userId) {
                    dueList.append("Book:
").append(borrowedBook.getBook().getTitle())
                        .append(",Due Date:
").append(borrowedBook.getDueDate()).append("\n");
```

```
     }
              }
              if (dueList.length() > 0) {
                 JOptionPane.showMessageDialog(null, dueList.toString());
              } else {
                 JOptionPane.showMessageDialog(null, "No borrowed books for this
user.");

              }
           } else {
              JOptionPane.showMessageDialog(null, "User not found.");
           }
        }
    }

    // Remove a user
    private static void removeUser() {
       String userIdInput = JOptionPane.showInputDialog("Enter User ID to
remove:");
       if (userIdInput != null && !userIdInput.trim().isEmpty()) {
           int userId = Integer.parseInt(userIdInput.trim());
           User user = findUserById(userId);
           if (user != null) {
              users.remove(user);
              JOptionPane.showMessageDialog(null,"User removed:"+
user.getUserName());
           } else {
              JOptionPane.showMessageDialog(null, "User not found.");
```

```
 }
        }
    }

    // View all users
    private static void viewUsers() {
        if (users.isEmpty()) {
            JOptionPane.showMessageDialog(null, "No users available.");
        } else {
            StringBuilder userList = new StringBuilder();
            for (User user : users) {
                userList.append("ID:       ").append(user.getUserId()).append(",Name:
").append(user.getUserName()).append("\n");
            }
            JOptionPane.showMessageDialog(null, userList.toString());
        }
    }

    // Add a new user and display the ID
    private static void addUser() {
        String userName = JOptionPane.showInputDialog("Enter user name:");
        if (userName != null && !userName.trim().isEmpty()) {
            User newUser = new User(userIdCounter++, userName);
            users.add(newUser);
            JOptionPane.showMessageDialog(null,       "User       added:       "       +
newUser.getUserName() + "\nUser ID: " + newUser.getUserId());
        } else {
```

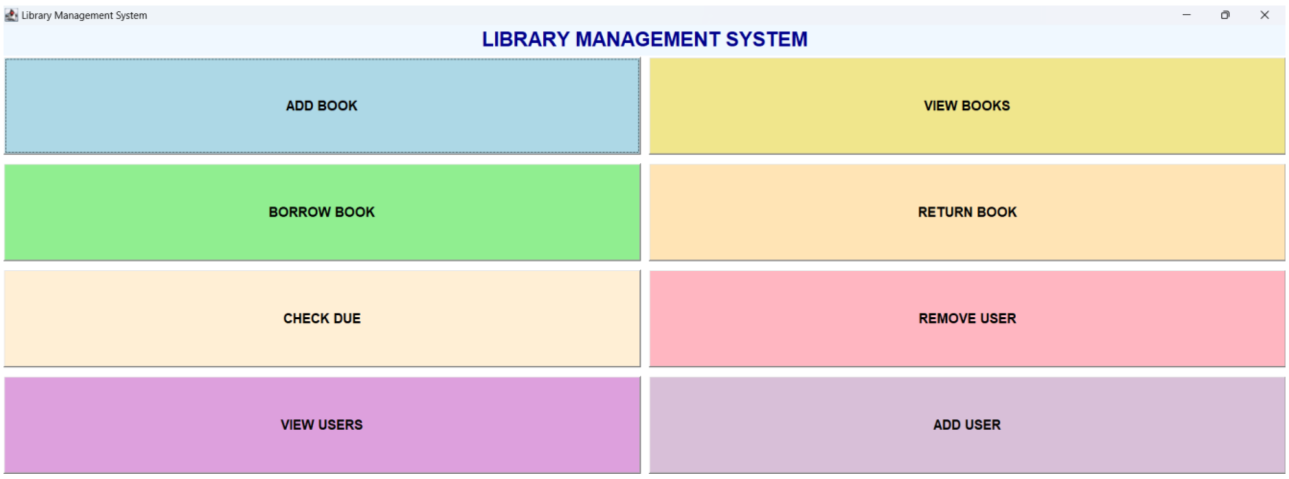
```
 JOptionPane.showMessageDialog(null, "User name cannot be empty.");
        }
    }

    // Utility methods to find books and users by their IDs
    private static Book findBookById(int id) {
        for (Book book : books) {
            if (book.getId() == id) {
                return book;
            }
        }
        return null;
    }

    private static User findUserById(int id) {
        for (User user : users) {
            if (user.getUserId() == id) {
                return user;
            }
        }
        return null;
    }

    private static BorrowedBook findBorrowedBookById(int userId, int bookId) {
        for (BorrowedBook borrowedBook : borrowedBooks) {
            if (borrowedBook.getUserId() == userId &&
borrowedBook.getBook().getId() == bookId) {
    return borrowedBook;
            }
        }
        return null;
    }
}

// Book class
class Book {
    private int id;
    private String title;
    private String author;

    public Book(int id, String title, String author) {
        this.id = id;
        this.title = title;
        this.author = author;
    }

    public int getId() {
        return id;
    }

    public String getTitle() {
        return title;
    }
```

```
  public String getAuthor() {
      return author;
    }
}

// User class
class User {
    private int  userId;
    private String userName;

    public User(int userId, String userName) {
        this.userId = userId;
        this.userName = userName;
    }

    public int getUserId() {
        return userId;
    }

    public String getUserName() {
        return userName;
    }
}

// BorrowedBook class to track borrowed books
class BorrowedBook {
    private int userId;
```

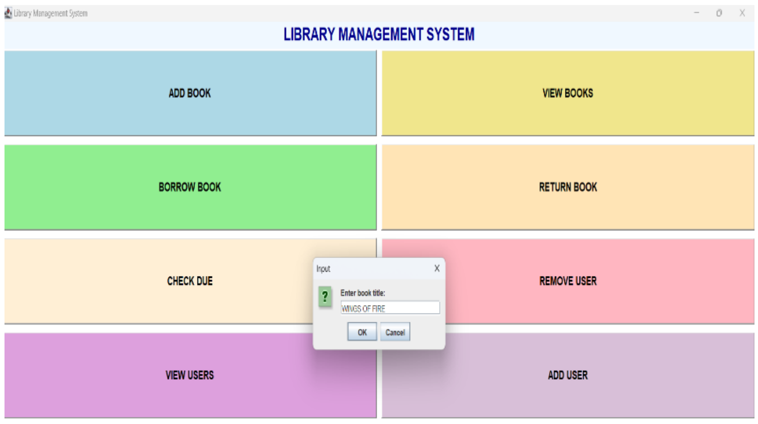
```
   private Book book;
    private Date dueDate;

    public BorrowedBook(int userId, Book book, Date dueDate) {
        this.userId = userId;
        this.book = book;
        this.dueDate = dueDate;
    }

    public int getUserId() {
        return userId;
    }

    public Book getBook() {
        return book;
    }

    public Date getDueDate() {
        return dueDate;
    }
}
```

APPENDIX B

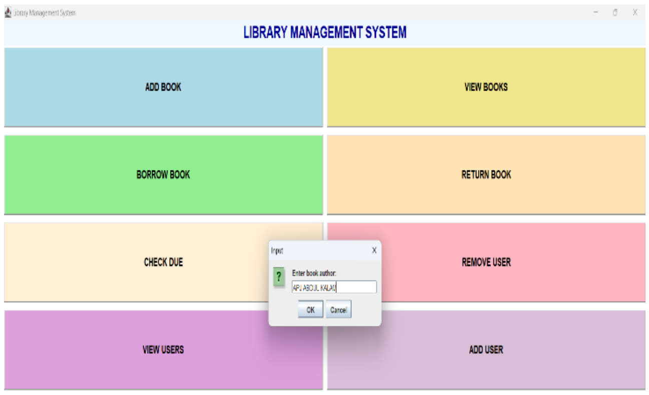
LIBRARY MANAGEMENT SYSTEM



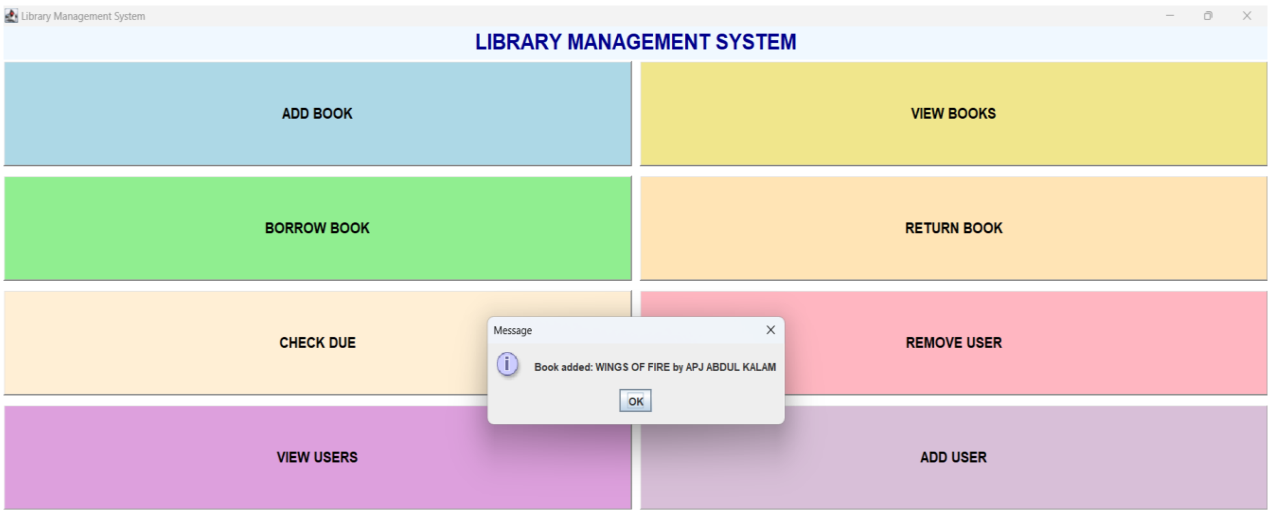
ADD BOOK



Enter book title: WINGS OF FIRE

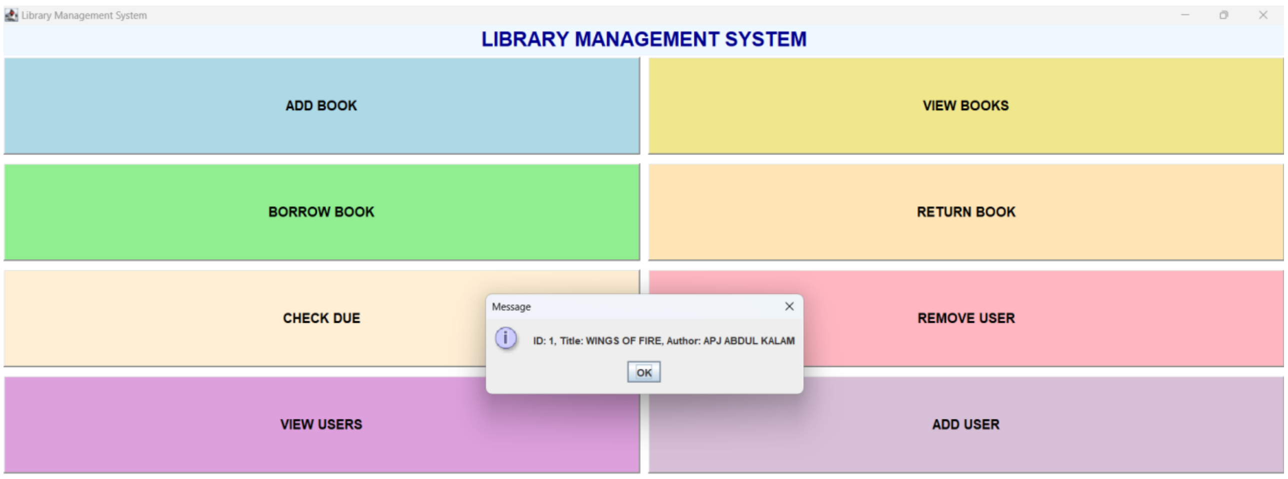


Enter book author: APJ ABDUL KALAM



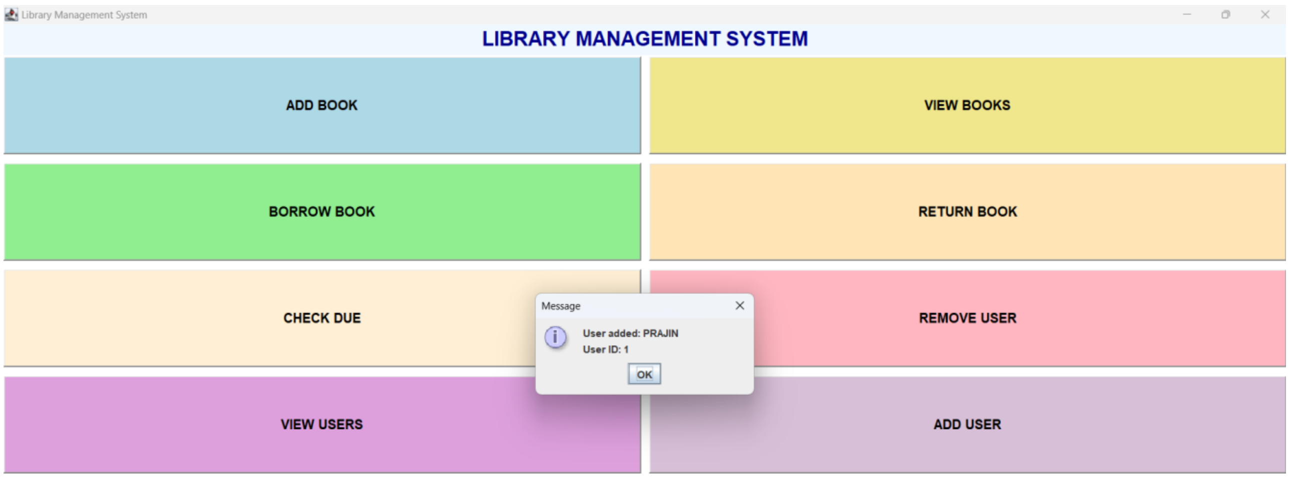
Book added: WINGS OF FIRE by APJ ABDUL KALAM

VIEW BOOK



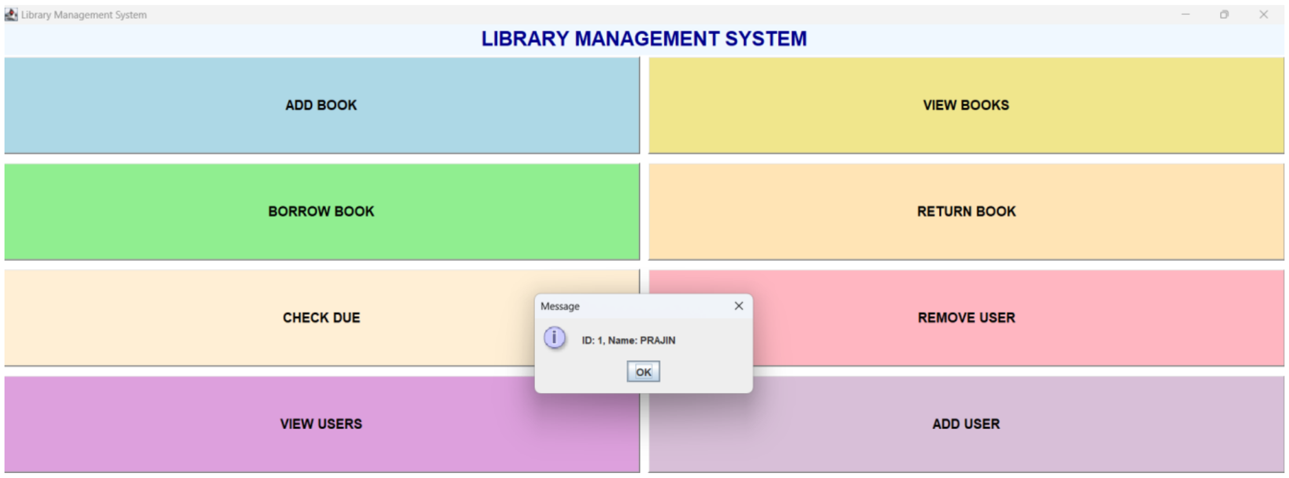
ID: 1, Title: WINGS OF FIRE, Author: APJ ABDUL KALAM

ADD USER



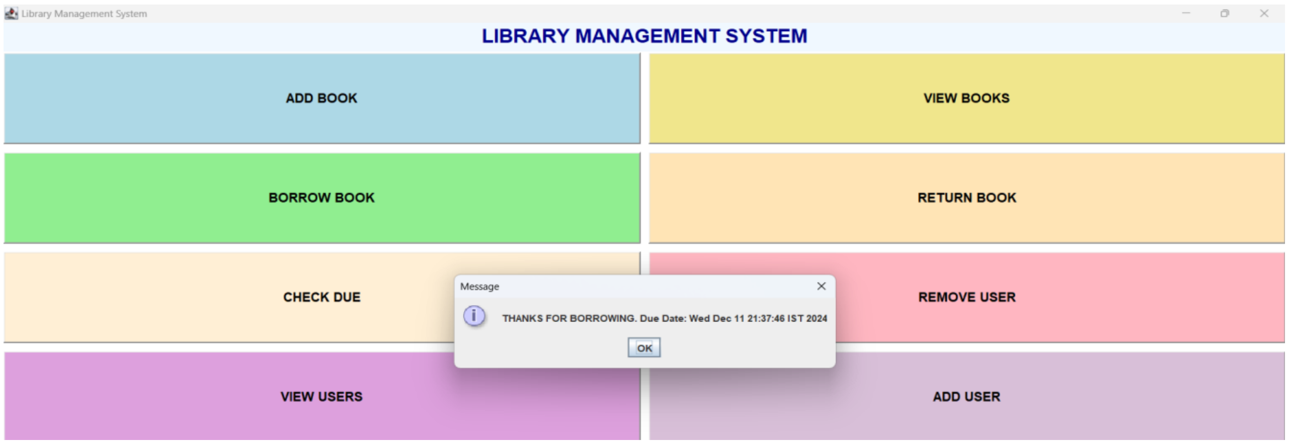
User added: PRAJIN
User ID: 1

VIEW USER



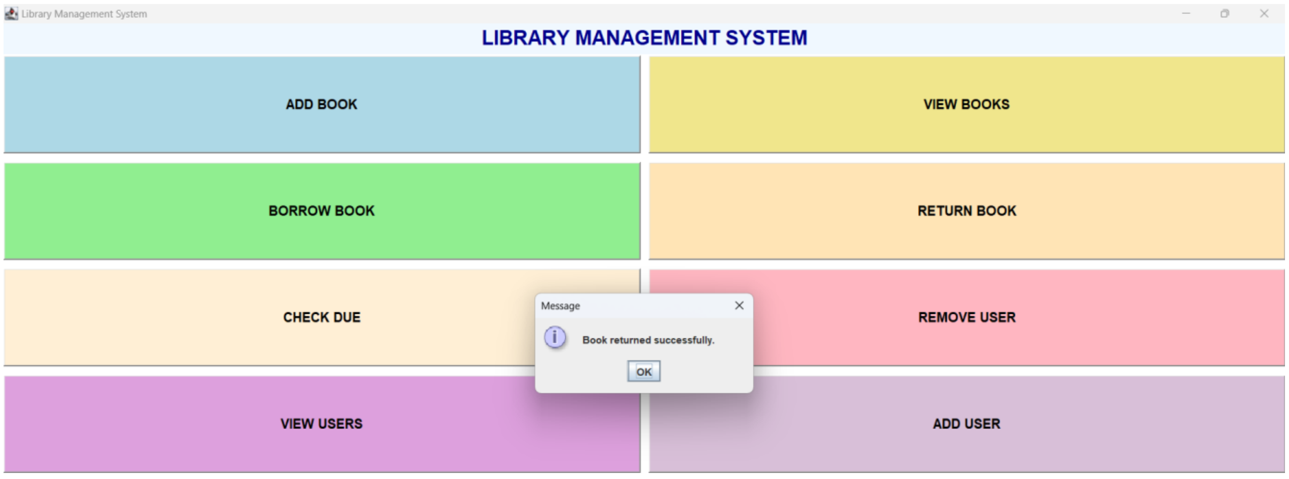
ID: 1, Name: PRAJIN

BORROW BOOK



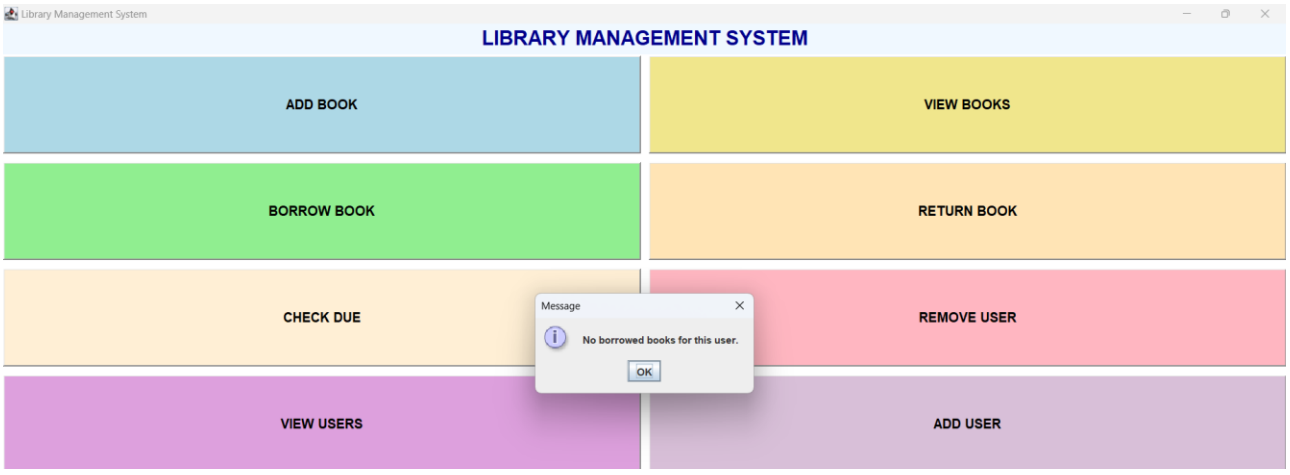
THANKS FOR BORROWING. Due Date: Wed Dec 11 21:37:46 IST 2024

RETURN BOOK



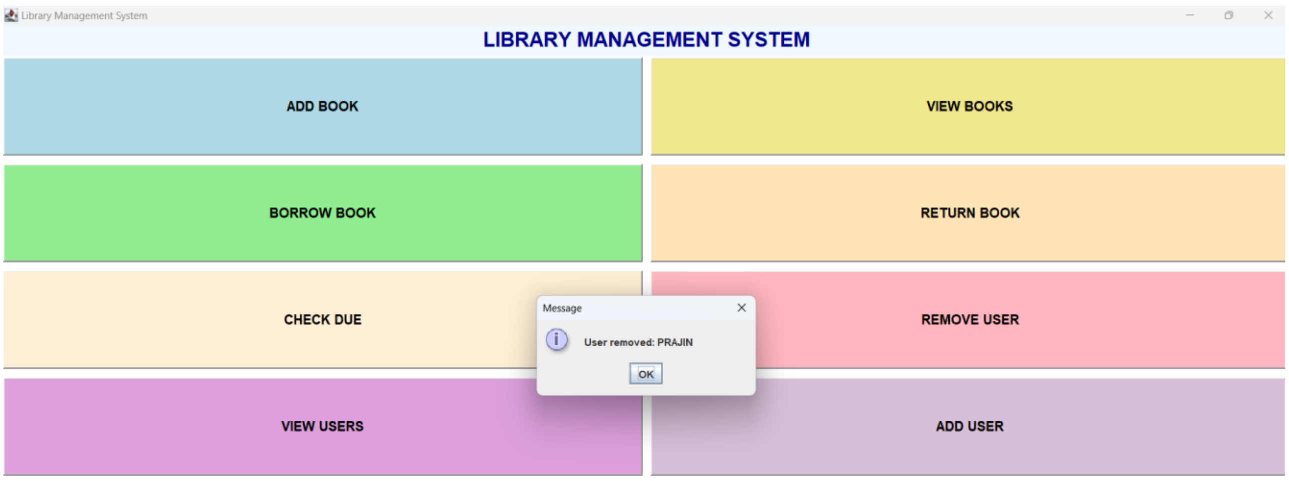
Book returned successfully.

CHECK DUE



No borrowed books for this user.

REMOVE USER



User removed: PRAJIN

REFERENCES

WEBSITES

- https://www.geeksforgeeks.org/library-management-system-using-switch-statement-in-java/
- https://www.javaguides.net/2024/04/library-management-system-project-in-java-with-source-code.html
- https://projectgurukul.org/java-library-management-system/

YOUTUBE LINK

- https://youtu.be/ukAUroj__BU
- https://youtu.be/A5zLwnUJtVc

BOOK

- "Java Projects" by Bpb Publications – Hands-on guide with real-world projects like library management.
- "Java Swing" by Marc Loy – Focuses on Java Swing for building the graphical user interface.