Project Title: Library Management System in Java

Author: [Shaikh Irafan Rashid]

Course/Department: [OOPJ/COMPUTER SCIENCE AND ENGINEERING]

Date of Submission: [11/11/2024]

TABLE OF CONTENTS

ACKNOWLEDGEMENT

We express our sincere gratitude to our mini project guide, **Ms. Nitu L. Pariyal**, for her invaluable guidance throughout this project. Working with her has been a unique and enriching experience, and we extend our thanks for her support, suggestions, and the many insightful discussions.

We would also like to acknowledge **Dr. Mrs. A. M. Rajurkar**, Head of the Computer Science & Engineering Department at MGM's College of Engineering, Nanded, for her encouragement and assistance.

We are deeply grateful to **Dr. Mrs. G. S. Lathkar**, Director of MGM's College of Engineering. Nanded, for providing the resources and support needed to carry out this project in Java, as well as for her kind guidance and inspiration. Lastly, we thank everyone who contributed, directly or indirectly, to the successful completion of this mini project.

With Deep Reverence,

Shaikh Irafan Rashid[133]

ABSTRACT

Library Management System

The Library Management System is a Java-based application developed to streamline basic library operations, including adding, displaying, borrowing, and returning books. The project is built using object-oriented programming (OOP) principles, featuring a structured approach with three main classes: Book, Library, and MAIN. The Book class represents individual book entities, storing attributes such as title, author, and availability status. The Library class manages a collection of books and provides core functionalities like searching, borrowing, and returning books, while the MAIN class acts as the user interface, allowing users to interact with the system through a menu-driven console application.

The primary goal of the project is to create a user-friendly and efficient system for managing a small-scale book inventory, demonstrating Java concepts such as class design, encapsulation, and method interactions. The system handles edge cases like checking for book availability and ensuring proper borrowing/returning procedures. Although limited to basic functionalities, this project provides a strong foundation for future enhancements such as integrating user accounts, tracking borrowed books, and scaling to support larger libraries. The project showcases practical applications of Java programming and can be expanded to meet more complex library management needs.

Class :SY -A

Shaikh Irafan Rashid[133]

Introduction

Problem Statement: The management of libraries, whether small or large, can be a time-consuming and error-prone process when done manually. Librarians face the challenge of accurately keeping track of book inventory, monitoring which books are available or borrowed, and ensuring the timely return of borrowed books. This task becomes exponentially more complex as the number of books increases, often resulting in inefficiencies, misplaced books, and user dissatisfaction. The manual approach lacks automation and real-time updates, leading to difficulties in handling library operations effectively and potential data inconsistencies.

Purpose of the Project: The purpose of this project is to develop a comprehensive yet straightforward Java-based library management system that automates the core operations associated with handling book inventory. This project aims to create a tool that allows librarians or users to easily add new books, view the available collection, search for specific titles, borrow books, and return them once read. By leveraging the object-oriented programming capabilities of Java, the project intends to build a scalable, maintainable, and easy-to-use system that streamlines these library tasks. The project not only helps reduce the manual workload but also minimizes the chances of human error, ensuring better reliability and efficiency in maintaining library records.

Scope of the Project: The scope of the **Library Management System** project is focused on implementing the fundamental functionalities required to manage a library's collection. This includes the addition of new books to the system, viewing details of all available books, searching for specific books by title, borrowing books if they are available, and returning them to update their status accordingly. Each book is stored with essential information such as the title, author, and its current availability status. The system supports a limited capacity for book storage, demonstrating how library operations can be managed efficiently in a basic setup. Advanced features such as user account management, fine calculation for overdue books, notifications for due dates, and integration with a database for larger storage capacity are not covered in this version of the project. These functionalities could be considered for future versions to make the system more comprehensive and suitable for larger-scale libraries.

This project serves as an essential model for understanding how object-oriented programming principles can be applied to real-world scenarios. The use of classes, encapsulation, and method

interactions creates a structured, maintainable codebase that can be expanded and improved over time. The **Library Management System** is thus positioned as a starting point for developing more sophisticated software solutions for library management, demonstrating the power of Java for building robust applications.

Project Requirements

Hardware Requirements: To run and develop the **Library Management System**, the following hardware specifications are recommended:

- **Computer Specifications**: A computer system with at least a dual-core processor (e.g., Intel Core i3 or equivalent) is required to ensure smooth execution of Java programs. For optimal performance, a quad-core processor or higher (e.g., Intel Core i5 or i7) is recommended.
- **Memory (RAM)**: The system should have a minimum of **4GB RAM** to handle basic Java program execution and development. For enhanced performance, especially when running an Integrated Development Environment (IDE) alongside other applications, **8GB RAM** or more is suggested.
- **Storage Space**: While the compiled Java application itself does not consume much storage, it is advisable to have at least **500 MB of free disk space** to accommodate the Java Development Kit (JDK), IDE, and project files. More space may be necessary if the system is used for other development projects or data storage.
- **Display and Peripherals**: A standard display with a minimum resolution of **1280x720 pixels** is sufficient for coding and running the application, though higher resolutions will enhance visibility. A keyboard and mouse or trackpad are essential for development tasks.

Software Requirements: The **Library Management System** relies on specific software tools and platforms to be developed and executed effectively:

- **Java Development Kit (JDK)**: Version **8 or higher** is required. The JDK includes the Java Runtime Environment (JRE) and command-line tools for compiling and running Java code. Using a more recent version, such as **JDK 11** or **JDK 17 (LTS)**, is recommended for long-term support and access to modern Java features.
- **Integrated Development Environment (IDE)**: To streamline development, an IDE such as **IntelliJ IDEA**, **Eclipse**, or **NetBeans** is highly recommended. These IDEs provide features like code auto-completion, debugging tools, and project management

capabilities. IntelliJ IDEA, known for its robust tools and user-friendly interface, is particularly suited for Java development.

- **Java Libraries**: Standard Java libraries are sufficient for this project; no additional third-party libraries are required unless future enhancements are added.

System Requirements: The **Library Management System** can be executed on any operating system that supports Java. The system requirements are as follows:

- **Operating System Compatibility**:
  - **Windows**: Windows 7, 8, 10, or 11 (with Java installed)
  - **macOS**: macOS 10.10 (Yosemite) or newer versions
  - **Linux**: Most distributions, including Ubuntu, Fedora, and Debian, with the JDK installed.
- **Java Installation**: The operating system must have the correct version of Java installed to compile and run the program. Confirm the installation using the java -version command in the terminal or command prompt.
- **Execution Environment**: A command-line interface (CLI) or terminal to run Java applications. An IDE or basic text editor with terminal support can also be used to compile and execute the code.

These hardware and software requirements provide the foundation for developing, running, and maintaining the **Library Management System** efficiently. Meeting these specifications ensures the application runs smoothly, without performance bottlenecks, allowing developers and users to experience seamless functionality.

System Design

Architecture Overview :

This project involves designing a simple library management system. The primary classes are:

1. **Book Class**: Represents a book's properties and behavior.
2. **Library Class**: Serves as a container for Book objects and manages them through various methods.
3. **MAIN Class**: Contains the main method to interact with users and invoke methods in the Library class.

Class Details

1. Book Class

- **Attributes**:
    - Name: The title of the book.
    - Author: The name of the book's author.
    - Available: A boolean flag indicating whether the book is available for borrowing.
- **Methods**:
    - getName(): Returns the name of the book.
    - getAuthor(): Returns the name of the author.
    - isAvailable(): Checks if the book is available for borrowing.
    - borrowBook(): Marks the book as borrowed (sets Available to false).
    - ReturnBook(): Marks the book as returned (sets Available to true).

2. Library Class

- **Attributes**:
    - books[]: An array or list containing Book objects.

- NumOfBook: The total number of books currently in the library.

- **Methods**:
  - AddBook(Book book): Adds a new Book object to the books collection.
  - displayBooks(): Displays all books in the library with their details.
  - BorrowBook(String bookName): Marks a book as borrowed if available.
  - ReturnBook(String bookName): Marks a book as returned.
  - SearchBook(String bookName): Searches for a book by its name and returns its details if found.

3. MAIN Class

- **Attributes**:
  - main(): The entry point of the application where the user interacts with the system, prompting for actions such as adding books, displaying available books, borrowing, or returning books.

* ULM Diagram *

Book	Library	MAIN
-Name -Author -Available	- books[] - NumOfBook	
+ getName() + getAuthor() + isAvailable() + borrowBook() + ReturnBook()	+ AddBook() + displayBooks() + BorrowBook() + ReturnBook() + SearchBook()	- main()

Implementation

The implementation of the Library Management System involves the integration and functionality of three main classes: Book, Library, and MAIN. Each class has its specific role and responsibilities that contribute to the overall functionality of the system.

1. Book Class Implementation:

- Purpose: The Book class is designed to represent a single book within the library. It holds attributes for the book's title, author, and availability status.
- **Attributes:**
    - private String Name; – stores the title of the book.
    - private String Author; – stores the name of the author.
    - private boolean Available; – a flag indicating whether the book is currently available for borrowing.
- **Constructor:**
    - public Book(String Name, String Author): Initializes a Book object with the given title and author. By default, Available is set to true, indicating that the book is available when first added to the library.
- **Methods:**
    - public String getName(): Returns the title of the book.
    - public String getAuthor(): Returns the author of the book.
    - public boolean isAvailable(): Returns the availability status of the book.
    - public void borrowBook(): Checks if the book is available and updates Available to false when borrowed. If the book is already borrowed, it outputs a message.
    - public void ReturnBook(): Sets Available to true if the book is being returned. If the book was not borrowed, an appropriate message is printed.
    - public String toString(): Overrides the toString() method to provide a string representation of the book, showing the title, author, and availability status.

2. Library Class Implementation:

- Purpose: The Library class manages a collection of Book objects and provides the functionality to add, display, borrow, return, and search for books within the library.
- **Attributes:**
    - private Book[] books; – an array to store Book objects.
    - private int NumOfBook; – a counter to keep track of the number of books currently in the library.
- **Constructor:**
    - public Library(int Capacity): Initializes the Library with a given capacity, creating an array to hold up to Capacity books and setting NumOfBook to 0.
- **Methods:**
    - public void AddBook(String Title, String Author): Adds a new book to the books array if there is space. If the array is full, it outputs an appropriate message.
    - public void displayBooks(): Prints the details of all the books currently in the library. If there are no books, it notifies the user that the library is empty.
    - public Book SearchBook(String Title): Searches for a book by title within the library and returns the Book object if found. If not, it returns null.
    - public void BorrowBook(String Title): Searches for a book by its title and checks if it is available for borrowing. If it is, the book's status is updated, and a confirmation message is printed. If the book is already borrowed or not found, appropriate messages are displayed.
    - public void ReturnBook(String title): Searches for a book by its title and updates its status to available when returned. If the book was not borrowed or not found, appropriate messages are displayed.

3. MAIN Class Implementation:

- Purpose: The MAIN class serves as the entry point for the application. It provides the user interface and handles user input to interact with the library.

- Implementation:
    - The main method initializes a Library object with a predefined capacity (e.g., 10 books).
    - A Scanner object is used to take input from the user for different operations.
    - A loop displays a menu with options for adding a book, displaying books, borrowing a book, returning a book, and exiting the application.
    - Depending on the user's choice, the relevant method from the Library class is called.
    - The switch statement handles the user's choice and ensures that the program responds with the correct operation or an error message if an invalid option is selected.

Sample Implementation Workflow:

1. **Adding a Book:**
    - The user chooses the "Add Book" option.
    - The program prompts the user for the book's title and author.
    - The AddBook method is called to add the book to the library array.
2. **Displaying Books:**
    - The user selects "Display All Books."
    - The displayBooks method prints out all book details, showing their availability status.
3. **Borrowing a Book:**
    - The user opts for "Borrow Book" and enters the title of the desired book.
    - The BorrowBook method checks if the book exists and is available. If yes, the status is updated, and the user is notified; otherwise, a relevant message is shown.
4. **Returning a Book:**

- The user selects "Return Book" and enters the title of the book they wish to return.
- The ReturnBook method updates the book's status to available if it was borrowed.

Challenges and Solutions:

- Challenge: Implementing robust error handling for cases where books are not found or operations are performed on unavailable books.
- Solution: The Library class methods include conditions and checks that print appropriate error messages and prevent operations that would otherwise lead to errors (e.g., returning a book that was never borrowed).

Testing and Validation

Testing is a crucial phase in the development of any software project. In the case of the Library Management System, it ensures that all the features (adding books, borrowing books, returning books, searching, etc.) work correctly, and that any errors or issues are identified and resolved. The goal of testing is to validate that the system behaves as expected under different scenarios and input conditions. Here's a detailed explanation of the testing and validation process for the Library Management System:

Test Plan: A test plan is essential for ensuring that all features of the system are thoroughly tested. For this project, the test plan focuses on validating the functionality of key operations such as adding a book, displaying books, borrowing a book, and returning a book.

Test Plan Goals:

- Verify the functionality of the Book and Library classes.
- Test the user interactions with the system through the MAIN class.
- Ensure that edge cases (e.g., trying to borrow a book that is already borrowed, or returning a book that wasn't borrowed) are handled correctly.
- Confirm that the program handles invalid input gracefully, providing appropriate feedback.

Test Cases:

Below are the sample test cases designed to validate different functionalities of the system:

Test Case 1: Add a Book

- Test Steps:
    1. Choose the "Add Book" option from the menu.
    2. Input a book title (e.g., "Harry Potter").
    3. Input an author name (e.g., "J.K. Rowling").
    4. Check if the book is added to the system.

- Expected Outcome: The system should confirm that the book is successfully added, and the book count in the library increases by 1.

Test Case 2: Display Books

- Test Steps:
    1. Choose the "Display All Books" option from the menu.
    2. Verify that the system correctly displays the list of books with their details (title, author, and availability).
- Expected Outcome: All books added to the library should be displayed with the correct details. If there are no books, the system should display a message: "No books available in the library."

Test Case 3: Borrow a Book (Available Book)

- Test Steps:
    1. Choose the "Borrow Book" option from the menu.
    2. Input the title of a book that is available (e.g., "Harry Potter").
    3. Verify that the book's status is changed to unavailable.
- Expected Outcome: The system should display a message indicating that the book has been successfully borrowed, and the availability status of the book should be updated to "No."

Test Case 4: Borrow a Book (Already Borrowed)

- Test Steps:
    1. Choose the "Borrow Book" option.
    2. Input the title of a book that has already been borrowed.
    3. Verify that the system prevents borrowing the book and displays an appropriate message (e.g., "The book is already borrowed.").
- Expected Outcome: The system should not allow the user to borrow a book that is already borrowed and should notify the user accordingly.

Test Case 5: Return a Book

- Test Steps:
    1. Choose the "Return Book" option.
    2. Input the title of a borrowed book (e.g., "Harry Potter").
    3. Verify that the book's status is changed to available.
- Expected Outcome: The system should confirm the successful return of the book, and the availability status should be updated to "Yes."

Test Case 6: Return a Book That Was Not Borrowed

- Test Steps:
    1. Choose the "Return Book" option.
    2. Input the title of a book that has not been borrowed.
    3. Verify that the system prevents the return and displays a message like "This book wasn't borrowed."
- Expected Outcome: The system should notify the user that the book was not borrowed and cannot be returned.

Test Case 7: Invalid Input (Non-Existing Book)

- Test Steps:
    1. Choose the "Borrow Book" or "Return Book" option.
    2. Enter a title that does not exist in the library (e.g., "NonExistentBook").
    3. Verify that the system displays a message like "Book not found."
- Expected Outcome: The system should display the appropriate error message when attempting to borrow or return a book that does not exist in the library.

Bug Tracking:

During the testing process, various bugs or issues might be discovered. These can include:

- Books not being properly marked as borrowed or returned: For example, a book might still appear as available after borrowing, or it might not become available after returning.
- Array index errors: If the system tries to access an index beyond the capacity of the books array, an ArrayIndexOutOfBoundsException could occur.
- Invalid user input handling: If the user inputs an invalid option or a non-existing book title, the system should handle such cases gracefully, ensuring that it does not crash or behave unpredictably.

Bugs should be tracked and resolved in a systematic manner, using error messages, debug logs, or an IDE's built-in debugger. Once issues are identified, they can be fixed and re-tested until the system works as expected.

Result

2.Adding Books.

1.

```
"C:\Program Files (x86)\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.3\lib\idea_rt.

--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 1
Enter book title: Rich Dad Poor Dad
Enter book Author: Robert Kiyosaki
Book Successfully Added !
```

2.

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 1
Enter book title: Angnipankh
Enter book Author: A.P.J Abdul Kalam
Book Successfully Added !
```

3.

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 1
Enter book title: The Powers Of The Mind
Enter book Author: Swami Vivekananda
Book Successfully Added !
```

4.

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 1
Enter book title: Atomic Habits
Enter book Author: James Clear
Book Successfully Added !
```

5.

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 1
Enter book title: Think and Grow Rich
Enter book Author: Napoleon Hill
Book Successfully Added !
```

Borrowing The Book.

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 3
Enter book title to borrow: Rich Dad Poor Dad
You Borrow A book Named Rich Dad Poor Dad
```

Returning The Book.

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 3
Enter book title to borrow: Rich Dad Poor Dad
You Borrow A book Named Rich Dad Poor Dad
```

Display All The Book.

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 2
Title:Rich Dad Poor Dad
Author:Robert Kiyosaki
Available:Yes
Title:Angnipankh
Author:A.P.J Abdul Kalam
Available:Yes
Title:The Powers Of The Mind
Author:Swami Vivekananda
Available:Yes
```

```
Title:Atomic Habits
Author:James Clear
Available:Yes
Title:Think and Grow Rich
Author:Napoleon Hill
Available:Yes
```

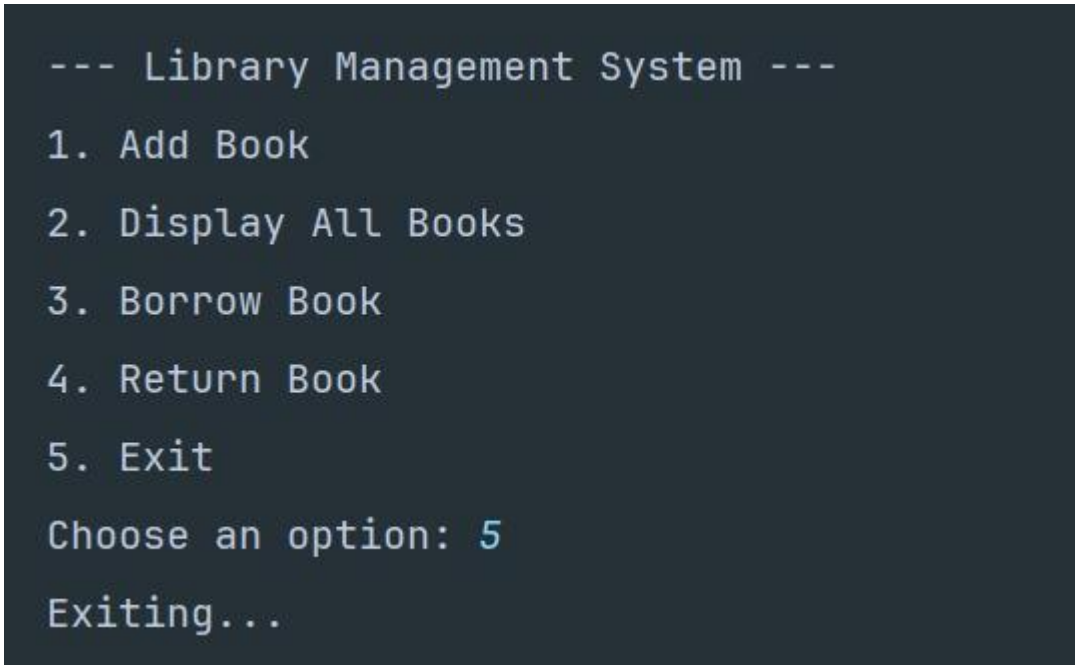
Borrowing The Book Which is Not Available in Library:

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 3
Enter book title to borrow: Raja Yoga
Book not found.
```

Returning The Book Which Is Not Available in Library:

```
--- Library Management System ---
1. Add Book
2. Display All Books
3. Borrow Book
4. Return Book
5. Exit
Choose an option: 4
Enter book title to return: Raja Yoga
Book not found.
```

Exiting From Project:



Project Performance Analysis

The Library Management System project was designed to be a straightforward Java application that performs basic operations for managing books within a library. This performance analysis evaluates the project's efficiency, scalability, and potential bottlenecks, as well as the strengths and limitations of the system.

Performance Metrics:

- **Execution Efficiency**: The project operates efficiently under the assumption that the number of books managed is relatively small (e.g., up to 10 books, as initialized). The operations of adding, searching, borrowing, and returning books are executed using linear search algorithms. This approach is suitable for a limited dataset, as each operation completes within O(n)O(n)O(n) time complexity, where n is the number of books.
- **Memory Usage:** The system uses an array to store books, which is pre-allocated with a fixed capacity. This choice simplifies the code structure but limits scalability. Memory is used efficiently for the given scope since only essential book details (title, author, availability) are stored.

- **User Interaction:** The program provides a simple menu-driven interface that facilitates user interaction through a command-line interface (CLI). This design ensures that the application is easy to use and does not require advanced technical skills from users.

Strengths:

- **Simplicity:** The project's simple design ensures fast execution and minimal resource usage, making it well-suited for small-scale library management tasks.
- **Modular Code Structure**: The separation of responsibilities into the Book, Library, and MAIN classes follows the principles of object-oriented programming (OOP). This modularity makes the code easier to maintain and extend with additional features.
- **Error Handling:** The system incorporates basic error handling, such as checking whether a book is already borrowed or trying to return a book that has not been borrowed. This improves the reliability of the application and prevents unexpected behavior.

Limitations:

- **Scalability:** As the project uses an array-based structure with a fixed capacity for storing books, it is not scalable to larger libraries. To support a larger number of books, the system would need to use a more flexible data structure, such as an ArrayList or a database.
- **Search Efficiency**: The linear search method used for finding books by title becomes less efficient as the number of books increases. For larger libraries, this approach would need to be replaced with more efficient search algorithms or data structures, such as hash maps or binary search trees.
- **Concurrency:** The current implementation is not designed for concurrent access, which would be necessary for larger, multi-user libraries. Ensuring thread safety and synchronization in a multi-threaded environment would be crucial for future versions of the project**.**

Discussion

The Discussion section of a report typically provides an analysis of the project's strengths, limitations, and potential for future enhancements. It gives a chance to reflect on the work done, evaluate its success, and outline areas for improvement. In the case of the Library Management System, the discussion should cover several key points: advantages, limitations, and opportunities for future development.

1. Advantages:

The Library Management System offers several key advantages that make it an efficient tool for managing a small library's operations. These advantages can be summarized as follows:

a. Simplification of Library Operations:

- The system provides a straightforward way to manage books in the library, including adding new books, searching for books, borrowing, and returning books. The process, which would otherwise require manual tracking, is automated, reducing the chances of human error and increasing efficiency.
- Automated Updates: The system automatically updates the status of books as borrowed or returned, which eliminates the need for librarians to manually check or update records.

b. Ease of Use:

- The user interface is simple and text-based, making it easy for users (even those with minimal technical expertise) to understand and interact with the system.
- Librarians or administrators can quickly add books, search for books by title, and manage the borrowing and returning process, without needing to worry about complicated functionality**.**

c. Manageable Data Set:

- The system handles a limited number of books (in this case, up to 10 books in the library). This is a manageable scale that simplifies development and testing. For smaller libraries or personal use, the program is effective and serves its intended purpose.

d. Low Hardware and Software Requirements:

- The project doesn't require high-end hardware or sophisticated software. It can run on most basic systems with a Java runtime environment (JRE) and requires minimal system resources (low RAM and CPU usage).

e. Open-Source and Customizable:

- The system can be freely modified and extended to suit more complex needs. For example, a user could add additional features, such as late fee calculations, tracking overdue books, or implementing a database for storing book records.

---

2. Limitations:

Despite its advantages, the Library Management System also has several limitations that can impact its usefulness, especially when scaled or used in a more complex environment.

a. Limited Scalability:

- The system is designed to handle only a small number of books (10 in this case), which limits its applicability in larger libraries. As the number of books increases, the system's current design (storing books in an array) would become inefficient and prone to errors. For larger systems, it would be better to use a database (e.g., MySQL, PostgreSQL) to store book records, providing greater scalability and performance.

b. Lack of User Management:

- The system does not include any functionality for user management. It assumes that anyone can borrow and return books without tracking who is borrowing the books. In a more realistic library environment, user management (i.e., registering users, tracking which user borrowed which book, etc.) would be essential.
- No support for user authentication (e.g., login/logout functionality). Anyone with access to the system can perform any action, which could lead to issues like unauthorized users borrowing books.

c. No Tracking of Overdue Books or Fines:

- The system does not keep track of overdue books, nor does it include a mechanism for calculating fines for overdue items. While this may be acceptable in a small or simple library, larger libraries require this functionality to maintain order and encourage timely returns.

d. Lack of Advanced Search and Filtering Features:

- Currently, the system can only search for books by title. It does not allow users to search by author, genre, or other criteria, which can be important in a library with a large and diverse collection. Adding search filters could improve the user experience and make it easier to find books in larger libraries.

e. No Persistence of Data:

- The current version of the system does not save data between sessions. This means that when the application is restarted, all added books are lost, and the status of borrowed or returned books is reset. Implementing data persistence using a database or file storage system would ensure that the data is preserved even after the program is closed or restarted.

f. No Graphical User Interface (GUI):

- The system uses a command-line interface (CLI), which can be less intuitive for some users, especially those who are not familiar with text-based systems. A GUI would make the application more user-friendly, providing users with interactive controls like buttons and menus for easier navigation**.**

3. Future Enhancements:

There are many opportunities for improving and extending the functionality of the Library Management System to address its limitations and meet the needs of larger libraries or more advanced use cases. Some of these potential enhancements include:

a. Transition to a Database:

- Implementing a relational database (e.g., MySQL, SQLite, or PostgreSQL) would allow the system to handle a larger collection of books and provide more sophisticated querying and filtering capabilities. A database would also support better scalability and data persistence, ensuring that the library's data is not lost when the program is restarted.

b. User Management System:

- Adding user accounts would allow librarians to track which books are borrowed by which users, manage user records, and enforce borrowing limits (e.g., a user can only borrow a certain number of books at a time). User authentication and access control could also be added to restrict certain actions (e.g., only librarians can add or remove books).

c. Overdue Book and Fine Management:

- A critical enhancement for a real-world library system would be to track overdue books and calculate fines for late returns. A simple algorithm could be added to calculate the number of days a book is overdue and multiply by a fixed fine rate. Alerts could be sent to users when their books are overdue.

d. Advanced Search and Filtering:

- The system could be upgraded to allow users to search for books by multiple criteria, such as author, genre, year of publication, or keywords. This would improve the system's usability, especially in larger libraries with diverse collections.

e. Graphical User Interface (GUI):

- A more user-friendly GUI could be developed using JavaFX or Swing, which would allow users to interact with the system visually rather than through the command line. Features like drag-and-drop book adding, search filters, and a more attractive layout would make the application more accessible to a broader audience.

f. Reporting and Statistics:

- The system could include features for generating reports on library usage, such as the number of books borrowed, the most popular books, and overdue book reports. These insights would be valuable for library management and decision-making.

Conclusion

The Library Management System serves as a simple and effective tool for managing a small library's inventory. By automating tasks like adding, borrowing, and returning books, it significantly reduces manual work and human error, making it a valuable asset for libraries with limited resources. The system is user-friendly, requiring minimal technical knowledge, and can be easily run on basic computer systems. Its low hardware and software requirements make it accessible to a wide audience, while the basic functionalities provided are sufficient for small-scale library management.

However, the system has several limitations, such as its inability to handle larger libraries, lack of user management, and absence of overdue tracking or fines. While the current design is appropriate for a small library, future enhancements—such as database integration, a user authentication system, and more advanced search options—could make the system more scalable and feature-rich. These improvements would help the system cater to a broader range of users and fulfill the requirements of larger, more complex library environments.

References

1.Oracle. (2024). *Java Documentation*. Retrieved from https://docs.oracle.com/javase/

2.Java™ Platform, Standard Edition 8 API Specification.

3. Java How to Program. Author=> Paul Deitle and HarveyDeitle

Appendices

Code of Project:

```
import java.util.Scanner;
class Book{
    private String Name;
    private String Author;
    private boolean Available;
    public Book(String Name,String Author){
        this.Name=Name;
        this.Author=Author;
        this.Available=true;
    }

     public String getName() {
         return Name;
     }

     public String getAuthor() {
         return Author;
     }

     public boolean isAvailable() {
         return Available;
     }
     public void borrowBook() {
         if (Available) {
             Available = false;  // Book is borrowed
         } else {
             System.out.println("The book is already borrowed.");
         }
     }
     public void ReturnBook(){
       if(!Available){
```

```
            Available=true;
        }
        else {
            System.out.println("Book Never Borrow");
        }
    }
    @Override
    public String toString(){
      return  "Title:"+ Name +"\nAuthor:"+Author+"\nAvailable:"+(Available?"Yes":"No");
    }
}
class Library{
    private Book[] books;
    private int NumOfBook;
    public Library(int Capacity){
        books=new Book[Capacity];
        NumOfBook=0;
    }
    public void AddBook(String Title,String Author)
    {
        if(NumOfBook<books.length){
            books[NumOfBook]=new Book(Title,Author);
            NumOfBook++;
            System.out.println("Book Successfully Added !");
        }
        else {
            System.out.println("Library is Full !");
        }
    }
    public void displayBooks() {
        if (NumOfBook == 0) {
            System.out.println("No books available in the library.");
            return;
        }
```

```
    for (int i = 0; i < NumOfBook; i++) {
        System.out.println(books[i].toString());
    }
}
public Book SearchBook(String Title){
    for(int i=0;i<=NumOfBook;i++){
        if(books[i].getName().equalsIgnoreCase(Title)){
            return books[i];
        }
    }
    return null;
}
public void BorrowBook(String Title){
    Book book=SearchBook(Title);
    if(book!=null&& book.isAvailable()){
        book.borrowBook();
        System.out.println("You Borrow A book Named "+book.getName());
    }
    else if(book!=null){
        System.out.println("The book is already borrowed.");
    }
    else {
        System.out.println("Book not found.");
    }
}
public void ReturnBook(String title) {
        Book book = SearchBook(title);
        if (book != null && !book.isAvailable()) {
            book.ReturnBook();
            System.out.println("You have returned: " + book.getName());
        } else if (book != null) {
            System.out.println("This book wasn't borrowed.");
        } else {
            System.out.println("Book not found.");
```

```
        }
    }
}
public class MAIN {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Library library = new Library(10);  // Initialize the library with a capacity of 10 books

        while (true) {
            System.out.println("\n--- Library Management System ---");
            System.out.println("1. Add Book");
            System.out.println("2. Display All Books");
            System.out.println("3. Borrow Book");
            System.out.println("4. Return Book");
            System.out.println("5. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine();  // Consume newline character

            switch (choice) {
                case 1:
                    // Add a new book
                    System.out.print("Enter book title: ");
                    String title = scanner.nextLine();
                    System.out.print("Enter book Author: ");
                    String author = scanner.nextLine();
                    library.AddBook(title, author);
                    break;

                case 2:
                    // Display all books
                    library.displayBooks();
                    break;
```

```
            case 3:
                // Borrow a book
                System.out.print("Enter book title to borrow: ");
                title = scanner.nextLine();
                library.BorrowBook(title);
                break;

            case 4:
                // Return a book
                System.out.print("Enter book title to return: ");
                title = scanner.nextLine();
                library.ReturnBook(title);
                break;

            case 5:
                // Exit the program
                System.out.println("Exiting...");
                scanner.close();
                return;

            default:
                System.out.println("Invalid option. Please try again.");
        }
    }
}
}
```