LIBRARY MANAGEMENT SYSTEM

Design Document

1. **Introduction**
   This document provides an overview of the design for the Library Management System (LMS). The system is built to manage books, patrons, and lending processes within a library. It includes functionality for adding, searching, and managing books, as well as tracking patrons and their borrowing history.

2. **Project Overview**
   The Library Management System allows for the management of books and patrons. It provides functionalities such as:

   - Adding and removing books.
   - Searching for books by title, author, or ISBN.
   - Tracking patrons and their borrowed books.
   - Checking out and returning books.

   The system is designed to be scalable and extensible with potential features like multi-branch support, reservation systems, and recommendations.

3. **Core Features**
   3.1 **Book Management**

   - Attributes: title, author, ISBN, publicationYear
   - Operations: Add, remove, update, and search books by title, author, or ISBN.

   3.2 **Patron Management**

   - Attributes: name, patronId, borrowedBooks
   - Operations: Add new patrons, update patron information, and track borrowing history.

   3.3 **Lending Process**

   - Operations: Check out and return books, updating their availability status.

   3.4 **Inventory Management**

   - Track available and borrowed books.

4. **Thought Process Behind the Design**
   4.1 **Singleton Pattern**

- **Why?** We don't want multiple libraries floating around (one library is already chaotic enough). Singleton ensures there's only one Library instance. This ensures all books, patrons, and their activities are consistently managed by a single central object.
- **How?** A private constructor and a static getInstance() method ensure that only one instance of the Library class exists. Every part of the system accesses the same instance, keeping everything centralized and synchronized.
- **Class**: Library
- **Purpose**: Ensures that only one instance of the library exists and is accessible globally.

4.2 **Factory Pattern**

- **Why?** To avoid the headache of manually creating books every time (books are complex creatures, after all). The BookFactory simplifies creation like a friendly assistant.
- The BookFactory class uses the Factory pattern to create books. It abstracts the instantiation of books, providing flexibility to create different types of books based on user input or requirements.
- **How?** The createBook() method takes in the required parameters and churns out Book objects effortlessly, ensuring consistency and flexibility in book creation.
- **Class**: BookFactory
- **Purpose**: Centralizes and simplifies the creation of Book objects.

4.3 **Strategy Pattern**

- **Why?** The SearchStrategy interface and its implementations (SearchByTitle, SearchByAuthor, SearchByISBN) implement the Strategy pattern. This allows the system to easily swap logics between different search strategies.
- **How?** Defined a SearchStrategy interface with SearchByTitle, SearchByAuthor, and SearchByISBN as flexible little minions.
- **Class**: SearchStrategy and its implementations
- **Purpose**: Provides different ways to search for books dynamically based on the selected strategy.

5. **OOP Concepts in Play**

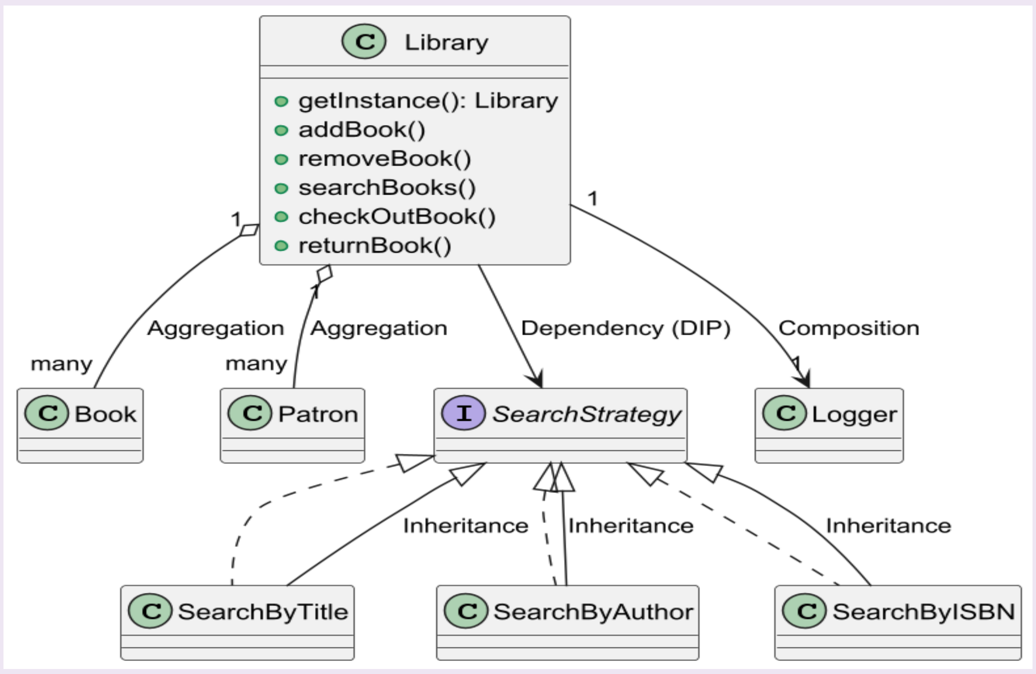
- **Inheritance**: Used for SearchByTitle, SearchByAuthor, and SearchByISBN classes that implement the SearchStrategy interface.

    - **Encapsulation**: Attributes of Book, Patron, and Library are hidden from outside access and accessed via getter/setter methods.
    - **Polymorphism**: Different search strategies (by title, author, ISBN) implement the same search() method from the SearchStrategy interface.
    - **Abstraction**: Library, Book, and Patron abstract away the complexities of the system to provide a simple interface for the user.

6. **Design Principles That Made the Cut**
    - **Single Responsibility Principle (SRP)**: Each class minds its own business—Logger logs, Book handles book stuff, and Library is the big boss.
    - **Open/Closed Principle (OCP)**: You can add new search strategies or book types without touching existing code.
    - **Liskov Substitution Principle**: Classes implementing SearchStrategy can be substituted without altering the behavior of the Library class.
    - **Dependency Inversion Principle (DIP)**: Library depends on SearchStrategy, not specific implementations. It's like hiring freelancers instead of full-time employees.
    - **DRY (Don't Repeat Yourself)**: The Logger ensures we log consistently everywhere instead of duplicating print statements like a broken record.
    - **Separation of Concerns**: No multitasking here—each class does one job, and it does it well.
7. **The Struggles Were Real**
   As a beginner backend developer, some of the challenges included:
    - Understanding and applying design patterns like Singleton and Factory.
    - Learning to structure code using OOP principles.
    - Ensuring that book status is properly managed (e.g., a book being checked out is unavailable).
    - Managing references between books and patrons to avoid unnecessary complexity.
8. **Future Enhancements**
    - **Multi-Branch Support**: The system can be extended to manage books and patrons across multiple branches.
    - **Reservation System**: Allow patrons to reserve books that are currently checked out.
    - **Recommendation System**: Suggest books to patrons based on their borrowing history and preferences.

9. **System Architecture**

9.1 **Class Diagram**



9.2 **Data Flow Diagram**
The following data flow describes the main operations:

1. A patron requests a book search by title.

2. The Library calls the appropriate search strategy (e.g., SearchByTitle).

3. If the book is found, the patron can check it out.

4. The Library updates the book's status and the patron's borrowed books.

9.3 **Logging**
The Logger class utilizes Java's java.util.logging framework to log important events (e.g., checkout, return operations, errors). This helps track system activity and troubleshoot issues.

11. **Technical Details**
   11.1 **Technologies Used**

- **Programming Language**: Java
- **Libraries/Frameworks**:
   - java.util.logging for logging.
- **Data Structures**:
   - List for managing books and patrons.

11.2 **Code Organization**

The code is divided into the following packages:

- **Book Management**: Contains the Book, BookFactory, and BookStatus classes.
- **Patron Management**: Contains the Patron class.
- **Library Management**: Contains the Library and SearchStrategy classes.
- **Logging**: Contains the Logger class for logging events.

11.3 **Error Handling**

Errors are logged using the Logger class and exceptions are thrown for invalid operations such as attempting to checkout an unavailable book or remove a non-existent book.

12. **Resources Used**

- ***Factory Design Pattern***
- ***Singleton Design Pattern – by Christopher Okhravi***
- ***Strategy Design Pattern – by Christopher Okhravi***
- ***Java Logging API***
- ***Learn SOLID Principles with CLEAN CODE Examples***
- ***BlackBox AI***

13. **Conclusion**

This document provides an overview of the design decisions made during the development of the Library Management System. It follows modern OOP principles and design patterns to ensure flexibility, scalability, and maintainability.