**Library Management System**

This project is a Java-based console application that simulates a basic library system using core OOP principles such as **classes, inheritance, abstraction, exception handling**, and **multithreading**. It allows users to **add, view, and search books**, as well as **manage user records**. The system handles invalid actions using **custom exceptions** and includes a **background search task** implemented with threads, ensuring modularity and real-time interaction.

**Key Features:**

* Add new books to the system
* View the complete list of available books
* Search books by title (case-insensitive)
* Add new users with unique user IDs
* View all registered users
* Handle invalid book searches using custom exceptions
* Background search operation using multithreading
* Modular architecture with service interface and implementation

**Technologies Used:**

* **Language:** Java
* **Concepts Applied:**
  + Object-Oriented Programming (OOP)
  + Exception Handling (Custom Exceptions)
  + Inheritance and Abstraction
  + Interfaces and Implementation
  + Multithreading
* **Tools Used:**
  + IDE (e.g., IntelliJ IDEA / Eclipse)
  + Java Development Kit (JDK)

**Flowchart:**

1. **Start**
2. **Display main menu:**
   * Add Book
   * View Books
   * Search Book
   * Add User
   * View Users
   * Exit
3. **User selects an option**
4. **Switch-case executes logic:**
   * If Add Book: Prompt for book details → Create Book object → Call addBook()
   * If View Books: Call viewBooks()
   * If Search Book: Prompt for title → Call searchBook() (throws exception if not found)
   * If Add User: Prompt for user details → Create User object → Call addUser()
   * If View Users: Call viewUsers()
5. If invalid input → throw InvalidOptionException
6. Ask if user wants to perform another operation
7. If yes → go back to Step 2, else → End

**Learning Outcomes:**

* Understood how to implement core OOP principles in real-world simulations
* Learned how to create and handle **custom exceptions** to manage error flow
* Implemented **interfaces** and followed the separation of logic and data
* Explored **multithreading** to run tasks concurrently (e.g., book search)
* Practiced writing modular, clean, and maintainable Java code
* Gained hands-on experience in designing an application with real-time interaction features