# Mini Project Report: Library Inventory Manager

**Course:** Programming for Problem Solving using Python

**Assignment Title:** Object-Oriented Design and Robust Programming in a Library Management System

**Submission Date:** November 30, 2025

**Submitted By:** Sagar Pawar

## 1. Introduction

### 1.1 Real-World Problem Context

Campus libraries manage vast collections of books and require efficient systems to track inventory. Manual tracking is prone to errors, particularly regarding the status of books (whether they are available or issued). This project aims to solve this problem by developing a lightweight, command-line-based application that allows staff to maintain records in a persistent format, search catalog entries, and manage book status dynamically.

### 1.2 Learning Objectives

The primary objectives of this project were to:

* Implement **Object-Oriented Programming (OOP)** principles such as encapsulation and modular design.
* Develop a **robust storage system** using JSON for data persistence.
* Handle runtime errors gracefully using **Exception Handling** (try-except blocks).
* Create a user-friendly **Command Line Interface (CLI)**.
* Structure the code into a professional Python **package hierarchy**.

## 2. System Design and Architecture

### 2.1 Project Structure

The project follows a modular directory structure to separate business logic from the user interface, ensuring maintainability and scalability.

library-inventory-manager/  
├── library\_manager/ # Core Logic Package  
│ ├── \_\_init\_\_.py # Package initialization  
│ ├── book.py # Task 1: Book class definition  
│ └── inventory.py # Task 2 & 3: Inventory management and JSON handling  
├── cli/ # User Interface Package  
│ ├── \_\_init\_\_.py  
│ └── main.py # Task 4: CLI entry point  
├── data/ # Data Storage  
│ └── catalog.json # Persistent database  
├── tests/ # Unit Tests (Bonus Task)  
│ └── test\_book.py # Automated tests for Book class  
├── library.log # Application logs (Task 5)  
└── requirements.txt # Dependencies

### 2.2 Class Design

The system is built around two primary classes:

1. **Book Class (book.py)**:
   * Represents an individual book entity.
   * **Attributes**: title, author, isbn, status.
   * **Methods**: issue(), return\_book(), to\_dict() (for serialization), and \_\_str\_\_ (for string representation).
2. **LibraryInventory Class (inventory.py)**:
   * Manages the collection of Book objects.
   * **Responsibilities**: Adding books, searching (by Title/ISBN), and handling File I/O.
   * **Persistence**: Automatically loads data from catalog.json on startup and saves changes after every transaction.

## 3. Implementation Details

### 3.1 Data Persistence (JSON Handling)

To ensure data is not lost when the program closes, the json module was used. The LibraryInventory class serializes Book objects into dictionaries and writes them to data/catalog.json.

* **Loading Data:** The system checks if the file exists using pathlib. If missing or corrupted, it initializes an empty list to prevent crashes.
* **Saving Data:** Updates are committed to the file immediately after a book is added, issued, or returned.

### 3.2 Exception Handling and Logging

Robustness was achieved through:

* **Input Validation:** Ensuring fields like Title and ISBN are not empty.
* **File I/O Safety:** Wrapping file operations in try-except blocks to catch FileNotFoundError or JSONDecodeError.
* **Logging:** The logging module records significant events (e.g., "Book issued: Python 101") and runtime errors to library.log for debugging and audit trails.

### 3.3 Command Line Interface (CLI)

The main.py script provides an interactive menu loop:

1. **Add Book**: Captures metadata and updates the inventory.
2. **Issue/Return**: Toggles the status of a book based on ISBN.
3. **Search**: Allows partial string matching for Titles and exact matching for ISBNs.
4. **View All**: Displays the formatted list of all books.

## 4. Testing and Results

### 4.1 Functional Testing

The following scenarios were tested successfully:

| **Test Case** | **Input** | **Expected Outcome** | **Result** |
| --- | --- | --- | --- |
| **Add Book** | Title: "AI Basics", ISBN: "101" | Book added to memory and JSON. | **Pass** |
| **Issue Book** | ISBN: "101" | Status changes to 'issued'. | **Pass** |
| **Re-issue Book** | ISBN: "101" (already issued) | Error message: "Already issued". | **Pass** |
| **Search** | Query: "AI" | Returns "AI Basics". | **Pass** |
| **Persistence** | Restart Application | "AI Basics" is still present. | **Pass** |

### 4.2 Automated Unit Tests (Bonus)

A unit test file tests/test\_book.py was created using the unittest framework to verify the Book class logic automatically.

**Output of Unit Tests:**

Ran 2 tests in 0.001s  
OK

## 5. Challenges and Solutions

* **Module Imports:** Initially, the CLI script could not find the library\_manager package.
  + *Solution:* Modified sys.path in main.py to dynamically include the project root directory.
* **JSON Serialization:** The json.dump method cannot natively save custom objects.
  + *Solution:* Implemented a to\_dict() method in the Book class to convert objects into standard Python dictionaries before saving.

## 6. Conclusion

The "Library Inventory Manager" successfully meets all the requirements of the assignment. It demonstrates a working application of OOP principles, file persistence, and Python package management. The system is robust, handling errors gracefully without crashing, and provides a clear audit trail via logging. This project solidified the understanding of how to structure larger Python applications for real-world scenarios.