CAPSTONE PROJECT: E-LIBRARY SYSTEM

Abstract
The e-Library Management System is a Python-based application designed to digitally manage books, users, and borrowing activities in a library. The system allows administrators to add and remove books, register users, issue and return books, and calculate fines for overdue returns. It follows an object-oriented design with separate classes for books, users, and the main library system, ensuring modularity and reusability.

Why this project ?
We chose this project because it is a real-world scenario that applies data management, persistence, and OOP principles. It handles users, books, and transactions practically and can be extended into a larger system (GUI, database).

Introduction
The e-Library Management System is a Python-based application designed to manage library resources digitally. It allows administrators to add/remove books, register users, issue and return books, and calculate fines for overdue returns.
This project is built using Object-Oriented Programming (OOP) principles and stores data persistently in JSON format.

Modules
- json → Store and load library data persistently.
- os → Check if the storage file exists.
- logging → Record important actions (book added, fine issued, etc.).
- datetime, timedelta → Manage issue dates, due dates, and calculate fines.
- typing → For type hints (Dict, List).

Constants

- STORE → JSON file to store data (library.json).
- LOGFILE → Log file for activities (library.log).
- LOAN_DAYS → Default book loan period (14 days).
- FINE_PER_DAY → Penalty for overdue returns (2 units per day).

Classes and Attributes

Class: A class in Python is like a blueprint for creating objects. It describes what properties (attributes) and what behaviors (methods) those objects will have.
Attributes: Attributes are variables that belong to a specific object. They store data related to that object.

CAPSTONE PROJECT: E-LIBRARY SYSTEM

Class 1: Book

```
class Book:
    def __init__(self, book_id: str, title: str, author: str, isbn: str, available: bool = True):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.isbn = isbn
        self.available = available
```

The Book class is a blueprint for creating book objects in the library system.
It describes what information every book must have.

Attributes

1. self.book_id
   Unique ID for the book (like "B1" or "B101").
   Helps identify a book uniquely in the system (like a roll number for a student).
2. self.title
   The name of the book (e.g., "Harry Potter and the Sorcerer's Stone").
   Helps users know which book they're borrowing.
3. self.author
   The author's name (e.g., "J.K. Rowling").
   Useful for searching/filtering books by author.
4. self.isbn
   ISBN = International Standard Book Number (e.g., "978-3-16-148410-0").
   Acts as a universal unique code for a book edition.
5. self.available
   Boolean (True or False).
   True = book is available to borrow.
   False = book is already issued to someone.

Class 2: User

```
class User:
    def __init__(self, user_id: str, name: str):
        self.user_id = user_id
        self.name = name
```

The User class represents a member of the library. It stores their identity so the system knows who borrowed which book.

CAPSTONE PROJECT: E-LIBRARY SYSTEM

Attributes

1. self.user_id
   Unique ID for the user (like "U1", "U202").
   Prevents confusion if two users have the same name.
2. self.name
   The actual name of the user (e.g., "Sahil").
   Helps in identifying and displaying user details.

Class 3: LibrarySystem

```
class LibrarySystem:
   def __init__(self):
      self.books: Dict[str, Book] = {}
      self.users: Dict[str, User] = {}
      self.transactions: List[dict] = []
      self._load()
```

The LibrarySystem class is the main controller of the program.
It manages all books, all users, and all borrow/return transactions.

Attributes

1. self.books

A dictionary (Dict[str, Book]).
Stores all book objects.
Key = book ID (e.g., "B1"), Value = Book object.

Example:

```
{
      "B1": Book("B1", "Harry Potter", "J.K. Rowling", "12345"),
      "B2": Book("B2", "Sherlock Holmes", "Arthur Conan Doyle", "54321")
     }
```

2. self.users

- A dictionary (Dict[str, User]).
- Stores all user objects.
- Key = user ID (e.g., "U1"), Value = User object.
- Example:

```
{
  "U1": User("U1", "Sahil"),
  "U2": User("U2", "Priya")
}
```

3. self.transactions

- A list of dictionaries.
- Each dictionary represents a record of issuing or returning a book.
- Example transaction:

```
{
  "book_id": "B1",
  "user_id": "U1",
  "issue_date": "2025-08-19T10:30:00",
  "due_date": "2025-09-02T10:30:00",
  "return_date": None
}
```

- If return_date is None → book is still with user.
- If return_date has a date → book was returned.

4. self._load()

- Not an attribute, but called in the constructor.
- Loads existing data from library.json so that previously saved books, users, and transactions are restored when the program starts.

5. JSON File Handling

```
with open(STORE, "r", encoding="utf-8") as f:
  data = json.load(f)
self.books = {b["book_id"]: Book.from_dict(b) for b in data.get("books", [])}
self.users = {u["user_id"]: User.from_dict(u) for u in data.get("users", [])}
self.transactions = data.get("transactions", [])
```

- Opens the file → Reads JSON → Rebuilds Book and User objects → Restores transactions.
- with open(STORE, "r", encoding="utf-8") as f: → Opens library.json file in read mode.

CAPSTONE PROJECT: E-LIBRARY SYSTEM

- data = json.load(f) → Reads the JSON file and converts it into a Python dictionary.
- self.books = {...} → Rebuilds all saved books as Book objects from JSON.
- self.users = {...} → Rebuilds all saved users as User objects from JSON.
- self.transactions = ... → Loads the list of transactions

Saving Data:

```
with open(STORE, "w", encoding="utf-8") as f:
  json.dump(data, f, indent=2)
```

- Converts Python objects into JSON and saves them with indentation.
- with open(STORE, "w", encoding="utf-8") as f: → Opens library.json in write mode (creates/overwrites the file).
- json.dump(data, f, indent=2) → Writes the Python data (books, users, transactions) into the file in JSON format with neat indentation.

6. Exception Handling

An exception is a signal that something unexpected happened while running a program.

- raise ValueError("Book ID exists") → Raised if book already exists.
- raise LookupError("Book not found") → Raised when searching for missing book/user.
- raise RuntimeError("Book already issued") → Raised when trying to issue an already borrowed book.

7. Fine Calculation

- First, the system checks if the book exists in the library. If the book ID is invalid, it immediately raises an error saying “Book not found”.
- Next, it looks through the transactions list to find an active loan for that book. An active loan means: the book ID matches, the user ID matches, and the book hasn’t been returned yet (return_date is None). If no such loan is found, it raises an error “Active transaction not found”.
- Once the transaction is found, the system records the current date and time as the return date, marks the book as available again, and saves the updated data into the JSON file.
- Then, it reads the due date from the transaction and compares it with today’s date. If the book is late, the system calculates how many days late it is. If the book is returned on time or earlier, late days become 0.
- The fine is then calculated as late days × FINE_PER_DAY (2 units per day).
- If a fine exists, the system logs it in the log file (for record-keeping) and returns the fine amount. If there is no fine, the method simply ends without returning anything.