**Library Management System**

**Explanation**

1. **Project Description**

Develop a console-based application to simulate a simplified library management system. The system will allow librarians (users of the application) to:

* Add and remove books from the library's collection.
* Register and deregister users.
* Facilitate borrowing and returning of books by users.
* Search for books by various criteria (title, author, ISBN).
* Display lists of all available books, all registered users, and books currently borrowed by a specific user.
* Persist library data (books, users, borrowed status) to a file so it's saved between application runs.

This project strongly emphasizes Object-Oriented Programming (OOP) principles, including encapsulation, class design, and object interaction.

1. **Core OOP Concepts to Apply**

* **Class & Object Design:** Model real-world entities (Book, User, Library) as Python classes.
* **Encapsulation:** Use private-like attributes (e.g., \_title, \_is\_borrowed) and expose them via public methods or @property decorators where appropriate, ensuring data integrity.
* **Constructors (\_\_init\_\_):** Properly initialize object states.
* **Instance Methods:** Define behaviors specific to an object instance.
* **Class Methods / Static Methods (Optional but good):** For utilities not tied to a specific instance (e.g., Library.generate\_unique\_id()).
* **Object Relationships:** Library will *contain* Book and User objects. User objects will

*contain* references to Book objects they have borrowed.

* **Magic Methods (\_\_str\_\_, \_\_repr\_\_):** Implement these for clear object representation when printing or debugging.
* **Error Handling:** Use try-except blocks for user input validation and file operations.

1. **Implementation Detail**

You are required to implement the following classes with the specified attributes and methods.

* 1. **Book Class**

Represents a single book in the library.

* + - **Attributes (Private-like for encapsulation):**
      * \_title: str - The title of the book.
      * \_author: str - The author of the book.
      * \_isbn: str - The unique International Standard Book Number.
      * \_is\_borrowed: bool - True if borrowed, False otherwise (default False).
    - **Properties:**
      * title (read-only): Returns \_title.
      * author (read-only): Returns \_author.
      * isbn (read-only): Returns \_isbn.
      * is\_borrowed (read/write): Returns \_is\_borrowed. The setter should only allow True or False.
    - **Methods:**
      * init (self, title: str, author: str, isbn: str):
        + Initializes a new Book object with the given title, author, and ISBN. Sets

\_is\_borrowed to False.

* + - * borrow(self) -> bool:
        + If the book is not already borrowed, set \_is\_borrowed to True and return True.
        + Otherwise, return False (book already borrowed).
      * return\_book(self) -> bool:
        + If the book is borrowed, set \_is\_borrowed to False and return True.
        + Otherwise, return False (book not currently borrowed).
      * str (self) -> str:
        + Returns a user-friendly string representation of the book, e.g., "Title: [Title], Author: [Author], ISBN: [ISBN], Status: [Available/Borrowed]".
      * to\_dict(self) -> dict:
        + Returns a dictionary representation of the book's attributes, useful for saving to file.
  1. **User Class**

Represents a registered library user.

* + - **Attributes (Private-like for encapsulation):**
      * \_name: str - The user's name.
      * \_user\_id: str - A unique identifier for the user.
      * \_borrowed\_books\_isbns: list[str] - A list of ISBNs of books currently borrowed by this user. (Store ISBNs for easy serialization/deserialization, the Library class will manage Book objects).
    - **Properties:**
      * name (read-only): Returns \_name.
      * user\_id (read-only): Returns \_user\_id.
      * borrowed\_books\_isbns (read-only): Returns a copy of

\_borrowed\_books\_isbns to prevent direct modification from outside.

* + - **Methods:**
      * init (self, name: str, user\_id: str):
        + Initializes a new User object. \_borrowed\_books\_isbns starts as an empty list.
      * add\_borrowed\_book\_isbn(self, isbn: str) -> None:
        + Adds an ISBN to \_borrowed\_books\_isbns if not already present.
      * remove\_borrowed\_book\_isbn(self, isbn: str) -> None:
        + Removes an ISBN from \_borrowed\_books\_isbns if present.
      * str (self) -> str:
        + Returns a user-friendly string representation of the user, e.g., "User: [Name] (ID: [User ID]), Borrowed Books: [Number of books]".
      * to\_dict(self) -> dict:
        + Returns a dictionary representation of the user's attributes, useful for saving to file.
  1. **Library Class**

Manages the collections of Book and User objects, and handles core library operations.

* + - **Attributes (Private-like for encapsulation):**
      * \_books: dict[str, Book] - A dictionary where keys are ISBNs and values are Book objects.
      * \_users: dict[str, User] - A dictionary where keys are user IDs and values are User objects.
      * \_data\_file\_books: str - Filename for saving/loading book data (e.g.,

'books.json').

* + - * \_data\_file\_users: str - Filename for saving/loading user data (e.g.,

'users.json').

* + - **Methods:**
      * init (self, book\_file: str = 'books.json', user\_file: str

= 'users.json'):

* + - * + Initializes empty dictionaries for \_books and \_users.
        + Sets data file names.
        + Calls \_load\_data() to load existing data from files.
      * \_load\_data(self) -> None: (Private Helper Method)
        + Loads book and user data from \_data\_file\_books and

\_data\_file\_users.

* + - * + Handles FileNotFoundError.
        + **Important:** When loading books, reconstruct Book objects from dictionary data. When loading users, reconstruct User objects.
      * \_save\_data(self) -> None: (Private Helper Method)
        + Saves current \_books and \_users data to respective JSON files.
        + Iterate through \_books and \_users to call to\_dict() on each object before saving.
      * add\_book(self, book: Book) -> bool:
        + Adds a Book object to \_books. Returns True if added successfully,

False if ISBN already exists. Calls \_save\_data().

* + - * remove\_book(self, isbn: str) -> bool:
        + Removes a book by ISBN from \_books. Returns True if removed,

False if not found or if the book is currently borrowed. Calls

\_save\_data().

* + - * register\_user(self, user: User) -> bool:
        + Registers a User object to \_users. Returns True if added, False if user ID already exists. Calls \_save\_data().
      * remove\_user(self, user\_id: str) -> bool:
        + Removes a user by ID from \_users. Returns True if removed, False if not found or if the user has borrowed books. Calls \_save\_data().
      * borrow\_book(self, isbn: str, user\_id: str) -> bool:
        + Checks if book and user exist and if book is available.
        + If so, calls book.borrow() and user.add\_borrowed\_book\_isbn().
        + Returns True on success, False otherwise (e.g., book not found, user not found, book already borrowed). Calls \_save\_data().
      * return\_book(self, isbn: str, user\_id: str) -> bool:
        + Checks if book and user exist and if user actually borrowed this book.
        + If so, calls book.return\_book() and

user.remove\_borrowed\_book\_isbn().

* + - * + Returns True on success, False otherwise. Calls \_save\_data().
      * search\_book(self, query: str) -> list[Book]:
        + Searches for books by title, author, or isbn (partial or full match).
        + Returns a list of matching Book objects. Case-insensitive for title/author search.
      * display\_all\_books(self, show\_available\_only: bool = False)

-> None:

* + - * + Prints details of all books. Option to show only available books.
      * display\_all\_users(self) -> None:
        + Prints details of all registered users.
      * display\_user\_borrowed\_books(self, user\_id: str) -> None:
        + Prints details of books borrowed by a specific user. Handles user not found.

1. **Coder Tasks (Implementation Steps)**
2. **Set up Project Structure:** Create library\_management.py (or similar).
3. **Implement Book Class:** Write the Book class with all specified attributes, properties, and methods. Test its basic functionality.
4. **Implement User Class:** Write the User class with all specified attributes, properties, and methods. Test its basic functionality.
5. **Implement Library Class (Core Logic):**
   * Implement init , \_load\_data, \_save\_data. Start with dummy data or empty files.
   * Implement add\_book, remove\_book, register\_user, remove\_user.
   * Implement borrow\_book, return\_book. This is where Book and User objects interact.
   * Implement search\_book, display\_all\_books, display\_all\_users, display\_user\_borrowed\_books.
6. **Create Console Interface:**
   * Develop a main() function or a run() method in the Library class.
   * Implement a main while loop that presents a menu to the user (e.g., "1. Add Book", "2. Register User", "3. Borrow Book", "4. Search Book", "5. Exit").
   * Use input() to get user choices and data.
   * Call the appropriate Library methods based on user input.
   * Include clear print() statements for feedback and displaying results.
7. **Implement Robust Error Handling:**
   * Handle ValueError for invalid numeric inputs (if applicable, e.g., for future quantity inputs).
   * Handle cases where books/users are not found, or operations are invalid (e.g., trying to borrow an already borrowed book).
8. **JSON File Persistence:** Ensure to\_dict() and object reconstruction from dictionaries work correctly for saving/loading. Use the json module.
9. **User Type Inheritance:**
   * Create an AbstractUser class (from abc module) if you want to enforce specific methods.
   * Implement Student and Faculty classes inheriting from User. Each might have unique attributes (e.g., student\_id, department) or borrowing limits.
   * Modify Library methods to handle these different user types polymorphically.
10. **Late Fees Implementation:**
    * Add a due\_date attribute to Book when borrowed.
    * Implement a method in Library to calculate and display late fees for overdue books.
11. **Basic Reporting:**
    * Add methods to Library to generate reports like "All currently borrowed books," "Overdue books," or "Books borrowed by a specific user."

Overall Functionality

The code provides a basic structure for a library management system where users can register, borrow, and return books. The system maintains the state of each book (whether it is borrowed or available) and tracks which books each user has borrowed.

Code

class Book:

    def \_\_init\_\_(self, title, author, isbn):

        self.title = title

        self.author = author

        self.isbn = isbn

        self.is\_borrowed = False

    def borrow(self):

        if not self.is\_borrowed:

            self.is\_borrowed = True

            return True

        return False

    def return\_book(self):

        if self.is\_borrowed:

            self.is\_borrowed = False

            return True

        return False

    def \_\_str\_\_(self):

        status = "Borrowed" if self.is\_borrowed else "Available"

        return f"{self.title} by {self.author} | ISBN: {self.isbn} | {status}"

class User:

    def \_\_init\_\_(self, name, user\_id):

        self.name = name

        self.user\_id = user\_id

        self.borrowed = []

    def borrow\_book(self, isbn):

        if isbn not in self.borrowed:

            self.borrowed.append(isbn)

    def return\_book(self, isbn):

        if isbn in self.borrowed:

            self.borrowed.remove(isbn)

    def \_\_str\_\_(self):

        return f"{self.name} (ID: {self.user\_id}) | Borrowed: {len(self.borrowed)} books"

class Library:

    def \_\_init\_\_(self):

        self.books = {}

        self.users = {}

    def add\_book(self, title, author, isbn):

        if isbn in self.books:

            return False

        self.books[isbn] = Book(title, author, isbn)

        return True

    def register\_user(self, name, user\_id):

        if user\_id in self.users:

            return False

        self.users[user\_id] = User(name, user\_id)

        return True

    def borrow\_book(self, isbn, user\_id):

        if isbn in self.books and user\_id in self.users:

            book = self.books[isbn]

            user = self.users[user\_id]

            if book.borrow():

                user.borrow\_book(isbn)

                return True

        return False

    def return\_book(self, isbn, user\_id):

        if isbn in self.books and user\_id in self.users:

            book = self.books[isbn]

            user = self.users[user\_id]

            if isbn in user.borrowed and book.return\_book():

                user.return\_book(isbn)

                return True

        return False

    def search\_books(self, query):

        result = []

        for book in self.books.values():

            if query.lower() in book.title.lower() or query.lower() in book.author.lower() or query in book.isbn:

                result.append(book)

        return result

    def show\_books(self):

        for book in self.books.values():

            print(book)

    def show\_users(self):

        for user in self.users.values():

            print(user)

    def user\_books(self, user\_id):

        if user\_id in self.users:

            user = self.users[user\_id]

            for isbn in user.borrowed:

                print(self.books[isbn])

        else:

            print("User not found")

def menu():

    lib = Library()

    while True:

        print("\n1.Add Book 2.Register User 3.Borrow 4.Return 5.Search 6.Show Books 7.Show Users 8.User's Books 0.Exit")

        ch = input("Choose: ")

        if ch == '1':

            t = input("Title: ")

            a = input("Author: ")

            i = input("ISBN: ")

            print("Added." if lib.add\_book(t, a, i) else "Book exists.")

        elif ch == '2':

            n = input("Name: ")

            uid = input("User ID: ")

            print("Registered." if lib.register\_user(n, uid) else "User exists.")

        elif ch == '3':

            i = input("ISBN: ")

            uid = input("User ID: ")

            print("Borrowed." if lib.borrow\_book(i, uid) else "Failed.")

        elif ch == '4':

            i = input("ISBN: ")

            uid = input("User ID: ")

            print("Returned." if lib.return\_book(i, uid) else "Failed.")

        elif ch == '5':

            q = input("Search: ")

            for b in lib.search\_books(q):

                print(b)

        elif ch == '6':

            lib.show\_books()

        elif ch == '7':

            lib.show\_users()

        elif ch == '8':

            uid = input("User ID: ")

            lib.user\_books(uid)

        elif ch == '0':

            break

        else:

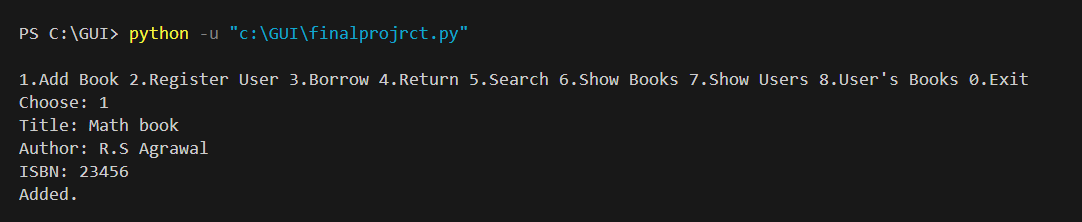
            print("Invalid choice.")

if \_\_name\_\_ == '\_\_main\_\_':

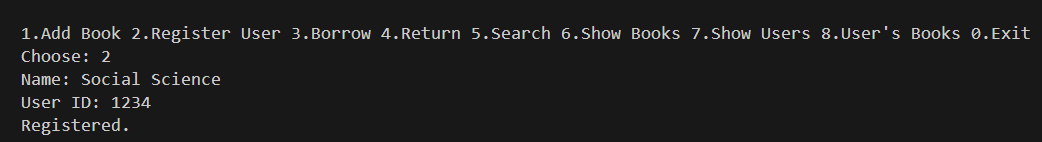
    menu()

Code Output

1.Output 1: Add Book



2. Output 2: Register Book



3. Output 3: Borrowed Book

A black screen with white text

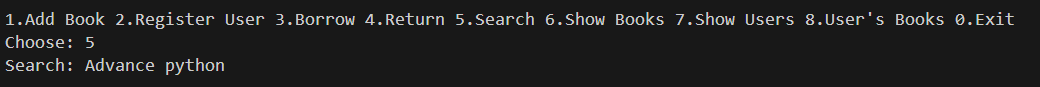
AI-generated content may be incorrect.

4 . Output 4: Return Book

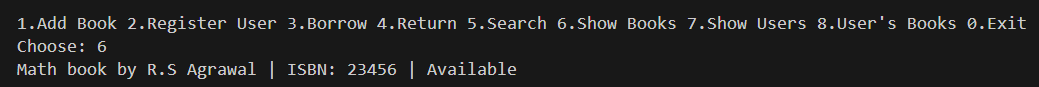
A black screen with white text

AI-generated content may be incorrect.

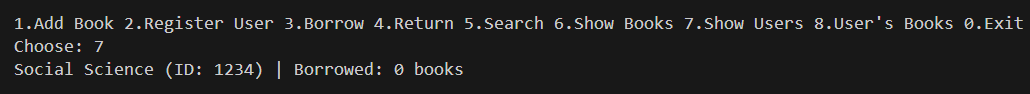
5 . Output 5: Search Book



6 . Output 6: Show Book



7 . Output 7: Show Users



8 . Output 8: User’s Books

