Library Management System Using Python

Objective

The objective of this project is to develop a Library Management System (LMS) using Python to automate and streamline library operations. This system provides functionalities for managing books and student records, issuing and returning books, and maintaining an efficient library workflow. By leveraging file-based storage with Python's pickle module, this system offers a user-friendly interface for both library staff and administrators to perform essential library management tasks.

Summary

The Library Management System (LMS) is designed to manage book and student records effectively and to facilitate smooth library operations. The system supports the following key features:

- **Student Management:** Create, display, modify, and delete student records.
- **Book Management:** Create, display, modify, and delete book records.
- **Book Issue and Return:** Issue books to students and manage book returns.
- **Administrative Menu:** An admin panel to perform all management tasks.

The application uses file-based storage (pickle serialization) to store data in files (book.dat, student.dat, and issued_books.dat). It provides an intuitive interface for users to interact with the library's operations.

Implementation Details

The Library Management System is implemented using Python. The code structure and logic for the system are as follows:

1. File Initialization

The initialize_files function checks if the data files (book.dat, student.dat, and issued_books.dat) exist. If not, it creates these files with empty lists to store book and student information.

2. Student Management

a. Create Student

The create_student function allows the user to add a new student record by entering details like Roll Number, Name, and Branch. The record is then stored in student.dat.

b. Display All Students

The display_all_students function retrieves and displays all student records stored in student.dat.

c. Display Specific Student

The display_specific_student function allows the user to input a Roll Number and display the specific student's record.

d. Modify Student

The modify_student function allows modification of an existing student record based on the Roll Number.

e. Delete Student

The delete_student function removes a student record from student.dat based on the Roll Number.

3. Book Management

a. Create Book

The create_book function allows the user to add a new book record by entering details like Book ID, Title, and Author. The record is stored in book.dat.

b. Display All Books

The display_all_books function retrieves and displays all book records stored in book.dat.

c. Display Specific Book

The display_specific_book function allows the user to input a Book ID and display the specific book's record.

d. Modify Book

The modify_book function allows modification of an existing book record based on the Book ID.

e. Delete Book

The delete_book function removes a book record from book.dat based on the Book ID.

4. Book Issue and Return

a. Issue Book

The issue_book function allows issuing a book to a student by updating both book and issued book records.

b. Return Book

The return_book function manages book returns by updating both book and issued book records.

c. Display Issued Books

The display_issued_books function retrieves and displays all issued book records from issued_books.dat.

5. Administrative Menu

The admin_menu function provides an interface for managing all library operations.

6. Entry Point

The main function serves as the entry point for the application.

Results

The Library Management System is successfully developed and tested. It allows efficient management of books and student records, facilitating the following tasks:

- **Student Management:** Creation, modification, display, and deletion of student records.
- **Book Management:** Creation, modification, display, and deletion of book records.
- **Book Issue and Return:** Issuing and returning books while maintaining records.
- **File-Based Storage:** Secure storage of data in book.dat, student.dat, and issued_books.dat using pickle.

Conclusion

The Library Management System using Python provides a robust solution for managing library operations. By automating book and student management, it reduces manual effort and improves efficiency in library administration. The use of file-based storage ensures data persistence and security.

The project demonstrates how Python can be utilized to create real-world applications with practical functionality. This system can be extended further to incorporate additional features, such as:

- **Database Integration:** Transition from file-based storage to a database like SQLite or MySQL for enhanced data management.
- **User Authentication:** Implement user authentication for different user roles (e.g., admin, student).
- **Graphical User Interface (GUI):** Create a GUI for a more user-friendly interface.
- **Enhanced Reporting:** Provide detailed reports and analytics on library activities.

Code :

'''

Certainly! Below is a Python program for a Library Management System (LMS) that includes the functionalities you described.

This program uses files (book.dat and student.dat) to store book and student information respectively using the pickle module

for serialization. It also includes an administrative menu for managing student and book records.

Let's walk through the implementation:

'''

```
import pickle
import os
from datetime import date

# Define global filenames for storing data
BOOK_FILE = 'book.dat'
STUDENT_FILE = 'student.dat'
ISSUED_BOOK_FILE = 'issued_books.dat'

# Function to create initial files if they don't exist
def initialize_files():
    if not os.path.exists(BOOK_FILE):
        with open(BOOK_FILE, 'wb') as file:
            pickle.dump([], file)
    if not os.path.exists(STUDENT_FILE):
        with open(STUDENT_FILE, 'wb') as file:
            pickle.dump([], file)
    if not os.path.exists(ISSUED_BOOK_FILE):
        with open(ISSUED_BOOK_FILE, 'wb') as file:
            pickle.dump([], file)

# Function to create student record
```

```
def create_student():
    print("\nEnter student details:")
    roll_no = input("Roll No: ")
    name = input("Name: ")
    branch = input("Branch: ")

    student = {'roll_no': roll_no, 'name': name, 'branch': branch}

    with open(STUDENT_FILE, 'rb') as file:
        students = pickle.load(file)

    students.append(student)

    with open(STUDENT_FILE, 'wb') as file:
        pickle.dump(students, file)

    print(f"\nStudent Record for Roll No {roll_no} created successfully.")

# Function to display all students
def display_all_students():
    with open(STUDENT_FILE, 'rb') as file:
        students = pickle.load(file)

    print("\nAll Students:")
    for student in students:
        print(f"Roll No: {student['roll_no']}, Name: {student['name']}, Branch: {student['branch']}")

    print(f"Total Students: {len(students)}")

# Function to display specific student record
def display_specific_student():
```

```
    roll_no = input("\nEnter Roll No of the student to display: ")

    with open(STUDENT_FILE, 'rb') as file:
        students = pickle.load(file)

    found = False
    for student in students:
        if student['roll_no'] == roll_no:
            print("\nStudent Details:")
            print(f"Roll No: {student['roll_no']}, Name: {student['name']}, Branch: {student['branch']}")
            found = True
            break

    if not found:
        print(f"\nStudent with Roll No {roll_no} not found.")

# Function to modify student record
def modify_student():
    roll_no = input("\nEnter Roll No of the student to modify: ")

    with open(STUDENT_FILE, 'rb') as file:
        students = pickle.load(file)

    found = False
    for student in students:
        if student['roll_no'] == roll_no:
            print("\nCurrent Student Details:")
            print(f"Roll No: {student['roll_no']}, Name: {student['name']}, Branch: {student['branch']}")
            print("\nEnter new details:")
            name = input("Name (leave blank to keep current): ").strip()
            branch = input("Branch (leave blank to keep current): ").strip()
```

```
            if name:
                student['name'] = name
            if branch:
                student['branch'] = branch

            found = True
            break

    if found:
        with open(STUDENT_FILE, 'wb') as file:
            pickle.dump(students, file)
        print(f"\nStudent Record for Roll No {roll_no} modified successfully.")
    else:
        print(f"\nStudent with Roll No {roll_no} not found.")

# Function to delete student record
def delete_student():
    roll_no = input("\nEnter Roll No of the student to delete: ")

    with open(STUDENT_FILE, 'rb') as file:
        students = pickle.load(file)

    new_students = [student for student in students if student['roll_no'] != roll_no]

    if len(new_students) < len(students):
        with open(STUDENT_FILE, 'wb') as file:
            pickle.dump(new_students, file)
        print(f"\nStudent Record for Roll No {roll_no} deleted successfully.")
    else:
        print(f"\nStudent with Roll No {roll_no} not found.")
```

```
# Function to create book
def create_book():
    print("\nEnter book details:")
    book_id = input("Book ID: ")
    title = input("Title: ")
    author = input("Author: ")

    book = {'book_id': book_id, 'title': title, 'author': author, 'available': True}

    with open(BOOK_FILE, 'rb') as file:
        books = pickle.load(file)

    books.append(book)

    with open(BOOK_FILE, 'wb') as file:
        pickle.dump(books, file)

    print(f"\nBook Record for Book ID {book_id} created successfully.")

# Function to display all books
def display_all_books():
    with open(BOOK_FILE, 'rb') as file:
        books = pickle.load(file)

    print("\nAll Books:")
    for book in books:
        status = "Available" if book['available'] else "Issued"
        print(f"Book ID: {book['book_id']}, Title: {book['title']}, Author: {book['author']}, Status: {status}")

    print(f"Total Books: {len(books)}")
```

```
# Function to display specific book record
def display_specific_book():
    book_id = input("\nEnter Book ID of the book to display: ")

    with open(BOOK_FILE, 'rb') as file:
        books = pickle.load(file)

    found = False
    for book in books:
        if book['book_id'] == book_id:
            status = "Available" if book['available'] else "Issued"
            print("\nBook Details:")
            print(f"Book ID: {book['book_id']}, Title: {book['title']}, Author: {book['author']}, Status: {status}")
            found = True
            break

    if not found:
        print(f"\nBook with Book ID {book_id} not found.")

# Function to modify book record
def modify_book():
    book_id = input("\nEnter Book ID of the book to modify: ")

    with open(BOOK_FILE, 'rb') as file:
        books = pickle.load(file)

    found = False
    for book in books:
        if book['book_id'] == book_id:
```

```
            print("\nCurrent Book Details:")
            print(f"Book ID: {book['book_id']}, Title: {book['title']}, Author: {book['author']}")
            print("\nEnter new details:")
            title = input("Title (leave blank to keep current): ").strip()
            author = input("Author (leave blank to keep current): ").strip()

            if title:
                book['title'] = title
            if author:
                book['author'] = author

            found = True
            break

    if found:
        with open(BOOK_FILE, 'wb') as file:
            pickle.dump(books, file)
        print(f"\nBook Record for Book ID {book_id} modified successfully.")
    else:
        print(f"\nBook with Book ID {book_id} not found.")

# Function to delete book record
def delete_book():
    book_id = input("\nEnter Book ID of the book to delete: ")

    with open(BOOK_FILE, 'rb') as file:
        books = pickle.load(file)

    new_books = [book for book in books if book['book_id'] != book_id]

    if len(new_books) < len(books):
```

```
        with open(BOOK_FILE, 'wb') as file:
            pickle.dump(new_books, file)
        print(f"\nBook Record for Book ID {book_id} deleted successfully.")
    else:
        print(f"\nBook with Book ID {book_id} not found.")

# Function to issue a book
def issue_book():
    student_roll_no = input("\nEnter Student Roll No: ")
    book_id = input("Enter Book ID: ")

    with open(STUDENT_FILE, 'rb') as file:
        students = pickle.load(file)

    with open(BOOK_FILE, 'rb') as file:
        books = pickle.load(file)

    with open(ISSUED_BOOK_FILE, 'rb') as file:
        issued_books = pickle.load(file)

    student_exists = any(student['roll_no'] == student_roll_no for student in students)
    book_exists = any(book['book_id'] == book_id for book in books)
    book_available = any(book['book_id'] == book_id and book['available'] for book in books)

    if not student_exists:
        print(f"\nStudent with Roll No {student_roll_no} not found.")
        return

    if not book_exists:
        print(f"\nBook with Book ID {book_id} not found.")
        return
```

```
    if not book_available:
        print(f"\nBook with Book ID {book_id} is already issued.")
        return

    issued_book = {'student_roll_no': student_roll_no, 'book_id': book_id, 'issue_date': date.today(), 'return_date': None}
    issued_books.append(issued_book)

    for book in books:
        if book['book_id'] == book_id:
            book['available'] = False
            break

    with open(ISSUED_BOOK_FILE, 'wb') as file:
        pickle.dump(issued_books, file)

    with open(BOOK_FILE, 'wb') as file:
        pickle.dump(books, file)

    print(f"\nBook ID {book_id} issued to Student Roll No {student_roll_no} successfully.")

# Function to deposit a book
def deposit_book():
    student_roll_no = input("\nEnter Student Roll No: ")
    book_id = input("Enter Book ID: ")

    with open(ISSUED_BOOK_FILE, 'rb') as file:
        issued_books = pickle.load(file)

    issued_book = next((book for book in issued_books if book['student_roll_no'] == student_roll_no and book['book_id'] == book_id and book['return_date'] is None), None)
```

```
    if not issued_book:
        print(f"\nNo record found for Book ID {book_id} issued to Student Roll No {student_roll_no}.")
        return

    issued_book['return_date'] = date.today()

    with open(ISSUED_BOOK_FILE, 'wb') as file:
        pickle.dump(issued_books, file)

    with open(BOOK_FILE, 'rb') as file:
        books = pickle.load(file)

    for book in books:
        if book['book_id'] == book_id:
            book['available'] = True
            break

    with open(BOOK_FILE, 'wb') as file:
        pickle.dump(books, file)

    print(f"\nBook ID {book_id} returned by Student Roll No {student_roll_no} successfully.")

# Main function to display menu and process user input
def main():
    initialize_files()

    while True:
        print("\n***** LIBRARY MANAGEMENT SYSTEM *****")
        print("1. BOOK ISSUE")
        print("2. BOOK DEPOSIT")
```

```
print("3. ADMINISTRATION MENU")
print("4. EXIT")

choice = input("\nEnter your choice: ")

if choice == '1':
    issue_book()
elif choice == '2':
    deposit_book()
elif choice == '3':
    print("\n***** ADMINISTRATION MENU *****")
    print("a. CREATE STUDENT RECORD")
    print("b. DISPLAY ALL STUDENTS RECORD")
    print("c. DISPLAY SPECIFIC STUDENT RECORD")
    print("d. MODIFY STUDENT RECORD")
    print("e. DELETE STUDENT RECORD")
    print("f. CREATE BOOK")
    print("g. DISPLAY ALL BOOKS")
    print("h. DISPLAY SPECIFIC BOOK")
    print("i. MODIFY BOOK RECORD")
    print("j. DELETE BOOK RECORD")
    print("k. BACK TO MAIN MENU")

    admin_choice = input("\nEnter your choice: ")

    if admin_choice == 'a':
        create_student()
    elif admin_choice == 'b':
        display_all_students()
    elif admin_choice == 'c':
        display_specific_student()
```

```
            elif admin_choice == 'd':
                modify_student()
            elif admin_choice == 'e':
                delete_student()
            elif admin_choice == 'f':
                create_book()
            elif admin_choice == 'g':
                display_all_books()
            elif admin_choice == 'h':
                display_specific_book()
            elif admin_choice == 'i':
                modify_book()
            elif admin_choice == 'j':
                delete_book()
            elif admin_choice == 'k':
                continue
            else:
                print("\nInvalid choice! Please enter a valid option (a-k).")
        elif choice == '4':
            print("\nThank you for using the Library Management System. Goodbye!")
            break
        else:
            print("\nInvalid choice! Please enter a valid option (1-4).")

# Entry point of the program
if __name__ == "__main__":
    main()



'''
1).File Initialization (initialize_files function):
```

This function ensures that the book.dat and student.dat files exist. If they do not exist, it creates them with an empty list using pickle.

2) Student Management Functions:

a) create_student:

Allows the user to create a new student record and stores it in student.dat.

b) display_all_students:

Displays all student records stored in student.dat.

c) display_specific_student:

Prompts the user for a roll number and displays the specific student's record.

d) modify_student:

Enables modification of an existing student record based on roll number.

e) delete_student:

Deletes a student record based on roll number.

2) Book Management Functions:

a) create_book:

Allows the user to create a new book record and stores it in book.dat.

b) display_all_books:

Displays all book records stored in book.dat.

c) display_specific_book:

Prompts the user for a book ID and displays the specific book's record.

d) modify_book:

Enables modification of an existing book record based on book ID.

e) delete_book:

Deletes a book record based on book ID.

3) Main Menu (main function):

Displays the main menu with options for book issue, book deposit, administration menu, and exit.

Based on user input, directs to respective functionalities or sub-menu of administrative tasks.

4) Error Handling:

Basic error handling is included to manage invalid user inputs and to inform the user appropriately.

This program provides a foundational structure for a Library Management System using

file-based storage (pickle serialization) for persistent data storage.

Adjustments can be made as per specific requirements, such as adding more features

(e.g., book issue and deposit functionalities using database) or enhancing error handling and validation.

'''