PES Institute of Technology and Management, Shivamogga

(Affiliated to Visvesvaraya Technological University "Jnana Sangama", Belgaum)

Department of Artificial Intelligence and Machine Learning

Subject: ARTIFICIAL INTELLIGENCE
[BAD402]

Academic Year: 2023-24

Semester: IV

Section: 'A'

Batch: 2022-2026

Mini project Title: Library Management System

Submitted By,

Name : Akash Shreekant Hegde

USN : 4PM22AI002

Marks : __/10

Real World Problem Identified:

This problem statement outlines a practical scenario where the provided code can be applied to streamline library operations, ensuring books are managed effectively and users have a clear understanding of book availability and return dates.

It helps the Librarian to ensure that the books are correctly returned within the deadline.It effectively displays the books which are available for borrow and effectively updates the system.

Algorithm:

- **Initialization**

  - Create a class `Library` with attributes:
  - `books`: Dictionary to store available books and their authors.
  - `borrowed_books`: Dictionary to track borrowed books along with borrower information.

- **Methods**:

`__init__` method:

- Initialize `self.books` with a predefined set of books and authors.
- Initialize `self.borrowed_books` as an empty dictionary.

`borrow_book` method:

- Parameters: `book_name` (string), `borrower_name` (string).
- Check if `book_name` exists in `self.books`.

If yes:

- Calculate a return date using `get_return_date()`.
- Store `(borrower_name, return_date)` in `self.borrowed_books` for the `book_name`.
- Remove `book_name` from `self.books`.

- Print a confirmation message with return date.
- Return `return_date`

If no:

- Print a message indicating the book is not currently borrowed.

`get_return_date` method:

- Returns the current date plus a fixed timedelta of 14 days, representing the return date for borrowed books.

`display_books` method:

- Print the list of books available in `self.books` along with their authors.

`display_borrowed_books` method:

- Print the list of books currently borrowed from `self.borrowed_books` along with borrower names and return dates.

Main Function (`main`):

- Create an instance of `Library`.
- Use a loop to repeatedly display a menu of options and handle user input:
- available books.
- Borrow a book (prompt for book name and borrower name).
- Return a book (prompt for book name).
- Display borrowed books.
- Exit the program.

Solution/ Program Implementation:

```
from datetime import datetime, timedelta

class Library:
    def _init_(self):
        self.books = {
            "Python Programming": "John Smith",
            "Introduction to Machine Learning": "Jane Doe",
            "Data Structures and Algorithms": "Alice Johnson"
        }
        self.borrowed_books = {}

    def borrow_book(self, book_name, borrower_name):
        if book_name in self.books:
            return_date = self.get_return_date()
            self.borrowed_books[book_name] = (borrower_name, return_date)
            del self.books[book_name]
```

```
            print(f"Book '{book_name}' borrowed by {borrower_name}. Return by:
{return_date}.")
            return return_date
        else:
            print(f"Book '{book_name}' not available in the library.")
            return None

    def return_book(self, book_name):
        if book_name in self.borrowed_books:
            self.books[book_name] = self.borrowed_books[book_name][0]
            del self.borrowed_books[book_name]
            print(f"Book '{book_name}' returned successfully.")
        else:
            print(f"Book '{book_name}' is not currently borrowed.")

    def get_return_date(self):
        # Assuming a fixed return date of 14 days from the borrowing date
        return datetime.now() + timedelta(days=14)

    def display_books(self):
        print("Books available in the library:")
        for book, author in self.books.items():
            print(f"{book} by {author}")

    def display_borrowed_books(self):
        print("Books currently borrowed:")
        for book, (borrower, return_date) in self.borrowed_books.items():
            print(f"{book} borrowed by {borrower}. Return by:
{return_date.strftime('%Y-%m-%d')}")

def main():
    library = Library()
    while True:
        print("\n1. Display available books\n2. Borrow a book\n3. Return a book\n4.
Display borrowed books\n5. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            library.display_books()
        elif choice == '2':
            book_name = input("Enter the name of the book you want to borrow: ")
            borrower_name = input("Enter your name: ")
            return_date = library.borrow_book(book_name, borrower_name)
            if return_date:
                print(f"Return by: {return_date.strftime('%Y-%m-%d')}")
        elif choice == '3':
            book_name = input("Enter the name of the book you want to return: ")
            library.return_book(book_name)
```

```
        elif choice == '4':
            library.display_borrowed_books()
        elif choice == '5':
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")


if _name_ == "_main_":
    main()
```

Conclusion:

- In conclusion, while this library management system code serves as a solid foundation for managing books in a small library setting, it represents a starting point rather than a comprehensive solution.
- Its clear structure, effective use of Python's datetime module, and user-friendly interface make it suitable for educational purposes or small-scale applications where managing a modest collection of books is required.
- Further development could expand its capabilities to meet specific operational needs of larger or more diverse libraries.
- The current implementation of the library management system provides a solid foundation for basic book management, further development can enhance its functionality, scalability, and usability.
- These enhancements would make it suitable for larger and more diverse libraries, meeting specific operational needs while maintaining user-friendly interactions and administrative efficiency.
- By integrating modern technologies and addressing additional features, the system can evolve into a robust solution capable of supporting libraries in today's digital age.