**Lab Project Report**

**OOP II: PYTHON**

**Semester: Spring 2024**

**Student Name: Anika Afrin Moumeta**

**Student ID: 221-15-5142**

**Section: 61-A2**

**Title of the Project: Bookstore Management System**

**Date of Submission: -05-2024**

**Project Title:** Bookstore Management System

**Introduction:**

For a physical book store to survive in this digital age, it needs an integrated system handling all its inventory management, sales process and customer satisfaction. The project designed here is a full Bookstore Management System that incorporates both admin side and also student side interfaces & functionalities.

This project aims to automate the work flow of a bookstore, so that even people with limited knowledge are able run this system which would be efficient in adding new books, removing old ones updating stock levels, enabling and connecting purchases done by customers and also tracking their orders. This system allows us to manage stocks in an orderly manner, provides a one-click consumer journey with Python together with Pandas library and effectively processes orders as well track them also. The book store management system is divided into a number of subsystems that provide features corresponding to the abstract base class for books, a book manager managing the collection of books, shopping cart for user purposes and admin or regular user panels in their respective sub-system areas providing an effective bookstore management solution.

**Motivation:**

This bookstore management system has been developed to help typical challenges faced by both Bookstore Administrator and customers. Conventional management methods can be manual, cumbersome and prone to errors – all of which are inefficient. The aim is to streamline, automate these steps digitally and have a seamless process built between store operator cataloging or inventory management perspective and customer purchasing end with minimal effort being put by either.

Key motivations include:

1. **Advanced Inventory Management**: Using automation for tracking stock and avoiding human errors, discrepancies or delays while updating inventory.
2. Improved User Experience: An easy to navigate catalog makes browsing, checking if it is in stock and potentially buying the desired products a more pleasant experience, increasing satisfaction and loyalty.
3. Streamlined Order Processing — Automate the checkout process, stock checks and order tracking; Ensuring fast and accurate processing, particularly in peak times.
4. Data Driven Decision Making: Make decisions about your inventory and marketing strategy with real time transactional data.
5. Scalability and Adaptiveness: Creating a scalable system that can expand with growing inventory and customer base without compromising service quality.

So, in short terms this project is about utilizing tech to solve the traditional issues of bookstore management which makes it easier and fun for Admins as well as customers. This will ultimately, according to the system, help to streamline operations and customer service needs while acting as a propellant for growth and bookstore success.

**Project Objectives:**

The Bookstore Management System project aims to provide a comprehensive solution for managing a bookstore's inventory and facilitating customer interactions. The objectives of the project are:

Objective 1: Efficient Book Management

* Develop a system for managing different book types: Create support for physical books and ebooks, each with specific attributes such as title, author, price, and stock.
* Flexible display of book details: Implement methods to display details of each book type, ensuring that both physical books and ebooks are represented appropriately.

Objective 2: User Interaction and Transaction Management

* User-friendly interface for administrators and customers: Design interfaces that allow administrators to add, remove, and display books in the catalog, and enable customers to browse, add to cart, and purchase books.
* Cart functionality: Enable customers to add books to their cart, view their cart contents, and remove items as needed.

Objective 3: Data Persistence and Management

* Data storage and retrieval: Utilize Pandas and Excel files to store book and order data persistently, ensuring that books and orders can be loaded and saved seamlessly.
* Order management: Develop features to handle orders, including tracking stock availability, calculating costs, and saving order details.

Objective 4: Enhanced System Flexibility and Maintenance

* Modular and maintainable codebase: Apply object-oriented programming principles to create a modular and maintainable codebase. Use abstract base classes and inheritance to manage different book types efficiently and reduce code duplication.
* Scalability and adaptability: Ensure that the system can scale as the bookstore grows, accommodating an increasing number of books and customers without compromising performance.

Objective 5: Administrative and User Panels

* Admin panel: Provide an administrative panel to manage the bookstore catalog, add new books, remove existing books, and save changes persistently.
* User panel: Create a user panel where customers can browse available books, add books to their cart, view their cart, and complete purchases.

Objective 6: Error Handling and User Feedback

* Error handling: Implement error-checking mechanisms to handle invalid user inputs and ensure robust system behavior.
* User feedback: Provide informative feedback to users, confirming successful actions such as adding books to the cart, removing items, and completing purchases.

**Project Description:**

The Bookstore Management System is designed to automate and simplify the operations of a bookstore, providing an integrated platform for both administrators and customers. It manages a comprehensive catalog of books, facilitates customer transactions, and ensures efficient inventory management.

**Key Features:**

* Book Management: Supports physical books and ebooks, displaying detailed information such as title, author, price, and stock.
* Cart System: Allows customers to add books to a cart, adjust quantities, and remove items before checkout.
* Order Processing: Manages orders, calculates costs, and updates stock levels in real-time.
* Admin Panel: Provides tools for managing the bookstore's inventory, adding new books, and updating existing ones.

**Workflow:**

Book Management:

* The system supports two types of books: physical books and ebooks.
* Administrators can add new books, update existing ones, and remove books from the catalog.
* Book details such as title, author, price, stock, and file size (for ebooks) are stored and displayed.

Customer Interaction:

* Customers can browse the catalog, search for specific books, and view details such as price and availability.
* A cart system allows customers to add books, adjust quantities, and remove items before checkout.
* Customers can complete purchases securely, with the system tracking stock levels and managing order details.

Administrative Tools:

* An admin panel provides tools for managing the bookstore's inventory.
* Admins can add, update, and remove books from the catalog, ensuring the database remains current and accurate.
* Book data is stored in Excel files, making it easy to manage and update.

**Benefits:**

Efficiency: Automates manual processes, reducing errors and improving accuracy in inventory management and order processing.

User-Friendly Interface: Provides an intuitive interface for customers and administrators, enhancing user experience.

Data Management: Uses Pandas and Excel files to store and retrieve book and order data, ensuring persistence and scalability.

**Project User:**

Bookstore Management System has only 2 users:

1. Administrators:

Role: Administer and manage the bookstore operations.

Tasks:

* Add new books to the catalog.
* Update existing book details (title, author, price, stock).
* Remove books from the catalog.
* View and manage orders.
* Access and use administrative tools to maintain the system.

2. Customers:

Role: Use the system to browse, select, and purchase books.

Tasks:

* Browse the catalog of books.
* Search for specific books by title or author.
* View detailed information about books (price, stock availability).
* Add books to a shopping cart.
* Adjust quantities or remove books from the cart.
* Complete purchases securely.
* View order history and details.

**Project Features:**

For Admin Panel:

* Add Book to Catalog
* Remove the Book from Catalog
* Display Catalog

For Admin Panel:

* Display Catalog
* Add Book to Cart
* Remove the Book from Cart
* View Cart
* Purchase Books

**Addressing OOP Concepts:**

I have created the Task Management System using Object-Oriented Programming (OOP) to following modularity, reusability, maintainable code. How The Project Follows of OOP Concepts

* **Encapsulation:**

Encapsulation is the bundling of data (attributes) and methods (functions that operate on the data) into a single unit (class), and restricting access to some of the object's components. This helps in preventing the accidental modification of data.

In the Book class, attributes like \_title, \_author, price, and stock are encapsulated within the class.They are accessed and modified through getter (get) and setter (set) methods, like get\_title(), get\_author(), get\_price(), set\_price(), get\_stock(), and set\_stock().This encapsulation ensures that the internal state of a Book object is protected and can only be modified through controlled methods, maintaining data integrity.

* **Inheritance:**

Inheritance is a mechanism in which one class acquires the properties (attributes and methods) and behaviors of another class. It supports the concept of hierarchical classification.

The BookItem class inherits from the Book class (class BookItem(Book)).By doing so, BookItem inherits attributes such as \_title, \_author, \_price, \_stock, as well as methods like get\_title(), get\_author(), get\_price(), set\_price(), get\_stock(), and set\_stock().BookItem specializes by adding its own display\_details() and to\_dict() methods, which are specific to the behavior of a physical book item.

* **Polymorphism:**

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It provides a way to perform a single action in different ways.

Abstract Base Class (Book): The Book class defines abstract methods display\_details() and to\_dict(), which are implemented differently in its subclasses (BookItem, EBook). This allows objects of different book types to be treated uniformly, as they can all be invoked using the same method names.

### **Abstraction:**

Abstraction involves hiding the complex implementation details of an object and only showing the necessary features of the object to the outside world. It focuses on what an object does rather than how it achieves it, facilitating simplicity and modularity.

Abstract Methods in Book: The Book class defines abstract methods display\_details() and to\_dict(), which are implemented differently in its subclasses. This abstraction allows users to interact with these methods without needing to know the internal implementation details of each book type.

**Conclusion:**

This particular project follows strong OOP principles with Python to create an Online Book Store System. It shows some very good example code using inheritance, polymorphism, encapsulation and abstraction as it was intended. It has elements like book management, user interaction by the admin and user panel and a shopping cart to buy books for users. It uses pandas for data manipulation, to make sure that information on the bookstore catalog and placed orders are stored permanently in Excel files. The project in a way encourages code reusability and modularity hence the maintenance of the app is easy ( The project is scalable as well). These could improve handling the errors, as it stands at now by printing an arbitrary message to console and exit immediately with a return code of 1; they also include making this as GUI and other specific feature additions similar to search functionality or authentication. In summary, the project demonstrates some ways that OOP can be used in reality.

**Coding:**

import pandas as pd

from abc import ABC, abstractmethod

# Abstract Base Class Book

class Book(ABC):

    def \_\_init\_\_(self, title, author, price, stock):

        self.\_title = title

        self.\_author = author

        self.\_price = price

        self.\_stock = stock

    @abstractmethod

    def display\_details(self):

        pass

    @abstractmethod

    def to\_dict(self):

        pass

    # Getter and Setter methods for common attributes

    def get\_title(self):

        return self.\_title

    def get\_author(self):

        return self.\_author

    def get\_price(self):

        return self.\_price

    def set\_price(self, price):

        self.\_price = price

    def get\_stock(self):

        return self.\_stock

    def set\_stock(self, stock):

        self.\_stock = stock

# Class BookItem

class BookItem(Book):

    def display\_details(self):

        print(f"Title: {self.\_title}")

        print(f"Author: {self.\_author}")

        print(f"Price: {self.\_price}")

        print(f"Stock: {self.\_stock}")

    def to\_dict(self):

        return {

            "title": self.\_title,

            "author": self.\_author,

            "price": self.\_price,

            "stock": self.\_stock

        }

# New Class EBook

class EBook(Book):

    def \_\_init\_\_(self, title, author, price, stock, file\_size):

        super().\_\_init\_\_(title, author, price, stock)

        self.\_file\_size = file\_size

    def display\_details(self):

        print(f"Title: {self.\_title}")

        print(f"Author: {self.\_author}")

        print(f"Price: {self.\_price}")

        print(f"Stock: {self.\_stock}")

        print(f"File Size: {self.\_file\_size} MB")

    def to\_dict(self):

        return {

            "title": self.\_title,

            "author": self.\_author,

            "price": self.\_price,

            "stock": self.\_stock,

            "file\_size": self.\_file\_size

        }

    # Getter and Setter methods for file\_size

    def get\_file\_size(self):

        return self.\_file\_size

    def set\_file\_size(self, file\_size):

        self.\_file\_size = file\_size

# BookManager Class

class BookManager:

    def \_\_init\_\_(self):

        self.books = []

        self.load\_books()

    def add\_book(self, book):

        self.books.append(book)

    def remove\_book(self, book):

        if book in self.books:

            self.books.remove(book)

            print("Book removed successfully.")

        else:

            print("Book not found.")

    def list\_books(self):

        if not self.books:

            print("No books available.")

        else:

            for book in self.books:

                book.display\_details()

                print()

    def save\_books(self):

        books\_dict\_list = [book.to\_dict() for book in self.books]

        df = pd.DataFrame(books\_dict\_list)

        df.to\_excel("/content/drive/MyDrive/Colab Notebooks/books.xlsx", index=False)

        print("Books saved to /content/drive/MyDrive/Colab Notebooks/books.xlsx")

    def load\_books(self):

        try:

            df = pd.read\_excel("/content/drive/MyDrive/Colab Notebooks/books.xlsx")

            for i, row in df.iterrows():

                if 'file\_size' in row:

                    book = EBook(row["title"], row["author"], row["price"], row["stock"], row["file\_size"])

                else:

                    book = BookItem(row["title"], row["author"], row["price"], row["stock"])

                self.books.append(book)

        except FileNotFoundError:

            print("/content/drive/MyDrive/Colab Notebooks/books.xlsx not found")

# Cart Class

class Cart:

    def \_\_init\_\_(self):

        self.items = []

    def add\_to\_cart(self, book, quantity):

        self.items.append((book, quantity))

        print(f"Added {quantity} of '{book.get\_title()}' to the cart.")

    def remove\_from\_cart(self, book):

        for item in self.items:

            if item[0] == book:

                self.items.remove(item)

                print(f"Removed '{book.get\_title()}' from the cart.")

                return

        print("Book not in the cart.")

    def view\_cart(self):

        if not self.items:

            print("Cart is empty.")

        else:

            print("Cart items:")

            for book, quantity in self.items:

                print(f"Title: {book.get\_title()}, Quantity: {quantity}")

                book.display\_details()

                print()

    def checkout(self):

        if not self.items:

            print("Cart is empty. Nothing to purchase.")

            return

        user\_info = self.\_get\_user\_info()

        print("Purchasing the following books:")

        total\_cost = 0

        order\_data = []

        for book, quantity in self.items:

            if book.get\_stock() >= quantity:

                print(f"- {book.get\_title()} by {book.get\_author()} (${book.get\_price()}) x {quantity}")

                total\_cost += book.get\_price() \* quantity

                book.set\_stock(book.get\_stock() - quantity)

                order\_data.append({

                    "title": book.get\_title(),

                    "author": book.get\_author(),

                    "price": book.get\_price(),

                    "quantity": quantity,

                    "user\_name": user\_info["name"],

                    "user\_email": user\_info["email"]

                })

            else:

                print(f"Insufficient stock for {book.get\_title()}. Only {book.get\_stock()} left.")

        print(f"Total cost: ${total\_cost}")

        print("Purchase successful!")

        print(f"Thank you, {user\_info['name']} ({user\_info['email']}).")

        self.\_save\_order\_info(order\_data)

        self.items.clear()

    def \_get\_user\_info(self):

        name = input("Enter your name: ")

        email = input("Enter your email: ")

        return {"name": name, "email": email}

    def \_save\_order\_info(self, order\_data):

        try:

            df\_existing = pd.read\_excel("/content/drive/MyDrive/Colab Notebooks/orders.xlsx")

            df = pd.DataFrame(order\_data)

            df\_combined = pd.concat([df\_existing, df], ignore\_index=True)

        except FileNotFoundError:

            df\_combined = pd.DataFrame(order\_data)

        df\_combined.to\_excel("/content/drive/MyDrive/Colab Notebooks/orders.xlsx", index=False)

        print("Order details saved to /content/drive/MyDrive/Colab Notebooks/orders.xlsx")

# Function to create a book

def create\_book():

    book\_type = input("Enter book type (1 for Physical Book, 2 for EBook): ")

    title = input("Enter book title: ")

    author = input("Enter book author: ")

    price = float(input("Enter book price: "))

    stock = int(input("Enter book stock: "))

    if book\_type == "1":

        return BookItem(title, author, price, stock)

    elif book\_type == "2":

        file\_size = float(input("Enter file size (MB): "))

        return EBook(title, author, price, stock, file\_size)

    else:

        print("Invalid book type.")

        return None

# Admin Panel

def admin\_panel(book\_manager):

    while True:

        print("\nAdmin Panel")

        print("1. Add Book to Catalog")

        print("2. Remove Book from Catalog")

        print("3. Display Catalog")

        print("4. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":

            book = create\_book()

            if book:

                book\_manager.add\_book(book)

                print("Book added successfully.")

        elif choice == "2":

            title = input("Enter book title to remove: ")

            book\_to\_remove = None

            for book in book\_manager.books:

                if book.get\_title() == title:

                    book\_to\_remove = book

                    break

            if book\_to\_remove:

                book\_manager.remove\_book(book\_to\_remove)

            else:

                print("Book not found.")

        elif choice == "3":

            print("\nAll books:")

            book\_manager.list\_books()

        elif choice == "4":

            book\_manager.save\_books()

            print("Exiting admin panel.")

            break

        else:

            print("Invalid choice. Please try again.")

# User Panel

def user\_panel(book\_manager):

    cart = Cart()

    while True:

        print("\nUser Panel")

        print("1. Display Catalog")

        print("2. Add Book to Cart")

        print("3. Remove Book from Cart")

        print("4. View Cart")

        print("5. Purchase Books")

        print("6. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":

            print("\nAvailable books:")

            book\_manager.list\_books()

        elif choice == "2":

            title = input("Enter book title to add to cart: ")

            quantity = int(input("Enter quantity: "))

            book\_to\_add = None

            for book in book\_manager.books:

                if book.get\_title() == title and book.get\_stock() >= quantity:

                    book\_to\_add = book

                    break

            if book\_to\_add:

                cart.add\_to\_cart(book\_to\_add, quantity)

            else:

                print("Book not found or insufficient stock.")

        elif choice == "3":

            title = input("Enter book title to remove from cart: ")

            book\_to\_remove = None

            for book, \_ in cart.items:

                if book.get\_title() == title:

                    book\_to\_remove = book

                    break

            if book\_to\_remove:

                cart.remove\_from\_cart(book\_to\_remove)

            else:

                print("Book not in the cart.")

        elif choice == "4":

            cart.view\_cart()

        elif choice == "5":

            cart.checkout()

        elif choice == "6":

            print("Exiting user panel.")

            break

        else:

            print("Invalid choice. Please try again.")

# Main Program Loop

def main():

    book\_manager = BookManager()

    while True:

        print("\n1. Admin Panel")

        print("2. User Panel")

        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":

            admin\_panel(book\_manager)

        elif choice == "2":

            user\_panel(book\_manager)

        elif choice == "3":

            book\_manager.save\_books()

            print("Exiting program.")

            break

        else:

            print("Invalid choice. Please try again.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()