


- ChatGPT
- Explore GPTs
- Today
- Library Management Facade ...
- Adapter Pattern Example
- Builder Pattern Library System
- Abstract Factory Pattern Example
- Check Python Init Method
- Enhanced Abstract Factory Pattern
- Movie Ticket Factory Pattern
- Design Patterns Refactoring
- Using He for Dog
- Isnull vs Coalesce SQL

facade pattern with python code, you can write code from below domain,

- Library Management System,
- Parking Lot,
- Online Shopping System - E-Commerce,
- Stack Overflow,
- Movie Ticket Booking System,
- Design an ATM,
- Airline Management System,
- Hotel Management System,
- Restaurant Management system,
- Design Chess,
- Design an Online Stock Brokerage System,
- Design a Car Rental System,
- Design LinkedIn,
- Design Cricinfo,
- Design Facebook Social Network,
- Design payment gateway

 Get citation

Let's use the **Facade Pattern** to design a **Library Management System**. The Facade Pattern is a structural pattern that provides a simplified interface to a complex subsystem. Here's how you might apply this pattern to a library management system:

1. Define Subsystems

In our library management system, we could have several subsystems:

- Book Management:** For handling books (add, remove, search).
- User Management:** For managing users (register, authenticate).
- Loan Management:** For managing book loans (check out, return).

2. Implement Subsystems

python  Copy code

```
# Book Management Subsystem
class BookManager:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)
        print(f"Book '{book}' added.")

    def remove_book(self, book):
        if book in self.books:
            self.books.remove(book)
            print(f"Book '{book}' removed.")
        else:
            print(f"Book '{book}' not found.")

    def search_book(self, book):
        if book in self.books:
            print(f"Book '{book}' is available.")
        else:
            print(f"Book '{book}' is not available.")

# User Management Subsystem
class UserManager:
    def __init__(self):
        self.users = []

    def register_user(self, user):
        self.users.append(user)
        print(f"User '{user}' registered.")

    def authenticate_user(self, user):
        if user in self.users:
            print(f"User '{user}' authenticated.")
        else:
```

```

        print(f"User '{user}' not found.")

# Loan Management Subsystem
class LoanManager:
    def __init__(self):
        self.loans = {}

    def check_out_book(self, user, book):
        if book not in self.loans:
            self.loans[book] = user
            print(f"Book '{book}' checked out by '{user}'.")
        else:
            print(f"Book '{book}' is already checked out.")

    def return_book(self, book):
        if book in self.loans:
            user = self.loans.pop(book)
            print(f"Book '{book}' returned by '{user}'.")
        else:
            print(f"Book '{book}' was not checked out.")

```

3. Implement the Facade

```

python Copy code

# Facade
class LibraryFacade:
    def __init__(self):
        self.book_manager = BookManager()
        self.user_manager = UserManager()
        self.loan_manager = LoanManager()

    def add_book(self, book):
        self.book_manager.add_book(book)

    def remove_book(self, book):
        self.book_manager.remove_book(book)

    def search_book(self, book):
        self.book_manager.search_book(book)

    def register_user(self, user):
        self.user_manager.register_user(user)

    def authenticate_user(self, user):
        self.user_manager.authenticate_user(user)

    def check_out_book(self, user, book):
        if self.user_manager.authenticate_user(user):
            self.loan_manager.check_out_book(user, book)

    def return_book(self, book):
        self.loan_manager.return_book(book)

```

4. Usage

```

python Copy code

# Client Code
library = LibraryFacade()

# Managing books
library.add_book("The Great Gatsby")
library.search_book("The Great Gatsby")
library.remove_book("The Great Gatsby")

# Managing users
library.register_user("John Doe")
library.authenticate_user("John Doe")

# Managing loans
library.check_out_book("John Doe", "The Great Gatsby")
library.return_book("The Great Gatsby")

```

Explanation

- 1. **Subsystems:** The `BookManager``, `UserManager``, and `LoanManager`` handle specific responsibilities.
- 2. **Facade:** The `LibraryFacade`` provides a simplified interface to these subsystems.
- 3. **Client Code:** Clients interact with the `LibraryFacade``, which internally manages the interactions between the subsystems.

This design allows the client to interact with the library system without needing to understand the complexity of the individual subsystems.



Plan VS Enterprise

Upgrade plan
Get GPT-4, DALL-E, and more

Message ChatGPT

ChatGPT can make mistakes. Check important info.

