

1

composite 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, Get citation Design Facebook Social Network, Design payment dateway



Sure! Let's use the Composite pattern to design a simple Library Management System. The Composite pattern is useful when you want to treat individual objects and compositions of objects uniformly.

Composite Pattern Overview

In the Composite pattern:

- Component: An abstract class or interface that declares the interface for all concrete objects and compositions.
- **Leaf**: Represents the individual objects that do not have any children.
- Composite: Represents a group of objects and implements the `Component` interface, allowing them to have children.

Example: Library Management System

Let's design a basic library system where we have books, magazines, and sections. The Composite pattern will help us treat both individual iten. and sections uniformly.

1. Define the Component

```
Copy code
from abc import ABC, abstractmethod
class LibraryComponent(ABC):
   @abstractmethod
   def get_name(self):
   @abstractmethod
   def get_description(self):
   @abstractmethod
   def add(self, component):
   def remove(self, component):
   @abstractmethod
   def display(self, depth):
```

2. Define the Leaf

```
Copy code
```

3. Define the Composite

```
class Section(LibraryComponent):
    def __init__(self, name):
        self.name = name
        self.components = []

    def get_name(self):
        return self.name

    def get_description(self):
        return f"Section: {self.name}"

    def add(self, component):
        self.components.append(component)

    def remove(self, component):
        self.components.remove(component)

    def display(self, depth):
        print(" " * depth + f"Section: {self.get_name()}")
        for component in self.components:
            component.display(depth + 1)
```

4. Use the Composite Pattern

```
# Create leaf components
book1 = Book("Book One", "Description of Book One")
book2 = Book("Book Two", "Description of Book Two")

# Create composite components
section1 = Section("Fiction")
section1.add(book1)
section1.add(book2)

# Create another composite
library = Section("Library")
library.add(section1)

# Display the library structure
library.display(0)
```

Output

```
yaml

Section: Library

Section: Fiction

Book: Book One - Description of Book One

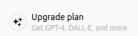
Book: Book Two - Description of Book Two
```

In this example:

- `LibraryComponent` is the abstract class that defines the interface.
- `Book` is a leaf node representing indiv ↓ al books.
- `Section` is a composite that can contain other `LibraryComponent` objects, either `Book` or other `Section`.

This setup allows you to add, remove, and display both individual books and sections of the library uniformly.

< 3/3 > ()) () () () () () () () ()





?