🏭 Pattern Composite per il Preventivatore

📝 Descrizione della Soluzione

Per questo problema, il pattern Composite è la scelta ideale perché:

- Abbiamo una struttura ad albero con sezioni, sottosezioni ed elementi base
- Vogliamo trattare sia gli elementi singoli che i gruppi di elementi in modo uniforme
- Il calcolo dei prezzi segue una logica ricorsiva dove il prezzo di una sezione dipende dai suoi componenti

✅ Vantaggi

- Permette di trattare oggetti singoli e composizioni di oggetti in modo uniforme
- Facilita l'aggiunta di nuovi tipi di componenti
- Semplifica la struttura del client
- Rende più facile aggiungere nuove funzionalità all'intera struttura

❌ Svantaggi

- Può rendere il design troppo generico
- Può essere difficile limitare i tipi di componenti che possono essere aggiunti
- Richiede una buona comprensione della ricorsione per l'implementazione

🏗️ Struttura del Pattern

- `ComponentePreventivo`: Interfaccia base per tutti gli elementi
- `ElementoBase`: Rappresenta gli elementi foglia (prodotti, servizi)
- `Sezione`: Rappresenta i nodi compositi (sezioni e sottosezioni)

💻 Implementazione Dettagliata

🔧 ComponentePreventivo (Componente Base)

```
class ComponentePreventivo(ABC):
    def __init__(self, nome: str, sconto: float = 0):
        self.nome = nome
        self.sconto = sconto
```

```
    @abstractmethod
    def get_prezzo_base(self) -> float:
        pass

    @abstractmethod
    def get_prezzo_scontato(self) -> float:
        pass

    def applica_sconto(self, prezzo: float) -> float:
        return prezzo * (1 - self.sconto)
```

Questa è la classe base astratta che definisce l'interfaccia comune per tutti i componenti del preventivo. Include:

- Metodi astratti per calcolare i prezzi base e scontati
- Metodo di utilità per applicare lo sconto

🌿 ElementoBase (Leaf)

```
class ElementoBase(ComponentePreventivo):
    def __init__(self, nome: str, prezzo: float, sconto: float = 0):
        super().__init__(nome, sconto)
        self.prezzo = prezzo

    def get_prezzo_base(self) -> float:
        return self.prezzo

    def get_prezzo_scontato(self) -> float:
        return self.applica_sconto(self.prezzo)
```

Questa classe rappresenta gli elementi foglia del composite:

- Implementa i metodi concreti per il calcolo dei prezzi
- Gestisce il prezzo base dell'elemento
- Applica lo sconto individuale

📦 Sezione (Composite)

```
class Sezione(ComponentePreventivo):
    def __init__(self, nome: str, sconto: float = 0):
        super().__init__(nome, sconto)
        self.componenti: List[ComponentePreventivo] = []
```

```
    def aggiungi_componente(self, componente: ComponentePreventivo):
        self.componenti.append(componente)

    def get_prezzo_base(self) -> float:
        return sum(comp.get_prezzo_base() for comp in self.componenti)

    def get_prezzo_scontato(self) -> float:
        somma_scontata = sum(comp.get_prezzo_scontato() for comp in
self.componenti)
        return self.applica_sconto(somma_scontata)
```

Questa classe gestisce la struttura composita:

- Mantiene una lista di componenti (che possono essere sia ElementoBase che altre Sezioni)
- Implementa la logica ricorsiva per il calcolo dei prezzi
- Applica lo sconto dopo aver calcolato la somma dei prezzi scontati dei componenti

🚀 Esempio di Utilizzo

```
def main():
    # Creazione della struttura del preventivo
    preventivo = Sezione("Preventivo Completo")

    # Sezione macchina base e accessori
    sez_macchina = Sezione("Macchina base e accessori", 0.05)

    # Configurazione base
    conf_base = Sezione("Configurazione base")
    conf_base.aggiungi_componente(ElementoBase("Macchina", 20000))
    conf_base.aggiungi_componente(ElementoBase("Accessorio1", 500))

    # Accessori extra con sconto del 10%
    acc_extra = Sezione("Accessori extra", 0.1)
    acc_extra.aggiungi_componente(ElementoBase("Accessorio4", 800))

    sez_macchina.aggiungi_componente(conf_base)
    sez_macchina.aggiungi_componente(acc_extra)

    # Calcolo e visualizzazione dei prezzi
    print(f"Prezzo base totale: €{preventivo.get_prezzo_base():.2f}")
    print(f"Prezzo scontato totale: €{preventivo.get_prezzo_scontato():.2f}")
```

📊 Output di Esempio

```
Prezzo base totale: €24800.00
Prezzo scontato totale: €23247.40
```

📝 Note Implementative

- La struttura permette di aggiungere facilmente nuove sezioni e sottosezioni
- Gli sconti vengono applicati in cascata, rispettando la gerarchia
- Il calcolo dei prezzi è completamente trasparente per il client
- La manutenibilità è elevata grazie alla separazione delle responsabilità