🏭 Preventivatore

🎯 Pattern Scelto: Composite

📝 Descrizione del Problema

Dovete costruire un preventivatore. Questo preventivatore è diviso in diverse sezioni, ogni sezione può contenere settosezioni e/o elementi base (accessori, servizi, ecc)
Ogni sezione/sottosezione/elemento può avere una quantità e uno sconto. Gli elementi base derivano da un listino e hanno un prezzo base di partenza.
Il software deve mostrare per ogni sezione/sottosezione il suo prezzo base e il prezzo applicato lo sconto.
Il prezzo base di una sezione/sottosezione è dato dalla somma dei prezzi base dei suoi elementi o sottosezioni. Il prezzo scontato è calcolato usando come base la somma dei prezzi scontati delle sue sottosezioni e elementi, sulla quale poi viene applicato lo sconto della sezione.
Il software mostra poi un costo totale con e senza sconti applicati

🧠 Ragionamento

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

- Abbiamo una struttura ad albero con sezioni, sottosezioni ed elementi base
- 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 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 Semplificata

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

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

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

🚀 Esempio di Utilizzo

```
preventivo = Sezione("Preventivo Completo")
sez_macchina = Sezione("Macchina base", 0.05)
sez_macchina.aggiungi_componente(ElementoBase("Macchina", 20000))
```

📊 Output di Esempio

```
Prezzo base totale: €20000.00
Prezzo scontato totale: €19000.00
```

📝 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