📋 Preventivatore v2 - Documentation

🎯 Pattern Scelto: Strategy

Il pattern Strategy è stato scelto per le seguenti ragioni:

- Permette di definire una famiglia di algoritmi di calcolo preventivi
- Rende gli algoritmi intercambiabili
- Isola la logica di calcolo dal resto dell'applicazione
- Facilita l'aggiunta di nuove strategie di calcolo

✅ Vantaggi

- Flessibilità nel cambiare l'algoritmo di calcolo a runtime
- Separazione delle responsabilità
- Facile aggiunta di nuove strategie
- Eliminazione di condizioni multiple switch/if

❌ Svantaggi

- Aumenta il numero di oggetti nel sistema
- Il client deve conoscere le differenze tra le strategie
- Overhead di comunicazione tra Strategy e Context

💻 Implementazione

```
from abc import ABC, abstractmethod

# Strategy Interface
class StrategiaPreventivo(ABC):
    @abstractmethod
    def calcola_preventivo(self, dati: dict) -> float:
        pass

# Esempio di Concrete Strategy
class PreventivoBase(StrategiaPreventivo):
    def calcola_preventivo(self, dati: dict) -> float:
        return dati.get('importo_base', 0) * 1.22  # IVA 22%

# Context
```

```
class Preventivatore:
    def __init__(self, strategia: StrategiaPreventivo):
        self._strategia = strategia
```

🖥️ Output

```
Preventivo Base: 1220.0
```

Il codice implementa un sistema flessibile per il calcolo dei preventivi con diverse strategie. L'esempio mostra solo la strategia base che calcola il prezzo con IVA, ma il sistema è progettato per supportare facilmente l'aggiunta di nuove strategie di calcolo.