📋 Preventivatore v2

🎯 Pattern Scelto: Facade

📝 Analisi del Problema

Vi viene chiesto di andare ad aggiornare un unico file excel su Google Drive ogni volta che un ordine viene aggiunto, modificato o eliminato. Ad ogni ordine corrisponde una riga.
La libreria che vi permette di scrivere su Drive lavora a basso livello, per ogni dato che dovete andare a scrivere vi serve il link del file (sempre lo stesso), il nome del foglio (sempre lo stesso), le coordinate su cui lavorare, e infine i dati da scrivere.
Ogni ordine ha il suo codice univoco, e deve rimanere all'oscuro del fatto che lo state esportando su Drive.
Possiamo riassumere i passaggi delle 3 operazioni da svolgere:

- Aggiunta ordine
- Modifica ordine
- eliminare la riga

🧠 Ragionamento

Ho scelto il Facade pattern per questi motivi:

- Use the Facade pattern when you need to have a limited but straightforward interface to a complex subsystem. ➡️ Questo è il caso d'uso tipico del Facade pattern e il problema che vogliamo risolvere rispecchia questa descrizione.
- Fornisce un'interfaccia semplificata per il sistema complesso di calcolo preventivi
- Nasconde la complessità del sistema sottostante
- Riduce le dipendenze tra il client e i sottosistemi
- Facilita l'utilizzo del sistema di preventivazione

✅ Vantaggi

- Semplifica l'interfaccia per il client
- Disaccoppia il sottosistema dai client
- Fornisce un punto di accesso unificato
- Migliora la manutenibilità del codice

❌ Svantaggi

- Può introdurre un livello di indirezione non necessario se non gestito correttamente

💻 Implementazione

```python
# Sottosistemi
class CalcolatoreIVA:
    def calcola_iva(self, importo: float) -> float:
        return importo * 0.22

class CalcolatoreBase:
    def calcola_base(self, dati: dict) -> float:
        return dati.get('importo_base', 0)

# Facade
class PreventivatoreFacade:
    def __init__(self):
        self._calcolatore_iva = CalcolatoreIVA()
        self._calcolatore_base = CalcolatoreBase()

    def calcola_preventivo(self, dati: dict) -> float:
        importo_base = self._calcolatore_base.calcola_base(dati)
        iva = self._calcolatore_iva.calcola_iva(importo_base)
        return importo_base + iva
```

Il codice implementa un sistema di preventivazione che utilizza il pattern Facade per nascondere la complessità del calcolo dei preventivi. La facade fornisce un'interfaccia semplice per il client, mentre gestisce internamente l'interazione con i vari sottosistemi di calcolo.