OO design

Attività dell'OO design

- Valutare modalità alternative per realizzare il sistema
  - a partire dai requisiti, modelli di analisi e dall'architettura software
  - ***Decisioni di progetto***
    - Utile documentarle a futura memoria: *perché si è fatto così?*
- Riuso della conoscenza di soluzioni di successo: **design pattern**
- Riuso di librerie e ***OO framework***
- Applicare principi di OO design per evitare il **debito tecnico**

Principi di OO design

- Information hiding
- Alta coesione
- Basso accoppiamento
- Presentazione separata
- Do Not Repeat Yourself (DRY)
- Principi SOLID

Principio di information hiding

Ogni componente deve custodire dei segreti al proprio interno

se la decisione che determina il segreto cambia, solo il componente interessato sarà modificato

- Per sottosistemi:
  - L'interfaccia del sottosistema (gruppo di operazioni) è pubblica: **servizio**
  - L'implementazione è privata (e quindi nascosta all'esterno)
- Per package:
  - Solo le classi strettamente necessarie sono pubbliche
  - Tutte le altre sono private
- Per classi:
  - Solo le operazioni strettamente necessarie sono pubbliche
  - Tutte le altre operazioni sono private
  - Tutte le variabili di istanza (attributi) sono private: **incapsulamento dei dati**

Alta coesione

Coesione: misura il grado di dipendenza tra elementi di uno stesso componente (sottosistema o classe)

- Un componente ad ***alta coesione*** ha una responsabilità ben definita
- Un componente a bassa coesione si occupa di cose disparate e quindi è:
  - difficile da comprendere
  - difficile da riusare
  - difficile da modificare (e le modifiche saranno frequenti)

Obiettivo: assegna le responsabilità in modo tale da ottenere componenti con responsabilità ben definite

- Una bassa coesione si risolve delegando le responsabilità ad altri componenti

Basso accoppiamento

Accoppiamento: misura il grado di dipendenza tra componenti diversi (tra sottosistemi o tra classi)

- Alto accoppiamento: un cambiamento a un componente si propaga facilmente a tutti i componenti che dipendono da esso
- ***Basso accoppiamento***: un cambiamento a un componente non si propaga ad altri componenti

Obiettivo:
assegna le responsabilità ai componenti in modo tale da limitare l'impatto dei cambiamenti

- Un alto accoppiamento si risolve applicando il principio di information hiding

Presentazione separata

- Si intende «*la parte di codice relativa alla presentazione deve essere tenuta separata dal resto dell'applicazione*»
  - Specializzazione di un principio più generale: ***separazione degli interessi***
- La logica di presentazione e la logica di dominio sono più facili da capire se tenute separate
- È possibile esporre la logica di dominio come API/servizio
- È possibile supportare presentazioni differenti senza duplicare il codice
- Senza parti grafiche è possibile scrivere i casi di test con asserzioni testuali

Do Not Repeat Yourself (DRY)

Ogni elemento di conoscenza deve avere una sola, non ambigua, autorevole rappresentazione all'interno di un sistema

ovvero

Ogni parte significativa di una funzionalità dovrebbe essere implementata in un unico posto del codice sorgente

- evitare sequenze di istruzioni uguali fra loro (*cloni*), spesso frutto di *copia-incolla*
- se ci si imbatte in ripetizioni occorre creare un'astrazione appropriata

Principi SOLID

Single Responsibility	**Non deve esistere più di una ragione per cui una classe debba essere modificata** 1 classe → 1 responsabilità
Open/Closed	**Le entità della programmazione devono essere aperte alle estensioni e chiuse alla modifica** Classi/Moduli/Funzioni estendibili
Liskov Substitution	**Le sottoclassi devono poter essere utilizzate al posto della loro classe padre** Uso corretto della generalizzazione
Interface Segregation	**I Client non devono dipendere da interfacce che non usano** Interfacce specializzate per ogni client sono migliori di una singola grossa interfaccia
Dependency Inversion	**Moduli di alto livello non devono dipendere da moduli di basso livello, entrambi devono dipendere da astrazioni** Le astrazioni non devono dipendere dai dettagli, ma viceversa