

# GRASP

- General Responsibility Assignment Software Patterns -

L'applicazione in un contesto di  
**RESPONSIBILITY-DRIVEN-DEVELOPMENT**

- gli oggetti software sono considerati dotati di responsabilità  
una responsabilità si intende un'estrazione di ciò che fa o rappresenta un oggetto software
- due tipi di responsabilità
  - di conoscere (metodi)
  - di fare (informazioni come campo o che può manipolare)
- Un progetto OO viene visto come una comunità di oggetti con responsabilità che collaborano

## Pattern GRASP

- CREATOR
- INFORMATION EXPERT
- LOW COUPLING
- CONTROLLER
- HIGH COESION
- POLYMORPHISM
- PURE FABRICATION
- INDIRECTNESS
- PROTECTED VARIATIONS

## CREATOR

Problema: Chi deve essere responsabile della creazione di una nuova istanza di una classe?

Soluzione: Assegna ad una classe B la responsabilità di instanziare A se almeno una delle seguenti condizioni è vera:

- B contiene o aggira un' composizione di oggetti di tipo A
- B registra A (AKA salva le reference, ha come campo un puntatore ad A)
- B utilizza strettamente A
- B possiede i dati per l'inizializzazione di A, che saranno passati ad A al momento della sua creazione (B è un EXPERT rispetto ad A)

Il pattern Creator favorisce il low coupling, è importante però individuare un creatore che abbia davvero bisogno di essere collegato all'oggetto creato.

In certi contesti è opportuno utilizzare un pattern di design "Factory" se la creazione può essere in alternativa a "nullo", se una proprietà esterna condiziona le scelte della classe creatrice tra un insieme di classi simili.

## INFORMATION EXPERT

**Problema:** Qual'è un principio di base, generale, per l'eseguzione di responsabilità agli effetti?

**Soluzione:** Assegna una responsabilità alla classe che possiede le informazioni necessarie per soddisfarla, ovvero alla classe che possiede le informazioni necessarie per soddisfare la responsabilità.

Si individuano informazioni per le classi diverse sono esperte, queste collaborano insieme per realizzare un obiettivo. (informazioni distribuite, classi più leggere, senza perdere l'incapacitamento)

Gli effetti software, a differenza di quelli reali, hanno la responsabilità di compiere azioni sulle cose che conoscono (Do it myself)

## LOW COUPLING (assieme ad High Cohesion sono i molti sufficienti)

**Problema:** Come ridurre l'impatto dei cambiamenti? Come sostener un dipendenza bassa, un impatto dei cambiamenti basso e una maggiore opportunità di riuscita?

**Soluzione:** Assegna le responsabilità in modo tale che l'accoppiamento non necessario rimanga basso. Usa questo principio per valutare le alternative.

Problemi nelle classi ad accoppiamento alto:

- I cambiamenti nelle classi correlate obbligano cambiamenti locali
- Sono più difficili da comprendere in insieme
- Sono più difficili da misurare

## Vantaggi delle classi con accoppiamento basso

- + Non sono influenzate dai cambiamenti nelle altre classi o componenti
- + È semplice da copiare in isolamento
- + È conveniente da risuonare

o perniciosi  
)

NOTA: Un accoppiamento con elementi ritentati stabili difficilmente costituisce un problema

- Le classi puramente generiche, che hanno un'altra probabilità di riuso, devono avere un grado di accoppiamento particolarmente basso
- Ogni sottoclasse è fortemente accoppiata alla sua superclasse. Estendere una classe è da evitare quando possibile
- Puntazioni di codice duplicato sono fortemente accoppiate tra di loro

## CONTROLLER

**Problema:** Qual'è il primo effetto oltre lo stato UI che riceve e coordina un'operazione di sistema?

**Soluzione:** Assegna le responsabilità a un effetto che rappresenta un delle seguenti scelte:

(facade controller). Rappresenta il sistema complessivo

- Rappresenta uno scenario di un caso d'uso

Nel controller di uso d'uso (o di sessione)

- Si utilizza la stessa classe controller per tutti gli eventi di sistema nello stesso scenario di uso d'uso
- Una sessione è un'istanza di una conversazione con un attore. La lunghezza delle sessioni corrisponde spesso all'esecuzione del caso d'uso stesso

Le classi dello stato UI **non** sono controller

I controller **non** sono classi di dominio

Un controller è un pattern che delega, è una sorta di facciata tra lo stato UI e quelli di dominio, consente di progettare gli effetti di dominio in maniera indipendente dagli effetti dell'interfaccia utente. Il controller non esegue di per sé molto lavoro.

### Vantaggi:

- + Maggiore potenzialità di uso ed interface inseribili
- + Opportunità di agire sullo stato del caso d'uso, è possibile concordare che le operazioni avvengano in una sequenza legale, oppure si desidera regolare sulle operazioni all'interno della sessione

## HIGH COESION

**Problema:** Come mantenere gli effetti focalizzati, comprensibili e gestibili e, come effetti collaterali, sostenerne il **low coupling**?

**Soluzione:** Assegnare le responsabilità in modo tale che la coesione rimanga alta. Usare questi principi per valutare alternative

Una classe con coesione bassa fa molte cose non correlate e soffre troppo diverso

- Sono difficili da comprendere
- Sono difficili da mantenere
- Sono difficili da risuonare
- Sono delicate, continuamente soggette a cambiamenti

**Tipologie di coesione:** (ordinate per bontà)

- DI DATI una classe implementa un tipo di dati
- FUNZIONALE gli elementi di una classe eseguono una singola funzione
- TEMPORALE gli elementi sono variati circa nello stesso tempo
- PER PURA COINCIDENZA

C'è una coesione funzionale alta quando gli elementi di un componente lavorano tutti assieme per fornire un comportamento ben circoscritto

Contesti nei quali è accettabile una gestione bassa:

- Quando per un determinato compito ci vuole un esperto
- Per gli oggetti distribuiti che servono per avere migliori prestazioni (RMI o middleware affini)

### Vantaggi:

- + Sostiene la maggiore chiarezza e facilità di comprensione del progetto
- + Sparsi sistemi low coupling
- + La manutenzione e i miglioramenti risultano semplificati
- + Maggiore niss di funzionalità → grande fine e elementi correlate, poiché una classe, se creata, può essere usata per uno scopo molto specifico