Introduzione a Spring

plansoft
advanced software solutions

SOLID

I 5 design principles della programmazione a oggetti

Single responsibility

Open closed

Liskov substitution

Interface segregation

Dependency inversion

SOLID

Rendere il software:

Più **comprensibile**

Più **flessibile**

Più **mantenibile**

Single responsibility

“Una classe dovrebbe avere un solo motivo per cambiare” 

“Una classe dovrebbe avere una sola responsabilità” 

“Una classe dovrebbe avere una singola responsabilità, dove per responsabilità si intende un motivo per cambiare”

Esempio: Pensiamo a un dipendente che ha come responsabilità quella di calcolare il suo stipendio, ma anche quella di controllare se entro l’anno riceverà una promozione e aggiungiamoci anche quella di modificare i suoi dati anagrafici o di ricordarsi anche tutte le ore che ha fatto durante l’anno.

Open Closed

“I componenti del software (classi, moduli, funzioni) devono essere aperti per essere estesi, ma chiusi rispetto alle modifiche”

Questo per permettere che il comportamento di un elemento sia estendibile senza modificare il suo codice sorgente.

Esempio: Pensiamo a una situazione in cui abbiamo un insieme di triangoli e cerchi di cui dobbiamo calcolare l’area. Creiamo un calcolatore solo per calcolare triangoli e cerchi. Successivamente viene richiesto di calcolare le aree di un insieme di quadrati, ma il nostro calcolatore è pensato solo per calcolare le aree delle prime 2 forme, quindi dobbiamo crearne uno nuovo.

Liskov substitution

“Se S è un sottotipo di T allora oggetti dichiarati in un programma di tipo T possono essere sostituiti con oggetti di tipo S senza alterare la correttezza dei risultati del programma”

Esempio: Immaginiamo una classe Persona che ha come funzione quella di restituire il suo nome, prendiamo ora Dipendente che è un sottotipo di Persona. Dipendente potrà ancora restituire il suo nome perchè è sottotipo di Persona.

Interface segregation

“Una classe non deve dipendere da metodi che non usa”

Questo significa che è meglio avere più interfacce piccole che poche interfacce con tanti metodi.

Questo permette di mantenere gli elementi del sistema il più indipendenti possibile permettendo anche una miglior capacità di gestione del codice.

Esempio: Pensiamo a un grande ristorante in cui una sola persona si occupa di prendere gli ordini, accompagnare i clienti, cucinare e accettare i pagamenti.

Dependency inversion

“I moduli di alto livello non devono dipendere da quelli di basso livello, ma entrambi devono dipendere da astrazioni”

“Le astrazioni non devono dipendere dai dettagli, ma sono i dettagli che dipendono dalle astrazioni”

Esempio: vedi slide successiva

Esempio di applicazione dei principi

Pensiamo a un dipendente che ha come responsabilità quella di calcolare il suo stipendio, ma anche quella di controllare se entro l'anno riceverà una promozione e aggiungiamoci anche quella di modificare i suoi dati anagrafici o di ricordarsi anche tutte le ore che ha fatto durante l'anno.

Applichiamo i principi e vediamo la sua trasformazione.

In breve i 5 design principles

Single responsibility = Una classe dovrebbe avere una singola responsabilità, dove per responsabilità si intende un motivo per cambiare

Open closed = I componenti del software (classi, moduli, funzioni) devono essere aperti per essere estesi, ma chiusi rispetto alle modifiche

Liskov substitution = Se S è un sottotipo di T allora oggetti dichiarati in un programma di tipo T possono essere sostituiti con oggetti di tipo S senza alterare la correttezza dei risultati del programma

Interface segregation = è meglio avere più interfacce piccole che poche interfacce con tanti metodi

Dependency inversion = Le astrazioni non devono dipendere dai dettagli, ma sono i dettagli che dipendono dalle astrazioni

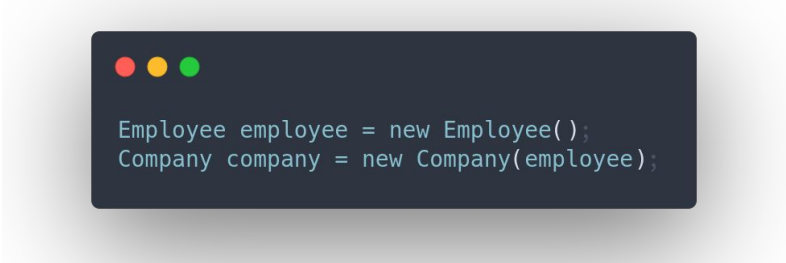
La tecnica del dependency injection

È un design che si basa sul dependency inversion.

Utilizzare questa tecnica significa iniettare a degli oggetti altri oggetti da cui dipendono.

In termini di codice questo significa passare per argomento l'intero oggetto da cui dipende un oggetto.

Esempio: un'azienda dipende dai suoi dipendenti



Esercizi

Esercizi

- riproduzione di un ristorante: arrivo dei clienti, camerieri/cameriere richiedono ordine, consegnano l'ordine al/alla cuoco/cuoca, clienti mangiano, clienti pagano al/alla cassiere/cassiera e se ne vanno
- riproduzione di una fabbrica: entrata dei/delle lavoratori/lavoratrici, pausa pranzo, rientro dei/delle lavoratori/lavoratrici, fine turno
- riproduzione di un negozio commerciale: inizializzazione dei/delle commessi/commesse, entrata dei clienti, assistenza clienti, accettazione pagamento clienti e rimozione clienti dal negozio
- riproduzione di quello che volete basta trovare un modo di applicare i 5 design