Cosa sono i Design Pattern, perché si usano o perché si dovrebbero usare

11 settembre 2018 Francesca Miotto

La progettazione di software è attività molto complessa, nella quale una delle maggiori difficoltà del progettista è individuare correttamente un insieme di oggetti che sia il più possibile riutilizzabile e per il quale siano definite al meglio le relazioni tra classi e le gerarchie di ereditarietà.

Ciò non esclude tuttavia la possibilità di imbattersi in altri problemi, spesso ricorrenti e prevedibili.

Per questo motivo, mutuandoli dall'utilizzo che se n'è fatto originariamente in architettura, anche in ambito software si utilizzano i **Design Pattern**, degli schemi utilizzabili nel progetto di un sistema che permettono di non inventare da capo soluzioni a problemi già risolti, ma di utilizzare "mattoni" di provata efficacia in situazioni simili.

*Potremmo definire un Design Pattern come "*una soluzione progettuale generale ad un problema ricorrente in un contesto specifico**"**.

Perché è importante studiarli:

A nessun programmatore verrebbe in mente di ri-implementare una libreria che funziona, o scrivere ogni volta da capo una lista. Per cui valgono le seguenti regole:

- "Don't reinvent the wheel"

il catalogo dei Design Pattern non è stato elaborato in maniera teorica ma proviene dall'esperienza "sul campo" di progettisti software ed offrono soluzioni collaudate a problemi tipici, della cui applicazione si conoscono benefici e limitazioni.

- Riconoscere al volo un problema di progettazione

i Design Pattern mantengono un impianto generico permettendone l'applicabilità a classi di problemi: se si conoscono in anticipo si farà molta meno fatica (e si perderà meno tempo) a risolvere problemi che hanno una struttura simile.

- Rendere più chiaro un progetto

i Design Pattern rappresentano spesso un "vocabolario" comune tra progettisti software. Nominarli e individuarli consente di risparmiare molti sforzi di comunicazione ed è quindi auspicabile citarli nel proprio progetto qualora se ne faccia uso.

- Qualità del progetto

Un corretto utilizzo dei Design Pattern rende il progetto (e il codice) più snello e comprensibile, favorisce il riuso del codice e la sua manutenzione (convenzioni, refactoring, estendibilità).

Le caratteristiche principali dei Design Pattern sono:

- **relativamente astratti:** devono poter essere usati e condivisi da progettisti con punti di vista diversi;

- **non complessi**

- **non domain specific**, ovvero non rivolti a specifiche applicazioni ma riusabili in parti di applicazioni diverse

Classificazione dei Pattern

Un primo criterio di classificazione dei Pattern riguarda lo scopo (purpose). Essi sono:

- **creazionali** -> riguardano il processo di creazione di oggetti (ex. Singleton)

- **strutturali** -> utilizzati per definire la struttura del sistema in termini della composizione di classi ed oggetti. Si basano sui concetti OO di ereditarietà e polimorfismo. (ex. Adapter)

- **comportamentali** -> si occupano di come gli oggetti interagiscono reciprocamente e distribuiscono tra di essi le responsabilità (ex. Strategy)

Un secondo criterio riguarda il raggio di azione (scope):

- **classi:** pattern che definiscono le relazioni fra classi e sottoclassi. Le relazioni sono basate prevalentemente sul concetto di ereditarietà e sono quindi **statiche** (definite a tempo di compilazione).

- **oggetti:** pattern che definiscono relazioni tra oggetti, che possono cambiare durante l'esecuzione e sono quindi più **dinamiche**

Classificazione completa

Raggio d'azione		Scopo: Creazionale	Scopo: Strutturale	Scopo: Comportamentale
	Classi	**Factory Method**	**Adapter** (class)	Interpreter **Template Method**
	Oggetti	**Abstract Factory** Builder Prototype Singleton	**Adapter** (object) Bridge **Composite** **Decorator** Facade Flyweight Proxy	Chain of responsability Iterator Mediator Memento **Observer** State **Strategy** Visitor

Descrizione dei Design Pattern

Nome e Classificazione: il nome illustra l'essenza di un pattern definendo un vocabolario condiviso tra i progettisti; la classificazione lo identifica in termini di scopo e raggio di azione.

Motivazione: scenario che descrive in modo astratto il problema al quale applicare il pattern; può includere la lista eventuali pre-condizioni necessarie a garantirne l'applicabilità.

Applicabilità: Descrive le situazioni in cui il pattern può essere applicato.

Struttura: descrive graficamente la configurazione di elementi che risolvono il problema (relazioni, responsabilità, collaborazioni).

Partecipanti: classi ed oggetti che fanno parte del pattern con le relative responsabilità.
Conseguenze: risultati che si ottengono applicando il pattern.

Implementazioni: tecniche e suggerimenti utili all'implementazione del pattern.

Codice di esempio: frammenti di codice che illustrano come implementare in un certo linguaggio di programmazione Usi conosciuti: esempi di applicazioneω(es. java o c++) il pattern. w in sistemi reali

Pattern correlati: altri pattern correlati

Per chi volesse approfondire il tema dei design pattern e imparare come metterli in pratica nella realizzazione dei propri progetti, consigliamo la partecipazione ai corsi di formazione specifici:

1. Corso di formazione sulla programmazione con Design Patterns

2. Corso di formazione in architettura e progettazione del software