Design Patterns

I **Design Pattern del Gang of Four (GoF)**, introdotti nel libro *"Design Patterns: Elements of Reusable Object-Oriented Software"* scritto da **Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides**, costituiscono un catalogo di soluzioni comprovate a problemi comuni nello sviluppo software.

I GoF Design Patterns promuovono le **migliori pratiche**, la **riusabilità del codice** e la **separazione delle responsabilità**, favorendo la creazione di applicazioni robuste e scalabili.

Cosa sono i Gang of Four (GoF) Design Patterns

I **Gang of Four Design Patterns** sono un insieme di soluzioni ai problemi ricorrenti nello sviluppo e nella progettazione del software.

Sono stati introdotti per la prima volta nel 1994 nel libro *"Design Patterns: Elements of Reusable Object-Oriented Software"*.

Gli autori – Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides – sono collettivamente conosciuti come **Gang of Four (GoF)**.

Perché si chiamano "Gang of Four"?

Il termine "Gang of Four" deriva semplicemente dal fatto che **i quattro autori** hanno scritto insieme il libro che ha rivoluzionato il modo di pensare la progettazione software, fornendo soluzioni eleganti e riutilizzabili ai problemi più comuni.

Tipologie di Design Pattern GoF

I **23 design pattern GoF** si dividono in **tre categorie principali**:

1. **Creazionali (Creational Patterns)**
2. **Strutturali (Structural Patterns)**
3. **Comportamentali (Behavioral Patterns)**

Questi pattern offrono soluzioni ai problemi più frequenti nella progettazione del software e rendono i sistemi più **modulari, flessibili e mantenibili**.

Design Pattern Creazionali

Immagina di essere in una pizzeria e desiderare una pizza deliziosa.

Il cuoco non butta a caso gli ingredienti: segue una **ricetta**.

Nel software, quando dobbiamo “creare” oggetti, servono delle “ricette” ben definite: i **Creational Design Patterns**.

Sono come **ricette segrete** che definiscono **modi intelligenti e organizzati per creare oggetti**, nascondendo i dettagli complessi della costruzione.

Tipi di Design Pattern Creazionali

- **Factory Method**

  Definisce un’interfaccia per creare oggetti, ma lascia alle sottoclassi la decisione su quale tipo di oggetto istanziare.

  È come avere un progetto flessibile per costruire vari tipi di prodotti.

- **Abstract Factory**

  Permette di creare **famiglie di oggetti correlati** (ad esempio piatti, posate, bicchieri coordinati), garantendo coerenza tra gli elementi creati.

- **Singleton**

  Garantisce che una classe abbia **una sola istanza** e fornisce un punto di accesso globale, utile per la gestione centralizzata (es. configurazioni, logger).

- **Prototype**

  Consente di **creare oggetti clonando istanze esistenti**, evitando di costruirli da zero.

  Ideale quando la creazione di un oggetto è costosa.

- **Builder**

  Guida la creazione di oggetti complessi **passo dopo passo**, separando la costruzione dalla rappresentazione finale.

- **Object Pool**

  Gestisce un insieme di oggetti riutilizzabili, come connessioni a un database o thread.

  Invece di creare e distruggere oggetti continuamente, li **ricicla** per risparmiare risorse.

Design Pattern Strutturali

I **Structural Design Patterns** spiegano **come combinare classi e oggetti** per costruire strutture più grandi e flessibili, proprio come un architetto unisce mattoni e travi per creare una casa.

Tipi di Design Pattern Strutturali

- **Adapter**

  Permette a classi con interfacce incompatibili di collaborare.

  Funziona come un adattatore elettrico tra due prese diverse.

- **Bridge**

  Separa **l'astrazione** (il comportamento) dall'**implementazione**, così da poterle modificare indipendentemente.

- **Composite**

  Permette di creare **gerarchie di oggetti** (es. rami e foglie in un albero) trattando oggetti singoli e composizioni allo stesso modo.

- **Decorator**

  Aggiunge **nuove funzionalità** a un oggetto senza modificare il suo codice originale.

  È come aggiungere strati di carta regalo a un pacco.

- **Facade**

  Fornisce un'interfaccia **semplice e unificata** a un sistema complesso, nascondendo i dettagli interni.

- **Flyweight**

  Riduce l'uso della memoria **condividendo oggetti simili** invece di crearne di nuovi ogni volta.

- **Proxy**

  Fornisce un **sostituto o intermediario** per controllare l'accesso a un oggetto reale (come un telecomando per una TV).

Design Pattern Comportamentali

I **Behavioral Design Patterns** si concentrano su **come gli oggetti collaborano e comunicano tra loro**.

Sono come le regole di un puzzle: definiscono come i pezzi devono incastrarsi per funzionare insieme in modo armonioso.

Tipi di Design Pattern Comportamentali

- **Chain of Responsibility**

  Passa una richiesta attraverso una **catena di oggetti**, ognuno dei quali può gestirla o passarla oltre.

  Separa chi invia la richiesta da chi la gestisce.

- **Command**

  Trasforma una richiesta in un **oggetto indipendente** che contiene tutte le informazioni necessarie per eseguirla.

  Separa l'emissione di un comando dalla sua esecuzione.

- **Iterator**

  Fornisce un modo per **scorrere una collezione** senza esporne la struttura interna.

- **Mediator**

  Crea un **oggetto intermediario** che gestisce la comunicazione tra diversi oggetti, riducendo le dipendenze reciproche.

- **Memento**

  Salva lo **stato interno di un oggetto** per poterlo ripristinare in seguito (come salvare una partita di un videogioco).

- **Observer**

  Definisce una relazione **uno-a-molti** tra oggetti: quando un oggetto cambia stato, tutti i suoi osservatori vengono notificati.

- **State**

  Permette a un oggetto di **cambiare comportamento dinamicamente** in base al proprio stato interno.

- **Strategy**

  Definisce una **famiglia di algoritmi** intercambiabili e permette di selezionarli a runtime senza modificare il codice che li usa.

- **Template Method**

  Definisce la **struttura generale di un algoritmo**, lasciando alle sottoclassi la possibilità di personalizzare alcuni passaggi.

- **Visitor**

  Aggiunge **nuove operazioni** a strutture di oggetti esistenti senza modificarne le classi.

  Utile per eseguire azioni su insiemi di oggetti eterogenei.

Conclusione

I **Design Pattern** sono come **ricette intelligenti** per scrivere codice più efficiente e ben progettato.

Aiutano a risolvere problemi comuni, a mantenere il codice flessibile e a semplificare le modifiche future.

I **Gang of Four Design Patterns**, suddivisi in **creazionali, strutturali e comportamentali**, rappresentano le fondamenta della progettazione orientata agli oggetti moderna.