Sommario:

* Introduzione
* Cos’è un design pattern
* Perché i design pattern
* Livelli di progettazione
* Utilizzare i design pattern

1. **Introduzione:**

Immagine che contiene testo, schermata, Carattere, Blu elettrico

Il contenuto generato dall'IA potrebbe non essere corretto.

**Progettare software object oriented è difficile**:

Riusabilità:

* Ci vuole molta esperienza.

Soluzioni comuni e ricorrenti (pattern):

* Flessibili, eleganti, riusabili.

**Design Pattern**:

* Raccolta dell’esperienza dei progettisti, presentata in un modo effettivamente utilizzabile.
* Riutilizzo di soluzioni architetturali vincenti

1. **Cos’è un Design Pattern?**

Ogni modello descrive un problema **che si verifica ripetutamente nel nostro ambiente**, e quindi descrive il nucleo della soluzione a quel problema, in modo tale da poter utilizzare questa soluzione un milione di volte, senza mai farlo due volte nello stesso modo.

Quattro elementi essenziali:

* Nome del pattern:
  + Vocabolario di progettazione.
* Il problema che il pattern risolve:
  + Descrizione del contesto.
* La soluzione:
  + Elementi, relazioni, responsabilità e collaborazioni
* Le conseguenze:
  + Risultati e limiti della soluzione

ESEMPI DI DESIGN PATTERN:

**Model-View-Controller (MVC)**:

Disaccoppia le tre componenti, rendendole riusabili.

* **Model**: dati di business e regole di accesso.
* **View**: rappresentazione grafica.
* **Controller**: reazione della UI agli input utente.

Immagine che contiene testo, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.

**MVC: Model – View**:

* Disaccoppiamento → protocollo di notifica.
* View deve garantire una visualizzazione consistente.

**Diversi design pattern:**

* Observer design pattern.
* Composite design pattern.

Immagine che contiene testo, schermata, diagramma, schizzo

Il contenuto generato dall'IA potrebbe non essere corretto.

**MVC: View – Controller**:

Permette la modifica del comportamento in risposta all’input.

Strategy/Algorithm design pattern.

**Descrizione di un Design Pattern**:

▪ Convenzioni per la specifica

▪Nome e classificazione

▪Sinonimi: altri nomi noti del pattern

▪Motivazione: problema progettuale

▪Applicabilità: contesti in cui il pattern può essere applicato

▪Struttura: rappresentazione grafica delle classi

▪Partecipanti: classi e/o oggetti partecipanti e responsabilità

▪Collaborazioni tra i partecipanti

▪Conseguenze: costi e benefici

▪Implementazione: suggerimenti, tecniche, errori comuni

▪Esempio di codice sorgente: possibile implementazione

▪Utilizzo comune: il pattern nei sistemi reali

▪Pattern correlati

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

1. **Perché i design pattern?**

Risolvere Problemi Architetturali:

1. Utilizzare gli oggetti **appropriati**.

È difficile decomporre il sistema in oggetti!

* Oggetti “reali” ≠ oggetti “non reali”.

DP permettono di identificare le astrazioni meno ovvie.

1. Utilizzare la giusta granularità:

Come decidere cosa deve essere un oggetto?

DP forniscono la granularità più appropriata.

1. Utilizzare le **giuste interfacce**:

Gli oggetti sono “conosciuti” solo per il proprio insieme di operazioni.

DP definiscono cosa mettere/non nell’interfaccia

CLOUD: Sito Web

Immagine che contiene testo, schermata, diagramma, design

Il contenuto generato dall'IA potrebbe non essere corretto.

Immagine che contiene testo, schermata, Carattere, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

Si occupano di questo i Cloud Architect, noi usiamo il cloud per caricarci i nostri software (li carichiamo su macchine virtuali).

1. **Livelli di Progettazione**

Progettare applicazioni:

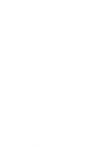
* Riuso “interno”
* Riduzione delle dipendenze

Toolkit (librerie software):

* Insieme di oggetti correlati e riusabili progettati per fornire funzionalità generiche.
* Riutilizzo del codice (code reuse): C++/Java I/O stream library

Immagine che contiene testo, schermata, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.



**Framework:**

* Insieme di classi che cooperano per costruire architetture riutilizzabili per sviluppare un dominio di applicazioni.
* Impone un **disegno architetturale**.
  + Riutilizzo architetturale.
* “Inversion of control”
* Minor grado di accoppiamento possibile.
* NON sono design pattern! Infatti:
  + DP sono più astratti
  + DP disegnano architetture più piccole
  + DP sono meno specializzati

**Tipologie di Pattern:**

**Immagine che contiene testo, schizzo, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.**

**Design Pattern Architetturali:**

* Pattern di alto livello.
  + Guida nella scomposizione in sottosistemi.
    - Ruoli e responsabilità
    - Regole di comunicazione fra le componenti
* Agglomerati di pattern architetturali
* **Esempi:**
* Paradigma client-server
* Model-View-Controller (MVC)
* Peer to peer

**Design Pattern Progettuali**:

Progettazione di dettaglio di componenti:

Definiscono micro-architetture: schema di comunicazione fra gli elementi di un sistema software.

Basi per costruire i pattern architetturali:

Sono i pattern (Gang of Four): GoF

* Factory
* Command
* Proxy
* Observer

**Design Pattern e Idiomi**:

* Basso livello di astrazione.
* Specifici del linguaggio di programmazione.
* Utilizzano direttamente direttive del linguaggio.
* Design Pattern:
  + Soluzione generica ad una classe di problemi
  + Propone un modello architetturale.
* Idioma:
  + Soluzione specifica ad un linguaggio
  + Legato alla tecnologia

Immagine che contiene testo, Carattere, schermata, linea

Il contenuto generato dall'IA potrebbe non essere corretto.

**Utilizzare i Design Pattern**:

Come selezionare i DP?

* Considerare come il DP risolve il problema.
* Conoscere le relazioni fra DP
* Considerare le possibili cause di cambiamento dell’architettura
* Considerare cosa può evolvere nell’architettura.

**Come utilizzare un design pattern**?

1. Leggere il DP attentamente
2. Rileggere le sezioni che descrivono Struttura, Partecipanti e Collaborazioni.
3. Analizzare il codice di esempio fornito.
4. Scegliere nomi appropriati per i partecipanti.
5. Definire la struttura delle classi del pattern all’interno della propria applicazione.
6. Scegliere nomi appropriati per le operazioni.
7. Implementare le operazioni

**Anti-Pattern**

Problemi ricorrenti che si incontrano durante lo sviluppo dei programmi e che dovrebbero essere evitati, non affrontati!

Evitare gli errori già commessi

Evidenziare perché una soluzione sbagliata può sembrare in principio corretta per un problema.

Descrivere come passare dalla soluzione sbagliata a quella corretta.