Appunti sui Design Pattern

A.A. 2023/2024

Rosso Carlo

Indice

1 AbstractFactory

Applicabilità:

1. Un sistema deve essere indipendente da come i suoi prodotti sono creati, composti e rappresentati.
2. Un sistema deve essere configurato con una famiglia di prodotti multipli.
3. Una famiglia di prodotti correlati è progettata per essere utilizzata insieme e si desidera far rispettare questa vincolo.
4. Si vuole fornire una libreria di classi per la creazione di prodotti, ma si vuole nascondere le implementazioni concrete.

2 Builder

Applicabilità:

1. L'algoritmo per la creazione di un oggetto complesso deve essere separato e indipendente dalle parti che compongono l'oggetto e da come vengono assemblate.
2. Il processo di costruzione deve permettere diverse rappresentazioni dell'oggetto da costruire.

3 Adapter

Applicabilità:

1. Si desidera utilizzare una classe esistente e la sua interfaccia non corrisponde a quella che si desidera.
2. Si desidera creare una classe riusabile che collabora con classi non correlate o non previste, cioè classi che non necessariamente hanno interfacce compatibili.
3. (solo per oggetti adapter) Si desidera utilizzare diverse sottoclassi esistenti, ma è impraticabile adattare la loro interfaccia per ogni sottoclasse. Un adapter può adattare l'interfaccia della sua classe progenitrice.

4 Decorator

Applicabilità:

1. Aggiungere responsabilità a singoli oggetti dinamicamente e in modo trasparente, cioè senza coinvolegere altri oggetti.
2. Si vuole poter rimuovere responsabilità aggiunte.
3. Quando l'ereditarietà è inapplicabile, ad esempio perché è staticamente definita e non può essere modificata o perché si desidera evitare la creazione di sottoclassi per ogni nuova responsabilità, che potrebbe portare a un'esplosione di classi.

5 Facade

Applicabilità:

1. Si desidera fornire un'interfaccia unificata a un insieme di interfacce in un sottosistema. Facade definisce un'interfaccia di livello più alto che rende il sottosistema più facile da usare.
2. Si desidera disaccoppiare un sottosistema da suoi clienti e da altri sottosistemi, rendendo il sottosistema più indipendente e scalabile.
3. Si desidera raggruppare un insieme di interfacce complesse in un'unica interfaccia più semplice da usare.
4. Si vogliono organizzare i sottosistemi in una struttura a livelli. Il pattern Facade definisce un'interfaccia per ciascun livello del sottosistema.

6 Proxy

Applicabilità:

1. **Accesso remoto**: un proxy remoto fornisce un'interfaccia locale per un oggetto in un sistema remoto.

2. **Proxy virtuale**: un proxy può creare un oggetto complesso solo quando viene richiesto.

3. **Proxy di protezione**: un proxy controlla l'accesso all'oggetto originale. Ad esempio, un proxy potrebbe verificare che il chiamante abbia i permessi necessari per eseguire un'operazione, oppure implementa un semaforo per controllare l'accesso concorrente.

4. **Riferimento intelligente**: un proxy può gestire il ciclo di vita di un oggetto originale. Ad esempio, un proxy può tenere il conto dei riferimenti ad un oggetto e lo elimina quando il conto raggiunge zero.

7 Command

Applicabilità:

1. Parametrizzare gli oggetti rispetto ad un'operazione da eseguire.

2. Definire, eseguire e accodare richieste in un oggetto. In particolare permette di eseguire operazioni indipendenti in modo asincrono.

3. Supportare operazioni annullabili. Il metodo `execute` può memorizzare lo stato necessario per annullare i suoi effetti nel comando stesso. I comandi che sono stati eseguiti tramite `execute` devono essere memorizzati in uno storico. In questo modo è possibile annullare i comandi eseguiti.

4. Organizzare un sistema in operazioni ad alto livello costruite su operazioni a basso livello. Per esempio, il Command pattern è una soluzione per la modellazione di transazioni.

8 Iterator

Applicabilità:

1. Accedere agli elementi di un oggetto aggregato senza esporre la sua rappresentazione sottostante.

2. Supportare più attraversamenti in modo uniforme.

3. Fornire un'interfaccia uniforme per attraversare diversi tipi di aggregati.

9 Observer

Applicabilità:

1. Un'astrazione presenta due aspetti, di cui uno dipende dall'altro. Incapsulando questi aspetti in oggetti separati, è possibile riusarli indipendentemente.

2. Un cambiamento di stato in un oggetto richiede modifiche in altri oggetti dipendenti, ma non si conosce il numero di oggetti dipendenti.

3. Un oggetto deve notificare altri oggetti senza fare ipotesi su chi siano questi oggetti. In altre parole, si vuole mantenere un alto livello di disaccoppiamento.

10 Strategy

Applicabilità:

1. Molte classi correlate differiscono fra loro solo per il comportamento. Strategy fornisce un modo per configurare una classe con un comportamento scelto fra tanti.

2. Sono necessarie più varianti di un algoritmo. Lo Strategy pattern può essere usato quando questi algoritmi sono implementati come classi separate che possono essere selezionate in fase di esecuzione.

3. Un algoritmo usa una struttura dati che non dovrebbe essere resa nota ai client. Il pattern Strategy può essere usato per evitare di esporre strutture dati complesse e specifiche dell'algoritmo.

4. Una classe definisce molti comportamenti che compaiono all'interno di scelte condizionali multiple. Al posto di molte scelte condizionali, si suggerisce di spostare i blocchi di codice correlati in una classe Strategy dedicata.

11 Template Method

Applicabilità:

1. Per implementare la parte invariante di un algoritmo e lasciare alle sottoclassi la possibilità di implementare la parte variabile.

2. Un comportamento comune fra sottoclassi deve essere portato a fattore comune e localizzato in una classe comune per evitare codice duplicato.

3. Controllare le estensioni di sottoclassi. Permette di definire metodi template che chiamano metodi hook in punti specifici, permettendo quindi alle sottoclassi di estendere solo quei punti.