CORSO JAVA AESYS
Parte Prima

2022

www.aesystech.it

DESIGN PATTERNS

- Definizione
- Classificazione
- Singleton Pattern

Definizione

I Design Pattern rappresentano soluzioni di progettazione generiche applicabili a problemi ricorrenti all'interno di contesti eterogenei. Consapevoli che l'asserzione precedente potrebbe non risultare chiara al lettore che non ha familiarità con determinate situazioni, cercheremo di descrivere il concetto presentando, oltre che la teoria, anche alcuni esempi di pattern all'interno del libro.
In questo modo si potranno meglio apprezzare sia i concetti sia l'applicabilità. L'idea di base, però, è piuttosto semplice. È risaputo che la bontà della progettazione è direttamente proporzionale all'esperienza del progettista. Un progettista esperto risolve i problemi che si presentano utilizzando soluzioni che in passato hanno già dato buoni risultati. La GoF non ha fatto altro che confrontare la propria (ampia) esperienza nel trovare soluzioni progettuali, scoprendo così alcune intersezioni evidenti. Siccome queste intersezioni sono anche soluzioni che risolvono spesso problemi in contesti eterogenei, possono essere dichiarate e formalizzate come Design Pattern. In questo modo si mettono a disposizione soluzioni a problemi comuni, anche ai progettisti che non hanno un'esperienza ampia come quella della GoF

Classificazione

❑Classificazione dei Pattern (GoF)

❑Pattern di Creazione:
- Factory
- Abstract Factory
- Singleton
- Builder

❑Pattern Strutturali
- Adapter
- Bridge
- Facade
- Decorator
- Composite
- Flyweight

❑ Pattern comportamentali
- Observer
- Strategy
- Command
- Template
- State
- Memento
- Interpreter
- Iterator
- Mediator

Pattern - Criteri di classificazione

❑I pattern sono distinguibili in base al diverso scopo (purpose):

- **Pattern di creazione**: si applicano al processo di creazione degli oggetti;
- **Pattern strutturali**: trattano la composizione di oggetti;
- **Pattern comportamentali**: modellano le modalità di mutua interazione e la distribuzione di responsabilità tra le classi.

❑Una ulteriore dimensione di classificazione è la portata (scope):

- **Classe**: pattern che trattano le relazioni tra classi e sottoclassi;
- **Oggetto**: relazioni dinamiche tra oggetti.

AESYS
colors your business

Pattern di Creazione

- **Factory**: definisce una classe che crea una delle possibili sottoclassi di una classe astratta di base in funzione dei dati forniti.

- **Abstract Factory**: fornisce un'interfaccia per creare famiglie di oggetti correlati senza specificare le loro classi concrete.

- **Singleton**: assicura che una classe abbia una sola istanza e ne fornisce un punto globale di accesso.

- **Builder**: consente di separare la procedura di costruzione di un oggetto complesso dalla sua rappresentazione, in modo che tale procedura possa creare più rappresentazioni differenti del medesimo oggetto.

AESYS
colors your business

Pattern Strutturali

- **Adapter**: converte l'interfaccia di una classe in un'altra interfaccia che si attende un certo *client*; consente l'interazione tra classi che non potrebbero altrimenti cooperare in quanto hanno interfacce incompatibili.
- **Bridge**: separa un'astrazione dalle sue implementazioni in modo da consentire ad entrambe di cambiare in modo indipendente.
- **Composite**: compone oggetti in strutture ad albero che rappresentano gerarchie di tipo parte-per-il-tutto. Consente di trattare singoli oggetti e composizioni di oggetti in modo uniforme
- **Flyweight**: utilizza la condivisione per gestire in modo efficiente grandi numeri di oggetti
- **Facade**: Fornisce un'interfaccia comune ad un insieme di interfacce in un sottosistema. Facade definisce un'interfaccia di alto livello che rende un sottosistema più facile da utilizzare.
- **Decorator:** Aggiunge dinamicamente responsabilità addizionali ad un oggetto. In questo modo si possono estendere le funzionalità d'oggetti particolari senza coinvolgere classi complete.

AESYS
colors your business

Pattern comportamentali

- **Observer**: definisce una dipendenza uno-a-molti tra oggetti in modo che quando un oggetto cambia stato, tutti gli oggetti che dipendono da esso ricevono una notifica e si aggiornano.
- **Strategy**: definisce una famiglia di algoritmi, incapsula ciascuno di essi in una classe opportuna e li rende intercambiabili. Strategy fa sì che gli algoritmi possano variare in modo indipendente dal codice che li utilizza.
- **Command**: incapsula le richieste di servizio in oggetti, consentendo di controllarne la selezione e la sequenza di attivazione, l'accodamento, l'annullamento, ecc.
- **Interpreter**: Dato un linguaggio, definisce una rappresentazione della sua grammatica e di un interprete che usa tale rappresentazione per valutare le frasi in quel linguaggio.
- **Iterator**: Fornisce un punto di accesso sequenziale agli elementi di un oggetto aggregato, senza rendere pubblica la sua organizzazione interna
- **Template**: Definisce lo scheletro di un algoritmo in un'operazione, delegando alcuni passi alle sottoclassi; in pratica, consente alle sottoclassi di ridefinire alcuni passi di un algoritmo senza modificarne la struttura
- **Mediator**: Definisce un oggetto che incapsula la logica di interazione di un insieme di oggetti; favorisce il disaccoppiamento evitando agli oggetti di riferirsi esplicitamente l'uno l'altro e consentendo così di variare in modo indipendente le loro interazioni.

AESYS
colors your business

Factory Pattern

Descrizione:

❑Definisce una classe che decide quale implementazione di una classe astratta restituire, effettuando la scelta sulla base dei dati forniti in input al factoryMethod.

❑Viene usato per delegare ad un Factory la responsabilità della creazione e della 'conoscenza' della reali implementazioni di una determinata interfaccia (o classe astratta). Gli oggetti così creati verranno trattati dal client utilizzando la loro interfaccia comune rendendolo così indipendente dalle implementazioni.

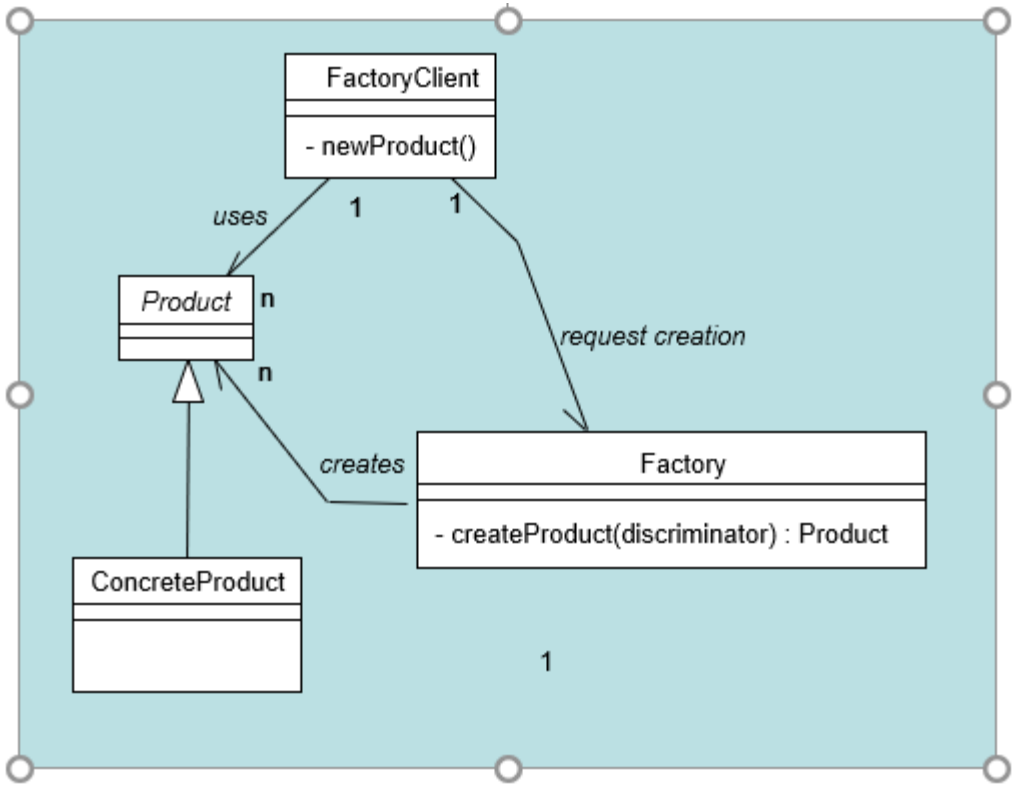
❑Problema: Gestione Documenti

Si consideri lo sviluppo di un'applicazione (framework) per eseguire operazioni comuni quali creazione, apertura, modifica su documenti di diverso tipo.

- La logica di realizzazione delle operazioni varia a seconda del tipo di documento, perciò esse sono delegate all'oggetto Documento specifico.
- Si crea una gerarchia di documenti che ereditano da una classe Documento di base che mette a fattore comune le definizioni delle operazioni.
- Si evita così che l'applicazione debba conoscere tutti i documenti supportati delegando la selezione del giusto documento ad una classe (Factory) che, in base ad un criterio impostato, sa quale oggetto creare.

AESYS
colors your business

Factory Pattern



AESYS
colors your business

Factory Pattern

- Product: superclasse astratta di oggetti prodotti dal pattern Factory (nell'es. si potrebbe chiamare Document);
- Concrete Product: una qualunque classe concreta istanzabile dagli oggetti che partecipano al pattern; se tali classi non hanno una logica comune, Product è un interfaccia, non una classe astratta (es. PDFDocument, TextDocument, …).
- Factory Client: classe indipendente dal contesto, che deve creare oggetti specifici del contesto attraverso l'uso di una classe Factory (FrameworkApplication)
- Factory: classe specifica del contesto che implementa il metodo per la creazione dell'oggetto specifico ConcreteProduct (DocumentFactory)

AESYS
colors your business

Factory Pattern

❑ Conseguenze:

- La classe che richiede la creazione è indipendente dalla classe degli oggetti concreti che vengono creati.
- L'insieme delle classi Product che possono essere istanziate può variare indipendentemente dalla classe che utilizza il pattern.

❑ Usi comuni:

- Quando una classe (client) non può prevedere con esattezza quali e quante classi di oggetti dovrà creare;
- Quando una classe usa le proprie sottoclassi per specificare quali oggetti creare;
- Quando si vuole localizzare la conoscenza delle classi da istanziare.

AESYS
colors your business

Singleton Pattern

- **Descrizione:**

  Definisce una classe della quale è possibile la istanziazione di un unico oggetto, tramite l'invocazione di un metodo della classe, incaricato della produzione degli oggetti. Le successive richieste di istanziazione, comportano la restituzione di un riferimento allo stesso oggetto.

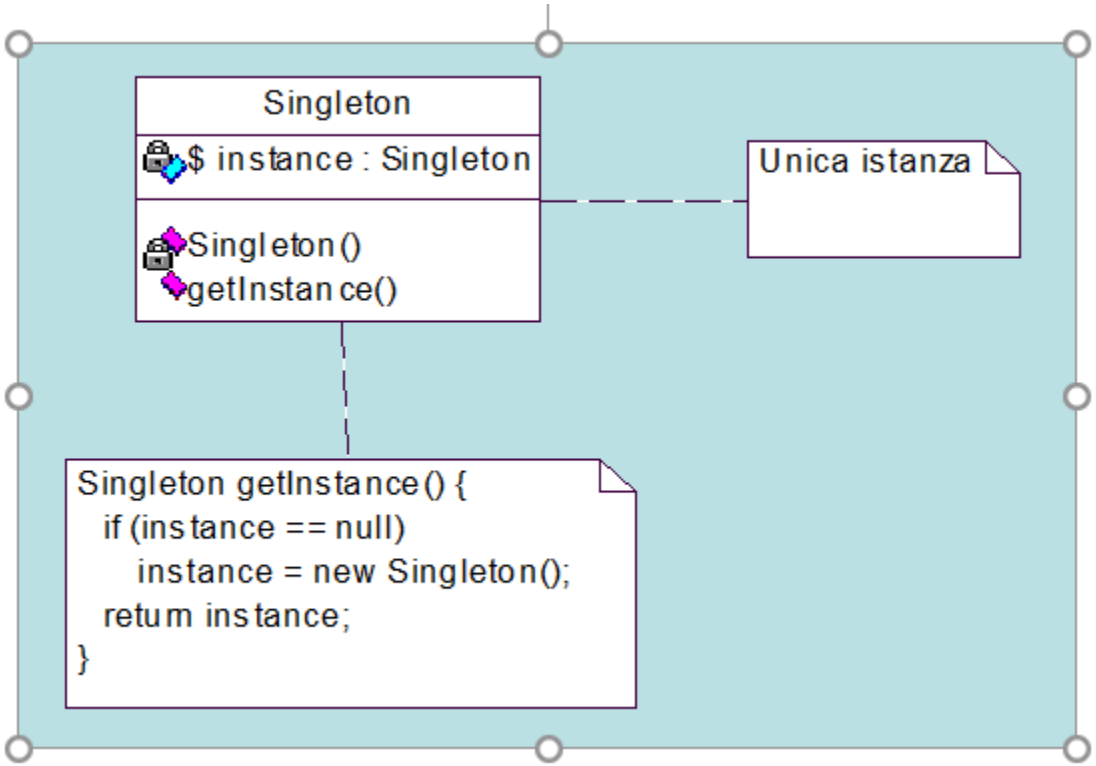
- **Esempio:**

Gestore di risorse condivise:

Un gestore di risorse è modellabile come una classe che può avere una sola istanza che, in genere, contiene internamente la risorsa che gestisce.
La classe gestore deve avere una sola istanza condivisa tra tutti gli utenti, in modo che tutte le richieste di uso passino da un unico oggetto centralizzato.

AESYS
colors your business

Singleton Pattern



AESYS
colors your business

Singleton Pattern

❑**Struttura**:
La classe contiene una variabile statica che fa riferimento all'unica istanza della classe che si intende utilizzare
L'istanza viene creata unicamente quando serve, cioè quando un client invoca per la prima volta il metodo pubblico statico getInstance; se l'istanza è nulla viene creata, altrimenti viene restituita l'istanza precedentemente creata.
Per evitare che un client possa creare una seconda istanza di Singleton i costruttori di tale classe devono essere dichiarati private.

❑**Conseguenze:**

- Garantisce l'esistenza di un'unica istanza di Singleton
- Le classi che si riferiscono a Singleton devono utilizzare il metodo getInstance fornito dalla classe piuttosto che costruire l'istanza direttamente
- Questa implementazione di Singleton in Java non è specializzabile, poiché richiederebbe di dichiarare un costruttore non privato e fare in modo di sovrascrivere il metodo statico getInstance dalle classi che ereditano (illegale per il linguaggio).

AESYS
colors your business