Singleton

E’ un pattern creazionale che consente di garantire che una classe abbia una sola istanza, fornendo al contempo un punto di accesso globale a tale istanza.

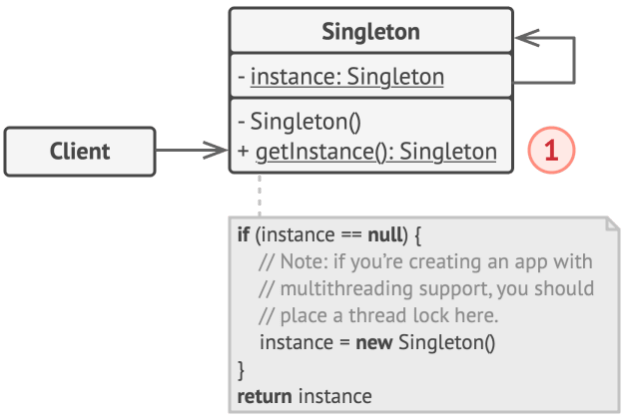
Il grande problema è che il singleton risolve due problemi contemporaneamente, violando il principio di singola responsabilità:

- Assicurarsi che una classe abbia una sola istanza: Nel caso, ad esempio, in cui volessimo controllare l’accesso a una risorsa condivisa (database, file ecc...);
- Fornisce un punto di accesso globale a quell’istanza: Consente di accedere a un oggetto da qualsiasi punto del programma. Tuttavia, protegge quell’istanza dalla sovrascrittura da parte di altro codice.

Tutte le implementazioni del singleton hanno questi due passaggi in comune:

- Rendere il costruttore privato per impedire ad altri oggetti di usare l’operatore new;
- Crea un metodo di creazione statico che funga da costruttore. Tale metodo chiama il costruttore privato per creare un oggetto e salvarlo in un campo statico. Tutte le chiamate successive a questo metodo restituisce l’oggetto memorizzato nella cache.

Struttura



- La classe Singleton dichiara il metodo statico getInstance che restituisce la stessa istanza della propria classe. Il costruttore di Singleton dovrebbe essere nascosto al codice client. L'oggetto Singleton è accessibile solo tramite il metodo getInstance.

Applicabilità

- Usare il modello Singleton quando una classe deve avere una sola istanza disponibile per tutti i client (ad esempio un singolo oggetto del DB da diverse parti del programma);
- Usare il modello Singleton quando è necessario un controllo più rigoroso sulle variabili globali.

Come implementare?

- Aggiungere un campo statico privato alla classe per memorizzare l'istanza Singleton;
- Dichiarare un metodo di creazione statico pubblico per ottenere l'istanza Singleton;
- Implementare l'inizializzazione lazy all'interno del metodo statico. Dovrebbe creare un nuovo oggetto alla prima chiamata e inserirlo nel campo statico. Il metodo deve restituire sempre quell'istanza a tutte le chiamate successive;
- Rendi privato il costruttore della classe;
- Sostituire tutte le chiamate dirette al costruttore del Singleton con chiamate al suo metodo di creazione statico;

Vantaggi

- Certezza che una classe abbia una sola istanza;
- Ottieni un punto di accesso globale a quell'istanza;
- L'oggetto Singleton viene inizializzato solo quando viene richiesto per la prima volta;

Svantaggi

- Viola il principio di singola responsabilità (come detto sopra);
- Tale modello può mascherare una cattiva progettazione;
- il modello richiede un trattamento speciale in un ambiente multithread, in modo che più thread non creino più volte un oggetto Singleton;

Relazione con altri modelli

- Una classe Facade può spesso essere trasformata in un Singleton poichè nella maggior parte dei casi è sufficiente un singolo oggetto Facade;
- Flyweight assomiglierebbe a Singleton se in qualche modo si riuscisse a ridurre tutti gli stati condivisi degli oggetti a un solo oggetto Flyweight. Ma ci sono due differenze fondamentali tra questi modelli:

1. Dovrebbe esserci una sola istanza Singleton, mentre una classe Flyweight può avere più istanze con stati intrinseci diversi;
2. L'oggetto Singleton può essere mutabile, mentre gli oggetti Flyweight sono immutabili;

- Le Abstract Factory, Builder e Prototype possono essere tutti implementati come Singleton.