

Modern Patterns

Introduzione

Obiettivi:

- Comprendere come attuare propriamente pattern moderni.

Null Object (Modern Pattern)

Problema

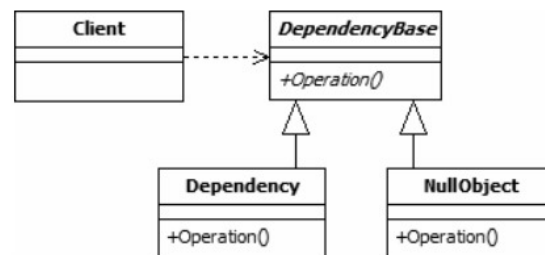
Come evitare che venga restituito un valore "null" quando viene richiesto un riferimento ad un'interfaccia?

Soluzione

Si utilizza un oggetto che implementa l'interfaccia attesa, ma in cui il corpo del metodo è vuoto

Null Object permette di semplificare l'utilizzo delle dipendenze che possono essere indefinite. Ciò si ottiene utilizzando istanze di una classe concreta che implementa un'interfaccia nota, invece di riferimenti null. In primo luogo, viene realizzata una classe astratta contenente diverse operazioni da eseguire, classi concrete che estendono questa classe e infine una classe di oggetti null che fornisce un'implementazione *do-nothing*, la quale verrà utilizzata senza problemi dove potrebbe essere riscontrato il valore null.

Caso di studio



Si analizzano le entità che rappresentano la raffigurazione, in cui:

- *Client*, ha una dipendenza che può essere richiesta o meno. Laddove non sia richiesta alcuna funzionalità nella dipendenza, eseguirà i metodi di un oggetto null
- *DependencyBase*, superclasse per le varie dipendenze disponibili che il Client può utilizzare. Laddove la superclasse non fornisce funzionalità condivise, può essere sostituita con un'interfaccia
- *Dependency*, dipendenza funzionale che può essere utilizzata dal Client
- *NullObject*, può essere utilizzata come dipendenza dal Client. Non contiene funzionalità ma implementa tutti i membri definiti dalla classe astratta *DependencyBase*.

Concludendo, il design pattern Null Object risulta funzionare nell'evitare problematiche relative all'eccezione *NullPointerException*, al tempo stesso però non sarà possibile controllare la consistenza di un oggetto.

Dependency Injection (Modern Pattern)

Problema

Come creare un'istanza di un'entità software senza che sia specificata la classe?

Tipicamente tale problematica avviene qualora classi di alto livello richiedano comportamenti sviluppati in classi di basso livello. Rispetto a capitoli precedenti, sono stati introdotti approcci capaci di rispondere attivamente alla domanda, come la *sovrapposizione* di layer *astratti* tra le due entità, affinché risultino totalmente isolate, oppure attraverso l'impiego di *pattern creazionali*, come *Factory Method*, in cui è delegata la responsabilità a termini software dediti solamente alla concretizzazione di istanze che saranno successivamente restituite al richiedente; tuttavia esiste un meccanismo migliorativo in grado di eliminare qualsiasi incertezza relativa ad errate gestioni di dipendenze, definita **Dependency Injection**.

Soluzione

Dependency Injection crea un grafo di dipendenze che adotti l'inversione di controllo per individuare gli oggetti associati, in maniera tale che le dipendenze siano immesse dal nucleo verso i nodi esterni.

Il *pattern* descritto è in grado di stabilire concretamente la rottura delle dipendenze tra *LLC* e *HLC*, contrariamente alle soluzioni precedenti, ossia mediante l'implementazione della sintassi *new(...)* oppure tramite *factories*, attuando la delega comportamentale.