Metodologie di Programmazione

Relazione Progetto

Pierpaolo D’Angelo

Questo progetto si propone di modellare la base di uno shop online, all’interno del quale sono rappresentati dei prodotti, che possono essere o degli articoli o dei raggruppamenti eventualmente scontati di altri prodotti.

Sono presenti inoltre una serie di meccanismi che offrono la possibilità di manipolare e organizzare insiemi di prodotti, ad esempio:

* Monitorare il prezzo di un prodotto nel tempo.
* Monitorare il numero di prodotti di un certo tipo nel tempo.
* Applicare uno sconto aggiuntivo se il numero di articoli di un certo tipo è superiore a una certa soglia.
* Estrarre il prezzo maggiore e minore dei prodotti di un determinato tipo.

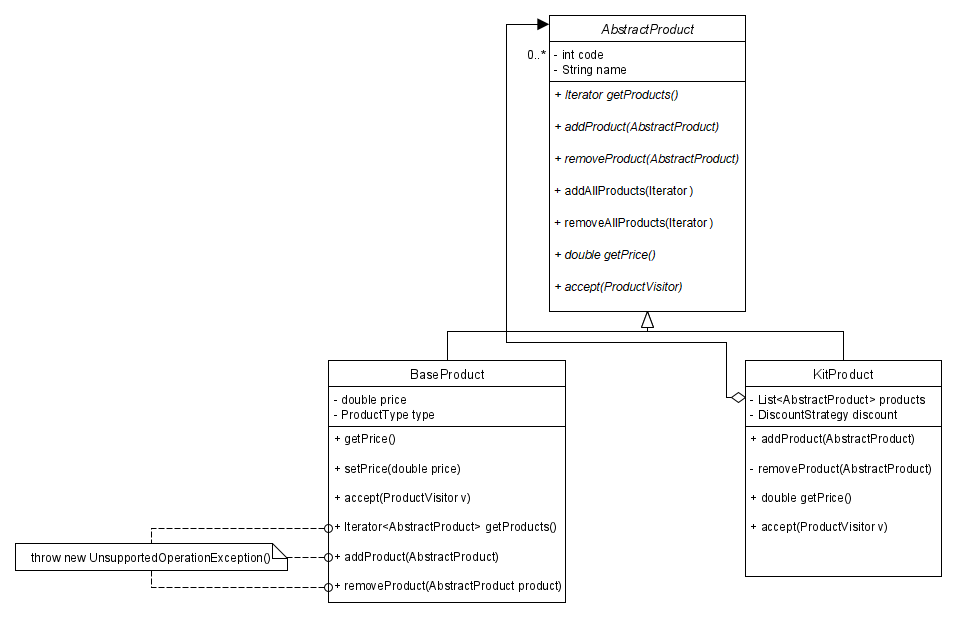
Scelte implementative

Composite:

Per la modellazione della gerarchia dei prodotti è stato scelto di utilizzare un Composite, nella sua variante Design for Uniformity. In particolare, la classe astratta AbstractProduct presenta un campo codice ed un campo nome ed espone i getters e i metodi per manipolare una lista di AbstractProduct. I metodi AddAllProducts e RemoveAllProduct sono dei template methods, poiché il loro algoritmo è definito ma si basa sull’aggiunta e la rimozione singola che invece vengono lasciate da implementare alle sottoclassi.

La classe BaseProduct consiste in una classe leaf, aggiunge i campi relativi a prezzo e tipo, e ridefinisce i metodi AddProduct, RemoveProduct e GetProducts, i quali, rappresentando operazioni non supportate dal BaseProduct sollevano un eccezione di tipo UnsupportedOperationException. Viene inoltre ridefinitio il metodo getPrice, che qui restituisce il valore mantenuto dal campo price e viene aggiunto un metodo per impostare il prezzo.

La classe KitProduct consiste in una classe composite, aggiunge due campi, una DiscountStrategy ed una Lista di AbstractProduct. Vengono ridefiniti i metodi addProduct e removeProduct, i quali aggiungono e rimuovono un prodotto dalla lista e getProducts che restituisce l’iterator di quest’ultima. Il prezzo in questa classe non è un campo, ma viene calcolato dal metodo getPrice() sommando i prezzi di tutti gli AbstractProduct presenti nella lista dei prodotti e applicando lo sconto specificato dalla DiscountStrategy.



Visitor:

Per poter manipolare un insieme di prodotti senza preoccuparsi della differenza tra BaseProduct e KitProduct, viene utilizzato un Visitor.

L’interfaccia ProductVisitor descrive due metodi visit in overload, i quali accettano come parametro rispettivamente un BaseProduct o un KitProduct.

Le implementazioni concrete di ProductVisitor sono le seguenti:

* DiscountProductVisitor: questo Visitor si occupa di decidere se uno sconto deve essere applicato o meno a seconda di quanti prodotti di uno stesso tipo sono contenuti all’interno di un AbstractProduct. Se al metodo visit viene passato un BaseProduct e questo è del tipo interessato, il contatore inizializzato in costruzione viene decrementato, altrimenti, viene invocato l’accept su ogni elemento del KitProduct. Il metodo getDiscountStrategy restituisce la DiscountStrategy passata in costruzione se si è raggiunto il numero di prodotti specificato, altrimenti la DiscountStrategy.NO\_DISCOUNT (che non applica nessuno sconto).
* HiLoPriceVisitor: questo Visitor si occupa, dato un determinato tipo, di determinare il prezzo più alto e più basso dei prodotti contenuti in un AbstractProduct. Se al metodo visit viene passato un BaseProduct del tipo interessato e il prezzo corrisponde al massimo o al minimo incontrato, la rispettiva variabile di istanza viene aggiornata, altrimenti, viene invocato l’accept su ogni elemento del KitProduct.
* TypeCounterVisitor: questo Visitor si occupa di contare il numero di prodotti di un determinato tipo all’interno di un AbstractProduct. Se al metodo visit viene passato un BaseProduct del tipo interessato il contatore viene incrementato, altrimenti, viene invocato l’accept su ogni elemento del KitProduct.