RELAZIONE DEL PROGETTO DI METODOLOGIE DI PROGRAMMAZIONE

A.A. 2016/2017

Autori:

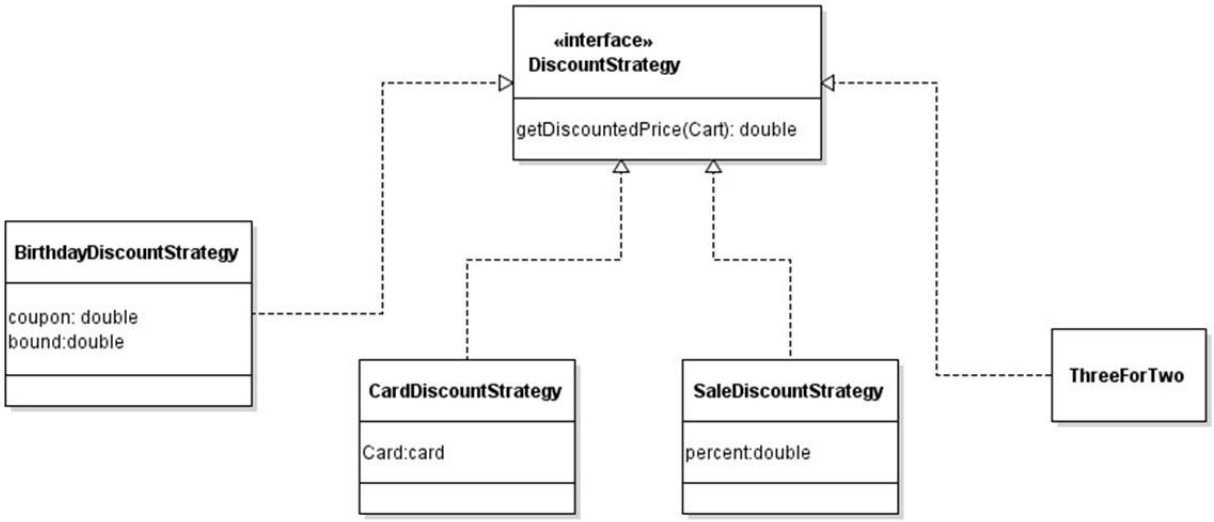
Levi Camillo

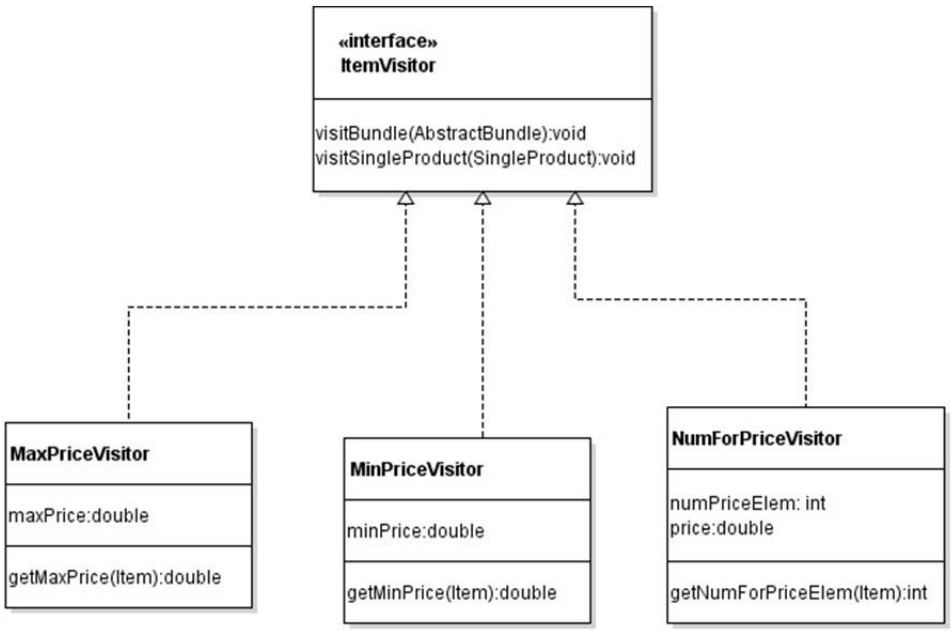
➢ DESCRIZIONE DELL'ESERCIZIO

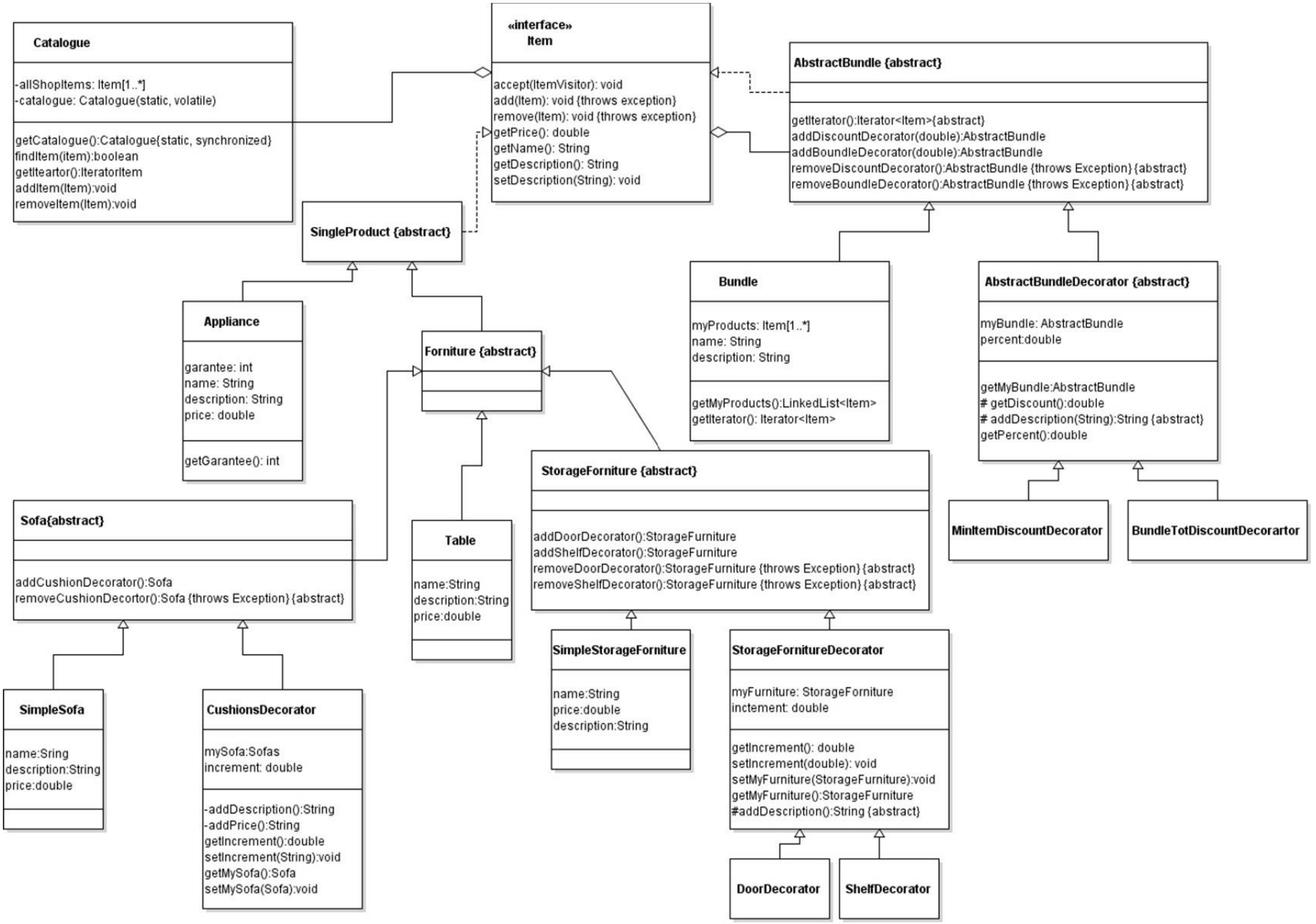
L'esercizio consiste nell'implementazione di uno *Shop Online* che vende articoli per la casa come oggetti singoli o come pacchetti di oggetti. Più specificatamente esso vende due categorie di oggetti: elettrodomestici o articoli di arredamento. In particolare per gli arredamenti si è deciso di specializzarlo sui tavoli, sui divani (a cui si possono aggiungere o rimuovere dei cuscini) e sui mobili (a cui si possono aggiungere o rimuovere vari ripiani e delle ante). Per favorire la vendita di pacchetti di oggetti si sono previste due tipologie di sconto da applicare esclusivamente in questo ultimo caso: la prima prevede di scontare di una data percentuale l'intero pacchetto, l'altra di scontare solamente l'articolo del pacchetto che costa di meno. Viene inoltre messo a disposizione del compratore un catalogo di prodotti presenti nel negozio in cui egli può cercare un articolo specifico. Sono disponibili anche altre tipologie di sconto, che vengono applicate sulla spesa totale per esempio quando è il compleanno dell'utente, in questo caso se ha speso più di una certa soglia vengono scalati alcuni euro dal totale, oppure in periodo di saldi quando si vuole applicare una percentuale di sconto, ed è anche previsto uno sconto 3x2 (nel caso si comprino tre oggetti dello stesso tipo il terzo viene regalato). *Shop Online* prevede anche una carta personale per i suoi clienti

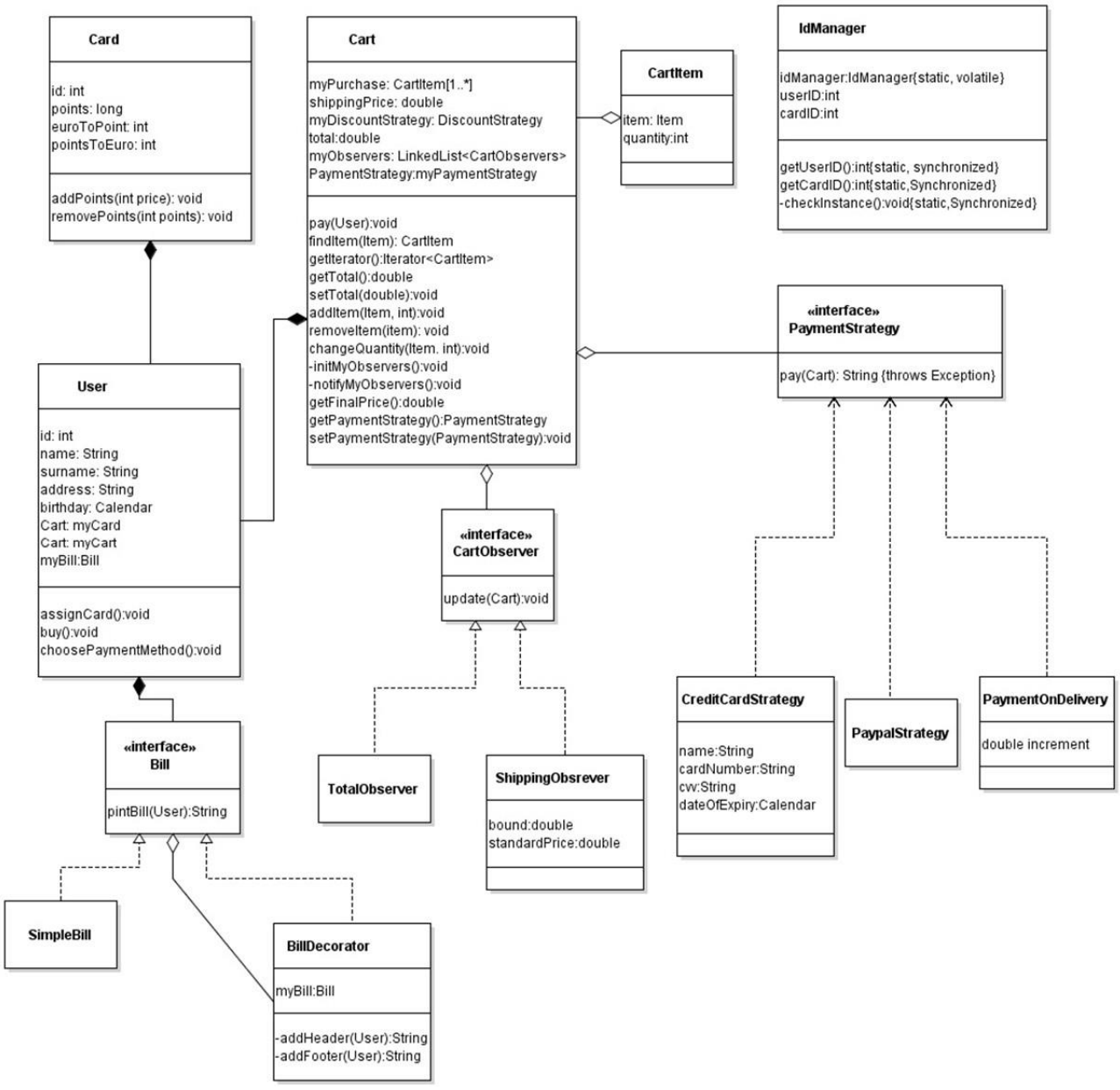
che contiene dei punti, che vengono inseriti al momento dell'acquisto in relazione all'importo speso. Quindi se il compratore possiede la carta del negozio, nel caso in cui il totale che deve pagare sia maggiore o uguale del valore in euro della carta, gli è permesso di convertire i punti presenti nella carta in denaro e di scalarli dal totale che deve pagare. Ad ogni compratore è associato un carrello, al quale può aggiungere o togliere prodotti e di cui può controllare in qualsiasi momento il prezzo totale. Si è anche stabilito che se il prezzo totale speso supera una certa soglia, decisa magari in relazione al periodo dell'anno, la spesa di spedizione viene azzerata. Al compratore è data la possibilità di scegliere la tipologia di pagamento tra: pagamento tramite PayPal, tramite carta di credito oppure *on delivery* , e al momento del pagamento viene anche stampata una fattura con l'importo pagato preceduto dal nome del negozio e seguito da una formula di ringraziamento. Infine viene data la possibilità di fare dei report sul negozio, in particolare di cercare il prezzo massimo o minimo dei prodotti disponibili nello *Shop Online*, oppure dato un prezzo di sapere il numero di prodotti che hanno quel prezzo.

- **DIAGRAMMA UML**


«interface»
DiscountStrategy
getDiscountedPrice(Cart): double
BirthdayDiscountStrategy
coupon: double
bound:double
CardDiscountStrategy
Card:card
SaleDiscountStrategy
percent:double
ThreeForTwo


«interface»
ItemVisitor
visitBundle(AbstractBundle):void
visitSingleProduct(SingleProduct):void
MaxPriceVisitor
maxPrice:double
getMaxPrice(Item):double
MinPriceVisitor
minPrice:double
getMinPrice(Item):double
NumForPriceVisitor
numPriceElem: int
price:double
getNumForPriceElem(Item):int


Catalogue
-allShopItems: Item[1..*]
-catalogue: Catalogue(static, volatile)
getCatalogue():Catalogue{static, synchronized}
findItem(item):boolean
getIteartor():IteratorItem
addItem(Item):void
removeItem(Item):void
«interface»
Item
accept(ItemVisitor): void
add(Item): void {throws exception}
remove(Item): void {throws exception}
getPrice(): double
getName(): String
getDescription(): String
setDescription(String): void
AbstractBundle {abstract}
getIterator():Iterator<Item>{abstract}
addDiscountDecorator(double):AbstractBundle
addBoundleDecorator(double):AbstractBundle
removeDiscountDecorator():AbstractBundle {throws Exception}{abstract}
removeBoundleDecorator():AbstractBundle {throws Exception}{abstract}
SingleProduct {abstract}
Bundle
myProducts: Item[1..*]
name: String
description: String
getMyProducts():LinkedList<Item>
getIterator(): Iterator<Item>
AbstractBundleDecorator {abstract}
myBundle: AbstractBundle
percent:double
getMyBundle:AbstractBundle
getDiscount():double
addDescription(String):String {abstract}
getPercent():double
Appliance
garantee: int
name: String
description: String
price: double
getGarantee(): int
Forniture {abstract}
StorageForniture {abstract}
addDoorDecorator():StorageFurniture
addShelfDecorator():StorageFurniture
removeDoorDecorator():StorageFurniture {throws Exception}{abstract}
removeShelfDecorator():StorageFurniture {throws Exception}{abstract}
MinItemDiscountDecorator
BundleTotDiscountDecorartor
Sofa{abstract}
addCushionDecorator():Sofa
removeCushionDecortor():Sofa {throws Exception}{abstract}
Table
name:String
description:String
price:double
SimpleStorageForniture
name:String
price:double
description:String
StorageFornitureDecorator
myFurniture: StorageForniture
inctement: double
getIncrement(): double
setIncrement(double): void
setMyFurniture(StorageFurniture):void
getMyFurniture():StorageFurniture
#addDescription():String {abstract}
SimpleSofa
name:Sring
description:String
price:double
CushionsDecorator
mySofa:Sofas
increment: double
-addDescription():String
-addPrice():String
getIncrement():double
setIncrement(String):void
getMySofa():Sofa
setMySofa(Sofa):void
DoorDecorator
ShelfDecorator



➢ MOTIVAZIONI DEI DESIGN PATTERN USATI

1. COMPOSITE

   Per quanto riguarda la gerarchia degli articoli venduti si è utilizzato il Design Pattern Composite, per poter inserire nel catalogo, come già detto precedentemente, anche pacchetti di più articoli. Sotto l'interfaccia *Item* troviamo quindi la suddivisione *SingleProduct* e *AbstractBundle*, che rappresenta i pacchetti composti da altri *Item*.

2. VISITOR

   Per realizzare i report si è predisposta una gerarchia di Visitor che opera specificatamente sulla gerarchia di *Item*. Per ognuna delle tre tipologie di report sopra citate esiste quindi un Visitor concreto a cui viene delegato il compito.

3. DECORATOR

   Sia sui singoli prodotti che sui pacchetti sono stati applicati dei Design Pattern Decorator. Più in particolare, sui pacchetti sono stati applicati due decoratori concreti che applicano gli sconti sopra descritti, mediante una percentuale eventualmente modificabile. Si noti che il decoratore che sconta soltanto il prodotto meno costoso fa uso del Visitor che cerca l'oggetto con prezzo minore.
   Sui prodotti singoli troviamo invece diversi decoratori, a seconda della categoria:

   - gli elettrodomestici e i tavoli non sono decorabili.
   - sui divani troviamo un unico decoratore concreto per l'aggiunta dei cuscini.
   - sui mobili ai quali si possono aggiungere ante e ripiani troviamo un decoratore concreto per l'aggiunta delle prime e uno per l'aggiunta dei secondi.

   Ogni decoratore aggiunge proprietà alla propria componente, ossia il prodotto che decora, come ad esempio arricchire la descrizione dell'articolo o incrementare il prezzo di un dato importo, eventualmente modificabile.
   Gli oggetti decorabili mettono a disposizione le funzionalità di aggiunta e di rimozione di decoratori.

   Un'altra applicazione del Pattern Decorator è presente sulla fattura, associata all'utente, tramite un solo decoratore che aggiunge *header* e *footer* alla stampa.

4. TEMPLATE

   In alcuni casi è stato utile applicare il Design Pattern Template, come ad esempio nel metodo che restituisce la descrizione dei mobili decorati con ante e sportelli. Infatti nel decoratore astratto troviamo un metodo Template che lascia astratta una parte dell'algoritmo, richiamando un metodo appunto astratto, definito nelle due classi concrete figlie, che arricchiscono la descrizione con l'informazione dell'aggiunta di ante oppure di sportelli. Stessa cosa succede nei decoratori dei pacchetti di articoli, sia con prezzo che con descrizione.

5. STRATEGY

   Per quanto riguarda gli sconti sulla spesa totale, cumulabili quindi con gli sconti sui pacchetti, ma non cumulabili tra di loro, è stata predisposta una gerarchia di strategie di sconto applicabili al totale del carrello, settando il campo relativo proprio del carrello. I quattro tipi di sconti, già descritti nel paragrafo precedente, corrispondono ognuno ad uno Strategy concreto, che va a modificare l'importo, mediante l'operazione *getFinalPrice()* richiamata sul carrello, che a sua volta interroga la propria componente.

   Anche il pagamento è stato implementato con uno Strategy, ancora una volta associato al carrello, con il campo relativo, e richiesto in fase di acquisto all'utente.

6. OBSERVER

   Il carrello è anche dotato di una lista di Observer, di due diversi tipi concreti, che si occupano uno dell'aggiornamento del totale e l'altro della spesa di spedizione, al momento dell'aggiunta, della rimozione, o della modifica della quantità dei prodotti nel carrello. Infatti da ognuna di queste operazioni, richieste dall'utente, viene richiamata la funzione che si occupa di notificare tutti gli osservatori della lista. Dall'osservatore del totale viene ricalcolato ogni volta l'importo, mentre dall'osservatore della spesa di spedizione, viene settato tale prezzo a zero se il totale supera una certa soglia, e viene rimesso al prezzo standard se invece il totale scende al di sotto di essa.

7. SINGLETON

   Il Design Pattern Singleton è stato utilizzato in due casi: per la realizzazione del catalogo dei prodotti e per l'assegnatore di ID ad utente e carta.
   Per quanto riguarda il catalogo, la sua lista di articoli deve essere la stessa per ogni utente. L'aggiunta e la rimozione di prodotti in vendita, da parte del negozio, saranno quindi visibili allo stesso modo da ogni utente.
   L'assegnatore di ID invece è un Singleton che tiene traccia, tramite due campi interi, degli ID assegnati ai nuovi utenti e alle nuove carte ad essi associate. Ogni volta che viene creato un nuovo utente, o una nuova carta, il campo ID relativo viene incrementato e assegnato, ma ovviamente per tenere traccia degli ID utilizzati e per non assegnare ID identici, questi campi devono essere unici e in comune per tutti.

➢ IMPLEMENTAZIONE E TEST

Il codice è stato suddiviso in pacchetti:

- *client*, contenente le classi dell'utente, del carrello, della carta e della fattura.

- *discounts*, contenente le strategie di sconto.
- *items*, con la gerarchia dei prodotti.
- *management,* con i Singleton *Catalogue* e *idManager*.
- *observers*, con gli osservatori del carrello.
- *payments,* contenente le strategie di pagamento. - *visitors,* con i Visitor degli *Item*.

Il pacchetto sorgente *tests* contiene le classi di test degli *Item*, divise per categorie e inclusive dei test dei Decorator, dei Visitor, delle strategie di sconto e di pagamento, e delle classi principali che caratterizzano il funzionamento del programma.
Ogni test dà una prova di esecuzione di specifiche funzionalità, guardando anche a casi particolari di errore (come il sollevamento di eccezioni).
È stato previsto anche un test specifico per l'utilizzo delle *Lambda Expressions*. Si prevede infatti di poter applicare uno sconto al totale del carrello tramite una funzione assegnata al campo relativo. Infatti poiché l'interfaccia *DiscountStrategy* è una *Functional Interface* le si può assegnare una *Lambda Expression*.