Progetto per il Corso di
MP
–A.A. 2023/2024 –

Gestione degli item e del carrello di un negozio online

Autore: Nabiollah Tavakkoli
E-mail: nabiollah.tavakkoli@edu.unifi.it
Numero di matricola: 5953800
Data di consegna: 00/00/2024

Online shop

Un negozio online offre diversi prodotti (catalogo): il cliente può

- selezionare alcuni articoli,
- controllare il carrello degli articoli scelti,
- chiedere una fattura (che descrive I prodotti acquistati, con il loro prezzo, e il totale),
- accettare o ritornare indietro,
- effettuare il pagamento.

Vengono effettuati periodicamente dei Report, per controllare

- gli articoli in ciascuna categoria
- il prezzo più alto o più basso degli articoli in catalogo,
- quanti sono gli oggetti con il prezzo più alto, basso oppure in uno specifico intervallo

Ci sono un certo numero di prodotti di base ma anche pacchetti di prodotti che, a loro volta, possono essere formati sia da prodotti semplici che da pacchetti (usando il pattern Composite)

Il negozio può applicare dinamicamente particolari sconti su alcuni articoli, oppure decorare il carrello con particolari sconti.

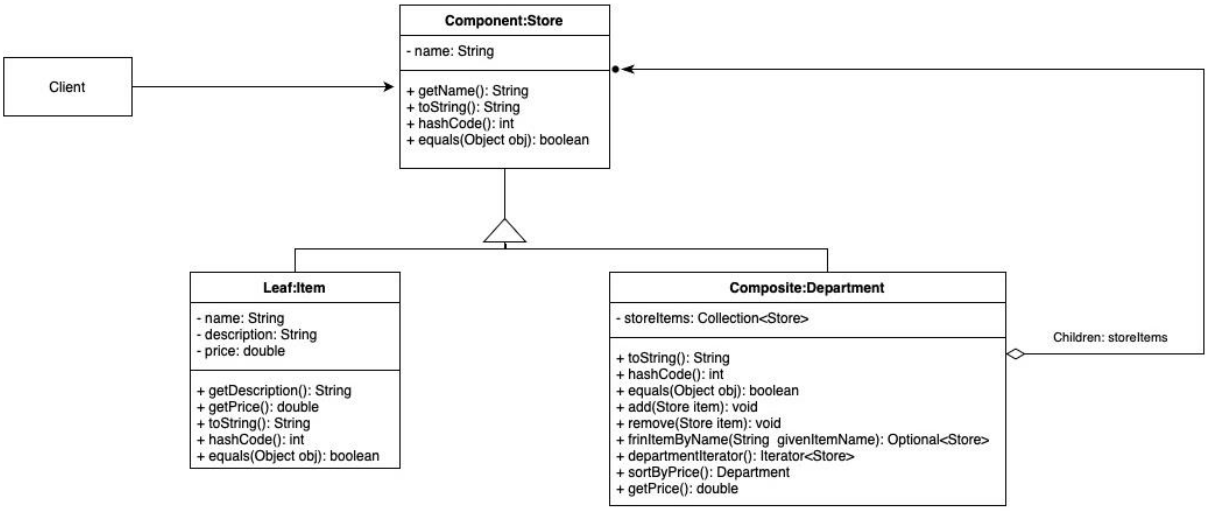
- In alternativa, prevedere l'uso del Pattern Strategy per modellare una possibile strategia di determinazione del prezzo di una vendita, che può essere
  - modificata periodicamente (ad es, sconto giornaliero del 10%)

Si può considerare più in dettaglio il problema della visualizzazione della fattura.

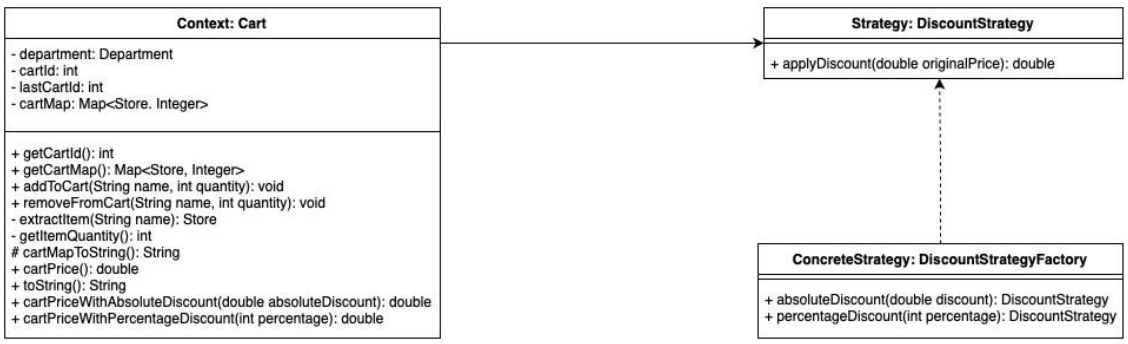
- Ad esempio, la fattura viene aggiornata man mano che si aggiorna il carrello (uso del Pattern Observer)

Pattern applicati

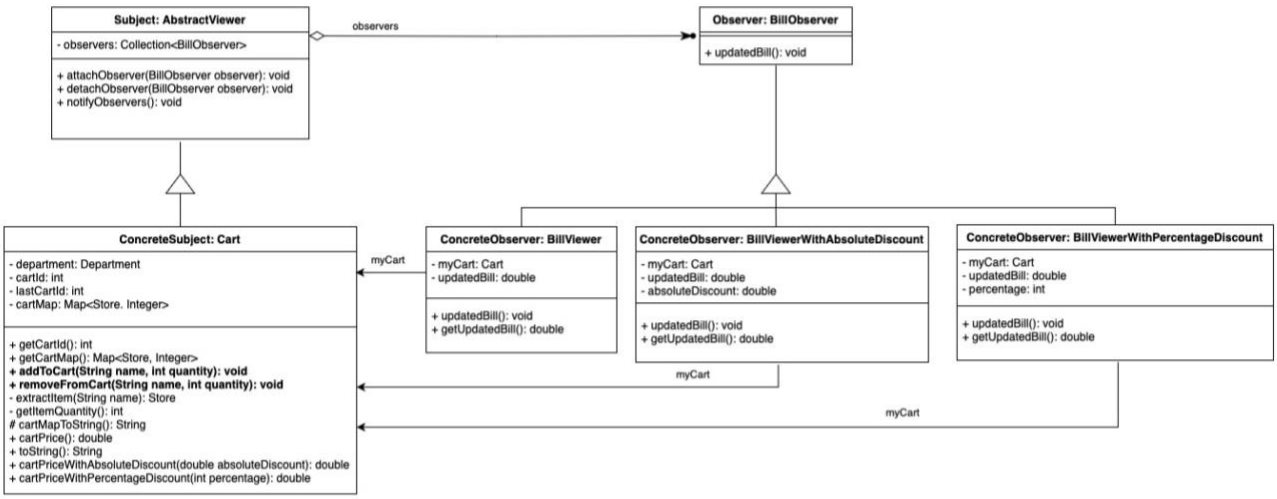
- Composite:
  - Il pattern viene applicato nella **forma type safe**, quindi la gestione dei *children* sarà presente solo nel componente composto (cioè il Department)
    - Component è *Store*
    - Oggetto composto è *Department*
    - Oggetto foglia è Item
  - Le tre classi del pattern hanno il metodo *getPrice* in comune che viene definito dalla classe *Store* e implementato dalle classi *Item e Department*
  - *Department* ha una collezione di oggetti (di tipo statico *Store*) detto *storeItems*



- Strategy
  - Il pattern strategy permette di applicare vari sconti sul carrello; cioè permette di modellare una possibile strategia di determinazione del prezzo di una vendita
    - Context è la classe *Cart* che al suo interno definisce i due metodi *cartPriceWithAbsoluteDiscount, cartPriceWithPercentageDiscount* che a run-time saranno implementati dai metodi definiti nella classe *DiscountStrategyFactory*
    - Strategy è l'interfaccia *DiscountStrategy* che definisce un solo metodo astratto *applyDiscount* che viene usato per applicare i vari sconti implementati dalla classe *DiscountStrategyFactory*
    - ConcreteStrategy è la classe *DiscountStrategyFactory* che implementa le varie strategy



- Observer
  - Subject è la classe *AbstractViewer* dove al suo interno vegono definiti i vari metodi per aggiunger, rimuovere, notificare gli oggetti osservatori
  - ConcreteSubject è la classe *Cart* che permette di notificare gli osservatori sul prezzo della fattura ogni volta che viene aggiunto o rimosso un nuovo item al/dal carrello
    - Ciò ottiene aggiungendo la chiamata del metodo *notifyObservers* all'interno dei due metodi *addToCart, removeFromCart*
  - Observer è l'interfaccia *BillObserver* che semplicemente definisce l'interfaccia per gli oggetti osservatori che devono essere notificati riguardo al cambiamento del prezzo della fattura
  - ConcreteObserver sono le classi *BillViewer, BillViewerWithAbsoluteDiscount, BillViewerWithPercentageDiscount* che mantengono un riferimento a ConcreteSubject (la classe *Cart*) ed implementano l'interfaccia *BillObserver*, notificando gli osservatori rispettivamente sul prezzo totale della fattura, sul prezzo totale della fattura dopo uno sconto assoluto e sul prezzo totale della fattura dopo uno sconto percentuale



Descrizione

Il progetto è stato organizzato in

- due source folder: *src* and *tests* che contengono rispettivamente il codice sorgente ed i vari test per testare tale codice sorgente
- tre pacchetti: *discount, shop, shopingCart*;
  Questi pacchetti contengono rispettivamente le classi che gestiscono
  - i vari item del negozio
  - le varie categorie (*Department*)
  - il carrello
  - lo sconto applicato al carrello

La classe

- *DiscountStrategy*
  - contiene i due metodi *absoluteDiscount e percentageDiscount* che ritornano un oggetto *DiscountStrategy* permettondo al metodo *applyStrategy* dell'interfaccia *DiscountStrategy* di applicare la strategia che ci interessa
    - questi metodi hanno come corpo una lambda che prende il prezzo originale (per esempio della fattura) e decrementa il prezzo totale di una quantità fissa (cioè il nostro discount)

- *Department*
  - Permette di dichiarare un department contente vari item oppure vari department; per questo motivo a ciascun oggetto Department è stato associato una variabile di istanza storeItems contenente i vari item inclusi in esso
  - Dichiara i vari metodi come:
    - *Add*: per aggiungere un item alla nostra collezione
    - *Remove*: per rimuovere un item dalla nostra collezione
    - *findItemByName*: per trovare item, usando il suo nome
      - tale metodo prende il nome del item passato come argomento e lo confronto con i nomi dei vari item presenti nella collezione storeItems e restituisce il primo che soddisfa il confronto
    - *departmentIterator*: restituisce un iterator per iterare la collezione di storeItems
    - *sortByPrice*: ordina la collezione storeItems in base al prezzo dei vari items e restituisce alla fine un oggetto department che la sua collezione dei suoi items risulta ordinato (in base al loro prezzo)
      - ciò viene fatto
        - costruendo un oggetto department, assegnandoli il nome dell'oggetto di partenza
        - aggiungendo alla collezione del nuovo oggetto department la collezione dell'oggetto department in ingresso ma ordinato
    - *getPrice*: restituisce la somma totale di tutti i prezzi dei vari items presenti nella collezione storeItems

- *AbstractViewer*
  - Permette di aggiungere, rimuovere, notificare gli osservatori chiamando rispettivamente i metodi attachObserver, detachObserver, notifyObservers

- *Cart*:
  - Indica un carrello, ha come campi
    - id del carrello (cartId),
    - una mappa che contiene la lista degli items e la loro quantità presenti nel carrello
  - prende in ingresso un oggetto department che ha la lista dei vari item o dei vari department contenente gli items oppure tutti due
  - contiene i metodi:
    - *addToCart*: per aggiungere un item nel carrello già presente nel department (atto di comprare)
      - ♦ se aggiungiamo un nuovo item, allora esso viene aggiunto alla mappa del carrello con la quantità data in ingrasso (dell'acquisto)
      - ♦ se aggoingiamo un item esistente, allora viene soltanto incrementata la sua quantità nella mappa

      In entrambi casi, l'osservatore viene notificato, indicando il nuovo prezzo del carrello
    - *removeFromCart*: per rimuovere un item dal carrello
      - ♦ se viene richiesto di rimuovere una quantità di item maggiore di quella presente nel carrello, allora viene sollevata una eccezione IndexOutOfBoundException, dicendo che non si può procedere
      - ♦ se viene richiesto di rimuovere una quantità di item minore di quella presente nel carrello, allora viene soltanto decrementiamo la quantità del item nella mappa del carrello
      - ♦ altrimenti, rimuoviamo item dalla mappa del carrello

      in entrambi casi di rimozione dal carrello, l'osservatore viene notificato, indicando il nuovo prezzo del carrello
    - *extractItem*: è un helper method
      - ♦ usato dal metodo *removeFromCart* per estrarre un item di tipo astratto *Store* dalla lista dell'oggetto department basandosi sul nome del item passato come argomento e poi usando tale oggetto per estrarre la sua quantità nella mappa del carrello,
      - ♦ se il nome del item non è presente nella lista del department allora viene sollevata una eccezione NoSuchItemException
    - *getItemQuantity*: è un helper method, usato per ricavare la quantià di tutti gli item presenti nel carrello
    - *cartMapToString*: restituisce una stringa contenente tutti gli item presenti nella mappa del carrello con la loro quantità
    - *cartPrice*: restituisce il prezzo totale del carrello, calcolando la somma dei prezzi di ciascun item moltiplicato per la sua quantità
    - *toString*: è praticamente la fattura, contenente
      - ♦ id del carrello
      - ♦ la quantità di tutti gli item comprati
      - ♦ la lista di tutti gli item con la loro quantità
      - ♦ il prezzo totale del carrello
    - *cartPriceWithAbsoluteDiscount*: restituisce il prezzo totale del carrello dopo l'applicazione di uno sconto assoluto (passato come argomento)
    - *cartPriceWithPercentageDiscount*: restituisce il prezzo totale del carrello dopo l'applicazione di uno sconto percentuale (passato come argomento)