# Esercizio design pattern

L'esercizio consiste nell'implementazione di un negozio. Il negozio può vendere prodotti singoli o bundle. I prodotti possono essere scontati, gli sconti devono poter essere applicati dinamicamente ai prodotti. I clienti possono usare vari metodi di pagamento. I clienti possono iscriversi al negozio ed essere notificati ogni volta che un nuovo prodotto è disponibile. I lavoratori del negozio devono poter effettuare operazioni con i prodotti.

## Stock

Rappresenta il magazzino dei prodotti. È un singleton, si accede all'istanza con ***getInstance()***.
È possibile aggiungere prodotti in magazzino con ***addProduct(Product product)*** e rimuoverli con ***removeProduct(Product product)***, inoltre è possibile controllare se un prodotto è in magazzino con ***hasProduct(Product product)***.
Per far visitare a un lavoratore l'intero magazzino si usa ***visitProducts(Worker worker)***.

## Shop Event Manager

Manda notifiche ai clienti iscritti con il pattern Observer. È anche un singleton, si accede all'istanza con ***getInstance()***.
I clienti si possono iscrivere con il metodo ***subscribe(Client client)*** e disiscrivere con il metodo ***unsubscribe(Client client)***.
Manda notifiche ai clienti con il metodo ***notifySubscribers(Product product)***.

## Client

Il metodo ***update(String notification)*** viene utilizzato dallo **Shop Event Manager** per inviare notifiche.
Contiene un oggetto **Cart** che rappresenta il carrello. Può aggiungere al carrello prodotti con ***addToCart(Product product)*** e rimuovere con ***removeFromCart(Product product)***.
Acquista i prodotti nel carrello con il metodo ***checkout()***, per acquistare ha bisogno di aggiungere un metodo di pagamento con ***addPaymentMethod(PaymentMethod paymentMethod)***.

## PaymentMethod

Interfaccia per i metodi di pagamento, è un'implementazione del pattern Strategy. Permette di usare procedure di pagamento diverse per metodi di pagamento diversi. Contiene il metodo ***pay(long cost).*** **PaymentCreditCard** e **PaymentPayPal** sono implementazioni concrete di **PaymentMethod**.

## Product

Interfaccia per i prodotti implementa i pattern Composite, Visitable e Decorator.
I product possono essere composti tra di loro con il metodo ***addProduct(Product product).*** Il metodo ***containsProduct(Product product)*** serve per evitare contenimenti ciclici tra due prodotti, altimenti si potrebbero contenere infinitamente tra di loro. I prodotti hanno un nome accessibile con ***getProductName()*** e un costo accessibile con ***getCost()***. Il metodo ***setDiscount(Discount discount)*** può assegnare un decorator **Discount** al prodotto, lo sconto viene applicato in ***getCost()*** con il metodo ***applyDiscount(long cost).*** I prodotti possono accettare i Visitatori **Worker** con il metodo ***accept(Worker worker)***.

**ProductSingle** è un'implementazione di **Product** che non può contenere altri prodotti, **ProductBundle** invece contiene prodotti.

## Worker

Interfaccia per Visitor, ***visitProduct(Product product)*** visita un prodotto generico, ***visitProductSingle(ProductSingle product)*** e ***visitProductBundle(ProductBundle product)*** visitano i tipi concreti ***ProductSingle*** e ***ProductBundle.*** Può visitare tutto il magazzino col metodo ***visitProducts(Worker worker)*** dell'oggetto **Stock.**

## Discount

Interfaccia per sconti da applicare, è un'implementazione del pattern Decorator.
il metodo ***applyDiscount(long cost)*** prende in ingresso il prezzo da scontare e restituisce il prezzo scontato.