UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

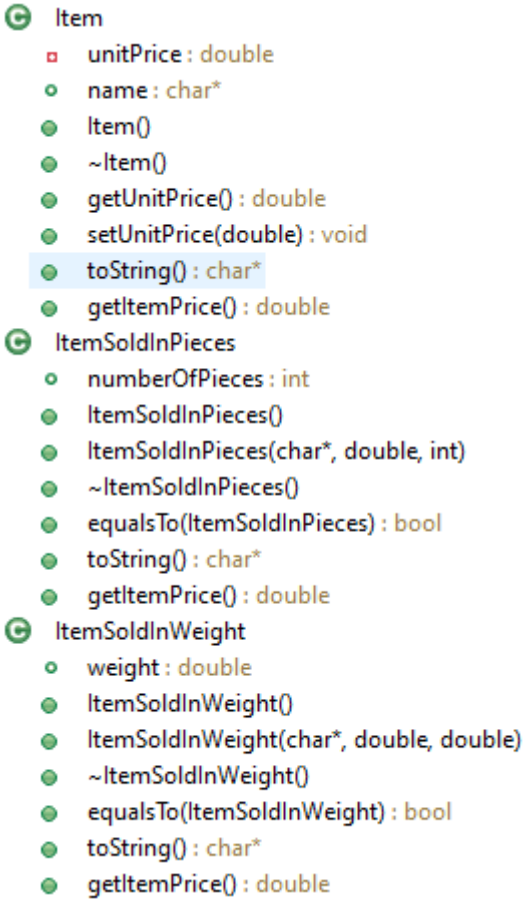
Progetto PA: C++

"Supermarket"

Boschetti Kevin, matricola n. 1057400

Obiettivo di questo applicativo è sviluppare un programma in C++ che permettesse di svolgere gli stessi task della controparte in Java per la gestione di un supermercato, seppur con qualche semplificazione. Ho scelto di implementare due tipologie di prodotti, quelli venduti in base alla quantità e quelli venduti in base al peso. Si possono aggiungere gli articoli a dei carrelli. Infine si può calcolare il costo della spesa che varierà in base alla carta fedeltà, presente o meno, e al giorno della settimana.

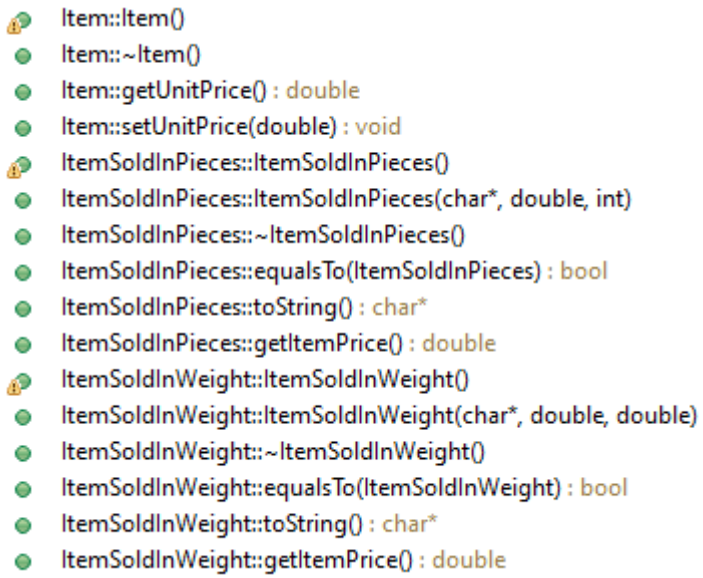
Item.h



Nel file *Item.h* sono presenti le dichiarazione di variabili, costruttori e metodi che poi verranno implementati nel file *Item.cpp*. Ogni prodotto è caratterizzato da un *nome* public e da uno *unitPrice* private che, per questa ragione, può essere estratto e impostato solo grazie ai metodi *getUnitPrice* e *setUnitPrice*. In particolare *Item* è una classe astratta ed è resa tale a causa dei metodi *toString* e *getItemPrice* che sono inizializzati a zero. Ciò significa che non può esistere un oggetto della classe *Item*, ma solo oggetti appartenenti alle classi che la estenderanno, e che dovranno necessariamente fornire un'implementazione specifica di questi metodi dichiarati *virtual*. Si noti anche la presenza del *distruttore virtual*, fondamentale che sia così in una classe che viene estesa al fine di distruggere anche l'oggetto della classe base oltre a quello della classe derivata.

Ogni prodotto può essere venduto a pezzi oppure in base al peso. Gli elementi della classe *ItemSoldInPieces* sono caratterizzati dal campo *numberOfPieces*, mentre gli oggetti della classe *ItemSoldInWeight* dalla variabile *weight*. Entrambi hanno un *costruttore base* e uno che accetta i parametri da inizializzare, oltre che un distruttore specifico non più virtual alla cui chiamata seguirà dunque quella del distruttore di *Item*. In entrambe queste due classi è presente anche un metodo *equalsTo* che permette di confrontare due articoli.

Item.cpp

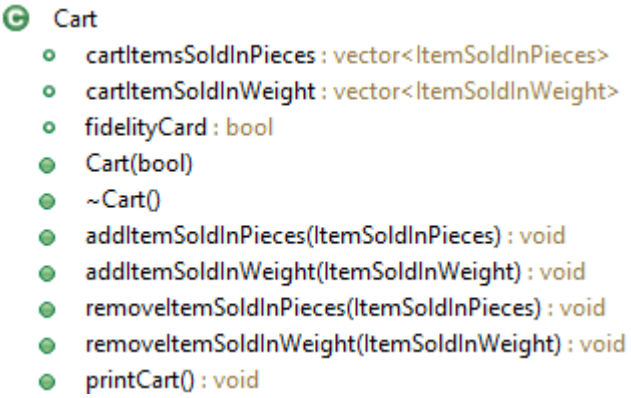


In questo file sono presenti le implementazioni dei costruttori e dei metodi del relativo file.h. Troviamo quindi i costruttori e i distruttori di *Item*, *ItemSoldInPieces* e *ItemSoldInWeight*. In particolare il campo *name* di ogni articolo creato tramite una chiamata alla funzione *malloc* che permette di allocare uno preciso spazio di memoria nello heap che verrà poi eventualmente deallocato tramite la chiamata al distruttore dell'oggetto. Nello specifico poi, dato che viene passato come parametro un *puntatore char*, il campo *name* viene creato copiando direttamente il contenuto della stringa ricevuta in input. Infine viene settato l'ultimo carattere a '\0' per terminare in sicurezza la stringa ed evitare *buffer overrun*.

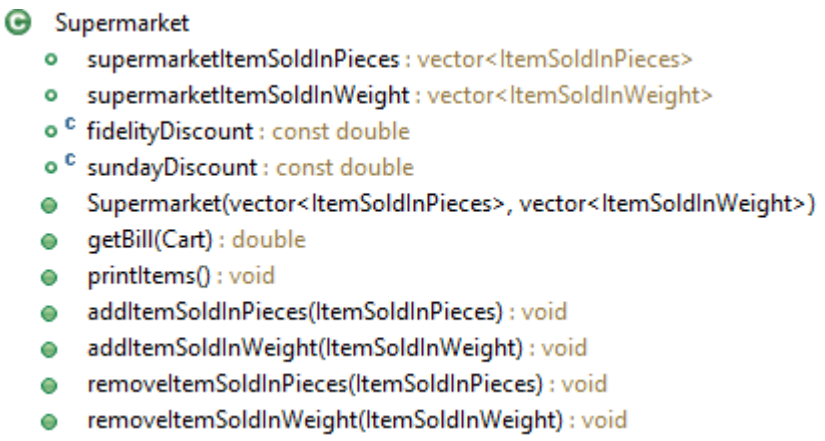
I metodi get e set per lo *unitPrice* appartengono alla classe *Item* ma che possono essere usati anche dalle classi che la ereditano. Il metodo *equalsTo* è implementato in due modi differenti: per *ItemSoldInPieces* confronta *name*, *numberOfPieces* e *unitPrice*, mentre per *ItemSoldInWeight* prende in considerazione il campo *weight*.

Troviamo poi il metodo *toString* che stampa il *name* dell'oggetto che lo invoca creando una stringa con lo stesso criterio utilizzato nei costruttori. Infine il la funzione *getItemPrice* restituisce il prezzo di un prodotto in base allo *unitPrice* e al peso, oppure al *weight*.

Supermarket.h



In questo file è possibile trovare due classi: *Cart* e *Supermarket*.
Nella prima troviamo i campi *cartItemsSoldInPieces* e *cartItemSoldInWeight*. Entrambi sono del tipo *vector*, una classe di *STL (Standard Template Library)*. Per utilizzarlo occorre capire che STL definisce degli algoritmi che operano su un intervallo all'interno di un contenitore. Ogni vector è composto dal tipo dell'elemento base (*cartItemsSoldInPieces* o *cartItemSoldInWeight*), e dal nome del contenitore che corrisponde al nome dell'oggetto istanziato. Infine un carrello è caratterizzato anche dalla presenza o meno del booleano *fidelityCard* che specifica se il cliente ha diritto o meno allo sconto fedeltà. Questo campo è l'unico mandatorio per il costruttore. I metodi che verranno poi implementati sono *addItemSoldInPieces* e *addItemSoldInWeight*, e di conseguenza *removeItemSoldInPieces* e *removeItemSoldInWeight*. Infine con *printCart* sarà possibile stampare il contenuto di un carrello.



Nella classe Supermarket invece trova spazio il vero oggetto del progetto.
Come vedremo nel main è possibile creare svariati supermercati e associargli molteplici carrelli. Un supermercato è caratterizzato da due costanti, un *fidelityDiscount* applicato al conto del carrello se il cliente possiede la carta fedeltà, e un *sundayDiscount* applicato se la spesa viene fatta di domenica. Inoltre il costruttore di un supermarket richiede come parametri due liste, ovvero *supermarketItemSoldInPieces* e *supermarketItemSoldInWeight*. I metodi che verranno implementati sono *getBill* che permette di calcolare il costo della spesa di uno specifico carrello all'interno di un supermercato; *printItems* che stampa i prodotti acquistabili nel supermercato. Infine è

possibile aggiungere o eliminare dalla lista dei prodotti un articolo in base alla classe a cui appartiene.

Supermarket.cpp

Questo file è quello che contiene il metodo *main* che permetterà di lanciare l'applicativo ed iniziare ad utilizzarlo. Al suo interno troviamo le implementazioni dei metodi e dei costruttori dichiarati nel rispettivo file.h.
Per quanto riguarda la classe *Cart* possiamo notare l'utilizzo della funzione *push_bac* che permette di aggiungere in coda ad un *vector* un elemento della classe specificata come elemento base. Nei metodi utili a rimuovere un articolo dalla lista dei prodotti si può notare la creazione di un oggetto *iterator* appartenente alla classe *vector*. Questo risulta fondamentale per poter utilizzare la funzione *erase* che permette di eliminare un elemento dal vettore in base alla posizione.
Nella classe *Supermarket* invece è presente il metodo *getBill* che permette di calcolare il costo della spesa considerando la carta fedeltà e anche se il giorno corrente è domenica.

- Supermarket::Supermarket(vector<ItemSoldInPieces>, vector<ItemSoldInWeight>)
- Supermarket::printItems() : void
- Supermarket::addItemSoldInPieces(ItemSoldInPieces) : void
- Supermarket::addItemSoldInWeight(ItemSoldInWeight) : void
- Supermarket::removeItemSoldInPieces(ItemSoldInPieces) : void
- Supermarket::removeItemSoldInWeight(ItemSoldInWeight) : void
- Supermarket::getBill(Cart) : double
- main() : int

Di seguito l'output del main in cui ho fatto qualche test:

```
---------------- Prodotti creati ----------------
detersivo
dentifricio
dopobarba
cesto banane
noci
mele
---------------- Creo liste di prodotti ----------------
---------------- creo supermarket ----------------
---------------- aggiungo dopobarba----------------
Prodotti venduti a pezzi:
detersivo
dentifricio
dopobarba
Prodotti venduti a peso:
noci
mele
```

```
---------- rimuovo dopobarba ----------
Prodotti venduti a pezzi:
detersivo
dentifricio
Prodotti venduti a peso:
noci
mele

---------- creo due carrelli, con e senza carta fedeltà ----------
---------- riempio i due carrelli ----------
---------- stampo prodotti nel carrello ----------
Prodotti nel carrello:
venduti a pezzi:
detersivo
dentifricio
venduti a peso:
cesto banane
noci

---------- rimuovo detersivo ----------
Prodotti nel carrello:
venduti a pezzi:
dentifricio
venduti a peso:
cesto banane
noci

---------- stampo conto carrelli ----------
cliente fedele: 11.67€
cliente non fedele: 13.73€
```