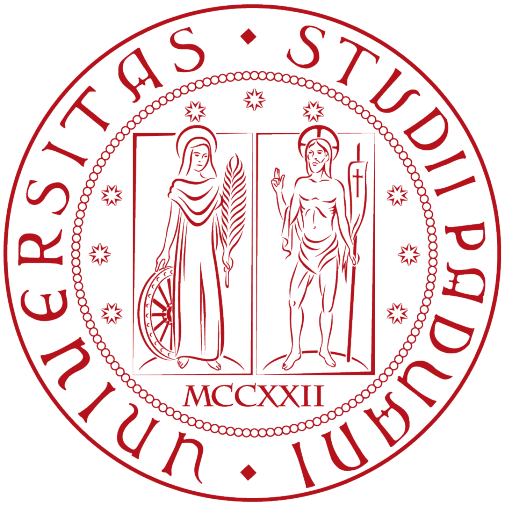
Università degli Studi di Padova
Corso di laurea in Informatica




Relazione sul progetto di
Programmazione ad Oggetti
a.a. 2016/2017

# easyMARKET

Lisa Parma
matricola: 1121284

# 1. Scopo del progetto

## Introduzione

L'applicazione easy MARKET ha l'obbiettivo di modellare e gestire un supermercato sia dalla parte amministrativa che dalla parte utente operando su contenitori (per gli utenti e per i prodotti).

## Note per la compilazione

L'applicazione è stata creata usando:

- Qt 5.8.0
- Qt Creator 4.2.1
- macOS Sierra 10.12.6

Dato che viene consegnato un file .pro i comandi per la compilazione da terminale saranno solamente qmake make.

## Dati d'accesso:

- admin:
  username: admin  password: admin
- userBase:
  username: userbase  password: userbase
- userPremium:
  username: premium  password: premium

## Vincoli

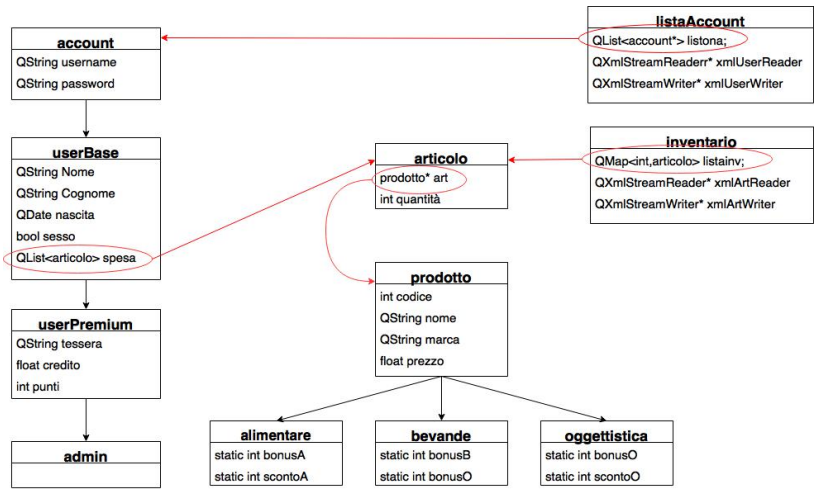
1. Il modulo di amministrazione deve permettere:
   - (a) Lettura da file a memoria e scrittura da memoria su file del contenitore C:
     Sia la lista degli utenti già registrati che quella dei prodotti disponibili vengono lette da file xml all'apertura dell'applicazione e vengono aggiornati ogni qualvolta dovessero avvenire modifiche nei due contenitori.
   - (b) Inserimenti, rimozioni e modifiche degli oggetti di C in memoria:
     L'amministratore può apportare modifiche al contenitore "inventario" aggiungendo nuovi prodotti, eliminandone o modificando le quantità in magazzino. Può inoltre modificare particolari valori "Sconto" e "Punti Bonus" per ogni sottotipo di prodotto.
   - (c) Gestione degli utenti del sistema:
     L'amministratore può gestire la lista di utenti eliminandoli e modificando il loro status facendo upgrade o downgrade dei loro account. La registrazione di un nuovo utente, invece, è libera (non è vincolata da un preventivo accesso di un amministratore).
   - (d) Si devono prevedere più tipi di registrazione per gli utenti, in cui ogni tipo di registrazione garantisce all'utente diversi servizi di accesso al contenitore C:
     Ci sono tre tipi di account registrabili: admin, accountBase e accountPremium, ognuno dei quali predispone di una sua interfaccia grafica con funzionalità a lui riservate.
2. Il modulo utente deve permettere ad un utente registrato U di accedere al contenitore C secondo i servizi di accesso garantiti ad U dal suo tipo di registrazione:
   Ai tre account disponibili corrispondono tre finestre di dialogo diverse ognuna delle quali predispone l'accesso alle funzionalità riservate al tipo di account.

# 2. Orientamento agli oggetti - MODEL

## Descrizione delle gerarchie

Le classi realizzate a livello MODEL sono undici: una gerarchia principale che rappresenta i vari tipi di utenti, una classe per gestire gli utenti in una lista, una seconda gerarchia di prodotti che verranno venduti dal supermercato utilizzata nella classe che sarà il contenuto del contenitore (un insieme di prodotti uguali) e, come per gli utenti, una classe che gestisce il contenitore.
Tutte queste classi sono rappresentate dal diagramma sottostante:

- per ognuna sono elencati i propri campi dati, sia privati che pubblici
- le relazioni tra classi nelle gerarchie sono raffigurate da frecce nere, da superclasse a sottoclasse
- solo per maggior chiarezza, sono state rappresentate in rosso le relazioni tra il tipo dei campi dati e la la classe di appartenenza utilizzata nel progetto




### Gerarchia *account*

- **account**: la classe account è una classe base astratta che serve solo per generare le successive classi da lei derivate, che avranno tutte la stessa interfaccia di base. Il suo costruttore è protetto dato che, essendo una classe astratta, non può essere istanziata da sola ma serve nella costruzione delle classi derivate.
- **userBase**: deriva pubblicamente da account, possiede un campo dati di tipo QList che verrà utilizzato come lista della spesa che l'utente può modificare a piacere. Ovviamente viene definito il metodo virtuale fornito dalla classe base che calcola il costo totale della lista della spesa.
- **userPremium**: classe derivante pubblicamente da userBase estende le sue funzionalità offrendo all'utente la possibilità di comprare i prodotti in lista direttamente dal programma. Avrà a disposizione un credito (che potrà ricaricare a piacimento) da cui scalare l'importo della spesa, un saldo punti e un particolare sconto che varierà per ogni tipologia di prodotto. Il metodo virtuale per calcolare il totale della spesa viene ridefinito integrando gli sconti possibili.
- **admin**: l'ultima sottoclasse della gerarchia è la classe admin che estende ulteriormente le sue funzionalità potendo gestire interamente la lista di utenti e gli articoli con aggiunte, rimozioni e modifiche.

La classe **listaAccount** permette di gestire tutti gli account in una lista, importandoli da un file xml e salvando le aggiunte o le modifiche.

Gerarchia **prodotto**

- **prodotto**: una seconda gerarchia parte dalla classe astratta prodotto che offre due metodi virtuali ai suoi eredi: la riduzione di prezzo e i punti. Ha tre sottoclassi:
    - **alimentare/bevanda/oggettistica**: derivazioni pubbliche di prodotto che modellano tre tipologie di prodotti vendibili dal supermercato. Per ognuna di esse ci sono due campi statici (modificabili dall'admin) che stabiliscono un certo sconto e numero di punti per ogni tipologia di prodotto.

La classe **articolo** contiene un puntatore ad un determinato prodotto e un intero positivo. Con questa classe si vuole facilitare la gestione di più prodotti uguali nella lista della spesa dei clienti e nell'inventario del negozio e, soprattutto, si vuole gestire al meglio la memoria (un puntatore e un intero n occuperanno meno spazio di n puntatori).

La classe **inventario** funge da contenitore: ha un campo dati QMap che mappa articoli in base al loro codice e li gestisce importandoli da file xml e salva eventuali modifiche. È stata scelta una QMap per una più veloce ricerca di un determinato articolo rispetto ad una QList.

## Incapsulamento

Nascondendo i dettagli implementativi delle classi si protegge il programma dalla propagazione di eventuali errori o cambiamenti. Si è scelto di garantirlo attraverso un attento uso degli specificatori di accesso *private* e *protected*, rendendo *public* i metodi getter e setter per i campi dati che ne prevedevano un uso.
Esempio: i costruttori delle classi base astratte user e prodotto sono stati definiti come *protected* dato che possono essere usati solamente dalle proprie sottoclassi. Un loro uso al di fuori delle sottoclassi sarebbe segnalato come errore e per evitare ciò sono stati resi inaccessibili dall'esterno della gerarchia.

## Modularità

Si è potuto separare completamente la parte logica e quella grafica del programma facendo in modo che ogni parte non dipendesse dall'implementazione dell'altra. Così si è potuto garantire il principio di single responsability in modo che ogni classe/metodo abbia una sola responsabilità.

## Estensibilità ed evolvibilità

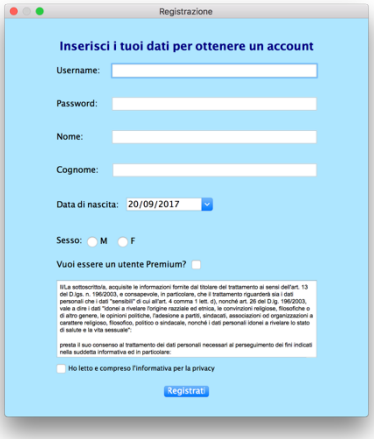
Entrambe le gerarchie sono state fatte partire da una classe base astratta che contiene dei metodi virtuali: in questo modo si garantisce la possibilità di estenderle in futuro aggiungendo sottoclassi, ognuna con un'implementazione diversa di quel metodo (override).
Esempio:
La classe base prodotto ha il metodo virtuale:

```
virtual float prezzoScontato() const = 0;
```

Per le sue sottoclassi alimentare, bevanda e oggettistica (che rappresentano tre diverse categorie di prodotto) il prezzo scontato per l'utente premium viene calcolato in modo diverso: ogni classe ha dei campi dati statici che sono decisi dall'amministratore e che possono variare run-time.
Il metodo che calcolerà il prezzo totale della spesa non dovrà preoccuparsi della tipologia di prodotto passatogli per calcolare il suo prezzo con lo sconto e utilizzerà soltanto il metodo prezzoScontato() const per ogni prodotto. Anche se in futuro si aggiungerà una nuova sottoclasse di prodotto basterà implementare la nuova classe con un override di prezzoScontato() per non causare problemi.

## 3. Funzionalità


Quando viene lanciato il programma viene innanzitutto caricato in una lista un database di utenti da un file xml.
Una volta inseriti i dati del proprio account e premuto il tasto LOGIN il programma effettuerà dei controlli per controllare se l'username esiste nel database e, in caso affermativo, se la password inserita corrisponde a quel determinato account.
Nel caso non ci siano errori si aprirà la finestra corrispondente al tipo di account immesso, nel caso contrario una QMessageBox avviserà dell'errore avvenuto.

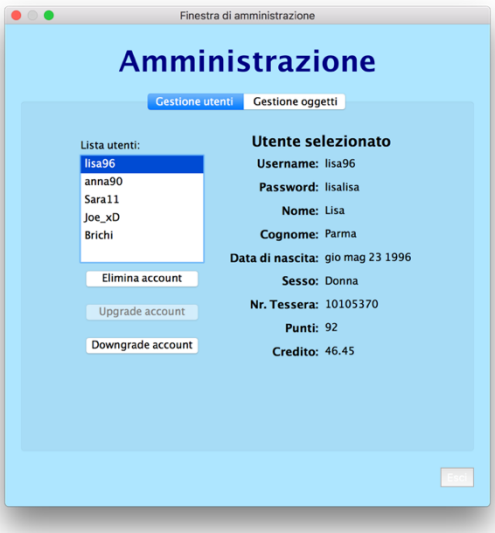



Se invece il file xml non viene trovato si genera in automatico un nuovo file xml con un solo utente admin (username: admin, password: admin) che viene caricato nella lista utenti. Successivamente potranno essere aggiunti al file xml i nuovi account che verranno creati tramite la registrazione.

Se l'account inserito è di tipo admin il programma richiederà se si vuole entrare come admin o come userPremium in quanto l'admin, essendo una sottoclasse di userPremium, può beneficiare delle sue funzionalità.
La finestra di amministrazione ha a sua disposizione due settori: uno per la gestione degli utenti e una per la gestione dei prodotti che vengono venduti nel supermarket:
La sezione di amministrazione utenti visualizza l'elenco degli utenti iscritti e per ogni account selezionato vengono visualizzati tutti i dati conosciuti che possono variare a seconda del tipo di account.
Anche le modifiche che si possono apportare agli account variano a seconda dell'account a cui devono essere applicate:

- Eliminazione account: questa funzione è possibile per per tutti gli account tranne che per un account di tipo admin.
- Upgrade: si può effettuare solo da un account di tipo userBase ad un account userPremium. In caso venga selezionato un userPremium il pulsante che permette l'upgrade viene disabilitato: questo è solo una ulteriore accortezza dato che il metodo esegue un ulteriore controllo per verificare che gli venga passato un account di tipo userBase.
- Downgrade: si può effettuare solo da un account di tipo userPremium ad un account di tipo userBase. Anche per questa funzione viene disabilitato il pulsante abbinato a questa funzione nel caso venga selezionato un account userBase sebbene il metodo preveda un controllo per il tipo di account passatogli.

La sezione di gestione oggetti visualizza l'elenco degli articoli venduti dal market con relativo codice identificativo e viene visualizzato in rosso se il numero di prodotti relativi a quel prodotto è sceso a 0. Per ogni articolo selezionato viene visualizzato la propria tipologia (sottoclasse), il codice identificativo che permette di cercarli nella QMap, un nome, una marca, un prezzo e la propria quantità nel negozio che può essere incrementata ordinando più pezzi. Ogni articolo presente può essere eliminato all'occorrenza e si possono aggiungere all'inventario nuovi prodotti.
Inoltre, sempre nella sezione di gestione prodotti, si possono modificare i valori della percentuale di sconto applicati agli utentiPremium e i punti bonus per ogni prodotto comprato di un determinato sottotipo.

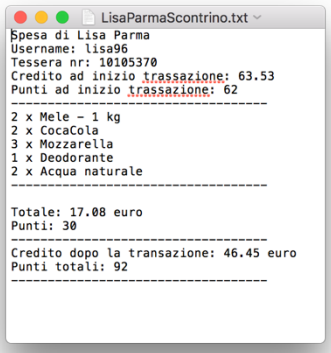




Accedendo invece con i dati di un userBase (esempio: username:anna90, password: annaanna) verrà aperta un finestra che visualizzerà i dati dell'account sulla sinistra e permetterà le interazioni con il contenitore sulla destra.
Verranno visualizzati tutti i prodotti disponibili nel market (compresi di marca, prezzo e quantità disponibile) e semplicemente facendo doppio click su un articolo questo verrà aggiunto alla propria lista della spesa. Si potrà, a sua volta, modificare la lista della spesa incrementando o decrementando la quantità di un articolo in essa o eliminandone tutte le occorrenze. Man mano che si faranno modifiche alla lista della spesa dell'utente verrà calcolato un prezzo totale degli articoli presenti in lista. Quando si è finito di selezionare e modificare gli articoli si potrà ricorrere ad una stampa della lista creata: si salverà un file txt con l'elenco dei prodotti scelti.
Un utente base può fare l'upgrade del proprio account direttamente da questa finestra senza dover ricorrere all'amministratore.

Se si accede con i dati di un account di tipo userPremium si aprirà la finestra di dialogo dedicata a questo tipo di utente. Simile per aspetto alla finestra dell'utenteBase aggiunge delle funzionalità e modifica alcuni aspetti. Innanzitutto vediamo che i dati dell'account visualizzati sono di più: il numero della tessera viene generato casualmente quando ci si iscrive e poi salvato nel file xml. I punti vengono guadagnati ogni euro speso ed in base al bonus che in un determinato momento può avere un tipo di prodotto. Il credito, invece, può essere ricaricato e viene decrementato ogni volta che viene fatta la spesa.


La visualizzazione di tutti gli articoli presenti nel market può essere scremata in base alle sottoclassi di prodotto, così che risulti più facile la ricerca di un determinato articolo.
Anche in questa finestra il calcolo del prezzo totale, del prezzo totale scontato e dei punti accumulati avviene in contemporanea alle modifiche alla lista.




Quando si è finito di scegliere i prodotti si può procedere all'acquisto: verrà generato uno scontrino in formato txt con i dati dell'account, gli articoli comprati, i soldi spesi e i punti guadagnati. I soldi verranno detratti dal credito disponibile nell'account (se questo è insufficiente non si potrà procedere all'acquisto finché non si ricaricherà il proprio credito) e verranno aggiunti al proprio saldo punti quelli dell'attuale spesa.
Un'altra differenza importante con l'userBase è che un utente Premium procedendo all'acquisto toglie i prodotti comprati dall'inventario del market, mentre un utente Base stampa soltanto una lista della spesa che non influirà nell'inventario.

## Impegno temporale

- Progettazione MODEL e GUI: 5-6 h
- Codifica MODEL: circa 8 h
- Codifica GUI: circa 30 h
- Debugging: 6-7 h
- Modifiche/aggiunte finali: circa 5-7 h

In particolare è stato difficile calcolare un dato preciso per l'ammontare ore della codifica della GUI in quanto la scrittura del codice è stato mescolato con quello dedicato all'apprendimento della libreria Qt e al tentativo di sfruttare al massimo le possibilità che il tool QtDesigner può offrire.