Relazione progetto Freedom Bookshop

Lorenzo Ferrarin - 1069405

Sistema operativo di sviluppo: Windows 10 e Ubuntu Linux 12.04.3
Compilatore: GCC 4.6.3
Libreria Qt: Qt 5.3

1. Scopo del progetto

Il progetto si pone lo scopo di realizzare il programma utilizzato dalla cassa di una libreria. A tale fine si vogliono fornire le seguenti funzionalità: aggiunta e rimozione di prodotti (libri e dvd); aggiunta e rimozioni di diverse tipologie di utenti (standard, premium e dipendenti della libreria); costruzione di uno scontrino che rappresenti la specifica transazione da parte di un utente. Il programma si occupa inoltre di salvare e caricare su un database xml i dati relativi ai prodotti e agli utenti.

2. Modello

Illustriamo ora le principali scelte progettuali effettuate per la realizzazione del modello dei dati del programma.

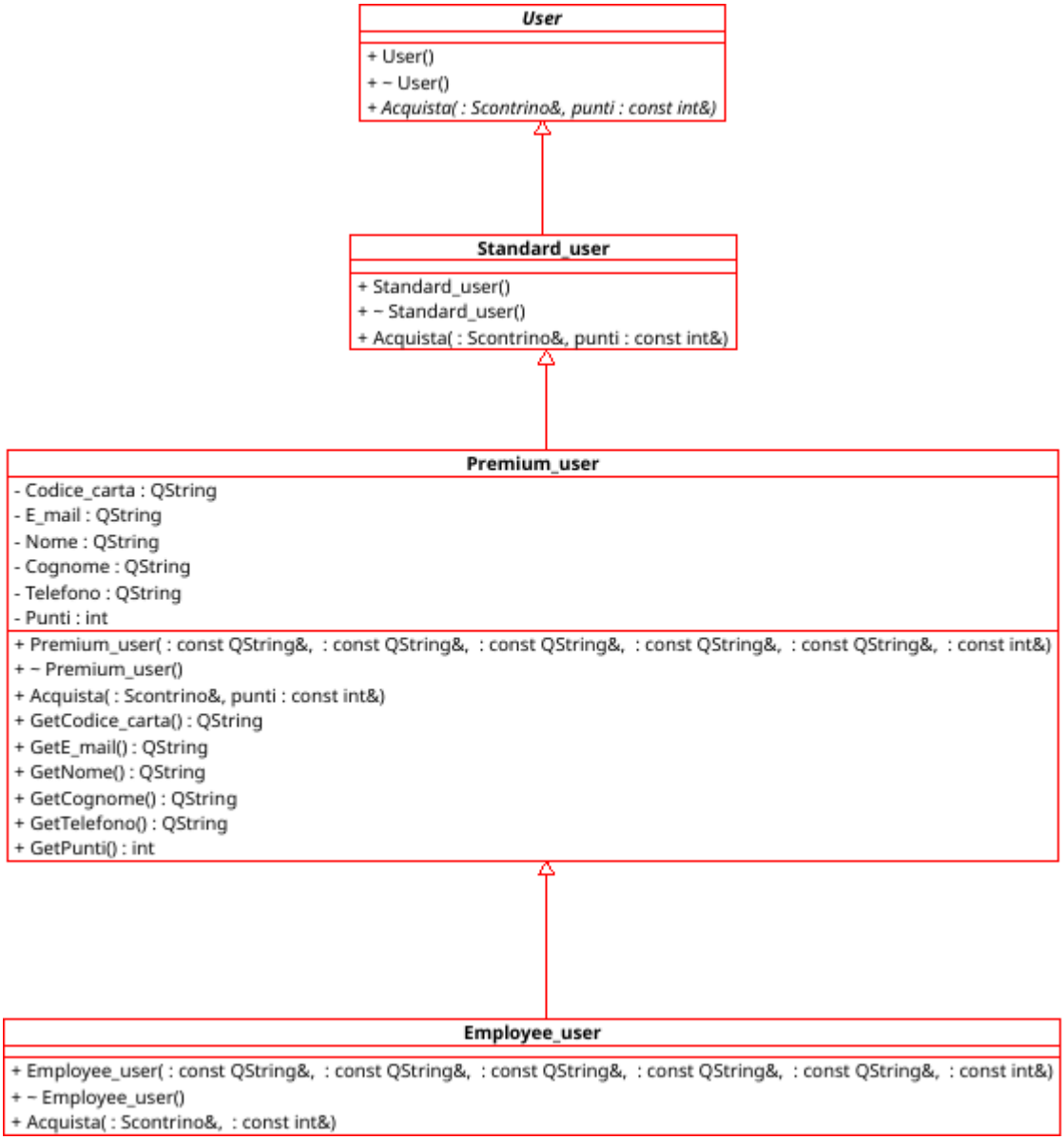
2.1 Gerarchie

Il progetto si basa su due gerarchie fondamentali, quella dei prodotti e quella degli utenti. La prima ha come base astratta *Item* di cui sono disponibili due implementazioni: *libri* e *DVD*. La seconda invece ha come base astratta *User* dalla quale è derivata una gerarchia verticale costituita da *Standard user, Premium user* e *Employee user.* Per la seconda gerarchia è stata costruita un'apposita classe contenitore *Userlist* che contiene dei puntatori alla base astratta della gerarchia. E' stata inoltre costruita la classe *Scontrino* che viene utilizzata per rappresentare un acquisto da parte di un utente.

2.1.1 Gerarchia degli utenti

La gerarchia degli utenti è costituita da quattro classi derivate verticalmente le une dalle altre. La classe base astratta della gerarchia è User: essa presenta il solo vincolo dell'intera gerarchia, ovvero l'implementazione del metodo virtuale puro *Acquista*. Il metodo *Acquista* rappresenta la funzione principale che compiono gli utenti, ovvero acquistare una specifica serie di prodotti rappresentata da uno scontrino. Questo metodo viene ridefinito ed espanso dalle classi derivate, aggiungendo progressivamente nuove funzionalità. La seconda classe della gerarchia è *Standard User*: rappresenta l'utente generico, senza caratteristiche particolari. Per tale ragione non dispone di ulteriori campi dati rispetto alla sua base astratta, ma è soltanto una realizzazione concreta di *User.* La classe *Standard User* dunque non ha bisogno di essere memorizzata sul database in quanto non dispone di alcun campo dati o di informazioni su un utente specifico. La terza classe della gerarchia, derivata pubblicamente da *Standard User,* è *Premium User. Premium User* rappresenta un utente che si è iscritto alla libreria tramite una tessera. A tale scopo la classe memorizza una serie di dati sull'utente aventi lo scopo di identificarlo e contattarlo in caso di necessità. Per ogni campo dati è stata anche implementata un'apposita funzione costante che permetta di ottenerne il valore. La classe *Premium User* estende e amplia le funzionalità rese disponibili da *Standard User,* motivo per cui è derivata dalla classe concreta e non dalla base astratta della gerarchia. La funzionalità introdotta dalla classe *Premium User* è quella di accumulare punti. Ogni volta che un *Premium User* effettua un acquisto guadagna un numero di punti pari alla cifra spesa diviso dieci. Negli

acquisti successivi il *Premium User* avrà quindi la possibilità di spendere i punti così guadagnati per avere uno sconto sul prezzo totale dell'acquisto. La quarta e ultima classe della gerarchia è *Employee User,* derivata pubblicamente da *Premium User.* La classe *Employee User* rappresenta gli utenti che sono anche dipendenti della libreria. Essa amplia ulteriormente le funzionalità di *Premium User* ridefinendo la funzione *Acquista* in modo tale che, oltre ad accumulare e spendere punti, fornisca all'utente uno speciale sconto del venti per cento su tutti i suoi acquisti. La gerarchia di classi è anche facilmente estensibile: per esempio si potrebbe aggiungere una nuova tipologia di utente *Loyal User*, derivata da *Premium User*, che rappresenti un utente con carta fedeltà. Questa tipologia di utente potrebbe avere, ogni cinque acquisti, uno sconto del 50% sull'acquisto successivo, ma senza disporre dello sconto speciale su tutti gli acquisti riservato agli impiegati della libreria.



2.1.2 Gerarchia dei prodotti

La gerarchia dei prodotti ha alla sua base la classe astratta Item, che fornisce un contratto per quelle che saranno le sue implementazioni concrete. Questo contratto è costituito dai due campi dati della classe, Codice e Prezzo, e dal metodo virtuale puro *GetNome.* Tutte e tre queste funzioni rappresentano anche l'interfaccia generale con la quale le altre classi interagiscono con la gerarchia.

In particolare Prezzo e *GetNome* vengono utilizzati per realizzare e stampare gli scontrini, mentre Codice è utilizzato per distinguere i diversi prodotti. Della classe *Item* sono state fornite due implementazioni concrete: *Libri* e *DVD*. Le due classi si distinguono per i diversi campi dati di cui sono provviste, ciascuno con il rispettivo metodo *Get*. A differenza della gerarchia degli utenti entrambe le classi ereditano direttamente dalla classe base astratta, in quanto nessuna delle due può essere considerata una reimplementazione dell'altra. Esse si distinguono per i diversi campi dati che rappresentano le informazioni più comunemente associate a queste tipologie di prodotti, ed entrambe reimplementano il metodo *GetNome*. La gerarchia si presta facilmente ad ulteriori estensioni sia in verticale che in orizzontale. Per esempio si potrebbe implementare una classe derivata da *Libro* chiamata *Audiolibro*. *Audiolibro* possiederebbe gli stessi campi dati base di *Libro* con l'aggiunta di alcuni specifici, come la durata del cd audio o l'identità del narratore. Allo stesso modo si potrebbe costruire una classe *Periodici* derivata dalla classe base astratta *Item*, che rappresenti le riviste vendute dalla libreria, le quali non rientrano in nessuna delle due tipologie di prodotti realizzate per il progetto.

2.2 Classe contenitore UserList

Al fine di contenere gli oggetti della gerarchia degli utenti è stata costruita la classe contenitore *Userlist*. *Userlist* dispone delle seguenti classi interne: *Nodo*, utilizzata per contenere i puntatori agli

oggetti di tipo *User* e per collegare i diversi *User* per costruire la lista; *smartUp*, un puntatore smart il cui unico campo dati è un puntatore a nodo e che si occupa di mantenere aggiornato il campo counter di nodo senza la necessità di farlo manualmente; *UserlistIterator,* che rappresenta l'iteratore della *Userlist* permettendo di scorrerla. *Userlist* dispone di due campi dati privati di tipo *smartUp,* first e last. Questi sono dei puntatori rispettivamente al primo e all'ultimo nodo della lista. Mantenere aggiornati entrambi questi campi rende possibile l'implementazione dei metodi *push_back, push_front, pop_back* e *pop_front* in tempo costante indipendentemente dalla lunghezza della lista. Per la classe contenitore sono state anche realizzate due funzioni di rimozione che permettono di rimuovere utenti specifici dalla lista, identificati in un caso da un iteratore e in un altro dal loro codice.

2.3 Uso del polimorfismo

Le principali chiamate polimorfe che sono state utilizzate durante il progetto sono quelle ai metodi *Acquista* di *User* e *GetNome* di *Item.* La prima permette di realizzare quella che è la funzione principale del progetto, ovvero la simulazione degli acquisti da parte degli utenti. La seconda invece viene utilizzata per costruire la tabella dei prodotti che appare sugli scontrini. La prima in particolare permette di effettuare la chiamata del metodo *Acquista* senza preoccuparsi della specifica tipologia di utente, in quanto, grazie al polimorfismo, la funzione che verrà chiamata andrà ad effettuare autonomamente le operazioni adatte alla tipologia di utente sulla quale viene invocata. Nel secondo caso invece il polimorfismo permette di fare in modo che la funzione che costruisce la tabella dei prodotti contenuti in uno scontrino non si debba preoccupare del tipo del prodotto per poterne stampare il nome, ma si limiti ad invocare il metodo virtuale che è stato appropriatamente ridefinito nelle diverse classi della gerarchia.

3. Architettura

Verrà ora illustrata l'architettura ad alto livello del progetto, per mostrare le connessioni tra le principali componenti del programma. In particolare, si aderisce al pattern MVC, mantenendo separate le diverse componenti: Model, View e Controller.

3.1 Interazione tra le componenti

Il progetto è stato sviluppato cercando di mantenere costanti e separati i ruoli delle diverse componenti, isolando le diverse parti. Il model fa operazioni e gestisce i dati, il controller richiama le operazioni del model e segnala i cambiamenti, e la vista richiede operazioni al controller e si occupa solo della parte di visualizzazione. Il model è rappresentato dalla classe *Database*, che si occupa di istanziare ed effettuare le operazioni sulle diverse tipologie di dati che verranno poi utilizzate dal programma. Il controller è rappresentato dalla classe *Controller*, che possiede un puntatore ad un database attraverso il quale richiede le operazioni sui dati. Il controller si occupa anche di segnalare tramite signal eventuali cambiamenti di stato del modello alle componenti di vista interessate. Il controller è stato volutamente sviluppato senza slot, e non si preoccupa di chi riceverà i suoi signal: è infatti compito delle *View* interessate andare a collegarsi ai segnali del controller che le riguardano. Questo procedimento viene sfruttato per esempio per aggiornare le tabelle dei prodotti delle diverse pagine. Il controller si occupa solo di mandare un segnale *LibriChanged* o *DVDChanged* quando gli viene richiesto di effettuare operazioni che modificano le rispettive tipologie dei prodotti, e spetta alle viste interessate collegarsi a tale segnale e andare a ridisegnare le tabelle. Questo permette potenzialmente di avere contemporaneamente diverse *View* collegate allo stesso controller, che è unico. Questa scelta progettuale permette inoltre di favorire l'estendibilità della *View* senza andare a modificare le altre componenti del progetto. La view è stata costruita tramite cinque classi, quattro delle quali dispongono di un puntatore al controller utilizzato per interagire con il modello. La quinta è una semplice classe di utilità sfruttata per caricare e

salvare i dati sul database. La classe principale della view si chiama *MainWindow* e si occupa di istanziare la finestra principale del programma. Essa infatti possiede solo un TabWidget all'interno del quale va ad inserire le diverse pagine costruite dalle altre classi. Compito della *MainWindow* è anche quello di occuparsi della gestione degli errori. Essa dispone infatti di uno specifico slot al quale si collegano i messaggi di errore, che risultano così standardizzati e consistenti indipendentemente dalla pagina in cui ci si trova. La classe *MainWindow* si occupa anche di invocare i costruttori delle altre tre classi che compongono la view e passare loro il puntatore al controller, in modo che possano interagire con il model. Le altre tre classi sono: *ProdottiView, UtentiView* e *AcquistiView.* In base alle operazioni che vengono loro richieste le diverse classi utilizzano il loro puntatore per invocare metodi o richiedere oggetti al controller, il quale si limita a ritornare gli oggetti o effettuare le operazioni senza preoccuparsi dell'autore della richiesta, rendendolo così effettivamente indipendente dall'implementazione della vista che è stata realizzata.

3.2 Layout della vista

Il layout della vista è gestito da quattro classi: *MainWindow, ProdottiView, UtentiView* e *AcquistiView*. La classe *MainWindow*, derivata pubblicamente da *QMainWindow,* si occupa di invocare i costruttori delle altre tre classi che, essendo derivate pubblicamente da *QWidget,* possono poi venire inserite come diverse pagine all'interno del TabWidget costruito dalla *MainWindow.* La prima pagina è *ProdottiView.* Essa si occupa di gestire la parte di vista che si occupa dei prodotti. Possiede dunque anch'essa un TabWidget le cui due pagine sono dedicate una ai libri e l'altra ai dvd. Entrambe le pagine sono divise a metà da uno splitter, con la parte sinistra dedicata al display di una tabella contenente la lista di tutti i prodotti del tipo appropriato, e la parte destra contenente un form utilizzato per costruire nuovi oggetti. La tabella di sinistra possiede come ultimo campo dati per ogni prodotto un bottone che ne permette la rimozione. La seconda pagina è gestita da *UtentiView.* La sua struttura è pressoché identica a quella di *ProdottiView*, divisa da uno splitter con una tabella per la visualizzazione e la rimozione degli utenti sulla sinistra e un form per l'aggiunta sulla destra. L'ultima pagina, gestita da *AcquistiView*, si occupa della creazione dello scontrino e del successivo acquisto da parte dell'utente. Essa presenta due tabelle, quella di sinistra elenca tutti i codici e i nomi dei prodotti, mentre quella di destra rappresenta la lista di prodotti aggiunta allo scontrino. Tra le due tabelle sono presenti dei LineEdit con dei PushButton utilizzati per aggiungere e togliere prodotti dallo scontrino. Sotto le tabelle sono presenti altri due LineEdit nei quali inserire il numero dei punti spesi per l'acquisto e il codice dell'utente che vuole eseguirlo nel caso sia appropriato. L'ultimo elemento della pagina è un PushButton che completa l'acquisto. Questo porta alla costruzione di una seconda pagina che presenta i risultati dell'acquisto tramite una rappresentazione dello scontrino, completa di elenco dei prodotti, prezzo, sconto e tutte le altre informazioni necessarie. Questa seconda pagina presenta anche un bottone che permette di ritornare alla costruzione di un nuovo scontrino, svuotando quello precendente tramite la funzione *Wipe* di scontrino.

4. Manuale Utente

Verrà qui brevemente spiegato come utilizzare le diverse funzionalità del programma.

4.1 Gestione prodotti

Il programma fornisce la possibilità di aggiungere e rimuovere libri e dvd. Queste funzionalità sono gestite dalla pagina prodotti. Sulla destra della pagina è presente un form nel quale andare ad inserire i dati. Il codice dei prodotti deve essere un numero intero, e deve essere unico per ogni prodotto, in caso si inserisca un codice non valido o già utilizzato apparirà un messaggio di errore. I codici sono indipendenti dal tipo del prodotto, quindi non si può avere un libro e un dvd con lo stesso codice. Per ottenere un valore valido nel campo data è necessario inserirla nel formato

dd.mm.yyyy. Per rimuovere un prodotto basta premere il pulsante corrispondente che appare come ultimo campo nella tabella di sinistra.

4.2 Gestione utenti

Le funzionalità di inserimento e rimozione degli utenti sono pressoché identiche a quelle per i prodotti. Il codice deve essere unico per ogni utente, tuttavia in questo caso è possibile inserire un codice non numerico. E' inoltre necessario selezionare una tipologia di utente prima dell'aggiunta, non farlo porterà ad un messaggio di errore.

4.3 Gestione acquisti

L'ultima pagina è riservata agli acquisti. Sulla sinistra abbiamo una tabella che mostra i nomi e i codici di tutti i prodotti presenti sul database. Sulla destra invece abbiamo la lista di prodotti che sono stati attualmente aggiunti allo scontrino. Per aggiungere un prodotto allo scontrino basta inserire il codice nella LineEdit sopra il bottone *aggiungi prodotto* e premere il bottone. Per rimuovere un prodotto dallo scontrino basta effettuare la stessa operazione con il bottone *rimuovi prodotto.* E' possibile aggiungere più volte lo stesso prodotto allo scontrino; in caso siano presenti più copie del medesimo prodotto esse dovranno essere tolte una alla volta. Sotto alle tabelle sono presenti due LineEdit: compilare questi campi è opzionale e dipende da che tipo di operazione si vuole eseguire. Se si intende effettuare l'acquisto come *Standard_User* allora non è necessario compilare alcun campo e basta premere il bottone acquista. Se invece si vuole effettuarlo come *Premium_User* o *Employee_User* allora è necessario inserire nel campo apposito il codice dell'utente che vuole effettuare l'acquisto. Nel campo punti si andrà invece ad inserire la quantità di punti che l'utente desidera spendere per quel determinato acquisto. Inserire un codice che non appartiene a nessun utente o un numero di punti maggiore di quello disponibile all'utente selezionato darà origine ad un messaggio di errore.