BiblioTech

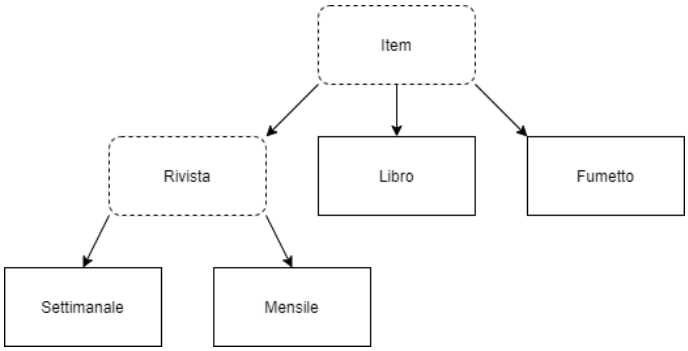
Progetto del corso di Programmazione ad Oggetti
A.A. 2020/2021
Matteo Tossuto 1193493

Indice

1 Introduzione

L'idea del progetto è di realizzare un preventivatore che possa essere utilizzato dalle librerie e/o edicole per il noleggio e la vendita di libri, fumetti e riviste. Viene data la possibilità di creare un preventivo, come PDF, al cliente, con la spesa totale dei prodotti acquistati o la spesa parziale(al giorno) dei prodotti noleggiati. Il preventivo, inoltre, riporta il titolo, il genere e il prezzo di ogni singolo prodotto. Il programma permette di creare dei cataloghi(in xml) che possono essere salvati e ricaricati all'avvio o durante l'utilizzo del programma.

2 Gerarchia



Lo scopo è stato di creare una gerarchia abbastanza generalizzata in caso si dovesse espanderla o modificarla su richiesta dei clienti. La classe base *Item* è astratta e contiene gli attributi generali dei prodotti, ossia: Titolo, Genere, Prezzo e PrezzoNoleggio. Dalla classe base derivano altre tre classi:

- **Libro:** una classe che rappresenta i libri e i suoi attributi sono: Autore, Anno Edizione ed Editore;
- **Fumetto:** una classe che rappresenta i fumetti e i suoi attributi sono: Autore, Numero edizione ed Editore;
- **Rivista:** una classe astratta che generalizza le riviste e i suoi attributi sono: Numero Edizione ed Editore;

La classe astratta *Rivista*, viene utilizzate per suddividere le riviste in *Settimanale* e *Mensile*.

3 Container

Siccome per rappresentare i prodotti viene utilizzato un catalogo è stato deciso di creare un template di classe che rappresenta un Container con una struttura simile a quella di una lista singolarmente linkata.

Viene utilizzata una classe privata nodo(campi: info e next) per definire i nodi della lista. Sono stati definiti costruttori e distruttore.

Oltre a questi metodi ne sono stati implementati altri per gestire le diverse funzioni di una lista:

- **copy:** crea una copia della lista;
- **destroy:** distrugge un nodo singolo;
- **search:** funzione per cercare un nodo;
- **isEqual:** funzione per comparare liste e/o sottoliste;

Inoltre sono stati implementati metodi per le normali funzioni di una lista come: creazione, rimpiazzo, ricerca o rimozione di nodi, funzioni per il ritorno della posizione di un determinato nodo o la dimensione della lista.

È stato fatto l'overload dei seguenti operatori:

- Operatore =
- Operatore ==
- Operatore !=

4 Deep Pointer

È stato deciso di creare un template di classe per l'implementazione di un Deep Pointer. Sono stati implementati i costruttori(default e copia) e il distruttore, inoltre è stato fatto l'overload dei seguenti operatori: =, ==, !=, <, >, * e -> .

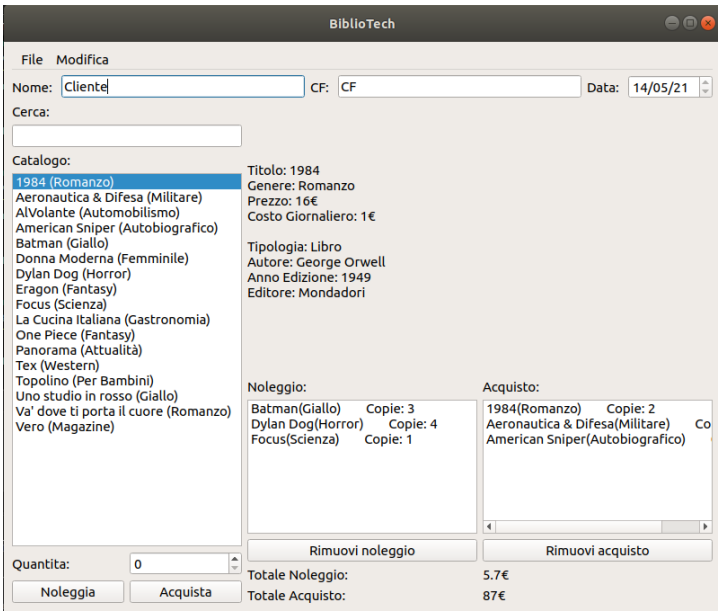
5 GUI

5.1 Finestra Principale

Qui troviamo, in altro a sinistra, due menù:

- **File:** contiene i tasti per caricare e salvare i cataloghi, creare un preventivo come PDF e uscire dal programma;
- **Modifica:** contiene i tasti per aggiungere, modificare o rimuovere un prodotto dal catalogo;

Appena al di sotto dei menu troviamo tre label modificabili usati per riportare il Nome e il CF del cliente e la data nell'eventuale preventivo.
A sinistra troviamo una finestra che riporta *Titolo* e *Genere* dei prodotti del catalogo. Al di sopra troviamo la barra di ricerca. Al di sotto troviamo una label modificabile tramite tastiera o bottoni per modificare la quantità dei prodotti da noleggiare o acquistare, scelta che verrà effettuata tramite apposito tasto(Noleggio o Acquisto).
Al centro vengono riportati i dettagli dell'oggetto selezionato.
Nella parte bassa troviamo due riquadri che rappresentano rispettivamente il carrello di noleggio e di acquisto e riportano *Titolo*, *Genere* e quantità dei prodotti desiderati. Al di sotto di essi troviamo un bottone(per carrello) per l'eventuale rimozione di un prodotto. Ancora più sotto vengono riportati la spesa totale e quella parziale.

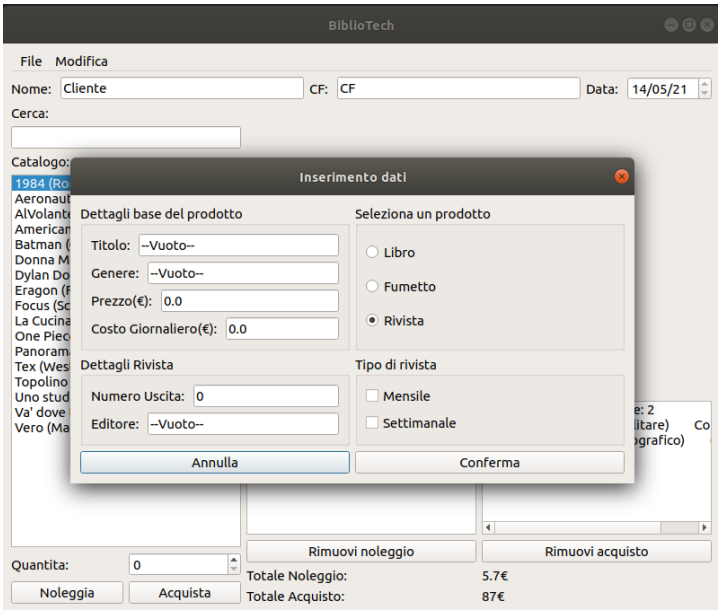


5.2 Finestra di Inserimento

Quando si preme il tasto "Aggiungi un elemento al catalogo" dal menù *Modifica*, viene aperta una nuova finestra per l'inserimento dei dati del nuovo prodotto. La nuova finestra si dividerà in tre/quattro parti.

- La prima parte è per l'inserimento degli attributi generali: *Titolo*, *Genere*, *Prezzo* e *PrezzoNoleggio*;
- La seconda parte è per la selezione del tipo del prodotto, ossia se sarà un *Libro*, un *Fumetto* o una *Rivista*. In caso venga selezionata la scelta *Rivista* si aprirà una quarta parte nella finestra;
- Dopo aver selezionato il tipo di prodotto comparirà un nuovo settore della finestra che permetterà di inserire gli attributi del tipo di prodotto selezionato;
- Se sarà stata scelta una *Rivista* comparirà un nuovo settore delle finestra che permetterà di scegliere tra *Settimanale* o *Mensile* per decidere il tipo di *Rivista*.

Se si cerca di inserire un prodotto già esistente si verrà avvisati dal programma con un messaggio di "Prodotto già esistente".
Se si cerca di inserire un prodotto senza averne deciso il tipo o , in caso di *Rivista*, senza averne deciso la periodicità si verrà avvisati dal programma con un messaggio di "Input Incompleto".
Gli attributi *Prezzo* e *PrezzoNoleggio* devono rispettare il formato "00.0" o "00.00", in caso contrario se si proverà a inserire un prodotto che usa un formato diverso si verrà avvisati dal programma con un messaggio di "Input Errato" e la finestra di modifica viene chiusa.



5.3 Finestra di Modifica

Una volta selezionato un elemento dal catalogo e premuto il tasto "Modifica Elemento" dal menù di *Modifica* si aprirà una finestra simile a quella di inserimento. Questo perché la finestra di modifica è figlia della finestra di inserimento. Le differenze sono le seguenti:

- Quando la finestra si apre vengono caricati i dati dell'oggetto modificato;
- Una volta confermata la modifica, viene eliminato il vecchio prodotto e inserito quello nuovo.

6 Lista smart

Questa classe viene utilizzata per facilitare alcuni metodi usati nella classi per la *GUI*.

7 XMLIO

Questa classe si occupa del caricamento e salvataggio dei cataloghi in formato *XML* nella cartella *cataloghi*.

8 Carrello

Questa classe viene utilizzata per connettere il Modello(model) alla gerarchia dei prodotti.

9 All'avvio

All'avvio del programma non saranno presenti prodotti nel catalogo e spetterà all'utente decidere se aggiungere nuovi prodotti manualmente o caricare un file di catalogo(ne è già presente uno nella cartella cataloghi).

10 Preventivo

Come già detto nell'introduzione è possibile creare un preventivo salvandolo come file PDF.
L'applicazione prima di creare il PDF esegue dei controlli sui seguenti campi dati:

- **Nome Cliente:** il programma controlla che la label non sia vuota o sia ancora presente la dicitura "Cliente";
- **CF:** il programma controlla che la label non sia vuota o che il *CF* sia della giusta lunghezza, ossia 16 caratteri alfanumerici.

11 Polimorfismo

Siccome il container utilizza molto oggetti del tipo DeepPtr<Item> vengono effettuare numerose operazioni polimorfe come l' == o il !=. Inoltre sono presenti i seguenti metodi polimorfi:

- Il metodo *clone*, utilizzato per la copia profonda;
- Il metodo *getTipo*, utilizzato per ritornare il tipo del prodotto;
- Il metodo *print*, utilizzato per ritornare e stampare i dettagli dell'oggetto selezionato;
- Il metodo *serializzaDati*, utilizzato per la serializzazione dei dati.

12 Dati Tecnici

12.1 Analisi delle Ore di Lavoro

- Analisi Specifiche Progetto: 1 ora;
- Progettazione e creazione della gerarchia: 10 ore;
- Progettazione e creazione GUI: 11 ore;
- Progettazione e creazione MVC: 21 ore;
- Test e Debug: 5 ore;
- Stesura Relazione: 2 ora;

Totale Ore: 50

12.2 Ambiente di Lavoro

Il progetto è stato sviluppato sulla VM così fornita. È stato testato sulla VM, sui computer del laboratorio dell'università tramite commando "SSH" e sul computer personale.

12.3 Modalità di Consegna

All'interno del file zip consegnato è stato incluso il file .pro generato automaticamente da QT. Se si prova a eseguire il commando qmake -project, qmake e make ci saranno dei problemi che impediranno la corretta esecuzione del commando make e non verrà generato l'eseguibile.
Se invece si utilizza il commando qmake e make utilizzando il .pro fornito non ci saranno problemi. Inoltre, avendo fornito il file .pro, è possibile eseguirlo tramite Qt Creator.

12.4 Problemi Conosciuti

Se si utilizza l'applicazione tramite Qt Creator usando il file .pro fornito non ci saranno problemi e funzionerà tutto perfettamente. Se invece, dopo aver eseguito i comandi qmake e make, si avvia l'eseguibile questo crashera se si tenta di carica un file o aggiungere un prodotto. Questo crash avviene per segmentation fault e quindi è molto probabile che ci siano delle varibili che non vengono inizializzate correttamente. Non sono riuscito a risolvere questo problema, chiedo scusa.