




29 AGOSTO 2017

RELAZIONE PROGETTO PROGRAMMAZIONE AD OGGETTI

DENIS MAZZUCATO

1122251



Sommario

Scopo del Progetto	2
Gerarchie di Tipo	2
Gerarchie di Tipo per il Modello.....	2
Gerarchie di Tipo per la GUI.....	3
Codice Polimorfo del Modello.....	4
Utente: getGrado	4
Utente: hash	4
Utente: print e input	4
Utente: search	4
Utente: isAdmin e isUtenteAvanzato	4
Utente: clone.....	4
Libro: getGrado, hash, input, print e clone	5
Codice Polimorfo della GUI	5
RightBox: setItem	5
LeftBox: deleteAll, deleteItem, updateList	5
Manuale Utente GUI	5
Indicazione Ore	6
Specifiche	6

Scopo del Progetto

Il progetto in esame consiste nel gestire una biblioteca dove utenti di diversa natura possono accedere (previa registrazione) e visualizzare, piuttosto che restituire o prendere in prestito i libri offerti dalla biblioteca in base al loro tipo di registrazione, inoltre l'applicazione offre possibilità di ricerca dei libri e di modifica dei dati personali.

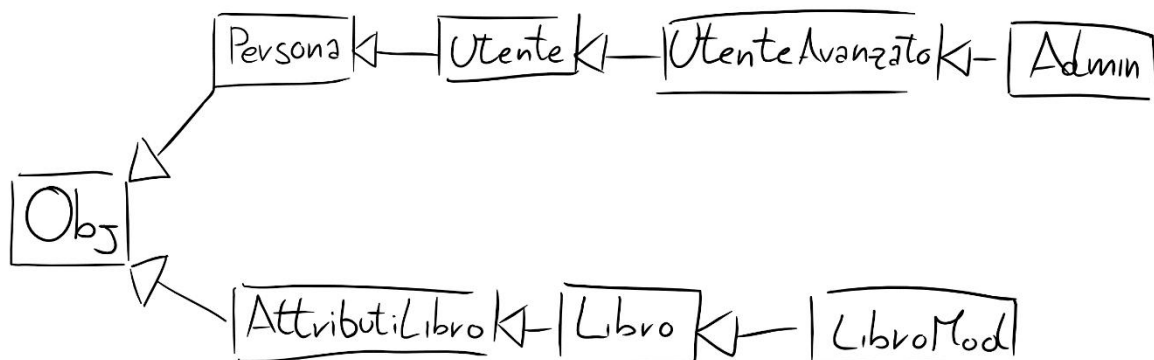
Per gli utenti saranno disponibili due tipi di registrazione, quella base e quella premium, la registrazione premium comporta l'inserimento di una carta di credito per il pagamento del canone e l'inserimento di un grado personalizzato, questo grado, più alto sarà, fino ad un massimo di 5, più libri si potranno visualizzare dal contenitore, l'amministratore è un'utente avanzato senza canone con grado massimo.

Anche i libri si suddividono tra libri base, ovvero a grado minore (0), e libri premium caratterizzati dal grado, i libri visualizzabili, dal contenitore, per un utente x sono solo quelli che hanno il grado(libro) minore o uguale al grado(utente) di x .

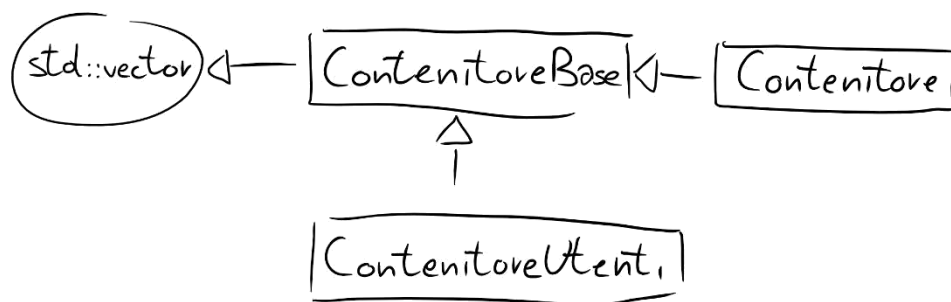
L'amministratore inoltre ha poteri su tutto l'applicativo potendo aggiungere, modificare o eliminare libri dal sistema e eliminare utenti, oltre ad essere l'unico a poter aggiungere altri amministratori, questa figura nell'ambito della biblioteca può essere vista come bibliotecario.

Gerarchie di Tipo

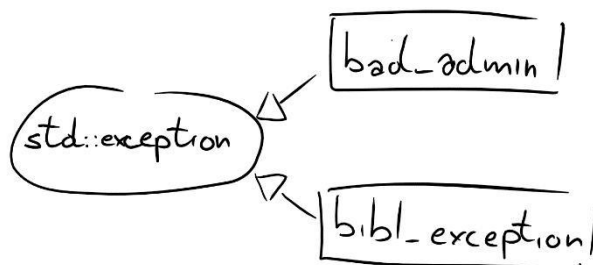
Gerarchie di Tipo per il Modello



- **Obj**, classe base a livello più alto, definisce un'interfaccia comune per tutte quelle classi che definiranno oggetti del progetto, in essa è definito il distruttore di default virtuale.
- **Persona**, sottoclasse di **Obj**, definisce un oggetto di persona, qui si racchiudono tutti i parametri aggiuntivi di un' **Utente**.
- **Utente**, sottoclasse di **Persona**, definisce un'utente base definito da username e password, campi che definiscono univocamente un'utente nel sistema, l' **Utente** è il tipo a livello più basso (0) di registrazione possibile, l'operatore di uguaglianza ritorna vero se e solo se username e password hanno un match completo, in **Utente** è definita anche la lista dei libri in prestito dall'utente, inizializzata in maniera furba, ovvero solo quando richiesta, con tutti i metodi richiesti per la sua gestione, infine viene anche definito un metodo virtuale di ricerca molto banale per match completo tra la chiave di ricerca e titolo o autore del libro.
- **UtenteAvanzato**, sottoclasse di **Utente**, aggiunge ad un' **Utente** il campo grado che corrisponde al livello gerarchico all'interno della biblioteca (min= 1, max= 5), e il campo per il numero di carta di credito, servirà poi per far pagare un canone per questo tipo di registrazione, implementata nuovamente la funzione di ricerca per dare una profondità maggiore alla ricerca dei libri.
- **Admin**, sottoclasse di **UtenteAvanzato** a livello minore nella gerarchia di utenti, definisce un **UtenteAvanzato** a livello massimo e senza il codice della carta di credito.
- **AttributiLibro**, sottoclasse di **Obj**, definisce attributi secondari per un **Libro**.
- **Libro**, sottoclasse di **AttributiLibro**, definisce un libro mediante la coppia autore libro, come per l' **Utente** due **Libro** o sottotipi ritornano vero, mediante l'operatore di uguaglianza, se e solo se autore e titolo hanno un match completo, in essa è racchiuso inoltre il campo per l'informazione del numero di copie disponibili.
- **LibroMod**, sottoclasse di **Libro** a livello minore nella gerarchia dei libri, aggiunge al libro il suo grado, ovvero il grado minimo che un **Utente**(o **UtenteAvanzato**) deve avere per avere accesso ad esso, un **Libro** di default ha grado 0 mentre un **LibroMod** può arrivare da 1 fino al grado 5.



- **ContenitoreBase**, sottoclasse templetizzata di `std::vector` che definisce funzionalità aggiuntive utili ad un contenitore per un tipo `*T`.
- **Contenitore**, sottoclasse di `ContenitoreBase<T=Libro>` serve ad implementare l'aggiornamento da/a file del contenitore, nel costruttore legge i libri dal file mentre nel distruttore aggiorna il file, solo se serve, inoltre sono definiti metodi per bloccare o modificare questo aggiornamento automatico (definiti tra `Contenitore` e `ContenitoreBase`).
- **ContenitoreUtenti**, sottoclasse di `ContenitoreBase<T=Utente>`, duale a `Contenitore` ma per gli utenti.



- **bad_admin**, sottoclasse di `std::exception` che definisce l'eccezione chiamata quando ci si aspetta un'utente *Admin* ma si trova un diverso tipo di utente.
- **bibl_expected**, sottoclasse di `std::exception`, usata più nella parte *fe* (front-end) dove si richiede obbligatoriamente (un puntatore a) *Biblioteca*.
- **global.h::std**, namespace che racchiude varie funzionalità che gestiscono l'hashing degli oggetti, la crittografia della password, l'eliminazione di file ecc..
- **Biblioteca**, classe che definisce tutti i metodi che gestiranno la biblioteca, al suo interno si trovano il riferimento del `Contenitore` principale dei libri e il riferimento all'utente loggato.

Gerarchie di Tipo per la GUI

- **ApplicationCore**, sottotipo di `QObject`, in ordine è la prima ad essere chiamata dal `main.cpp`, racchiude l'oggetto *Biblioteca*, gestisce inoltre il cambio di finestre.
- **BWidget**, sottotipo di `QWidget`, definisce un `QWidget` a cui serve il riferimento alla *Biblioteca*.
- **Window**, sottotipo di `BWidget`, gestisce il concetto di finestra, quindi non avrà nel costruttore il passaggio del parent come specificato nella doc di Qt per i `QWidget` finestra-
- **LoginWindow, RegWindow, InfoWindow**, sottoclassi di `Window`, che definiscono le varie interfacce banali per la GUI.
- **MainWindow**, anch'essa sottoclasse di `Window`, ma definisce la finestra principale dell'applicativo, come spiegato nel *manuale della GUI* più avanti, gestisce le tre fasce (*Frame*) principali.
- **SearchBox, SetBookData, SetUserData**, sottoclassi (secondarie) di `Window`, dentro risiedono tutte le interfacce secondarie spostate in finestre esterne per non appesantire troppo la visuale della *MainWindow*.
- **UserFrame, BookFrame**, sottoclassi di `BWidget`, che definiscono rispettivamente la parte GUI dell'utente e la GUI per i libri.
- **AdminFrame**, sottoclasse di `BWidget`, definisce la GUI dell'*admin zone*, divisa in due fasce, la superiore delle impostazioni utente composta dai due *Box*: *LeftUserBox* e *RightUserBox*, mentre la fascia inferiore delle impostazioni libro composta anch'essa dai due *Box*: *LeftBookBox* e *RightBookBox*.
- **LeftBox**, sottoclasse astratta di `BWidget`, che definisce un *Box* a sinistra, ovvero composta da una lista di elementi.

- **RightBox**, sottoclasse astratta di *BWidget*, che definisce un *Box* a destra, composto da un *Form* per l'aggiunta di elementi.
- **LeftUserBox**, **LeftBookBox**, sottoclassi concrete di *LeftBox*, definiscono rispettivamente l'aggiornamento della lista con i "giusti" parametri e le azioni da eseguire in caso di click (doubleClick) su elementi.
- **RightUserBox**, **RightBookBox**, sottoclassi concrete di *RightBox*, definiscono i *Form* da utilizzare nei *Box* "a destra" e le azioni da eseguire per l'oggetto costruito dentro al *Form*.
- **Form**, sottoclasse di *QWidget*, definita per avere una base comune alle derivazioni di *UserForm* e *BookForm*.
- **UserForm**, **BookForm**, sottoclassi di *Form*, gestiscono la GUI del modulo per la costruzione di un utente e un libro rispettivamente.
- **Message**, classe che esegue un *QMessageBox*.
- **WGlobal**, classe che definisce i metodi di "conversione" da (puntatore a) *Utente* a *QString* e da (puntatore a) *Libro* a *QString*, oltre a definire la misura usata per alcuni *QWidget* a cui serve una lunghezza minima per una GUI migliore.

Codice Polimorfo del Modello

Utente: getGrado

Questo metodo ritorna il grado di un'utente, cioè il grado minimo per un qualsiasi utente ossia 0, mentre se un utente è un *UtenteAvanzato* ritorna il suo grado, compreso tra 1 e 5, per gli admin infine ritorna 5, il massimo disponibile.

Viene utilizzato quando si ha a disposizione un puntatore ad *Utente* come in *wglobal.cpp* linea 8.

Utente: hash

Questo metodo ritorna l'hash dell'utente, ovvero un numero identificativo con tutte le proprietà di un hash, sfruttando le funzionalità di c++11, lo ritorna in base al tipo dinamico di quest'ultimo, se *UtenteAvanzato* ritornerà l'hash dell'*UtenteAvanzato*, mentre se *Admin* ritorna l'hash del sotto oggetto di tipo *UtenteAvanzato* dato che non hanno parametri differenti.

Viene utilizzato nel *ContenitoreBase* [*T=Utente*] nella funzione che calcola l'hash complessivo del vettore, *ContenitoreBase.h* linea 42.

Utente: print e input

Questi due metodi servono, rispettivamente, per l'output e input da file, selezionati in base al tipo dinamico, dato che *Utente*, *UtenteAvanzato* e *Admin* hanno diversi formati di scrittura e lettura.

Viene utilizzato nel *ContenitoreUtente*, nel costruttore che nel distruttore, (*print*) *ContenitoreUtente.cpp* linea 25, (*input*) *ContenitoreUtente.h* linea 23.

Utente: search

Questo metodo serve a ritornare un insieme di libri generati dalla ricerca in una lista di libri, l'insieme di ritorno sarà differente in base al tipo dinamico del puntatore su cui è chiamato, per permettere una ricerca migliore a *UtenteAvanzato*.

Viene utilizzata in un metodo di *Biblioteca* che passa in automatico la lista corretta, *Biblioteca.cpp* linea 130, mentre è richiamata dalla GUI in *SearchBox.cpp* linea 23.

Utente: isAdmin e isUtenteAvanzato

Questi metodi ritornano true se il tipo dinamico corrisponde all'etichetta del metodo, per un *Admin* *isAdmin* ritornerà true e per un *UtenteAvanzato* *isUtenteAvanzato* ritornerà true.

Vengono utilizzati in *ContenitoreUtenti*, e non solo, per sapere in quale file scrivere in base al tipo dinamico, *ContenitoreUtenti.cpp* linee 19 e 21.

Utente: clone

Questo metodo ritorna il puntatore ad una copia di **this*.

Utilizzato quando inserisco nel *ContenitoreUtenti* un puntatore ad un nuovo *Utente* e non voglio interferenze tra puntatori, la distruzione dell'oggetto creato porterebbe alla distruzione dell'oggetto inserito all'interno del contenitore, oltrech  avere un puntatore ad un'oggetto di tipo dinamico corretto per il contesto, *ContenitoreBase.h* linea 86.

Libro: getGrado, hash, input, print e clone

Questi metodi sono duali ai casi precedentemente spiegati per l'*Utente*.

Esempi d'utilizzo si possono trovare in *wglobal.cpp* linea 21 (getGrado), per *[T=Libro]* *ContenitoreBase.h* linea 42 (hash), *Contenitore.cpp* linea 17 (input), *Contenitore.cpp* linea 46 (print) e per *[T=Libro]* *ContenitoreBase.h* linea 86 (clone).

Codice Polimorfo della GUI

RightBox: setItem

Questo metodo   virtuale puro, al momento non se ne conosce l'implementazione, delegata alle varie sottoclassi concrete *RightBookBox* e *RightUserBox*, per  c'  bisogno di esso per connetterlo all'oggetto creato dal modulo all'interno del box a destra.

Utilizzato in *RightBox.cpp* linea 16.

LeftBox: deleteAll, deleteItem, updateList

Questi tre metodi servono ad eseguire una distruzione, dal contenitore principale o direttamente nei file, su uno o pi  oggetti oppure di aggiornare la lista per una visualizzazione corretta; per *deleteItem* e *deleteAll* servono direttamente nel costruttore di *LeftBox* per *updateList* serve la virtualizzazione dato che in *AdminFrame*, avr  a disposizione un puntatore a *LeftBox*, con tipo dinamico *LeftBookBox* o *LeftUserBox*.

Utilizzati in *LeftBox.cpp* linea 6, 13 e *AdminFrame.cpp* linea 26/27.

Manuale Utente GUI

L'applicazione tra il login e l'avvio mostra una finestra informativa per 5/8/10 secondi, a seconda della quantit  di informazioni che deve mostrare, per facilitare l'uso dell'applicazione, se premuto invio si passa direttamente alla finestra successiva.

L'interfaccia grafica, escluse le finestre secondarie come quelle per il login o registrazione molto intuitive, si suddivide in un'unica finestra primaria sviluppata in due regioni, per utenti base e non, mentre si aggiunge un'ulteriore fascia se si accede al sistema come amministratore.

La prima area a sinistra racchiude la parte utente, mostrando i dati dell'utente acceduto e 4 pulsanti il cui che consentono, in ordine, di restituire tutti i libri, modificare i propri dati, uscire ed infine di eliminare il proprio account.

La parte centrale invece mostra la lista di tutti i libri a cui abbiamo accesso mediante il contenitore e la lista di tutti i libri ancora in nostro possesso, pi  sotto troviamo inoltre la barra di ricerca che permette di effettuare una ricerca sul contenitore, ricerca molto scadente se effettuata da un utente base invece molto migliore se effettuata dall'amministratore o da un'utente avanzato, i libri a cui un utente ha accesso son quelli con grado minore o uguale al grado dell'utente in questione, solo questi saranno visualizzati, se disponibili in quantit .

Infine la zona destra, la admin zone, è suddivisa in due fasce, quella superiore per la gestione degli utenti mentre quella inferiore per la gestione dei libri, nella fascia superiore troviamo una lista contenente gli utenti, utenti avanzati e amministratori del sistema che se premuti si potranno eliminare, e un modulo che permette la registrazione degli admin, non registrabili mediante il classico modulo di registrazione iniziale, si noti che esiste (finché non eliminato da un altro admin) un **admin di default che ha come username "admin" e come password "admin"**, questo amministratore viene creato ad ogni avvio del applicativo se non presente; nella fascia inferiore invece ci si occupa della gestione dei libri dove a sinistra la lista contenente i libri, che se premuti si dovrà decidere se eliminare o modificare, mentre a destra il modulo per l'inserimento di nuovi libri.

spesso i bottoni usati più frequenti sono stati settati come `setAutoDefault(true)` quindi con un `tab` e poi `enter` si simulerà il click del mouse.

In tutti i moduli presenti all'interno dell'applicazione tutti i loro campi sono a compilazione obbligatoria (se non specificato diversamente) altrimenti all'invio si perderanno i campi già compilati all'interno del modulo.

Il progetto sarà inizialmente vuoto se però si preferisce accedere con un sistema già avviato basterà spostarsi dentro `"files/"` e rinominare il file `"~libri.txt"` in `"libri.txt"`.

Indicazione Ore

Progettazione Modello: 4h

Codifica Modello: 25h

Progettazione GUI: 2h

Codifica GUI: 20h

Debugging e Testing: 5h

Tot: 60h

Specifiche

Il progetto è stato sviluppato su windows 10 Pro x64 (Microsoft Surface Pro 4 con risoluzione 2736x1824), compilatore GCC(MinGW) versione 5.3.0 e libreria Qt alla versione 5.5.1.