

UNIVERSITÀ DEGLI STUDI DI PADOVA

LAUREA IN INFORMATICA

CORSO DI PROGRAMMAZIONE AD OGGETTI

Progetto di fine corso

Studente:
Carlo SINDICO

Matricola:
1069322

3/02/2017

LOGIN AMMINISTRATORE:

USERNAME: *admin*
PASSWORD: *admin*

1 Ambiente di sviluppo

- Versione del compilatore: GCC 4.9.2 32bit
- Sistema operativo utilizzato: Windows 10 (test funzionamento effettuati anche con Ubuntu 16.04LTS)
- Versione delle librerie Qt: 5.6.2 32bit

2 Ore lavorative

- Sono state spese 35 ore per la parte relativa alla logica del progetto
- Sono state spese 25 ore per la parte relativa all'interfaccia grafica, test e debugging

3 Scopo del progetto

- Il progetto ha lo scopo di fornire un semplice ed intuitivo sistema per l'amministrazione, attraverso una interfaccia grafica, di una biblioteca. La biblioteca contiene opere che sono caratterizzate da un titolo e da un identificativo univoco, ed ogni opera può essere di due tipi: libro o rivista.
- Un libro è una particolare opera caratterizzata da un autore, mentre invece una rivista è caratterizzata da un anno di uscita.
- Le opere della biblioteca possono venire prestate agli utenti, ogni opera contiene l'informazione relativa al suo stato se essa è disponibile al prestito e se presente nella biblioteca. Per quanto riguarda i libri non vi è alcun vincolo sul prestito, invece le riviste che hanno un anno di uscita superiore a 20 anni non possono essere prestate né all'utente basic né all'utente pro.
- La biblioteca fornisce un'area utente, attraverso la quale ogni utente può prendere in prestito le opere. (possibilità di visualizzare le opere presenti nella biblioteca, e contemporaneamente quelle prese in prestito). L'utente attraverso questa interfaccia può ricevere e restituire le opere.
- Un utente è caratterizzato da un nome, un cognome, un codice fiscale, un identificativo univoco, una password e il numero di opere che ha in prestito. Un utente basic è un particolare utente, caratterizzato da un numero di opere che può ricevere in prestito, fissato a 5. Un utente pro è un particolare utente, caratterizzato da un numero di opere che può ricevere in prestito, fissato a 8.
- Andremo ad analizzare successivamente più in dettaglio la gerarchia opera.

3.1 Funzionalità disponibili all'amministratore

- l'amministratore attraverso il sistema deve essere in grado di eseguire semplici azioni quali:
 1. Visualizzare tutte le opere presenti nella biblioteca
 2. Visualizzare i dettagli di tutte le opere presenti nell'archivio, ed eventualmente di modificarne i parametri quali titolo autore o anno di uscita, a seconda che sia un libro o una rivista
 3. Visualizzare tutti gli utenti registrati alla biblioteca
 4. Visualizzare i dettagli di tutti gli utenti presenti nell'archivio, ed eventualmente di modificarne i parametri quali nome e cognome
 5. Aggiungere un'opera per aumentare l'offerta concessa dalla biblioteca
 6. Eseguire una ricerca di un'opera per titolo
 7. Eseguire una ricerca di un utente per nome
 8. Rimuovere un'opera
 9. Aggiungere un utente all'archivio della biblioteca
 10. Rimuovere un utente solo se ha restituito tutte le opere prese in prestito

3.2 Funzionalità disponibili all'utente

- l'utente attraverso il sistema deve essere in grado di eseguire semplici azioni quali:
 1. Visualizzare tutte le opere presenti nella biblioteca
 2. Eseguire una ricerca di un'opera per titolo
 3. Ricevere in prestito secondo i limiti descritti precedentemente libri e riviste
 4. Restituire alla biblioteca libri e riviste

4 Descrizione della gerarchia costituita da opera e dai suoi sottotipi

- La classe opera alla base della gerarchia ed caratterizzata dai seguenti campi dati:
- QString titolo: titolo dell'opera
- Bool statoPresenza: indica se l'opera é presente nella biblioteca oppure in prestito, (1 indica la presenza dell'opera nell'archivio) (0 indica la non presenza, ossia che in prestito).

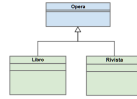


Figure 1: vedi file opera.png

- Int id: identificativo univoco dell'opera, attribuito a ciascuna opera al momento della costruzione dell'opera stessa.
- Int maxid: il massimo id dell'opera, l'opera con id più elevato di tutte le altre viene memorizzato in questa variabile.
- Int appartenenza: variabile che indica a chi appartiene l'opera (-1 valore attribuito quando l'opera appartiene alla biblioteca) (idutente valore attribuito quando l'opera appartiene ad un utente).
- Inoltre sono presenti dei metodi non virtuali quali il costruttore:
- Opera(const QString& , bool =0): costruttore a 1,2 argomenti che consente di costruire un'opera inserendo il solo titolo e segnalando di default che l'opera è presente nella biblioteca; il corpo del costruttore consente di assegnare in automatico un ID univoco all'opera prendendo il massimo ID finora assegnato e facendo +1; in questo modo si assicura l'univocità degli ID.
- Metodi get:
- QString GetTitolo()const: metodo costante che consente di ritornare il titolo dell'opera.
- int Get Id() const: metodo costante che consente di ritornare l'Id dell'opera.
- int Get MaxId() const: metodo costante che consente di ritornare l'Id massimo assegnato ad un'opera fino a questo momento.
- Metodi di set:
- void Set Id(const int n) : metodo che consente di modificare l'ID di un'opera facendolo diventare = n.
- void Set maxid(const int): metodo che consente di aggiornare il campo dati maxId in modo che resti sempre il massimo Id assegnato ad un'opera.
- void Setappartenenza(const int): metodo che consente di aggiornare il campo dati appartenenza in modo che si identifichi a chi appartiene quell'opera.
- int Getappartenenza() const: metodo costante che consente di ritornare l'appartenenza di quell'opera.

- void Set Titolo(QString): metodo che consente di aggiornare il campo dati Titolo di un'opera.
- void Riscatta(): metodo che simula il ritorno in biblioteca di un'opera (setutando statoP=1).
- Metodi virtuali:
- virtual bool disponibile()const: metodo virtuale che ritorna true se l'opera di invocazione può essere prestata, false altrimenti (in particolare questa è una versione base che ritorna true se l'opera è presente negli archivi della biblioteca).
- virtual void Write opera(QXmlStreamWriter& xmlWriter) const =0 metodo che consente di scrivere nell' XML l'opera di invocazione. Esso è puro in quanto Libro e Rivista hanno campi dati diversi.
- virtual void PrestaOpera(): metodo che consente di prestare un'opera. È stato scelto di segnalarlo come virtuale in quanto vi sono delle regole diverse nei due sottotipi.
- virtual info opera info tot() const =0: metodo virtuale puro che consente di ritornare un oggetto di tipo info contenente tutte le informazioni relative all'opera di invocazione; il metodo è virtuale puro in quanto le informazioni presenti nei sottotipi sono diverse.
- virtual QString Get tipo() const =0: metodo virtuale puro che ritorna una stringa che rappresenta il tipo dell'opera di invocazione quindi se l'opera è un libro il metodo ritorna la stringa libro mentre se è una rivista ritorna rivista. (il metodo viene utilizzato solo in lettura)
- **Classe libro:**
- Metodi get:
- QString GetAutore() const: metodo costante che consente di ritornare l'autore dell'opera
- void SetAutore(QString): metodo che consente di aggiornare il campo dati Autore di un libro.
- Metodi virtuali:
- virtual void Write Opera(QXmlStreamWriter& xmlWriter) const: implementazione del metodo virtuale puro della classe base che consente la scrittura su database di un'opera.
- virtual info opera info tot()const: metodo che ritorna un oggetto di tipo info contenente tutte le informazioni riguardo all'oggetto di invocazione: titolo, autore, Id, presenza in biblioteca e disponibilità al prestito.

- virtual QString Get tipo()const: metodo che ritorna la stringa libro.
- **Classe rivista:**
- Metodi get:
- int GetMaxAnni() const: metodo costante che consente di ritornare il numero massimo di anni entro il quale un'opera deve essere uscita per permettere il prestito.
- int GetAnnoUscita() const: metodo costante che consente di ritornare l'anno di uscita della rivista di invocazione.
- Metodi virtuali:
- virtual bool disponibile() const: metodo che ritorna true se l'opera in biblioteca e se la sua data di uscita é minore o uguale a maxAnni. Nel caso in cui ritorna true allora significa che l'opera può venire prestata.
- virtual void Write opera(QXmlStreamWriter& xmlWriter) const: implementazione del metodo virtuale puro della classe base che consente la scrittura su database di un'opera.
- virtual info opera info tot()const: metodo che ritorna un oggetto di tipo info contenente tutte le informazioni riguardo all'oggetto di invocazione: titolo, anno di uscita, Id, presenza in biblioteca e disponibilità al prestito
- virtual void PrestaOpera(): metodo che testa la disponibilità della rivista di invocazione e presta l'opera se essa é diponibile al prestito.
- virtual QString Get tipo()const: metodo che ritorna la stringa rivista.

5 Descrizione della gerarchia costituita da utente e dai suoi sottotipi:

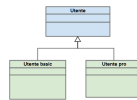


Figure 2: vedi file utente.png

- **La classe utente é alla base della gerarchia ed é caratterizzata dai seguenti metodi:**
- database utente opere* GetdbOpereUtente() const: metodo che permette all'utente di interfacciarsi con il database delle sue opere.

- `database* GetopereBiblioteca() const`: metodo che permette all'utente di interfacciarsi con database delle opere presenti nella biblioteca
- `void ricevi libro() const=0`: metodo virtuale puro che permette all'utente di ricevere in prestito un libro, rispettando i limiti descritti precedentemente.
- `void ricevi rivista() const=0`: metodo virtuale puro che permette all'utente di ricevere in prestito una rivista, rispettando i limiti descritti precedentemente.
- `void restituisci libro() const=0`: metodo virtuale puro che permette all'utente di restituire un libro.
- `void restituisci rivista() const=0`: metodo virtuale puro che permette all'utente di restituire una rivista.
- `bool checklimite() const=0`: metodo virtuale puro che restituisce true se l'utente é abilitato a ricevere l'opera, false altrimenti.
- **Classe utente pro:**
 - nella classe utente pro vengono implementati tutti i metodi virtuali puri della classe base utente, rispettando il limite imposto ad un utente pro ossia `limiteopere=8`.
- **Classe utente basic:**
 - nella classe utente basic vengono implementati tutti i metodi virtuali puri della classe base utente, rispettando il limite imposto ad un utente basic ossia `limiteopere=5`.

6 Descrizione dell'interfaccia grafica

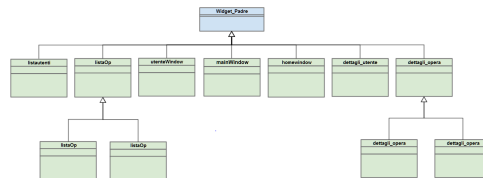


Figure 3: vedi file view.png

6.1 Descrizione della gerarchia costituita da Widget padre e dei suoi sottotipi:

- Questa gerarchia é stata scelta per due motivi principali:
 1. Tutte le view derivano da una base comune Widget padre, perciò in futuro sarà possibile aggiungere nuove funzionalità al sistema, implementando altre view.
 2. Le view che derivano da Widget padre hanno delle caratteristiche in comune:
 - Una MODEL di riferimento rappresentata dalla classe database, databaseutente e databaseutenteopere.
 - I dati che vengono visualizzati nelle view vengono aggiornati in seguito ad azioni eseguite dall'utente.
 - Alle view che vengono costruite, viene applicato uno stile.
- **Classe Widget padre:**
- La classe Widget padre costituita é dai seguenti metodi:
- database* get model() const: permette di accedere al database delle opere presenti nella biblioteca.
- database utente* get modelutenti() const: permette di accedere al database degli utenti registrati nella biblioteca.
- virtual void set style(): metodo che permette di applicare lo stile alle view costruite.
- virtual void aggiorna vista() =0: questo metodo permette di aggiornare i dati presenti nella vista di invocazione successivamente ad una azione dell'utente.
- virtual void costruisci contenuto() =0: il metodo permette di inserire per la prima volta i dati all'interno della view.
- void centra finestra(): questo metodo privato consente di centrare una finestra nel display al momento della creazione della stessa.

6.2 Classe dettagli opera e lista opere:

- La classe dettagli opera é la classe base per 2 altre classi: dettagli rivista e dettagli libro. La scelta di inserire un'altra classe base é stata fatta in modo tale da favorire l'estensibilità del codice; in particolare nel caso in cui la biblioteca decida di gestire un' altra tipologia di opera ad esempio Documentario , il codice prevede già una view base (lista opera) per visualizzare i dettagli della nuova opera rendendo così facile l'inserimento di una nuova view con caratteristiche simili alle altre.