Università degli Studi di Padova

Laurea in Informatica

Corso di Programmazione ad Oggetti

Progetto di fine corso

Studente:
Carlo Sindico

Matricola:
1069322

29/01/2017

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

1.1 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

1.2 Scopo del progetto

- Il progetto ha lo scopo di fornire un semplice ed intuitivo sistema per lamministrazione, 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 pu essere di due tipi: libro o rivista.

  Un libro una particolare opera caratterizzata da un autore, mentre invece una rivista caratterizzata da un anno di uscita.

  Le opere della biblioteca possono venire prestate agli utenti, ogni opera contiene linformazione relativa al suo stato se essa disponibile al prestito e se presente nella biblioteca. Per quanto riguarda i libri non vi alcun vincolo sul prestito, invece le riviste che hanno un anno di uscita superiore a 20 anni non possono essere prestate n allutente basic n allutente pro.

  La biblioteca fornisce unarea utente, attraverso la quale ogni utente pu prendere in prestito le opere. (possibilit di visualizzare le opere presenti nella biblioteca, e contemporaneamente quelle prese in prestito). Lutente attraverso questa interfaccia pu ricevere e restituire le opere.

  Un utente 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 un particolare utente, caratterizzato da un numero di opere che pu ricevere in prestito, fissato a 5. Un utente pro un particolare utente, caratterizzato da un numero di opere che pu ricevere in prestito, fissato a 8.

- **Andremo ad analizzare successivamente pi in dettaglio la gerarchia opera**

  Il bibliotecario (amministratore), attraverso il sistema deve essere in grado di eseguire semplici azioni quali:

- Visualizzare tutte le opere presenti nella biblioteca

- Visualizzare i dettagli di tutte le opere presenti nellarchivio, ed eventualmente di modificarne i parametri quali titolo autore o anno di uscita, a seconda che sia un libro o una rivista.
- Visualizzare tutti gli utenti registrati alla biblioteca
- Visualizzare i dettagli di tutti gli utenti presenti nellarchivio, ed eventualmente di modificarne i parametri quali nome e cognome.
- Aggiunta di un'opera per aumentare l'offerta concessa dalla biblioteca
- Rimuovere un'opera
- Aggiungere un utente allarchivio della biblioteca.
- Rimuovere un utente (solo se ha restituito tutte le sue opere.

  Lutente, attraverso il sistema deve essere in grado di eseguire semplici azioni quali:
- Visualizzare tutte le opere presenti nella biblioteca.
- Ricevere in prestito secondo i limiti descritti precedentemente libri e riviste.
- Restituire alla biblioteca libri e riviste.

2 Descrizione della gerarchia costituita da opera e dai suoi sottotipi

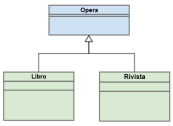

Figure 1: vedi file opera.png

2.0.1 Classe opera

- La classe opera alla base della gerarchia ed caratterizzata dai seguenti campi dati:
- QString titolo: titolo dellopera.

- Bool statoPresenza: indica se lopera presente nella biblioteca oppure in prestito, (1 indica la presenza dellopera nellarchivio) (0 indica la non presenza ossia che in prestito).
- Int id: identificativo univoco dellopera, attribuito a ciascuna opera al momento della costruzione dellopera stessa.
- Int maxid: il massimo id dellopera, lopera con id pi elevato di tutte le altre e viene memorizzato in questa variabile.
- Int appartenenza: variabile che indica a chi appartiene lopera (-1 valore attribuito quando lopera appartiene alla biblioteca) (idutente valore attribuito quando lopera 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 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'univocit degli ID.
- Metodi get:
- QString GetTitolo()const metodo constante che consente di ritornare il titolo dell'opera.
- int Get _Id() const metodo constante che consente di ritornare l'Id dell'opera.
- int Get _MaxId() const metodo constante 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 idenifichi a chi appartiene quell'opera.
- int Getappartenenza() const metodo constante che consente di ritornare l'appartenenza di quell'opera.
- void Set _Titolo(QString) metodo che consente di aggiornare il campo dati maxId in modo che resti sempre il massimo Id assegnato ad un'opera.
- void Riscatta() metodo che simula il ritorno in biblioteca di un'opera (settando statoP=1).

- Metodi virtuali:
- virtual bool disponibile( )const metodo virtuale che ritorna true se l'opera di invocazione pu essere prestata, false altrimenti (in particolare questa una versione base che ritorna true se l'opera presente negli archivi della biblioteca).
- virtual void Write _opera(QXmlStreamWriter& xmlWriter) const =0 metodo che consente di scrivere nel file XML l'opera di invocazione. Esso puro in quanto Libro e Rivista hanno campi dati diversi.
- virtual void PrestaOpera() metodo che consente di prestare un'opera. stato scelto di segnarlo 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 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 un libro il metodo ritorna la stringa libro mentre se una rivista ritorna rivista. (il metodo viene utilizzato solo in lettura)

2.0.2 Classe libro

- Metodi get:
- QString GetAutore() const metodo constante 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, prensenza in biblioteca e disponibilit al prestito.
- virtual QString Get _tipo()const metodo che ritorna la stringa libro.

2.0.3 Classe rivista

- Metodi get:
- int GetMaxAnni( ) const metodo constante 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 constante 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 minore o uguale a maxAnni. Nel caso in cui ritorna true allora significa che l'opera pu 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, prensenza in biblioteca e disponibilit al prestito
- virtual void PrestaOpera() metodo che testa la disponibilit della rivista di invocazione e presta l'opera se essa diponibile al prestito.
- virtual QString Get _tipo()const metodo che ritorna la stringa rivista.

3 Descrizione della gerarchia costituita da utente e dai suoi sottotipi

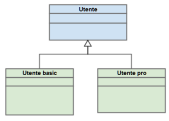

Figure 2: vedi file utente.png

3.0.1 Classe utente

- La classe opera alla base della gerarchia ed caratterizzata dai seguenti metodi (verranno descritti in particolare i metodi considerati pi˘ importanti). **Di un utente vogliamo memorizzare le opere che ha in prestito.**
- 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 abilitato a ricevere l'opera, false altrimenti.

3.0.2 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.

3.0.3 Classe utente basic

- 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=5.

3.1 Descrizione della interfaccia grafica

3.1.1 Descrizione della gerarchia costituita da Widget padre e dei suoi sottotipi

- Questa gerarchia stata scelta per due motivi prinicpali: 1) Tutte le view derivano da una base in comune Widget padre, perci in futuro sar possibile aggiungere nuove funzionalit al sistema, implementando altre view. 2) Le view che derivano da Widget padre hanno delle caratteristiche in comune:

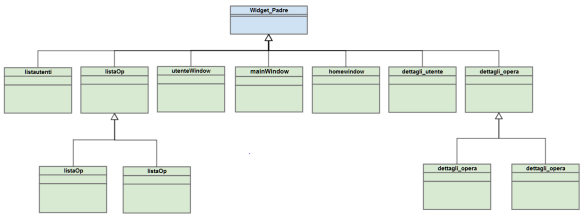

Figure 3: vedi file view.png

- una MODEL di riferimento rappresentata dalle classe database , database_utente e database_utente_opere.
- i dati che sono visualizzati nelle view vengono aggiornati in seguito ad azioni eseguite dall'utente.
- alle view quando vengono costruite, viene applicato uno stile.

3.1.2 Classe Widget padre

- La classe Widget padre costituita dai seguenti metodi:
- database* get _model() const permette di accere 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 costruiscicontenuto 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.
- Classe dettagli_opera e lista_opere:

  La classe dettagli_opera la classe base per 2 altre classi: dettagli_rivista e dettagli_libro. La scelta di inserire un'altra classe base stata fatta in modo tale da favorire l'estensibilit del codice; in particolare nel caso in cui la biblioteca decida di gestire un' altra tipologia di opera ad esempio

Film , il codice prevede gia una view base (lista_opera) per visualizzare i dettagli della nuova opera redendo cosi facile l'inserimento di una nuova view con caratteristiche simili alle altre.