FreedomLand City Hall

Relazione sul progetto di Programmazione ad Oggetti

Lahmer Abdelilah
Matricola 1072695

1. Ambiente di sviluppo

Sistema operativo di sviluppo: Ubuntu OS, versione 14.04 LTS
Versione compilatore: gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)
Versione libreria Qt: 5.3.2

2. Descrizione del progetto

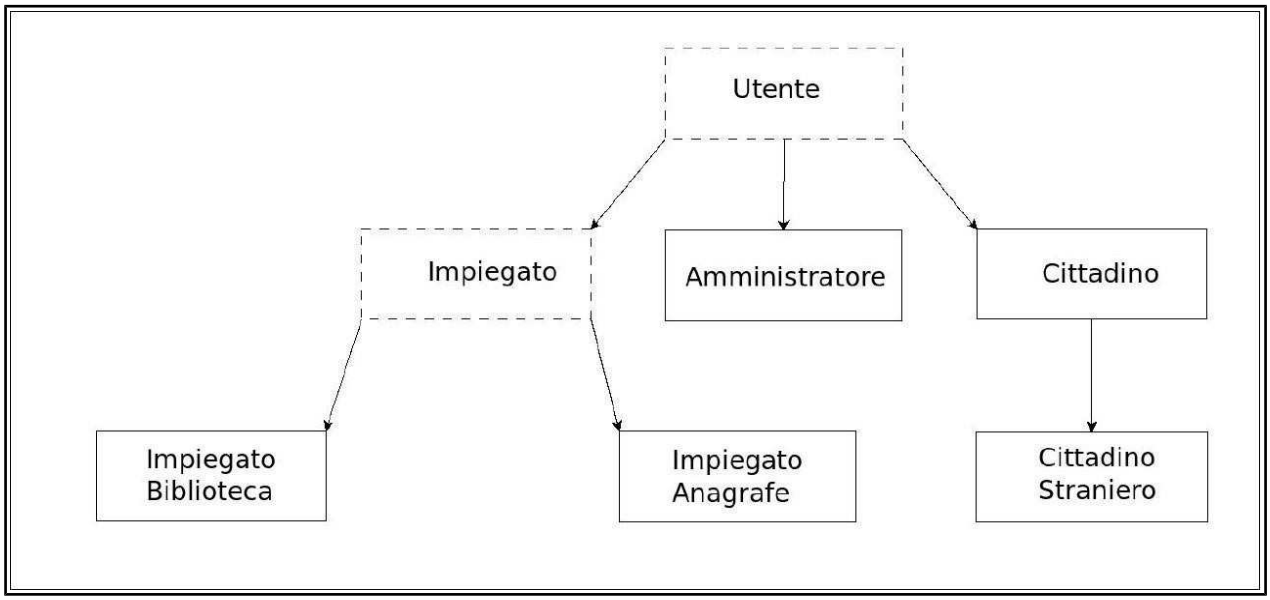
FreedomLand City Hall è il gestionale per i comuni di "FreedomLand". Il sistema mette a disposizione molteplici funzionalità che permettono ad ogni comune del paese la gestione dei propri abitanti, che anch'essi possono accedervi ma con funzionalità limitate. Il progetto è stato sviluppato in C++/Qt seguendo il pattern Model-View-Controller con cui si è cercato di separare il più possibile la progettazione logica da quella della GUI cercando di combinare i due tramite il Controller. Di seguito sono elencate le scelte progettuali ritenute più significative, suddivise per Model, View e Controller.

3. Model

3.1 Gerarchia di classi

I cinque tipi di utenti che possono accedere a *FreedomLand City Hall* con ognuno le sue funzionalità e i suoi permessi sono ***Amministratore, Impiegato Anagrafe, Impiegato Biblioteca, Cittadino e Cittadino Straniero.*** Le tipologie di utente del sistema sono state implementate tramite la seguente gerarchia di classi:

- ***Utente*****,** classe base polimorfa e *astratta* ;
- **Amministratore,** classe concreta derivata da ***Utente*** ;
- ***Impiegato*****,** classe *astratta* derivata da ***Utente*** ;
- **Impiegato Anagrafe,** classe concreta derivata da ***Impiegato*** ;
- **Impiegato Biblioteca,** classe concreta derivata da ***Impiegato*** ;
- **Cittadino,** classe concreta derivata da ***Utente*** ;
- **Cittadino Straniero,** classe concreta derivata da ***Cittadino*** ;



3.2 Modellazione della classe Utente

Per poter modellare la classe *Utente* si è scelto di incapsularci le seguenti classi:

- **Info**: contiene tutte le informazioni dell'utente(nome, cognome, data di nascita, professione, mail, residenza).
- **Credenziali**: contiene le informazioni inerenti al login dell'utente (username e password criptata).

Nella parte privata della classe Utente sono presenti quindi due puntatori:

- ***info*** di tipo Info*
- ***cred*** di tipo Credenziali*

3.3 Polimorfismo offerto da Utente

All'interno della classe *Utente* sono dichiarati i seguenti metodi virtuali puri che permettono l'uso del polimorfismo e l'uso del *binding dinamico* per gli oggetti delle classi derivate:

```
virtual void StoreData(QXmlStreamWriter& xmlWriter) = 0;
virtual void CreaModel(const QString& un) = 0;
virtual QString getCittadinanza() const = 0;
virtual QString getCartaDidentita() const = 0;
virtual QString getPassaporto() const = 0;
virtual QString getPermessoDiSoggiorno() const = 0;
```

I metodi sopraelencati sono stati tutti ridefiniti nelle classi derivate concrete, mentre in quelle derivate astratte sono rimasti virtuali puri in modo da ereditare l'astrattezza di *Utente*.
Questi metodi virtuali sono stati sfruttati in situazioni in cui sarebbe stato necessario distinguere il da farsi in base al tipo di utente; così facendo si ha sfruttato la potenza del polimorfismo.

3.4 Modellazione della classe Book

Uno dei tipi di utenti progettato è l'Impiegato della Biblioteca. Per poter gestire I libri della biblioteca ad esso affidati si è scelto di immagazzinarli sotto forma di oggetti di una classe Book.
Tali oggetti sono caratterizzati da:

```
QString ISBN;
QString titolo;
QString autore;
QString casaEditrice;
QString  usernamePossessoreTemporaneo;
```

Sarà poi compito esclusivo dell'Impiegato della Biblioteca aggiungere, rimuovere o prestare libri a qualche cittadino.

3.5 MyContainer

MyContainer è il contenitore templetizzato progettato. Il contenitore contiene un iteratore myIterator che sfrutta la potenza degli *Smart Pointer* per la gestione profonda della memoria. Il contenitore permette l'inserimento e la rimozione tramire i seguenti metodi:

```
void Push(T*);          //INSERIMENTO IN TESTA
T* Pop();               //RIMOZIONE IN TESTA
void Remove(T*);        //RIMOZIONE DELL'ELEMENTO PASSATO
```

In seguito nel progetto il contenitore templetizzato sarà istanziato in due modi nei corrispondenti database: usando oggetti polimorfi della classe Utente oppure usando oggetti di tipo Book.

3.6 Database

3.6.1 UsersDatabase

Il database degli utenti iscritti a *FreedomLand City Hall* è stato rappresentato con una classe **UsersDatabase** che sfrutta il contenitore templetizzato myContainer.
Nella classe UsersDatabase sono presenti metodi per la rimozione, inserimento e ricerca di utenti in base al loro username. Per quanto riguarda il salvataggio/lettura su file si è optato per il formato XML sfruttando le classi QXmlStreamReader e QXmlStreamWriter offerte dalla libreria Qt.
La lettura da file viene effettuata una volta sola al momento della creazione del database (un oggetto di tipo UsersDatabase) nel file main.cpp: tutti gli utenti letti vengono quindi inseriti nel database. Il salvataggio su file invece viene effettuato anch'esso una volta sola al momento della distruzione del database, cioè al momento della terminazione del programma, nel nostro caso all'uscita dal main. Il file in cui la classe va a scrivere e leggere è stato denominato "UsersDatabase.xml" e in assenza di esso alla prima esecuzione questo viene creato automaticamente e riempito con un utente amministratore con username *admin*.

3.6.2 BooksDatabase

Il database dei libri della biblioteca è stato rappresentato con una classe **BooksDatabase** che sfrutta il contenitore templetizzato myContainer progettato. Nella classe BooksDatabase sono presenti metodi per la rimozione, inserimento e ricerca di libri in base al loro ISBN. Per quanto riguarda il salvataggio/lettura su file sono state fatte le stesse scelte della classe **UsersDatatabase**. Il file in cui la classe va a scrivere e leggere è stato denominato "BooksDatabase.xml" e in assensza di esso alla prima esecuzione questo viene creato automaticamente.

3.7 Model Utenti

3.7.1 ModelAmministratore

E' stata creata la classe **ModelAmministratore** che rappresenta i concetti lato utente di tipologia Amministratore. Nella classe ModelAmministratore sono definiti metodi che il Contoller usa per la rimozione, inserimento e modifica dal/nel database. Nella parte privata è presente un puntatore di tipo UsersDatabase* per effettuare queste operazioni assieme ad un puntatore Utente* che conterrà le informazioni dell'Amministratore loggato.

3.7.2 ModelImpiegatoAnagrafe

E' stata creata la classe **ModelImpiegatoAnagrafe** che rappresenta i concetti lato utente di tipologia Impiegato Anagrafe. Nella classe ModelImpiegatoAnagrafe sono definiti metodi che il Contoller usa per la lettura e modifica dal/nel database. Nella parte privata è presente un puntatore di tipo UsersDatabase* per effettuare queste operazioni assieme ad un puntatore Utente* che conterrà le informazioni dell'Impiegato dell'Anagrafe loggato.

3.7.3 ModelImpiegatoBiblioteca

E' stata creata la classe **ModelImpiegatoBiblioteca** che rappresenta i concetti lato utente di tipologia Impiegato Biblioteca. Nella classe ModelImpiegatoBiblioteca sono definiti metodi che il Contoller usa per la lettura di utenti e rimozione e inserimento di libri dal/nel database. Nella parte privata sono presenti due puntatori di tipo UsersDatabase* e BooksDatabase* per effettuare queste operazioni assieme ad un puntatore Utente* che conterrà le informazioni dell'Impiegato della Biblioteca loggato.

3.7.4 ModelCittadino

E' stata creata la classe **ModelCittadino** che rappresenta i concetti lato utente di tipologia Cittadino. Nella classe ModelCittadino sono definiti metodi che il Contoller usa per la lettura e modifica dei dati dell'utente loggato dal/nel database. Nella parte privata è presente un puntatore di tipo UsersDatabase* per effettuare queste operazioni assieme ad un puntatore Utente* che conterrà le informazioni del Cittadino loggato.

3.7.5 ModelCittadinoStraniero

E' stata creata la classe **ModelCittadinoStraniero** che rappresenta i concetti lato utente di tipologia Cittadino Straniero. Nella classe ModelCittadinoStraniero sono definiti metodi che il Contoller usa per la lettura e modifica dei dati dell'utente loggato dal/nel database. Nella parte privata è presente un puntatore di tipo UsersDatabase* per effettuare queste operazioni assieme ad un puntatore Utente* che conterrà le informazioni del Cittadino Straniero loggato.

4. View

4.1 StartWindow

La schermata iniziale del progetto dà la possibilità di effettuare il login con il proprio username e la propria password o di chiudere l'applicazione: essa è un oggetto della classe **StartWindow**, derivata da QMainWindow, che inizialmente imposta come widget centrale un oggetto della classe **HomeView**, derivata da QGroupBox, che effettivamente contiene la parte grafica che è stata descritta in precedenza.

4.2 View Utenti

4.2.1 AmministratoreView

Se si accede come utente di tipo Amministratore, si avrà accesso alla view chiamata **AmministratoreView**, che diventerà quindi widget centrale della StartWindow. La view si presenta con una tabella con tutti gli utenti presenti nel database sul lato sinistro, mentre sul lato destro si ha la possibilità di: eliminare l'utente selezionato, inserire un nuovo utente, modificare il proprio profilo, effettuare il logout (cioè tornare alla schermata iniziale) oppure chiudere l'applicazione usando il bottone “Esci”. Se si decide di inserire un nuovo utente apparirà un oggetto della classe **InserisciNuovoUtente**, derivata da QWidget**.** Se invece si decide di modificare il proprio profilo apparirà un oggetto della classe **ModificaProfilo**, derivata da QDialog.

4.2.2 ImpiegatoAnagrafeView

Se si accede come utente di tipo Impiegato Anagrafe, si avrà accesso alla view chiamata **ImpiegatoAnagrafeView**, che diventerà quindi widget centrale della StartWindow. La view si presenta con una tabella con tutti gli utenti di tipologia Cittadino e CittadinoStraniero presenti nel database sul lato sinistro, mentre sul lato destro si ha la possibilità di: decretare la cittadinanza di un CittadinoStraniero (quindi cambiargli tipo in Cittadino), visualizzare o modificare i dati anagrafici e personali dell'utente selezionato, inserire un nuovo cittadino, modificare il proprio profilo, effettuare il logout (cioè tornare alla schermata iniziale) oppure chiudere l'applicazione usando il bottone “Esci”. Se si decide di inserire un nuovo cittadino apparirà un oggetto della classe **InserisciNuovoCittadino**, derivata da QWidget**.**
Se si decide di visualizzare o modificare i dati anagrafici e personali di un utente apparirà un oggetto della classe **SchedaUtente**, derivata da QDialog. Se invece si decide di modificare il proprio profilo apparirà un oggetto della classe **ModificaProfilo**, derivata sempre da QDialog.

4.2.3 ImpiegatoBibliotecaView

Se si accede come utente di tipo Impiegato Biblioteca, si avrà accesso alla view chiamata **ImpiegatoBibliotecaView**, che diventerà quindi widget centrale della StartWindow. La view si presenta con due tabelle sul lato sinistro, una con tutti gli utenti di tipologia Cittadino e CittadinoStraniero presenti nel database e una con tutti i libri presenti nel rispettivo database contenente elementi di tipo Book. Sul lato destro invece si ha la possiblità di: modificare il proprio profilo, imprestare il libro selezionato al cittadino selezionato, rientrare un libro attualmente imprestato ad un dato cittadino, inserire un nuovo libro, eliminare il libro selezionato, effettuare il logout (cioè tornare alla schermata iniziale) oppure chiudere l'applicazione usando il bottone "Esci". Se si decide di inserire un nuovo libro apparirà un oggetto della classe **InserisciNuovoLibro**, derivata da QWidget**.** Se invece si decide di modificare il proprio profilo apparirà un oggetto della classe **ModificaProfilo**, derivata da Qdialog.

4.2.4 CittadinoView

Se si accede come utente di tipo Cittadino, si avrà accesso alla view chiamata **CittadinoView**, che diventerà quindi widget centrale della StartWindow. La view si presenta con le proprie informazioni sul lato sinistro, mentre sul lato destro invece si ha la possiblità di: modificare il proprio profilo(limitatamente), effettuare il logout (cioè tornare alla schermata iniziale) oppure chiudere l'applicazione usando il bottone "Esci".
Se si decide di modificare il proprio profilo apparirà un oggetto della classe **ModificaProfilo**, derivata da QDialog.

4.2.5 CittadinoStranieroView

Se si accede come utente di tipo CittadinoStraniero, si avrà accesso alla view chiamata **CittadinoStranieroView**, che diventerà quindi widget centrale della StartWindow. La view si presenta con le proprie informazioni sul lato sinistro comprensive di numero identificativo del permesso di soggiorno che fa dell'utente Cittadino Straniero una distinzione da Cittadino, mentre sul lato destro invece si ha la possiblità di: modificare il proprio profilo, effettuare il logout (cioè tornare alla schermata iniziale) oppure chiudere l'applicazione usando il bottone "Esci". Se si decide di modificare il proprio profilo apparirà un oggetto della classe **ModificaProfilo**, derivata da QDialog.

5. Controller

5.1 Controller

Il controller è un'interfaccia che è incaricata per unire i vari model e view di ciascun tipo di utente loggato, nello specifico ha come funzione principale quella di operare sul model e restituire i risultati da mostrare alla view, mentre invece quest'ultima legge i dati dal model e quando è necessario modificarli invia un segnale al controller. Un oggetto di tipo **Controller** viene creato nel file main.cpp subito dopo la creazione dei databases ed esso crea la schermata principale di tipo StartWindow gestendo infine il login e creando un tipo preciso di Model e View in base al tipo di Utente che sta cercando di loggarsi. La classe Controller possiede cinque "figli", **AmministratoreController**, **ImpiegatoAnagrafeController**, **ImpiegatoBibliotecaController, CittadinoController** e **CittadinoStranieroController** che sono cinque classi che rappresentano rispettivamente i controller per AmministratoreView, ImpiegatoAnagrafeView, ImpiegatoBibliotecaView, CittadinoView e CittadinoStranieroView.
I cinque controller "figli" gestiscono i segnali provenienti dalle rispettive view e ripercuotono gli effetti sui rispettivi model, analogamente quando operano sul model ripercuotono gli effetti sulle rispettive view.

5.2 Login

Come spiegato nel punto precedente il controller crea un tipo preciso di Model e View in base al tipo di Utente che sta cercando di effettuare il login. Per far questo si è usata la potenza del polimorfismo che ha permesso in poche parole di rendere pure la View "polimorfa"; per far questo si è usato il metodo implementato nella classe Controller:

```
void Controller::creaModelUtente(const QString& username, const QString& password) {
        if(databaseUtenti->Login(username,password))  {
                Utente* temp = databaseUtenti->Find_Utente(username);
                temp->CreaModel(username);
        } else finestraIniziale->utenteNonTrovato();
}
```

Questo metodo della classe Controller infatti richiama il metodo:

```
virtual void CreaModel(const QString& un) = 0;
```

della classe *Utente* che permette il binding dinamico in base al tipo di utente e manda un segnale per la creazione del rispettivo Model.

5.3 Logout

È il Controller (assieme alla schermata iniziale StartWindow) che gestisce il passaggio tra le varie view: quando si effettua il logout, la StartWindow crea la HomeView che verrà impostata come widget centrale e cancella le view precedentemente create. Il Controller invece cancella i controller "figli" e dopodiché cancella i model.

6. Informazioni per l'utilizzo

Per accedere per la prima volta al sistema effettuare il login con le seguenti credenziali che vengono create in automatico dal Controller nel caso in cui il database degli utenti fosse vuoto:

Username: **admin** Password: **admin**

Dopo aver fatto l'accesso con tali credenziali sarà possibile effettuare le operazioni permesse al tipo di utenti Amministratore e quindi si avrà la facoltà di aggiungere ogni tipo di utente e testare le rispettive funzionalità. *Enjoy*!