LinQedin

Università degli studi di Padova
Corso di laurea in informatica
2014/2015
Studente: Emanuele Carraro
Matricola: 1070742

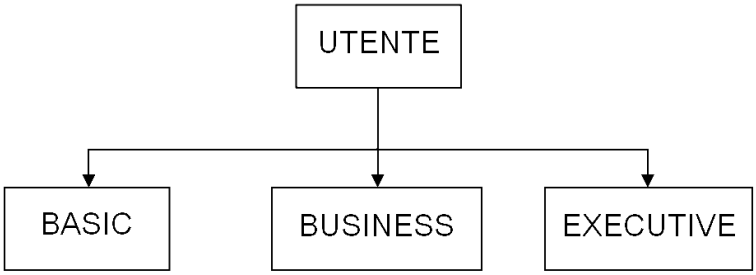
Progetto di programmazione a oggetti

Descrizione delle classi principali

Classi logiche

Utente

È la classe base astratta degli utenti. Da essa derivano le classi Basic, Business ed Executive:



Un utente è dotato di username (univoco), password, una rete di contatti e un profilo (composto da dati anagrafici e curriculum).
In base alla tipologia di account, sono previste diverse funzionalità di ricerca.

LinqedinAdmin

Rappresenta l'amministratore di LinQedin. Dispone di metodi per la gestione del database, come il caricamento e il salvataggio di tutti i dati, l'inserimento e l'eliminazione di utenti (singoli). Inoltre si occupa di cambiare la tipologia di account di un utente, qualora sia presente una richiesta.

SmartUtente

Contiene un **puntatore polimorfo** alla classe base astratta Utente. Uno smartutente ha le caratteristiche principali di uno **smart pointer**, perciò si occupa di gestione della memoria. Viene utilizzato da varie classi logiche, in particolare dalla classe Database.

Rete

Gestisce i contatti di un utente LinQedin. Contiene un vector di username che determinano in modo univoco ciascun contatto.

Database

Contiene le informazioni degli utenti LinQedin. Utilizza la classe SmartUtente, di conseguenza gode dei vantaggi degli smart pointers.

Intestazione della classe:

```
class Database
{
   friend class SmartUtente;
private:
   vector<SmartUtente> database;
...
};
```

Il database utilizza il contenitore vector.
In questa classe sono implementate tutte le funzioni che riguardano strettamente il database (inserimento/eliminazione/ricerca/cambio di un utente).
I dati sono salvati in un file xml.

Controller

Agisce da collegamento tra la parte grafica e la parte logica.
Gestisce le comunicazioni: viewAdmin - modelAdmin e viceversa, viewUtente - modelUtente e viceversa. Principalmente, tutto quello che fa è richiamare metodi delle classi logiche, a fronte di determinati eventi verificatisi nella parte grafica. In alcuni casi fa del lavoro in più, ad esempio, si occupa del passaggio dei dati quando un nuovo utente si iscrive (dalla FormClass alle classi logiche dell'Utente: profilo, curriculum ecc...) e di creare effettivamente il nuovo utente. Una parte dell'intestazione è:

```
class Controller
{
private:
   SmartUtente utente; // "punta" all'utente attualmente collegato
   LinqedinAdmin* admin; // puntatore all'amministratore
   Database* database;  // puntatore al database
...
};
```

Si nota che la parte privata contiene gli agganci alle **tre principali classi logiche.**

Tutte le classi grafiche vengono inizializzate con un puntatore allo stesso (**unico**) controller. Non è stato seguito pienamente il pattern Model-View-Controller, ma si è cercato di coglierne alcuni degli aspetti principali.

Classi grafiche

HomeWindow/HomeAdmin

Sono le due classi grafiche per l'interfaccia d'accesso dell'utente e dell'amministratore, divise da QSplitter nella mainwindow.
Entrambe dispongono di un'intestazione data dalla classe Rettangolo e di un box d'accesso. La parte riservata all'utente permette, naturalmente, anche l'iscrizione a LinQedin.

L’unico scopo della classe Rettangolo è quello di mostrare un semplice esempio di overloading del metodo protetto void *paintEvent*(QPaintEvent* e), che permette di disegnare sullo schermo un rettangolo colorato.

UserWindow/AdminWindow

Sono le due classi grafiche per le pagine dell’utente e dell’amministratore.
La pagina dell’amministratore è suddivisa in:
- box di gestione del database
- box di ricerca
- box per la gestione dei cambi di account

La pagina dell’utente è suddivisa in:
- box per le informazioni sul profilo
- box di ricerca
- box di gestione della rete di contatti

FormClass

È la classe grafica riservata all’iscrizione di un nuovo utente.
Quando l’utente clicca su “Iscriviti” viene effettuato un controllo sull’input. Si verifica che i campi “Nome”, “Cognome”, “Password”, “Username” e “Sesso” non siano vuoti e che non esista nessun altro utente con lo stesso username inserito.
Sono previsti inoltre due box per l’inserimento delle esperienze lavorative e dei titoli di studio.
Un oggetto FormClass può essere costruito secondo due modalità: iscrizione e modifica.
Nel caso sia in modalità modifica, tutti i dati dell’utente vengono mostrati nei rispettivi campi e nei box delle esperienze e dei titoli, predisposti per essere eventualmente modificati.

EspLavoroBox/TitoliStudioBox

Gestiscono l’inserimento di una sequenza, rispettivamente, di esperienze lavorative e titoli di studio. Il box permette di percorrere (e modificare) la sequenza inserita fino a un certo punto grazie ai testi precedente, salva e cancella. Gli slot “precedente” e “successivo” si occupano di salvare i dati nuovi o modificati e di mostrare correttamente i dati già inseriti nel caso si voglia scorrere la sequenza.

RicercaBox

Permette la ricerca di un utente all’interno del database.
La ricerca può avvenire tramite nome, cognome e username. (operazioni garantite all’utente basic).
Per gli utenti business ed executive sono previste ricerche in base alla provincia, alle aziende dove gli utenti hanno lavorato, ai titoli di studio e alle aree di competenza.
Per i soli utenti executive è prevista la possibilità di effettuare ricerche combinate.
Oltre a limitare le funzionalità di ricerca, il tipo di un utente determina quanti (e quali) dati vengono restituiti, nel caso in cui una ricerca abbia successo.
Questa classe viene utilizzata sia dalla UserWindow che dalla AdminWindow, dato che gli scopi nei due casi sono simili, se non coincidenti.

L’amministratore non ha restrizioni sulle modalità di ricerca e ottiene sempre le informazioni complete su un utente.

MessaggiWindow

E’ la classe grafica per la gestione dei messaggi.
Solo un utente business o executive può decidere di inviare dei messaggi (fino a un massimo definito). Il destinatario può essere scelto solamente fra i propri contatti.
Un messaggio LinQedIn è caratterizzato da un testo e dal destinatario (non è necessario memorizzare anche il mittente).
Il sistema di messaggi LinQedIn è stato progettato per effettuare delle brevi comunicazioni tra gli utenti, quindi una volta che vengono visualizzati sono “scartati”. Inoltre sono usati anche dall’amministratore per comunicare determinati eventi che interessano un utente (ad esempio, l’inserimento nel database o il cambio di account).

In seguito sono descritti due aspetti rilevanti del progetto: le funzionalità di ricerca e il cambio di tipologia di account.

Funzionalità di ricerca

Uno dei metodi che rende la classe Utente virtuale pura è:

```
virtual QString userSearch(const SmartUtente &, bool) const = 0;
```

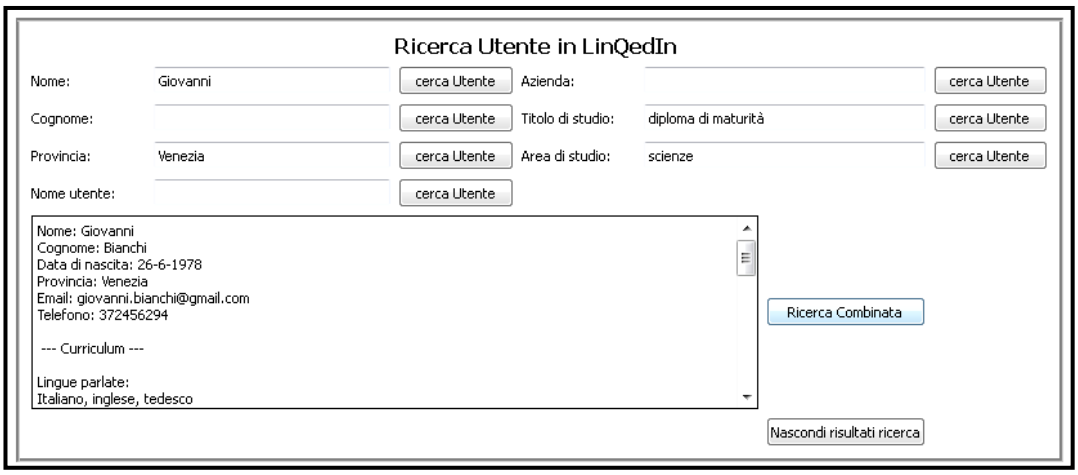
Questa funzionalità di ricerca è implementata nelle classi basic, business ed executive facendo uso dei **funtori**:

```
class Utente
{ …
protected:
   class SearchFunctor {
      public:
         int searchType;
         SearchFunctor(int x=0): searchType(x) {}
         QString operator() (const SmartUtente & x, bool m) const {
            switch(searchType) {
            case 1:
               ... // ritorna informazioni per la ricerca basic
            case 2:
               ... // ritorna informazioni per la ricerca business
            case 3:
               ... // ritorna informazioni per la ricerca executive
            }
      }
};
```

Ad esempio, nella classe basic:

```
QString Basic::userSearch(const SmartUtente & us, bool m) const{
   SearchFunctor f(1);
   return f(us,m); }
```

Il parametro booleano m indica se l'utente su cui si vogliono ottenere informazioni fa parte o no della propria rete di contatti (in questo caso si ottengono più informazioni).



Finestra di ricerca nella pagina Utente

Cambio tipologia di account

Un utente può richiedere il cambio della tipologia di account. L'amministratore può decidere se soddisfare la richiesta oppure no. In caso affermativo, LinqedinAdmin usufruisce del seguente metodo pubblico della classe Database:

```
void cambiaTipoAccount(QString, int);
```

Che riceve come parametri un username e un intero che determina la nuova tipologia di account.

Analisi "grafica" *– La notazione usata è la stessa del libro*

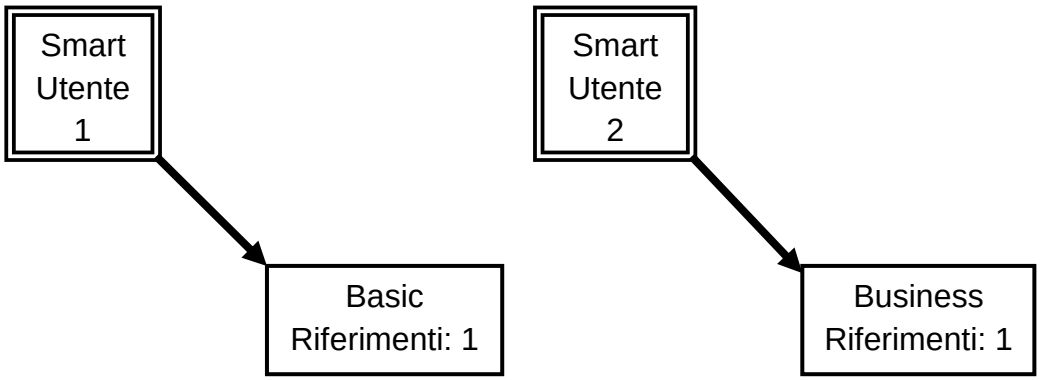
Si suppone che:
- un utente basic voglia cambiare il suo account in business.
- tale utente non sia loggato quando l'amministratore prende in carico la richiesta.

Verranno eseguite le seguenti istruzioni:

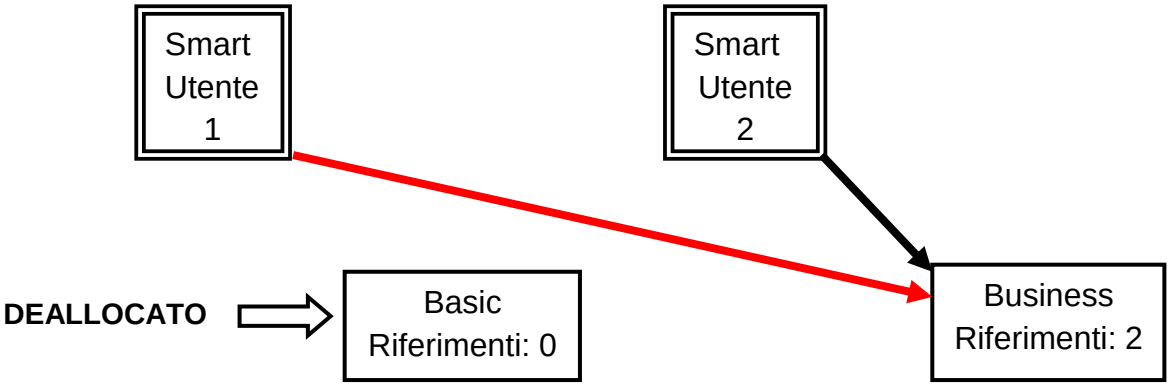
```
(1) SmartUtente SmartUtente2(new Business(*database[i]));
(2) database[i] = SmartUtente2; // assegnazione tra SmartUtente
break;
```

Dove database[i] "punta" all'utente basic da cambiare. In seguito lo chiamerò SmartUtente1.

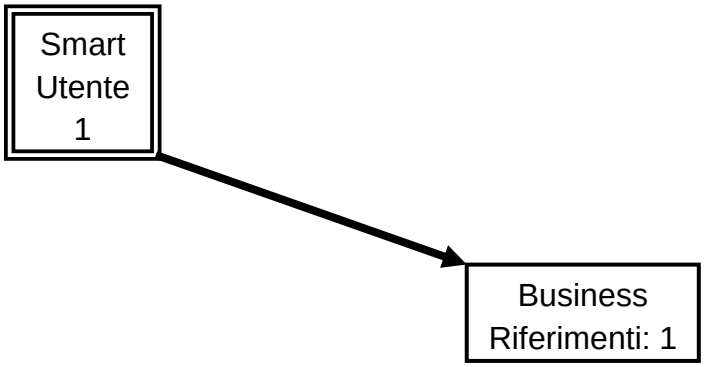
L’istruzione (1) causa l’invocazione del costruttore della classe SmartUtente e del costruttore di copia della classe Utente (ridefinito). I dati dell’utente basic sono tutti copiati nel nuovo utente business.



L’istruzione (2) corrisponde all’assegnazione tra smartutenti e causa:



Poiché SmartUtente2 è una variabile locale, la situazione finale è la seguente.



Nel database, lo smartutente1 iniziale ora punta a un nuovo oggetto business, con tutti i dati dell’oggetto basic precedente (il contratto del metodo è soddisfatto).

Sistema operativo di sviluppo: Windows Vista Home Premium
Compilatore: MinGW 4.8.2
Libreria: Qt 5.3