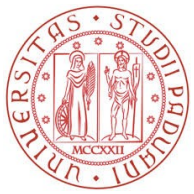


Università degli Studi di Padova
Anno accademico 2014-2015

Relazione per il corso di
Programmazione ad Oggetti
su
LinQedIn

Candidato: Rudy Berton
Matricola: 1049443

Ambiente di sviluppo :

Sistema Operativo: Ubuntu 14.04 LTS
Compilatore: GCC 4.6.1, 32 bit
Qt: versione 5.3.2
Qt Creator: versione 3.2.1

Credenziali per l'accesso:

- Amministratore:
  username: rberton
  password: rb123

- Utente Basic:
  username: tania11
  password: tany_

- Utente Business:
  username: simo52
  password: simone132

- Utente Executive:
  username: marco_18
  password: mr12

Scelte progettuali per lo Scheletro Logico:

La parte logica si compone delle classi di cui si struttura il progetto, seguendo principalmente la linea guida fornita nelle specifiche. È stato scelto di non usufruire di una classe SmartUtente, composta da puntatori smart, dal momento che la gestione della memoria condivisa non presentava enorme difficoltà per le funzionalità che venivano richieste da tale progetto.
Di seguito per ogni classe ne verrà elencata la funzione e gli aspetti che la caratterizzano:

Classe Profilo

Questa classe contiene i dati personali di un utente.
I primi quattro campi che caratterizzano il profilo di un utente sono di tipo stringa e riguardano il nome, il cognome,la data di nascita e l'email; i rimanenti campi (titoli di studio, lingue conosciute, competenze lavorative ed esperienze professionali) invece sono rappresentati come vettori di stringhe. Per questi ultimi è stato infatti scelto il contenitore *vector* dalla libreria STL (e non una semplice lista di stringhe) per favorire l'eliminazione di un elemento all'interno del contenitore in qualsiasi posizione esso sia attraverso l'utilizzo di iteratori ad accesso casuale dal momento che *vector* garantisce l'accesso casuale ai suoi elementi in tempo costante. Questa ottimizzazione non si sarebbe potuta ottenere utilizzando un contenitore *list* poiché per eliminare un suo elemento sarebbe stato necessario percorrere ogni volta la lista dall'inizio fino all'elemento da rimuovere.
La classe presenta, oltre ai costruttori, dei metodi per il recupero dei campi dati, per la loro visualizzazione e modifica (aggiunta o rimozione di elementi all'interno dei contenitori *vector* e modifica del campo email).
Il metodo *stampaProfilo* permette la stampa dei dati personali di un utente all'interno del file FileDB.txt che verrà usato come database per mantenere le informazioni di tutti gli utenti registrati in LinQedIn.
Per quanto riguarda il costruttore a quattro parametri di tipo stringa è stato deciso di non inserire anche la costruzione per gli oggetti di tipo *vector<string>*: l'amministratore avrà la possibilità di inserire un nuovo utente all'interno di LinQedIn utilizzando questo costruttore ed inserendo solo i dati personali necessari (come al momento di una registrazione), sarà poi l'utente in questione che una volta effettuato il login potrà iniziare ad arricchire il proprio profilo.

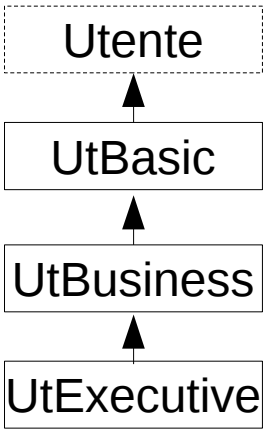
- **Classe Username**

  La classe Username presenta due campi di tipo stringa, login e password, che saranno necessari all'utente per effettuare l'autenticazione. Il campo login viene considerato univoco per ogni utente: non possono pertanto esistere all'interno di LinQedIn due utenti con lo stesso campo login.
  I metodi presenti in questa classe servono per il recupero dei campi dati privati, per il controllo della password e la sua modifica; c'è inoltre la ridefinizione degli operatori di uguaglianza e disuguaglianza per effettuare il confronto tra due Username ed il metodo per la stampa dei campi nel file FileDB.txt.

- **Classe Rete**

  Questa classe funge da contenitore di contatti amici di un determinato utente; presenta per questo nella parte privata un campo di tipo lista di stringhe che conterrà il campo login (della classe *Username*) di tali amici. La scelta di questo contenitore deriva dalle basilari funzionalità richieste da questa classe, ovvero il semplice inserimento, la rimozione e la visualizzazione dei contatti di un utente. Questa volta la scelta è ricaduta su un contenitore *list,*e non un *vector*, di stringhe prediligendo l'inserimento in coda e la visualizzazione dell'intera lista rispetto all'eliminazione di un elemento che avverrà percorrendo l'intera lista dall'inizio.
  I metodi presenti per l'appunto sono quelli per l'inserimento di un nuovo contatto, la sua eliminazione, la visualizzazione dell'intera rete; inoltre ci sono i metodi per l'eliminazione completa della rete, per l'ottenimento del campo privato e per il controllo sulla presenza o meno di un contatto nella rete, funzione che sarà eseguita prima di qualsiasi inserimento nella rete.

- **Classe Utente**

  Questa classe è il vero cuore del progetto poiché risulta essere una classe base polimorfa e astratta.
  Racchiude come campi privati oggetti legati alle classi descritte in precedenza: un campo di tipo *Profilo*, un campo di tipo *Username* ed un puntatore alla classe *Rete*. Presenta inoltre un campo di tipo stringa, *tipoAccount*, che stabilisce a quale delle tre categorie appartenga l'utente: Basic, Business oppure Executive (categorie che evidenzieranno la loro distinzione successivamente riguardo al tipo di ricerca che sarà possibile effettuare).
  Il polimorfismo di tale classe è dato dalla presenza di due metodi virtuali, di cui uno è il distruttore, che verranno ereditati dalle classi derivate e ridefiniti. Uno di questi è un metodo virtuale puro (ciò rende pertanto Utente anche una classe base astratta) che troverà la sua implementazione nelle sottoclassi di Utente: tale metodo è *ricercaUt()* che permette all'utente autenticato di cercare un altro utente nel database (attraverso il suo username) a seconda della propria tipologia di account .
  La gerarchia scelta per rappresentare le tre tipologie di utenti possibili derivanti dalla classe base astratta è lineare: la classe *UtBasic* è classe derivata diretta della classe baseUtente , la classe *UtBusiness* è derivata direttamente dalla classe UtBasic, mentre la classe *UtExecutive* è derivata direttamente da UtBusiness. Questa scelta è stata spinta unicamente dall'implementazione del metodo *ricercaUt()* nella classe UtBasic e dal suo overriding nelle sottoclassi: un utente Basic può solamente ottenere i dati principali di un utente cercato (nome, cognome, data di nascita ed email), un utente Business oltre ai dati precedenti può ottenere anche i titoli di studio, le lingue che la persona conosce, le sue competenze professionali e le esperienze lavorative, infine l'utente Executive ottiene le stesse informazioni delle altre due tipologie di account ed in aggiunta anche la rete di contatti dell'utente cercato.
  La classe base astratta presenta inoltre i metodi per ottenere i campi privati, per modificare i campi delle classi *Profilo*, *Username* ed il campo proprio *TipoAccount* e per stampare l'utente, funzione che richiamerà gli altri metodi di stampa delle classi *Profilo*, *Username* e *Rete*.

- **Classe Database**

  Database è la classe che rappresenta la raccolta di tutti gli utenti registrati in LinQedIn. Per questo dispone un campo privato di tipo vettore di puntatori ad *Utente*; la scelta di questo contenitore è dettata dal fatto che, essendo associativo, permette l'inserimento e la ricerca di un utente in tempo costante, ottimizzando quindi le prestazioni. I metodi infatti che vengono forniti sono il salvataggio degli utenti all'interno del file FileDB.txt, il loro caricamento da file, la ricerca di un utente attraverso il suo username ed un metodo per svuotare completamente il database che sarà

utilizzare appena prima di caricare i dati.

- **Classe LinQedInAdmin**
  Questa classe racchiude tutte le funzionalità che l'amministratore può effettuare in LinQedIn. Contiene come campi privati un puntatore di tipo *Database* ed campo statico di tipo *Username* che contiene le credenziali dell'amministratore per l'accesso.
  Le funzionalità principali sono il caricamento ed il salvataggio da e verso il database.
  L'amministratore può inoltre inserire un nuovo utente (come descritto precedentemente) oppure rimuoverlo completamente, in questo caso la rimozione comporta per gli altri utenti l'eliminazione di tale utente tra i loro contatti di rete. La funzione di ricerca da parte dell'amministratore si differenzia: se si fa una ricerca basandosi sullo username di un utente allora viene restituito il puntatore a quell'utente, altrimenti se la ricerca avviene attraverso il nome e cognome allora si presenta la possibilità di avere persone omonime e pertanto viene restituito una lista di puntatori ad Utente. Un altro metodo appartenente a questa classe riguarda il cambio di account di un utente: attraverso l'utilizzo del dynamic_cast l'amministratore cambia la tipologia di un qualsiasi utente, eliminandolo completamente e creandone uno nuovo con la tipologia aggiornata.

- **Classe LinQedInClient**
  La classe LinQedInClient, come per la classe *LinQeInAdmin*, comprende tutte le attività che un utente può svolgere in LinQedIn una volta autenticato. Come campi privati presenta un puntatore a *Database* ed un puntatore a *Utente*, che rappresenta l'utente corrente.
  I metodi principali sono la modifica dei propri dati (aggiunta o rimozione), l'inserimento oppure la rimozione di un contatto tra la propria rete di conoscenti e la ricerca di un utente attraverso il suo username, funzione che richiama il metodo *ricercaUt()* della classe base astratta *Utente* ed esegue la ricerca diversificata a seconda della propria tipologia (come descritto in precedenza).
  Sono stati inoltre inseriti due metodi che sono utili all'interfaccia grafica per il salvataggio nel database dei cambiamenti effettuati al profilo e il caricamento del database aggiornato.

- **Classe Errore**
  Questa classe serve per la gestione delle eccezioni che possono verificarsi durante l'esecuzione di LinQedIn. Gli errori più comuni riscontrati sono quelli di autenticazione (qualora l'autenticazione di un utente o dell'amministratore non vada a buon fine), ricerca (se la ricerca di un utente fallisce), riguardante i dati dell'utente (ad esempio se si cerca di eliminare un elemento del proprio profilo che non esiste) oppure riguardante l'apertura del file FileDB.txt per la lettura o la scrittura al suo interno. Viene fornito il metodo *mostraErrore()* per garantire la visualizzazione dell'eccezione, sarà infatti utile per la parte grafica.
  Altre possibili eccezioni vengono invece risolte a livello grafico nella realizzazione del codice per la GUI.

Scelte progettuali per la GUI:

Per la realizzazione dell'interfaccia grafica è stato utilizzato il programma Qt Creator e si è cercato di creare una GUI semplice ma efficiente, tenendo separato il più possibile il codice logico da quello per la grafica.
L'interfaccia si divide in lato amministratore e lato utente, fatta eccezione per la finestra principale, dalla quale si può accedere ad uno dei due lati. Infatti *MainWindow* presenta la schermata in cui utente oppure amministratore possono autenticarsi dopo aver scelto la modalità di accesso e aver inserito le proprie credenziali.

L'autenticazione come utente porta all'apertura di una finestra *mainClient* in cui l'utente può eseguire le proprie funzionalità. Nella parte superiore sono presenti dei *QpushButton* tra cui l'utente può scegliere se visualizzare il proprio profilo, modificare il profilo oppure gestire la rete di contatti. In fondo invece c'è la possibilità di eseguire la ricerca di un utente tramite il suo username; il risultato della ricerca, come anche la visualizzazione del proprio profilo viene mostrato nell'area centrale della finestra in cui viene creata una *QScrollArea* contenente un *QLabel* con le informazioni.
Premendo invece i pulsanti “Modifica il profilo ” oppure “Gestisci i tuoi contatti” avviene la creazione di una nuova finestra in cui poter eseguire ulteriori attività: nel primo caso (*ModificaProfilo*) la schermata presenta dei pulsanti che permettono di scegliere quale aspetto del profilo modificare(in cui sarà possibile aggiungere oppure eliminare dei dati), mentre nel secondo caso (*GestisciContatti*) la nuova finestra presenta un elenco sulla sinistra (*QListWidget*) degli utenti che sono già presenti nella propria rete

e sulla destra (un altro *QlistWidget*) quelli che lo potrebbero diventare, lasciando sempre la possibilità di inserire o eliminare utenti, attraverso i rispettivi elenchi.

L'autenticazione come amministratore invece fa aprire la finestra *mainAdmin* contenente una *QScrollBar* e quattro *QPushButton* che permettono all'amministratore di eseguire tutte le funzionalità a lui concesse.
Con il pulsante “Cerca utente” viene aperta una finestra nuova (*RicercaUt*) in cui è possibile scegliere se eseguire la ricerca attraverso lo username oppure il nome e cognome riempiendo poi una corrispondente form.
La finestra *InserimentoUt* che viene creata una volta premuto il pulsante “Inserisci utente” presenta dei *QLineEdit* da riempire con i dati principali del nuovo utente (come precedentemente discusso) e la scelta a quale delle tre tipologie di account debba appartenere.
Il pulsante “Rimuovi utente” apre anch'esso una nuova finestra (*RimozioneUt*) in cui un *QListWidget* contiene l'elenco di tutti gli utenti registrati in LinQedIn, tra cui è possibile selezionare quale si voglia eliminare.
Infine l'apertura della finestra *CambioAccUt*, tramite il pulsante “Cambia account utente”, presenta come per la finestra di rimozione un elenco di tutti gli utenti e la possibilità per ognuno di cambiarne la tipologia dell'account.
La ricerca, l'inserimento e il cambio di account alla conclusione dell'operazione conducono alla pagina precedente, la *mainAdmin*, mostrando all'interno della *QScrollArea* il profilo dell'utente cercato/aggiunto/modificato, mentre per quanto riguarda la rimozione, nella pagina principale esce solamente un messaggio per informare che l'operazione è andata a buon fine.