LinQedIn

Pol Alessandro

matricola 1052596

Progetto del Corso di Programmazione ad Oggetti 2014\2015

1 Documento

il seguento documento ha lo scopo dare una panoramica completa del progetto del corso di programmazione ad oggetti dell'anno 2014/2015: **LinQedln**.

Verranno analizzati separatamente la parte logica e la parte grafica. Verrà posta maggiore attenzione nella parte logica descrivendo le classi che la compongono ed il loro funzionamento all'interno del programma.

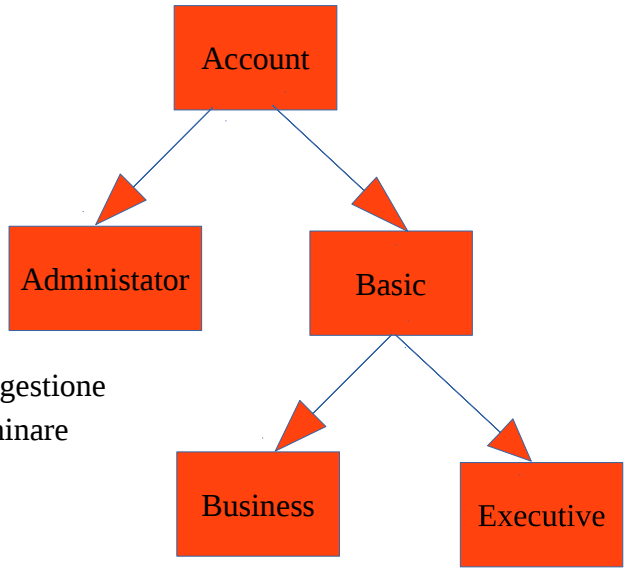
2 Lo scheletro logico

Lo scheletro logico è formato da tre insiemi di classi: l'insieme degli utenti, l'insieme del database e l'insieme degli oggetti che permettono lo scambio di informazioni tra i primi due.

Per ogni gruppo è stata utilizzata la tecnologia dell'ereditarietà con le sue conseguenze che, come vedremo, è stata particolarmente utile non solo nella semplificazione del lavoro ma anche nel suo aggiornamento e del potenziamento.

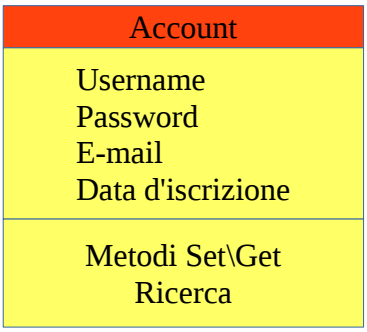
2.1 Utente

L'insieme degli utenti rappresenta le persone che useranno il programma per accedere alle informazioni contenute nel database. In particolare ogni utente ha la possibilità di aggiornare i dati che lo rappresentano ed effettuare ricerche per scoprire nuovi utenti che esistono nella rete.



Gli utenti sono visti come una gerarchia con alla base la classe **Account**. Dalla tale classe abbiamo due tipi di classi derivate: gli account **amministratori** e gli account **utente**. I primi si occupano alla gestione degli utenti, cioè hanno la possibilità di aggiungere o eliminare utenti e offrire servizi maggiori agli stessi. Gli utenti invece sono un ulteriore gerarchia avente **Basic** come classe base da cui derivano le classi **Business** ed **Executive**.

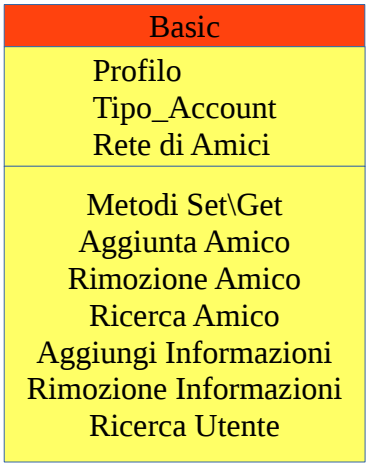
Descriviamo le vari classi:



2.1.1 Account

La classe base **Account** non ha nessuna funzione attiva all'interno del database. il suo scopo è semplicemente contenere le informazioni di base che un utente deve avere. Troviamo quindi i membri **username**, **password**, **email** e **data d'iscrizione**. La classe dispone di metodi per modificare i membri ed un metodo virtuale di ricerca che verrà implementato nelle classi superiori.

2.1.2 Basic



Basic è il primo utente vero e proprio. Contiene le informazioni personali dell'utente, una rete di amici ed una serie di funzioni per mantenere aggiornato il proprio profilo.

Profilo è una gerarchia di classi che contiene tutte le informazioni dell'utente che verrà analizzata in seguito. **Tipo_Account** specifica la tipologia di account ed infine la **rete di amici** che contiene gli utenti conosciuti dell'utente stesso.

Per gestire la rete di amici, l'utente Basic dispone di funzioni che gli consentono di trovare un amico, aggiungerlo o rimuoverlo. Gli stessi tipi di metodi sono disponibili anche per aggiungere le informazioni contenuti nel profilo.

Le classi **Business** ed **Executive** sono identiche alla classe base con l'unica eccezione del comportamento del metodo di ricerca.

2.1.3 Administrator

Administrator si occupa di gestire il database. L' amministratore è l'utente con i privilegi più alti all'interno della gerarchia. Tale superiorità è data dalla capacità di effettuare qualsiasi operazione che un utente può fare con l'aggiunta di funzioni che consentono l'inserimento e la cancellazione di un utente. Un amministratore è in grado di poter cambiare la tipologia di un utente.

Un amministratore è descritto da un **nome**,**cognome** e **ruolo** con relativi metodi per la modifica e l'estrazione di tali attributi.

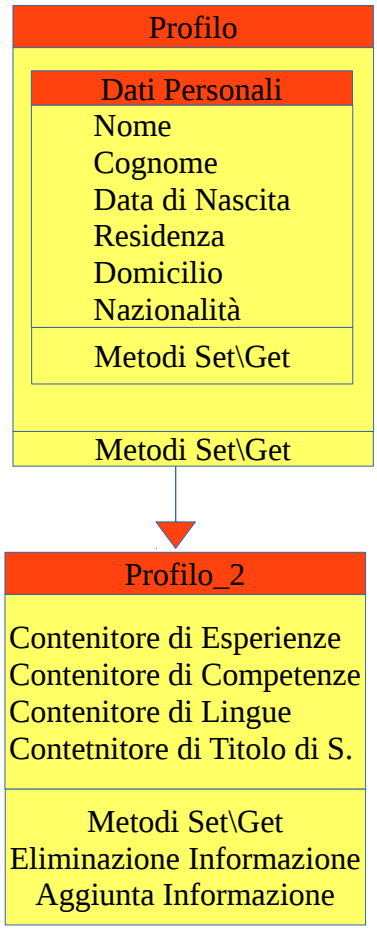
2.1.4 Profilo

Il profilo è una gerarchia di classi che descrive le informazioni dell'utente. Sono presenti dei contenitori che contengono informazioni che oltre a descrivere in maniera dettagliata le caratteristiche di un utente, permettono allo stesso di rendersi più visibile agli altri utenti all'interno della rete LinQedln.

La classe base è **Profilo** che contiene al suo interno una classe annidata chiamata **Dati Personali**.

Dati Personali memorizza le **informazioni anagrafiche** dell'utente. Troviamo i membri **nome**, **cognome**, **data di nascita**, **residenza**, **domicilio** e **nazionalità**. La classe è dotata di metodi set\get che permettono di modificare ed accedere ad ogni singolo attributo.

La classe derivata **Profilo_2** è un contenitore di altri contenitori. Se la classe Profilo si occupa dei dati anagrafici, la classe Profilo_2 estende il bagaglio di informazioni dell'utente gestendo i **dati professionali**. Troviamo quattro strutture dati contenenti oggetti che rappresentano elementi come le esperienze aziendali o le lingue parlate.



Ogni elemento è rappresentato da una classe che contiene i vari membri che la descrivono e le relative funzioni per accedervi.

Un' **Esperienza** descrive un esperienza lavorativa dell'utente ed è caratterizzata dal nome dell'azienda, la data di inizio e di fine, il ruolo ed una descrizione per l'aggiunta di dettagli specifici.

Una **Competenza** è una particolare capacità acquisita dall'utente in seguito ad una determinata esperienza o titolo di studio. È descritta dal nome, dal livello di conoscenza e da una relazionava descrizione.

Una **Lingua**, oltre al nome, contiene il livello di scritto ed il livello orale dell' utente. Un campo apposito informa se è la lingua madre dell'utente.

Titolo di Studi contiene il nome di un titolo di studio, la data di inizio, la data di fine ed il voto finale.

La classe Profilo_2 mette a disposizione opportuni metodi per accedere ai vari contenitori e modificare il contenuto.

2.1.5 Ricerca

La ricerca è il metodo che contraddistingue le varie classi all'interno della gerarchia utenti. Tale funzione ha lo scopo di cercare nel database un profilo che rispetti determinate caratteristiche volute dall'utente. Per interrogare il database un utente dispone di oggetti chiamati **objSearch** che descrivono un profilo da cercare. L'objSearch sarà inviato al database che una volta elaborata la ricerca risponderà con un **objResult** contenente la lista di utenti richiesti.
Le ricerche di ogni utente si distinguono in base al tipo di domande che un utente può fare al database. Un l'utente Basic può cercare un utente solo se ne conosce il cognome. Un cliente Bussiness, oltre per cognome, può cercare per competenza, mentre il tipo Executive dispone di objsearch che possono descrivere utenti che parlano determinate lingue o che sono in possesso di particolari titoli di studio.

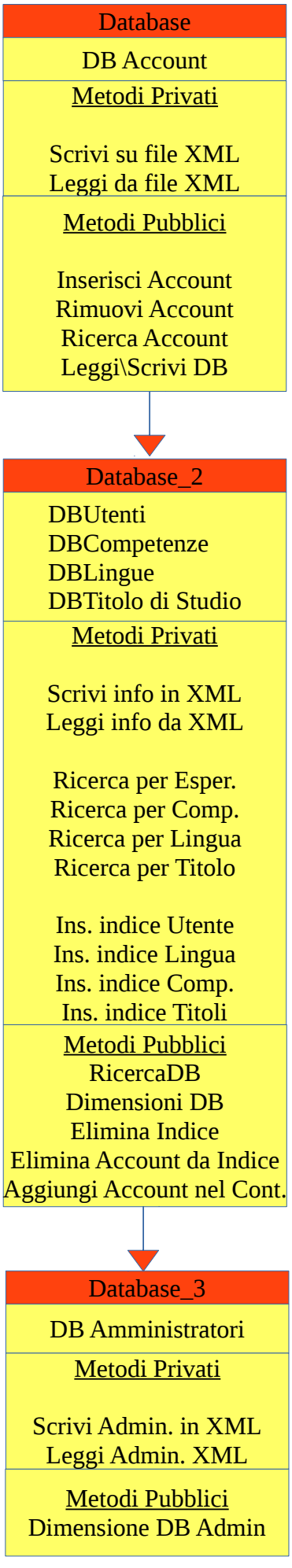
2.2 Database

Database è il cuore del software LinQedln. È una gerarchia di classi ed ogni derivazione contiene una struttura dati che memorizza le informazioni di ogni utente in uno specifico ordine. La classe base è **Database** mentre quelle derivate sono le classi **Database_2** e D**atabase_3.**

I dati sono fisicamente salvati nella memoria esterna usando un file XML. Ad ogni livello gerarchico, il database dispone di differenti funzioni per codificare l'informazione in entrata ed in uscita.

2.2.1 Database

Database ha una classe che memorizza tutti gli utenti usando come chiave il loro **username** (ciò fa di username la chiave primaria di ogni utente) che verrà spiegata nel prossimo paragrafo. Sono presenti tutti gli account ad esclusione degli amministratori che sono memorizzati nel livello più alto della gerarchia.



Il metodo **LeggiDB** comanda al database di leggere tutte le informazioni contenute all'interno del file XML e caricare in memoria tutti gli utenti salvati. Con questa operazione il database usa i propri metodi privati per reperire le informazioni che descrivono un account in formato XML per poi inserirle nella mappa DBAccount. La funzione **ScriviDB** fa esattamente il lavoro inverso, prende ogni utente presente nella mappa e lo salva nel file XML.

Le operazioni rimanenti gestiscono il **DBAccout**. È possibile inserire un nuovo account (via XML oppure su ordine di un amministratore), eliminarlo oppure cercarlo tramite il suo username.

2.2.2 Database_2

Database_2 è la forma evoluta di Database. Contiene lo stesso tipo di metodi ma oltre ad essere in un numero maggiore, si distinguono per il tipo di contenitore in cui agiscono. Sono presenti classi che memorizzano utenti tenendo in considerazione il loro cognome, le lingue parlate, le competenze ed infine i titoli di studio.

Le classi **DB-X** sono contenitori di utenti ordinati per una determinata caratteristica del proprio profilo. Un DB è composto da una mappa (**M1**) in cui ogni nodo è un contenitore di utenti **(M2)** che hanno una proprietà che combacia con la chiave di ricerca di M1. DB quindi suddivide gli utenti della rete in sottoinsiemi formati da utenti che hanno una determinata caratteristica comune. In DBCompetenze per esempio viene usato il nome della competenza come chiave di ricerca per M1.
Con il metodo pubblico di **ricercaDB** possiamo recuperare la mappa M2 dando come parametro un nome di una competenza. L'inserimento di una nuova mappa combacia con l'inserimento di una caratteristica posseduta da un utente fino a quel momento sconosciuta dal DB di riferimento. La rimozione di utenti da una mappa viene eseguita con la funzione rimuovi utenti; Non esistono mappe vuote, se svuotiamo una mappa di ogni account allora la mappa viene immediatamente distrutta.

I metodi privati di Database_2 hanno lo scopo di gestire le vari classi contenitrici. Oltre alle funzioni di codifica in XML di ogni tipo di proprietà, sono presenti metodi che gestiscono le vari chiavi all'interno dei DB e l'inserimento (o cancellazione) di un utente in un contenitore.

È possibile richiedere le statistiche del database utilizzando le funzioni che restituiscono il numero di elementi memorizzati in ogni classe contenitrice. All'arrivo di un oggetto **objSearch**, RicercaDB effettua la ricerca su due livelli. Nel primo viene cercato il contenitore richiesto. Nel secondo vengono selezionati gli utenti che rispettano un particolare attributo specificato nella richiesta. Tale funzione quindi

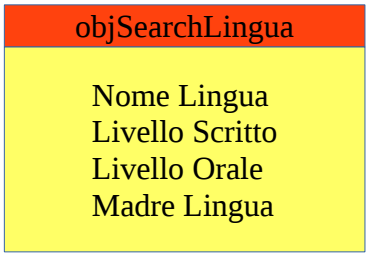
resterà relativamente efficiente anche con una forte crescita di utenti in quanto, essendo divisi in gruppi, sarà necessario trovare quel gruppo e applicare i filtri richiesti. È comunque possibile estendere le classi contenitrici DB-X sostituendole con classi derivate più efficienti.

2.2.3 Database_3

Database_3 è l'ultimo livello della gerarchia. È arricchito con il contenitore per gli amministratore ordinati per cognome. Al suo interno contiene un **DBAmministratori** con tutte le funzioni per la gestione della classe.

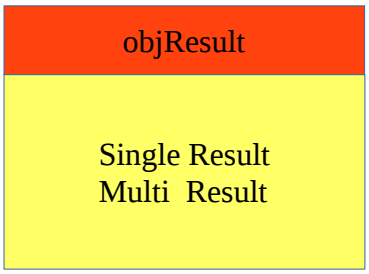
2.3 ObjSearch e ObjResult

ObjSearch e ObjResult sono gli oggetti utilizzati dagli utenti e dal database per comunicare e scambiarsi informazioni.



ObjSearch è una gerarchia di classi. La classe ObjSearch è la classe base che comunica solo con **Database**. Contiene un unico campo su cui si specifica lo username da cercare.

Le classi derivate fanno tutte riferimento a Database_2. Si distinguono per il tipo di contenitore che conterrà l'informazione da reperire. Ogni classe descrive un particolare caratteristica che deve possedere l'utente cercato. Nella figura è presente la classe ObjSearchLingua come esempio. Il primo campo indica l'insieme degli utenti che parlano una determinata lingua; i restanti campi servono per filtrare il risultato.



ObjResult invece è il risultato della ricerca che il database invia all'utente. Al suo interno ci sono tre campi: il primo contiene l'indice dell' utente cercato (usato solo nel caso che si cerchi per username), il secondo è un contenitore di utenti richiesti.

La classe objSearch contiene anche l'indirizzo del database a cui inviare le richieste di ogni utente.

3 GUI

La parte grafica è composta da tre principali finestre. La finestra di accesso, la finestra dell'utente e quella dell'amministratore.

3.1 Finestra di accesso

Nella finestra di accesso vengono richiesti nome e cognome dell'account che desidera accedere. Viene quindi creato il database ed interrogato sull'esistenza dell'utente che vuole accedere. In base alla risposta vengono aperte le finestre relative all'utente oppure all'amministratore.

3.2 Finestra dell'utente

La finestra dell'utente si distingue in due sezioni. Nella parte a sinistra sono presenti le informazioni dell'utente con la possibilità di aggiornare il profilo.

Nella parte destra sono presenti i comandi di ricerca. Sono disponibili diversi objSearch che possono essere

usati dagli utenti per interrogare il database. Il risultato è rappresentato in una tabella dove vengono elencati i profili trovati. Cliccando nel corrispondente pulsate l'utente può accedere alle informazioni dell'utente e togliere o aggiungere l'amicizia.

3.3 Finestra dell'amministratore

La finestra dell'amministratore è strutturata come quella dell'utente. A sinistra sono presenti le informazioni dell'amministratore ed una console per inserire o togliere utenti.

Nella parte destra sono presenti i comandi di ricerca. in questa sezione un amministratore può cercare un utente e aver la possibilità di eliminarlo o cambiare la tipologia.

4 Conclusioni

Il progetto è stato pensato e modellato con i cinque principi della programmazione SOLID con lo scopo di renderlo il più estensibile possibile.
Ogni insieme principale (account, database e objsearch) è formato da un componente base. Da tale componente, ed il relativo livello di sviluppo, il progetto si evolve gradualmente in ogni sua parte. Il programma si presenta facilmente potenziabile ed estensibile. Ciò è dimostrato dalla possibilità di migliorare componenti cruciali e di aggiungere nuove funzionalità e tipo di informazioni alle varie classi.
Grazie all'ereditarietà della classe profilo, un utente può essere arricchito da nuove classi che descrivono altri aspetti importanti riguardanti una carriera lavorativa. Il database a sua volta può essere aggiornato di conseguenza aggiungendo nuovi contenitori oppure modificando quelli già esistenti creando classi derivate. Infine una nuova derivazione degli objSearch può essere creata per consentire l'uso di tali aggiornamenti.

Nel progetto si possono notare tutti gli aspetti positivi dalla programmazione ad oggetti con ereditarietà. La classe Database è stata estesa e modificata in maniera semplice ed efficace ad ogni cambiamento apportato ad utente. Man mano che il profilo si arricchiva i cambiamenti venivano rispecchiati nel Database in modo naturale e soprattutto retro compatibile. Anche la classe utenti gode dei vantaggi dell'ereditarietà. Qualsiasi sia la tipologia, ogni utente viene visto dal database come account. In questo modo è possibile aggiungere altri tipi di account senza cosi dover modificare drasticamente il DB.

Le mappe sono state scelte come strutture dati principale per il database. La struttura ad albero infatti presenta notevoli livelli di efficienza nell'inserimento e ricerca di un dato elemento. La ricerca nel database è strutturata in modo da rendere minimo il numero di account da visitare.

Nel progetto non sono stati usati gli smartPointer per un'iniziale ed errata convinzione di un loro inutilizzo. La loro mancanza si nota soprattutto nel cambio tipologia di un utente che per avvenire costringe il database duplicare ogni singola caratteristica dell'utente (lingua, competenza ecc.) per poi cancellare quelle vecchie. Per questo motivo ogni classe ha il proprio distruttore implementato in modo da non lasciare sporcizia in memoria ed eliminare ogni oggetto che non sia associato ad un utente presente nel DB. Anche in questo caso l'ereditarietà ed il distruttore virtuale sono risultati utili alla causa.

La parte grafica è sprovvista della tecnologia MVC. Ogni finestra si scambia il puntatore dell'account connesso e gli eventuali risultati delle ricerche.

*Per accedere usare le seguenti credenziali: utente: "**admin**", password:"**admin**".*

Non usare le 'X' della finestra per uscire, cliccare sempre su gli appositi pulsanti di chiusura dei widget.