Progetto di Programmazione ad Oggetti

A.A. 2019/2020

Relazione di:
Terrani Giulia - 1142468

Altri componenti:
Fenu Lucia - 1125521

Bcc Unipd

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO MATEMATICA

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

INDICE

1. Funzionalità

Il progetto realizzato è stato pensato per essere utilizzato dal personale dipendente di una banca.
Dopo aver effettuato la registrazione e l'accesso al programma, l'utente potrà inserire, modificare o eliminare dipendenti, se appartiene all'Ufficio del personale, o clienti, se lavora allo Sportello, visualizzare e gestire i loro dati per effettuare calcoli od ottenere determinate informazioni di interesse. Se il dipendente che ha effettuato l'accesso è un operatore a contatto con i clienti della banca, e quindi lavora allo sportello, potrà, oltre a gestire le informazioni relative ai clienti, anche inserire, eliminare ed effettuare operazioni sui servizi offerti dalla banca a loro associati.

2. Gerarchie dei tipi

Il programma è stato realizzato seguendo il pattern MVC, separando la parte del modello dalle parti relative a controller e vista.
Il modello presenta due gerarchie tra loro collegate, una relativa alle persone e una ai servizi offerti dalla banca.

Per quanto riguarda la gerarchia relativa alle persone essa presenta:

- ***Persona***: la classe base astratta della gerarchia da cui derivano le classi astratte:
    - ***Operatore***: per gestire i dipendenti della banca;
    - ***Ospite***: per gestire i clienti della banca;

  Da queste due classi astratte derivano le classi concrete:
    - ***Dipendente***: per gestire le persone che sono esclusivamente dipendenti della banca;
    - ***Cliente***: per gestire le persone che sono esclusivamente clienti della banca;
    - ***Dipendente_cliente***: per gestire le persone che sono dipendenti e anche clienti della banca.

All'interno della gerarchia si ha un esempio di ereditarietà multipla a diamante realizzata dalle classi: Persona, Operatore, Ospite e Dipendente_cliente, quest'ultima classe acquisisce infatti le funzionalità sia della classe Operatore che della classe Ospite che a loro volta derivano virtualmente da Persona.
La classe contenitore realizzata per gestire le persone è la classe ***Soggetti*****,** implementata come una lista doppiamente linkata.

Per quanto riguarda la gerarchia relativa ai servizi essa presenta:

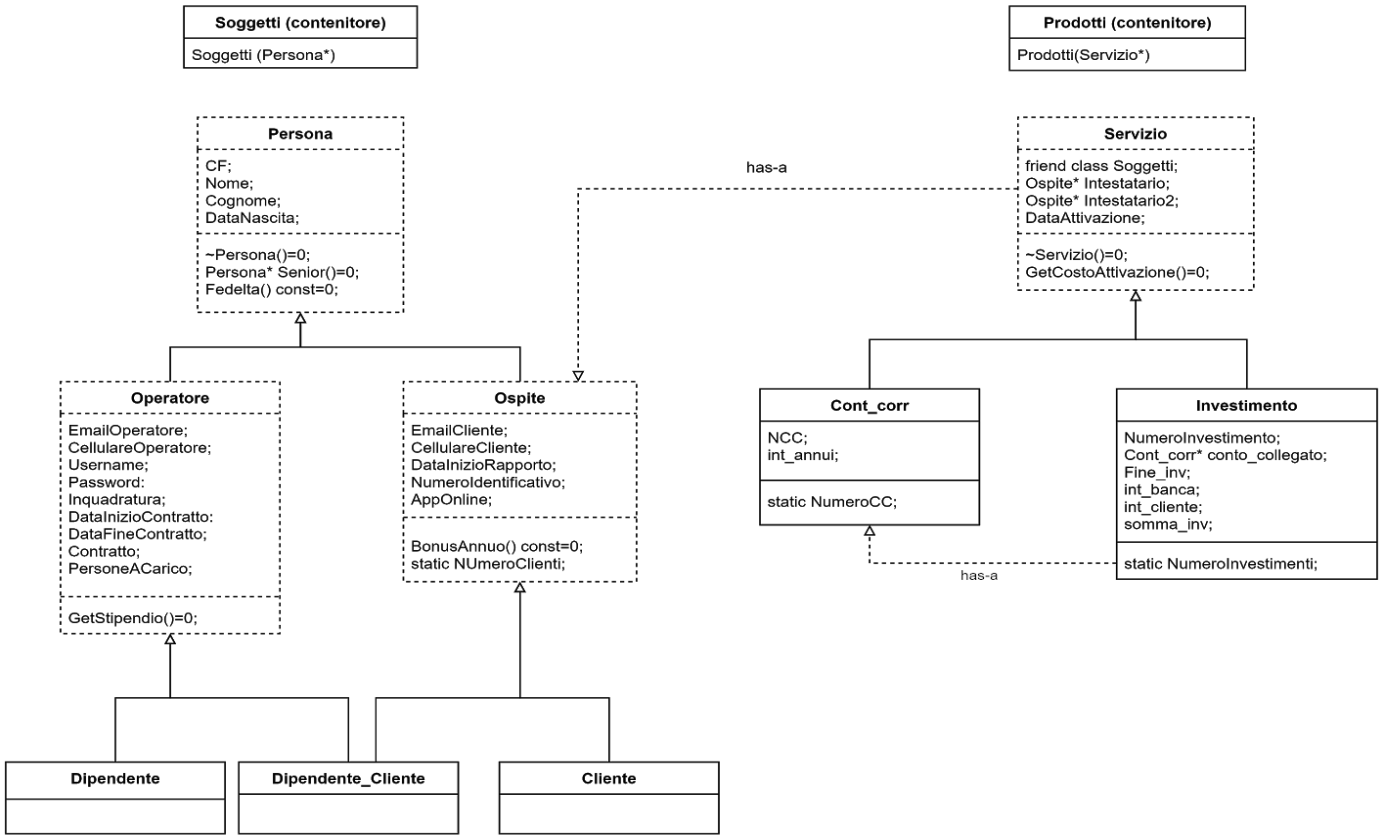
- ***Servizio*****:** la classe base astratta della gerarchia da cui derivano le classi concrete:
    - ***Cont_Corr***: per gestire i conti correnti degli ospiti;
    - ***Investimento*****:** per gestire gli investimenti degli ospiti;

Dato che ogni servizio esiste solo se associato a uno o più intestatari, attraverso una relazione di tipo has-a è stato possibile collegare questa gerarchia con la gerarchia delle persone, infatti, ogni servizio ha 2 puntatori (Intestatario1 e Intestatario2) che puntano ad un ospite.
Un'altra relazione has-a si ha tra l'investimento e il conto corrente ad esso collegato.
La classe contenitore realizzata per gestire i servizi è la classe ***Prodotti*****,** implementata come una lista doppiamente linkata.

Il controller comprende un'unica classe, ***MainWindowController***, che gestisce le connessioni tra il modello e le viste utilizzate attraverso dei puntatori.

Infine la vista comprende le classi:

- ***Login_window***: per la pagina iniziale che comprende il form per registrazione e login;
- ***MainWindow***: per la pagina principale;
- ***GestionePersone***: per le finestre secondarie per la gestione di persone e servizi;
- ***Table_lists***: per tutte le tabelle presenti all'interno del programma.



3. Polimorfismo

Oltre ai distruttori virtuali per gestire il corretto rilascio di memoria sia per Persona che per Servizio, sono presenti i seguenti metodi virtuali in grado di sfruttare il polimorfismo:

- GetStipendio(): permette di ottenere il valore dello stipendio dell'operatore in base al numero di persone a suo carico, se l'operatore è dipendente_cliente allora avrà un bonus aggiuntivo giustificato dal fatto di essere anche cliente della banca.
  Il metodo può essere invocato dalla pagina principale dell'operatore che ha effettuato l'accesso e che lavora nell'ufficio del personale, un operatore dello sportello non deve infatti aver accesso alle informazioni relative ai suoi colleghi.

- Fedelta(): permette di ottenere un numero che determina la fedeltà della persona rispetto al rapporto con la banca, per i clienti dipende dal numero di anni come clienti della banca, per i dipendenti dal numero di anni come dipendenti della banca e dalla loro posizione all'interno della banca. Il metodo non può essere invocato tramite la vista ma viene utilizzato da altri metodi invocabili.

- Senior(): restituisce la persona se questa ha un valore determinato dal metodo Fedelta() superiore ad una certa soglia per clienti e dipendenti, mentre per i dipendenti_clienti restituisce la persona se questa ha un valore superiore ad un certa soglia sia come cliente che come dipendente. Come il metodo precedente non viene invocato dalla vista ma viene utilizzato da un metodo invocabile da qualunque operatore per visualizzare la lista dei dipendenti o dei clienti "Senior".

- BonusAnnuo(): restituisce un bonus annuo da corrispondere all'ospite in base alla durata del rapporto di clientela e dall'utilizzo o meno dell'app, il metodo è invocabile solo se l'operatore che ha effettuato l'accesso è un operatore dello sportello.

- GetCostoAttivazione(): permette di ottenere il costo per l'attivazione del servizio. Se il servizio è un conto corrente il costo varia in base alla fedeltà del cliente partendo dal metodo virtuale Fedelta(), se il servizio è un investimento oltre a questo verranno considerati la durata dell'investimento e da quanto tempo è stato aperto il conto corrente ad esso collegato. Questo metodo può essere invocato come il precedente solo da operatori dello sportello.

Le gerarchie così realizzate possono essere facilmente estese, pensiamo ad esempio alla possibilità di aggiungere tra le persone i dirigenti o tra i servizi le assicurazioni, e i precedenti metodi facilmente modificabili e adattabili.

All'interno di tutto il programma oltre all'utilizzo di questi metodi virtuali, si utilizza il dynamic_cast per determinare il tipo dinamico dei puntatori ed effettuare operazioni in base ad esso.

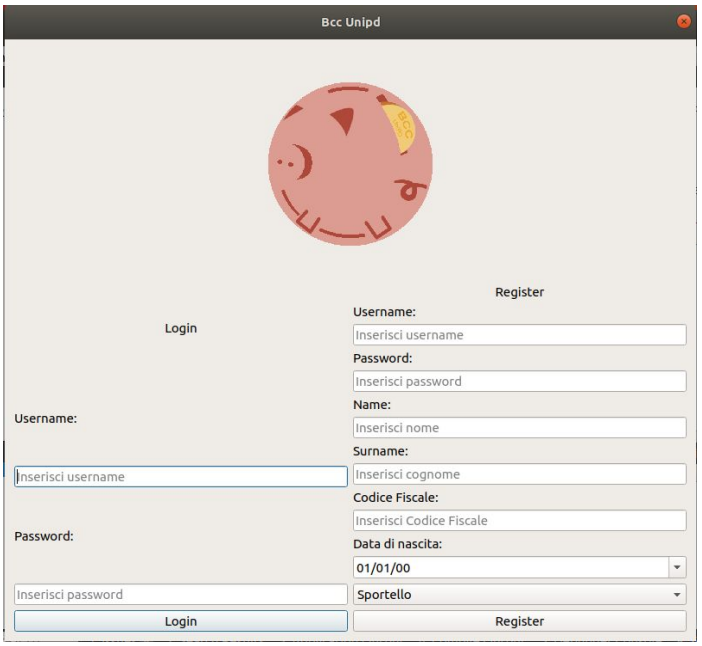
4. Note sulla progettazione

Di seguito sono riportate alcune scelte sulle progettazione del programma.

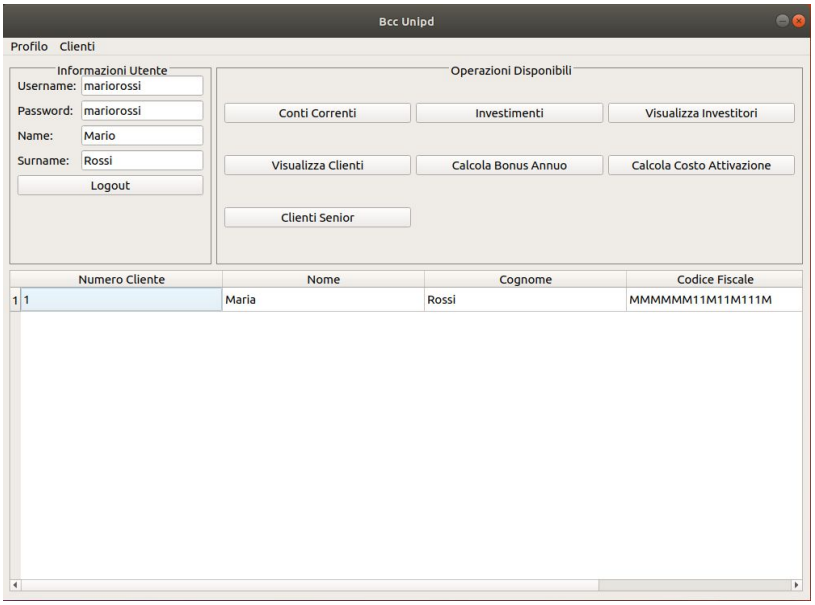
1) Si è scelto di utilizzare due doubly linked list per gestire le persone e i servizi invece dei vettori perché abbiamo ritenuto che il numero di inserimenti e rimozioni di questi oggetti sia significativo all'interno di una banca, allo stesso tempo però, rispetto ad una singly linked list, ha comunque migliori performance durante le operazioni di ricerca;
2) Nell'implementare il progetto si è seguito il pattern MVC in modo da tenere separata la parte logica da controller e vista, si è scelto però di utilizzare la classe QDate per i campi dati che indicavano date senza crearne una nuova in grado di gestirle, perché non hanno una rilevanza centrale all'interno del modello ma vengono utilizzate solo in alcuni controlli;
3) Inizialmente si era pensato di gestire con due liste differenti operatori e ospiti, le operazioni sarebbero state più efficienti ma allo stesso tempo era molto più complesso gestire le due liste considerando la presenza di eventuali dipendenti_clienti, per questo motivo si è ritenuto di dare più importanza alla correttezza del programma creando quindi un'unica lista comprendente tutte le persone e gestendo le varie operazioni attraverso il cast dinamico, in questo modo è inoltre stato possibile sfruttare al meglio l'utilizzo del polimorfismo.
4) Dato che gli operatori dello sportello non possono gestire e quindi vedere i dipendenti e viceversa gli operatori dell'ufficio del personale non possono farlo con i clienti, la creazione e l'eliminazione di un dipendente_cliente avviene in modo trasparente all'utente. Se è stato inserito un cliente e successivamente viene inserito un dipendente con gli stessi dati personali, sarà il programma a creare un nuovo dipendente_cliente con tutti i dati relativi alla persona ed eliminerà il precedente cliente.
5) I clienti possono essere inseriti anche in un momento successivo rispetto alla reale data di inizio rapporto ma ovviamente non possono essere inseriti con data di inizio rapporto successiva alla data corrente. I conti correnti e gli investimenti non possono essere inseriti con data di apertura successiva alla data corrente.
   I dipendenti non possono essere inseriti con data di inizio rapporto successiva alla data corrente.
6) Se viene eliminato un conto corrente intestato a due persone, verrà eliminato per entrambe, di conseguenza può essere eliminato solo se nessuno dei due intestatari ha il conto collegato ad un investimento ancora attivo.

5. Manuale utente

All'avvio del programma viene visualizzata la schermata per registrazione e login dell'utente.

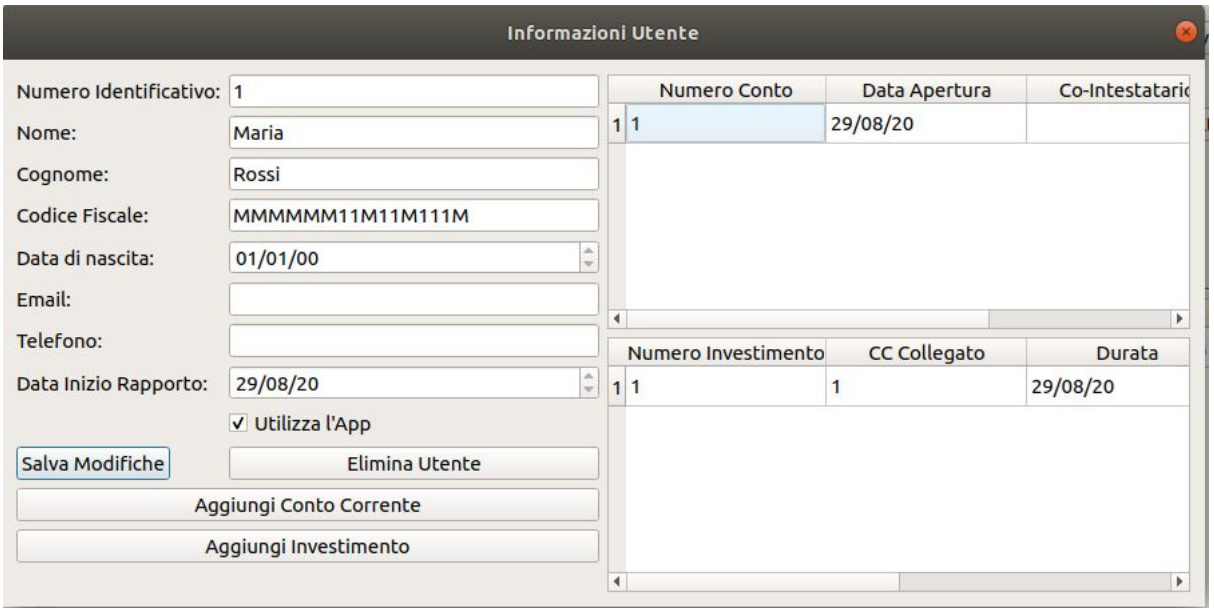


Effettuato l'accesso verrà visualizzata la pagina principale comprendente a sinistra i dati principali della persona che ha effettuato la login, a destra una serie di operazioni attuabili a seconda del ruolo della persona loggata e in basso una tabella con la lista delle persone gestibili dall'utente.
Attraverso la barra del menù è possibile visualizzare i dati del proprio profilo, aggiungere e cercare nuove persone.

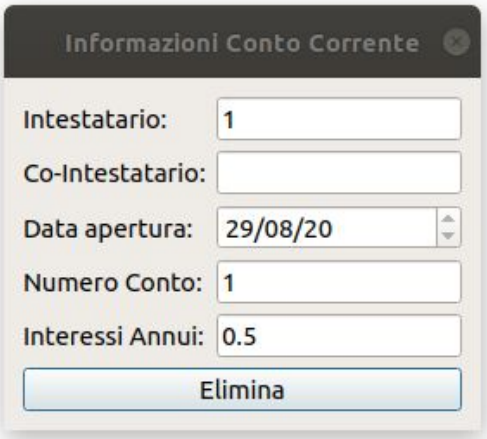


La tabella della pagina principale è sempre cliccabile e a seconda che siano visualizzate persone o servizi, quando l'utente fa doppio click su una qualunque cella di una sua riga si aprirà rispettivamente una finestra che permette la gestione della persona o del servizio selezionato.
Se la tabella visualizza Ospiti, allora la finestra di gestione che si aprirà cliccando su una sua riga comprenderà oltre alle informazioni dell'ospite anche due tabelle con i suoi conti correnti e i suoi investimenti.

Da qui sarà possibile modificare l'ospite e gestire i suoi servizi.



Se viene cliccato un conto corrente o un investimento nelle tabelle presenti nella finestra di modifica dell'ospite, verranno rispettivamente visualizzate le loro informazioni e da qui sarà possibile eliminare il servizio



6. Ambiente di sviluppo e comandi

Lo sviluppo del progetto e il suo test sono stati effettuati nella virtual machine fornitaci.

Per compilare il codice da terminale ed eseguirlo occorre invocare i seguenti comandi:

qmake progetto_p2.pro → make → ./progetto_p2

7. Suddivisione del lavoro progettuale

Il lavoro è stato suddiviso considerando le due gerarchie presenti, la parte di cui mi sono occupata è quella riguardante le persone, quindi tutti i file .h/.cpp:

- Persona
- Operatore
- Ospite
- Dipendente
- Dipendente_cliente
- Cliente
- MainWindowController (parte delle persone)
- Login_window
- MainWindow (parte delle persone)
- GestionePersone (parte delle persone)
- Table_Lists (parte delle persone)

Il collegamento delle due parti e il test del programma sono state svolte insieme.

Il tempo impiegato è stato di circa 60 ore così suddivise:

- ❏ Analisi preliminare del problema: 1 ora
- ❏ Progettazione modello: 4 ore
- ❏ Progettazione GUI: 2 ore
- ❏ Apprendimento libreria Qt: 15 ore
- ❏ Codifica modello: 10 ore
- ❏ Codifica GUI: 20 ore
- ❏ Debug e test: 8 ore

Il monte ore supera di 10 ore il totale previsto dovuto al tempo dedicato all'apprendimento della libreria Qt, alla codifica della vista e al relativo collegamento con il controller.