UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI MATEMATICA
Corso di Laurea in Informatica

Corso di Programmazione ad Oggetti

Relazione del progetto

BankQ

Presentato da

FRANCESCHINI MARCO matricola n. 1072066

Anno Accademico 2015-2016

Indice

1. **Scopo del progetto**

Lo scopo del progetto BankQ è quello di simulare un home banking garantendo alcune delle operazioni più diffuse all'interno di questi sistemi.

È prevista la presenza di tre tipi di utenti:

- amministratori;
- utenti basic;
- utenti pro.

I primi non possiedono un conto BankQ ed hanno il compito di svolgere alcune richieste effettuate dagli utenti, come chiudere un conto o assegnare un bonus. Inoltre hanno la possibilità di aggiungere nuovi utenti al sistema.

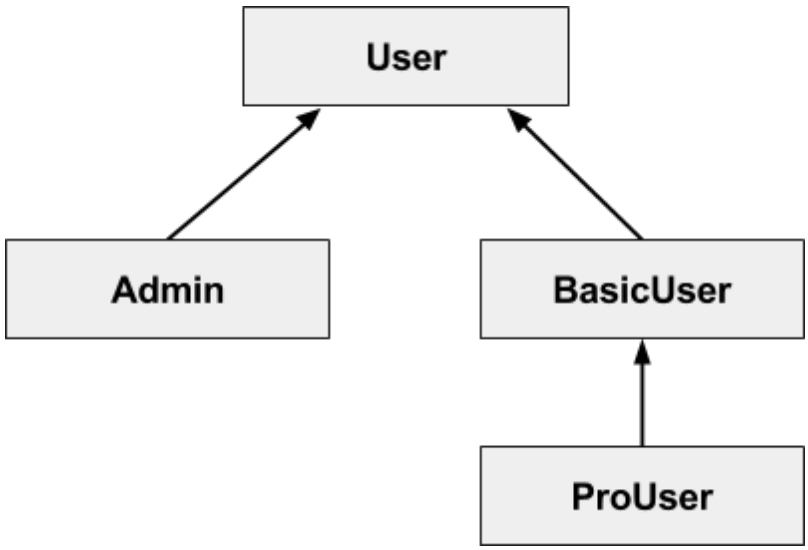
Gli altri due invece fanno parte degli utenti che possiedono un conto. Gli utenti basic, a differenza di quelli pro, hanno meno operazioni eseguibili ed una maggiore tassazione del conto. Un utente Basic può passare a Pro solamente se il saldo è di almeno 100000 (centomila) €.

Tutti gli utenti del sistema hanno in comune dei dati di base e differiscono per campi specifici al tipo di appartenenza, come bonus, tasse e salario.

Gli utenti hanno la possibilità di interagire tra di loro; gli utenti basic e pro possono mandare dei messaggi agli amministratori i quali possono effettuare le operazioni già citate. Inoltre gli utenti pro e basic possono effettuare delle ricariche ad altri conti, purché essi esistano.

2. **Descrizione della gerarchia**

La gerarchia è formata da una classe base User, con due classi derivate Admin e BasicUser. Quest'ultima è a sua volta una classe base per la classe derivata ProUser. La gerarchia polimorfa appena descritta presenta solamente classi concrete, quindi sono istanziabili. Questa scelta è dovuta all'utilizzo di una lista di User per contenere le informazioni presenti nel database. Lo schema della gerarchia si presenta quindi in questo modo:



2.1 **Incapsulamento delle informazioni logiche**

Tutte queste classi si occupano di gestire le informazioni degli utenti, cambia solamente il tipo di utente a cui fanno riferimento.

La classe base User ha come campi: nome, cognome, indirizzo, numero di telefono, codice fiscale, username e PIN. Oltre a questi un oggetto della classe Admin aggiunge il salario, invece nel caso di un BasicUser si aggiungono: il numero di conto, il saldo e una tassa con valore statico. ProUser estendendo quest'ultima aggiunge: un bonus, una nuova tassa, entrambi statici, ed un campo richiesta, per sapere se è già stato richiesto il bonus anticipato.

La classe base che si occupa di gestire gli utenti generici è "user.h" e gli amministratori vengono invece dichiarati in "admin.h". I dati riguardanti un conto bancario comune; saldo, numero di conto e tassa, sono dichiarati dentro "basicuser.h" e vengono estesi da "prouser.h" per la gestione degli utenti pro.

2.2 **Messaggi**

La classe adibita a dichiarare i messaggi è Message, questa è formata da un destinatario, un mittente ed il testo del messaggio. Escludendo i messaggi scritti da un utente per un amministratore, gli altri vengono creati in automatico dopo lo svolgimento di determinate operazione, come la ricezione di un ricarica o di un bonus oppure per il passaggio del tipo di conto da basic a pro.

3. **Contenitore**

Come richiesto nelle specifiche del progetto è stato utilizzato un contenitore dichiarato nella classe Container ("container.h"). Viene impiegata una lista, di smart pointer, essi vengono dichiarati nell'omonima classe annidata e sono formati da un nodo. Quest'ultimo è composto da un intero che

rappresenta lo shift del nodo e da un campo per l'informazione rappresentato da un template. Per quanto riguarda la lista, si impiegano due smart pointer come puntatori dell'inizio e della fine di essa.

3.1 **Metodi**

I metodi principali che permettono il funzionamento del contenitore sono:

- *getSize*: restituisce il numero di elementi all'interno del contenitore;
- *isEmpty*: verifica se il contenitore è vuoto;
- *at*: restituisce il valore del nodo nella posizione passata;
- *push_back*: inserisce in coda l'elemento passato;
- *remove*: rimuove il nodo nella posizione passata;
- *replace*: sostituisce il valore di un nodo, nella posizione passata, con un altro passato.

3.2 **Operatori**

Per facilitare l'utilizzo del contenitore sono stati ridefiniti alcuni operatori:

- *operator=*: per fare l'assegnazione tra i nodi del container;
- *operator[]*: restituisce il valore del nodo nella posizione passato.

3.3 **Iteratori**

Il contenitore è munito di iteratori dichiarati nella classe Iteratore, annidata dentro Container. Essi implementano un puntatore ad uno smart pointer. Così, come per il contenitore vengono descritti i principali operatori ridefiniti:

- *operator==* e *!=*: confrontano se un elemento puntato è uguale o meno a quello puntato da un analogo iteratore;
- *operator++*: incrementa lo shift del nodo putanto;
- *operator**: restituisce il valore del nodo puntato.

4. **GUI**

La parte logica viene mantenuta separata dall'interfaccia grafica. Al seguito di un click su di un bottone vengono eseguite le relative operazioni del caso.

Ad eccezione della classe Container, tutte le classi mantengono usa separazione tra dichiarazioni e ridefinizioni dei metodi. Le prime si trovano nei file con estensione ".h" le seconde, invece, in quelli con estensione ".cpp".

La parte riguardante l'interfaccia grafica è contenuta in "userinfo", per le schermate degli utenti, "admininfo", per quanto riguarda gli amministratori e in "mainwindow" è contenuta la finestra di login. Tutte e tre sono state realizzate con dei *QWidget* che ereditano da *QDialog* nel caso dell'interfaccia relativa agli utenti e *QMainWindow* per quanto riguarda la schermata di login.

Come appena accennato la suddivisione grafica riguarda i due tipi di utenti principali. Entrambe hanno un layout a schede, in modo da rendere facile, veloce ed intuitiva la navigazione. Queste schede differiscono in base al tipo di utente che accede a BankQ:

- **Basic e Pro**
    - *Info*: contiene le informazioni relative all'account e al conto dell'utente, inoltre da questa scheda è possibile uscire dalla sessione o chiudere il proprio conto;
    - *Ricarica*: in questa scheda è possibile ricaricare un altro conto, purché esista;
    - *Invia Messaggio*: da qui un utente può inviare un messaggio agli amministratori per eventuali richieste;
    - *Messaggi*: nella tabella presente in questa schermata un utente può visualizzare i propri messaggi ricevuti, come quello relativo ad una ricarica ricevuta oppure per il passaggio ad un nuovo tipo di utente.
- **Pro**
    - *Richiedi Bonus*: in questa scheda un utente Pro può richiedere il bonus anticipato, richiedibile una sola volta.
- **Amministratore**
    - *Info*: contiene le informazioni relative al proprio account, è inoltre presente, come per gli altri utenti, il pulsante per terminare la sessione;
    - *Aggiungi Utente*: qui un amministratore può inserire un nuovo utente all'interno del sistema a patto che le informazioni inserite siano coerenti, come l'username non occupato oppure il PIN troppo corto o lungo;
    - *Modifica Utente*: in questa scheda è possibile, dopo averlo selezionato, modificare un utente, rispettando sempre i vincoli delle informazioni da inserire;
    - *Rimuovi Utente*: nel caso in cui un utente faccia richiesta di chiudere il proprio conto, l'amministratore può rimuovere l'account con conseguente perdita di tutti i dati relativi;
    - *Assegna Bonus*: se un utente Pro ha richiesto il bonus anticipato questa scheda permette di assegnarlo a esso, ma anche a tutti gli altri in caso di necessità. È anche possibile "sbloccare" gli utenti per un eventuale riassegnazione del bonus;
    - *Richieste*: in questa scheda è possibile visualizzare le richieste che arrivano dagli utenti, come messaggi specifici oppure richieste di chiusura del conto.

4.1 **Manuale utente della GUI**

La prima schermata riguarda l'accesso al sistema BankQ, vengono richiesti username e password (PIN numerico di cinque cifre), come nei più comuni sito web, e per accedere basta premere il bottone "Accedi".

Può succedere che dopo aver effettuato l'accesso vengano notificati degli avvisi tramite finestre pop-up, ad esempio per la presenza di messaggi in arrivo.

Come già spiegato il layout è a schede e per navigare basta premere sui titoli che esse hanno. Nel caso in cui si abbia intenzione di tornare alla schermata di login basta premere sul pulsante "Esci" situata nella scheda "Info".

Alcune delle operazioni possibili richiedono il riempimento di caselle di testo e seguite da un click su di un bottone, altre invece, per l'inserimento di dati richiedono l'utilizzo di combo box. Ad esempio nella scheda per gli amministratori, "Modifica Utente", le varie line edit vengono riempite in automatico dopo la selezione dell'utente utilizzando l'apposita combo box posta nella parte superiore della schermata, è da notare che, sempre in questa scheda, non è possibile modificare il fatto che l'utente abbia già richiesto il bonus (se è pro, modificabile nella relativa scheda) ed il saldo. Altre restrizioni riguardano il PIN; solo numerico e composto da almeno una cifra diversa da zero, il numero di telefono; solo numerico come il saldo ed il numero di conto.

Sia per gli amministratori che per gli utenti i messaggi in arrivo vengono visualizzati su di una una tabella. È possibile eliminare un singolo messaggio, dopo aver confermato l'azione, effettuando un click su di esso oppure, utilizzando il bottone sotto la tabella, si possono cancellare tutti.

Inoltre, certe operazioni, considerate più rilevanti, hanno bisogno di una conferma per procedere, come ad esempio la ricarica verso un altro utente oppure l'eliminazione di un conto da parte dell'amministratore.

5. **Modellazione del database**

Vengono impiegati due database in formato XML "users.xml" e "messages.xml", il primo contiene tutti gli utenti, il secondo invece i messaggi. Questa separazione evita un caricamento o la scrittura di uno dei due quando non serve. Entrambi memorizzano i dati sotto forma di tag e vengono letti e scritti utilizzando i metodi forniti dalle librerie Qt *QXmlStreamReader* e *QXmlStreamWriter*.

5.1 **Metodi**

Sia la classe DataBase che MessagesDataBase definite utilizzano queste due librerie rispettivamente nel metodo "load" e "write". Come suggerisce il nome il primo carica nel contenitore il contenuto del database ed il secondo scrive nel file xml ciò che è presente dentro di esso. Sono presenti anche altri metodi che interagiscono con il contenitore permettendo l'esecuzione di certe operazioni di BankQ.

I metodi per il database degli utenti sono quindi:

- *load*: carica nel contenitore il contenuto del database;

- *verifyAdmin*: verifica se l'username passato è di un amministratore;
- *verifyExistingUsername*: verifica se l'username passato esiste già;
- *verifyExistingCountNumber*: verifica se il numero di conto passato esiste già;
- *verifyLogin*: verifica che i dati di login siano corretti;
- *addUser*: aggiunge un utente al contenitore;
- *getUser*: ritorna l'oggetto di tipo User corrispondente all'username passato;
- *getUserByCountNumber*: ritorna l'oggetto di tipo User corrispondete al numero di conto passato;
- *verifyStillSame*: verifica che il tipo di utente sia coerente con il saldo del conto, in caso negativo provvede a cambiare tipo all' utente;
- *remove*: rimuove un utente dal database e dal contenitore;
- write: scrive il contenuto del contenitore nel database;
- *charge*: effettua le operazioni di ricarica del conto dell'utente passato e decrementa quello del chiamante;
- *getUserNoAdmin*: ritorna un contenitore con gli utenti di BankQ;
- *getUserNoRequest*: ritorna un contenitore di utenti pro che non hanno richiesto il bonus anticipato;
- *verifyPro*: verifica se sono presenti utenti pro all'interno del contenitore;
- *verifyRequest*: verifica se ci sono utenti pro che hanno ricevuto il bonus anticipato;
- *giveBonus*: assegna il bonus all'utente pro passato (se non lo ha già ricevuto);
- *giveBonusToAll*: assegna il bonus a tutti gli utenti pro che non lo hanno già ricevuto;
- *unlockAll*: setta il campo "request" degli utenti pro a "false";
- *replace*: passando due utenti permette di sostituire il secondo al posto del primo all'interno del contenitore e del database.

Invece per quanto riguarda il database dei messaggi si ha:

- *loadMessages*: carica nel contenitore il contenuto del database;
- *countMessage*: ritorna il numero di messaggi per un utente;
- *getMessageByUser*: ritorna un contenitore con i messaggi per un utente;

- *deleteMessages*: rimuove tutti i messaggi per un utente;
- *addMessage*: aggiunge il messaggio passato al contenitore;
- *writeMessages*: scrive il contenuto del contenitore nel database;
- *deleteOneMessage*: elimina un messaggio preciso.

6. **Ambiente di sviluppo**

Il progetto è stato sviluppato in ambiente Elementary OS 0.3 (basato su Ubuntu 14.04 LTS) utilizzando Qt Creator 3.0.1 basato su Qt 5.2.1.

“””Sono stati svolti successivamente dei test sul suo funzionamento anche nella macchine del laboratorio LabTA utilizzando un ambiente Ubuntu con Qt Creator *2.41. basato su Qt 4.8.1 con compilatore GNU g++ 4.6.3 64bit*”””

7. **Compilazione e utilizzo**

Viene fornito un database già popolato per facilitare la fase di analisi e test, alcuni dati per l’accesso sono:

Username	**Password**	**Tipologia**
admin	12345	amministratore
prossi	01234	utente basic
mfrances	54321	utente pro