INFO-F105 - Langages de programmation 1
Projet C++

Luciano Porretta

Année académique 2015-2016

Définition du problème

On souhaite mettre au point une librairie pour la gestion de comptes bancaires. Pour représenter cela, nous vous demandons de créer parmi l'approche ADT une classe `Bank` contenant une liste de comptes bancaires. L'ADT `Bank` comportera comme attribut un pointeur `_head` sûr un objet de type `_Account` (cette nouvelle classe qui représente les comptes sera donc imbriquée dans `Bank` et sera privée). Voici le squelette de `Bank` :

Code 1– ADT Bank

```
class Bank{
private:
  class _Account{
  public:
    _Account(string, float);
  private:
    string _client;
    float _balance;
    Account *_next;
  };
  Account *_head;
public:
  Bank() : _head(nullptr) {};
  float withdraw(string, float);
  void createAccount(string, float);
  float deleteAccount(string);
  void mergeAccounts(string, string);
  void displayAccounts();
};
```

Dans la classe `_Account` nous avons `_client` qui represente le nom du client du compte, `_balance` qui stocke le solde du compte, et `_next` qui est un pointeur vers un autre compte. Attention, il faudra empêcher la création d'un compte mal défini où le solde est négatif où le nom du client est vide. Il faudra bien entendu aussi ajouter les getters/setters nécessaires.

La liste des comptes bancaires doit satisfaire les contraintes suivantes :

— chaque client a un seul compte bancaire ;
— chaque compte bancaire peut avoir seulement un solde positive ou nul ;
— le retrait sur un compte bancaire est seulement possible si le solde est positif ;
— le dépôt sur un compte bancaire est seulement possible si le montant à deposer est positif ;
— la fusion entre deux comptes est seulement possible pour des comptes déjà existants ;

Nous vous demandons de réaliser les six méthodes suivantes pour la classe `Bank` :

Code 2– Signature des fontionnes à réaliser

```
float withdraw(string, float);
float deposit(string, float);
void createAccount(string, float);
float deleteAccount(string);
void mergeAccount(string, string);
void displayAccount();
```

La première méthode permet de prélever le montant `money` depuis le compte du client `_owner` et de renvoyer le montant retiré. La deuxième méthode permet de déposer le montant `money` sûr le compte du client `owner` et de renvoyer le nouveau solde. La troisième méthode permet de créer un compte bancaire avec `money` comme solde et `_client` comme client. La quatrième méthode permet d'effacer un compte bancaire associé au client `_client` et de renvoyer le solde corresponant. La cinquième méthode permet de fusionner les comptes associés aux clients `_client1` et `_client2`. Le resultat de l'execution de cette fonction est d'effacer le compte associé aux clients `_client1` et `_client2` afinde créer un nouveau compte ayant comme client la concatenation de leur nom et comme solde la somme de leur solde. Enfin, la sixième méthode permet d'afficher à l'écran le nom du client et le solde de tous les comptes stockés dans la banque.

Evidemment, vous êtes libres de rajouter des méthodes au squelette que ce soit pour `_Account` où de `Bank` si cela est pertinent. Faites attention à bien libérer la mémoire de tous les compes devenus inutiles.

Illustration

Vous trouvez ci-dessous une classe Test pour l'ADT `Bank` et son exécution.

Code 3– Classe de Test

```
#include "Bank.hpp"
void main(){
  Bank b;
  std::cout << "Create Tom's account with 10.000$"<<std::endl;
  b.createAccount("Tom", 10000);
  std::cout << "Create Dick's account with 20.000$"<<std::endl;
  b.createAccount("Dick", 20000);
  std::cout << "Create Harry's account with 30.000$"<<std::endl;
  b.createAccount("Harry", 30000);
  std::cout << "Create Tom's account with 15.000$"<<std::endl;
  b.createAccount("Tom",15000);
  std::cout << "Create Bill's account with -20000$"<<std::endl;
  b.createAccount("Bill",-20000);
  std::cout << "Print list of accounts:"<<std::endl;
  b.displayAccounts();
  std::cout << "withdraw 5.000$ from Tom's account."<<std::endl;
  b.withdraw("Tom",5000);
  std::cout << "withdraw 11.000$ from Dick's account."<<std::endl;
  b.withdraw("Dick",11000);
  std::cout << "withdraw 21.300$ from Dick's account."<<std::endl;
  b.withdraw("Dick",21300);
  std::cout << "Print list of accounts:"<<std::endl;
  b.displayAccounts();
  std::cout << "Close Dick's account."<<std::endl;
  balance=b.deleteAccount("Dick");
  std::cout << "Dick' account closed. He took "<<balance<< "$."<<std::endl;
  std::cout << "Merge Tom and Harry's accounts"<<std::endl;
  b.mergeAccounts("Tom","Harry");
  std::cout << "Print list of accounts:"<<std::endl;
  b.displayAccounts();
}
```

Figure 1 – Test Output

```
Create Tom's account with 10.000$
Create Dick's account with 20.000$
Create Harry's account with 30.000$
Create Tom's account with 15.000$
DENIED OPERATION! --> Tom's account already exists.
Create Bill's account with -20000$
ERROR: Can't create account with balance<=0
Print list of accounts:
Name: Tom   Balance: 10000$
Name: Dick  Balance: 20000$
Name: Harry Balance: 30000$
withdraw 5.000$ from Tom's account.
withdraw 11.000$ from Dick's account.
withdraw 21.300$ from Dick's account.
DENIED OPERATION! --> Not enough money to withdraw.
Print list of accounts:
Name: Tom Balance: 5000$
Name: Dick  Balance: 9000$
Name: Harry Balance: 30000$
Close Dick's account.
Dick's account closed. He took 9000$.
Merge Tom and Harry's accounts.
Print list of accounts:
Name: Tom and Harry Balance: 35000$
```

Indications complémentaires

Vous devez soumettre un fichier compressé `.zip` qui contient un dossier. Ce dossier doit s'appeler `<nom>_<prenom>` (pas de majuscules ni de caractères accentués), le fichier zip doit porter le même nom. Par exemple, Alan Turing doit soumettre un fichier `turing_alan.zip` qui contient le dossier `turing_alan` avec son projet.

Les fichiers à soumettre :

— `Bank.hpp` qui contient le code C++ du ADT ;
— `Test.cpp` qui contient une classe qui fait le test de l'ADT. Pour cela vous pouvez prendre l'inspiration de l'exemple dans cet énoncé ;
— `Makefile` qui produit un fichier exécutable `BankTest`.

Consignes pour la remise du projet

À respecter scrupuleusement !

1. Votre projet doit comporter **votre nom** et **votre section**.
2. Votre projet doit être **dactylographié**. Les projets écrits à la main ne seront **pas corrigés**.
3. Votre code doit être **commenté**.
4. Vous devez respecter les modalités suivantes :
   — Rendre une copie papier du projet au secrétariat étudiant.
   — Poster votre fichier avec le projet sur l'UV.
   — Date : **Le mardi 17 mai 2016**
   — Heure : **Avant 10 heures sur l'UV** et **entre 10 heures et 12 heures au secrétariat**
   Après ces heures de remise, les projets sont considérés comme en retard et **ne seront plus acceptés**.