Programmation Objet : introduction

Soutien algo/prog AS 2009

6 novembre 2009

1 Exemple

Voici un programme source contenant une définition de classe `Compte` (compte bancaire) et quelques tests de bon fonctionnement.

```
#include <iostream>
#include <cassert>

using namespace std;

// ------------------------------------------------------

class Compte
  {
  private:
    int    _numero;
    string _client;
    float  _solde;

  public:
    Compte(int numero, string client, float solde)
    {
      _numero = numero;
      _client = client;
      _solde  = solde;
    }

    float solde()
    {
      return _solde;
    }

    void crediter(float montant)
    {
      _solde += montant;
    }
  };

// ------------------------------------------------------

int main()
{
  cout << "** Test Constructeur" << endl;
```

```
  Compte a1 (1,"Albert", 1200.00);
  assert( a1.solde() == 1200.00);

  a1.crediter(34.00);
  assert( a1.solde() == 1234.00);

  cout << "-- ok" << endl << endl;
}
```

La classe `Compte` contient des entités qui ont trois champs (ce qu'on appelle des *données membres*) : un numéro de compte (`_numero`),[1] un nom de client, et un solde.

Quelques opérations sont définies sur les comptes :

- un *constructeur*, qui initialise le compte
- une fonction qui renvoie la valeur du solde
- une fonction qui augmente la valeur du solde.

ces opérations sont appelées les *méthodes de la classe*.

1.1 Utilisation de la classe

- A la ligne
  `Compte a1(1,"Albert", 1200.00);`
  la fonction `main()` déclare une variable `a1` en faisant un appel au constructeur défini dans la classe, ce qui crée un objet et initialise ses 3 champs.
- Elle fait ensuite un appel à la méthode `solde()`. Appliquée à l'objet `a1`, elle renvoie son solde, qui, normalement, est identique à la valeur donnée au moment de la création.
- l'instruction " `a1.crediter(34.00);`" appelle une méthode qui modifie le solde.

1.2 Données/Méthodes publiques/privées

Ici toutes les données membres sont déclarées "private", et les méthodes "public".

Ceci indique ce qui est visible de l'extérieur, par exemple dans le `main()` on peut écrire

```
cout << "il reste " << a1.solde() << " pour finir le mois";
```

mais pas

```
cout << "il reste " << a1._solde << " pour finir le mois";
```

C'est tout à fait volontaire. On veut forcer les programmeurs qui utilisent la classe `Compte` à passer par les méthodes indiquées, et les empêcher d'accèder directement aux données membres.

2 Découpage

Ci-dessus on a présenté un source combinant la définition de la classe et un programma principal qui l'utilise. C'est acceptable pour faire quelques essais, mais pour un programme plus sérieux on va découper ce source en (au moins) 3 morceaux

- les entêtes de la classe `Compte`
- le code de la classe `Compte`
- le programme principal

Voici le fichier `Compte.h` où figurent les données membres et les prototypes des méthodes :

[1] Il existe diverses conventions pour nommer les données membres. Ici nous utilisons le préfixe "_", d'autres écriraient "my_numero", "m_numero" ou encore "mNumero".

```
// fichier Compte.h

#ifndef COMPTE_H
#define COMPTE_H

#include <iostream>
using namespace std;

class Compte
  {
  private:
    int    _numero;
    string _client;
    float  _solde;

  public:
    Compte(int numero, string client, float solde);
    float solde();
    void crediter(float montant);
  };
#endif
```

Le source `Compte.cc` contient le code des méthodes. Le préfixe "`Compte : :`" indique la classe à laquelle se rattachent les méthodes.

```
// Fichier Compte.cc

#include "Compte.h"

Compte::Compte(int numero, string client, float solde)
{
  _numero = numero;
  _client = client;
  _solde  = solde;
}

float Compte::solde()
{
  return _solde;
}

void Compte::crediter(float montant)
{
  _solde += montant;
}
```

Le programme principal se contente d'inclure le fichier des entêtes.

```
// fichier exemple.cc

#include <iostream>
#include <cassert>

using namespace std;
```

```
#include "Compte.h"

int main()
{
  cout << "** Test Constructeur" << endl;

  Compte a1 (1,"Albert", 1200.00);
  assert( a1.solde() == 1200.00);

  a1.crediter(34.00);
  assert( a1.solde() == 1234.00);

  cout << "-- ok" << endl << endl;
}
```

Un Makefile simple ne peut que faciliter le développement

```
# Makefile

exemple: exemple.o Compte.o
	g++ exemple.o Compte.o -o exemple

exemple.o: exemple.cc Compte.h
	g++ -c exemple.cc

Compte.o: Compte.cc Compte.h
	g++ -c Compte.cc

# ------------------------------------

pretty:
	astyle --style=gnu *.cc *.h

clean:
	rm -f *~ *.o *.orig

mrproper: clean
	rm -f exemple
```

3 Exercices

1. Déclarez une méthode `debiter` qui retranchera une certaine somme d'un compte. Ajoutez une séquence de test pour vérifier son fonctionnement.
2. Déclarez une méthode `transferer` à deux paramètres qui fera passer certaine somme (donnée en premier paramètre) du compte vers un autre compte (second paramètre). Ajoutez une séquence de test.