Leçon 7 : La Programmation Orientée Objet 2 en C++

Université Virtuelle de Côte d'Ivoire

Table des matières

1- L'héritage et le Polymorphisme

I

Définition : 1.1- L'héritage

L'héritage permet de former une nouvelle classe à partir de classes existantes :

- *La nouvelle classe (classe fille, sous-classe)* hérite des attributs et des méthodes *des classes à partir desquelles elle a été formée (classes mères, super-classes).*
- *De nouveaux attributs et de nouvelles méthodes* peuvent être définis dans *la classe fille.*
- *Des méthodes des classes mères* peuvent être *redéfinies dans la classe fille.*

Il y a héritage quand on peut dire : « A *est un* B » on peut avoir comme exemple :

1. une voiture est un véhicule (Voiture hérite de Véhicule) ;
2. un bus est un véhicule (Bus hérite de Véhicule) ;
3. un moineau est un oiseau (Moineau hérite d'Oiseau) ;
4. un corbeau est un oiseau (Corbeau hérite d'Oiseau) ;
5. un chirurgien est un médecin (Chirurgien hérite de médecin) ;

Syntaxe : 1.2-Syntaxe héritage

- *Cas 1 : Héritage simple*

```
class NomClasseFille: public NomClasseMere {
// Déclaration des nouveaux attributs
// Déclaration des nouvelles méthodes
// Redéfinition de méthodes
...
};
```

On parle d'héritage multiple quand une classe fille a une classe mère.

- *Cas 2 : Héritage multiple*

```
class NomClasseFille: public NomClasseMere1, public NomClasseMere2 ... {
// Déclaration des nouveaux attributs
// Déclaration des nouvelles méthodes
```

// Redéfinition de méthodes

...

};

On parle d'héritage multiple quand une classe fille a plusieurs classes mères.

Remarque : en C++, une classe peut être dérivée à partir de plusieurs classes de bases (héritage multiple)

Exemple :

```
class Personne
{
private:
string nom,prenom;
public:
void saisie(){
cout<<"Donner le nom :"<<endl;
cin>>nom;
cout<<"Donner le prénom :"<<endl;
cin>>prenom;};
void affichage()
{
cout<<"Le nom :"<<nom<<endl;
cout << "Le prénom :" <<prenom<< endl ;
};
};
class etudiant : public Personne    // classe étudiant hérite de la classe Personne
{
private:
 double notes[5];
public:
float moyenne();
};
Personne unePersonne;
Etudiant unEtudiant;
unePersonne.saisie(); // légal : unePersonne est une Personne
unEtudiant.moyenne(); // légal : unEtudiant est un Etudiant
unEtudiant.affichage(); // légal : unEtudiant est une Personne (par héritage)
unePersonne.moyenne(); // illégal : unePersonne n'est pas un Etudiant

```

1.3- Le Polymorphisme

Le nom de polymorphisme vient du grec et signifie qui peut prendre plusieurs formes. Cette caractéristique est un des concepts essentiels de la programmation orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), *le polymorphisme est relatif aux méthodes des objets.*

Le *polymorphisme* est un moyen de manipuler des objets hétéroclites de la même manière, pourvu qu'ils disposent d'une interface commune.

Un *objet polymorphe* est un objet susceptible de prendre plusieurs formes pendant l'exécution.

Le *polymorphisme* représente la capacité du système à choisir dynamiquement la méthode qui correspond au type de l'objet en cours de manipulation.

Le *polymorphisme* est implémenté en C++ avec l*es fonctions virtuelles (virtual) et l'héritage*.

Remarque :

- En C++, on doit d'ajouter le mot-clé *virtual* (devant une méthode) parce que, par défaut, les fonctions membres ne sont pas liées dynamiquement. Les fonctions virtuelles permettent d'exprimer des différences de comportement entre des classes de la même famille. Ces différences sont ce qui engendre un comportement polymorphe.

Exemple :

```
#include <iostream>
using namespace std;
//*************** Création de la classe Forme ******************
class Forme {
public:
//******* Constructeur de la classe forme
Forme()
{
    cout << "constructeur de  Forme "<<endl;
 }
//********la méthode dessiner sera virtuelle et fournira un comportement polymorphe
virtual void dessiner()
{
    cout << "je dessine une forme ?"<<endl;
}
};
//*************** classe Cercle hérite de la classe Forme ******************
class Cercle : public Forme {
public:
//******* Constructeur de la classe Cercle
Cercle()
{
    cout << "Constructeur de Cercle"<<endl;
}
//***** Methode dessiner de la classe cercle
void dessiner()
{
    cout << "je dessine un Cercle !"<<endl;
}
};
//*************** classe Triangle hérite de la classe Forme ******************
class Triangle : public Forme {
public:
//******* Constructeur de la classe Triangle
Triangle()
{
    cout << "Constructeur de Triangle"<<endl;
 }
//***** Methode dessiner de la classe cercle
void dessiner()
{
```

```
42      cout << "je dessine un Triangle !"<<endl;
43  }
44  };
45   // ***Procédure pour afficher une forme, on utilise  le passage par
    réference
46  void faireQuelqueChose(Forme& f)
47  {
48  f.dessiner(); // dessine une Forme
49  }
50  main()
51  {
52  Cercle c;
53  Triangle t;
54  faireQuelqueChose(t); // desine le  cercle
55  faireQuelqueChose(t); // desine le triangle
56  }
57
```

Exercice

II

[solution n°1 p.13]

Soit un programme en C++, veillez renseigner les cases vides.

using namespace std;

Personne

{

nom,prenom;

public:

void saisie(){

"Donner le nom :"

"Donner le prénom :"

};

void affichage()

{

"Le nom :"<< <<endl;

"Le prénom :" << << endl ;

};

};

Employe :

{

double sal;

public:

```
float salaire(){
};
};
main(){
// Créer unSal de type Employe

// Faite appel à la méthode saisie

// Faite appel à la méthode salaire

// Faite appel à la méthode affichage

}
```

2- Surcharges et Classe abstraite

2.1- Surcharges sur les classes

En programmation C++ vous pouvez utiliser plusieurs fonctions avec le même nom. La définition de chaque fonction doit être différente par rapport aux autres fonctions qui portent le même nom, cette différence peut être au nouveau du type de la fonction, ou bien du nombre de paramètres qu'ils envoient.

Une sur-définition ou surcharge (overloading) permet d'utiliser plusieurs méthodes qui portent le même nom au sein d'une même classe avec une signature différente.

Exemple :

```
#include <iostream>
#include <string>
using namespace std;
//**************** création de la classe afficherData
class afficherData {
   public:
//************* création de la première methode afficher pour un entier
      void afficher(int i) {
         cout << "Affichage d'un entier: " << i << endl;
      }
//************* création de la première methode afficher pour un réel
      void afficher(double  f) {
         cout << "Affichage d'un réel: " << f << endl;
      }
//************* création de la première methode afficher pour une chaîne
      void afficher(string c) {
         cout << "Affichage d'une chaîne: " << c << endl;
      }
};

main() {
   afficherData pd;
   // appelle de la fonction afficher pour afficher un entier
   pd.afficher(5);

   // appelle de la fonction afficher pour afficher un réel
   pd.afficher(500.263);

   // appelle de la fonction afficher pour afficher un chaîne
   pd.afficher("Bonjour, vous développez avec C++");
```

```
31 }
32
```

2.2- Classe abstraite

En C++ une classe est dite abstraite si elle contient au moins *une fonction virtuelle pure*. *Une fonction membre est dite virtuelle pure* lorsqu'elle est déclarée de la façon suivante : *virtual type nomMethode (paramètres) = 0;*

NB : Une classe abstraite ne peut pas être instanciée,on ne peut créer d'objet à partir d'une classe abstraite.

Exemple :

```
1 #include <iostream>
2 using namespace std;
3 //*************** Création de la classe Forme ******************
4 class Forme {
5 public:
6 //******* Constructeur de la classe forme
7 Forme()
8 {
9     cout << "constructeur de  Forme "<<endl;
10 }
11 //********la méthode dessiner est virtuelle pure et ne possède aucune
  définition
12 virtual void dessiner() = 0; // cela oblige tous les descendants à
  contenir une méthode dessiner()
13 };
14 //*************** classe Cercle hérite de la classe Forme
  ******************
15 class Cercle : public Forme {
16 public:
17 //******* Constructeur de la classe Cercle
18 Cercle()
19 {
20     cout << "Constructeur de Cercle"<<endl;
21 }
22 //***** Methode dessiner de la classe cercle
23 void dessiner()
24 {
25     cout << "je dessine un Cercle !"<<endl;
26 }
27 };
28 //*************** classe Triangle hérite de la classe Forme
  ******************
29 class Triangle : public Forme {
30 public:
31 //******* Constructeur de la classe Triangle
32 Triangle()
33 {
34     cout << "Constructeur de Triangle"<<endl;
35 }
36 //***** Methode dessiner de la classe cercle
37 void dessiner()
38 {
39     cout << "je dessine un Triangle !"<<endl;
40 }
```

```
41 };
42
```

Exercice

IV

[solution n°2 p.15]

Énoncé :

Soit la classe abstraite personne

Solution :

{

string nom,prenom; double notes[5];

//**********Constructeur par défaut

//**********Constructeur par défaut

(, ,);

saisie()

};

Solutions des exercices

> **Solution** n°1

Exercice p. 7

Soit un programme en C++, veillez renseigner les cases vides.

```
#include<iostream>
#include<string>
using namespace std;
class Personne
{
private:
string nom,prenom;
public:
void saisie(){
cout<<"Donner le nom :"<<endl;
cin>>nom;
cout<<"Donner le prénom :"<<endl;
cin>>prenom;
};
void affichage()
{
cout<<"Le nom :"<<nom<<endl;
cout << "Le prénom :" <<prenom<< endl ;
};
};
class Employe : public Personne
{
private:
double sal;
public:
float salaire(){
};
};
```

main(){

// Créer unSal de type Employe

```
Employe unSal;
```

// Faite appel à la méthode saisie

```
unSal.saisie();
```

// Faite appel à la méthode salaire

```
unSal.salaire();
```

// Faite appel à la méthode affichage

```
unSal.affichage();
```

}

> **Solution** n°2

Exercice p. 12

Énoncé :

Soit la classe abstraite personne

Solution :

```
class personne
```

{

```
private:
```

string nom,prenom; double notes[5];

```
public:
```

//**********Constructeur par défaut

```
etudiant();
```

//**********Constructeur par défaut

```
etudiant(string nom, string prenom,double notes[5]);
```

```
virtual void saisie() =0;
```

};