TD Héritage - POO

Victor Lezaud

23 novembre 2017

Sommaire

1 Objet dynamique et virtual

1.1 Présentation du programme

Le programme contient une classe Personne avec un attribut nom de type string et une classe Etudiant avec un attribut anne de type annee, qui hérite de Personne. Chaque classe doit seulement pouvoir être affichée. Le programme de test est le suivant :

```
#include "Personne.h"
#include "Etudiant.h"

static void afficher(const Personne * pt)
{
    pt->Afficher();
}

int main()
{
    Personne * pp;

    pp = new Personne("Marie");
    afficher(pp);
    delete pp;

    pp = new Etudiant("Mathieu", 4);
    afficher(pp);
    delete pp;

    return 0;
}
```

1.2 Définition des classes

1.2.1 Personne

```
#include <iostream>

class Personne
{
    public:
        Personne(String unNom) : nom(unNom){}
        virtual ~Personne(){}
        // Afficher va dépendre des questions

    private:
        String nom;
}
```

1.2.2 Etudiant

```
#include <iostream>
```

```
class Etudiant : public Personne
{
    public:
        Etudiant(String unNom, int uneAnnee) : Personne(unNom),
                 annee(uneAnnee){}
        virtual ~Etudiant(){}
        // Afficher va dépendre des questions

    private:
        int annee;
}
```

1.3 Questions

1.3.1 virtual

Méthodes classique : Dans cette question on implémente les méthodes Afficher sans le mot-clé virtual dans les deux classes :

- Personne : `void Afficher(){cout<<nom<<endl)`
- Etudiant : `void Afficher(){cout<<annee<<endl)`

Dans ce cas-là le compilateur va créer des une liaison statique vers la méthode `Personne::Afficher()` lors de l'appel `pt->Afficher();` dans la fonction ordinaire `afficher()`.
Ainsi le programme va appeler la méthode de la classe mère dans tous les cas et affichera :

```
Marie
Mathieu
```

virtual dans Personne : Dans cette question on ajoute le mot-clé virtual dans la classe Personne.

- Personne : `virtual void Afficher(){cout<<nom<<endl)`
- Etudiant : `void Afficher(){cout<<annee<<endl)`

Dans ce cas-là, à l'appel de la méthode `Personne::Afficher()`, le compilateur va créer une liaison dynamique car il va trouver le mot-clé `virtual`.
La méthode appelée va donc dépendre de l'exécution du programme. Dans le cas de notre test il va d'abord appeler
`Personne::Aficher()` puis `Etudiant::Afficher()` et affichera :

```
Marie
4
```

virtual dans Etudiant : Dans cette question on met le mot-clé virtual dans la classe Etudiant uniquement.

- Personne : `void Afficher(){cout<<nom<<endl)`
- Etudiant : `virtual void Afficher(){cout<<annee<<endl)`

On revient dans le premier cas car la liaison est faite avec la méthode `Personne::Afficher()` qui n'est ici plus virtuelle.
La méthode appelée sera donc toujours `Personne::Afficher()` et le test affichera :

```
Marie
Mathieu
```

virtual dans les deux : Dans cette question on met le mot-clé virtual dans les deux classes.

- Personne : `virtual void Afficher(){cout<<nom<<endl)`
- Etudiant : `virtual void Afficher(){cout<<annee<<endl)`

On revient au deuxième cas avec la liaison dynamique parce qu'on retrouve le mot-clé virtual devant la méthode
`Personne::Afficher()`.
Le programme de test va donc appeler `Personne::Afficher()` puis
`Etudiant::Afficher()`

```
Marie
4
```

1.3.2 Tout afficher

Problème : Comment afficher l'ensemble des caractéristiques de l'objet dans chaque cas, sans ajouter de nouvelle méthode au classes ?

Résolution : Il faut appeler la méthode de Personne dans la méthode de Etudiant. On redéfinie donc la méthode `Etudiant::Afficher()` de la façon suivante :

```
void Etudiant::Afficher() : Personne::Afficher()
{cout<<annee<<endl;}
```

virtual ? Bien sûr la méthode `Personne::Afficher()` doit encore être virtuelle ! !

2 Construction & Destruction

2.1 Présentation du programme

On cherche à compter le nombre de création et de destruction des instances d'une classe et de ses descendants. On prendra une classe Comptage et un descendant FilsComptage. Les classes possèdent un constructeur par défaut, un constructeur de copie et un destructeur. Chaque objet doit avoir un numéro d'instance afin de faciliter le suivi.
Le programme principal écrit en fin d'exécution la valeur de la différence entre le nombre de création et de destruction, que l'on appellera `nbRestant`.

2.2 Définition des classes

2.2.1 Comptage

Comptage.h

```
class Comptage
{
    public:
        static void AfficheNbRestant;

        Comptage();
        Comptage(const & Comptage) ;
        virtual ~Comptage();

    protected:
        int id;
        static int nbRestant;
        static int nbTotal;
}
```

Comptage.cpp

```
Comptage::nbRestant = 0;
Comptage::nbTotal = 0;

static void Comptage::AfficheNbRestant()
{
    cout<<"nbRestant = "<<nbRestant<<endl;
}

Comptage::Comptage()
{
    ++nbTotal;
    ++nbRestant;
    id = nbTotal;
    cout<<"Creation de l'objet Comptage
           numéro "<<id<<endl;
}

Comptage::Comptage(const & Comptage)
{
    ++nbTotal;
    ++nbRestant;
    id = nbTotal;
    cout<<"Creation par copie de l'objet Comptage
           numéro "<<id<<endl;
}

Comptage::~Comptage()
{
    cout<<"Destruction de l'objet Comptage
           numéro "<<id<<endl;
    --nbRestant;
}
```

2.2.2 FilsDeComptage

FilsDeComptage.h

```
class FilsDeComptage : public Comptage
{
    public:
        FilsDeComptage();
        FilsDeComptage(const & FilsDeComptage) ;
        virtual ~FilsDeComptage();
    protected:
        static nbTotalFils;
        int idFils;

}
```

FilsDeComptage.cpp

```
FilsDeComptage::nbTotalFils = 0;

FilsDeComptage::FilsDeComptage()
                : Comptage()
{
    ++nbTotalFils;
    id = nbTotalFils;
    cout<<"Creation de l'objet FilsDeComptage
           numéro "<<idFils<<endl;
}

FilsDeComptage::FilsDeComptage(const & FilsDeComptage)
                : Comptage(FilsDeComptage)
{
    ++nbTotalFils;
    id = nbTotalFils;
    cout<<"Creation par copie de l'objet FilsDeComptage
           numéro "<<idFils<<endl;
}

FilsDeComptage::~FilsDeComptage()
{
    cout<<"Destruction de l'objet FilsDeComptage
           numéro "<<idFils<<endl;
}
```

2.3 Question

2.3.1 Variables de classe

Comment est géré le nombre restant ? Le nombre restant est stockée dans une variable de classe soit une variable accessible et partagée par l'ensemble des instances de la classe. Elle est donc incrémenté par chaque instance à la construction et décrémentée à la destruction.

Et pour id et idFils ? Ils sont définit à partir d'une variable de classe qui compte les constructions (celle-ci ne doit pas être décrémentée pour assurer l'absence de doublons dans la valeur `id` et `idFils`).

2.3.2 Série de tests

Pour chacun des tests suivant on donnera le resultat de l'exécution sachant que le programme principal appelle la procédure test avant d'afficher le nombre restant.

```
----Test 0
static void test()
{
    Comptage c1;
}
----Resultat :
Création de l'objet Comptage numéro 1
Destruction de l'objet Comptage numéro 1
nbRestant = 0
----Fin Test 0

----Test 1
static void test()
{
    Comptage c1;
    Comptage c2(c1);
}
----Resultat :
Création de l'objet Comptage numéro 1
Création par copie de l'objet Comptage numéro 2
Destruction de l'objet Comptage numéro 2
Destruction de l'objet Comptage numéro 1
nbRestant = 0
----Fin Test 1

----Test 2
static void test()
{
    Comptage c1;
    Comptage c2(c1);
}
----Resultat :
Création de l'objet Comptage numéro 1
Création de l'objet FilsDeComptage numéro 1
Création par copie de l'objet Comptage numéro 2
Création par copie de l'objet FilsDeComptage numéro 2
Destruction de l'objet FilsDeComptage numéro 2
Destruction de l'objet Comptage numéro 2
Destruction de l'objet FilsDeComptage numéro 1
Destruction de l'objet Comptage numéro 1
```

```
nbRestant = 0
----Fin Test 2
```