TP 5 - Polymorphisme

Programmation Orientée Objet

Objectif

– Comprendre le principe du polymorphisme

1 Première fonction polymorphe

1) Utilisez une classe `Personne` simplifiée ne contenant qu'un `nom`, un `prénom` et un constructeur permettant d'affecter les attributs et une méthode d'affichage `affiche()`.

Créez une classe `Etudiant` héritant de Personne et contenant en attribut le numéro d'étudiant. Redéfinissez la fonction `affiche()` afin d'ajouter à l'affichage le numéro d'étudiant et le fait que la personne est un étudiant.

Créez une classe `Enseignant` héritant de `Personne` et contenant en attribut métier de l'enseignant (Professeur, maître de conférence ou vacataire). Redéfinissez la fonction `affiche()` afin d'ajouter à l'affichage le métier de l'enseignant et le fait que la personne est un enseignant.

Créez un programme principal instanciant un enseignant et un étudiant. Puis appeler l'affichage pour les deux. Que remarquez-vous ?

Rep : L'affichage est celui des classes hérités

Créez un pointeur sur Personne et affectez lui tour à tour l'étudiant puis l'enseignant. Appeler via le pointeur la méthode affiche() après chaque affectation. Que remarquez-vous ?

Rep : Cela est possible car on peut affecter à un pointeur l'adresse d'un élément de la classe dérivée. L'affichage est celui de la classe mère (Personne).

Que faire pour y remédier ?

Rep : Déclarez l'affichage de Personne en tant que fonction virtuelle.

Faites le.

2) Vérifier le fonctionnement de l'affichage pour un tableau de pointeur sur `Personne` auquel on prendra soin d'affecter alternativement des enseignants, des étudiants et des personnes. Pour cela vous implémenterez pour chaque classe un constructeur par recopie.

```
#ifndef PERSONNE_H
#define PERSONNE_H
#include<iostream>
#include<string>

using namespace std;
////////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe Personne
////////////////////////////////////////////////////////////////////////////////////////////////

const int TAILLE=20;

//class DRH;

class Personne{
        string sNom;
        string sPrenom;

        public:
                // Constructeur et destructeur
                Personne(const string &, const string &);
                Personne (const Personne & p);
                Personne(){};
                ~Personne();

                //Accesseurs
                string getNom() const;
```

```
                void setNom(const string &);
                string getPrenom() const;
                void setPrenom(const string &);

                // fonctions utilitaires
                virtual void affiche(){
                        cout << "Personne : " << getNom() << " " << getPrenom() << endl;
                }
};


#endif
```

```
#include "Personne.h"

// ****************************************
// Constructeur
// ****************************************

Personne::Personne(const string & sBuffer1, const string & sBuffer2){
        // Saisie du nom et du prénom
        setNom(sBuffer1);
        setPrenom(sBuffer2);
}
Personne::Personne(const Personne &p){
        this->setNom(p.sNom);
        this->setPrenom(p.sPrenom);
}


// ****************************************
// Destructeur
// ****************************************
Personne::~Personne(){
}

// ****************************************
// Accesseurs
// ****************************************
string Personne::getNom() const{
        return(sNom);
}
void Personne::setNom(const string &Nom){
        sNom= Nom;
}
string Personne::getPrenom() const{
        return(sPrenom);
}
void Personne::setPrenom(const string & Prenom){
        sPrenom= Prenom;
}

// ****************************************
// Autres fonctions membres
// ****************************************
```

```
#ifndef ETUDIANT_H
#define ETUDIANT_H
#include<iostream>
#include<cstring>
#include "Personne.h"
using namespace std;
////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe Etudiant
////////////////////////////////////////////////////////////////////////////////////////////

//class DRH;

class Etudiant : public Personne{
        int iNumEtud;

        public:
                // Constructeur et destructeur
                Etudiant(const string & n, const string & p, int i=0):Personne(n,p){
                        iNumEtud=i;
                };
                Etudiant(const Etudiant &e):Personne(e){
                        setNumEtud(e.getNumEtud());
                }

                ~Etudiant(){};
```

```
                //Accesseurs
                int getNumEtud() const{
                        return iNumEtud;
                };
                void setNumEtud(const int i){
                        iNumEtud=i;
                };

                // fonctions utilitaires
                void affiche(){
                                Personne::affiche();
                                cout << "Je suis étudiant et mon numéro est : "<< iNumEtud << endl;
                }
};


#endif

#include "Enseignant.h"

#include "Personne.h"
#include "Etudiant.h"
#include "Enseignant.h"

const int TAILLE_TAB=5;
int main()
{
        // Buffer de saisie
        string sBuffer1;
        string sBuffer2;

        // Saisie du nom et prenom
        cout << "Entrez le nom de la personne : ";
        cin >> sBuffer1;

        cout << "Entrez le prenom de la personne : ";
        cin >> sBuffer2;

int i=1234;

        Enseignant ens(sBuffer1, sBuffer2, "MCF");
        Etudiant etud(sBuffer1, sBuffer2, i);

        ens.affiche();
        etud.affiche();

// Un pointeur sur personne peut récupérer l'adresse d'un décendant
        Personne *p;

// Mais dans ce cas, la fonction affiche appelée est celle de Personne
        p=&ens;
        p->affiche();

        p=&etud;
        p->affiche();

// Dans ce cas, pour appeler la bonne fonction affiche(), il faut déclarer Personne::affiche()
// comme une fonction virtuelle

        Personne * tab[TAILLE_TAB];
        Personne pers;
        char continuer='o';
        int choix=0;
        int iNbPers=0;
while ((continuer != 'n') && (iNbPers<TAILLE_TAB))
{
                // Saisie du nom et prenom
        cout << "Entrez le nom de la personne : ";
        cin >> sBuffer1;
        cout << "Entrez le prenom de la personne : ";
        cin >> sBuffer2;
        cout << "Est-ce un enseignant (1), un étudiant (2) ou une personne (3): ";
        cin >> choix;

        if (choix==1)
        {
                //Enseignant

                ens.setNom(sBuffer1);
                ens.setPrenom(sBuffer2);
```

```
                ens.setMetier("Professeur");

                //creation par recopie
                //tab[iNbPers]=new Enseignant(ens);
                tab[iNbPers++]=new Enseignant(sBuffer1,sBuffer2, "Professeur"); //possible aussi sans le const
        }
        else if (choix==2)
        {
                //Etudiant
                etud.setNom(sBuffer1);
                etud.setPrenom(sBuffer2);
                etud.setNumEtud(i);

                //creation par recopie
                tab[iNbPers++]=new Etudiant(etud);
        }
        else{
                //Personne
                pers.setNom(sBuffer1);
                pers.setPrenom(sBuffer2);

                //creation par recopie
                tab[iNbPers++]=new Personne(pers);
        }
        if (iNbPers<TAILLE_TAB)
        {

                cout << "Voulez vous ajouter une autre personne (o/n)?"<<endl;
                cin >> continuer;
        }
}
//affichage
for (i=0; i<iNbPers; i++)
{
        tab[i]->affiche();
}

}
```

```
all: main

main: main.cpp Enseignant.o Etudiant.o Personne.o Personne.h
        g++ Personne.o Enseignant.o Etudiant.o main.cpp -o main
Personne.o: Personne.cpp Personne.h
        g++ -c Personne.cpp
Etudiant.o : Etudiant.cpp Personne.h
        g++ -c Etudiant.cpp
Enseignant.o: Enseignant.cpp Personne.h
        g++ -c Enseignant.cpp

clean:
        rm *.o main
```

2 Constructeur et polymorphisme

On vous demande de créer la classe `T1`, dont le constructeur appelle la méthode membre virtuelle `afficher()`. Celle-ci affiche le message "`Classe T1`". Cette classe disposera d'un destructeur affichant le message "`On détruit T1`" puis appellant elle-aussi la méthode virtuelle `afficher()`.

Créer une classe T2, héritant de T1, dont le constructeur appelle la méthode membre `afficher()` de la classe `T2`. Celle-ci affiche le message "`Classe T2`". Cette classe disposera d'un destructeur affichant le message "`On détruit T2`" puis appellant elle-aussi la méthode virtuelle `afficher()` de `T2`.

```
#include <iostream>
using namespace std;
class T1
{
public:
        virtual void afficher(void){
                cout << "Classe T1" << endl;
        }
        T1(void){
                afficher();
        }
        ~T1(){
                cout << "On detruit T1" << endl << "Affichage ";
```

```
                afficher();
        }
};

class T2 : public T1
{
public:
        void afficher(void){
                cout << "Classe T2" << endl;
        }
        T2(void) : T1(){};
        ~T2(){
                cout << "On detruit T2" << endl << "Affichage ";
                afficher();
        }
};

int main ()
{
        cout << "Avant la construction d'un objet de classe T1" << endl;
        T1 t1;
        cout << "Avant la construction d'un objet de classe T2" << endl;
        T2 t2;
        return 0;
}
```

Résultat de l'exécution :

```
Avant la construction d'un objet de classe T1
Classe T1
Avant la construction d'un objet de classe T2
Classe T1
On detruit T2
Affichage Classe T2
On detruit T1
Affichage Classe T1
On detruit T1
Affichage Classe T1
```

On remarque que le constructeur appelle toujours la méthode `T1::afficher()`. Il n'applique donc pas le polymorphisme, alors que le destructeur le fait sans problème. En effet, lors de la construction d'un objet, la table des méthodes virtuelles n'est pas encore affectée à l'objet et les différentes méthodes appelées le sont toujours par des liaisons statiques.

3 Destructeur virtuels

Reprendre l'exercice 1 et vérifier la nécessité d'utiliser un destructeur virtuel pour le polymorphisme.

```
#ifndef PERSONNE_H
#define PERSONNE_H
#include<iostream>
#include<string>

using namespace std;
////////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe Personne
////////////////////////////////////////////////////////////////////////////////////////////////

//class DRH;

class Personne{
        string sNom;
        string sPrenom;

        public:
                // Constructeur et destructeur
                Personne(const string &, const string &);
                Personne (const Personne & p);
                Personne(){};
                virtual ~Personne();

                //Accesseurs
                string getNom() const;
                void setNom(const string &);
                string getPrenom() const;
                void setPrenom(const string &);
```

```
                // fonctions utilitaires
                virtual void affiche(){
                        cout << "Personne : " << getNom() << " " << getPrenom() << endl;
                }
};


#endif
```

```
#include "PersonneExo3.h"

// ****************************************
// Constructeur
// ****************************************

Personne::Personne(const string &sBuffer1, const string & sBuffer2){
        // Saisie du nom et du prénom
        setNom(sBuffer1);
        setPrenom(sBuffer2);
}
Personne::Personne(const Personne &p){
        this->setNom(p.sNom);
        this->setPrenom(p.sPrenom);
}


// ****************************************
// Destructeur
// ****************************************
Personne::~Personne(){
        cout << "   ---Destructeur Personne\n";
}

// ****************************************
// Accesseurs
// ****************************************
string Personne::getNom() const{
        return(sNom);
}
void Personne::setNom(const string & Nom){
        sNom = Nom;
}
string Personne::getPrenom() const{
        return(sPrenom);
}
void Personne::setPrenom(const string & Prenom){
        sPrenom = Prenom;
}

// ****************************************
// Autres fonctions membres
// ****************************************
```

```
#ifndef ETUDIANT_H
#define ETUDIANT_H
#include<iostream>
#include<string>
#include "PersonneExo3.h"
using namespace std;
////////////////////////////////////////////////////////////////////////////////////////////////////
// Definition de la classe Etudiant
////////////////////////////////////////////////////////////////////////////////////////////////////

//class DRH;

class Etudiant : public Personne{
        int iNumEtud;

        public:
                // Constructeur et destructeur
                Etudiant(const string &n, const string & p, int i=0):Personne(n,p), iNumEtud(i){
                };
                Etudiant(const Etudiant &e):Personne(e){
                        setNumEtud(e.getNumEtud());
                }

                ~Etudiant(){
                        cout << "   ---Destructeur Etudiant\n";
                };

                //Accesseurs
```

```
                int getNumEtud() const{
                        return iNumEtud;
                };
                void setNumEtud(const int i){
                        iNumEtud=i;
                };

                // fonctions utilitaires
                void affiche(){
                                Personne::affiche();
                                cout << "Je suis étudiant et mon numéro est : "<< iNumEtud << endl;
                }
};


#endif

#include "EnseignantExo3.h"

#include "PersonneExo3.h"
#include "EtudiantExo3.h"
#include "EnseignantExo3.h"

const int TAILLE_TAB=5;
int main()
{
        // Buffer de saisie
        string sBuffer1;
        string sBuffer2;
        int i;
        Personne * tab[TAILLE_TAB];
        Personne pers;
        char continuer='o';
        int choix=0;
        int iNbPers=0;
while ((continuer != 'n') && (iNbPers<TAILLE_TAB))
{
                // Saisie du nom et prenom
        cout << "Entrez le nom de la personne : ";
        cin >> sBuffer1;
        cout << "Entrez le prenom de la personne : ";
        cin >> sBuffer2;
        cout << "Est-ce un enseignant (1), un étudiant (2) ou une personne (3): ";
        cin >> choix;

        if (choix==1)
        {
                //Enseignant

                tab[iNbPers++]=new Enseignant(sBuffer1,sBuffer2, "Professeur"); //possible aussi sans le const
        }
        else if (choix==2)
        {
                //Etudiant
                tab[iNbPers++]=new Etudiant(sBuffer1,  sBuffer2, 1234);
        }
        else{
                //Personne
                tab[iNbPers++]=new Personne(sBuffer1, sBuffer2);
        }
        if (iNbPers<TAILLE_TAB)
        {

                cout << "Voulez vous ajouter une autre personne (o/n)?"<<endl;
                cin >> continuer;
        }
}
//affichage
for (i=0; i<iNbPers; i++)
{
        tab[i]->affiche();
}

//destruction
for (i=0; i<iNbPers; i++)
{
        delete tab[i];
}

}
```

```
all: mainExo3

mainExo3: mainExo3.cpp EnseignantExo3.o EtudiantExo3.o PersonneExo3.o PersonneExo3.h
	g++ PersonneExo3.o EnseignantExo3.o EtudiantExo3.o mainExo3.cpp -o mainExo3
PersonneExo3.o: PersonneExo3.cpp PersonneExo3.h
	g++ -c PersonneExo3.cpp
EtudiantExo3.o : EtudiantExo3.cpp PersonneExo3.h
	g++ -c EtudiantExo3.cpp
EnseignantExo3.o: EnseignantExo3.cpp PersonneExo3.h
	g++ -c EnseignantExo3.cpp

clean:
	rm *.o mainExo3
```

Remarque : Si le destructeur de Personne n'est pas virtuel, la destruction des objets du tableau de pointeurs sur Personne implique l'appel unique du destructeur de Personne. S'il est virtuel, le destructeur d'étudiants et de enseignant sera appelé.