Université Ibn Tofail
جامعة ابن طفيل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵟⴼⵢⵍ
Ibn Tofail University

POO sous C++
Le Polymorphisme

Med. AMNAI

Filière SMI - S5

Département d'Informatique

5 janvier 2021

Plan

1. Redéfinition des fonctions

Plan

1. Redéfinition des fonctions
2. Fonction Virtuelle

Plan

1. Redéfinition des fonctions
2. Fonction Virtuelle
3. Classe Absraite

Plan

1. Redéfinition des fonctions
2. Fonction Virtuelle
3. Classe Absraite
4. Interfaces

Plan

1. Redéfinition des fonctions
2. Fonction Virtuelle
3. Classe Absraite
4. Interfaces
5. Polymorphisme

Principe

(redif0.cpp)

```
class A {

    private :   int i;
    protected : int j;
    public :    int k;

    public :
    A (int i, int j, int k) {
        this->i = i;
        this->j = j;
        this->k = k;
    }
    void afficher () {
        cout << i << " " << j << " " << k << endl;
    }
};
```

```
int main () {

    A a (2, 4, 5) ;

    B b (6, 4, 3, 2);

    a.afficher();
    b.afficher();

}
```

```
class B : public A {

    int l;

    public :
        B (int i, int j, int k, int l) : A(i, j, k) {
            this->l = l;
        }
        void afficher () {
            cout << l << endl;
        }
};
```

```
void afficher () {
    A::afficher();
    cout << " " << l;
}
```

```
2 4 5
6 4 3 2
```

```
2 4 5
2
```

Exemple

```
class materiel{  //RedifFct.cpp
    protected :
    char ref[20];
    char marque[20];
    public :
        materiel(char *r, char *m){
            strcpy(ref,r);
            strcpy(marque,m);
        }
        void affiche(){
            cout<<"Reference : "<<ref<<"\tMarque : "<<marque<<endl;
        }
};
//-------------------------------------------------------------
class micro:public materiel{
    char processeur[20];
    int disque;
    public :
        micro(char *r,char *m,char *p,int d):materiel(r,m){
            strcpy(processeur,p);
            disque=d ;
        }
    void affiche(){
        cout<<"Reference : "<<ref<<"\tMarque : "<<marque
        <<"\tProcesseur : "<<processeur
        <<"\tDisque : "<<disque<<endl;
    }
};
```

```
main (){
    materiel *p;
    p=new materiel((char *)"HP5010", (char *)"HP");
    p->affiche();
    p=new micro((char *)"IBM7", (char *)"IBM", (char *)"P4", 40);
    p->affiche();

    delete p;
}
```

Discussion

- Le pointeur '**p**' est utilisé pour stocker l'adresse d'un objet de la classe '**materiel**', appeler la fonction '**affiche**' et détruire l'objet créé.
- Le pointeur '**p**' est ensuite utilisé pour réaliser les mêmes opérations sur un objet de la classe '**micro**'.
- Dans le cas des deux objets créés, c'est la méthode '**affiche**' de la classe '**matériel**' qui a été utilisée. Le compilateur a donc tenu compte du type de **pointeur** (**matériel***) et non du type d'**objet** pointé par '**p**'.

Discussion (suite)

- L'idéal serait donc que le compilateur appelle **affiche** de **micro** quand **p** pointe sur **micro** et **affiche** de **matériel** quand **p** pointe sur **matériel**.
- Pour obtenir ce résultat, il suffit d'indiquer au compilateur que la fonction **affiche** est une fonction **polymorphe**, c.à.d une fonction **virtuelle**.

Définition

- Une fonction **virtuelle** est une fonction membre **déclarée** dans la classe de base et **redéfinie** (Overriden) par la classe dérivée.
- Les fonctions virtuelles garantissent que la **fonction correcte est appelée pour un objet** (voir le polymorphisme).
- Les **fonctions virtuelles** sont déclarées avec un mot clé **virtual** dans la classe de base.

```
virtual void afficher() {

}
```

Règles d'utilisation

- Une fonction virtuelle **public**.
- Une fonction virtuelle **Ne peut pas être statique** et ne peut pas être une fonction **amie** d'une autre classe.
- Le prototype d'une fonction virtuelle doit être **identique** dans la **classe de base** et dans la classe **dérivée**.
- La fonction virtuelle est toujours **définie dans la classe de base et redéfinie dans la classe dérivée.** Il n'est pas obligatoire que la classe dérivée écrase (ou redéfinisse la fonction virtuelle).
- Une classe **peut** avoir un **destructeur virtuel**, mais elle **ne peut pas avoir de constructeur virtuel**.

Fonction virtuelle pure

- Une fonction **virtuelle pure** est déclarée en affectant **0** à la déclaration..
- Une fonction **virtuelle pure** (ou **fonction abstraite**) est une fonction virtuelle, mais sans implémentation.

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

Définition

Une **classe** est considérée **abstraite** si elle contient **au moins une fonction virtuelle pure (Abstraite)**.

```
class A{   //ClsAbst.cpp

    public :

        virtual void afficher()=0;
};

main (){
    A *p ; //Ceci est correcte
    A a ;  //Ceci est correcte
}
```

Compile Log | Debug | Find Results | Close

:0202021_CoursPolyTDs\Demo...	In function 'int main()':
)202021_CoursPolyTDs\Demos\D...	[Error] cannot declare variable 'a' to be of abstract type 'A'
)202021_CoursPolyTDs\Demos\D...	[Note] because the following virtual functions are pure within 'A':
)202021_CoursPolyTDs\Demos\D...	[Note] virtual void A::afficher()

Remarques

- On peut avoir un pointeur ou une référence de type classe Abstraite mais **pas un object de type classe Abstraite**.
- **Si on ne redéfinit pas la fonction virtuelle** pure dans la classe dérivée, cette classe est considérée elle aussi **abstraite**.
- Une classe abstraite peut **avoir des constructeurs**.
- Une classe qui **n'est pas abstraite** est nommée **classe concrête**.

Définition

- Une **interface** est une classe particulière qui ne définit aucune de ses fonctions. **Toutes les fonctions** doivent être **virtuelles** pures (**Abstraites**).
- Une interface décrit le comportement d'une classe C ++ sans s'engager dans une implémentation particulière de cette classe.

Polymorphisme (1)

- Signifie **avoir plusieurs formes**. En général, le polymorphisme se produit lorsqu'il existe une **hiérarchie** de classes et qu'elles sont liées **par héritage**.
- Signifie qu'un **appel** à une fonction membre entraînera **l'exécution d'une fonction** différente en fonction **du type d'objet qui appelle la fonction** (poly0.cpp).

```
int main () {

    A *p;
    A a;

    p = &a;

    p->afficher(); // Que va-t-elle afficher ? A ou B


    B b;
    p = &b;

    p->afficher(); // Que va-t-elle afficher ? A ou B

}
```

```
class A {

    public:
        void afficher() {
            cout << "Je suis A" << endl;
        }
};
```

```
class B : public A {

    public:
        void afficher() {
            cout << "Je suis B" << endl;
        }
};
```

Polymorphisme (2)

- Si la fonction de la classe de base **n'est pas virtuelle**, le compilateur considère que c'est la **version finale de cette fonction**. C'est appelé Résolution statique. La fonction à appeler est précisée avant l'exécution.
- Pour changer ce comportement et laisser le **choix** de la fonction se fait à l'**exécution**, la **fonction de la classe de base doit être virtuelle**.

```
int main () {

    A *p;
    A a;

    p = &a;

    p->afficher(); // Que va-t-elle afficher ? A ou B


    B b;
    p = &b;

    p->afficher(); // Que va-t-elle afficher ? A ou B

}
```

```
class A {

    public:
        virtual void afficher() {
            cout << "Je suis A" << endl;
        }
};
```

```
class B : public A {

    public:
        void afficher() {
            cout << "Je suis B" << endl;
        }
};
```

Polymorphisme et Surcharge (1) ??

Que va afficher ce programme ?

```
class A{    //polySurCh.cpp

     public :
          virtual void afficher(double d){ // !!!!
               cout << " Je suis A "<< endl;
          }

};

class B : public A{

     public :

          void afficher(int i){  //!!!!
               cout << " Je suis B "<< endl;
          }
};
main (){
     int i = 0 ;

     A *p ;
     B b;

     p = &b;

     p->afficher(i) ; //Que va-t-elle afficher ? A ou B
}
```

Polymorphisme et Surcharge (2) ??

Que va afficher ce programme (polySurCh2.cpp) ?

```
class A{    //polySurCh2.cpp

    public :
        virtual void afficher(double d){
            cout << " Je suis A "<< endl;
        }

};

class B : public A{

    public :
        void afficher(int i){
            cout << " Je suis B int "<< endl;
        }
        void afficher(double d){
            cout << " Je suis B float "<< endl;
        }

};
main (){
    int i = 0 ;

    A *p ;
    B b;

    p = &b;

    p->afficher(i) ; //Que va-t-elle afficher ? A ou B
}
```