Polymorphisme et Classe Abstraite

2020-2021

Polymorphisme (1): c’est quoi?

- Une méthode est dite **polymorphe** si elle dispose de **plusieurs définitions** pour le même nom

- Par exemple: la méthode **marcher()** a différente définition selon la classe ou le type: humain, animal, oiseau, voiture, avion, …

Polymorphisme (1): Type

- Il existe deux type de polymorphisme:

1) par **surcharge** (overloading)
2) par **héritage**

Polymorphisme par surcharge

- Au sein de **la même classe**, on peut avoir **plusieurs méthodes** qui ont le **même nom** mais **différente signatures** et différente définitions

```
class Point{
    //public:
        float x,y;//par défaut
public:
        Point(float x);
        Point (float x0, float y0);
        void Translater (float d);
        void Translater (float d1, float d2);
};
```

Remarque sur l'hértiage (1): Exemple

- Soit le code suivant:

```
class B{
    public:
    int x;
public:
    B();
    ~B();
};
```

```
class A:public B{
public: A(); ~A();
};
```

- Les **objets de A** peuvent remplacer les objets B dans des fonctions qui prend comme paramètre des objets de B

Remarque sur l'hértiage (2): Exemple

```
class B{
    public:
    int x;
public:
    B();
    ~B();
};
```

```
class A:public B{
public: A(); ~A();
};
```

```
void f(B b){
cout<<"objet B ou A"<<endl;
}
```

Remarque sur l'hértiage (3): Exemple

- La fonction f peut prendre comme paramètre effectif des objets de A ou des objets de B.

```
int main() {
    cout << "!!!Hello World!!!2" << endl;
    A a;
    f(a);
    return 0;
}
```

Remarque sur l'hértiage (4): Exemple

- La définition de la fonction f peut utiliser des adresses des objets de B.

```
void f(B &b){
cout<<"objet B ou A"<<endl;
}
int main() {
    cout << "!!!Hello World!!!2" << endl;
    A a;
    f(a);
    return 0;
}
```

Problème??? (1)

- Supposant que nous avons une méthode M définie dans la classe mère B
- La méthode M est aussi redéfinie dans la classe fille A

```
class B{
    public:
    int x;
public:
    B();
    ~B();
    void M(){cout<<"je suis M dans B"<<endl;}
};
```

```
class A:public B{
public: A(); ~A();
        void M(){cout<<"je suis M dans A"<<endl;}
};
```

Problème??? (2)

- Si on modifier le code de la fonction f comme suit:

```
void f(B &b){
cout<<"objet B ou A"<<endl;
b.M();
}
```

- Question: quand appelle **f(a):** est ce qu'on va exécuter la fonction **a.M()** ou plutôt **a.B::M()**?????

Problème??? (3) Réponse

- En effet, l'appel de f(a) va exécuter toujours la méthode a.B::M()

!!!Hello World!!!2
objet B créé
objet A créé
objet B ou A
je suis M dans B
objet A détruit
objet B détruit

Problème??? (4) Réponse

- Pour pouvoir exécuter **a.M()**, il faut permettre le polymorphisme au niveau de la classe **B** et A et ceci en rendant la fonction **B::M()** une **méthode virtuelle**.

```
class B{
    public:
    int x;
public:
    B();
    ~B();
    virtual void M(){cout<<"je suis M dans B"<<endl;}
};
```

Problème??? (5) Réponse

!!!Hello World!!!2

objet B créé

objet A créé

objet B ou A

je suis M dans A

objet A détruit

objet B détruit

Polymorphisme par hértiage (1)

- Une classe mère A dispose d'une méthode M
- Une classe fille B de A héritera la même méthode M
- 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 les fonctions virtuelles (**virtual**) et **l'héritage**.

Polymorphisme par hértiage (2)

- En C++, on doit se souvenir d'ajouter **le mot-clé virtual** (devant une méthode).

- 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**.

Polymorphisme par hértiage (3): Exemple

- Soit le code suivant

```
class Forme {
   public:
      Forme() { cout << "constructeur Forme <- "; }
      // la méthode dessiner sera virtuelle et fournira un comportement polymorphe
      virtual void dessiner() { cout << "je dessine ... une forme ?\n"; }
};
```

```
class Cercle : public Forme {
   public:
      Cercle() { cout << "Cercle\n"; }
      void dessiner() { cout << "je dessine un Cercle !\n"; }
};
```

```
class Triangle : public Forme {
   public:
      Triangle() { cout << "Triangle\n"; }
      void dessiner() { cout << "je dessine un Triangle !\n"; }
};
```

Polymorphisme par hértiage (4): Exemple

```
void faireQuelqueChose(Forme &f)
{
  f.dessiner(); // dessine une Forme
}

int main()
{
   Cercle c;
   Triangle t;

   faireQuelqueChose(c); // avec un cercle
   faireQuelqueChose(t); // avec un triangle

   return 0;
}
```

Classe abstraite (1)

- 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 :

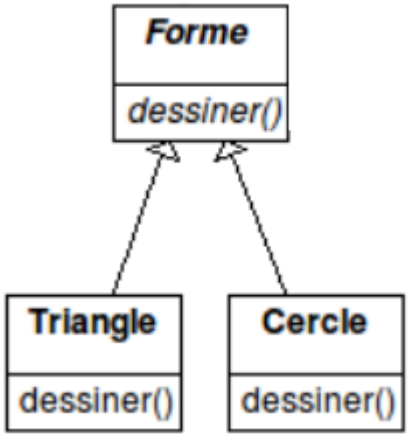
  virtual type nomMethode(paramètres) = 0;

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

- Il est obligatoire d'avoir une **définition pour les fonctions virtuelles pures au niveau des classes dérivées**

Classe abstraite (2)

- Une classe abstraite permet d'introduire certaines fonctions virtuelles **dont on ne peut encore donner aucune définition**.



Classe abstraite (2)

- Dans cet exemple, on **ne sait pas programmer ce que doit faire la fonction dessiner()** dans le contexte de Forme.

- On veut juste s'assurer de sa présence dans toutes les classes filles, sans devoir la définir dans la classe parente.

Classe abstraite (2)

Exemple : classe abstraite

```
// La classe Forme ne peut pas être instanciée : elle est dite abstraite
class Forme {
   public:
      Forme() {}
      // la méthode dessiner est virtuelle pure et ne possède aucune définition
      virtual void dessiner() = 0; // cela oblige tous les descendants à contenir
            une méthode dessiner()
};
```

Classe abstraite (3)

```
// Une classe dans laquelle il n'y a plus une seule fonction virtuelle pure est
    dite concrète et devient instanciable
class Cercle : public Forme {
  public:
    Cercle() {}
    void dessiner() { cout << "je dessine un Cercle !\n"; }
};
```

```
class Triangle : public Forme {
  public:
    Triangle() {}
    void dessiner() { cout << "je dessine un Triangle !\n"; }
};
```

référence

- http://tvaira.free.fr/dev/cours/cours-heritage.pdf