Đa hình - Polymorphism

Bài toán

- Giả sử ta cần quản lý một danh sách các đối tượng có kiểu có thể khác nhau. Khi đó ta cần giải quyết hai vấn đề:
  - Cách lưu trữ
  - Thao tác xử lý
- Ví dụ: Xét trường hợp cụ thể, các đối tượng có thể là người, hoặc là sinh viên hoặc là công nhân.

Đa hình và Hướng đối tượng

- Đa hình: là hiện tượng các đối tượng thuộc các lớp *khác nhau* có khả năng hiểu *cùng* một thông điệp theo các cách *khác nhau*
- Ví dụ: nhận được cùng một thông điệp "nhảy", một con kangaroo và một con cóc nhảy theo hai kiểu khác nhau: chúng cùng có hành vi "nhảy" nhưng các hành vi này có nội dung khác nhau

Phương thức ảo

- Con trỏ thuộc lớp cơ sở (lớp cha) có thể trỏ đến lớp dẫn xuất (lớp con):

  Nguoi pn = new SinhVien("Le Vien Sinh", 123456, 1982);*

- Ta mong muốn thông qua con trỏ thuộc lớp cơ sở có thể truy xuất hàm thành phần được định nghĩa lại ở lớp con:

```
pn->Xuat();  // Mong muon: goi Xuat cua lop sinh vien,
             // Thuc te: goi Xuat cua lop  Nguoi
```

Phương thức ảo

- Phương thức ảo cho phép giải quyết vấn đề. Ta qui định một hàm thành phần là phương thức ảo bằng cách thêm từ khoá **virtual** vào trước khai báo hàm.
- Trong ví dụ trên, ta thêm từ khoá virtual vào trước khai báo của hàm ***xuat*** của lớp Nguoi.

Phương thức ảo

```
class Nguoi {
   protected:
        char *HoTen;
        int NamSinh;
   public:
        Nguoi(char *ht, int ns):NamSinh(ns)
               {HoTen = strdup(ht);}
        ~Nguoi() {delete [] HoTen;}
        void An() const
               { cout << HoTen << " an 3 chen com";}
        void Ngu() const
               { cout << HoTen << " ngu ngay 8 tieng";}
        virtual void Xuat() const {
               cout << "Nguoi, ho ten: " << HoTen
               << " sinh " << NamSinh;
               }
};
```

Phương thức ảo

```
class SinhVien : public Nguoi
{
protected:
   char *MaSo;
public:
   SinhVien(char *n, char *ms, int ns) : Nguoi(n,ns) { MaSo
   = strdup(ms);}
   ~SinhVien() {delete [] MaSo;}
   void Xuat() const {
        cout << "Sinh vien " << HoTen
             << ", ma so " << MaSo;
   }
};
```

```
class NuSinh : public SinhVien
{
public:
  NuSinh(char  *ht,  char  *ms,  int  ns)  :
     SinhVien(ht,ms,ns) {}
  void An() const {
     cout << HoTen << " ma so "
          << MaSo << " an 2 to pho";
  }
};
```

Phương thức ảo

```
class CongNhan : public Nguoi
{
protected:
   double MucLuong;
public:
   CongNhan(char *n, double ml, int ns) :
       Nguoi(n,ns), MucLuong(ml) { }
   void Xuat() const {
       cout << "Cong nhan, ten "
       << HoTen << " muc luong: " << MucLuong;
   }
};
```

```
void XuatDs(int n, Nguoi* an[])
{
  for (int i = 0; i < n; i++)
  {
     an[i]->Xuat();
     cout << "\n";
  }
}
```

Phương thức ảo

```
const int N = 4;
void main()
{
   Nguoi *a[N];
   a[0] = new SinhVien("Vien Van Sinh", "200001234", 1982);
   a[1] = new NuSinh("Le Thi Ha Dong", "200001235", 1984);
   a[2] = new CongNhan("Tran Nhan Cong", 1000000, 1984);
   a[3] = new Nguoi("Nguyen Thanh Nhan", 1960);
   XuatDs(4,a);
}
```

Phương thức ảo ***xuat*** được khai báo ở lớp Nguoi cho phép sử dụng con trỏ đến lớp cơ sở (Nguoi) nhưng trỏ đến một đối tượng thuộc lớp con (Sinh viên, công nhân) gọi đúng thao tác ở lớp con:

Phương thức ảo

```
Nguoi *pn;
pn = new SinhVien("Vien Van Sinh","12345",1982);
pn->Xuat(); // Goi thao tac xuat cua lop Sinh vien
```

- Con trỏ pn thuộc lớp Nguoi nhưng trỏ đến đối tượng sinh viên, vì vậy **pn->Xuat()** thực hiện thao tác xuất của lớp sinh viên.
- Trở lại ví dụ trên, khi a[i] lần lượt trỏ đến các đối tượng thuộc các loại khác nhau, thao tác tương ứng với lớp sẽ được gọi.

```
void XuatDs(int n, Nguoi* an[])
{
        for (int i = 0; i < n; i++)
        {
                an[i]->Xuat();
                cout << "\n";
        }
}
```

Thêm lớp con mới

- Dùng phương thức ảo, ta dễ dàng nâng cấp sửa chữa
  - Việc thêm một loại đối tượng mới rất đơn giản, ta không cần phải sửa đổi thao tác xử lý (hàm XuatDs).
- Qui trình thêm chỉ là xây dựng lớp con kế thừa từ lớp cơ sở hoặc các lớp con đã có và định nghĩa lại phương thức (ảo) ở lớp mới tạo nếu cần

Thêm lớp con mới

```
class CaSi : public Nguoi
{
protected:
  double CatXe;
public:
  CaSi(char *ht, double cx, int ns) :
  Nguoi(ht,ns), CatXe(cx) {}
  void Xuat() const { cout << "Ca si, " << HoTen
  << " co cat xe " << CatXe;}
};
```

Thêm lớp con mới

```
void XuatDs(int n,Nguoi *an[])
{
   for (int i = 0; i < n; i++)
   {
       an[i]->Xuat();
       cout << "\n";
   }
}
```

- Hàm ***XuatDs*** không thay đổi, nhưng nó có thể hoạt động cho các loại đối tượng ca sĩ thuộc lớp mới ra đời.
- Có thể xem như thao tác ***XuatDs*** được viết trước cho các lớp con cháu chưa ra đời.

Lưu ý khi sử dụng phương thức ảo

- Phương thức ảo chỉ hoạt động thông qua con trỏ.
- Muốn một hàm trở thành phương thức ảo có hai cách: Khai báo với từ khoá virtual hoặc hàm tương ứng ở lớp cơ sở đã là phương thức ảo.
- Phương thức ảo chỉ hoạt động nếu các hàm ở lớp cơ sở và lớp con có nghi thức giao tiếp giống hệt nhau.
- Nếu ở lớp con định nghĩa lại phương thức ảo thì sẽ gọi phương thức ở lớp cơ sở (gần nhất có định nghĩa).

Phương thức thuần ảo và lớp cơ sở trừu tượng

- Lớp cơ sở trừu tượng là lớp cơ sở không có đối tượng nào thuộc chính nó. Một đối tượng thuộc lớp cơ sở trừu tượng phải thuộc một trong các lớp con.
- Xét các lớp Circle, Rectangle, Square kế thừa từ lớp Shape
- Trong ví dụ trên, các hàm trong lớp Shape có nội dung nhưng nội dung không có ý nghĩa. Đồng thời ta luôn luôn có thể tạo được đối tượng thuộc lớp Shape, điều này không đúng với tư tưởng của phương pháp luận hướng đối tượng.

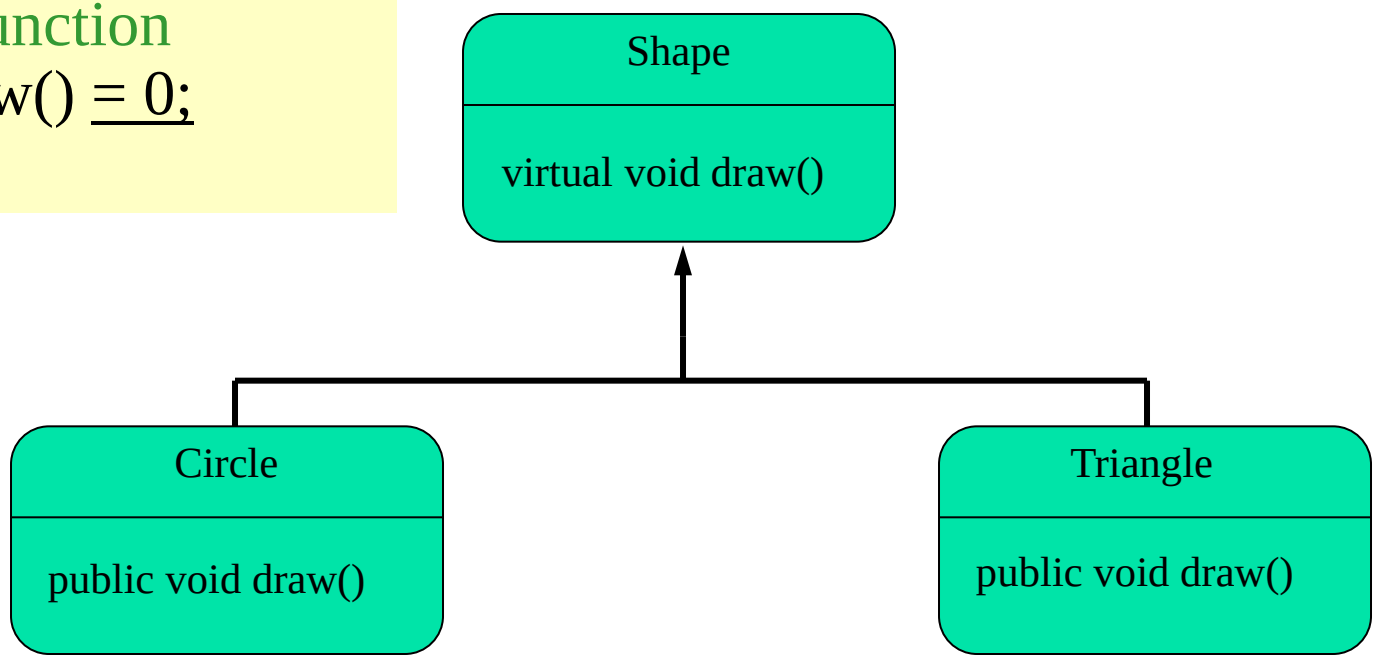
Phương thức thuần ảo và lớp cơ sở trừu tượng

- Ta có thể thay thế cho nội dung không có ý nghĩa bằng phương thức ảo thuần tuý. Phương thức ảo thuần tuý là phương thức ảo không có nội dung.
- Khi lớp có phương thức ảo thuần tuý, lớp trở thành lớp cơ sở trừu tượng. Ta không thể tạo đối tượng thuộc lớp cơ sở thuần tuý.
- Ta có thể định nghĩa phương thức ảo thuần tuý, nhưng chỉ có các đối tượng thuộc lớp con có thể gọi nó.

Phương thức thuần ảo và lớp cơ sở trừu tượng

- Trong ví dụ trên, các hàm thành phần trong lớp Shape là phương thức ảo thuần tuý. Nó bảo đảm không thể tạo được đối tượng thuộc lớp Shape. Ví dụ trên cũng định nghĩa nội dung cho phương thức ảo thuần tuý, nhưng chỉ có các đối tượng thuộc lớp con có thể gọi.
- Phương thức ảo thuần tuý có ý nghĩa cho việc tổ chức sơ đồ phân cấp các lớp, nó đóng vai trò chừa sẵn chỗ trống cho các lớp con điền vào với phiên bản phù hợp.
- Bản thân các lớp con của lớp cơ sở trừu tượng cũng có thể là lớp cơ sở trừu tượng

```
class Shape    //Abstract
{
  public :
  //Pure virtual Function
  virtual void draw() = 0;
}
```



```
class Circle : public Shape { //No draw() - Abstract
    public :
    void print(){
      cout << “I am a circle” << endl;
    }
class Rectangle : public Shape {
    public :
    void draw(){ // Override Shape::draw()
      cout << “Drawing Rectangle” << endl;
    }
```

```
Shape s;
Rectangle r; // Valid
Circle c; // error : variable of an abstract class
```