C5_Đa hình

Ngày học	@November 15, 2022
Property	Bai giang LTHDT - Chuong 5 (Phan 2).pdf
Property 1	

Đa hình: con trỏ thuộc lớp CHA, làm việc với con trỏ, con trỏ mảng thì dùng hàm ẢO

Nếu đề yêu cầu danh sách có chứa nhiều đối tượng: phải có hàm ẢO

1. Tính đa hình:

Tính đa hình là khả năng để cho một thông báo có thể ***thay đổi cách thực hiện của nó theo lớp cụ thể của đối tượng nhận thông báo***. Nghĩa là

- Khi một lớp con được tạo ra, nó có thể **thay đổi cách thực hiện các phương thức nào đó mà nó thừa hưởng từ lớp cha**.
- Nếu một lớp con ***định nghĩa lại một phương thức thừa hưởng từ lớp cha*** thì một thông báo có cùng tên với phương thức này, khi được gởi tới một đối tượng của lớp dẫn xuất sẽ gọi phương thức đã định nghĩa cho lớp dẫn xuất.

Để thực hiện được tính đa hình, C++ dùng cơ chế **liên kết động** (dynamic binding) thay cho cơ chế **liên kết tĩnh** (static binding).

- C++ sử khái niệm phương thức ảo để biểu diễn cho cơ chế liên kết động

a. Liên kết tĩnh:

```
class A{
  public:
    void xuat(){
      cout << "\n Lop A";
    }
};

class B : public A{
  public:
    void xuat(){
      cout << "\n Lop B";
    }
```

```
};

int main(){
  A a;
  B b;
  a.xuat(); //goi ham cua lop A
  b.xuat(); // goi ham cua lop B
  b.A::xuat();//goi ham cua lop A
}
```

Cả 2 lớp này đều có hàm thành phần là xuat(). Lớp B có lớp cơ sở là A và B kế thừa các phương thức của A. Do đó một đối tượng của B sẽ có **2 hàm xuat()**. Xem các câu lệnh sau:

```
B ob; // ob là đối tượng của lớp con B
ob.xuat(); // Gọi tới hàm thành phần xuat() của lớp B
ob.A::xuat() ; // Gọi tới hàm thành phần xuat() của lớp A
```

Các lời gọi hàm thành phần trong ví dụ trên đều xuất phát từ đối tượng ob và mọi lời gọi đều **xác định rõ hàm cần gọi**

Ta xét tiếp tình huống các lời gọi không phải từ một biến đối tượng mà từ một **con trỏ đối tượng**. Xét các câu lệnh:

```
A *p; // p là các con trỏ kiểu A
A a;
B b;
```

Bởi vì con trỏ của lớp cha có thể dùng để chứa địa chỉ các đối tượng của lớp con, nên cả 3 phép gán sau đều hợp lệ:

```
p = &a; //p trỏ vào đối tượng lớp A

// Ta xét lời gọi hàm thành phần từ các con trỏ p:
p->xuat(); //goi ham lop A
p = &b; //p trỏ vào đối tượng lớp B
p->xuat() //vẫn gọi hàm xuat() của lớp A
```

Cả 2 câu lệnh trên đều gọi tới hàm thành phần xuat() của lớp A, bởi vì **con trỏ p có kiểu lớp A**. Sở dĩ như vậy là vì một lời gọi (xuất phát từ một đối tượng hay con trỏ) tới hàm thành phần luôn luôn liên kết với một hàm thành phần cố định và sự liên kết này xác định trong quá trình biên dịch chương trình. Ta bảo đây là sự liên kết tĩnh.

> 1. Nếu **lời gọi xuất phát từ một đối tượng của lớp nào đó, thì phương thức của lớp đó sẽ được gọi.**
> 2. Nếu **lời gọi xuất phát từ một con trỏ kiểu lớp, thì phương thức của lớp đó sẽ được gọi bất kể con trỏ chứa địa chỉ của đối tượng nào.**

=⇒ Vấn đề đặt ra: Ta muốn một lời **gọi phương thức từ con trỏ sẽ phụ thuộc vào lớp mà con trỏ đang trỏ đến đối tượng của lớp đó.**

Ví dụ: khi ta cho p trỏ đến đối tượng của lớp B và thực hiện lời gọi p->xuat(); thì phải gọi đến hàm xuất của lớp B. Điều này sẽ thực hiện được gọi là sự liên kết động và C++ dung phương thức ảo.

b. Phương thức ảo

> Phương thức ảo được **định nghĩa trong lớp cơ sở (cha)** và được **định nghĩa lại trong lớp dẫn xuất (con)**

Định nghĩa phương thức ảo: bắt đầu bằng từ khóa **virtual**.

Khi một lớp có chứa phương thức ảo được kế thừa, lớp **dẫn xuất sẽ định nghĩa lại phương thức ảo đó cho chính mình (**không cần phải sử dụng lại từ khóa virtual)

2. Quy tắc gọi phương thức ảo

Phương thức ảo chỉ khác phương thức thông thường khi được **gọi từ một con trỏ của lớp cơ sở.**

Lời gọi tới phương thức ảo từ một con trỏ sẽ **phụ thuộc vào đối tượng cụ thể mà con trỏ đang trỏ tới**: **con trỏ đang trỏ tới đối tượng của lớp nào thì phương thức**

của lớp đó sẽ được gọi.

```
class A{
  public:
    // * them tu virtual de bien thanh ham ao
    virtual void xuat(){
      cout << "\n Lop A";
    }
};

class B : public A{
  public:
    void xuat(){
      cout << "\n Lop B";
    }
};

int main(){
  A a;
  B b;

  cout << "\nCon tro: ";
  //Con tro p co kieu cua lop A:
  A *p;
  p = &a;
  p->xuat(); // goi ham lop A
  // sau khi them keyword virtual vao ham xuat o lop cha -> se goi ham xuat o lop B
  p = &b;
  p->xuat(); // se goi duoc ham xuat cua lop B
}
```