Bài 17: Hàm ảo - Đa hình runtime

Nguyễn Hoàng Anh

Tính đa hình

- Tính đa hình (Polymorphism) có nghĩa là "nhiều dạng" và nó xảy ra khi chúng ta có nhiều class có liên quan với nhau thông qua tính kế thừa.
- Tính đa hình có thể được chia thành hai loại chính:
  - **Đa hình tại thời điểm biên dịch (**Compile-time Polymorphism**)**.
  - **Đa hình tại thời điểm chạy (**Run-time Polymorphism**)**.

Khái niệm

- Hàm ảo là một hàm thành viên được khai báo trong **class cha** với từ khóa **virtual.**
- Khi một hàm là **virtual**, nó có thể được ghi đè (**override**) trong class con để cung cấp cách triển khai riêng.
- Khi gọi một **hàm ảo** thông qua một con trỏ hoặc tham chiếu đến lớp con, hàm sẽ được quyết định dựa trên **đối tượng thực tế** mà con trỏ hoặc tham chiếu đang trỏ tới chứ không dựa vào kiểu của con trỏ.

Khái niệm

```
class Base{
    public:
        virtual void display(){
            cout << "Display from Base class" << endl;
        }
};
```

Override và tính đa hình run - time

- Override là việc ghi đè hàm ảo ở class con bằng cách định nghĩa lại nó.
- Khi một hàm ảo được ghi đè, hành vi của nó sẽ phụ thuộc vào kiểu của đối tượng thực tế, chứ không phải kiểu của con trỏ hay tham chiếu.
- Tính đa hình run-time xảy ra khi quyết định gọi hàm nào (phiên bản của class cha hay class con) được đưa ra tại thời điểm chạy, không phải lúc biên dịch, giúp mở rộng chức năng. Điều này giúp chương trình linh hoạt hơn, cho phép việc mở rộng chức năng mà không cần sửa đổi mã nguồn hiện tại.

Override

```cpp
#include <iostream>
using namespace std;

class cha{
    public:
        virtual void display(){                              // Hàm ảo
            cout << "display from class cha" << endl;
        }
};

class con : public cha{
    public:
        void display() override{                             // Ghi đè hàm
ảo
            cout << "display from class con" << endl;
        }
};

int main(){
    cha *ptr;
    con obj;

    // trỏ con trỏ class cha đến đối tượng class con
    ptr = &obj;

    // Gọi hàm ảo
    ptr->display();
}
```

Override

```cpp
#include <iostream>
#include <string>

using namespace std;

class DoiTuong{
    protected:
        string ten;
        int id;

    public:
        DoiTuong(){
            static int ID = 1;
            id = ID;
            ID++;
        }

        void setName(string _ten){
            // check chu ỗi nhập vào
            ten = _ten;
        }

        virtual void display(){
            cout << "ten: " << ten << endl;
            cout << "id: " << id << endl;
```

Pure Virtual Function

- Hàm thuần ảo là một **hàm ảo** không có phần định nghĩa trong class cha, được khai báo với **cú pháp = 0** và khiến class cha trở thành **class trừu tượng (abstract class)**, nghĩa là không thể tạo đối tượng từ class này.

Pure Virtual Function

```cpp
#include <iostream>
using namespace std;

class cha{
    public:
        virtual void display() = 0; // Hàm ảo thuần túy
};

class con : public cha{
    public:
        void display() override{    // Ghi đè hàm ảo thuần
túy
            cout << "display from class con"  << endl;
        }
};

int main(){
    // cha ptr; // wrong
    cha *ptr;
    con obj;

    ptr = &obj;
    ptr->display();

    return 0;
```

Pure Virtual Function

```cpp
#include <iostream>
#include <string>
using namespace std;

class Xe
{
    protected:
        string model;
        int namSanXuat;

    public:
        Xe(string m, int nam): model(m), namSanXuat(nam){}

        virtual void hienThiThongTin() = 0;
};

class Toyota : public Xe
{
    private:
        string dongCo;

    public:
        Toyota(string m, int nam, string dongCo): Xe(m,nam), dongCo(dongCo){}
```

Đa kế thừa

- Đa kế thừa trong C++ cho phép một class kế thừa từ nhiều class khác.
- Đa kế thừa thường dùng để kết hợp các chức năng từ nhiều class.

Đa kế thừa

```cpp
#include <iostream>
using namespace std;

class Sensor{
    public:
        void initialize(){
            cout << "Initializing sensor" << endl;
            // code khởi tạo cảm biến
        }

        int readData(){
            cout << "Reading sensor data" << endl;
            // code đọc dữ liệu cảm biến
            return 30;
        }
};

class Communication{
    public:
        void setupCommunication() {
            cout << "Setting up communication protocol" << endl;
            // code thiết lập giao thức truyền thông (SPI, I2C, UART,...)
        }

        void sendData(int data) {
```

Đa kế thừa

- Khi nhiều lớp cha có các phương thức hoặc thuộc tính trùng tên, việc gọi chúng từ lớp con có thể gây ra sự nhầm lẫn.
- Khi một lớp con kế thừa từ hai lớp cha, mà hai lớp cha này đều cùng kế thừa từ cùng một lớp khác. Tình huống này tạo ra cấu trúc hình thoi (diamond), do đó được gọi là vấn đề "Diamond".

Đa kế thừa

```
#include <iostream>

using namespace std;

class A{
    public:
        A(){ cout << "Constructor A \n"; }

        void hienThiA(){ cout << "Day la lop A \n"; }
};

class B : public A{
    public:
        B(){ cout << "Constructor B \n"; }

        void hienThiB(){ cout << "Day la lop B \n"; }
};

class C : public A {
    public:
        C(){ cout << "Constructor C \n"; }

        void hienThiC(){ cout << "Day la lop C \n"; }
};
```

Kế thừa ảo

- Kế thừa ảo giúp tránh vấn đề diamond problem trong đa kế thừa.
- Chỉ có **một bản sao duy nhất** của lớp cơ sở chung được kế thừa.
- Kế thừa ảo giúp quản lý các lớp liên quan đến phần cứng và giao tiếp. Điều này giúp tránh trùng lặp tài nguyên và quản lý hiệu quả trong hệ thống nhúng.

Kế thừa ảo

```
#include <iostream>
using namespace std;

class A {
    public:
        A(){ cout << "Constructor A\n"; }

        void hienThiA(){ cout << "Day la lop A\n"; }
};

class B : virtual public A{
    public:
        B(){ cout << "Constructor B\n"; }

        void hienThiB(){ cout << "Day la lop B\n"; }
};

class C : virtual public A {
    public:
        C(){ cout << "Constructor C\n"; }

        void hienThiC(){ cout << "Day la lop C\n"; }
};

class D : public B, public C{
```