NGUYÊN LÝ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 5: HÀM ẢO VÀ ĐA HÌNH

Trần Thị Ngân

Email: ngantt@tlu.edu.vn

Nội dung

- **Cơ bản về hàm ảo, đa hình**
- **Con trỏ và hàm ảo**

Câu hỏi

- So sánh nạp chồng hàm và khuôn mẫu hàm?
- So sánh nạp chồng hàm và định nghĩa hàm trong lớp kế thừa?

Cơ bản về hàm ảo

- Kết gán **muộn**
- Thi hành hàm ảo
- Khi nào sử dụng hàm ảo
- Lớp trừu tượng và hàm ảo thuần túy

Một vài khái niệm cơ bản

- Khái niệm **kết gán**: là gắn thông báo với 1 hàm cụ thể
- **Kết gán sớm**: kết gán khi biên dịch chương trình (các hàm có sẵn hoặc các hàm người dùng định nghĩa)
- **Kết gán muộn**: kết gán khi thực hiện chương trình (các hàm ảo)

Một vài khái niệm cơ bản

- **Đa hình (polymorphism)**
  - Liên kết nhiều ngữ nghĩa với một hàm
  - Hàm ảo cung cấp khả năng này
  - Là nguyên tắc cơ bản của lập trình hướng đối tượng
- **Ảo:** Tồn tại về bản chất mặc dù trên thực tế không tồn tại
- **Hàm ảo (virtual):** Một hàm ảo là một hàm thành viên trong lớp chính mà bạn cần định nghĩa lại trong lớp kế thừa

Ví dụ về hàm ảo

- **Trong một trò chơi (game) có lớp vũ khí – Weapon (lớp cơ sở) và các lớp con (lớp kế thừa) gồm:**

- Lớp Bomb (Bom)
- Lớp Gun (Súng)
- …

- **Cần xây dựng 1 hàm features() để nạp các tính năng cho vũ khí trong game.**

Ví dụ về hàm ảo

- **Lớp vũ khí (lớp cơ sở) - Weapon được xây dựng trước**

```
#include <iostream>
using namespace std;
class Weapon
{
public: void features()
        { cout << "Loading weapon features.\n"; }
 };
```

Ví dụ về hàm ảo

- **Các lớp kế thừa của Weapon được xây dựng**

```
class Bomb: public Weapon
{
public: void features()
            {cout<<"Loading bomb features.\n"; }
};
class Gun : public Weapon
{ public: void features()
            { cout << "Loading gun features.\n"; }
};
```

Ví dụ về hàm ảo

➢ **Gọi hàm như sau sẽ cho kết quả như thế nào?**

```
int main()
{
    Weapon *w = new Weapon;
    Bomb *b = new Bomb;
    Gun *g = new Gun;
    w->features();
    b-> features() ;
    g-> features() ;
    return 0;
}
```

Ví dụ về hàm ảo

- **Kết quả: Màn hình sẽ hiển thị các thông báo sau**

Loading weapon features.

Loading bomb features.

Loading gun features.

Ví dụ về hàm ảo

- ***Đặt vấn đề***: Trò chơi ngày càng phát triển, các vũ khí ngày càng nhiều nên cần tạo 1 lớp riêng Loader để nạp tính năng cho các loại vũ khí.
- Lớp Loader được định nghĩa như sau:

```
class Loader
{
    public: void loadFeatures(Weapon *weapon)
                { weapon->features(); }
};
```

Ví dụ về hàm ảo

```
int main()
{
    Loader *l = new Loader;
    Weapon *w;
    Bomb b;
    Gun g;
    w = &b;
    l->loadFeatures(w);
    w = &g;
    l->loadFeatures(w);
    return 0;
}
```

Ví dụ về hàm ảo

- **Kết quả: Màn hình sẽ hiển thị các thông báo sau**

Loading weapon features.

Loading weapon features.

- **Giải thích kết quả ???**

Ví dụ về hàm ảo

➢ Giải thích kết quả ???

- Ban đầu, đối tượng Weapon *w* đang trỏ vào đối tượng *b* (thuộc lớp Bomb).
- Sau đó chúng ta cố gắng nạp tính năng của đối tượng Bomb này bằng cách truyền nó vào hàm loadFeatures() sử dụng đối tượng con trỏ *l* (thuộc lớp Loader).
- Tương tự, chúng ta đã thử nạp vào tính năng của đối tượng Gun.
- Tuy nhiên, hàm loadFeatures() của lớp Loader nhận con trỏ trỏ tới đối tượng của lớp Weapon làm đối số

Ví dụ về hàm ảo

- **Giải quyết vấn đề:**
  - Xây dựng hàm để lời gọi hàm thực thi theo đối tượng gọi nó chứ không phải kiểu đối tượng được khai báo ban đầu

Ví dụ về hàm ảo

- **Giải quyết vấn đề:**

```
int main()
{
    Loader *l = new Loader;
    Weapon *w;
    Bomb b;
    Gun g;
    w = &b;//con trỏ trỏ tới đối tượng Bomb
    l->loadFeatures(w); //Nạp tính năng cho bom
    w = &g; //con trỏ trỏ tới đối tượng Gun
    l->loadFeatures(w); //Nạp tính năng cho súng
    return 0;
}
```

Giải pháp

Để giải quyết vấn đề này, bạn cần biến hàm *void features()* thuộc lớp chính (lớp Weapon) thành hàm ảo

Xử lý

Thêm từ khóa **virtual** vào trước hàm *void features():*

```
class Weapon
{
    public:
      virtual void features()
        { cout << "Loading weapon features.\n"; }
};
```

Chương trình – Xây dựng lớp

```
1   #include <iostream>
2   using namespace std;
3   class Weapon
4   {
5       public:
6         virtual void features()
7             { cout << "Loading weapon features.\n"; }
8   };
9   class Bomb : public Weapon
10  {
11      public:
12         void features()
13            { cout << "Loading bomb features.\n"; }
14  };
15  class Gun : public Weapon
16  {
17      public:
18         void features()
19            { cout << "Loading gun features.\n"; }
20  };
21  class Loader
22  {
23     public:
24       void loadFeatures(Weapon *weapon)
25       {
26          weapon->features();
27       }
28  };
```

Chương trình – Hàm main()

```
29  int main()
30  {
31      Loader *l = new Loader;
32      Weapon *w;
33      Bomb b;
34      Gun g;
35      w = &b;
36      l->loadFeatures(w);
37      w = &g;
38      l->loadFeatures(w);
39      return 0;
40  }
```

Kết quả

```
Loading bomb features.
Loading gun features.


--------------------------------
```

Lợi thế của hàm ảo

- ✓ Duy trì tính đa hình: gọi hàm thành viên của lớp khác với cùng lời gọi hàm phụ thuộc vào ngữ cảnh khác nhau.

```
Weapon *w; //con trỏ kiểu Weapon
    Bomb b;
    Gun g;
    w = &b; //trỏ tới đối tượng kiểu Bomb
    l->loadFeatures(w);//Gọi hàm features() của lớp Bomb
    w = &g; //trỏ tới đối tượng kiểu Gun
    l->loadFeatures(w);//Gọi hàm features() của lớp Gun
```

Lợi thế của hàm ảo

- ✓ Sử dụng hàm ảo khiến chương trình không chỉ rõ ràng hơn mà còn khả chuyển hơn.

Ví dụ: Nếu chúng ta muốn thêm vào một vũ khí khác (ví dụ như dao - Knife), chúng ta có thể dễ dàng thêm vào và nạp tính năng của nó.

Bổ sung lớp Knife

```
class Knife : public Weapon
{
   public:
      void features()
        { cout << "Loading knife features.\n"; }
};
```

Chỉnh sửa hàm main() có gọi hàm nạp tính năng cho đối tượng Knife

Chỉnh sửa hàm main()

```
int main()
{
    Loader *l = new Loader;
    Weapon *w;
    Bomb b;
    Gun g;
    Knife k;
   w = &b;
    l->loadFeatures(w);
    w = &g;
    l->loadFeatures(w);
    w = &k;
    l->loadFeatures(w);
    return 0;
}
```

Ví dụ

- Xây dựng lớp Nguoi gồm

```
protected:
string hoten;
public:
virtual void nhap();
virtual int thuong()=0;//thưởng
virtual void xuat();
```

Xây dựng 2 lớp kế thừa từ lớp Nguoi là Sinhvien có thêm biến dttn (điểm thi tốt nghiệp) và Giangvien có thêm biến sobaibao. Sinhvien được thưởng khi dttn > 8 và Giangvien được thưởng khi sobaibao>3

Ví dụ

Lớp Nguoi:

```
#include <iostream>
#include<string>
#include<vector>
using namespace std;
class Nguoi
{   protected:
        string hoten;
    public:
        virtual void nhap();
        virtual int thuong()=0;
        virtual void xuat();
};
void Nguoi::nhap()
{   fflush(stdin);
    cout<<"Ho Ten : ";getline(cin,hoten);
}
void Nguoi::xuat()
{
    cout<<" Ho ten : "<<hoten;
}
```

Ví dụ

Lớp Sinhvien:

```
class Sinhvien:public Nguoi
{
    public:
        void nhap();
        void xuat();
        int thuong();
    private:
        float dttn;
};
void Sinhvien::nhap()
{
    cout<<endl<<"Sinh vien : "<<endl;
    Nguoi::nhap();
    cout<<"Diem thi tn: ";cin>>dttn;
}
int Sinhvien::thuong()
{
    if (dttn>8) return 1;
    else return 0;
}
void Sinhvien::xuat()
{
```

Ví dụ

Lớp Giangvien:

```
class Giangvien:public Nguoi
{
    public:
        void nhap();
        void xuat();
        int thuong();
    private :
        int sbb;
 };
void Giangvien::nhap()
{
    cout<<endl<<"Giang vien : "<<endl;
    Nguoi::nhap();
    cout<<"So bai bao : ";cin>>sbb;
}
int Giangvien::thuong()
{
    if (sbb>3) return 1;
    else return 0;
}
void Giangvien::xuat()
{
```

Lợi thế của hàm ảo

- ✓ Một lợi thế khác của hàm ảo là giải phóng bộ nhớ của các đối tượng như yêu cầu

```
class Animal
{ public:
        ~Animal() {cout<<"This is Animal's destructor\n";}
};
class Cat : public Animal
{ public:
        ~Cat(){cout<<"This is Cat's destructor\n";}
};
```

Ví dụ

Hàm main()

```
int main()
{
        Animal* tom = new Cat();
        delete tom; //gọi hàm hủy của đối tượng tom
}
```

Kết quả: ???

This is Animal's destructor

Đánh giá kết quả

- Chúng ta có thể thấy ngay là hàm hủy của lớp Cat đã không được gọi, mặc dù tom được khởi tạo bằng hàm tạo của lớp Cat.
- Điều này thật sự rất nguy hiểm, vì hàm hủy của lớp Cat không được gọi nên các đối tượng riêng của lớp đó cũng không được giải phóng và vì thế đối tượng tom chỉ được giải phóng 1 phần tài nguyên.
- Điều này gây ra rò rỉ bộ nhớ (hiện tượng bộ nhớ đã được cấp phát không thu hồi lại được).
- Để khắc phục chúng ta thêm từ khoá **virtual** vào trước hàm hủy của lớp cơ sở

Thực hiện

```
class Animal {
public:
    virtual ~Animal(){printf("This is Animal's destructor\n");};
};

class Cat : public Animal {
public:
    ~Cat(){printf("This is Cat's destructor\n");};
};


int main()
{
    Animal* tom = new Cat();
    delete tom;
}.
```

Ghi đè

- Khi định nghĩa hàm ảo thay đổi trong lớp dẫn xuất: hàm này được ghi đè
- Tương tự như được định nghĩa lại cho các hàm chuẩn
- Do vậy:

  - Hàm ảo bị thay đổi: ghi đè
  - Hàm không phải hàm ảo bị thay đổi: định nghĩa lại

Nhược điểm của hàm ảo

- Nhược điểm chính: phụ phí

- Sử dụng nhiều bộ nhớ hơn

- Kết gán muộn nghĩa là kết gán "trong khi chạy", do vậy chương trình chạy chậm hơn

- Do vậy không nên sử dụng hàm ảo nếu không thực sự cần thiết

Hàm ảo thuần túy (pure virtual)

Nhắc lại ví dụ về lớp Nguoi: Tất cả mọi người đều là các đối tượng của các lớp dẫn xuất khác nhau: sinh viên, giảng viên, công chức, nông dân,….

Hàm ảo thuần túy (pure virtual)

Tuy nhiên không phải đối tượng nào cũng được thưởng hoặc được thưởng với các tiêu chí như nhau. Do đó ở lớp cơ sở, không biết xây dựng hàm thuong() như thế nào.

➢ ***Giải pháp:***

Đặt nó là một **hàm ảo thuần túy**:

virtual void thuong() = 0;

Hàm ảo thuần túy (pure virtual)

- Hàm ảo thuần túy là hàm ảo trong lớp cơ sở mà **không có định nghĩa rõ ràng (có ý nghĩa).**
- Mục đích chỉ để các lớp khác kế thừa

Lớp cơ sở trừu tượng

- Hàm ảo thuần túy không đòi hỏi phải định nghĩa. Tất cả các lớp dẫn xuất định nghĩa phiên bản của riêng chúng
- Lớp với một hoặc nhiều hàm ảo thuần túy là lớp cơ sở trừu tượng:

- Có thể được sử dụng như lớp cơ sở
- Không có đối tượng nào có thể được tạo ra từ nó

Lớp cơ sở trừu tượng

- Lớp cơ sở trừu tượng không có các định nghĩa hoàn chỉnh cho tất cả các thành viên của nó.
- Nếu lớp dẫn xuất thất bại trong việc định nghĩa tất cả các hàm ảo thuần túy, nó cũng là lớp cơ sở trừu tượng.

Ví dụ

- Với các lớp Nguoi, Sinhvien, Giangvien được định nghĩa như ở ví dụ trên thì **khai báo nào** trong hàm main() như dưới đây sẽ báo lỗi?

```
int main()
{
  Nguoi A;
  Sinhvien SV;
  Giangvien GV;
}
```

```
Message
In function 'int main()':
[Error] cannot declare variable 'A' to be of abstract type 'Nguoi'
[Note] because the following virtual functions are pure within 'Nguoi':
[Note] virtual int Nguoi::thuong()
```

VÍ DỤ

Đoạn chương trình sau cho kết quả như thế nào?

VÍ DỤ

```
#include<iostream>
using namespace std;
class BaseClass{
public:
    void Display()          {
    cout<<"Goi ham cua lop co so!"<<endl;
     }
    void Show()         {
            Display();
      }
};
class DerivedClass : public BaseClass{
    public:
        void Display()      {
            cout<<"Goi ham cua lop ke thua!"<<endl;     }
};
int main(){
    DerivedClass Dr;
    Dr.Show();
}
```

VÍ DỤ

```
#include<iostream>
using namespace std;
class BaseClass{
public:
    virtual  void Display()         {
    cout<<"Goi ham cua lop co so!"<<endl;
     }
    void Show()         {
            Display();
      }
};
class DerivedClass : public BaseClass{
    public:
        void Display()       {
            cout<<"Goi ham cua lop ke thua!"<<endl;      }
};
int main(){
    DerivedClass Dr;
    Dr.Show();
}
```

VÍ DỤ

Sử dụng định nghĩa các lớp Nguoi, Giangvien, Sinhvien như trên xây dựng hàm main() khai báo và nhập thông tin cho **1 danh sách bao gồm cả giảng viên và sinh viên**. Hiển thị danh sách các giảng viên và sinh viên được thưởng trong danh sách đã nhập.

VÍ DỤ

Gợi ý:

- Sử dụng 1 danh sách kiểu Nguoi,
- Cho lựa chọn nhập giảng viên-sinh viên-thoát (dạng menu)
  - ✓Nếu chọn giảng viên: Nhập thông tin giảng viên,
  - ✓Nếu chọn sinh viên: Nhập thông tin sinh viên
  - ✓Nếu chọn thoát: Hiển thị danh sách các giảng viên, sinh viên được thưởng trong danh sách đã nhập.

Tương thích kiểu mở rộng

➢ Biết rằng: Lớp con được dẫn xuất từ lớp cha, khi đó

- Các đối tượng kiểu lớp con có thể được gán cho các đối tượng kiểu lớp cha

- Nhưng điều ngược lại thì không đúng

➢ Xét ví dụ trước:

Một đối tượng kiểu sinhvien là một đối tượng kiểu Nguoi nhưng ngược lại thì không đúng

Ví dụ tương thích kiểu mở rộng

```
class Pet
{
public:
string name;//có thể truy cập trực tiếp
virtual void print();
};
class Dog : public Pet {
public:
string breed;
void print();
};
```

Lớp Pet và Dog

- Khai báo:

  Dog vdog;

  Pet vpet;

Sử dụng lớp Pet và Dog

- Một chú chó cũng là một thú cưng:

  vdog.name = "Tiny";

  vdog.breed = “Husky";

  vpet = vdog;// *được phép*

- Có thể gán các giá trị cho kiểu cha, nhưng ngược lại thì không.

  Một thú cưng không phải là một chú chó

Vấn đề tách lớp

- Lưu ý rằng giá trị được gán cho vpet mất trường breed của nó nên lệnh in:

```
cout<<vpet.breed;
```

Tạo ra thông điệp lỗi: Được gọi là vấn đề tách lớp

- Thích hợp: biến kiểu Dog được chuyển thành biến kiểu Pet, do vậy nó sẽ được đối xử như một Pet. Và do vậy không có các tính chất của chó
- Tạo ra tranh cãi triết học thú vị

Chỉnh sửa vấn đề tách lớp

- Trong C++, vấn đề tách lớp là trở ngại

- Vpet vẫn là một Husky tên là Tiny

- Chúng ta muốn nói tới giống của nó thậm chí khi nó được đối xử như một Pet

- Có thể làm như vậy bằng các con trỏ trỏ tới các biến động

Ví dụ vấn đề tách lớp

```
Pet *ppet;
Dog *pdog;
pdog = new Dog;
pdog->name = "Tiny";
pdog->breed = "Great Dane";
ppet = pdog;
//Không thể truy cập trường breed của đối tượng
//được trỏ tới bởi ppet:
cout << ppet->breed;
```

Ví dụ vấn đề tách lớp

```
Pet *ppet;
Dog *pdog;
pdog = new Dog;
pdog->name = "Tiny";
pdog->breed = "Great Dane";
ppet = pdog;
//Không thể truy cập trường breed của đối tượng
//được trỏ tới bởi ppet:
cout << ppet->breed;
```

Ví dụ vấn đề tách lớp

Phải sử dụng hàm thành viên ảo:

ppet->print();

Lệnh này gọi hàm thành viên print() trong lớp Dog bởi vì nó là ảo.

C++ đợi để xem đối tượng nào con trỏ ppet đang trỏ tới trước khi gọi kết gán.

Ví dụ: Định nghĩa các lớp Pet, Dog

```
1  #include <iostream>
2  using namespace std;
3  class Pet
4  {
5  public:
6      string name;//chi dung cho vi du
7      virtual void print();
8  };
9  void Pet::print()
10 {
11     cout<<"Name: "<<name;
12 }
13 class Dog : public Pet {
14 public:
15     string breed;
16     void print();
17 };
18 void Dog::print(){
19     cout<<"Name: "<<name;
20     cout<<"\nBreed: "<<breed;
21 }
```

Ví dụ: Hàm main()

```
22  int main()
23  {
24      Pet *ppet;
25      Dog *pdog;
26      pdog = new Dog;
27      pdog->name = "Tiny";
28      pdog->breed = "Husky";
29      ppet = pdog;
30      ppet->print();
31  }
```

Ép kiểu

- Xét:

```
Pet vpet;
Dog vdog;
vdog = static_cast<Dog>(vpet);
```

- Không thể ép một thú cưng thành một chú chó, nhưng:

```
vpet = vdog;
vpet = static_cast<Pet>(vdog);
```

- Ép kiểu lên thực hiện được từ kiểu con cháu đến kiểu tổ tiên

Ép kiểu xuống

- Ép kiểu xuống khá nguy hiểm: Ép từ kiểu cha xuống kiểu con.

- Giả sử thông tin được thêm vào
- Có thể thực hiện với dynamic_cast:

```
Pet *ppet;
ppet = new Dog;
Dog *pdog = dynamic_cast<Dog*>(ppet);
```

- Hợp lệ, nhưng nguy hiểm
- Ép kiểu xuống hiếm khi được sử dụng do:

- Phải theo dõi tất cả thông tin được thêm vào
- Tất cả các hàm thành viên phải là ảo

Hàm hủy ảo

- Xét:

```
Base *pBase = new Derived;
delete pBase;
```

Sẽ gọi hàm hủy lớp cơ sở mặc dù đang trỏ tới đối tượng lớp Derived

- Khai báo hàm hủy là virtual để chỉnh sửa vấn đề này
- Có thể để tất cả các hàm hủy là ảo

Công việc nội tại của hàm ảo

- Không cần biết sử dụng nó như thế nào: Quy tắc che dấu thông tin
- Bảng hàm ảo

- Bộ biên dịch tạo ra nó
- Có các con trỏ cho mỗi hàm thành viên ảo
- Trỏ tới vị trí mã lệnh của hàm đó

- Các đối tượng của các lớp như vậy cũng có con trỏ: Các con trỏ này trỏ tới bảng hàm ảo

MỘT SỐ CHÚ Ý

- Nếu không có từ khóa virtual, các hàm được xây dựng ở lớp cha khi thực thi sẽ không có tính đa hình
- Không được tạo đối tượng thuộc lớp trừu tượng
- Nếu hàm ảo ở ở lớp cha mà chưa thể xác định cách thực hiện thì sử dụng hàm ảo thuần túy
- Cách thức thực hiện xác định cụ thể ở lớp bằng các định nghĩa lại hàm ảo thuần túy

MỘT SỐ CHÚ Ý

- Kết gán muộn hoãn việc quyết định hàm thành viên nào được gọi cho đến khi chương trình chạy
  - Trong C++ các hàm ảo sử dụng kết gán muộn
- Các hàm ảo thuần túy không có định nghĩa
  - Các lớp có ít nhất một hàm ảo thuần túy là trừu tượng
  - Không có đối tượng nào có thể tạo ra từ lớp trừu tượng
  - Được sử dụng làm cơ sở cho các lớp khác dẫn xuất

MỘT SỐ CHÚ Ý

- Đối tượng lớp dẫn xuất có thể được gán cho đối tượng lớp cơ sở. Các thành viên lớp cơ sở bị mất: vấn đề tách lớp
- Phép gán con trỏ và các đối tượng động: Cho phép chỉnh sửa vấn đề tách lớp
- Khai báo tất cả các hàm hủy là ảo:
  - Áp dụng tốt cho việc lập trình
  - Đảm bảo bộ nhớ được giải phóng đúng và giải phóng hoàn toàn.

BÀI TẬP THỰC HÀNH

Sử dụng định nghĩa các lớp Nguoi, Giangvien, Sinhvien như ở ví dụ trên (cả 3 hàm Nhap(), Xuat(), thuong() đều là hàm ảo) xây dựng hàm main() khai báo và nhập thông tin cho **1 danh sách bao gồm cả giảng viên và sinh viên**. Hiển thị danh sách các giảng viên và sinh viên được thưởng trong danh sách đã nhập.