Bài giảng trên lớp

Bài giảng trên lớp là tài liệu gợi ý và hỗ trợ giảng viên trong quá trình lên lớp, tuy nhiên giảng viên cần có sự chuẩn bị của riêng mình cho phù hợp với từng lớp học.

Nếu giảng viên thiết kế bài giảng tốt hơn tài liệu đã cung cấp, xin hãy chủ động làm và gửi lại cho chúng tôi.

Nếu giảng viên cần thay đổi tài liệu đã cung cấp, những thay đổi có liên quan đến cấu trúc, nội dung kiến thức và có tính đổi mới, xin hãy chủ động làm và gửi lại cho chúng tôi.

Mọi ý kiến xin gửi cho Phòng NC-PTCT để được xem xét và ban hành chính thức và góp phần ngày càng hoàn thiện hơn học liệu của trường.

Trân trọng cảm ơn.

Liên hệ: Phòng NC-PTCT – FPT Polytechnic

Những ý tưởng đổi mới sẽ được xem xét và gửi qua Dự án CNGD để có sự hỗ trợ giảng viên làm nghiên cứu khoa học, hoặc hướng dẫn viết bài báo đăng trên tạp chí CNGD.

Lập trình C++
Bài 8 Kết hợp và kế thừa

Hệ thống bài cũ

- Tổng quan về toán tử nạp chồng
- Cú pháp khai báo toán tử
- Phân loại hàm toán tử
- Toán tử chuyển kiểu
- Con trỏ this

MỤC TIÊU

- Hiểu được các khái niệm kế thừa, đa hình, hàm hủy và cách sử dụng

Nội dung

- Kết hợp
- Kế thừa
- Hàm virtual và Tính đa hình
- Hàm hủy ảo

Kết hợp

Chỉ việc sử dụng một hoặc nhiều lớp trong phần định nghĩa của một lớp khác.

Ví dụ:

```
class Person{
 public:
     Person (char* n="", char* nat="U.S.A.", int s=1):
     name(n), nationality(nat), sex(s) { };
     void printName() { cout << name; }
      void printNationality() { cout << nationality; }
private:
      string name, nationality;
      int sex;
};
```

Ví dụ này mô tả tính kết hợp của lớp string trong lớp Person.

Tính kết hợp là một cách để tái sử dụng chương trình cũ để tạo ra chương trình mới.

Tính kết hợp

Kế thừa

Vấn đề: Để quản lý nhân sự của một công ty, ta có thể định nghĩa các lớp tương ứng như sau

```
class CongNhan{
private:
string ten; float luong;
int bac;
public:
string layTen( )  {...}
void luong( ) {...}
void congviec( ) {...}
...
};
```

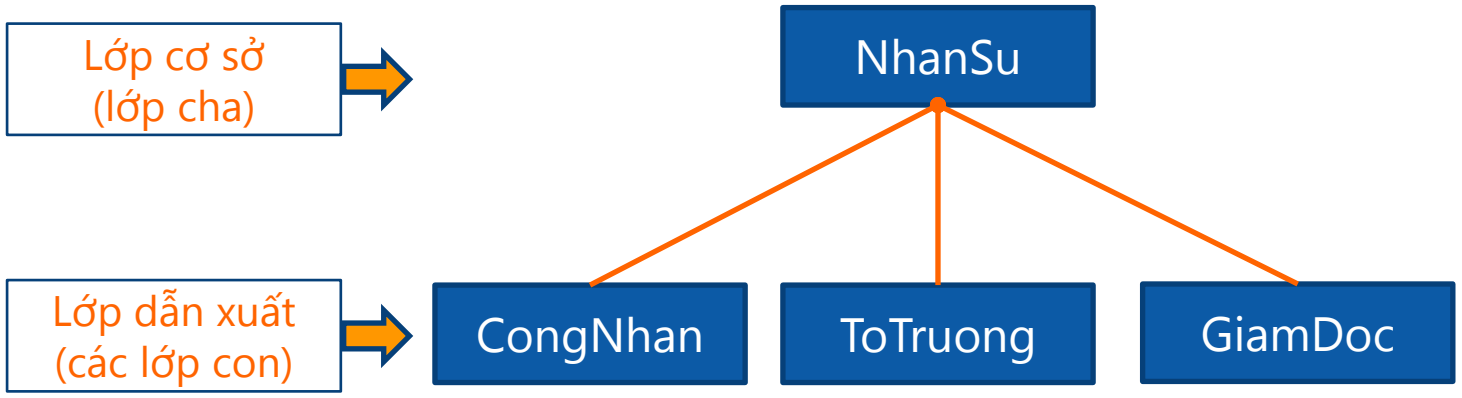
```
class ToTruong{
private:
string ten; float luong;
public:
string layTen( )   {...}
void   luong( ) {....}
void congviec( ) {...}
...
};
```

```
class GiamDoc{
private:
string ten; float luong;
public:
string layTen( )   {...}
void   luong( ) {....}
void congviec( ) {...}
...
};
```

Cả 3 lớp trên đều có những biến và hàm giống hệt nhau về nội dung -> tạo ra một lớp NhanSu chứa các thông tin chung đó để sử dụng lại. Do đó nó có các lợi ích như sau:

- Sử dụng lại code
- Giảm số code cần viết
- Dễ bảo trì, sửa đổi về sau
- Rõ ràng hơn về mặt logic trong thiết kế chương trình

Kế thừa



Cụ thể hoá: lớp con là một trường hợp riêng của lớp cha (như ví dụ trên)

Tổng quát hoá: mở rộng lớp cha (vd: Point2D thêm biến z để thành Point3D)

Kế thừa cho phép các lớp con sử dụng các biến và phương thức của lớp cha như của nó, trừ các biến và phương thức private

Kế thừa với public và private:

public: các thành phần public của lớp cha vẫn là public trong lớp con

private: toàn bộ các thành phần của lớp cha trở thành private của lớp con

Kế thừa Public

```
class NhanSu{
private:
    string ten;
    float luong;
public:
     string layTen() ;
     void luong();
};
```

1

```
class CongNhan: public NhanSu{
private:
            int bac;
            .....
public:
            void   congviec( )     {....}
              ..............
void hienthi( ){
   cout << layTen()<< luong; // lỗi
}
};
```

2

```
CongNhan cn;
cn.layTen();
cn.congviec();
cn.luong();
cn.luong = 50; // lỗi
cn.hienthi();

NhanSu objns= cn; // OK
CongNhan cn 2 = objns;     // lỗi
CongNhan cn 3 = (NhanSu ) objns;  // lỗi
```

3

Các thành phần public của lớp cha vẫn là public trong lớp con

Lớp con chuyển kiểu được thành lớp cha, nhưng ngược lại không được

Kế thừa Private

```
class LinkedList {
 private:
   .....
public:
void insertTail (int x)
void insertHead (int x)
void deleteHead ( ) {...}
void deleteTail (){...}
int getHead ( ) { ... }
int getTail ( ) { ... }
};
```

1

```
class Stack : private LinkedList {
public:
    void push(int x)
       { insertHead(x); }
int pop() {
   int x = getHead();
   deleteHead();
return x;
}
};
```

2

```
Stack s;
s.push(30);
s.push(20);
s.pop ( );
s.insertTail(30);       // lỗi
s.getTail( );           // lỗi
```

3

Tất cả các thành phần của lớp cha đều trở thành private của lớp con

Thành phần protected

Ngoài public và private, còn có các thành phần protected: có thể được sử dụng bởi các phương thức trong lớp dẫn xuất từ nó, nhưng không sử dụng được từ ngoài các lớp đó

```
class NhanSu{
protected:
    string ten;    float gia;     int gio;
    int luong( )  {return gio*gia;}
public:
   void nhapTen(const char* s)  { ten= s; }
   string layTen( ) { return ten; }
   void luong();
.....
};
```

1

```
class CongNhan : public NhanSu{
public:
        void   congviec( )     {....}
        void hienthi( )
        {
         cout << "Ten: " << ten << "Luong: "
         << luong( );
         }
...
};
```

2

```
CongNhan cnv;
cnv.congviec();
cnv.luong();
cnv.hienthi();

CongNhan cnv.ten = "NV Hoa";              // lỗi
cout << cnv.luong();  // lỗi
```

3


DEMO
Tính kế thừa

Tổng kết các kiểu kế thừa

		Kiểu kế thừa		
		private	protected	public
Phạm vi	private	(không)	(không)	(không)
	protected	private	protected	protected
	public	private	protected	public

Cột: các kiểu kế thừa

Hàng: phạm vi các biến/phương thức thành phần trong lớp mẹ

Kết quả: phạm vi các biến/phương thức trong lớp dẫn xuất

Constructor và Destructor trong kế thừa

- Constructor và destructor không được các lớp con thừa kế
- Mỗi constructor của lớp dẫn xuất phải gọi một constructor của lớp cha, nếu không sẽ được ngầm hiểu là gọi constructor mặc định

```
class Pet {
public:
      Pet ()  {...}
      Pet(string name) {...}
};
class Dog: public Pet {
public:
      Dog (){...}            //Pet()
      Dog(string name): Pet(name) {...}
};
```

```
class Bird {
public:
       Bird(bool canFly) {...}
};
class Eagle: public Bird {
public:
          // sai: Eagle() {...}
    Eagle(): Bird(true) {...}
};
```

Destructor của các lớp sẽ được gọi tự động theo thứ tự ngược từ lớp dẫn xuất tới lớp cơ sở

~Dog() -> ~Pet()

~Eagle() ~Bird()

Gọi Constructor của lớp cha trong Constructor của lớp con

Không thể gọi Constructor của lớp cha trong Constructor của lớp con như hàm, mà phải gọi ở danh sách khởi tạo.

Ví dụ:

```
class Point3D: private Point2D {
protected: float z;
public:
    Point3D(): Point2D(0., 0.), z(0.)       //đúng
        { ... }
    Point3D(double x, double y, double z)
    // gọi Constructor mặc định Point2D()
    {
    Point2D(x, y); // sai: tạo đối tượng Point2D tạm this->z = z;
    };
...
};
```


DEMO
Constructor &
Destructor trong
kế thừa

Phương thức ảo

Giả sử có các lớp Animal, Dog và Cat như sau:

```
class Animal{
protected:
    char name[30];
public:
    Animal(char *name);
    void Speak();
};

Animal::Animal(char *name)
   { strcpy(this->name, name); }

void Animal::Speak()
   { cout<<"Hello, I am an animal"}
```

Animal, Dog và Cat

```
// lớp Dog
class Dog : public Animal
{
public:
    Dog(char *name) : Animal(name) {}
    void Speak(); // nạp chồng Speak của lớp cha
};

void Dog::Speak()
{
    cout<<"My name is "<<name<<", go="" !"<<endl;
}
```

Animal, Dog và Cat

```
// lớp Cat
class Cat : public Animal
{
public:
	Cat(char *name) : Animal(name) {}
	void Speak(); // nạp chồng Speak của lớp cha
};

void Cat::Speak()
{
	cout<<"My name is "<<name<<", meoo="" !"<<endl;
}
```

Animal, Dog và Cat

```
// hàm main
void main()
{
   Animal* ani; // con trỏ tới các đối tượng Animal
   Dog dog("Tony"); // đối tượng thuộc lớp Dog
   Cat cat("Fluffy"); // đối tượng thuộc lớp Cat

   ani = &dog;
   ani ->Speak();

   ani = &cat;
   ani ->Speak();
}
```

Animal, Dog và Cat

Kết quả màn hình:

- Hello, I am an animal
- Hello, I am an animal

Giải thích:

- Khi biên dịch, chương trình sẽ gắn lời gọi Speak với đối tượng của lớp Animal. Nó sẽ gọi Speak của lớp Animal khi câu lệnh được thực hiện.
- Quá trình này gọi là kết nối tĩnh (static binding) - phương thức gọi được xác định tại thời điểm dịch (compile time)

Phương thức ảo (virtual method)

- Để cho kết quả đúng như mong muốn, ta cần khai báo Speak trong Animal là phương thức ảo.

```
virtual void Speak();
```

- Phương thức ảo là phương thức của lớp cơ sở và được định nghĩa lại trong lớp dẫn xuất.
- Khi một con trỏ của lớp cơ sở gọi phương thức ảo, chương trình sẽ chọn phương thức cần thiết (dựa trên đối tượng gọi) để thực hiện.
- Quá trình này gọi là kết nối động (dynamic binding) - phương thức gọi được xác định vào lúc chạy (execution time).

Phương thức ảo (virtual method)

Kết quả sau khi khai báo Speak() là hàm ảo:

- My name is Tony, go go !
- My name is Fluffy, meoo !

Chú ý: Phương thức ảo phải được gọi thông qua con trỏ hoặc tham chiếu.

Phương thức ảo

Tính đa hình (polymorphism)

- Tính đa hình giúp dễ mở rộng chương trình. Chương trình có thể được viết để xử lý các đối tượng tổng quát rồi sau đó đưa vào các đối tượng cụ thể. Đa hình cho phép nhiều cách xử lý khác nhau với cùng một phương thức, tuỳ vào mỗi đối tượng cụ thể.
- Tính đa hình trong C++ thể hiện qua các hàm ảo (virtual). Khi một con trỏ của lớp cơ sở gọi hàm ảo, chương trình sẽ chọn hàm được gọi dựa vào đối tượng đang trỏ tới tại thời điểm chạy (execution time). Quá trình này gọi là kết nối động (dynamic binding).

Một số chú ý đối với hàm ảo

- Hàm ảo trong lớp dẫn xuất phải giống hàm của lớp cơ sở.
- Đặt từ khoá virtual với hàm ảo trong lớp cơ sở và nên đặt virtual trong cả lớp dẫn xuất.
- Nếu lớp dẫn xuất không định nghĩa lại hàm ảo của lớp cơ sở, nó sẽ sử dụng hàm của lớp cơ sở.
- Không thể khai báo cấu tử là hàm ảo.
- Có thể khai báo huỷ tử là hàm ảo.


DEMO
Tính đa hình

Hàm hủy ảo

Ví dụ:

- Animal *ani = new Cat("Buddy");
- delete ani; // huỷ tử của lớp Animal sẽ được gọi

- Nếu khai báo huỷ tử của Animal và Cat là virtual thì huỷ tử của Cat được gọi:
  - Animal *ani = new Cat("Buddy");
  - delete ani; // huỷ tử của Cat được gọi (sau đó đến // huỷ tử của Animal)
- Các lớp sử dụng hàm ảo nên khai báo huỷ tử ảo để việc huỷ đối tượng được chính xác, đặc biệt trong các lớp sử dụng bộ nhớ động.

Lớp cơ sở trừu tượng

- Lớp cơ sở trừu tượng được dùng để định nghĩa các tính chất tổng quát, chung cho các lớp khác.
  - Lớp cơ sở trừu tượng không có thể hiện (instance).
  - Trong định nghĩa của lớp cơ sở trừu tượng phải có ít nhất một hàm ảo thuần tuý.
- Hàm ảo thuần túy
  - Là hàm ảo không có cài đặt.
  - Được khai báo khởi tạo = 0;

Lớp cơ sở trừu tượng

Ví dụ:

- Lớp Hình (Shape) có thể là lớp cơ sở trừu tượng của lớp Hình tròn (Circle), Hình chữ nhật (Rectangle). Hàm ảo thuần tuý là Tính diện tích, Tính chu vi.

- Lớp Nhân sự (Employee) có thể là lớp cơ sở trừu tượng của lớp Công nhân (Worker), Người quản lý (Manager).Hàm ảo thuần tuý là Tính lương, Hiển thị thông tin.

Chú ý:
Hàm ảo thuần tuý của một lớp cơ sở trừu tượng phải được định nghĩa lại trong lớp dẫn xuất của nó, nếu không thì lớp dẫn xuất sẽ kế thừa lại hàm ảo thuần tuý đó và trở thành một lớp cơ sở trừu tượng khác.

Lớp cơ sở trừu tượng

Tổng kết

- Kết hợp
- Kế thừa
- Hàm virtual và Tính đa hình
- Hàm hủy ảo

Fpt University

FPT POLYTECHNIC

THANK YOU!

www.poly.edu.vn