Giáo trình: Xóa mù OOP

(Lập trình hướng đối tượng)

Ngôn ngữ lập trình: C++

Tác giả: Nguyên Nguyễn

Với sự hỗ trợ của AI

Mục lục

A,Các thứ tạo nên class cơ bản

I, Cấu trúc class cơ bản

```
class A{
    //code
};
```

II, Các khoá thể hiện mức độ truy cập

1.Các mức truy cập

-**private:** Được ví như két sắt. Chỉ mình mới biết mật mã

-**public:** Cửa chính mở toang. Ai cũng có thể vào. Kể cả thằng trộm cũng có thể vào được.

-**protected:** Cửa chính chỉ được mở cho con cháu họ hàng.

2, Các mức truy cập được sử dụng như thế nào

-**Private**: chủ yếu sẽ để các biến thành viên

```
private:
    int x;
    string ten;
    double y;
```

-**Public**: chủ yếu sẽ để các hàm xử lí.

```
public:
    void nhap();
    void xuat();
```

Lưu ý: Trong hàm nhap() sẽ có 2 hàm làm cho việc viết hàm nhap() trở nên trơn tru hơn.

- **cin.ignore()**: Dùng để xóa kí tự thừa trong bộ đệm

```
void nhap(){
    cout<<"nhap so: ";cin>>so; // nhap chu so
    cin.ignore() //se dat o day
    cout <<"nhap ten: "; //nhap chuoi ki tu

}
```

→Ta sẽ đặt hàm cin.igonre() giữa kiểu dữ liệu int, double, float với string

- **getline():** dùng để nhập cả khoảng trắng.

```
cout <<"nhap ten: "; //nhap chuoi ki tu
getline(cin,ten); //nhap ca khoang trang trong ten
```

III, Constructor: hàm khởi tạo

1) Constructor không tham số

Cú pháp:

```
class ClassName {
public:
    ClassName() {
        // khoi tao mac dinh
    }
};
```

Ví dụ:

```
class Student {
public:
    Student() {
        cout << "Constructor mac dinh duoc goi !" << endl;
    }
};
```

2) Constructor có tham số

Cú pháp:

```
class ClassName {
public:
    ClassName(kieu_du_lieu1 tham_so1, kieu_du_lieu2 tham_so2, ...) {
        // Gán giá tri cho thuoc tính
    }
};
```

Ví dụ:

```
class Student {
private:
    string name;
    int age;

public:
    Student(string n, int a) {
        name = n;
        age = a;
    }
};
```

IV) Destructor:

Khái niệm: Destructor là một hàm đặc biệt được gọi **tự động khi đối tượng bị hủy**, dùng để **giải phóng tài nguyên**, **dọn dẹp bộ nhớ**, hoặc **in thông báo thoát**.

Cú pháp:

```
class ClassName {
public:
    ~ClassName() {
        // Code don dep o dây
    }
};
```

Ví dụ

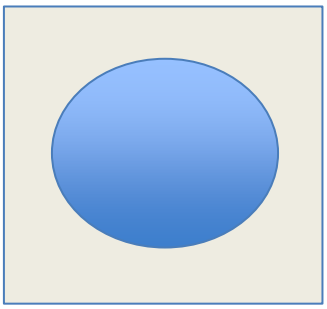
```
class Student {
public:
    Student() {
        cout << "Constructor: Doi tuong duocc tao!" << endl;
    }

    ~Student() {
        cout << "Destructor: Doi tuong bi huy!" << endl;
    }
};

int main() {
    Student s;
    // Khi ket thúc hàm main, destructor se tu chay
    return 0;
}
```

Chú ý: Nếu bạn có dùng con trỏ trong class (như cấp phát bằng new) thì destructor **cực kỳ quan trọng** để tránh **rò rỉ bộ nhớ**!

B) 4 tính chất của lập trình hướng đối tượng

1, Tính đóng gói

Ví dụ đơn giản về tính đóng gói: Chúng ta có một ô vuông khép kín. Ở trong đó có 1 dấu chấm. Nếu chúng ta muốn lấy dấu chấm đó ra, chúng ta sẽ dùng hàm getter() để lấy dấu chấm đó là và dùng setter() để cập nhật giá trị.



Cú pháp hàm getter():

```
class Student {
private:
    string name;

public:
    string getName() {   // getter
        return name;
    }
};
```

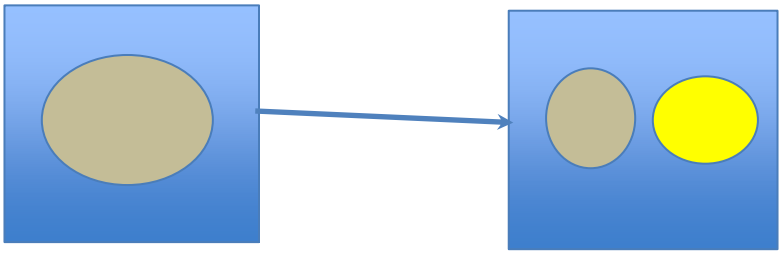
Cú pháp hàm setter():

```
class Student {
private:
    string name;

public:
void setName(string n) {  // setter neu can
        name = n;
    }
};
```

Ghi chú: Nếu bạn là người mới học lập trình hoặc đang học lại từ đầu, thì có thể tạm thời sử dụng **public** cho tất cả các thuộc tính để dễ quan sát và thử nghiệm. Tuy nhiên, khi đã hiểu rõ hơn về lập trình hướng đối tượng, bạn nên sử dụng **private** để đảm bảo an toàn cho dữ liệu – đây là nguyên tắc quan trọng trong **tính đóng gói (Encapsulation)**.

2,Tính kế thừa

Ví dụ về tính kế thừa: Từ tính đóng gói, ta sẽ vẽ thêm ô thứ 2. Sau đó ta sẽ lấy chấm tròn từ ô thứ nhất sang ô thứ 2 và sẽ tô thêm 1 chấm tròn nữa. Đó chính là Tính kế thừa.



Cú pháp để khai báo tính kế thừa:

```
class LopCha {
    // thuoc tính & phuong thuc chung
};

class LopCon : public LopCha {
    // ke thua tu LopCha
};
```

3, Tính đa hình.

Có 2 loại đa hình chính:

Nạp chồng (Overloading): cùng tên, khác tham số (compile time).

Ghi đè (Overriding): lớp con định nghĩa lại phương thức lớp cha (runtime).

Ví dụ về nạp chồng hàm:

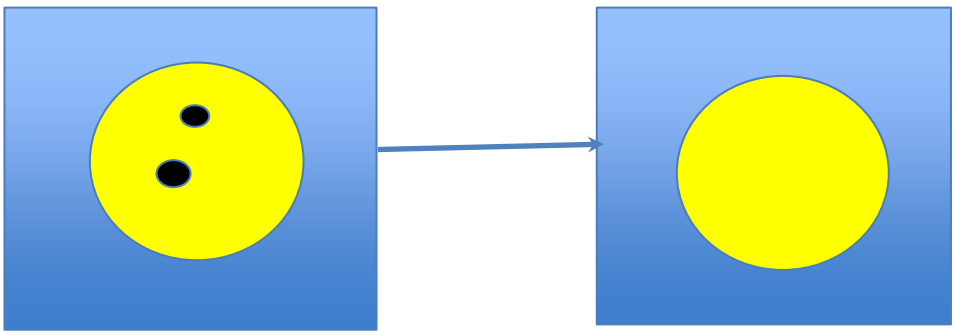
```
class Printer {
public:
    void print(int x) {
        cout << "In so: " << x << endl;
    }

    void print(string s) {
        cout << "In chuoi: " << s << endl;
    }
};
```

Ví dụ về ghi đè:

```
class Animal {
public:
    virtual void speak() {
        cout << "Animal is speaking" << endl;
    }
};

class Dog : public Animal {
public:
    void speak() override {
        cout << "Dog is barking" << endl;
    }
};
```

4, Tính trừu tượng:

Tính trừu tượng: Giúp **ẩn giấu logic phức tạp**, chỉ hiển thị phần cần thiết.



C++ dùng **pure virtual function.**

```
class Animal {
public:
    virtual void makeSound() = 0;  // pure virtual
};

class Dog : public Animal {
public:
    void makeSound() override {
        cout << "Dog says: Woof!" << endl;
    }
};
```

Ví dụ:

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

// Lop truu tuong
class Animal {
public:
    virtual void makeSound() = 0; // hàm thuan ao
};

// Lop con cu the
class Dog : public Animal {
public:
    void makeSound() override {
        cout << "Woof woof!" << endl;
    }
};

int main() {
    Animal* a = new Dog();    // dùng con tro lop cha
    a->makeSound();           // Ket qua: Woof woof!
    delete a;
}
```

C) Các đơn vị kiến thức khác

I) Nạp chồng toán tử:

Operator overloading cho phép bạn định nghĩa lại **cách hoạt động của các toán tử** (+, -, ==, <<, v.v.) với các **kiểu dữ liệu do bạn tự định nghĩa (class)**.

Ví dụ:

+ Khi bạn viết a + b với hai số → máy hiểu ngay.
+ Nhưng với a + b là **hai đối tượng** → bạn phải dạy máy **cách cộng chúng**, bằng cách **nạp chồng toán tử** +.

1. Toán tử + (hàm thành viên)

```
ClassName operator+(const ClassName& other);
```

Ví dụ:

```
class Point {
    int x, y;
public:
    Point(int a = 0, int b = 0): x(a), y(b) {}

    Point operator+(const Point& other) {
        return Point(x + other.x, y + other.y);
    }

    void display() { cout << "(" << x << ", " << y << ")\n"; }
};
```

2. Toán tử so sánh

```
bool operator==(const Point& other) {
    return x == other.x && y == other.y;
}
```

3. Toán tử >> và <<

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

class Point {
    int x, y;
public:
    Point(int a = 0, int b = 0): x(a), y(b) {}

    // friend function ð? n?p ch?ng <<
    friend ostream& operator<<(ostream& os, const Point& p);
};

ostream& operator<<(ostream& os, const Point& p) {
    os << "(" << p.x << ", " << p.y << ")";
    return os;
}
```

II) Hàm bạn , lớp bạn.

Cho phép một hàm hoặc class **truy cập vào thành phần private** của class khác. Thường dùng để nạp chồng toán tử <<, >>.

Cú pháp hàm bạn:

```
friend ostream& operator<<(ostream& os, const MyClass& obj)
```

Cú pháp lớp bạn

```
friend class AnotherClass;
```

III) Biến static

Dùng chung cho tất cả các đối tượng. Không phụ thuộc vào từng instance.

Biến static: dùng để đếm số đối tượng chẳng hạn.

Hàm static: gọi trực tiếp qua class.

```
class A {
public:
    static int count;
    static void showCount() { cout << count; }
};
```

Ví dụ:

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

class Student {
    string name;
    static int count; // dùng chung cho moi doi tuong
public:
    Student(string n) { name = n; count++; }
    static void showCount() {
        cout << "Tong sinh viên: " << count << endl;
    }
};

int Student::count = 0;

int main() {
    Student a("An"), b("Binh");
    Student::showCount();
}
```

D) Phần Main()

I) Khai báo một đối tượng

1) Tính đóng gói

a) Cấu trúc

```
TenClass obj;              // goi constructor không tham so
TenClass obj(thamso);      // goi constructor có tham s?

obj.setX(...);             // goi setter de gán giá tri?
cout << obj.getX();        // goi getter de lay giá tri?
```

b) Ví dụ:

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

class NhanVien {
private:
    string ten;
    int tuoi;
public:
    void setTen(string t) { ten = t; }
    void setTuoi(int t) { tuoi = t; }
    string getTen() { return ten; }
    int getTuoi() { return tuoi; }
};

int main() {
    NhanVien nv;
    nv.setTen("Nguyen Van A");
    nv.setTuoi(25);

    cout << "Ten: " << nv.getTen() << endl;
    cout << "Tuoi: " << nv.getTuoi() << endl;
    return 0;
}
```

2) Tính kế thừa

a)Cautruc

```
LopCon doiTuong; // lop con ke thua tu lop cha
```

a) Ví dụ:

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

class DongVat {
public:
    void keu() {
        cout << "Đ?ng v?t đang kêu" << endl;
    }
};

class Cho : public DongVat {
public:
    void sua() {
        cout << "Gâu gâu!" << endl;
    }
};

int main() {
    Cho a;
    a.keu();  // Ke thua tu lop cha
    a.sua();  // Hàm riêng cua lop con
    return 0;
}
```

3) Tính trừu tượng:

a)Cấu trúc:

```
LopCha* doiTuong = new LopCon();
doiTuong->phuongThuc(); // Goi phuong thuc ao
```

b)Ví dụ:

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

class DongVat {
public:
    virtual void keu() {
        cout << "Đ?ng v?t đang kêu" << endl;
    }
};

class Meo : public DongVat {
public:
    void keu() override {
        cout << "Meo meo" << endl;
    }
};

class Cho : public DongVat {
public:
    void keu() override {
        cout << "Gâu gâu" << endl;
    }
};

int main() {
    DongVat* a = new Meo();
```

```
    }
};

int main() {
    DongVat* a = new Meo();
    a->keu();  // Meo meo

    DongVat* b = new Cho();
    b->keu();  // Gâu gâu

    delete a;
    delete b;
    return 0;
}
```

4) Tính trừu tượng

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

class HinhHoc {
public:
    virtual double tinhDienTich() = 0;
};

class HinhTron : public HinhHoc {
private:
    double banKinh = 5;
public:
    double tinhDienTich() override {
        return 3.14 * banKinh * banKinh;
    }
};

int main() {
    HinhHoc* h = new HinhTron();
    cout << h->tinhDienTich(); // 78.5
    delete h;
    return 0;
}
```

II) Danh sách đối tượng

1)Khai báo:

Ta sẽ có 3 cách khai báo để có thể tạo danh sách các phần tử:

+Cách 1: Dùng mảng tĩnh

```
Lop lopArr[100]; // Khai báo mang gom 100 dôi tuong kieu Lop

for(int i = 0; i < n; i++) {
    cin >> lopArr[i]; //  nhap tung doi tuong (n phai nho hon 100)
}
```

⚠Cảnh báo: Giới hạn cố định, không thay đổi kích thước sau khi khai báo.

+Cách 2: Dùng mảng động:

```
Lop* lopArr = new Lop[n]; // tao mang dong gom n doi tuong

for(int i = 0; i < n; i++) {
    cin >> lopArr[i]; // nhap tung doi tuong
}

delete[] lopArr; // giai phóng bo nho
```

✓Linh hoạt hơn mảng tĩnh. Tuy nhiên, cần **quản lý bộ nhớ thủ công**.

+Cách 3: Dùng vector

```
#include <vector>
vector<Lop> ds;

for(int i = 0; i < n; i++) {
    Lop temp;
    cin >> temp;
    ds.push_back(temp);
}
```

✓Linh hoạt, tiện dụng, tự động quản lý bộ nhớ.

➕Dễ dàng thêm/xoá phần tử với **.push_back(), .erase()...**

2) for

a) Cấu trúc:

```
for (int i = min; i <= max; ++i) {
    cout << "Nhap doi tuong thu " << i << ": \n";
    ds[i - min].nhap(); // neu mang bat dau tu 0
}
```

b) Lưu ý:

"Khung **for** từ min đến max giúp dễ điều chỉnh số lượng đối tượng cần nhập mà không bị giới hạn cứng. Đây là cách tổ chức linh hoạt, phù hợp với nhiều bài toán quản lý."

E) Bài tập tổng hợp

Viết chương trình quản lý sinh viên với các yêu cầu sau:

-Tạo lớp SinhVien gồm các thuộc tính:

-Mã sinh viên,họ tên, điểm toán, điểm văn, điểm anh.
Các thuộc tính để **private.**

-Viết đầy đủ getter/setter, constructor, destructor
-Tạo hàm tinhDiemTrungBinh() cho sinh viên.
-Viết hàm in ra thông tin sinh viên.
-Tạo lớp SinhVienCNTT kế thừa từ **SinhVien**, thêm điểm lập trình.
-Ghi đè hàm tinhDiemTrungBinh() trong lớp **SinhVienCNTT.**

-Trong hàm main(), thực hiện:
Nhập danh sách sinh viên (tối thiểu 3 sinh viên).
In danh sách sinh viên ra màn hình.

Tìm sinh viên có điểm TB cao nhất.

Sắp xếp sinh viên theo điểm TB giảm dần.

Lời kết:

Giáo trình "Xóa mù OOP" được biên soạn với tâm huyết nhằm giúp các bạn dễ dàng tiếp cận lập trình hướng đối tượng bằng C++ và Java, theo cách đơn giản – dễ hiểu – dễ áp dụng.

Một phần nội dung trong giáo trình có sự hỗ trợ của trí tuệ nhân tạo (AI) để đảm bảo tính hệ thống, khoa học và rõ ràng. Tuy nhiên, toàn bộ cấu trúc, phong cách, ví dụ minh họa, bài tập... đều được thiết kế theo cách riêng – gần gũi với người học Việt Nam, đặc biệt là các bạn mới bắt đầu.

Hy vọng giáo trình này sẽ là người bạn đồng hành hữu ích trên hành trình học lập trình của bạn. Đừng quên rằng, học lập trình không khó – quan trọng là học đúng cách và đúng lộ trình.

Chúc bạn học tốt và thành công!