TÍNH ĐA HÌNH

Khoa Công nghệ phần mềm

Nội dung

1. Phương thức tĩnh
2. Phương thức ảo
3. Tính đa hình
4. Lớp cơ sở trừu tượng
5. Ứng dụng của tính đa hình

Con trỏ trong thừa kế

Con trỏ của lớp cơ sở có thể dùng để chứa địa chỉ đối tượng của lớp dẫn xuất.

- Nghĩa là, có thể gán địa chỉ đối tượng của một lớp dẫn xuất cho con trỏ của lớp cơ sở.

  Ví dụ: class **B**: public **A**{...};

  A *p; **B** b;

  p = &b; //Đúng

- Tuy nhiên, không cho phép gán địa chỉ đối tượng của lớp cơ sở cho con trỏ của lớp dẫn xuất.

  Ví dụ: class **B**: public **A**{...};

  B *p; **A** a;

  p = &a; //Sai

Phương thức trùng tên và sự thừa kế

- Lớp dẫn xuất thừa kế các phương thức public của lớp cơ sở (trừ hàm tạo, hàm hủy và hàm toán tử gán).
- Các lớp có quan hệ thừa kế ***(trực tiếp hoặc gián tiếp)*** có thể có các phương thức có tên, kiểu trả về và danh sách đối số hoàn toàn giống nhau.
- Khi đó, nếu lời gọi xuất phát từ **đối tượng của lớp nào thì phương thức của lớp đó sẽ được gọi**.

  Nếu lớp đó không có phương thức tương ứng thì sẽ gọi phương thức tương ứng của lớp cơ sở gần nhất.

1. Phương thức tĩnh

- Qui tắc gọi phương thức tĩnh
- Sự hạn chế của phương thức tĩnh

1.1 Qui tắc gọi phương thức tĩnh

- Nếu lời gọi xuất phát từ một **đối tượng của lớp nào thì phương thức của lớp đó sẽ được gọi**.
- Nếu lời gọi xuất phát từ một **con trỏ kiểu lớp nào thì phương thức của lớp đó sẽ được gọi** bất kể con trỏ chứa địa chỉ của đối tượng nào (xem Ví dụ 1,2,3).

1.1 Qui tắc gọi phương thức tĩnh – Ví dụ 1

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

```
void main() {
    A *p, *q, *r;
    A a;
    B b;
    C c;
    p = &a;
    q = &b;
    r = &c;
    p->Xuat(); //Gọi tới A::Xuat()
    q->Xuat(); //Gọi tới A::Xuat()
    r->Xuat(); //Gọi tới A::Xuat()
}
```

1.1 Qui tắc gọi phương thức tĩnh – Ví dụ 2

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

```
void HienThi(A *p) { //Hàm toàn cục
    p->Xuat();
}
void main() {
    A a;
    B b;
    C c;
    D d;
    HienThi(&a); //Gọi tới A::Xuat()
    HienThi(&b); //Gọi tới A::Xuat()
    HienThi(&c); //Gọi tới A::Xuat()
    HienThi(&d); //Gọi tới A::Xuat()
}
```

1.1 Qui tắc gọi phương thức tĩnh – Ví dụ 3

```
class TS {
private:
    char hoten[50];
public:
    void Nhap() {…}
    void In() { cout << hoten; }
    void Xem_In() {
        if (toupper(getch()) == 'C')
            this->In(); //Gọi hàm TS::In()
    }
}; //end of class TS
class TS2 : public TS {
private:
    char diachi[100];
public:
    void In() {
        TS::In();
        cout << diachi;
    }
    void Nhap() {
        TS::Nhap();
        gets_s(diachi);
    }
}; //end of class TS2
void main() {
    TS2 t[100];
    int i, n;
    cin >> n; //Nhập số thí sinh
    for(i=0; i < n; ++i)
        t[i].Nhap(); //Gọi hàm TS2::Nhap()
    for(i=0; i < n; ++i)
        t[i].Xem_In(); //Gọi hàm TS::Xem_In()
}
```

1.2 Sự hạn chế của phương thức tĩnh

- Lời gọi phương thức tĩnh bao giờ cũng **xác định rõ phương thức nào được gọi** trong số các phương thức trùng tên của các lớp có quan hệ thừa kế.
- Nếu lời gọi xuất phát từ một con trỏ kiểu lớp nào thì phương thức của lớp đó sẽ được gọi bất kể con trỏ chứa địa chỉ của đối tượng nào.

  Điều chúng ta muốn là con trỏ đang trỏ tới đối tượng của lớp nào thì phương thức của lớp đó sẽ được gọi.

2. Phương thức ảo

- Định nghĩa phương thức ảo
- Qui tắc gọi phương thức ảo
- Liên kết động
- Ứng dụng của phương thức ảo

2.1 Định nghĩa phương thức ảo

- **Cách 1:**

  Thêm từ khóa virtual vào dòng tiêu đề của phương thức trùng tên bên trong định nghĩa của lớp cơ sở.

- **Cách 2:**

  Thêm từ khóa virtual vào dòng tiêu đề của phương thức trùng tên bên trong định nghĩa của lớp cơ sở và tất cả lớp dẫn xuất.

Lưu ý: Nếu phương thức được cài đặt bên ngoài định nghĩa lớp thì từ khóa virtual chỉ xuất hiện ở prototype của phương thức (đặt bên trong định nghĩa lớp).

2.1 Định nghĩa phương thức ảo – Ví dụ

Cách 1:

```
class A {
public:
    virtual void Xuat() { cout << "\n Lop A"; }
};
class B : public A {
public:
    void Xuat() { cout << "\n Lop B"; }
};
class C : public B {
public:
    void Xuat() { cout << "\n Lop C"; }
};
class D : public C {
public:
    void Xuat() { cout << "\n Lop D"; }
};
```

Cách 2:

```
class A {
public:
    virtual void Xuat() { cout << "\n Lop A"; }
};
class B : public A {
public:
    virtual void Xuat() { cout << "\n Lop B"; }
};
class C : public B {
public:
    virtual void Xuat() { cout << "\n Lop C"; }
};
class D : public C {
public:
    virtual void Xuat() { cout << "\n Lop D"; }
};
```

2.2 Qui tắc gọi phương thức ảo

- Phương thức ảo chỉ khác phương thức tĩnh **khi được gọi từ một con trỏ.**
- Lời gọi phương thức ảo từ một con trỏ **chưa cho biết rõ phương thức nào sẽ được gọi** trong số các phương thức trùng tên của các lớp có quan hệ thừa kế.
- Phương thức được gọi 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.

2.2 Qui tắc gọi phương thức ảo – Ví dụ 1

```
class A {
public:
    virtual void Xuat() { cout << "\n Lop A"; }
};
class B : public A {
public:
    void Xuat() { cout << "\n Lop B"; }
};
class C : public B {
public:
    void Xuat() { cout << "\n Lop C"; }
};
class D : public C {
public:
    void Xuat() { cout << "\n Lop D"; }
};
```

```
void main() {
    A *p; //p là con trỏ kiểu A
    A a;
    B b;
    C c;
    D d;
    p = &a;
    p->Xuat(); //Gọi tới A::Xuat()
    p = &b;
    q->Xuat(); //Gọi tới B::Xuat()
    p = &c;
    p->Xuat(); //Gọi tới C::Xuat()
    p = &d;
    p->Xuat(); //Gọi tới D::Xuat()
}
```

2.2 Qui tắc gọi phương thức ảo – Ví dụ 2

```
class A {
public:
    virtual void Xuat() { cout << "\n Lop A"; }
};
class B : public A {
public:
    void Xuat() { cout << "\n Lop B"; }
};
class C : public A {
public:
    void Xuat() { cout << "\n Lop C"; }
};
class D : public C {
public:
    void Xuat() { cout << "\n Lop D"; }
};
```

```
void HienThi(A *p) { //Hàm toàn cục
    p->Xuat();
}
void main() {
    A a;
    B b;
    C c;
    D d;
    HienThi(&a); //Gọi tới A::Xuat()
    HienThi(&b); //Gọi tới B::Xuat()
    HienThi(&c); //Gọi tới C::Xuat()
    HienThi(&d); //Gọi tới D::Xuat()
}
```

2.3 Liên kết động

- Lời gọi tới phương thức tĩnh (xuất phát từ con trỏ) luôn luôn liên kết với một phương thức cố định.

  Sự liên kết này được xác định trong quá trình biên dịch chương trình.

- Lời gọi tới phương thức ảo (xuất phát từ con trỏ) không liên kết với một phương thức cố định mà tùy thuộc vào nội dung con trỏ.

  Phương thức mà nó liên kết chưa xác định trong giai đoạn dịch chương trình.

  Phương thức được liên kết thay đổi mỗi khi có sự thay đổi nội dung con trỏ trong quá trình chạy chương trình.

  Đây chính là sự liên kết động.

2.3 Liên kết động (tt)

- Điều kiện để liên kết động là phương thức ảo phải được định nghĩa thống nhất từ lớp cơ sở cho đến các lớp dẫn xuất:
  - Trùng kiểu trả về;
  - Trùng tên phương thức;
  - Trùng danh sách đối.

2.4 Ứng dụng của phương thức ảo

- Có thể dùng phương thức ảo để giải quyết bài toán quản lý một danh sách các đối tượng có kiểu khác nhau.

  Khi đó, tuỳ thuộc vào kiểu của từng đối tượng mà phương thức tương ứng sẽ được gọi.
- Phương thức ảo làm cho việc bổ sung nâng cấp chương trình dễ dàng hơn: thêm các lớp dẫn xuất và cài đặt lại phương thức ảo của lớp cơ sở mà không cần phải sửa đổi trên lớp cơ sở (đảm bảo tính đóng gói).

2.4 Ứng dụng của phương thức ảo – Ví dụ

```
class Nguoi {
protected:
    char *HoTen;
    int NamSinh;
public:
    Nguoi(char *ht, int ns):NamSinh(ns){ HoTen = _strdup(ht); }
    ~Nguoi() {delete [ ] HoTen;}
    void An() const { cout << "Nguoi::An()";}
    void Xuat() const {
            cout << "Nguoi, ho ten: " << HoTen;
            cout << “, sinh “ << NamSinh; }
};
```

2.4 Ứng dụng của phương thức ảo – Ví dụ (tt)

```
class SinhVien : public Nguoi{
protected:
    char *MaSo;
public:
    SinhVien(char *ht, int ns, char *ms) : Nguoi(ht,ns) {
        MaSo = _strdup(ms);
    }
    ~SinhVien() { delete [ ] MaSo;}
    void Xuat() const {
        cout << "Sinh vien " << HoTen << ", ma so " << MaSo;
    }
};
```

2.4 Ứng dụng của phương thức ảo – Ví dụ (tt)

```
class NuSinh : public SinhVien
{
  public:
      NuSinh( char *ht, int ns, char *ms) : SinhVien(ht,ns,ms) { }
      void An() const
      {
          cout << "NuSinh::An()";
      }
};
```

2.4 Ứng dụng của phương thức ảo – Ví dụ (tt)

```
class CongNhan : public Nguoi{
protected:
    double MucLuong;
public:
    CongNhan( char *ht, int ns, double ml) : Nguoi(ht,ns),
                                             MucLuong(ml){ }
    void Xuat() const {
          cout << "Cong nhan, ten " << HoTen;
          cout << " muc luong: " << MucLuong;
    }
};
```

2.4 Ứng dụng của phương thức ảo – Ví dụ (tt)

```
void XuatDS(int n, Nguoi *ds[ ]){ //Hàm toàn cục, có đối là con trỏ kiểu Nguoi
    for (int i = 0; i < n; i++)
    {
        ds[i]->Xuat();
        cout << "\n";
    }
}
const int N = 4;
void main(){
    Nguoi *a[N];
    a[0] = new SinhVien("Vien Van Sinh", 1982, "200001234");
    a[1] = new NuSinh("Le Thi Ha Dong", 1984, "200001235");
    a[2] = new CongNhan("Tran Nhan Cong", 1984, 1000000);
    a[3] = new Nguoi("Nguyen Thanh Nhan", 1960);
    XuatDS(4,a);
}
```

```
Nguoi, ho ten: Vien Van Sinh, sinh 1982
Nguoi, ho ten: Le Thi Ha Dong, sinh 1984
Nguoi, ho ten: Tran Nhan Cong, sinh 1984
Nguoi, ho ten: Nguyen Thanh Nhan, sinh 1960
```

Nhận xét

Ta thấy muốn xuất dữ liệu tương ứng với đối tượng ta phải có cách nhận diện đối tượng cần xuất.

- **Cách 1:** thêm 1 thuộc tính phân loại vào lớp cơ sở để nhận diện đối tượng, thuộc tính này được khai báo là public

  Cách làm này có một số nhược điểm: mã lệnh dài dòng; dễ sai sót và khó sửa chữa; khó nâng cấp, bảo trì.
- **Cách 2:** khai báo phương thức xuất của lớp cơ sở là phương thức ảo, các lớp dẫn xuất sẽ cài đặt lại phương thức này và chúng sẽ được gọi thông qua con trỏ.

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 1

```
class Nguoi{
public:
    enum LOAI {NGUOI, SV, CN};
protected:
    char *HoTen;            int NamSinh;
public:
    LOAI pl;
    Nguoi(char *ht, int ns):NamSinh(ns),pl(NGUOI) {
        HoTen = _strdup(ht); }
    ~Nguoi() { delete [] HoTen; }
    void An() const { cout << “Nguoi::An()"; }
    void Xuat() const {
        cout << "Nguoi, ho ten: " << HoTen << “, sinh " << NamSinh;
    }
};
```

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 1 (tt)

```
class SinhVien : public Nguoi{
protected:
    char *MaSo;
public:
    SinhVien(char *ht, int ns, char *ms) : Nguoi(ht,ns) {
        MaSo = _strdup(ms);
        pl = SV;
    }
    ~SinhVien() { delete [ ] MaSo; }
    void Xuat() const {
        cout << "Sinh vien “ << HoTen << ", ma so " << MaSo;
    }
};
```

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 1 (tt)

```
class NuSinh : public SinhVien
{
  public:
      NuSinh( char *ht, int ns, char *ms) : SinhVien(ht,ns,ms) {
          pl = SV;
      }
      void An() const
      {
          cout << "NuSinh::An()";
      }
};
```

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 1 (tt)

```
class CongNhan : public Nguoi{
protected:
    double MucLuong;
public:
    CongNhan( char *ht, int ns, double ml) : Nguoi(ht,ns),
    MucLuong(ml){
        pl = CN;
    }
    void Xuat() const{
        cout << "Cong nhan " << HoTen;
        cout << “, muc luong " << MucLuong;
    }
};
```

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 1 (tt)

```
void XuatDS(int n, Nguoi *ds[ ]) { //phải thay đổi hàm này mỗi khi có thêm lớp mới
    for (int i = 0; i < n; i++){
        switch(ds[i]->pl){
            case Nguoi::SV:
                    ((SinhVien*)ds[i])→Xuat(); //Gọi hàm SinhVien::Xuat()
                    break;
            case Nguoi::CN:
                    ((CongNhan*)ds[i])→Xuat(); //Gọi hàm CongNhan::Xuat()
                    break;
            default:
                    ds[i]->Xuat(); //Gọi hàm Nguoi::Xuat()
            }
            cout << "\n";
    } //end of for
} //end of XuatDS
```

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 1 (tt)

```
const int N = 4;
void main(){
    Nguoi *a[N];
    a[0] = new SinhVien("Vien Van Sinh", 1982, "200001234");
    a[1] = new NuSinh("Le Thi Ha Dong", 1984, "200001235");
    a[2] = new CongNhan("Tran Nhan Cong", 1984, 1000000);
    a[3] = new Nguoi("Nguyen Thanh Nhan", 1960);
    XuatDS(4,a);
}
```

```
Sinh vien Vien Van Sinh, ma so 200001234
Sinh vien Le Thi Ha Dong, ma so 200001235
Cong nhan Tran Nhan Cong, muc luong 1000000
Nguoi, ho ten: Nguyen Thanh Nhan, sinh 1960
```

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 2

```
class Nguoi {
protected:
    char *HoTen;
    int NamSinh;
public:
    Nguoi( char *ht,int ns):NamSinh(ns){HoTen = _strdup(ht);}
    ~Nguoi() {delete [ ] HoTen;}
    void An() const { cout << “Nguoi::An()"; }
    virtual void Xuat() const {
            cout << "Nguoi, ho ten: " << HoTen;
            cout << “, sinh " << NamSinh;
    }
};
```

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 2 (tt)

```
class CaSi : public Nguoi{
protected:
    double CatXe;
public:
    CaSi( char *ht, int ns, double cx):Nguoi(ht,ns),CatXe(cx) { }
    void Xuat() const {
        cout << “Ca si “ << HoTen << “, co cat xe “ << CatXe;
    }
};
```

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 2 (tt)

```
void XuatDS( int n, Nguoi *ds[]){
    for ( int i = 0; i < n; i++){
        ds[i]->Xuat();
        cout << "\n";
    }
}
```

=> Cách này thì không cần thay đổi hàm XuatDS nhưng nó vẫn hoạt động đúng như mong muốn, kể cả khi ds[i] trỏ tới một đối tượng thuộc lớp mới là CaSi

2.4 Ứng dụng của phương thức ảo
Ví dụ – Cách 2 (tt)

```
const int N = 4;
void main(){
    Nguoi *a[N];
    a[0] = new SinhVien("Vien Van Sinh", 1982, "200001234");
    a[1] = new NuSinh("Le Thi Ha Dong", 1984, "200001235");
    a[2] = new CongNhan("Tran Nhan Cong", 1984, 1000000);
    a[3] = new CaSi("Ngoc Lan", 1955, 3000000);
    XuatDS(4,a);
    for ( int i = 0; i < 4; i++)
        delete a[i];
    XuatDS(4,a);
    system("pause");
}
```

```
Sinh vien Vien Van Sinh, ma so 200001234
Sinh vien Le Thi Ha Dong, ma so 200001235
Cong nhan Tran Nhan Cong, muc luong 1000000
Ca si Ngoc Lan, co cat xe 3000000
```

3. Tính đa hình

- Ta thấy cùng một câu lệnh (ví dụ p->Xuat()) tương ứng với nhiều phương thức khác nhau.
- Đây chính là **tương ứng bội hay tính đa hình** trong lập trình hướng đối tượng.
- Các phương thức thể hiện tính đa hình phải được định nghĩa là các phương thức ảo trong lớp cơ sở.

  Các phương thức này sẽ được các lớp dẫn xuất cài đặt lại theo cách riêng của mình.

3. Tính đa hình (tt)

- Trong quan hệ thừa kế, lớp cơ sở được coi là lớp tổng quát hóa của các lớp dẫn xuất và các lớp dẫn xuất là chi tiết hóa của lớp cơ sở.
- Như vậy lớp cơ sở sẽ chứa những phương thức tổng quát cho mọi lớp dẫn xuất và các phương thức này sẽ được cài đặt chi tiết ở các lớp dẫn xuất, tùy thuộc vào nội dung và tính chất của từng lớp dẫn xuất.

Ví dụ: Lớp cơ sở hình đa giác sẽ có phương thức tính diện tích của nó, nhưng nội dung tính diện tích như thế nào sẽ được xác định ở các lớp dẫn xuất là hình tam giác, hình tứ giác, …

4. Lớp cơ sở trừu tượng

- Khái niệm
- Phương thức thuần ảo
- Lớp trừu tượng và phương thức thuần ảo

4.1 Khái niệm

- Lớp cơ sở trừu tượng (còn gọi là lớp trừu tượng) là lớp chỉ được dùng làm cơ sở cho các lớp khác.
- Lớp trừu tượng được dùng để định nghĩa một số khái niệm tổng quát, chung cho các lớp khác.

Ví dụ: Lớp trừu tượng Shape dùng làm lớp cơ sở cho các lớp Circle, Rectangle, Square, Triangle.

4.1 Khái niệm (tt)

- Lớp trừu tượng không được dùng để khai báo đối tượng bởi vì không có đối tượng nào của một lớp trừu tượng có thể được phát sinh.

  Ví dụ: Shape s;

  -> Báo lỗi: “Cannot create instance of abstract class”

- Tuy nhiên biến con trỏ và biến tham chiếu kiểu lớp trừu tượng vẫn được chấp nhận.

  Ví dụ: Shape *s; //Đúng

4.2 Phương thức thuần ảo

- **Cách định nghĩa phương thức thuần ảo:**

  virtual void **Tên_Phương_Thức() = 0;**

  //Gán = 0 thay cho việc cài đặt phương thức

- Lưu ý là phương thức thuần ảo khác với phương thức ảo có thân rỗng:

  virtual void **Tên_Phương_Thức(){ }**

  //Phương thức này vẫn là phương thức ảo chứ không phải là phương thức thuần ảo

4.3 Lớp trừu tượng và Phương thức thuần ảo

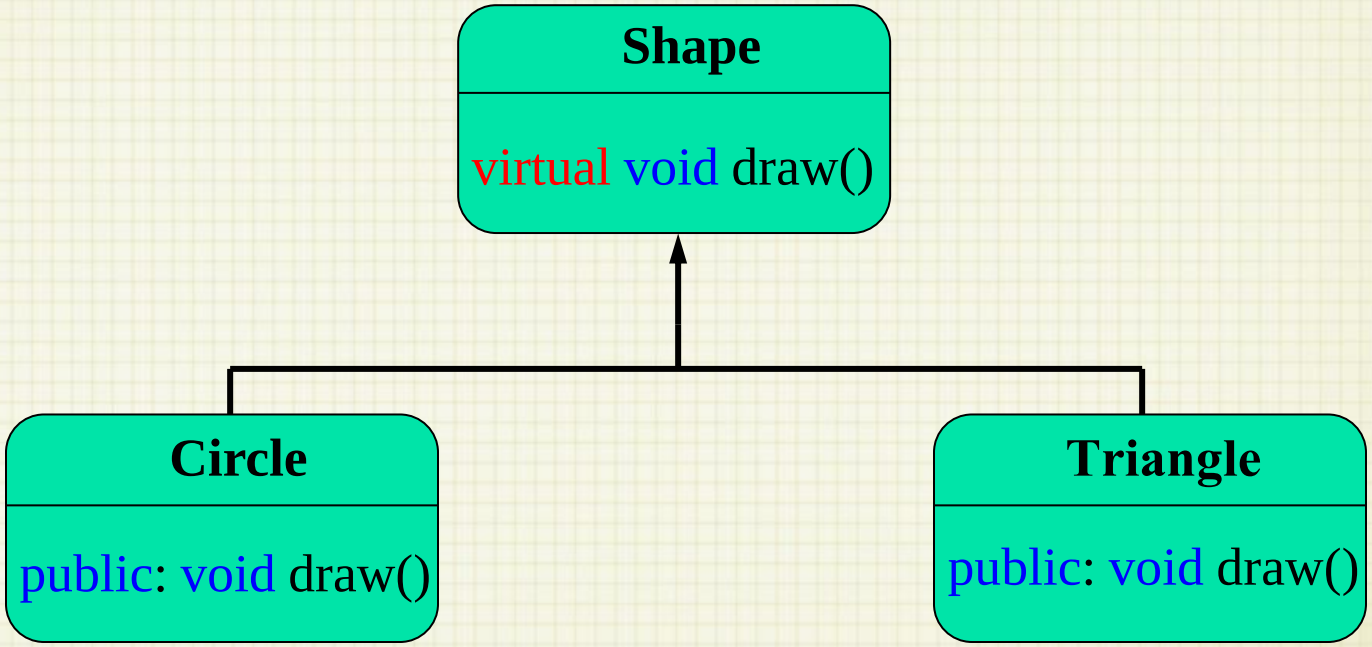
- Các phương thức thuần ảo thường được chứa trong các lớp trừu tượng.
- Theo quan điểm chung về cách thức sử dụng thì một lớp trừu tượng không nhất thiết phải có chứa phương thức thuần ảo.
- Tuy nhiên, theo quan điểm của C++ thì lớp trừu tượng bắt buộc phải có chứa ít nhất một phương thức thuần ảo (phương thức ảo = 0;).

  Nếu không lớp đó vẫn không phải là lớp trừu tượng (mặc dù về mặt ý nghĩa nó là lớp trừu tượng).

4.3 Lớp trừu tượng và Phương thức thuần ảo (tt)

- Bất kỳ lớp nào dẫn xuất từ lớp trừu tượng phải định nghĩa lại tất cả các phương thức thuần ảo của lớp trừu tượng:
  - Hoặc tiếp tục là phương thức thuần ảo;
  - Hoặc được cài đặt chi tiết trong lớp dẫn xuất.
- Vì vậy, bản thân các lớp dẫn xuất của lớp trừu tượng cũng có thể là lớp trừu tượng.
- Phương thức thuần ảo có ý nghĩa trong việc tổ chức phân cấp các lớp: phương thức thuần ảo trong lớp cơ sở sẽ là phương thức chung cho các lớp dẫn xuất thừa kế và cài đặt.

4. Lớp cơ sở trừu tượng – Ví dụ



4. Lớp cơ sở trừu tượng – Ví dụ (tt)

```
class Shape    //Lớp trừu tượng
{
   public:
          virtual void draw() = 0; //Hàm thuần ảo
};
class Circle : public Shape {
     public:
          void draw(){ cout << “I am a circle” << endl; }
};
class Triangle : public Shape {
     public:
          void draw(){ cout << “I am a triangle” << endl; }
};
void main(){
     Shape *s;     Triangle t;     Circle c;
     s = &t;   s->draw();
     s = &c;  s->draw();
     system“(“pause”);
}
```

5. Ứng dụng của tính đa hình

1) **Tạo ra sự linh hoạt trong việc sử dụng tính thừa kế để nâng cấp, phát triển chương trình.**

Khi thêm mới một lớp dẫn xuất, lớp này sẽ thừa kế **các phương thức ảo** của lớp cơ sở và cài đặt lại các phương thức này theo cách riêng của nó mà không cần phải sửa đổi trên lớp cơ sở (đảm bảo tính đóng gói) và do đó không làm ảnh hưởng đến các lớp dẫn xuất khác.

5. Ứng dụng của tính đa hình (tt)

2) **Tính đa hình cho phép giải quyết vấn đề trên các đối tượng khác nhau theo cùng một lược đồ chung.**

CÁC BƯỚC ÁP DỤNG TÍNH ĐA HÌNH:

B1: Xây dựng lớp cơ sở A có chứa:

+ Các thuộc tính chung nhất của các thực thể cần quản lý;

+ Các phương thức ảo/thuần ảo để thực hiện các thao tác chung của các lớp dẫn xuất.

B2: Xây dựng các lớp dẫn xuất thừa kế lớp A để mô tả các đối tượng cụ thể cần quản lý.

B3: Cài đặt các phương thức ảo/thuần ảo trong B1 ở các lớp dẫn xuất (đã xây dựng trong B2).

5. Ứng dụng của tính đa hình (tt)

CÁC BƯỚC ÁP DỤNG TÍNH ĐA HÌNH (tt):

B4: Xây dựng lớp quản lý các đối tượng DS_A có chứa:

+ Thuộc tính là 1 dãy con trỏ kiểu lớp cơ sở **A,** khai báo:

A **h; hoặc **A *h[n];**

Các con trỏ này sẽ chứa địa chỉ đối tượng của các lớp dẫn xuất, tùy theo kiểu của đối tượng mà con trỏ đang trỏ tới để gọi phương thức của lớp dẫn xuất tương ứng.

+ Các phương thức tương ứng với các yêu cầu của bài toán.

5. Ứng dụng của tính đa hình – Ví dụ 1

Bài toán: Giả sử có 20 chuồng, mỗi chuồng có thể nuôi 1 con chó hoặc 1 con mèo.

Xây dựng chương trình gồm các chức năng:

1) Nhập một con vật mới mua vào chuồng rỗng đầu tiên.
2) Xuất bán một con vật.
3) Thống kê các con vật đang nuôi trong 20 chuồng.

Cách giải: Áp dụng các bước trong 5.2) để giải bài toán trên.

5. Ứng dụng của tính đa hình – Ví dụ 1 (tt)

B1: Xây dựng lớp cơ sở CONVAT có chứa:

+ Thuộc tính chung của các thực thể cần quản lý:

```
char * ten;
```

+ Phương thức ảo thực hiện thao tác chung của các lớp dẫn xuất:

```
virtual void xungten()
```

```
class CONVAT {
protected:
    char *ten;
public:
    CONVAT() {
        ten = NULL;
    }
    CONVAT(char *t) {
        ten = _strdup(t);
    }
    virtual void xungten();
};
```

5. Ứng dụng của tính đa hình – Ví dụ 1 (tt)

B2: Xây dựng các lớp dẫn xuất thừa kế lớp CONVAT để mô tả các đối tượng cụ thể cần quản lý:

class MEO : public CONVAT

class CHO : public CONVAT

B3: Cài đặt phương thức ảo trong B1 ở các lớp dẫn xuất.

```
class MEO : public CONVAT {
public:
        MEO(): CONVAT(){ }
        MEO(char *ten): CONVAT(ten){ }
        void xungten() {
                cout << "\nToi la chu meo " << ten;
        }
};
class CHO : public CONVAT {
public:
        CHO() : CONVAT() { }
        CHO(char *ten) : CONVAT(ten) { }
        void xungten() {
                cout << "\nToi la chu cho " << ten;
        }
};
```

5. Ứng dụng của tính đa hình – Ví dụ 1 (tt)

B4: Xây dựng lớp quản lý các đối tượng DS_CONVAT có chứa:

+ Thuộc tính là 1 dãy con trỏ kiểu lớp cơ sở **CONVAT**

CONVAT **h;

+ Các phương thức tương ứng với các yêu cầu của bài toán:

```
int nhap(CONVAT *c);
CONVAT* xuat(int n);
void thongke();
```

```
class DS_CONVAT {
private:
        int soconvat_hienco;
        int soconvat_toida;
        CONVAT **ds;
public:
        DS_CONVAT(int max) {
                soconvat_hienco = 0;
                soconvat_toida = max;
                ds = new CONVAT*[max];
                for (int i = 0; i < max; ++i)
                        ds[i] = NULL;
        }
        int nhap(CONVAT *c);
        CONVAT* xuat(int n);
        void thongke();
};
```

5. Ứng dụng của tính đa hình (tt)

3) **Sử dụng tính đa hình để tổ chức thực hiện các thuật toán khác nhau trên cùng một bài toán**, trong đó:

+ Lớp cơ sở sẽ chứa dữ liệu của bài toán và 1 phương thức ảo.

+ Mỗi lớp dẫn xuất sẽ cài đặt phương thức ảo của lớp cơ sở bằng một thuật toán cụ thể.

+ Sử dụng một mảng con trỏ kiểu lớp cơ sở và gán cho mỗi phần tử mảng địa chỉ của một đối tượng thuộc lớp dẫn xuất.

5. Ứng dụng của tính đa hình – Ví dụ 2

Bài toán: Sắp xếp một dãy số nguyên theo thứ tự tăng dần bằng cách dùng các thuật toán Select_Sort, Quick_Sort và Heap_Sort.

Cách giải: Áp dụng các bước trong 5.3) để giải bài toán trên.

5. Ứng dụng của tính đa hình – Ví dụ 2 (tt)

B1: Xây dựng lớp cơ sở SORT

có chứa:

+ Dữ liệu của bài toán:

```
int *a;
```

+ Phương thức sắp xếp ảo:

```
virtual void sapxep(int *aa,int n);
```

```
class SORT {
protected:
    int *a;
    void hoanvi(int &i, int &j);
    //Sử dụng trong các lớp dẫn xuất
public:
    virtual void sapxep(int *aa, long n){
        a = aa;
    }
};
```

5. Ứng dụng của tính đa hình – Ví dụ 2 (tt)

B2: Xây dựng các lớp dẫn xuất tương ứng với các thuật toán sắp xếp (thừa kế lớp SORT).

B3: Mỗi lớp dẫn xuất sẽ cài đặt phương thức ảo của lớp cơ sở bằng một thuật toán cụ thể.

```
class SELECT_SORT : public SORT {
public:
    virtual void sapxep(int *aa, long n);
};
class QUICK_SORT : public SORT {
public:
    virtual void sapxep(int *aa, long n);
};
class HEAP_SORT : public SORT {
public:
    virtual void sapxep(int *aa, long n);
};
```

5. Ứng dụng của tính đa hình – Ví dụ 2 (tt)

B4: Sử dụng một mảng con trỏ kiểu lớp cơ sở và gán cho mỗi phần tử mảng địa chỉ của một đối tượng thuộc lớp dẫn xuất.

Tùy thuộc vào kiểu của đối tượng mà con trỏ đang trỏ tới để gọi phương thức sapxep của lớp tương ứng.

```
void main(){
        int *a,
        long n;
        SORT *s[3];
        SELECT_SORT ss;
        QUICK_SORT qs;
        HEAP_SORT hs;
        s[0] = &ss;
        s[1] = &qs;
        s[3] = &hs;
        for (int i = 0; i < 3; ++i) {
                s[i]->sapxep(a, n);
        }
}
```

Bài tập

Công ty ABC là công ty sản xuất kinh doanh thú nhồi bông. Công ty có nhiều nhân viên làm việc, mỗi nhân viên thuộc một trong ba bộ phận: bộ phận quản lý, bộ phận văn phòng, bộ phận sản xuất. Mỗi bộ phận có cách tính tiền lương khác nhau, cụ thể:

- **Bộ phận quản lý:**

  Lương = Lương Cơ Bản (LCB) * Hệ số chức vụ + Thưởng

- **Bộ phận văn phòng:**

  Lương = LCB + Số ngày làm việc * 200.000 + Trợ Cấp

- **Bộ phận sản xuất:**

  Lương = LCB + Số Sản Phẩm * 2.000

Bài tập (tt)

Công ty cần quản lý các thông tin về nhân viên của mình như: họ tên, ngày sinh và bộ phận mà nhân viên làm việc để tính lương cho từng nhân viên trong công ty.

Bài tập (tt)

Yêu cầu: Thiết kế các lớp thích hợp để thực hiện các công việc sau:

- Nhập thông tin của các nhân viên để phục vụ cho việc tính lương.
- Tính lương cho từng nhân viên.
- Xuất thông tin của các nhân viên.
- Tính tổng lương của công ty.
- Tìm kiếm một nhân viên theo họ tên.

Cách giải: Áp dụng các bước trong 5.2) để giải bài toán trên.

5. Ứng dụng của tính đa hình – Ví dụ 3

B1: Xây dựng lớp cơ sở NHANVIEN

có chứa:

+ Thuộc tính chung của các thực thể cần quản lý:

```
char hoten[50];
char ngaysinh[20];
char diachi[100];
```

+ Phương thức thuần ảo thực hiện các thao tác chung của các lớp dẫn xuất:

```
virtual void Nhap() = 0;
virtual void Xuat() = 0;
virtual double TinhLuong() = 0;
```

```
class NHANVIEN {
private:
    char hoten[50];
    char ngaysinh[20];
    char diachi[100];
    double luongcoban;
public:
    void NhapThongTinCaNhan();
    void XuatThongTinCaNhan();
    virtual void Nhap() = 0;
    virtual void Xuat() = 0;
    virtual double TinhLuong() = 0;
};
```

5. Ứng dụng của tính đa hình – Ví dụ 3 (tt)

B2: Xây dựng các lớp dẫn xuất thừa kế lớp NHANVIEN để mô tả các đối tượng cụ thể cần quản lý.

B3: Cài đặt phương thức ảo trong B1 ở các lớp dẫn xuất.

```
class NVQUANLY : public NHANVIEN{
private:
     float hesochucvu;
     double thuong;
public:
    void Nhap();
    void Xuat();
    double TinhLuong();
};
```

```
class NVVANPHONG : public NHANVIEN {
private:
     int songaylamviec;
     double trocap;
public:
     void Nhap();
     void Xuat();
     double TinhLuong();
};
class NVSANXUAT : public NHANVIEN {
private:
     int sosanpham;
public:
     void Nhap();
     void Xuat();
     double TinhLuong();
};
```

5. Ứng dụng của tính đa hình – Ví dụ 3 (tt)

B4: Xây dựng lớp quản lý các đối tượng DS_CONVAT có chứa:

+ Thuộc tính là 1 dãy con trỏ kiểu lớp cơ sở **NHANVIEN**

+ Các phương thức tương ứng với các yêu cầu của bài toán.

```
class DS_NHANVIEN {
private:
        NHANVIEN *ds[MAXNV];
        int soluongNV;
public:
        void NhapDSNhanVien();
        void XuatDSNhanVien();
        void XuatBangLuong();
        double TinhTongLuong();
        NHANVIEN * TimNhanVien(char *hoten);
};
```

Q & A