

## Tuần 06: Polymorphism

(Cảm ơn thầy Trần Ngọc Đạt Thành đã cung cấp tài liệu tham khảo cho bài thực hành)

### I. Tính chất đa hình trong lập trình hướng đối tượng

#### a. Liên kết tĩnh, liên kết động và phương thức ảo

Trong quan hệ kế thừa, bên cạnh việc tạo trực tiếp một đối tượng cụ thể, ta có thể sử dụng biến con trở để lưu trữ đối tượng cần tạo. Cơ chế trong quan hệ kế thừa cho phép tạo một **biến con trở có kiểu dữ liệu là lớp cha, trỏ đến đối tượng kiểu lớp con.**

- Xem ví dụ sau, ta có quan hệ giữa heo, mèo và động vật là quan hệ kế thừa, trong đó con heo, con mèo kế thừa từ lớp động vật.

```
class DongVat
{
    //Các thuộc tính...
public:
    void TiengKeu()
    {
        cout << "Chua biet dong vat gi!!" << endl;
    }
};

class ConHeo : public DongVat
{
    //Các thuộc tính riêng của con heo
public:
    void TiengKeu()
    {
        cout << "Ut Ut!!" << endl;
    }
};

class ConMeo : public DongVat
{
    //Các thuộc tính riêng của con mèo
public:
    void TiengKeu()
    {
        cout << "Meo meo!!" << endl;
    }
};
```

```

void main()
{
    ConMeo conmeo1;
    DongVat* conmeo2;
    //Con trỏ có kiểu của lớp cha ĐƯỢC
    //tham chiếu đến đối tượng lớp con
    conmeo2 = &conmeo1;
    conmeo2->TiengKeu();

    //Con trỏ có kiểu của lớp cha ĐƯỢC
    //tham chiếu đến đối tượng lớp con
    DongVat* conheo1 = new ConHeo();
    conheo1->TiengKeu();

    delete conheo1;
}

```

```

Chua biet dong vat gi!!
Chua biet dong vat gi!!
Press any key to continue . . .

```

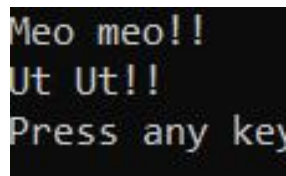
- ✚ Dựa trên tính chất cho phép trỏ đến những lớp con trong quan hệ kế thừa, ta có thể sử dụng nạp chồng hàm (function overloading) để cài đặt mã nguồn thao tác trên đối tượng của lớp cha vẫn có thể chạy được cho đối tượng của lớp con, tuy nhiên cần kết hợp với tính đa hình.
  - Lưu ý : chỉ cho phép biến con trỏ có kiểu dữ liệu lớp cha trỏ đến đối tượng lớp con chứ không được phép làm ngược lại : **biến con trỏ có kiểu dữ liệu lớp con không được phép trỏ đến đối tượng lớp cha**
- ✚ Trong ví dụ trên, ta có thể thấy khi biến con trỏ mang kiểu dữ liệu của lớp cha (lớp Động Vật) trỏ đến đối tượng lớp con (con mèo, con heo), khi gọi phương thức nạp chồng (tiếng kêu) thì đều gọi lại phương thức này ở lớp cha !!! Đây chính là **liên kết tĩnh**.
- ✚ Để có thể gọi được đúng phương thức của đối tượng mà con trỏ đang trỏ đến, cần phải cài đặt **cơ chế liên kết động cho phương thức ở lớp cha**, bằng cách thêm từ khóa **virtual**
  - Phương thức nào được cài đặt liên kết động bằng cách thêm từ khóa virtual được gọi là **phương thức ảo hay hàm ảo**
- ✚ Phương thức ảo hay hàm ảo mang **tính chất đa hình** :
  - Sử dụng con trỏ có kiểu thuộc lớp cha :
    - Trỏ đến đối tượng thuộc lớp con nào thì gọi hàm hay phương thức của lớp con đó tương ứng
- ✚ Để cài đặt một phương thức hay thuộc tính mang tính đa hình :
  - Cần phải có quan hệ kế thừa, cần phải sử dụng biến con trỏ với kiểu ở lớp cha, trỏ đến đối tượng lớp con

- Xác định phương thức nào cần đa hình, thêm từ khóa virtual cho phương thức đó ở lớp cha
- Lớp con cài đặt nạp chồng phương thức đó (không thêm virtual ở phương thức của lớp con, có thể thêm nếu kế thừa nhiều tầng !!!)

```
class DongVat
{
    //Các thuộc tính...
public:
    virtual void TiengKeu() //Phương thức ảo
    {
        cout << "Chua biet dong vat gi!!" << endl;
    }
};

void main()
{
    ConMeo conmeo1;
    DongVat* conmeo2;
    //Con trỏ có kiểu của lớp cha ĐƯỢC
    //tham chiếu đến đối tượng lớp con
    conmeo2 = &conmeo1;
    conmeo2->TiengKeu(); //Đa hình

    //Con trỏ có kiểu của lớp cha ĐƯỢC
    //tham chiếu đến đối tượng lớp con
    DongVat* conheo1 = new ConHeo();
    conheo1->TiengKeu(); //Đa hình
    delete conheo1;
}
```



```
Meo meo!!
Ut Ut!!
Press any key
```

- ✚ Cài đặt tính kế thừa kết hợp với đa hình giúp chương trình viết ngắn hơn, mang tính tổng quát đồng thời có khả năng mở rộng, tùy biến linh hoạt
- ✚ Xem xét ví dụ về khả năng mở rộng và viết mã nguồn ngắn hơn :
  - Giả sử ta cần cài đặt hàm để phát ra tiếng kêu của bất kỳ động vật nào truyền vào
    - Nếu không cài đặt đa hình : cần cài thêm xử lý riêng cho mỗi lớp con thêm mới vào do chưa biết cụ thể động vật truyền vào là động vật gì.

- Nếu có cài đặt đa hình : không cần cài xử lý riêng cho mỗi lớp con, lớp con khi thêm mới đã tự cài đặt sẵn nạp chồng phương thức đa hình.
- Minh họa mã nguồn cho cả 2 cách cài đặt :

```
//KHÔNG sử dụng tính đa hình
void PhatRaTiengKeu(DongVat dv, string loaidv)
{
    if (loaidv == "ConHeo")
    {
        //Xử lý cho trường hợp con heo
    }
    else if (loaidv == "ConMeo")
    {
        //Xử lý cho trường hợp con mèo
    }
    //Nếu có thêm con khác:
    //Phải cài lớp con, phải viết thêm xử lý cho mỗi lớp con
    //else if (loaidv == "ConVit")
    //.... con gà, con chó, .....
}

//CÓ sử dụng tính đa hình (cài đặt virtual)
void PhatRaTiengKeu(DongVat *dv)
{
    //Tính đa hình tổng quát
    //Thêm bất kỳ lớp con nào cũng không cần viết xử lý thêm
    dv->TiengKeu();
}
```

### ***b. Phương thức thuần ảo, hàm thuần ảo (pure virtual) và lớp trừu tượng (abstract class)***

- ✚ Khi cài đặt kế thừa và đa hình, thường có xu hướng đưa những cái chung lên ở lớp cha và cài đặt xử lý linh hoạt ở lớp con. Do đó, lớp cha thường sẽ không có ý nghĩa ứng dụng trong thực tế, vậy không nên tạo ra đối tượng của lớp cha.
  - Lớp con heo, con mèo đều kế thừa từ lớp động vật. Ta không thể tạo đối tượng cho lớp động vật do lớp này mang ý nghĩa trừu tượng (chưa xác định được động vật này tên gọi là gì, có tiếng kêu như thế nào ?) mà phải tạo đối tượng cụ thể là con heo hay con mèo.
  - Cách xử lý tốt hơn: ở lớp cha phương thức ảo chỉ có khai báo, không có phần cài đặt

- ✚ Các phương thức ảo chỉ có khai báo và không có phần cài đặt được gọi là phương thức thuần ảo (hay hàm thuần ảo – pure virtual method), cài đặt bằng việc gán `virtual = 0`
- ✚ Lớp nào có chứa phương thức thuần ảo được gọi là lớp trừu tượng.
- ✚ Các lớp con kế thừa từ lớp trừu tượng **bắt buộc phải cài lại toàn bộ** các phương thức thuần ảo có khai báo ở lớp cha
- ✚ Quy ước UML : các phương thức đa hình đều được viết chữ in nghiêng

```
class DongVat //Lớp cha chính là lớp trừu tượng do có 1 phương
thức thuần ảo
{
    //Các thuộc tính...
public:
    //Khai báo phương thức thuần ảo
    virtual void TiengKeu() = 0;
};

class ConHeo : public DongVat
{
public:
    //Lớp con kế thừa bắt buộc phải cài lại phương thức thuần ảo
    void TiengKeu()
    {
        cout << "Ut Ut!!" << endl;
    }
};

class ConMeo : public DongVat
{
    //Các thuộc tính riêng của con mèo
public:
    //Lớp con kế thừa bắt buộc phải cài lại phương thức thuần ảo
    void TiengKeu()
    {
        cout << "Meo meo!!" << endl;
    }
}
```

### c. Phương thức hủy ảo (hàm hủy ảo – virtual destructor)

- ✚ Do sử dụng tính chất đa hình, cần phải sử dụng con trỏ có kiểu thuộc lớp cha, trỏ đến đối tượng thuộc lớp con nào thì gọi hàm hay phương thức của lớp con đó tương ứng.
- ✚ Việc khai báo con trỏ ở lớp cha trỏ đến đối tượng ở lớp con, khi được khai báo, phương thức tạo lập ở lớp cha sẽ được gọi trước, sau đó đến phương thức tạo lập ở lớp con.

- Tuy nhiên, phương thức hủy chỉ được gọi theo đối tượng của lớp cha, do biến con trỏ ban đầu tạo lập là biến con trỏ trỏ đến kiểu dữ liệu của lớp cha
  - Để gọi được phương thức hủy của lớp con kế thừa, cần cài đặt phương thức hủy ở lớp cha là phương thức ảo, được gọi là phương thức hủy ảo.
- Trong trường hợp lớp con có biến kiểu con trỏ, nếu không cài đặt phương thức hủy ảo ở lớp cha, lớp con không được hủy, dẫn đến việc không gọi delete con trỏ được, chương trình sẽ bị **Memory Leak!!!**
- Ví dụ minh họa sau:

```
class DongVat
{
    //Các thuộc tính...
public:
    //Cài đặt phương thức hủy ảo để được gọi ở lớp con
    virtual ~DongVat()
    {
        cout << "Dong vat bi huy!!" << endl;
    }
};
class ConHeo : public DongVat
{
    int *cannang;
public:
    ~ConHeo()
    {
        delete cannang;
        cout << "Con heo bi huy!!" << endl;
    }
};

void main()
{
    //Constructor của lớp cha được gọi trước
    //Constructor của lớp con được gọi sau
    DongVat* conheo1 = new ConHeo();
    conheo1->TiengKau(); //Đã hình

    //Chỉ phương thức hủy của lớp cha được gọi
    //Nguyên nhân: biến con trỏ khai báo ban đầu là kiểu lớp cha
    //Cần cài đặt hủy ảo ở lớp cha để gọi hủy ở lớp con trước,
    rồi đến hủy lớp cha
    delete conheo1;
}
```

## II. Bài tập thực hành

**Bài tập:** Công ty ABC là công ty sản xuất kinh doanh thú nhồi bông. Công ty có nhiều nhân viên làm việc trong 3 bộ phận khác nhau: bộ phận quản lý, bộ phận sản xuất, bộ phận văn phòng. Việc tính lương cho nhân viên dựa vào các yếu tố sau:

- Đối với nhân viên văn phòng:

- $Lương = Lương\ cơ\ bản + Số\ ngày\ làm\ việc * 100.000 + Trợ\ cấp$

- Đối với nhân viên sản xuất:

- $Lương = Lương\ cơ\ bản + Số\ sản\ phẩm * 2.000$

- Đối với nhân viên quản lý:

- $Lương = Lương\ cơ\ bản * Hệ\ số\ chức\ vụ + Thưởng$

- Ngoài ra, công ty cần quản lý các thông tin về nhân viên của mình như: họ tên, năm vào làm và

các thông số trên để tính lương của nhân viên trong công ty.

Sử dụng mã nguồn của bài tập trước, hãy dựa trên đa hình và cài đặt chỉnh sửa class CongTy như sau:

1. Gom toàn bộ nhân viên của công ty trong 1 danh sách duy nhất.
2. Phương thức nhập danh sách nhân viên từ bàn phím
3. Xuất lên màn hình danh sách các nhân viên.
4. Tính tổng tiền lương của tất cả nhân viên.
5. Tìm nhân viên có lương cao nhất
6. Có bao nhiêu NVSX trong công ty
7. Có bao nhiêu NVVP trong công ty
8. Tính lương trung bình trong công ty
9. Liệt kê các nhân viên có lương thấp hơn 3.000.000
10. Nhập vào mã, tìm nhân viên tương ứng
11. Nhập vào tên, tìm nhân viên tương ứng
12. Có bao nhiêu nhân viên sinh trong tháng 5
13. Chỉnh sửa lại sơ đồ lớp hôm trước

### Yêu cầu khác:

- Các Class cần được tách ra 2 phần:
  - Phần định nghĩa thuộc tính và phương thức ở file .h
  - Phần cài đặt nằm ở file .cpp
- Hàm main nằm ở file Main.cpp, chứa các luồng demo các yêu cầu của từng bài tập như mẫu bên dưới

```
//File Main.cpp nằm Source File. Chứa hàm main  
  
void main()  
{  
    //Bài 1  
    ...  
  
    //Bài 2  
    ...  
  
    //Bài 3  
    ...  
}
```

**Các trường hợp không tuân thủ sẽ bị trừ từ 5%-20% số điểm tùy theo mức độ.**

### III. Nộp bài

**Trên lớp:** Hoàn thành ở lớp: ít nhất 5 yêu cầu.

**Về nhà:** Hoàn thành tất cả yêu cầu.

Tổ chức thư mục theo cấu trúc sau:

- **Source code:** Thư mục chứa mã nguồn. Cần xóa các tập tin trung gian của quá trình biên dịch cho nhẹ bớt (Build > Clean solution và xóa đi thư mục ẩn .vs)
- **Release:** Thư mục chứa tập tin thực thi .exe biên dịch ra từ mã nguồn.
- **Other:** Thư mục chứa các tài liệu nộp thêm ví dụ ảnh sơ đồ lớp.

Nén tất cả ở dạng <MSSV>\_Week<X>.zip/rar.

- **MSSV:** mã số sinh viên
- **X:** mã tuần
- **Y:** số bài đã làm
- Ví dụ: **21127001\_Week06\_13.zip**