Đại học Khoa Học Tự Nhiên Tp.HCM
Khoa Công Nghệ Thông Tin

PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Đa hình (Polymorphism)

Nguyễn Lê Nguyên Ngữ
nlnngu@fit.hcmus.edu.vn

Tham khảo

- TS Đinh Bá Tiến
- ThS Nguyễn Tấn Trần Minh Khang
- ThS Nguyễn Minh Huy
- ThS Lê Xuân Định
- ThS Nguyễn Hoàng Anh

Quan hệ IS-A và HAS-A

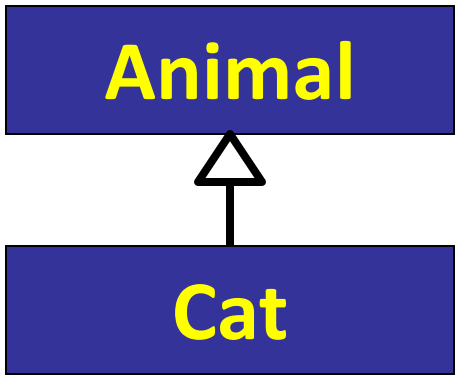
- Không được tự tiện định nghĩa là kế thừa khi chúng ta cần sử dụng lại các thuộc tính và phương thức của 1 lớp.
- Việc kế thừa chỉ nên được thực hiện khi và chỉ khi chúng có quan hệ IS-A (nghĩa là lớp này là trường hợp đặc biệt của lớp kia).
- Hai lớp có quan hệ HAS-A khi các thành phần thuộc tính của lớp này có chứa đối tượng của lớp kia. Quan hệ HAS-A là quan hệ "bao hàm" hay "bộ phận-toàn thể"

Ví dụ

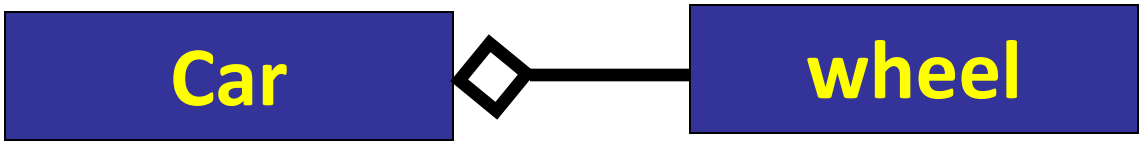
- Cat và Dog là những trường hợp đặc biệt của Animal, do đó Cat và Dog có thể kế thừa từ Animal.
- Sinh Viên Chính Quy (SVCQ) là một trường hợp đặc biệt của Sinh Viên, do đó SVCQ có thể kế thừa từ Sinh Viên.
- Tuy nhiên, 1 chiếc xe có thể có 2 hay nhiều cái bánh xe ➔ quan hệ “has a”

Ví dụ

- Cat “is an” animal.



- Car “has a” wheel.



Bài tập

- Vẽ sơ đồ lớp:
  - SinhVien.
  - SinhVienCaoDang
  - SinhVienHoanChinh.
  - LopHoc: chứa danh sách các đối tượng sinh viên.

Con trỏ đến lớp cơ sở

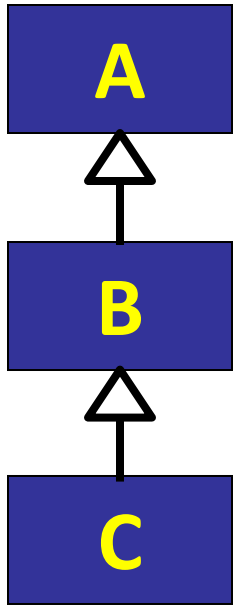
- Theo ngầm định, con trỏ của lớp cơ sở có thể được gán bằng địa chỉ của 1 đối tượng có kiểu dữ liệu là lớp dẫn xuất
- Ví dụ:

```
Animal *pAni;
Cat c;
pAni = &c; //OK
```

Liên kết tĩnh (static binding)

Xét trường hợp sau:

- `Class A` có hàm `print()`
- `Class B` viết lại hàm `print()`
- `Class C` cũng viết lại hàm `print()`



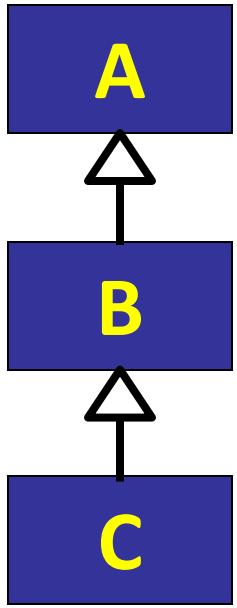

```
int main() {
    C varC;
    B varB;
    varC.print();      // print() của C
    varC.B::print();   // print() của B
    varB.print();      // print() của B
}
```

Liên kết tĩnh (tt)

- Xét tiếp trường hợp sau

```
int main() {
    A varA;
    B varB;
    C varC;
    A *var1, *var2

    var1 = &C;
    var2 = &B;
    var1->print(); // print() của A
    var2->print(); // print() của A
}
```



Hàm ảo (virtual function)

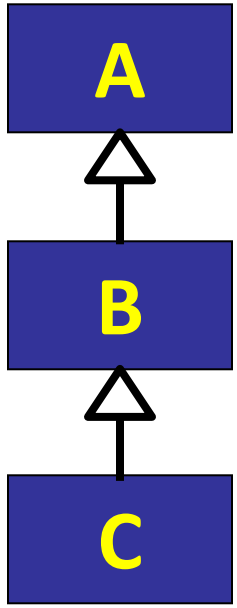
- Hàm ảo: ở lớp cơ sở, các hàm thành viên muốn trở thành hàm ảo bằng cách thêm từ khóa **virtual** vào trước tên hàm trong khai báo.
- Ví dụ: `virtual void print();`
- Ở các lớp dẫn xuất, nội dung các hàm ảo được định nghĩa lại theo đặc thù của lớp đó.
  Note: có thể không cần viết lại từ khóa virtual ở lớp dẫn xuất

Liên kết động (dynamic binding)

- Khi dùng hàm ảo, trình biên dịch sẽ đảm bảo việc sử dụng hàm thành viên tương ứng với đối tượng đang sử dụng. Ví dụ nếu `print()` là hàm ảo:

```
int main() {
    A varA;
    B varB;
    C varC;
    A *var1, *var2

    var1 = &C;
    var2 = &B;
    var1->print(); // print() của C
    var2->print(); // print() của B
}
```

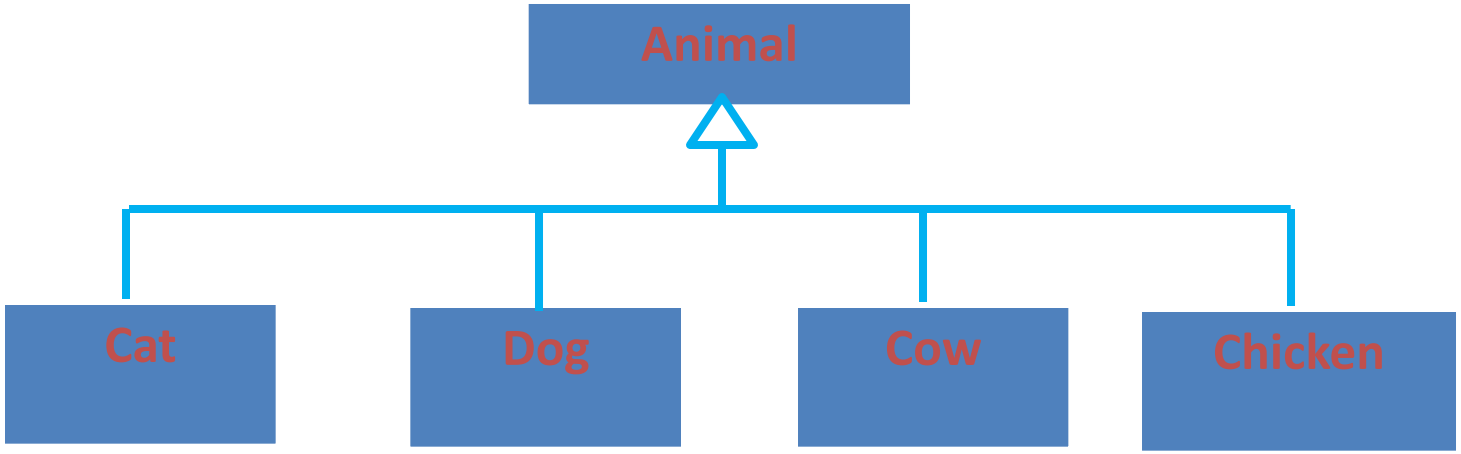


Lưu ý: hàm ảo & liên kết động

- Ngoài việc sử dụng từ khóa **`virtual`**, liên kết động chỉ xảy ra khi đối tượng được xử lý thông qua biến con trỏ hoặc tham chiếu.
- Riêng đối với lớp cơ sở, hàm ảo vẫn được sử dụng như 1 hàm bình thường.
- Nếu ở lớp dẫn xuất, hàm ảo của lớp cơ sở không được định nghĩa lại thì trình biên dịch sẽ chọn hàm tương ứng mà được định nghĩa gần nhất trong dây chuyền kế thừa.

Tính chất đa xạ (polymorphism)

- Thông qua 1 giao tiếp, có nhiều cách cài đặt khác nhau cho 1 hàm
- Hành động tương ứng của đối tượng sẽ được gọi thực hiện



Sử dụng hàm ảo để làm gì?

- Gọn gàng, đơn giản, uyển chuyển, linh động.

  ➔ Chương trình có tính dễ mở rộng, nâng cấp.

```
void giveATalk(Animal *p)
{
    p->talk();
}
```

```
void giveATalk(Animal obj, int
iType)
{
    if (iType == 0)
    {
        Cat   c = (Cat)obj;
        c.talk();
    }
    else if (iType == 1)
    {
        Dog   d = (Dog)obj;
        d.talk();
    }
}
```

Hàm dựng ảo & Hàm hủy ảo

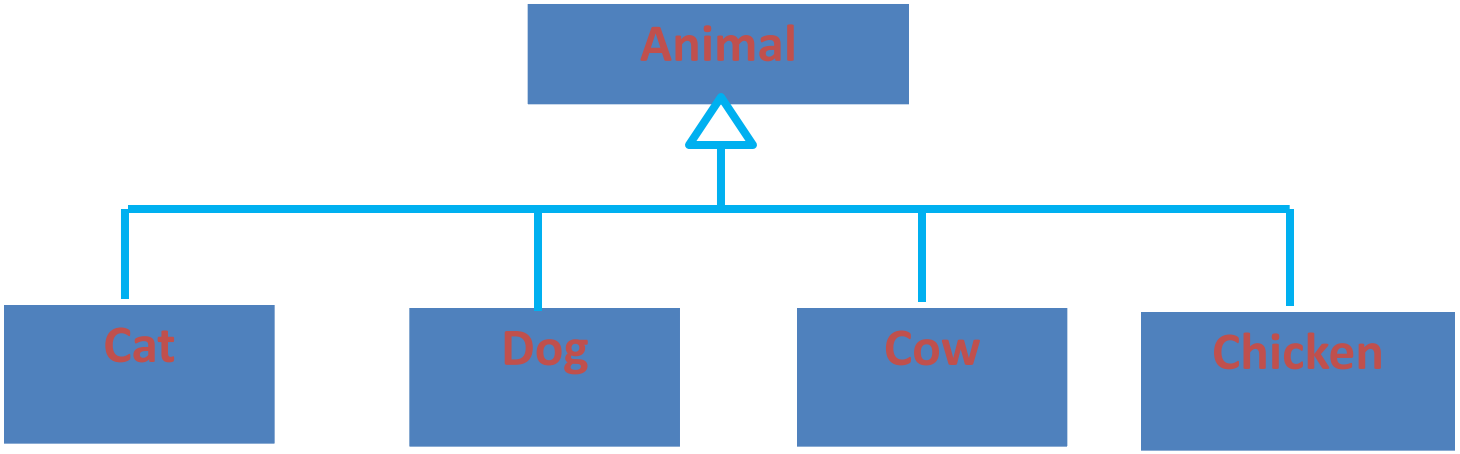
- KHÔNG có hàm dựng ảo
  - Vì hàm dựng không được kế thừa nên cũng không cần có nhu cầu để có hàm dựng ảo
- Hàm hủy nên được viết là hàm ảo
  - Vì nếu lớp này được kế thừa nhưng hàm hủy vẫn sử dụng theo liên kết tĩnh sẽ dễ dẫn đến các vấn đề về xử lý bộ nhớ

Hàm thuần ảo (pure virtual)

- Xét lại bài Animal, giả sử ta có hàm ảo:

```
virtual void talk();
```

// thể hiện tiếng kêu của con vật



Hàm thuần ảo (tt)

- Khi đó

```
int main()
{
   Animal a; //không thực tế
   a.talk(); //không có ý nghĩa
}
```

- `talk()` có thể được khai báo lại như sau:

```
virtual void talk() = 0;
```

Hàm thuần ảo & lớp trừu tượng

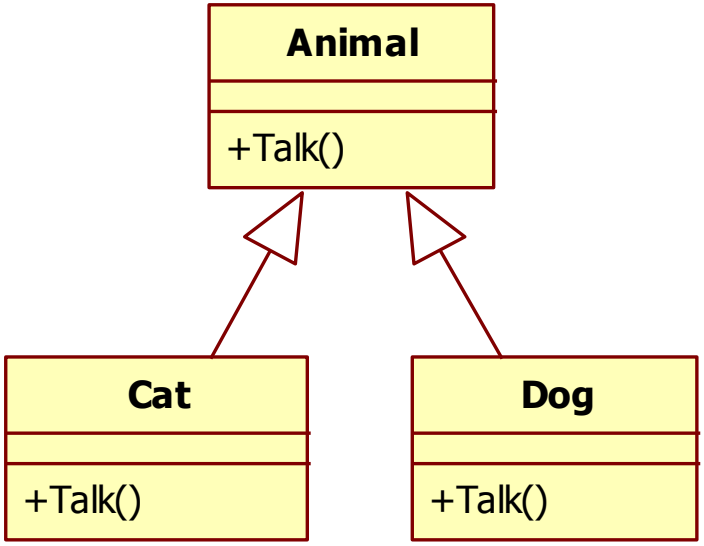
```
virtual void talk() = 0;
```

- Hàm `talk()` trở thành hàm thuần ảo
- Khi lớp chứa hàm thuần ảo thì được gọi là lớp trừu tượng (abstract class) và đối tượng sẽ không được khởi tạo từ lớp đó:

```
Animal a; //báo lỗi biên dịch
```

- Ví dụ:

```
class Animal
{
public:
        void talk() {   cout << “Don’t talk!”;   }
};
class Cat: public Animal
{
public:
        void talk() {   cout << “Meo meo!”;   }
};
class Dog: public Animal
{
public:
        void talk() {   cout << “Gau gau!”;   }
};
```



- Ví dụ:

```
void giveATalk(Animal *p)
{
        p->talk();
}

void main()
{
        Cat     c;
        Dog     d;

        giveATalk(&c);
        giveATalk(&d);
}
```

Animal talks!!

Animal talks!!

```
void main()
{
        Animal  a;
        Cat     c;
        Dog     d;

        Animal  *p;
        p = &a;
        p->talk();

        p = &c;
        p->talk();

        p = &d;
        p->talk();
}
```

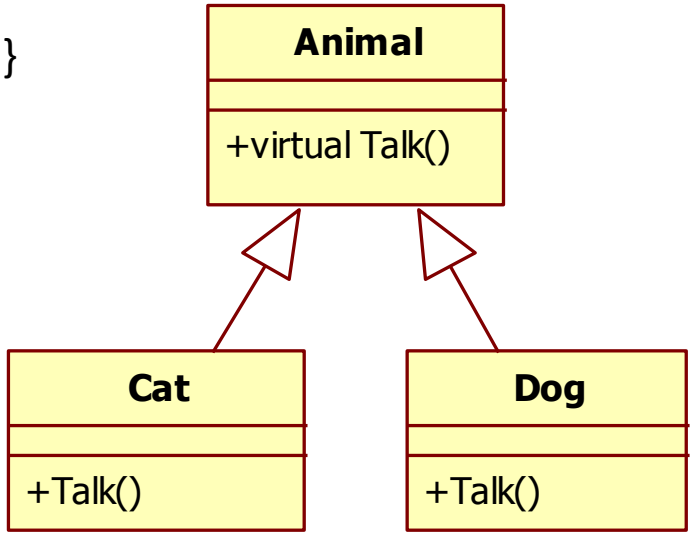
Animal talks!!

Animal talks!!

Animal talks!!

- Ví dụ:

```
class Animal
{
public:
        virtual void talk() {   cout << “Don’t talk!”;   }
};
class Cat: public Animal
{
public:
        void talk() {   cout << “Meo meo!”;   }
};
class Dog: public Animal
{
public:
        void talk() {   cout << “Gau gau!”;   }
};
```



- Ví dụ:

```
void giveATalk(Animal *p)
{
        p->talk();
}

void main()
{
        Cat         c;
        Dog         d;

        giveATalk(&c);
        giveATalk(&d);
}
```

Cat talks!!

Dog talks!!

```
void main()
{
        Animal    a;
        Cat       c;
        Dog       d;

        Animal    *p;
        p = &a;
        p->Talk();

        p = &c;
        p->Talk();

        p = &d;
        p->Talk();
}
```

Animal talks!!

Cat talks!!

Dog talks!!