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

PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Tính kế thừa

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

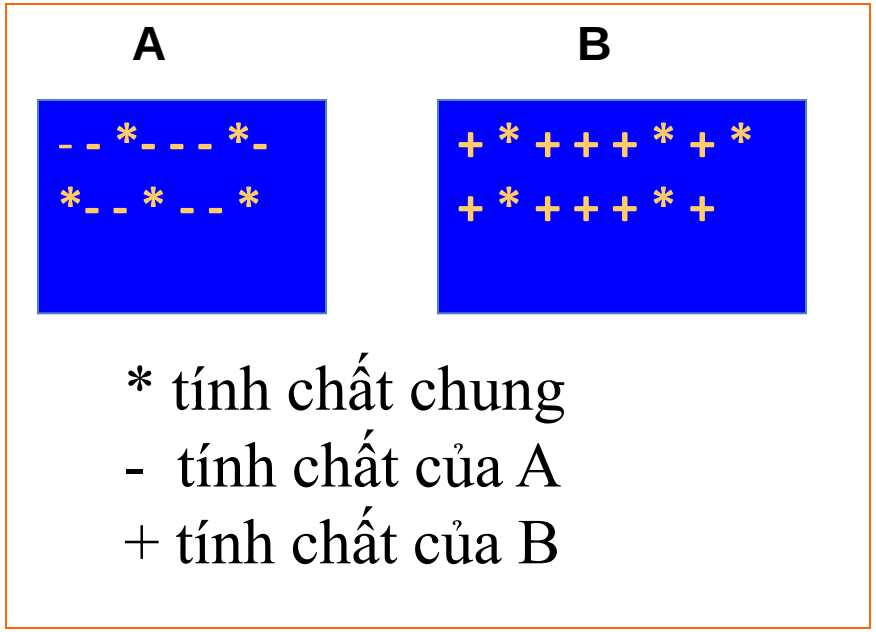
Tham khảo

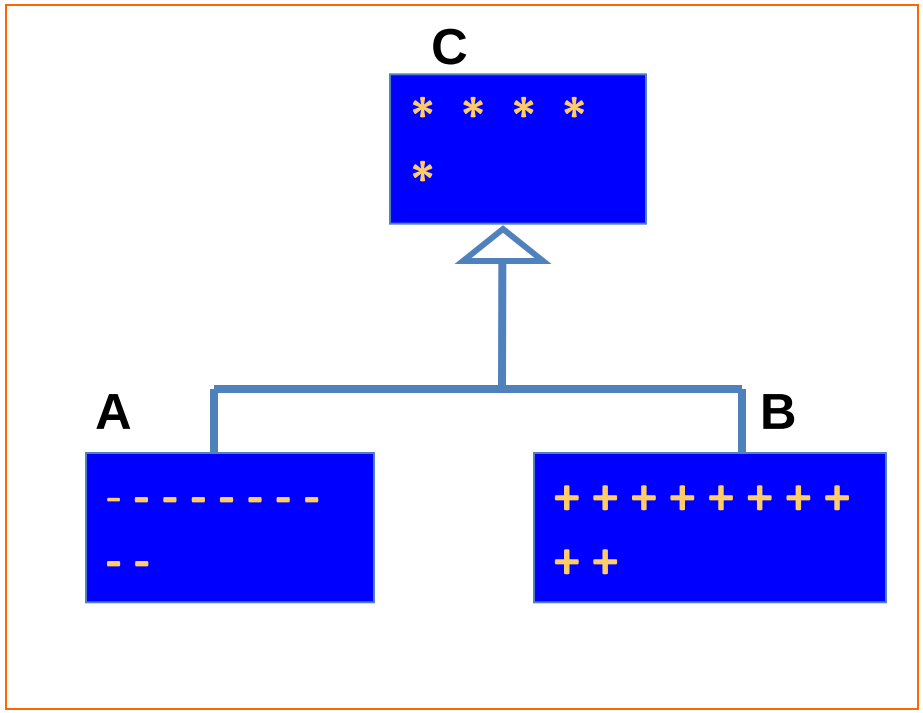
- TS Đinh Bá Tiến
- ThS Nguyễn Tấn Trần Minh Khang
- ThS Nguyễn Minh Huy
- ThS Lê Xuân Định
- ThS Nguyễn Hoàng Anh

Giới thiệu

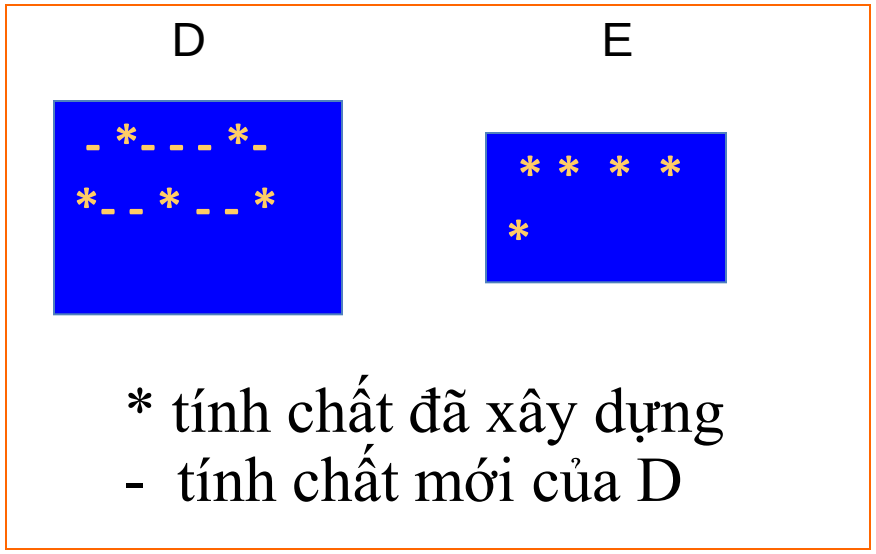
- Một trong những tính chất quan trọng trong phương pháp lập trình hướng đối tượng là khả năng tái sử dụng các lớp đã được định nghĩa.
- Với tính kế thừa, người lập trình có thể định nghĩa lớp đối tượng mới dựa trên 1 hay nhiều lớp đối tượng đã có sẵn.
- Lớp có sẵn được gọi là lớp cơ sở (based class) và lớp kế thừa được gọi là lớp dẫn xuất (derived class)

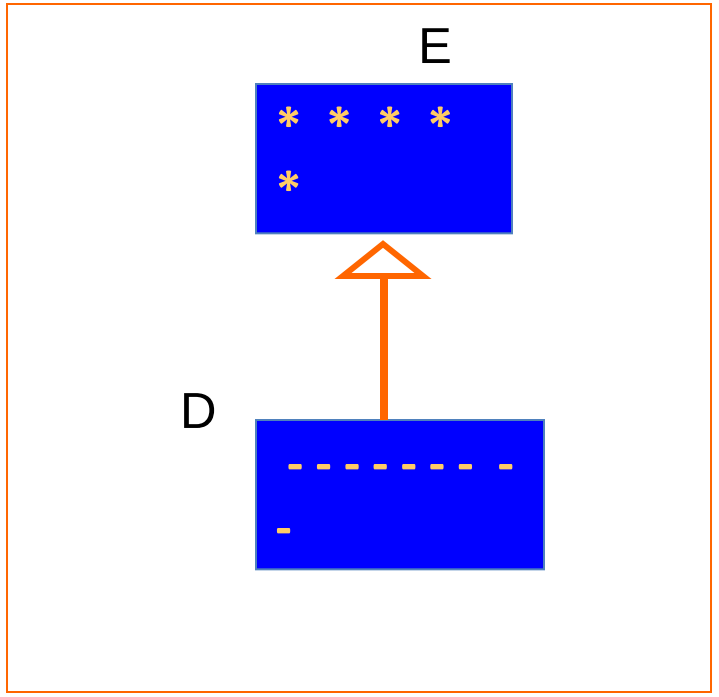
Kế thừa



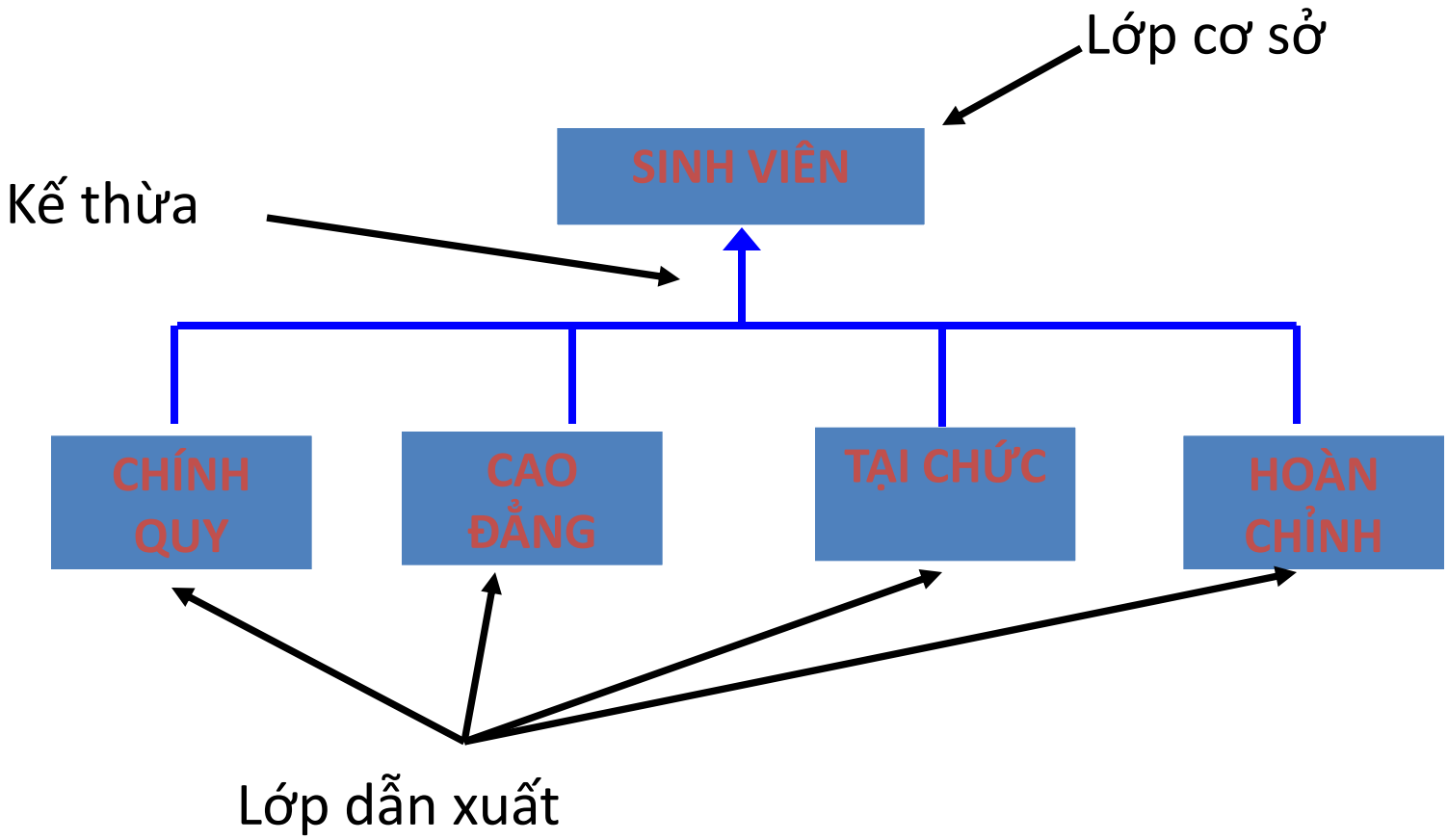


Kế thừa





Ví dụ


Lớp cơ sở
SINH VIÊN
Kế thừa
CHÍNH QUY
CAO ĐẲNG
TẠI CHỨC
HOÀN CHỈNH
Lớp dẫn xuất

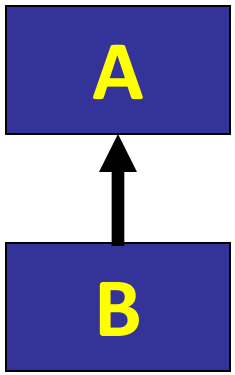
Cú pháp

Trong C++, kế thừa được khai báo như sau:

```
class <Tên lớp kế thừa> : <Loại kế thừa> <Tên lớp cơ sở>
{
      private:
            <Khai báo thành phần riêng>

      public:
            <Khai báo thành phần công cộng>
};
```

Một số lưu ý khi kế thừa



- Các thành phần thuộc tính và hành động **`public`** của A sẽ là các thành phần trong B
- Các thành phần **`private`** của A sẽ là 1 phần trong B nhưng chỉ được truy xuất qua các hàm **`public`** hay **`protected`** của A

Từ khóa `protected`

- Các thành phần protected của lớp cơ sở sẽ được nhìn thấy (hay truy xuất trực tiếp) được trong lớp dẫn xuất nhưng không được nhìn thấy/truy xuất từ bên ngoài.

Các loại kế thừa

Có 3 loại kế thừa:

- **`public`**
- **`protected`**
- **`Private`**

Lưu ý: Nếu không nói rõ là loại kế thừa gì, chúng ta ngầm định đó là kế thừa **`public`**

Các loại kế thừa

- **`public`**: các thành phần **`public`** và **`protected`** của lớp cơ sở là các thành phần **`public`** và **`protected`** của lớp dẫn xuất.
- **`protected`**: Các thành phần **`public`** và **`protected`** của lớp cơ sở là các thành phần **`protected`** của lớp dẫn xuất.
- **`private`**: Các thành phần **`public`** và **`protected`** của lớp cơ sở là các thành phần **`private`** của lớp dẫn xuất.

Kế thừa phương thức

- Các phương thức của lớp cơ sở sẽ được kế thừa trong lớp dẫn xuất, ngoại trừ:
  - Hàm dựng
  - Hàm hủy
  - Toán tử gán bằng

- Lưu ý: các phương thức **`private`** của lớp cơ sở vẫn tồn tại trong lớp dẫn xuất nhưng phải được truy xuất thông qua các phương thức **`protected`** hay **`public`** của lớp cơ sở.

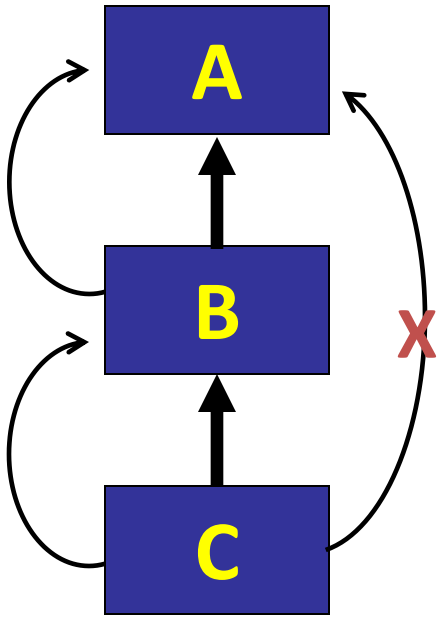
Hàm dựng trong kế thừa

- Khi một đối tượng thuộc lớp dẫn xuất được tạo lập:
  - Hàm dựng của lớp cơ sở sẽ tự động được gọi thực hiện trước
  - Sau đó, hàm dựng của lớp dẫn xuất sẽ được thực hiện.
  - Trong hàm dựng của lớp dẫn xuất, chúng ta có thể chỉ định hàm dựng nào của lớp cơ sở sẽ được gọi thực hiện. Nếu không, hàm dựng mặc định của lớp cơ sở sẽ được gọi.

Hàm dựng trong lớp dẫn xuất

Lưu ý:

- Hàm dựng của lớp dẫn xuất chỉ được phép chỉ định hàm dựng nào của **lớp cơ sở trực tiếp** của nó thực hiện chứ không thể can thiệp đến các lớp cơ sở kế thừa xa hơn.



Ví dụ

```
class A
{
public:
      A();
      A(int);
};

class B : public A
{
public:
      B(int);
};
```

```
class A {
public:
      A();
      A(int);
};

class B : public A
{
public:
      B(int t) : A(t) {
            ...
      }
};
```

Hàm hủy trong kế thừa

- Khi một đối tượng thuộc lớp kế thừa bị hủy:
  - Hàm hủy của lớp kế thừa được gọi thực hiện trước
  - Sau đó mới đến hàm hủy của lớp cơ sở.

Định nghĩa lại phương thức

- Đôi khi chúng ta cần định nghĩa lại các phương thức của lớp cơ sở trong lớp dẫn xuất

➔ Việc này được thực hiện bằng cách khai báo và cài đặt lại các phương thức này trong lớp dẫn xuất.

Phương thức được định nghĩa lại còn được gọi là hàm nạp chồng (overriden function).

Lưu ý: hàm được định nghĩa lại này sẽ che đi các hàm trùng tên khác (nếu có) trong lớp cơ sở.

Toán tử gán cho lớp dẫn xuất

- Toán tử gán cho các lớp cơ sở
- Xây dựng toán tử gán cho lớp dẫn xuất, trong đó sử dụng toán tử gán của lớp cơ sở và xây dựng việc gán thêm các thành phần riêng của lớp dẫn xuất

Toán tử gán cho lớp dẫn xuất

```
B& B::operator=(const B& src)
{
   if (this == &src)
      return *this;
   A::operator=(src);
   delete [] ptr;
   iSize = src.iSize;
   ptr = new int [iSize];
   for (int i=0; i<iSize; ++i)
      ptr[i] = src.ptr[i];
   return *this;
}
```