Chương 10

Lập trình hướng đối tượng

-- chủ đề nâng cao --

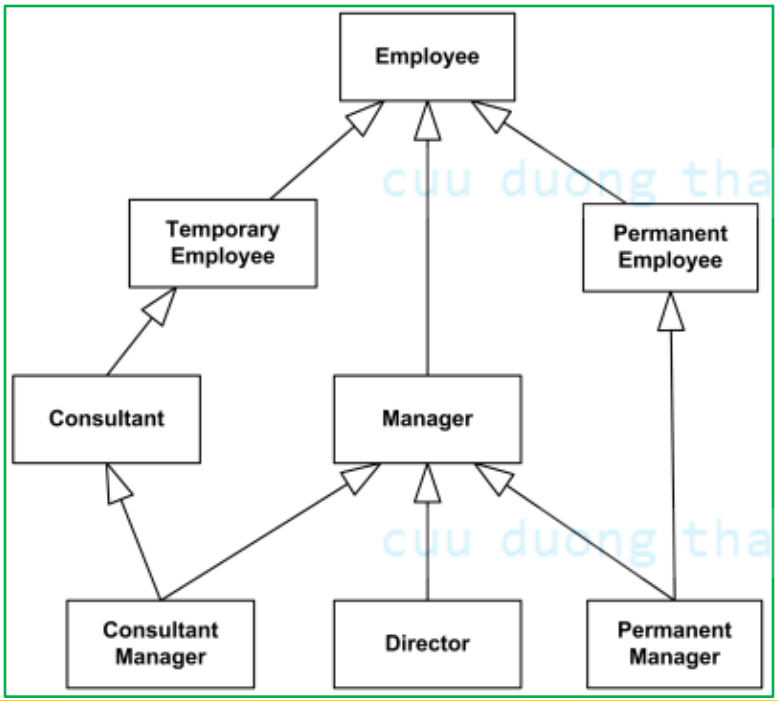
Lê Thành Sách

Nội dung

- Đa thừa kế
- Đa thừa kế: thừa kế ảo
  - Khởi động lớp cha trong thừa kế ảo
- Đa hình (polymorphism)
- Đa hình: hàm có tính abstract
- Tổng kết

Đa thừa kế

- Là gì?
  - Là một lớp thừa kế các thành viên từ nhiều hơn một lớp, như ví dụ sau.



Lớp **ConsultantManager** và **PermanentManager**, có đến 2 lớp cha.
Trường hợp tổng quát: có thể có nhiều cha.

http://www.uml-diagrams.org/generalization.html

Đa thừa kế

- Mô tả đa thừa kế ntn?
  - Liệt kê các lớp cha như ví dụ sau.
  - Sử dụng dấu phẩy để ngăn cách.

Dùng dấu phẩy ",” để liệt kê các lớp cha

```
class Employee{
};
class TemporaryEmployee: public Employee{
};
class PermanentEmployee: public Employee{
};
class Consultant: public TemporaryEmployee{
};
class Manager: public Employee{
};
class ConsultantManager: public Consultant, public Manager{
};
class Director: public Manager{
};
class PermanentManager: public Manager, public PermanentEmployee{
};
```

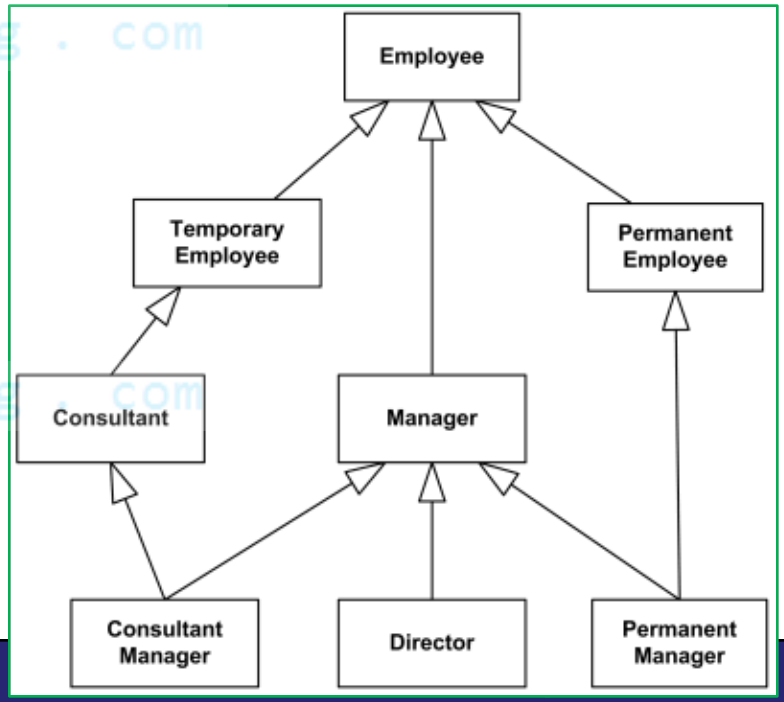
Đa thừa kế

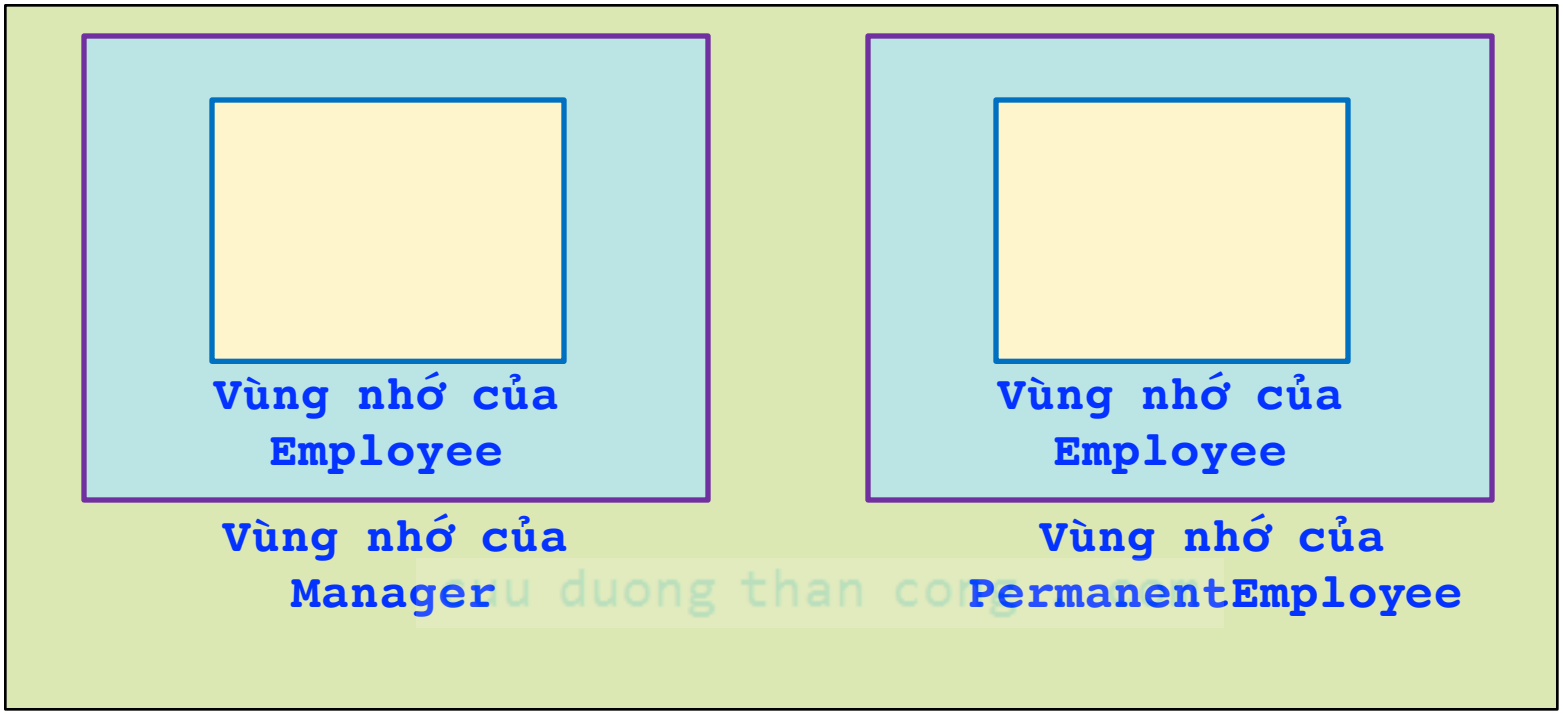
- Sơ đồ bộ nhớ của đối tượng
  - Giả sử có hệ thống lớp như hình vẽ,
  - Cũng giả sử code C++ được sinh ra như slide trước.

  - Xét dòng khai báo biến (tạo đối tượng) như sau:

```
PermanentManager obj;
```

  - Bộ nhớ của đối tượng "obj" được tổ chức ntn?

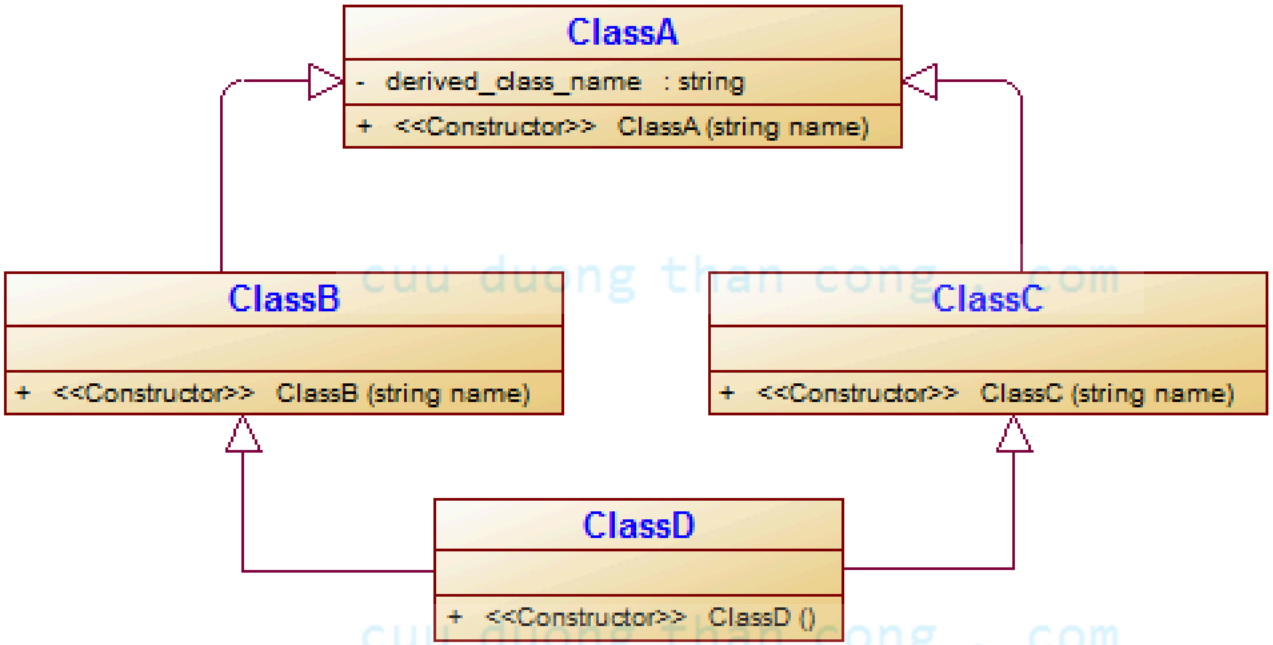




Theo cách mô tả thừa kế như slide trước:
Bên trong đối tượng kiểu "**PermanentManager**" có đến 2 đối tượng kiểu "**Employee**" hoàn toàn riêng biệt và khác nhau

Đa thừa kế: Minh hoạ (I)

- Xét sơ đồ đa thừa kế như hình vẽ



Hiện thực cho các lớp cho ở các slide kế tiếp

Đa thừa kế: Minh hoạ (I)

Chứa tên của lớp con: **ClassB** hoặc **ClassC**

Khởi động biến

```
class ClassA{
private:
        string derived_class_name;
public:
        ClassA(string name): derived_class_name(name){
                cout << "In ClassA(string name)" << endl;
        }
        void display(){
                cout << "My Drived Class is "
                << this->derived_class_name << endl;
        }
};
```

Sẽ in ra tên lớp chứa trong biến **derived_class_name**
➔ Sẽ là **ClassB** hoặc **ClassC**

Đa thừa kế: Minh hoạ (I)

ClassB thừa kế **ClassA**, với tính public

Gọi hàm khởi tạo lớp **ClassA**

```
class ClassB:  public ClassA{
public:
        ClassB(string name): ClassA(name){
                cout <<  "In ClassB(string name)"
                << endl;
        }
};
```

Đa thừa kế: Minh hoạ (I)

ClassC thừa kế **ClassA**, với tính public

Gọi hàm khởi tạo lớp **ClassA**

```
class ClassC:  public ClassA{
public:
        ClassC(string name): ClassA(name){
                cout <<  "In ClassC(string name)"
                << endl;
        }
};
```

Đa thừa kế: Minh hoạ (I)

ClassD thừa kế cả hai lớp **ClassB** và **ClassC**

Gọi hàm khởi tạo của hai lớp cha: **ClassB** và **ClassC**

```
class ClassD: public ClassB,public ClassC{
public:
        ClassD():
                ClassB("ClassB"),
                ClassC("ClassC")
        {
        }
};
```

Đa thừa kế: Minh hoạ (I)

(1) obj: chứa bên trong đến 2 đối tượng kiểu ClassA

(2): nếu gọi "**display**" như dòng này **sẽ báo lỗi**.
Vì: có hai phiên bản của "**display**" cùng tồn tại, bộ biên dịch không biết phải dùng hàm nào.

```
int main(){
    ClassD obj;
    obj.ClassB::display();
    obj.ClassC::display();
    //obj.display();

    return 0;
};
```

Đa thừa kế: Minh hoạ (I)

In ra 4 dòng, vì sao?

Kết quả chạy chương trình

```
In ClassA(string name)
In ClassB(string name)
In ClassA(string name)
In ClassC(string name)
My Derived Class is ClassB
My Derived Class is ClassC
```

```
int main(){
    ClassD obj;
    obj.ClassB::display();
    obj.ClassC::display();
    //obj.display();

    return 0;
};
```

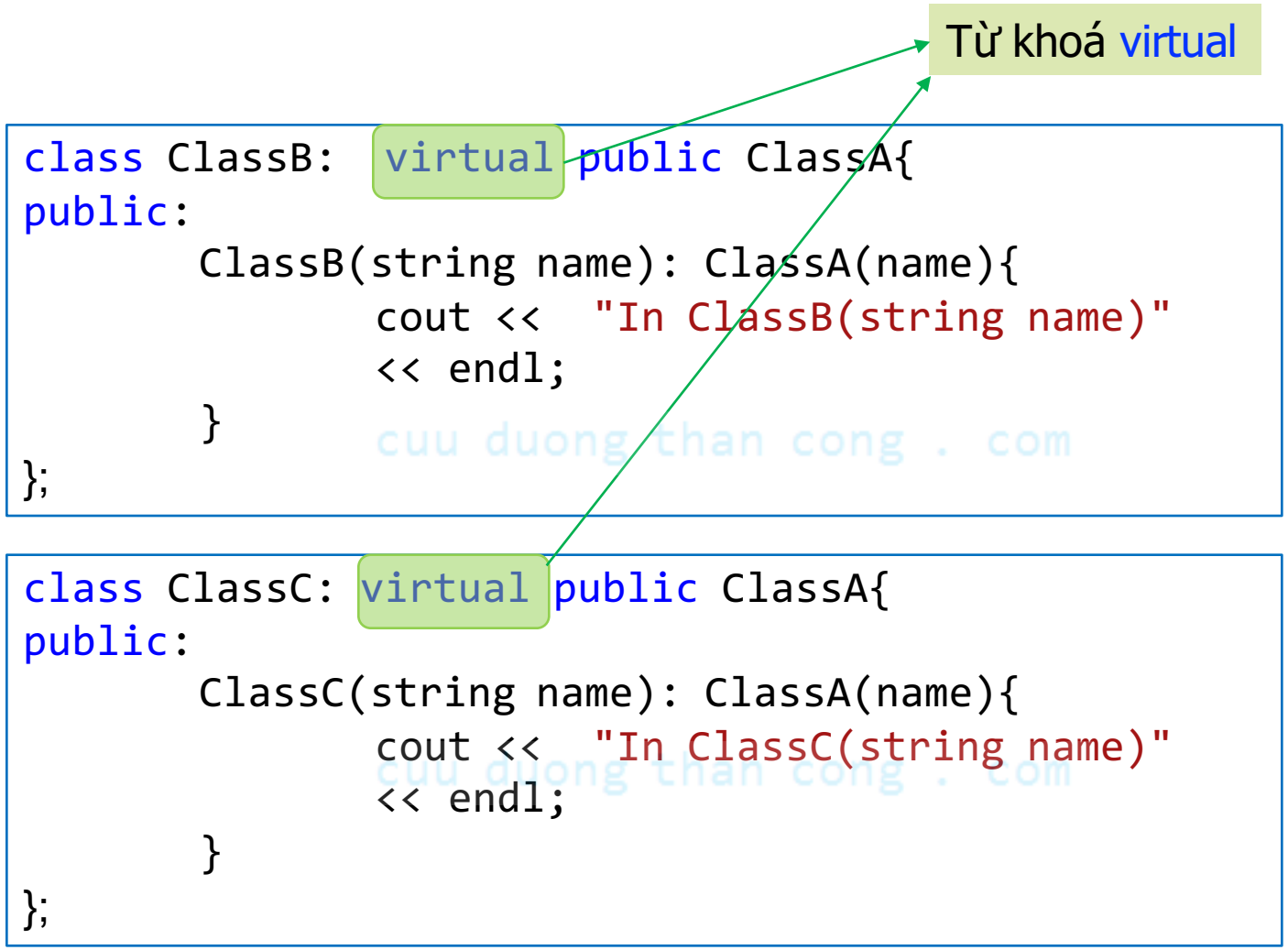
Sẽ báo lỗi nếu dùng!

Đa thừa kế: thừa kế ảo (virtual)

- Thừa kế ảo là gì?
  - Như trường hợp ở slide trước: đối tượng của lớp cha (như ClassA ở trên) có thể được cấp phát lặp lại nhiều hơn 1 lần → không mong muốn
    - Đây là bài toán: "**diamon problem**"

  - Thừa kế ảo (**virtual**) giúp cho đối tượng của lớp cha (như ClassA ở trên) **chỉ được cấp phát một lần**.
  - Khai báo ntn? Như slide sau:

Đa thừa kế: thừa kế ảo (virtual)



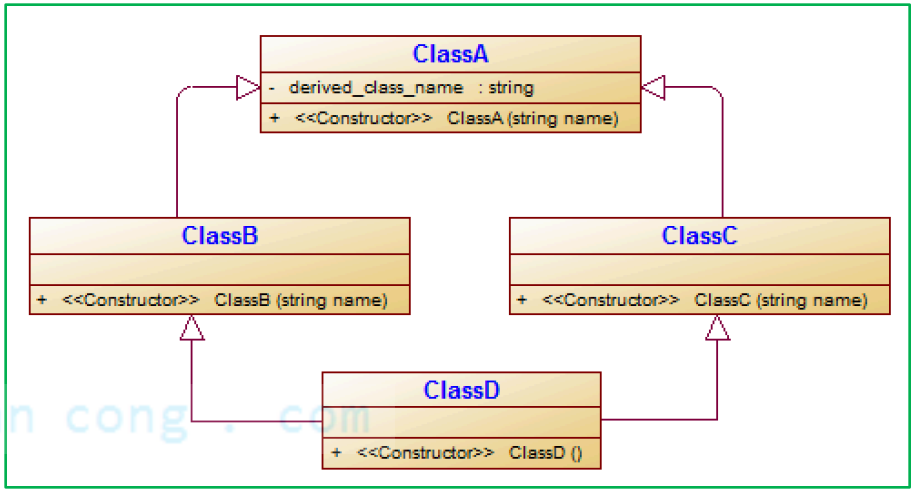
Đa thừa kế: thừa kế ảo (virtual)

Không cần dùng virtual với ClassB và ClassC

```
class ClassD: public ClassB,public ClassC{
public:
        ClassD():
                ClassB("ClassB"),
                ClassC("ClassC")
        {
        }
};
```

Đa thừa kế: thừa kế ảo (virtual)

- Vấn đề khởi động lớp cha:
  - (1) Gọi hàm khởi động cho lớp cha chung, như ClassA, phải từ lớp con chung, như lớp ClassD.
    - Các khởi động ở lớp trung gian, như ClassB và ClassC đều không có tác dụng.
  - (2) Nếu lớp con chung không gọi hàm khởi động của lớp cha chung thì hàm khởi tạo mặc nhiên (không thông số) của lớp cha chung sẽ được gọi.



Đa thừa kế: thừa kế ảo: Minh hoạ (I)

- Nếu chỉ đơn giản thêm từ khoá "virtual" vào khai báo cho các lớp ClassB và ClassC như slide trước, sẽ có lỗi biên dịch
  - Lớp ClassA không có hàm khởi tạo mặc nhiên.
  - Lý do:
    - Lớp ClassD (con chung) không gọi hàm khởi tạo cho lớp ClassA
      ➔ Hàm khởi tạo mặc nhiên của ClassA sẽ được gọi, nhưng nó không có – xem lớp ClassA.

```
class ClassD: public ClassB, public ClassC{
public:
        ClassD():
                ClassB("ClassB"),
                ClassC("ClassC")
        {
        }
};
```

ClassD không khởi động cho ClassA

Đa thừa kế: thừa kế ảo: Minh hoạ (I)

```
class ClassD: public ClassB, public ClassC{
public:
        ClassD():
                ClassA("From ClassD"),
                ClassB("ClassB"),
                ClassC("ClassC")
        {
        }
};
```

Nếu khởi động lớp ClassA tại lớp ClassD, và hàm main cho sau đây:

Đa thừa kế: thừa kế ảo: Minh hoạ (I)

```
int main(){
    ClassD obj;
    obj.ClassB::display();
    obj.ClassC::display();
    obj.display();

    return 0;
};
```


Vì sao?

Kết quả chạy chương trình

```
In ClassA(string name)
In ClassB(string name)
In ClassC(string name)
My Derived Class is From ClassD
My Derived Class is From ClassD
My Derived Class is From ClassD
```

Tính đa hình trong lập trình hướng đối tượng

Đa hình là gì?

- Thuật ngữ
  - Đa hình = Polymorphism
- Đa hình là một khả năng của OOP, hoạt động như sau:
  - **(1)** Một con trỏ kiểu lớp cha:
    - Như ví dụ:
      - `Base* ptr;`
    - Tại lúc chương trình thực thi (run-time), có thể được gán địa chỉ của đối tượng kiểu lớp con, như ví dụ sau:

```
ptr = new DerivedClass()
```

  - **(2)** Khi gọi hàm với con trỏ lớp cha (ptr ở trên) thì hàm trong lớp con được gọi, không phải hàm trong lớp cha.

```
ptr->foo();
```

Đa hình là gì?

- Tại sao gọi là đa hình
  - Hàm foo(), như slide trước, hoạt động như thế nào là tuỳ thuộc vào phiên bản nào (của lớp con nào) thật sự được gọi tại thời điểm thực thi
  - Cũng có nghĩa: chỉ mỗi một dòng lệnh

```
ptr->foo();
```

    - Cách hoạt động (hành xử) khác nhau
    - Nên được gọi là đa hình

Đa hình là gì?

- Cho đến thời điểm này:
  - Lời gọi hàm:

```
ptr->foo();
```

  - Vẫn cứ gọi hàm foo() trong lớp cha, nghĩa là lớp **BaseClass**.
  - Xem ví dụ sau

Đa hình: Minh hoạ (I)

```
class BaseClass{
public:
        void foo(){
                cout << "BaseClass" << endl;
        }
};
class DerivedClass: public BaseClass{
public:
        void foo(){
                cout << "DerivedClass" << endl;
        }
};

int main(){
    BaseClass* ptr;
    ptr = new DerivedClass();
    ptr->foo();
    return 0;
};
```


Kết quả chạy chương trình, in ra: BaseClass

Vì sao?

Đa hình: Minh hoạ (I)

```
class BaseClass{
public:
        void foo(){
                cout << "BaseClass" << endl;
        }
};
class DerivedClass: public BaseClass{
public:
        void foo(){
                cout << "DerivedClass" << endl;
        }
};

int main(){
    BaseClass* ptr;
    ptr = new DerivedClass();
    ptr->foo();
    return 0;
};
```


Lý do: mô tả hàm foo() như thế này, bộ biên dịch hiểu rằng hàm **foo()** trong phát biểu: **`ptr->foo()`** là của lớp BaseClass.

Nghĩa là: xác định hàm được gọi tại thời điểm biên dịch, nên còn gọi là "ràng buộc sớm" (**early binding**)

Đa hình: Minh hoạ (I)

Khai báo nào sẽ hỗ trợ tính đa hình?

Trả lời: sử dụng từ khoá "virtual" như ví dụ sau.

Đa hình: Minh hoạ (I)

```
class BaseClass{
public:
        virtual void foo(){
                cout << "BaseClass" << endl;
        }
};
class DerivedClass: public BaseClass{
public:
        void foo(){
                cout << "DerivedClass" << endl;
        }
};

int main(){
    BaseClass* ptr;
    ptr = new DerivedClass();
    ptr->foo();
    return 0;
};
```

Sử dụng từ khoá "virtual"

Không cần lặp lại ở lớp con

Khi chạy, in ra:

DerivedClass

Đa hình: Minh hoạ (I)

Có thể dùng từ khoá `override` nếu muốn cho rõ ràng, từ khoá này nằm sau cùng, liền trước dấu {

```
class BaseClass{
public:
        virtual void foo(){
                cout << "BaseClass" << endl;
        }
};
class DerivedClass: public BaseClass{
public:
        void foo() override{
                cout << "DerivedClass" << endl;
        }
};

int main(){
    BaseClass* ptr;
    ptr = new DerivedClass();
    ptr->foo();
    return 0;
};
```

Khi chạy, in ra:

DerivedClass

Đa hình: Minh hoạ (I)

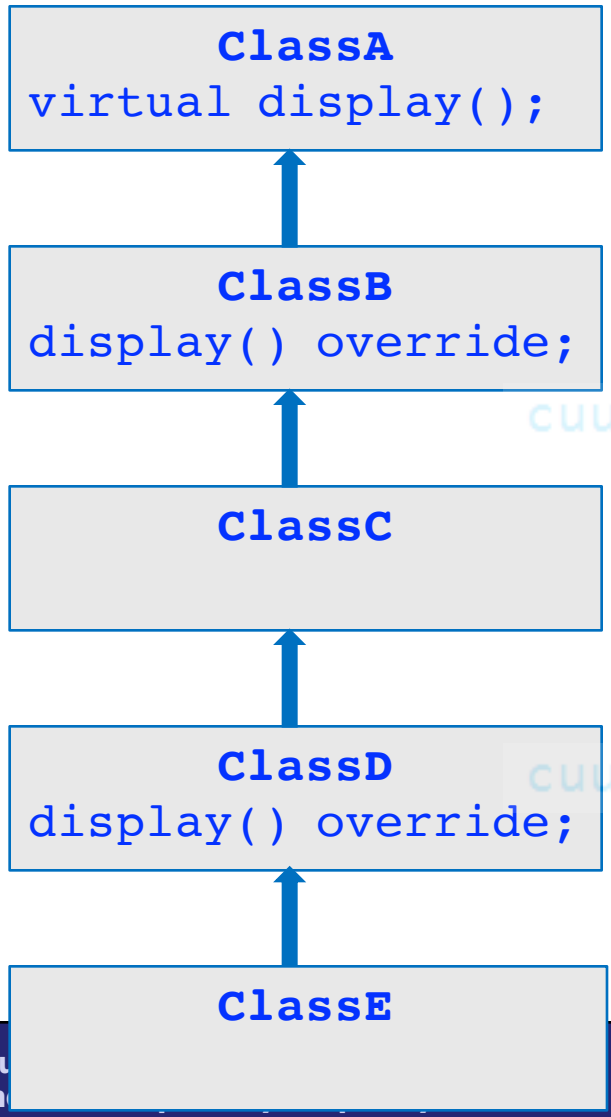
```
class BaseClass{
public:
        virtual void foo(){
                cout << "BaseClass" << endl;
        }
};
class DerivedClass: public BaseClass{
public:
        void foo() override{
                cout << "DerivedClass" << endl;
        }
};

int main(){
    BaseClass* ptr;
    ptr = new DerivedClass();
    ptr->foo();
    return 0;
};
```

Hàm foo() nào được gọi, trong lớp cha hay lớp con; trường hợp tổng quát thì lớp con nào.

➔ Trong thời điểm thực thi mới biết.

➔ Gọi là ràng buộc động (**dynamic binding**)

Đa hình: Bài tập



Giả sử có hệ thống các lớp như hình, tất cả thừa kế đều có tính public.

Con trỏ "ptr" được khai báo:

```
ClassA* ptr;
```

Phiên bản "display" của lớp nào được gọi trong các dòng sau đây?

```
ptr = new ClassA(); ptr->display();
ptr = new ClassB(); ptr->display();
ptr = new ClassC(); ptr->display();
ptr = new ClassD(); ptr->display();
ptr = new ClassE(); ptr->display();
```

Đa hình: Hàm có tính abstract

- Hàm (phương thức) có tính abstract nghĩa là nó được khai báo rằng chưa có bản hiện thực nào gắn với nó, như ví dụ:

```
class BaseClass{
public:
        virtual void foo() = 0;
};
```

Hàm có tính abstract

Đa hình: Hàm có tính abstract

- Hàm (phương thức) có tính abstract:
  - Các lớp con phải hiện thực hàm có tính này.
  - Lớp X nào có chứa hàm có tính này thì không thể tạo được đối tượng với kiểu lớp X được.

- Tại sao phải dùng tính abstract:
  - Khi thiết kế các dự án lớn, có những lớp được thiết kế để chỉ quy định rằng lớp đó có hỗ trợ tính năng (hàm gì), còn hiện thực thì nằm ở lớp con thừa kế từ nó.

  - Với tính năng này cộng với đa thừa kế: C++ có thể hiện thực được khái niệm “interface” của Java.

Tổng kết

- Đến lúc này, mọi tính năng quan trọng của phương pháp lập trình hướng đối tượng đã được giới thiệu và minh hoạ.
- Các tính năng đã học trong phần này
  - Đa thừa kế.
  - Chú ý:
    - Khi đa thừa kế thì có khả năng tạo thành vòng
      - => dùng thừa kế ảo (từ khoá virtual)
    - Một khi đã dùng từ khoá virtual, thì
      - => Chú ý đến việc khởi động lớp cha chung từ lớp con chung.

Tổng kết

- Các tính năng đã học trong phần này
  - Tính đa hình
    - Đây là tính năng khá hay của OOP
    - Tính năng này có được là do công việc xác định hàm được gọi được thực hiện tại thời điểm chương trình thực thi (run-time), không phải tại thời điểm biên dịch (compile-time).
  - Cũng có nghĩa,
    - Những hàm không có tính "virtual" thì bộ biên dịch biết được phiên bản nào của hàm đó được dùng ngay tại thời điểm biên dịch.
  - Hàm có tính abstract, giúp cho quá trình thiết kế tách biệt giữa những quy định và hiện thực những quy định (khái niệm trừu tượng!)