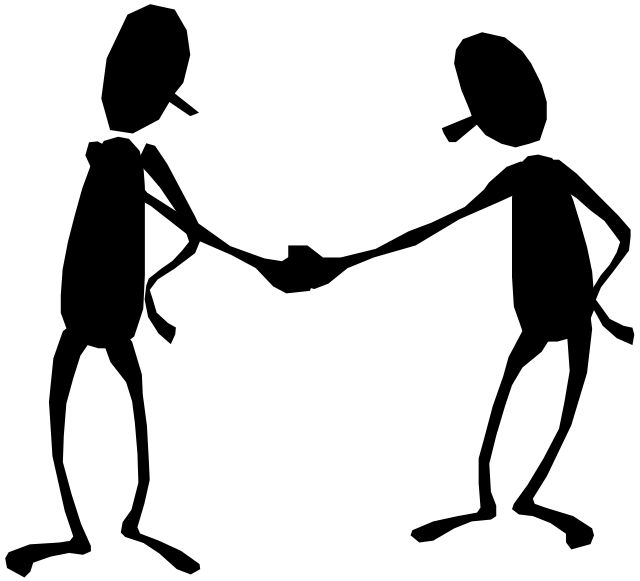
C/C++

Version: 1.0

Tác giả:



Date of training course (automatically update

Monday, June 20, 2022

Thời lượng: 4 hours

Giảng viên:

<Instructor title>

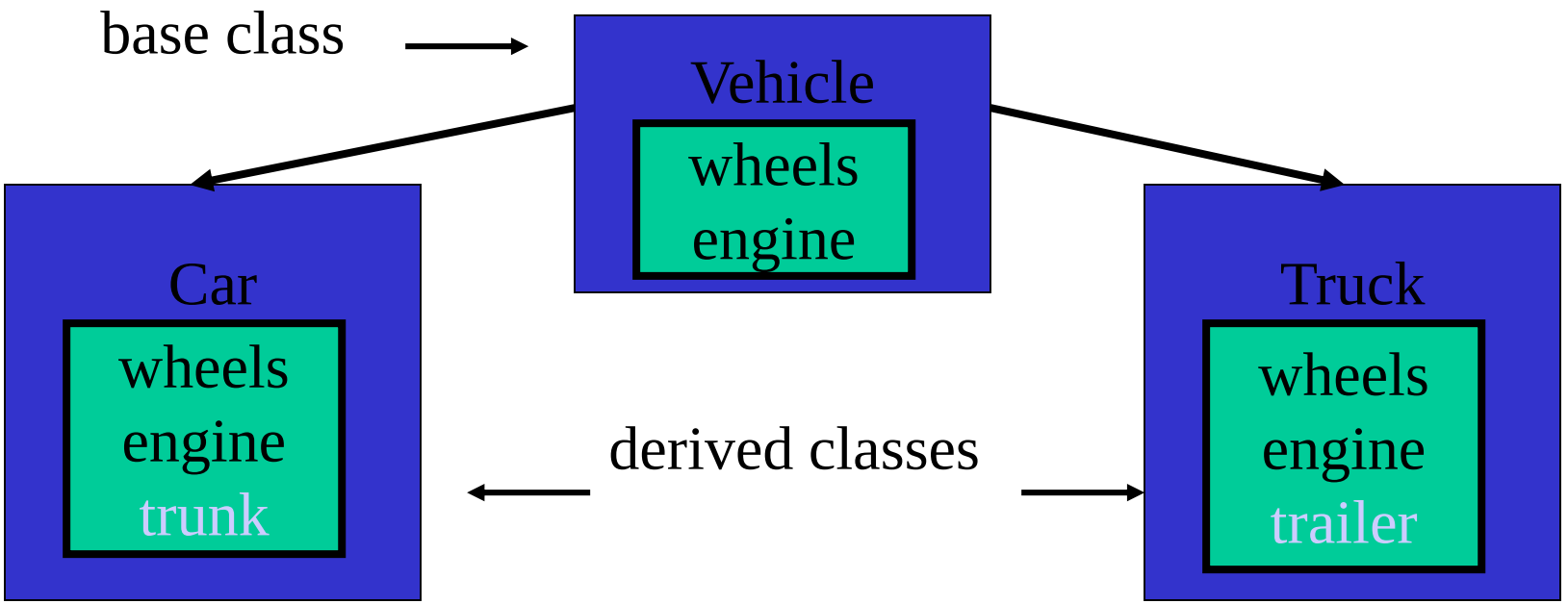
Contents

- Inheritance
- Polymorphisym

KẾ THỪA
(Inheritance)

Lớp cơ sở và lớp dẫn xuất

- Chúng ta có thể chia các lớp thành những lớp nhỏ hơn.
- Một lớp có thể được các lớp khác kế thừa các thuộc tính và phương thức
  - Lớp cơ sở (Base Class)
  - Lớp dẫn xuất (Derived Class)



Base Class Date

```
class Date
{
    private:
      int day, month, year;                    // member data
    public:                                  // member functions
       Date(int d, int m, int y);              // constructor
       Date operator+(int days);
       int operator-(Date &date);
       bool LeapYear();
       int DaysInMonth();
       void Display();
};
```

Derived Class DateTime

```
class DateTime : public Date
{
    private:
       int hours, minutes;                    // bổ sung thêm member
    data
    public:                                   // bổ sung thêm member functions
       DateTime(int d, int m, int y, int h, int mi);  // constructor
       void SetTime(int h, int m);
       void AddMinutes(int m);
       void AddHours(int h);
       void Display();     // overrides Date::Display()
};
```

Derived Class DateTime ...

```
DateTime::DateTime(int d, int m, int y, int h, int mi)
   : Date(d,m,y) , hours(h) , minutes(mi)                    // viện dẫn Date
     Constructor
{
}
void DateTime::Display()
{
    Date::Display();                                  // gọi Display() của lớp Date
    cout << " " << hours << ":" << minutes;
}
void DateTime::AddHours(int h)
{
   hours+=h;
   if (hours > 23)
     {
       hours-=24;
       *this++;                                                //sử dụng toán tử ++
     }
}
```

Derived Class DateTime ...

```
Date justdt(12,6,1998);                    // constructor Date
DateTime  dt(13,2,1997,20,15);             //constructor
    DateTime
dt=dt+5;
if (dt.LeapYear())
    cout << ”Date is in a leapyear” << endl;
else
        cout << ”Date is not in a leapyear” << endl;
dt.AddHours(2);
dt.Display();                          // Display() của lớp DateTime
justdt.Display();                          // Display() của lớp
    Date
```

Derived Class Constructors

```
DateTime::DateTime(int d, int m, int y, int h, int m1)
{
    Date(d,m,y) ;
    hours=h;
     minutes=m1;
}
DateTime dt(12,4,1999,17,30);
```

- Chương trình dịch sẽ tạo ra một đối tượng kiểu DateTime và sau đó gọi hàm tạo DateTime() để khởi tạo nó.
- Hàm tạo DateTime() sẽ gọi hàm tạo Date() để khởi tạo day, month, year
- Hàm tạo DateTime khởi tạo hours, minutes của chính nó.

Overriding Member Functions

```
class Date
{
  public:
     void Display();
};

class DateTime : public Date
{
public:
     void Display(){};                    // overrides Date::Display()
};
Int main(){
DateTime *a = new DateTime(); a->Display();
```

Overriding Member Function...

- Khi có các hàm cùng tên và cùng danh sách tham số trong base class và derived class thì hàm trong derived class sẽ được thực hiện.

```
DateTime dt(12,3,2001,12,30);
dt.Display();                    // Display() của DateTime
```

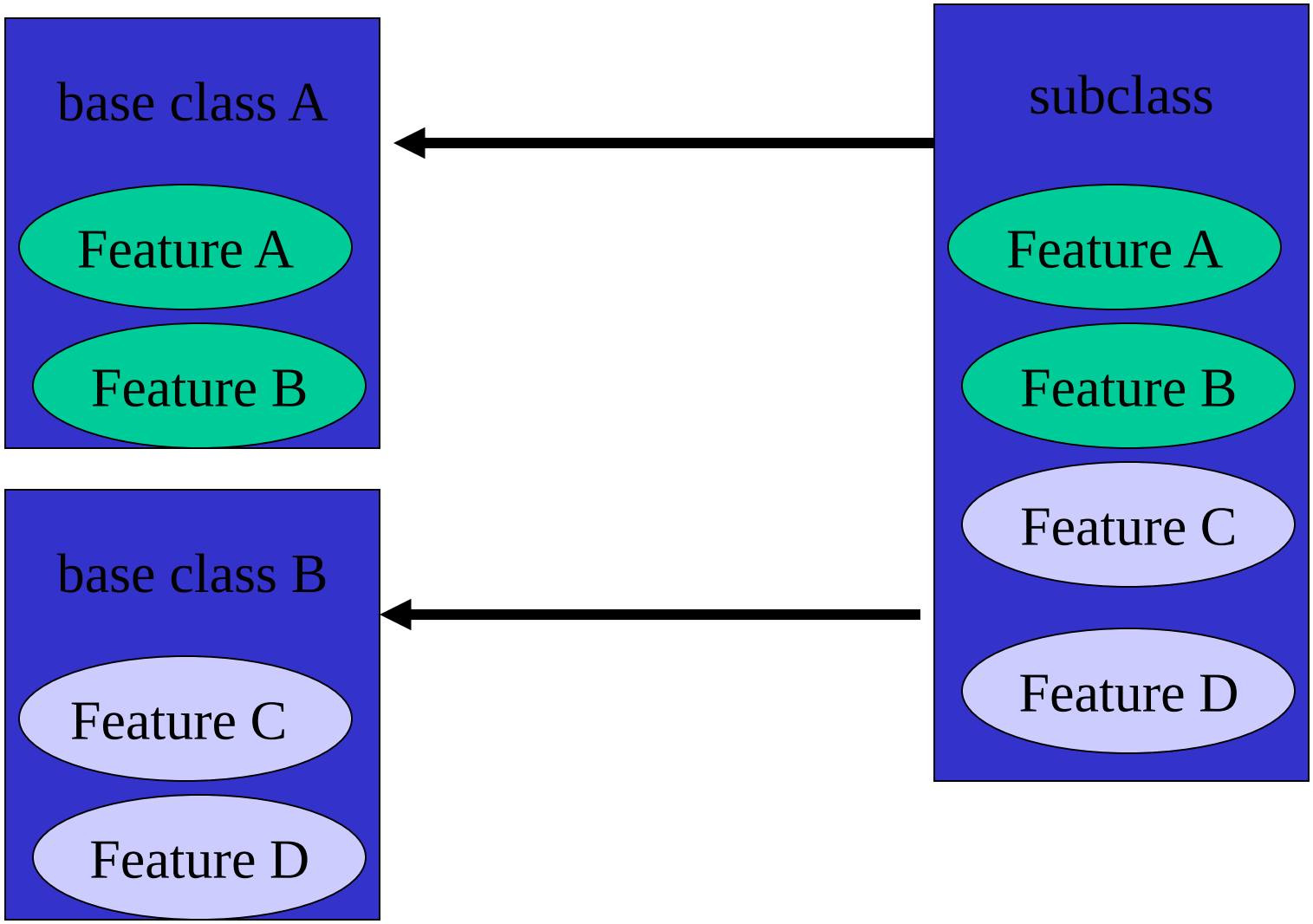
- Ta có thể xác định rõ cần sử dụng member function của lớp nào.

```
void DateTime::Display()
{
        Date::Display();
        cout << " " << hours << ":" << minutes;
}
```

Truy nhập vào các thành phần của lớp cơ sở

- Từ khoá (Access Specifier) private, protected và public xác định khả năng truy nhập tới các thành phần của lớp cơ sở.
  - private: chỉ được truy nhập trong lớp cơ sở
  - protected:chỉ được truy nhập trong lớp cơ sở và lớp dẫn xuất của nó
  - public: được truy nhập ở cả bên ngoài phần định nghĩa lớp.

Đa thừa kế


base class A
Feature A
Feature B
base class B
Feature C
Feature D
subclass
Feature A
Feature B
Feature C
Feature D

Đa thừa kế ...

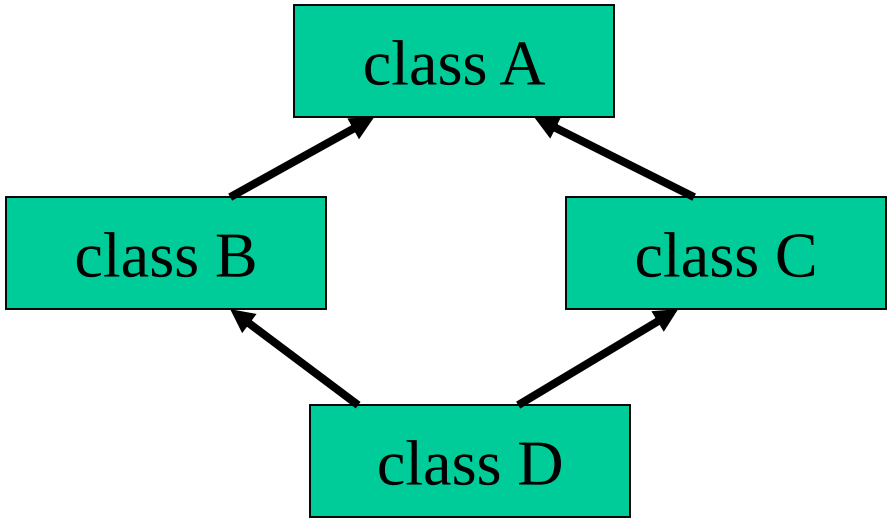
```
class Date
{
    private:
      int day, month, year;
    …
};
class Time
{
    private:
      int hours, minutes;
    …
};
class DateTime : public Date, public Time
{
    public:
      DateTime(int d, int m, int y, int h, int mi)  : Date(d,m,y), Time(h, mi)
     {};
    …
};
```

Nhập nhằng trong đa thừa kế

```
class Date
{
    void add(int days);
};
class Time
{
   void add(int minutes);
};
DateTime dt(13,2,1998,23,10);
dt.add(3);   // báo lỗi
dt.Date::add(4);  // sử dụng của lớp Date
dt.Time::add(5);  // sử dụng của lớp Time
```

Nhập nhằng trong đa thừa kế

```
class A {  public:  void F(); };
class B : public A { ...};
class C : public A { ...};
class D : public B, public C {};
D d;
d.F();   // không dịch
```



ĐA HÌNH
(Polymorphism)

Đa hình

- Đa hình là một hàm có nhiều hình thức thể hiện khác nhau tùy từng hoàn cảnh cụ thể
- Đa hình là một đặc trưng của ngôn ngữ lập trình hướng đối tượng
- Phân loại:
  - Đa hình tĩnh
  - Đa hình động

Đa hình tĩnh

- Overload một phương thức, tức là tạo ra nhiều phương thức có cùng tên nhưng khác nhau về danh sách tham số
- Overrid một phương thức, tức là tạo ra một phương thức trong lớp dẫn xuất có cùng prototype với một phương thức trong lớp cơ sở.

Ví dụ: Overloading Function

```
class Mammal
{
    public:
        void Move() {cout << "Mamal moves 1 step";}
        void Move(int d) {cout << "Mamal moves <<d<<"
    steps";}
};

int main()
{
    Mamal m;
    m.Speak();
    m.Speak(10);
}
```

Ví dụ: Overriding Function

```
class Mammal
{
    public:
        void Speak() {cout << "Mamal";}
};
class Dog : public Mammal
{
    public:
        void Speak() {cout << "Dog";}
};
int main()
{
    Mammal m;
    Dog d;
    m.Speak();          // Mamal
    d.Speak();          // Dog
}
```

Ẩn phương thức của lớp cơ sở

- Nếu lớp cơ sở có một phương thức bị chồng và lớp dẫn xuất lại override phương thức này, thì phương thức của lớp dẫn xuất sẽ ẩn tất cả các phương thức của lớp cơ sở có cùng tên với nó.

Ví dụ: Ẩn phương thức của lớp cơ sở

```
class Mammal
{
    public:
        void Move() {cout << "Mamal moves 1 step";}
        void Move(int d) {cout << "Mamal moves <<d<<" steps";}
};
class Dog : public Mammal
{
    public:
        void Move() {cout << "Dog moves 1 step";}
};
int main()
{
    Mamal m;
    Dog d;
    m.Move();
    m.Move(10);
    d.Move();
    d.Move(10);     // lỗi
}
```

Đa hình động

- Được thể hiện thông qua hàm ảo
- Từ khoá *virtual* xác định hàm thành phần của lớp cơ sở sẽ bị override bởi lớp dẫn xuất.
- Khi lớp cơ sở định nghĩa các hàm ảo thì C++ sẽ tìm kiếm hàm đó trong lớp dẫn xuất trước, sau đó mới đến lớp cơ sở.

```
#include <iostream.h>
class base {
    public:
        void a(void) { cout << "base::a called\n"; }
        virtual void b(void){ cout << "base::b called\n"; }
        virtual void c(void) { cout << "base::c called\n"; }
};
class derived: public base {
    public:
        void a(void) { cout << "derived::a called\n"; }
        void b(void) { cout << "derived::b called\n"; }
};
void do_base(base& a_base)
{
    cout << "Call functions in the base class\n";
    a_base.a();
    a_base.b();
    a_base.c();
}
main()
{
    derived a_derived;
    cout << "Calling functions in the derived class\n";
    a_derived.a();
    a_derived.b();
    a_derived.c();
    do_base(a_derived);
}
```

Ví dụ: Hàm ảo

```
void do_base(base& a_base)
{
      cout << "Call functions in the base class\n";
      a_base.a();
      a_base.b();
      a_base.c();
}
main()
{
      derived a_derived;
      cout << "Calling functions in the derived class\n";
      a_derived.a();
      a_derived.b();
      a_derived.c();
      do_base(a_derived);
      return (0);
}
```

Khi nào sử dụng hàm ảo?

- Lớp Parent và Child cùng có phương thức f
- Khai báo một con trỏ thuộc kiểu của lớp Parent
  - Parent* p;
- Con trỏ này trỏ đến đối tượng của lớp Child
  - p = new Child;
- Sau đó, thực hiện lời gọi
  - p->f;
- Kết quả: f của lớp Parent sẽ được viện dẫn
- Nếu f được khai báo là hàm ảo trong lớp Parent thì f của lớp Child sẽ được viện dẫn.

Ví dụ: Không sử dụng hàm ảo

```
class Mammal
{
    public:
        void Move() {cout << "Mammal moves 1 step";}
};
class Dog : public Mammal
{
    public:
        void Move() {cout << "Dog moves 1 step";}
};
int main()
{
    Mammal* p = new Dog();
    p->Move(); // "Mammal moves 1 step"
}
```

Ví dụ: Sử dụng hàm ảo

```
class Mammal
{
    public:
        void virtual Move() {cout << “Mammal moves 1 step”;}
};
class Dog : public Mammal
{
    public:
        void Move() {cout << “Dog moves 1 step”;}
};
int main()
{
    Mamal* p = new Dog();
    p->Move(); // “Dog moves 1 step”
}
```

Hàm thuần ảo

- Nếu khai báo hàm ảo như sau:

  virtual void send_it(*void*) = 0;

  - "=0" có nghĩa là hàm thuần ảo (pure virtual function). Tức là, nó sẽ không được gọi một cách trực tiếp.
  - Hàm thuần ảo **phải được** overload trong subclass
  - Hàm ảo **có thể được** overload trong subclass

Lớp trừu tượng (abstract class)

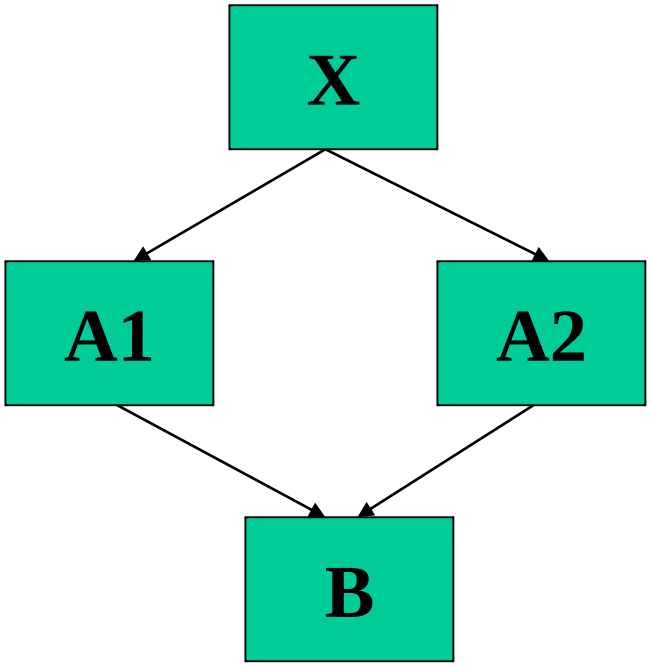
- Hàm thuần ảo được khai báo trong lớp sẽ làm cho lớp đó trở thành lớp cơ sở trừu tượng.
- Lớp cơ sở trừu tượng là lớp cơ sở không có đối tượng nào và chỉ sử dụng để cho các lớp khác kế thừa.
- Một lớp chỉ đóng vai trò là lớp cơ sở cho các lớp khác và không có đối tượng cụ thể của nó được tạo ra thì gọi là lớp trừu tượng.

Ví dụ: hàm thuần ảo

```
class Mammal
{
        public:
                virtual void  Move() = 0;
};
class Dog : public Mammal
{
        public:
                void Move() {cout << "Dog moves 1 step";}
};
void main()
{
        Dog p;
        p.Move();                   // “Dog moves 1 step”
        Mammal m;                   // ”Lỗi”
        m.Move();
}
```

Lớp cơ sở ảo

Nhập nhằng trong đa kế thừa



- Lớp B sẽ có hai bản sao của tất cả các thành phần từ lớp X.
- Khi gọi đến một trong những thành phần này từ lớp B, chương trình dịch sẽ thông báo lỗi.

Giải quyết xung đột

- Gọi tường minh
  - Ví dụ lớp X có phương thức x được thừa kế
  - Lời gọi x từ một đối tượng của lớp B

```
B b;
b.A1 :: x;
b.A2 :: x;
```

- Sử dụng lớp cơ sở ảo

Lớp cơ sở ảo

- Lớp cơ sở ảo đảm bảo trong lớp dẫn xuất chỉ tạo ra một bản sao của các thành phần được thừa kế từ lớp cơ sở.

```
#include <iostream.h>
class Automobile
{
    private:
        char *Manufacturer;
        double Price;
    public:
        Automobile(){}
        void GetPrice()
        {
            cin >> Price;
        }
};
class Car : public virtual Automobile
{
    private:
        int Price;
    public:
        Car(){}
        //...
};
class SportsVehicle : public virtual Automobile
{
    private:
        int Price;
    public:
        SportsVehicle(){}
        //...
};
class SportsCar : public Car, public SportsVehicle
{};
void main()
{
    SportsCar SC;
    SC.GetPrice();
}
```

Lớp cơ sở ảo

- Hàm tạo của lớp cơ sở chỉ được gọi trong hàm tạo của lớp dẫn xuất trực tiếp từ nó.
- Hàm tạo của lớp cơ sở ảo thì được gọi ở tất cả các lớp dẫn xuất nó.
- Quy tắc như sau:
  - Hàm tạo của lớp cơ sở ảo được gọi đầu tiên
  - Tiếp theo đó là hàm tạo của các lớp dẫn xuất trực tiếp
  - ...

```
class Temporary : public Employee
{
    int DaysWorked;
public:
    Temporary(){}
    Temporary(char *EName, int Days) : Employee(EName)
    {
        DaysWorked = Days;
    }
};
class Clerk : public Employee
{
    char *Location;
public:
    Clerk(){}
    Clerk(char *EName, char *City) : Employee(EName)
    {
        Location = City;
    }
};
class TempClerk : public Temporary, public Clerk
{
    int Salary;
public:
    TempClerk(){}
    TempClerk(char *EName, char *City, int Days, int Sal)
              : Clerk(EName, City), Temporary(EName, Days)
    {
        Salary = Sal;
    }
```

Questions and Answers