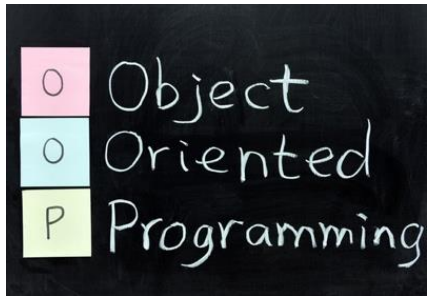




Môn học

Lập trình hướng đối tượng

Đỗ Phúc Hảo

Khoa Công nghệ thông tin




Nội dung

❖Chương 5: Tính thừa kế

- Khái niệm
- Các kiểu kế thừa
- Constructor và destructor trong kế thừa
- Lớp trừu tượng (abstract class)
- Tính đa hình (polymorphism)
- Bài tập




1. Khái niệm

❖Để quản lý nhân sự của công ty, ta có thể định nghĩa các lớp tương ứng với vị trí làm việc của công ty (Worker: công nhân, Manager: quản lý, Director: giám đốc)

```
class Worker {
    private:
        string name;
        string salary;
        int level;
    public:
        string getName(){};
        void pay(){};
        void doWork(){};
};
```

```
class Manager {
    private:
        string name;
        string salary;
        int dept;
    public:
        string getName(){};
        void pay(){};
        void doWork(){};
};
```

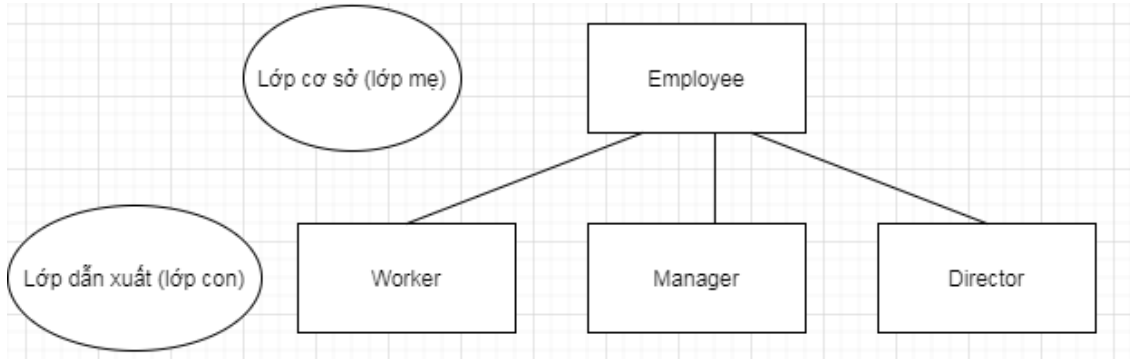
```
class Director {
    private:
        string name;
        string salary;
    public:
        string getName(){};
        void pay(){};
        void doWork(){};
};
```




1. Khái niệm (t)

❖Cả 3 lớp trên đều có những biến và hàm giống hệt nhau về nội dung → tạo ra một lớp **Employee (nhân viên)** chứa các thông tin chung đó để sử dụng lại

- Sử dụng lại code
- Giảm số code cần viết
- Dễ bảo trì, sửa đổi về sau
- Rõ ràng hơn về mặt logic.




```
class Worker {
    private:
        string name;
        string salary;
        int level;
    public:
        string getName(){};
        void pay(){};
        void doWork(){};
};
```

```
class Manager {
    private:
        string name;
        string salary;
        int dept;
    public:
        string getName(){};
        void pay(){};
        void doWork(){};
};
```

```
class Director {
    private:
        string name;
        string salary;
    public:
        string getName(){};
        void pay(){};
        void doWork(){};
};
```




1. Khái niệm (t)

- **Kế thừa** là việc tái sử dụng lại một số thuộc tính, phương thức đã có sẵn từ lớp cơ sở
- Kế thừa cho phép các lớp con sử dụng các biến và phương thức của lớp mẹ như là của nó, *trừ các biến và phương thức private*




2. Các kiểu kế thừa

- Kế thừa public
- Kế thừa private
- Kế thừa protected




2. Các kiểu kế thừa (t)

- Cột: các kiểu thừa kế
- Hàng: phạm vi biến/ phương thức trong lớp mẹ (lớp cơ sở)
- Kết quả: phạm vi biến/phương thức trong lớp con (lớp dẫn xuất)

		Kiểu kế thừa		
		private	**protected**	**public**
Phạm vi	**private**	private	private	private
	protected	private	protected	protected
	public	private	protected	public




2. Các kiểu kế thừa – Kế thừa public

```
class A {
        protected:
                int protectedA;
        private:
                int privateA;
        public:
                A(int v1 = 0, int v2 = 0);
                void showValue();
};
```

```
A::A(int v1, int v2){
        this->privateA = v1;
        this->protectedA = v2;
}
void A::showValue(){
        cout<< privateA + protectedA ;
}
```

Định nghĩa class A




2. Các kiểu kế thừa – Kế thừa public (t)

```
class B : public A {
        private:
                int privateB;
        protected:
                int protectedB;
        public:
                B();
                void show();
};  void B::show(){
        showValue(); // public function
        protectedA = 4; // OK (protected variable)
        //privateA = 2; // ERROR
}
```

class B
public từ A

```
A a(1,2);
//a.protectedA; // Error. Biến protected của A.
a.showValue();
/* ke thua public */
B b;
b.show();
b.showValue();
//b.protectedA = 0; // error (no la protect cua lop co so)
//b.protectedB = 9; // error (no la bien protected cua lop B)
```

Hàm Main




2. Các kiểu kế thừa – Kế thừa private

```
class C : private A {
        private:
                int privateC;
        protected:
                int protectedC;
        public:
                C();
                void show();
};
void C::show(){
        showValue(); // public function
        protectedA = 4; // OK (protected variable)
        //privateA = 2; // ERROR
}
```

class C
private từ A

```
A a(1,2);
//a.protectedA; // Error. Biến protected của A.
a.showValue();
/* ke thua private */
C c;
c.show();
//c.showValue(); // Error: Ke thua private nen ham public(showValue) tu lop A --> Qua lop C (private)
//c.protectedA = 0; // error (no la protect cua lop co so)
//c.protectedC = 9; // error (no la bien protected cua lop C)
```

Hàm Main




2. Các kiểu kế thừa – Kế thừa protected

```
class D : protected A {
        private:
                int privateD;
        protected:
                int protectedD;
        public:
                D();
                void show();
};
void D::show(){
        showValue(); // public function
        protectedA = 4; // OK (protected variable)
        //privateA = 2; // ERROR
}
```

class D protected từ A

```
A a(1,2);
//a.protectedA; // Error. Biến protected của A.
a.showValue();
/* ke thua protected */
D d;
d.show();
//d.showValue(); // Error: Ke thua protected nen ham public(showValue) tu lop A --> Qua lop D (protected)
//d.protectedA = 0; // error (no la protect cua lop co so)
//d.protectedD = 9; // error (no la bien protected cua lop D)
```

Hàm Main




3. Constructor và destructor trong kế thừa

- Constructor và destructor không được các lớp con thừa kế.
- Mỗi constructor của lớp dẫn xuất phải gọi một constructor của lớp mẹ, nếu không sẽ được ngầm hiểu là gọi constructor mặc định
- Destructor của các lớp sẽ được gọi tự động theo thứ tự ngược từ lớp dẫn xuất tới lớp cơ sở




3. Constructor và destructor trong kế thừa(t)

Định nghĩa class A

```
class A {
        private:
                int privateA;
        public:
                A();
                A(int v);
                void showValue();
                ~A();
};
A::~A(){
        cout << "ham huy A()" << endl;
}
```

```
A::A(){
        this->privateA = -1;
}
void A::showValue(){
        cout << privateA << endl;
}
A::A(int v){
        this->privateA = v;
}
```




3. Constructor và destructor trong kế thừa(t)

Class B public từ A

```
class B : public A {
        private:
                int privateB;
        public:
                B();
                B(int vA, int vB);
                void show();
                ~B();
};
B::~B(){
        cout << "Ham huy B()" << endl;
}
```

```
B::B(int vA, int vB):A(vA){
        this->privateB = vB;
}
B::B(){
}
void B::show(){
        cout << "show () B" << endl;
        cout << "call showValue class A" << endl;
        showValue();
        cout << "value privateB: " << privateB <<
endl;
}
```




3. Constructor và destructor trong kế thừa(t)

```
int main(){
        B b (1,2);
        b.show();
        return 0;
}
```

Hàm main

```
D:\learning\iTeaching\_OOP\_NEW\2018-2019\Coding\prepare\c5\C5_Demo02.exe
show () B
call showValue class A
1
value privateB: 2
Ham huy B()
ham huy A()

--------------------------------
Process exited after 0.01512 seconds with return value 0
Press any key to continue . . .
```




4. Lớp trừu tượng (abstract class)

- Phương thức ảo thuần tuý (pure virtual method): là phương thức được khai báo nhưng chưa được định nghĩa → cần được định nghĩa trong các lớp dẫn xuất
- Lớp trừu tượng là lớp có phương thức ảo thuần tuý
  - Không thể tạo được đối tượng từ lớp trừu tượng




4. Lớp trừu tượng (abstract class) (t)

```
class A {
        private:
                int privateA;
        public:
                virtual void showValue() = 0; //
luc nay, lop A tro thanh lop abstract.
};
// class B
class B : public A {
        private:
                int privateB;
        public:
                void show();
                virtual void showValue();
};
```

```
void B::showValue(){
        cout << "ShowValue() of class B" << endl;
}

void B::show(){
        cout << "show class B()" << endl;
        showValue();
}
```

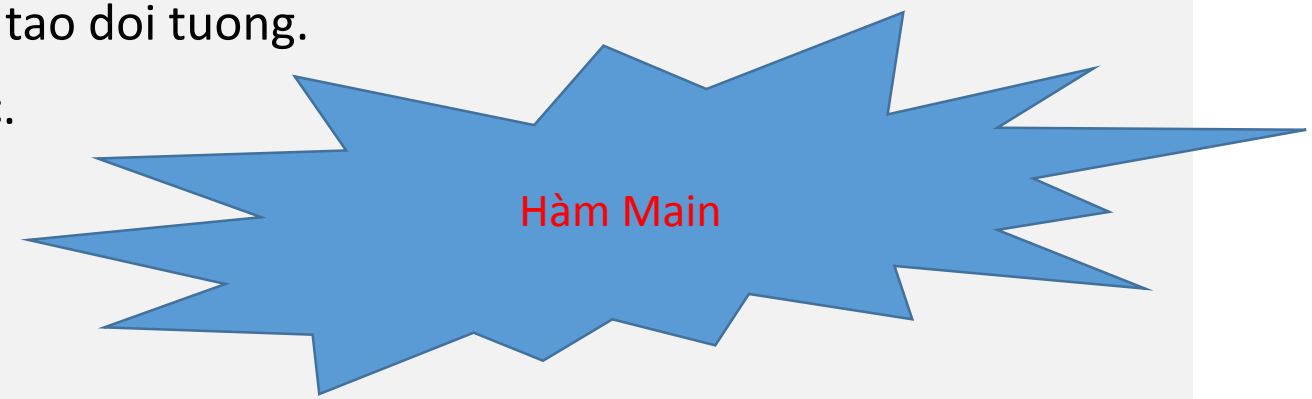
Định nghĩa class A và class B kế thừa public từ A




4. Lớp trừu tượng (abstract class) (t)

```
B b;
//A a = b; //error:  luc nay lop A la lop abstract (nen khong duoc tao object)
b.show();
//B b1 = a; //error:  B la lop con, con a la doi tuong cua lop me (A) nen khong the gan cho con (B)

//A a; // error:  no la abstract nen khong the tao doi tuong.
A *a = &b; // nhung tao mot con tro thi duoc.
a->showValue();
```






5. Tính đa hình (polymorphism)

- Thừa kế và định nghĩa các hàm ảo giúp quản lý đối tượng dễ dàng hơn: có thể gọi đúng phương thức mà không cần quan tâm tới lớp thực sự của nó là gì!!!!
- Là một trong 3 đặc điểm quan trọng của OOP: đóng gói (encapsulation), kế thừa (inheritance), đa hình (polymorphism)




5. Tính đa hình (polymorphism) (t)

```
class C : public A {
        private:
                int privateC;
        public:
                void show();
                virtual void showValue();
};
void C::showValue(){
        cout << "ShowValue() of class C" << endl;
}
void C::show(){
        cout << "show class C()" << endl;
        showValue();
}
```

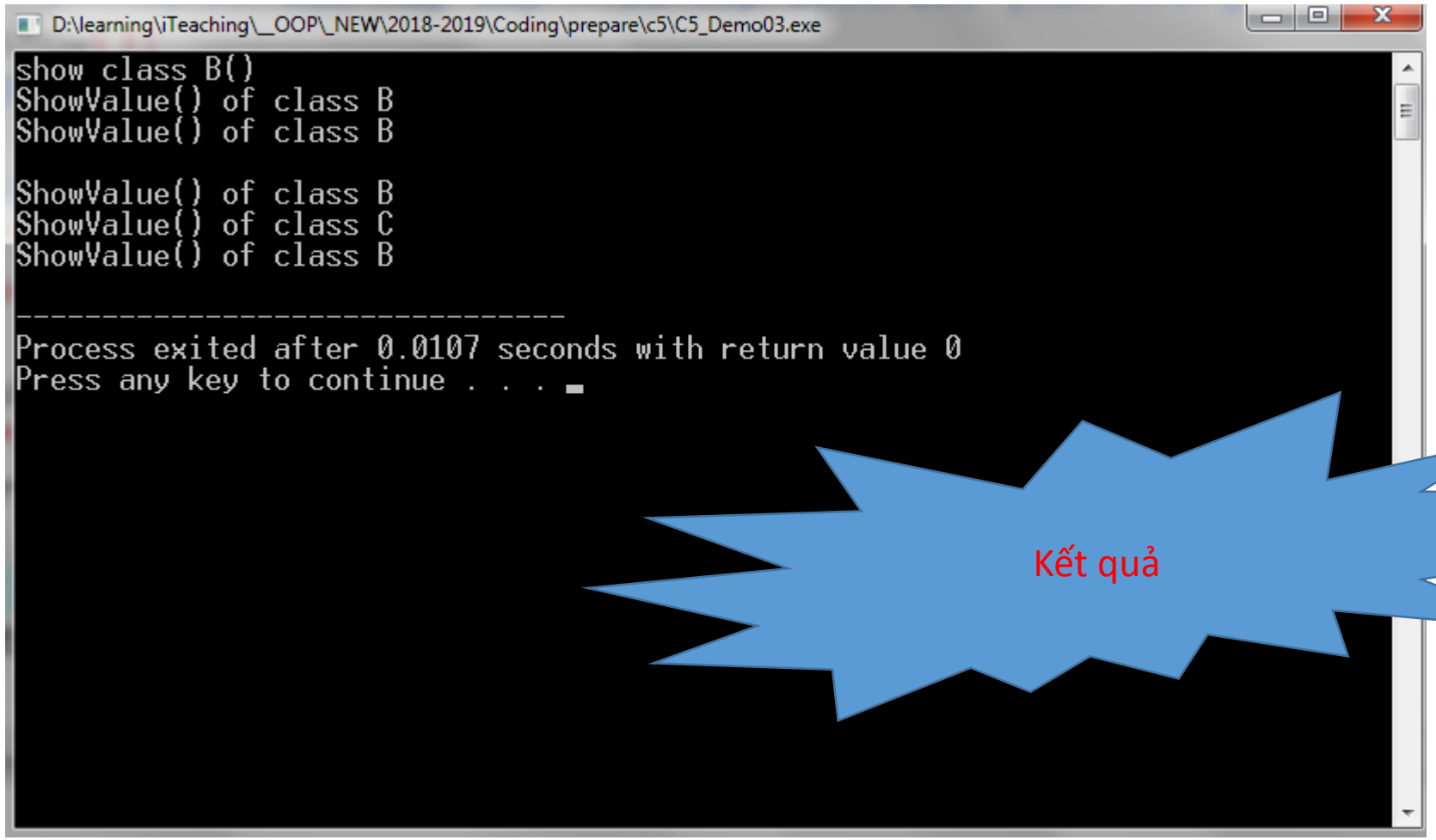
Class C

```
A *arr[3] = {new B(), new C(), new B()}; // Tinh da hinh.
for (int i = 0; i < 3; i++){
        arr[i]->showValue();
}
```

Hàm Main - tính đa hình




5. Tính đa hình (polymorphism) (t)



Kết quả




6. Bài tập

- Câu 1: Viết các lớp Shape (trừu tượng) và Circle, Square, Rectangle, Ellipse, Sphere. Hãy thiết kế việc kế thừa sao cho hợp lý.
- Câu 2: Hoàn tất các lớp Employee, Worker, Manager, Director và viết một chương trình thử.




Thank you!

Any questions?