

第6讲 类的继承



目录

CONTENTS



1 继承与派生的概念

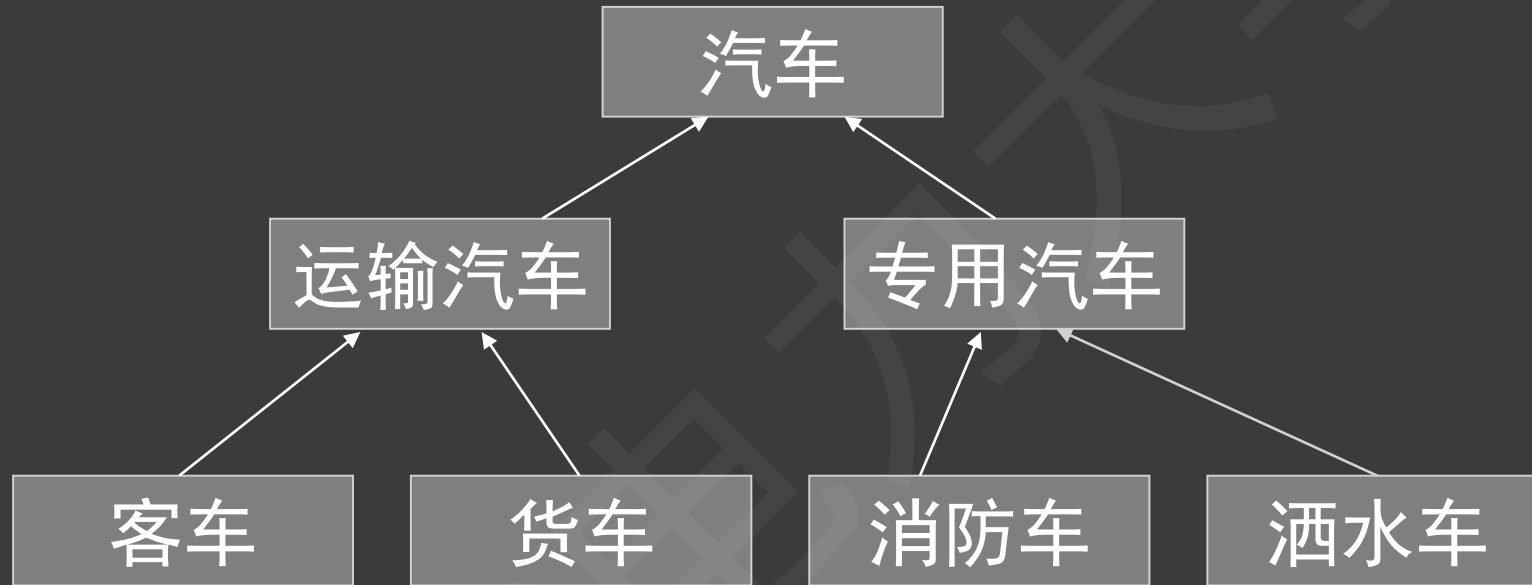
2 派生类的构造和析构函数

3 基类成员的访问属性

4 同名成员

5 访问声明*

1 继承与派生的概念



简单的汽车分类图

类的继承

假设有两个类---类A 和类B，若类B继承类A

A (基类或父类)



B (派生类或子类)

则类B具有类A的基本特性（包括数据和程序代码）。

类的继承就是新的类从**已有类**那里得到已有的特性。

继承机制的作用

- ◎ 类的继承机制使程序员只需在**已有类**的基础上，通过增加少量代码或修改少量代码的方法**得到**新的类，从而较好地解决了**代码重用**的问题。
- ◎ 由已有类产生新类时，新类便包含了已有类的特征，同时也可以加入自己的新特性。

基类的意义

- 基类描述了**共性**。所有从基类派生出来的类都继承了基类的功能。
- 在面向对象的设计过程中，设计者先寻找并提取出构成所需基类的共性，然后再通过继承从基类派生出超出基类功能的定制派生类。
- 不适合于派生类的基类成员可以在派生类中重新定义。

派生类的构成

- ◎ 如果类B是类A的**派生类**,那么,在构造类B的时候:
 1. **不必**重新描述与类A相同的特性,**只需**让它继承类A,即可拥有类A的特性;
 2. **增加**类B与基类A不同的那些特性。

派生类的特性 { **继承来的特性**
新增加的特性

2 派生类的声明

```
class CPerson{  
    string name;  
    string id_number;  
    int age;
```

```
public:
```

```
    void show();
```

```
};
```

```
class CStudent {  
    string name;  
    string id_number;  
    int age;
```

```
    int score;
```

```
public:
```

```
    void show();
```

```
};
```

相同部分

基类与派生类

基类名

```
class CPerson{  
    string m_strName;  
    string m_strId;  
    int age;  
public:  
    void Show();  
};
```

派生类名

继承方式

```
class CStudent :public CPerson  
{  
    int m_nScore;  
public:  
    void Study();  
};
```

被继承的部分

新增加的部分

单一继承派生类

什么是单一继承派生类？

即派生类只有1个基类。

一个已经定义类

声明一个派生类的一般格式为：

```
class 派生类名:派生方式（访问控制） 基类
{
    // 派生类新增的数据成员和成员函数
};
```

要声明的新类名

“派生方式” 可以是关键字
private、protected或public

继承方式

① class CStudent : **public** CPerson{
 // ...
};

公有继承

② class CStudent : **private** CPerson{
 // ...
};

私有继承

③ class CStudent : **protected** CPerson{
 // ...
};

保护继承

默认的继承方式为: **private**

如何构造一个派生类

(1) 派生类从基类**继承**成员

- 在C++的类继承中，派生类把基类的**全部成员**(除构造函数和析构函数之外)继承过来。

如何构造一个派生类

(2) **调整**从基类继承来的成员，对基类成员的调整包括两个方面：

- 一方面是改变基类成员在派生类中的**访问属性**，这主要是通过派生类声明时的继承方式来控制的。
- 另一方面是派生类可以对基类的成员进行**重定义**。

如何构造一个派生类

(3) 在派生类中增加新的成员

- 可以根据实际情况的需要，设计需要**增加**的数据成员和成员函数，来实现新增功能。

基类中公有成员的访问规则

```
class Base{
```

```
public:
```

```
    int y;
```

```
    void Show()
```

```
    { cout<<y; }
```

```
};
```

```
class Derived : public Base{
```

```
    public:
```

```
        void Print( )
```

```
        { cout<<y; }
```

```
        ...
```

```
};
```

```
Base obj;
```

```
cout<<obj.y;
```

基类中的公有成员

基类的成员函数可以访问

派生类的成员函数可以访问

(在类外)基类的对象可以访问

基类中私有成员的访问规则

```
class Base{  
    private:  
        int y;  
        void Show()  
        { cout<<y; }  
};  
  
class Derived : public Base{  
    public:  
        void Print( )  
        { cout<<y; }  
        ...  
};  
...  
Base obj;  
cout<<obj.y;
```

基类中的私有成员

基类的成员函数可以访问

派生类的成员函数不能直接访问

(在类外)基类的对象不能直接访问

基类中保护成员的访问规则

```
class Base{  
    protected:
```

```
        int y;
```

```
        void Show()
```

```
        { cout<<y; }
```

```
};
```

```
class Derived : public Base{
```

```
    public:
```

```
        void Print( )
```

```
        { cout<<y; }
```

```
        ...
```

```
};
```

```
...
```

```
Base obj;
```

```
cout<<obj.y;
```

基类中的保护成员

基类的成员函数可以访问

派生类的成员函数可以访问

(在类外)基类的对象不能直接访问

说明

- ◎ 派生类继承了基类的**所有成员**（但是不包括基类的构造函数和析构函数），并添加了自己的新成员。
- ◎ **派生方式**用于规定基类成员在派生类中的访问权限。包括派生类的成员和对象对其访问权限。
- ◎ 派生方式有三种：private、protected或public。

基类成员在派生类中的访问属性

- ◎ 派生类继承的基类成员的访问属性在派生过程中是可以调整的，由派生方式控制。

在基类中的 访问属性 派生方式	public	protected	private
public	public	protected	不可直接访问
protected	protected	protected	不可直接访问
private	private	private	不可直接访问

protected成员的作用

基类中的私有成员不允许

- 外部函数
- 派生类的成员函数

访问。

在基类设计中，总要为它的私有数据成员提供公有成员函数，以便外部函数和派生类的成员函数的访问，**函数**的调用开销大。

protected成员

为了让派生类的成员函数**直接访问**基类私有数据成员，提供了具有另一种访问属性的成员---protected成员。

保护成员的访问属性是：

- **外部函数不能访问**
- **派生类中的成员函数可直接访问**

保护成员的访问属性

- 在公有派生的情况下，基类中的保护成员在派生类中也都是**保护成员**。
- 在私有派生的情况下，基类中的所有保护成员在派生类中也都变成**私有**的。

一个例子

```
#include <iostream>
using namespace std;
class CPoint{
    float x, y;
public:
    void SetPoint ( float i , float j ) { x = i ; y = j ; }
    void ShowPoint ( ) { cout << x << "," << y<<endl ; }
};
```

```
class CCircle: public CPoint
{
    float radius;
public:
    void SetRadius ( float r ) { radius=r ; }
    void ShowCircle() { cout << radius << endl ; }
};

void main()
{
    CCircle c;
    c.SetPoint(1.0,2.0);
    c.ShowPoint();    //直接访问point的公有成员
    c.SetRadius(6);
    c.ShowCircle();   //访问派生类自己的公有成员
}
```



```
class CCircle: private CPoint{
    float radius;
public:
    void SetRadius ( float r ) { radius = r ; }
    void ShowCircle( ) { cout << radius << endl ; }
};

void main()
{
    CCircle c;
    c.SetPoint(1.0,2.0); //非法
    c.ShowPoint(); //直接访问point的公有成员 非法
    c.SetRadius(6);
    c.ShowCircle(); //访问派生类自己的公有成员 正确
    CPoint p;
    p.SetPoint(1.0,2.0);
    p.ShowPoint(); //基类对象访问point的公有成员 正确
}
```

```
class CCircle: protected CPoint{
    float radius;
public:
    void SetRadius ( float r ) { radius = r ; }
    void ShowCircle( ) { cout << radius << endl ; }
};

void main()
{
    CCircle c;
    c.SetPoint(1.0,2.0); //非法
    c.ShowPoint(); //直接访问point的公有成员 非法
    c.SetRadius(6);
    c.ShowCircle(); //访问派生类自己的公有成员 正确
    CPoint p;
    p.SetPoint(1.0,2.0);
    p.ShowPoint(); //基类对象访问point的公有成员 正确
}
```

此题未设置答案，请点击右侧设置按钮

在main函数中通过派生类对象只能访问基类中的

- ☐ A 公有继承的公有成员
- ☐ B 公有继承的私有成员
- ☐ C 公有继承的保护成员
- ☐ D 私有继承的公有成员

提交

3 派生类的构造函数和析构函数

- ◎ 派生类构造函数和析构函数的构造规则
- ◎ 派生类构造函数和析构函数的调用顺序
- ◎ 含有对象成员的派生类的构造函数

派生类不能继承的内容

- ◎ 基类的构造函数
- ◎ 基类的析构函数
- ◎ 基类中的友元函数

3.1 构造规则

1. 派生类的构造函数

当基类的构造函数没有参数，或没有定义构造函数时，派生类**可以**不向基类传递参数，甚至可以不定义构造函数（默认）。

当基类含有带参数的构造函数时，派生类**必须**定义构造函数，以提供把参数传递给基类构造函数的途径。

派生类不能继承构造函数，但是可以通过**初始化列表**调用基类的构造函数。

派生类构造函数的格式

在C++中，派生类构造函数的一般格式为：

```
派生类名(参数总表)：基类名(参数表)  
{  
    派生类新增成员的初始化语句  
}
```

其中基类构造函数的参数表，通常来源于派生类构造函数的参数表，也可以用常数值。

```
#include<iostream>
using namespace std;
class Base{                                //声明基类
    private:
        int i;
    public:
        Base(int n)                        //基类的构造函数
        { cout<<"Base类对象构造中"<<endl;
          i=n;
        }
        ~Base()                           //基类的析构函数
        { cout<<"析构Base类对象"<<endl; }
        void Displ()
        { cout<<i<<endl; }
};
```



```
class Derived:public Base{           //声明公有派生类
private:
    int j;
public:
    Derived(int n,int m):Base(m)      //派生类构造函数时
    {
        cout<<" Derived类对象构造中"<<endl;
        j=n;
    }
    ~ Derived()                      //派生类的析构函数
    { cout<<"析构Derived类对象"<<endl; }
    void DispJ()
    { cout<<j<<endl; }
};
```

```
int main()
{
    Derived obj(50,60);
    obj.displ();
    obj.dispJ();
    return 0;
}
```

运行结果如下：
Base类对象构造中
Derived类对象构造中
60
50
析构Derived类对象
析构Base类对象

说明

(1)可以将派生类构造函数定义在**类的外部**，而在类体内只写该函数的声明。

例如，在派生类中声明构造函数的原型：

Derived(int n,int m);

而在类的外部定义派生类的构造函数：

Derived (int n,int m):Base(m)

{

cout<<"Derived类对象构造中\n"<<endl;

j=n;

}

在此，要列出基类构造函数名及其参数表(即:Base(m))。

说明

(2) 若基类使用不带参数的构造函数或带缺省参数的构造函数，则在派生类中定义构造函数时可略去“:基类(参数表)”

```
class Base{
```

```
...
```

```
Base() {...}
```

```
...
```

```
};
```

```
class Derived:public Base{
```

```
...
```

```
Derived(){...}
```

```
...
```


```
};
```

说明

(3) 当基类的构造函数带有至少一个参数时，它的派生类必须定义构造函数，并缀上“: **基类(参数表)**”，甚至所定义的派生类构造函数的函数体可能为空，仅仅起参数的传递作用。

```
class Base{  
    ...  
    Base(int x ){...}  
    ...  
};
```

```
class Derived:public Base{  
    ...  
    Derived( int x,int y):Base(x)  
    { ... }  
};
```

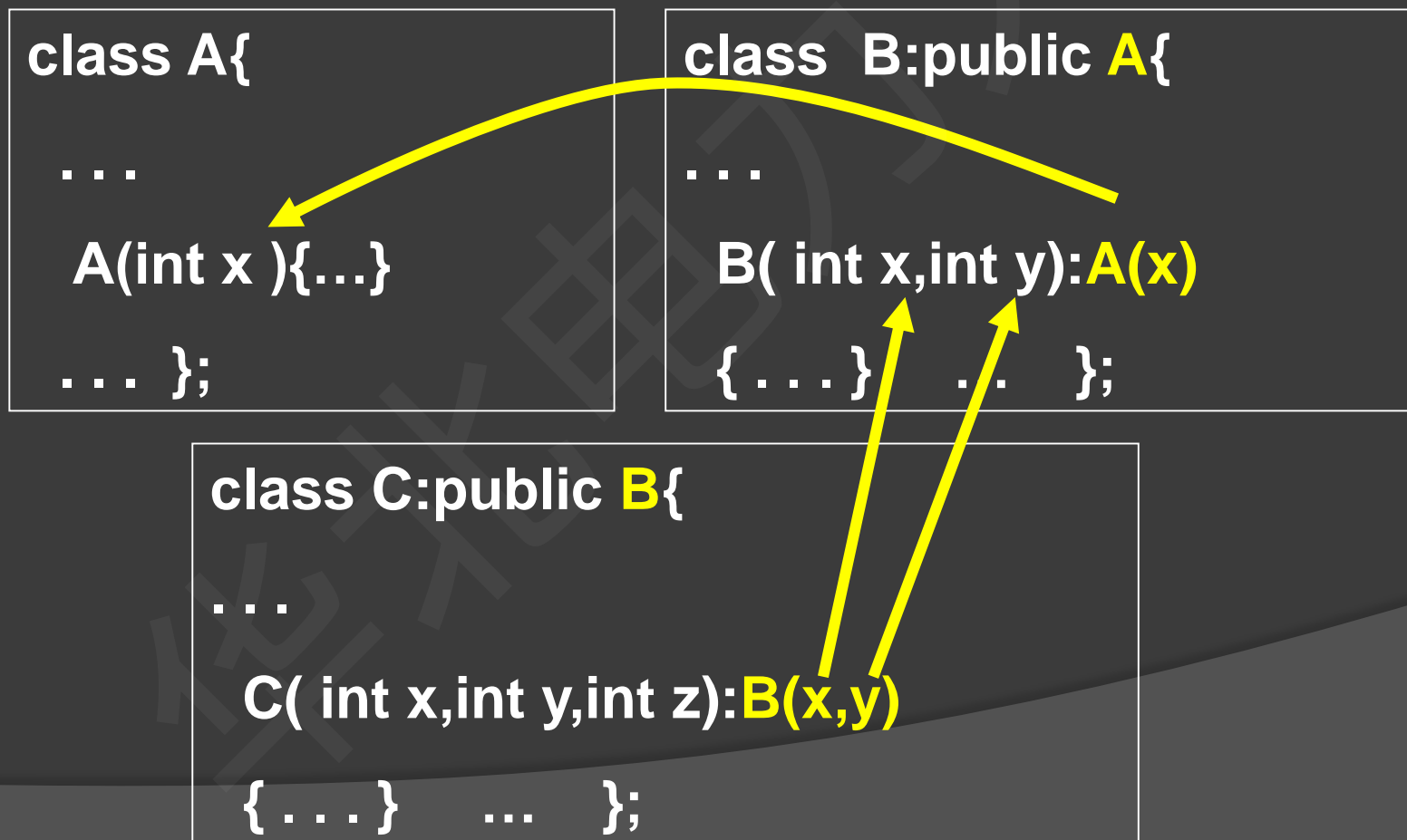


```
class Derived:public Base{           //声明公有派生类
private:
    int j;
public:
    Derived(int n,int m):Base(m)      //派生类构造函数时
    {
        cout<<" Derived类对象构造中"<<endl;
        j=n;
    }
    ~ Derived()                      //派生类的析构函数
    { cout<<"析构Derived类对象"<<endl; }
    void DispJ()
    { cout<<j<<endl; }
};
```

在这个程序段中,派生类Derived不使用参数m,m只能被传递给了基类构造函数。

说明

(4)如果派生类的基类也是一个派生类，每个派生类只需负责其**直接基类**的构造，依次上溯。



派生类的析构函数

- 在派生类中可以根据需要定义自己的析构函数，用来对派生类中的**所增加**的成员进行清理工作，基类的清理工作仍然由基类的析构函数负责。
- 在执行派生类的析构函数时，系统会**自动**调用基类的析构函数，对基类的对象进行清理。


```
class Base{
public:
    ...
    ~Base()
    { cout<<"基类的析构函数\n";}
};
class Derived:public Base {
public:
    ...
    ~Derived()
    { cout<<"派生类的析构函数\n";}
};
int main( )
{ Derived op; return 0; }
```

析构函数是不带参数的,在派生类中是否要自定义析构函数与它所属基类的析构函数无关,它们各自是独立的。

3.2 构造函数和析构函数的调用顺序

通常情况下:

当**创建**派生类对象时:

基类的构造函数→派生类的构造函数;

当**撤消**派生类对象时:

派生类的析构函数→基类的析构函数。

```
#include <iostream>
using namespace std;
class Base{                                //声明基类
public:
    Base()                                //基类的构造函数
    { cout<<"Base类对象构造中"<<endl; }
    ~Base()                                //基类的析构函数
    { cout<<"析构Base类对象"<<endl; }
};
class Derived:public Base{                 //公有派生类
public:
    Derived()                             //派生类的构造函数
    { cout<<"Derived类对象构造中"<<endl;}
    ~Derived()                             //派生类的析构函数
    { cout<<"析构Derived类对象"<< endl; }
};
int main()
{ Derived d;
  return 0;
}
```

运行结果如下：
Base类对象构造中
Derived类对象构造中
析构Derived类对象
析构Base类对象

3.3 含有对象成员的构造函数

```
class B {  
    int x;  
    public:  
    B(int i); //基类的构造函数  
    //...  
};
```

```
class D:public B{  
    Sub s;  
    public:  
    D()  
    //...  
};
```

问题：当派生类中含有对象成员时,其构造函数应该如何构造？

构造函数的一般形式

- 当派生类中含有对象成员时，其构造函数的形式为：

```
派生类名(参数总表): 基类名(参数表0), 对象名  
成员1(参数表1), ..., 对象成员名n (参数表n)  
{  
    //派生类新增成员的初始化语句 ...  
}
```

构造函数的执行顺序

定义派生类对象时,构造函数的执行顺序如下:

- ① 调用**基类**的构造函数,对基类数据成员初始化;
- ② 调用**对象成员**的构造函数,对对象成员的数据成员初始化;
- ③ 执行**派生类**的构造函数体,对派生类数据成员初始化。

【例】 含有对象成员的派生类

```
#include <iostream>
using namespace std;
class Base
{
protected:
    int i;
public:
    Base(int n)
    {    i = n;
        cout<<"类Base的构造函数\n";
    }
    ~Base()
    {cout<<"类Base的析构函数\n";}
};
```

基类

派生类子对象的类

```
class Sub
{
public:
    int k;
    Sub(int l)
    {
        k = l;
        cout<<"类Sub的构造函数\n";
    }
    ~Sub()
    {   cout<<"类Sub的析构函数\n";   }
};
```



```
class Derived: public Base
{
    int j;
    Sub s;
public:
    Derived(int n, int m, int l):Base(m),s(l)
    {
        j = n;
        cout<<"类Derived的构造函数\n";
    }
    void show()
    {
        cout<<"i="<<i<<" "<<"k="<<s.k<<
        " "<<"j="<<j<<endl;
    }
    ~Derived()
    {cout<<"类Derived的析构函数\n";}
};
```

派生类

对象成员

调用基类和对象成员的构造函数，完成基类数据成员和对象成员的初始化

```
int main()
{
    Derived obj(1, 2, 3);
    obj.show();
    return 0;
}
```

程序的运行结果：
类Base的构造函数
类Sub的构造函数
类Derived的构造函数
i=2 k=3 j=1
类Derived的析构函数
类Sub的析构函数
类Base的析构函数

此题未设置答案，请点击右侧设置按钮

下面关于类的派生的描述，哪一个是错误的

- ☐ A 基类也称为父类，派生类也称为子类
- ☐ B 派生方式有三种类型：public、protected、private
- ☐ C 基类中不能定义析构函数
- ☐ D 派生类中可以含有对象成员

提交

4 同名成员

- ◎ 在定义派生类的时候，允许在派生类中说明的成员与基类中的成员**名字相同**。
 - 同名成员函数
 - 同名成员变量

4.1 同名成员函数

```
class B{  
    public:  
        int f();  
};  
class D:public B{  
    public:  
        int f();  
        void g();  
};  
void D::g( ) { f(); }  
void main( )  
{  
    D obj;  
    obj.f();  
}
```

同名成员

访问哪个类中的f () ?

访问哪个类中的f () ?

C++规定：如果在派生类中定义了与基类成员同名的成员，则派生类成员**覆盖**了基类的同名成员

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

class D:public B{
    public:
        int f();
        void g();
};

void D::g( ) { f(); }

void main( )
{
    D obj;
    obj.f();
}
```

同名成员

表示访问派生类中的f(),
即被调用的函数是D::f()

被访问的函数是D::f()

```
class B{
    public:
        int f();
};
class D:public B{
    public:
        int f();
        void g();
};
void D::g( ) { f(); }
void main( )
{
    D obj;
    obj.f();
}
```

C++规定：如果在派生类中定义了与基类成员同名的成员，则派生类成员覆盖基类的同名成员

同名成员

问题:若要访问基类中的f(),怎么办?

```
class B{
    public:
        int f();
};
class D:public B{
    public:
        int f();
        void g();
};
void D::g( ) { B::f(); }
void main( )
{
    D obj;
    obj.f();
}
```

若要访问基类中的f(),
可改写成B::f()

被访问的函数仍然是D::f()


```
class B{
public:
    void f(){cout<<"class B"<<endl;}
};
class D:public B{
public:
    void f(){cout<<"class D"<<endl;}
};
void main(){
    B obj1,*p;
    D obj2;
    obj1.f();
    obj2.f();
    p=&obj2;
    p->f();
}
```

正常使用主观题需2.0以上版本雨课堂

作答

4.2 同名成员变量

```
class B{  
    public:  
        int m_i;  
};
```

```
class D:public B{  
    public:  
        int m_i;  
        void g();  
};
```

```
void D::g( ) { cout<<m_i<<endl; }
```

```
void main( )  
{  
    D obj;  
    obj.m_i;  
}
```

同名成员

访问哪个类中的m_i?

访问哪个类中的m_i?

5 访问声明*

```
class B{
    int x;
public:
    B(int x1){ x=x1;}
    void print(){ cout<<"x="<<x;}
};

class D:public B{
    int y;
public:
    D( int x1,int y1):B(x1){ y=y1;}
};

int main( )
{ D d(10,20);
  d.print();
  return 0;
}
```

公有派生

正确！

执行结果：
x=10

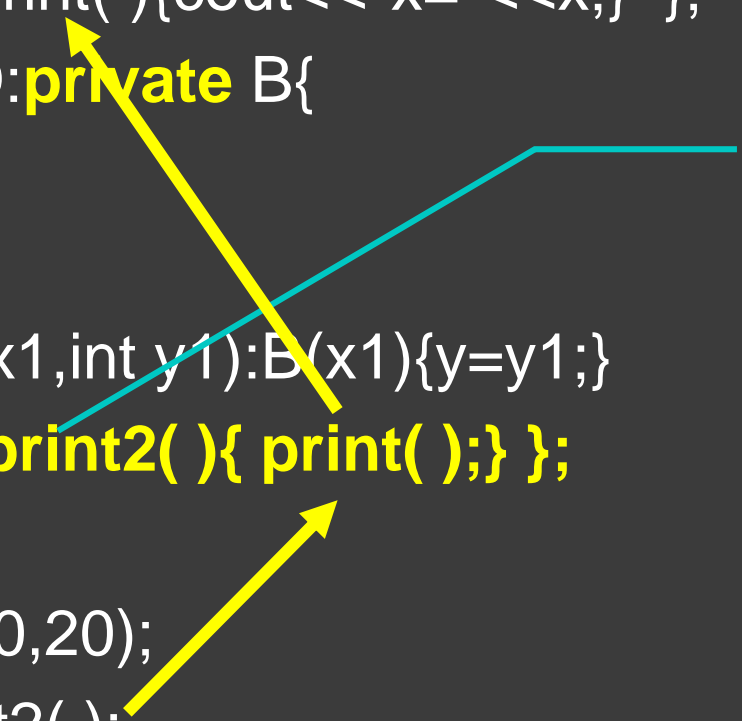
当私有派生时，能否访问基类的公有成员或保护成员？

```
class B{
    int x;
public:
    B(int x1){ x=x1;}
    void print(){ cout<<"x="<<x;}
};
class D:private B{
    int y;
public:
    D( int x1,int y1):B(x1){ y=y1;}
};
int main( )
{ D d(10,20);
  d.print();
  return 0;
}
```

私有派生

出现错误！

```
class B{
    int x;
public:
    B(int x1){x=x1;}
    void print( ){cout<<"x="<<x;} };
class D:private B{
    int y;
public:
    D(int x1,int y1):B(x1){y=y1;}
    void print2( ){ print( );} };
main( )
{ D d(10,20);
  d.print2( );
  return 0; }
```



解决方法1 :通过公有成员函数
间接访问

派生类的对象通过派生类的公有成员函数间接访问私有继承的基类的公有成员或保护成员。

执行结果:
x=10

解决方法2:使用访问声明

```
class B{
    int x;
public:
    B(int x1){x=x1;}
    void print( ){cout<<"x="<<x;} ;
class D: private B{
    int y;
public:
    D(int x1,int y1):B(x1){y=y1;}
    B::print;
};
int main( )
{ D d(10,20);
  d.print( );
  return 0; }
```

访问声明:把基类中的函数print()调整为派生类的公有成员

执行结果:
X=10

解决方法2:使用访问声明

```
class B{
    int x;
public:
    B(int x1){x=x1;}
    void print( ){cout<<"x="<<x;} ;
}

class D: private B{
    int y;
public:
    D(int x1,int y1):B(x1){y=y1;}
    B::print;
};

int main( )
{ D d(10,20);
  d.print( );
  return 0; }
```

访问声明：

把基类的保护成员名或公有成员名直接写到私有派生类定义式中的**同名段**中，

同时在成员名前冠以“**基类名::**”。

利用这种方法,该成员就成为派生类的保护成员或公有成员了。

说明

- ① 访问声明中只能是不带类型和参数的函数名或参数变量名。

例如： `B::print` ;

`void B::print;`
`B::print();`
`void B::print();` } 都是错误的

说明

② 数据成员也可以使用访问声明，例如：

```
class B {
```

```
.....
```

```
public:
```

```
    int x2;
```

```
};
```

```
class D : private B {
```

```
.....
```

```
public:
```

```
    B::x2;
```

```
};
```

把基类中的数据成员x2调整为派生类的公有成员

说明

③ 访问声明不能改变类成员在基类中原来的性质。

```
class B{  
    private:  
        int x1;  
    protected:  
        int x2;  
    public:  
        int x3;  
};
```

```
class D: private B{  
    private:  
        B::x1;    //错误  
    protected:  
        B::x1;    //错误  
        B::x2;    //正确  
        B::x3;    //正确, 有待确认  
    public:  
        B:: x1;    //错误  
        B:: x2;    //正确  
        B:: x3;    //正确  
};
```

④ 对于基类中的重载函数使用访问声明时要慎重。

```
class B{
    int x;
public:
    B(int x1){x=x1;}
    void print( ){cout<<"x="<<x;}
    void print(int a){cout<<"a="<<a;} };
class D:private B{
    int y;
public:
    D(int x1,int y1):B(x1){y=y1;}
    B::print; };
int main( )
{ D d(10,20);
  d.print(); d.print(1);
  return 0; }
```

关于私有继承

- ◎ 派生类仅仅能在类内部使用基类的功能，而不能将基类的功能以接口的形式开放给外部，也就是说。子类的对象无法直接使用到父类的功能接口。
- ◎ 所以私有继承的含义：派生类能够借助基类的功能来实现自己的功能，典型的has-a的关系，也就是组合。
- ◎ 但不同的是：私有继承的方式能够使用同一基类的protected成员。