



# C++

## 第十六讲 虚函数与多态性（二）

基础课教研室C++ 课程组



# 上一讲教学目标

- 掌握C++中虚函数的概念
- 理解重载、隐藏、覆盖



# 本讲教学目标

- 掌握纯虚函数与抽象类
- 理解什么是接口类
- 理解动态多态的原理与本质

1

纯虚函数与抽象类

2

接口类

3

动态多态的原理与本质

# 纯虚函数与抽象类

- ❖ 纯虚函数：纯虚函数是一种特殊的虚函数，通常只有函数的声明而没有任何定义实体。
- ❖ 格式：

```
class <类名>
{
    virtual <类型><函数名>(<参数表>) = 0;
};
```

# 纯虚函数与抽象类

```
class Shape {
public:
    virtual void Draw() = 0;
};
class Line:public Shape {
public:
    void Draw(){
        cout << "Line::Draw; }
};
class Circle:public Shape {
public:
    void Draw(){
        cout << "Circle::Draw;}
};
```

```
void DrawObject(Shape *p)
{
    p->Draw();
}

int main(void)
{
    Line LinObj;
    Circle CirObj;
    DrawObject(&LinObj);
    DrawObject(&CirObj);

    return 0;
}
```

# 纯虚函数与抽象类

## ❖ 注意：

- 通常是在一个基类中定义纯虚函数。
- 在该基类的所有派生类中都应该**覆盖(重写)**该函数。
- 纯虚函数的作用在于基类给派生类提供一个标准的。
- 函数原型，统一的接口，为实现动态多态打下基础。

# 纯虚函数与抽象类

- ❖ 抽象类：包含纯虚函数的类称为抽象类。

```
class <类名>
{
    virtual <类型><函数名>(<参数表>) = 0;
};
```

- ❖ 注意：不能实例化抽象类的对象

```
class Shape {
public:
    virtual void Draw() = 0;
};
Shape shapeObj; ❌
```

编译错误  
"shape": 不能实例化抽象类



# 纯虚函数与抽象类

问题：为什么要使用纯虚函数和创造抽象类呢？

在很多情况下，基类本身生成对象是不合情理的。



例如，动物作为一个基类可以派生出老虎、孔雀等子类，但动物本身生成对象明显不合常理。

# 纯虚函数与抽象类

## ❖ 抽象类的特点及用法：

- 抽象类指类中至少包含了一个纯虚函数。
- 类是现实生活中有相同属性和行为的事物的抽象，而抽象类是对类的抽象。
- 抽象类含有纯虚函数，抽象类不能被实例化，但可以定义抽象类的指针或者引用。

```
int main(void) {  
    Shape obj;  
}  
void DrawObject(Shape p) {  
    p->Draw();  
}
```



```
void DrawObject(Shape *p) {  
    p->Draw();  
}  
void DrawObject(Shape &p) {  
    p->Draw();  
}
```

# 纯虚函数与抽象类

- 抽象类中可以含有普通成员
- 纯虚函数和抽象类是一对息息相关的概念
- 可以定义指向抽象类的指针和引用，但抽象类不能用作参数类型、函数返回类型或显示转换的类型（指针和引用除外）。

```
class Shape
{
public:
    virtual void Draw() = 0;
    void foo(); // Right
};
```

```
Void draw(Shape *base){} // OK
Void draw(Sahpe &base){} // OK

Void draw(Sahpe s){} // ERROR
Sahpe draw(){} // ERROR
(Sahpe)derived; // ERROR
```

1

纯虚函数与抽象类

2

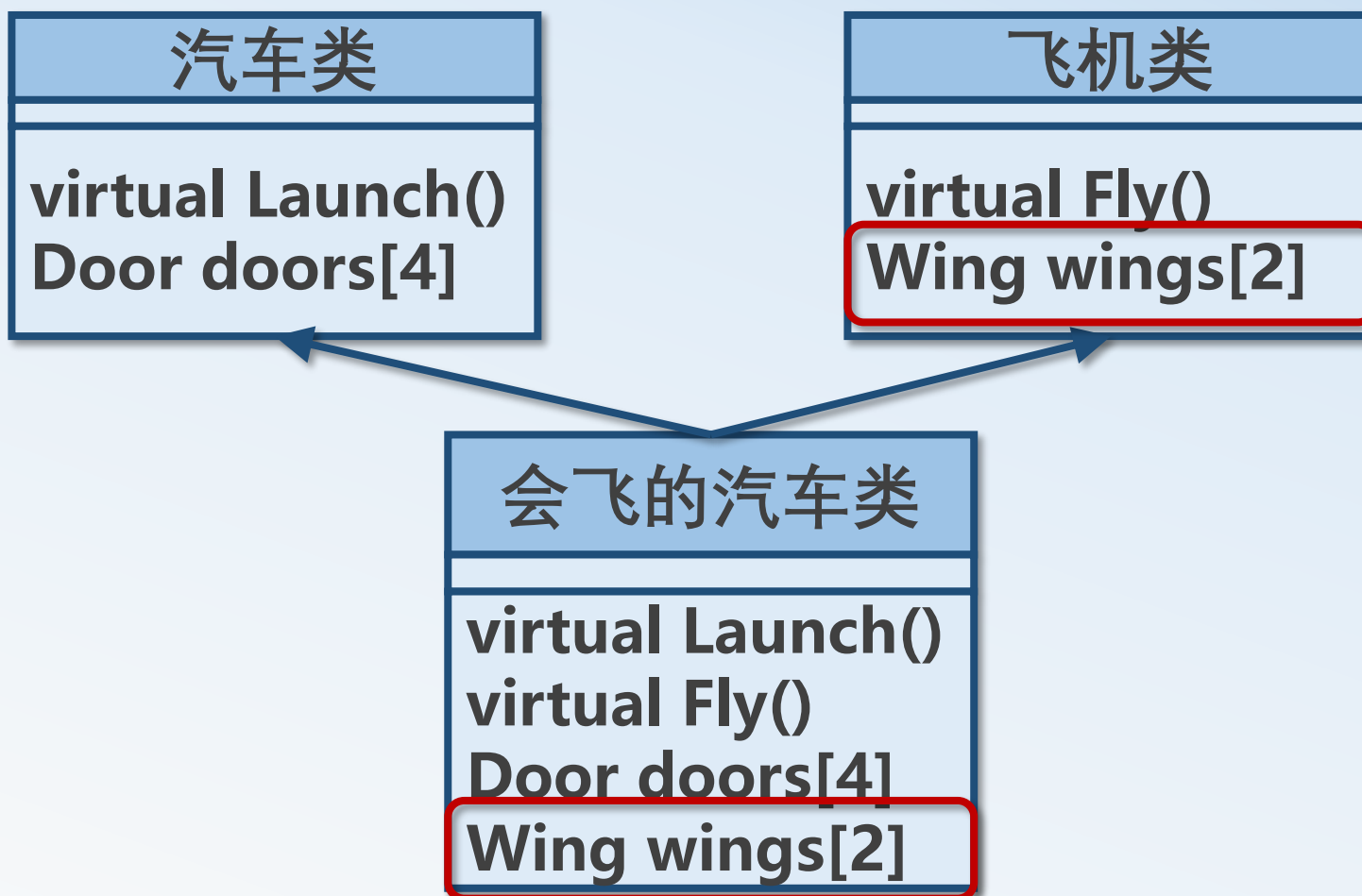
接口类

3

动态多态的原理与本质

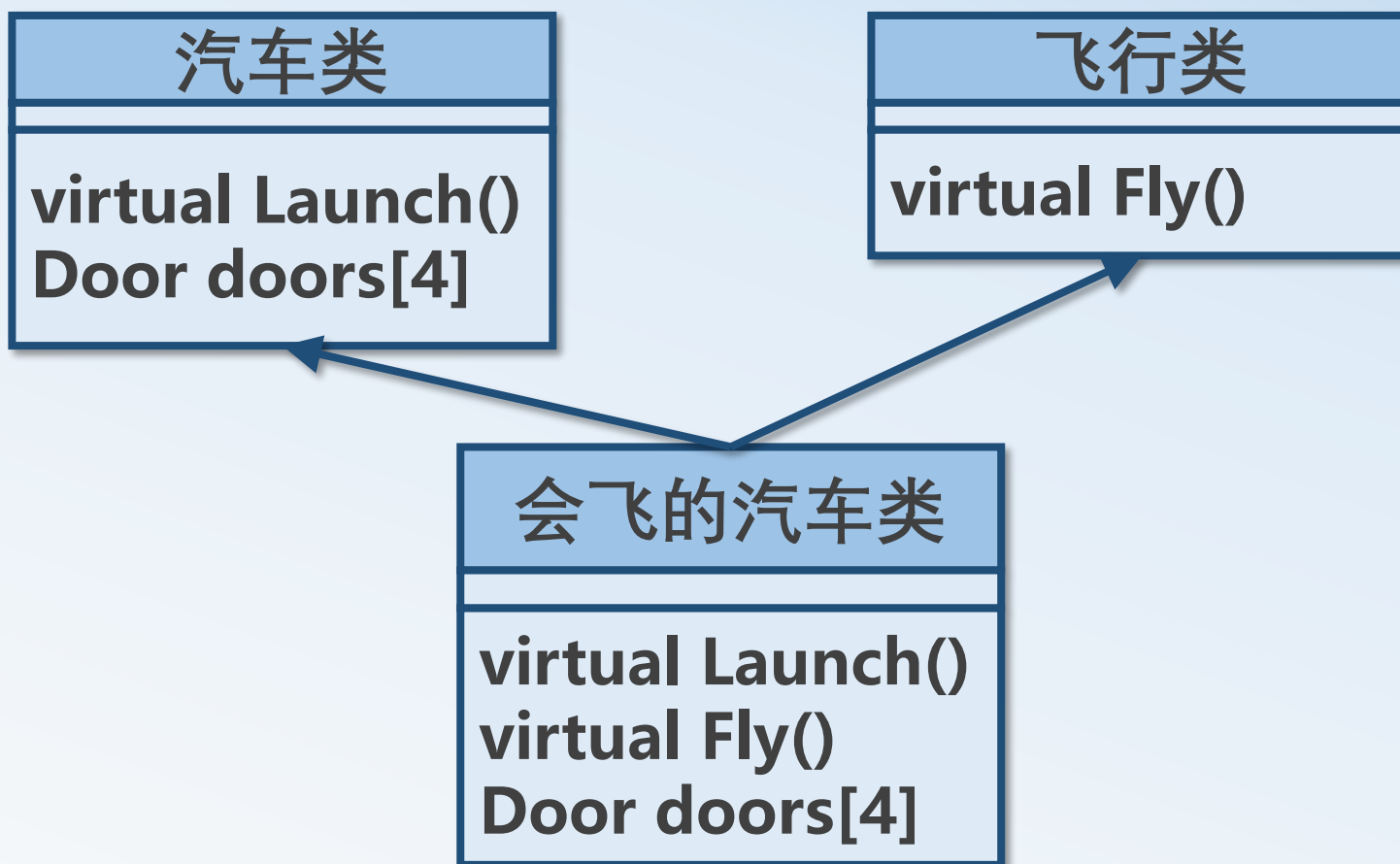
## 接口类

- ❖ 现实生活中有这样一种情况，就是一个类具有另一个类的功能，但是不具有另一个类的属性。



## 接口类

- ❖ 这样只具有父类的一些动作的父子类关系，我们可以称作父子类之间具有一种 **can do** 的关系。

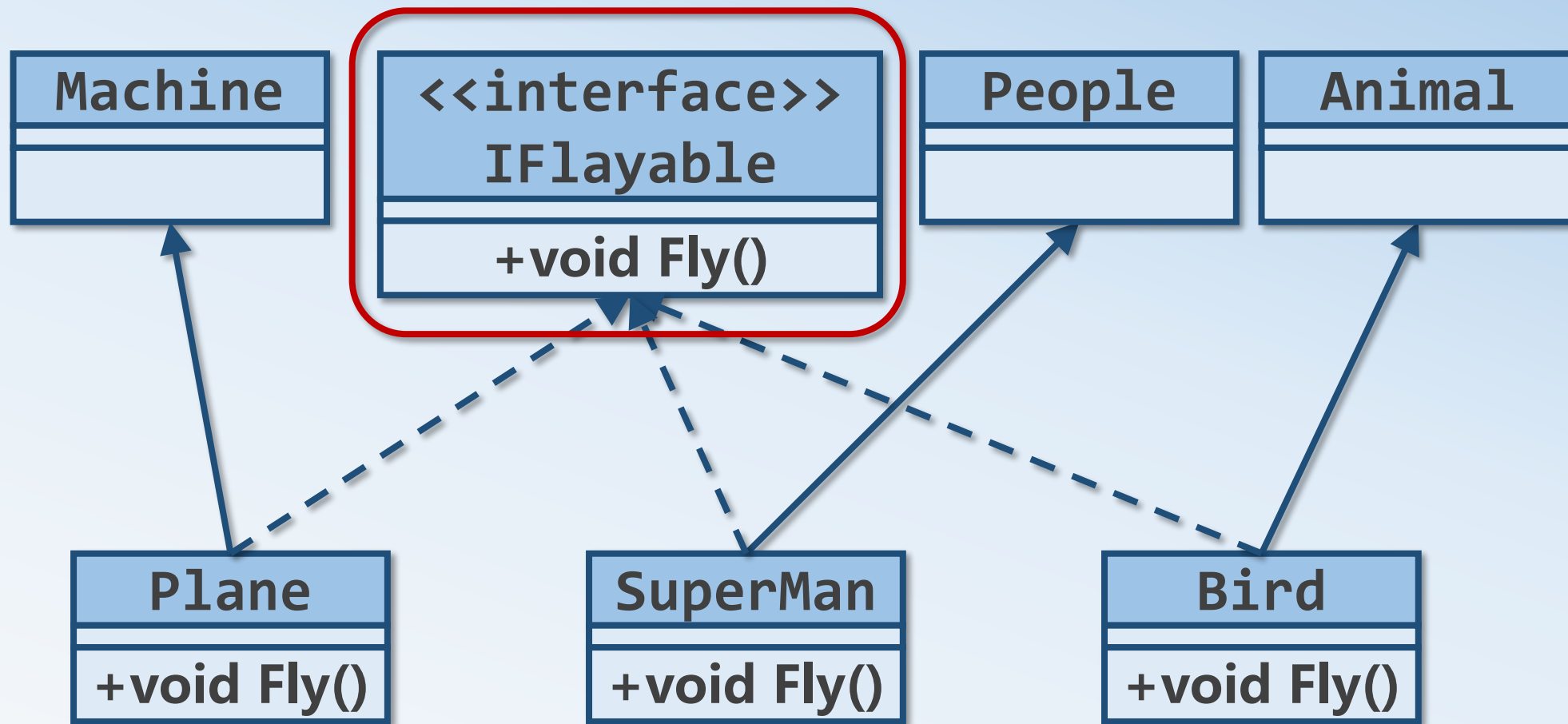


## 接口类

- ❖ 接口类：只含有纯虚函数的类称为接口类。
- ❖ 命名规则：以I开头

```
class IFlyable
{
    virtual void fly() = 0;
};
```

# 接口类





# 接口类

```
class Machine
{
};
class Animal
{
};
class People
{
};
class IFlyable
{
    virtual void fly()= 0;
};
```

```
class Bird : public Animal,
             public IFlyable
{
    virtual void fly();
};
class Plane : public Machine,
              public IFlyable
{
    virtual void fly();
};
class Superman : public People,
                 public IFlyable
{
    virtual void fly();
};
```

# 接口类

```
void makeItFly( IFlyable* aFy )
{
    aFy->fly();
}
int main(void)
{
    makeItFly(new Plane());
    makeItFly(new Bird());
    makeItFly(new Superman());

    return 0;
}
```

**Plane flying**  
**Bird flying**  
**SuperMan flying**

# 接口类

## ❖ 注意：

- 接口类是一个特殊的抽象类，很像java中的interface类。
- 接口类和子类是一种类似(CAN\_DO)的关系。

1

纯虚函数与抽象类

2

接口类

3

动态多态的原理与本质

# 动态多态的原理与本质

## ❖ 虚函数表：

- 创建对象时，C++会为每一个含有虚函数的类生成一张虚函数表，表中存放该类中所有虚函数的入口地址

```
class Base {  
public:  
    virtual void a() {  
        cout << "Base::a\n";  
    }  
    virtual void b() {  
        cout << "Base::b\n";  
    }  
    virtual void c() {  
        cout << "Base::c\n";  
    }  
};
```

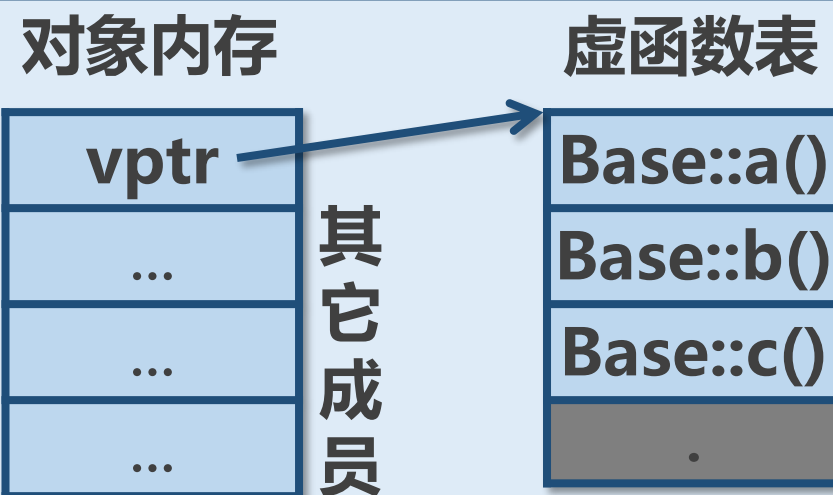
虚函数表

Base::a()	Base::b()	Base::c()	.
-----------	-----------	-----------	---

# 动态多态的原理与本质

- C++还会为每一个含有虚函数的类**自动生成一个“指针成员”**，指向该类对应的虚函数表
- **这个指针成员总是最先分配内存**

```
class Base {  
public:  
    virtual void a();  
    virtual void b();  
    virtual void c();  
    void *vptr; // 自动生成的指针成员  
};  
思考：sizeof(Base)？
```



# 动态多态的原理与本质

- 虚函数表的最后有一个结点，这是虚函数表的结束结点，就像字符串的结束符“\0”一样，其标志了虚函数表的结束。这个结束标志的值在不同的编译器下是不同的。在WinXP+VS2003下，这个值是NULL。而在Ubuntu 7.10 + Linux 2.6.22 + GCC 4.1.3下，这个值是1，表示还有下一个虚函数表，如果值是0，表示是最后一个虚函数表。

## 虚函数表

Base::a()	Base::b()	Base::c()	.
-----------	-----------	-----------	---

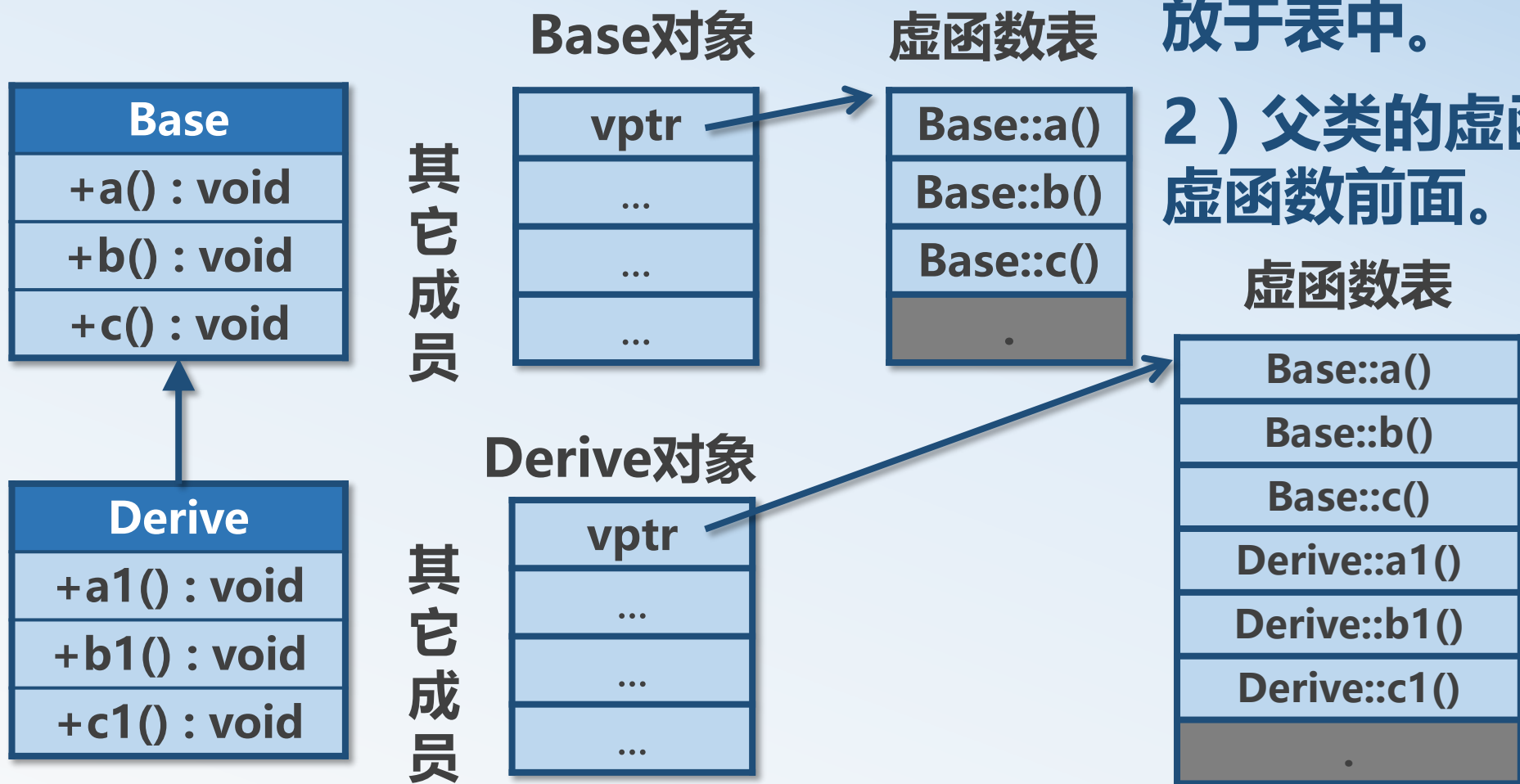
# 动态多态的原理与本质

❖ 下面，将分别说明 “无覆盖” 和 “有覆盖” 时的虚函数表的样子。



# 动态多态的原理与本质

## ❖ "无覆盖" (单重继承):

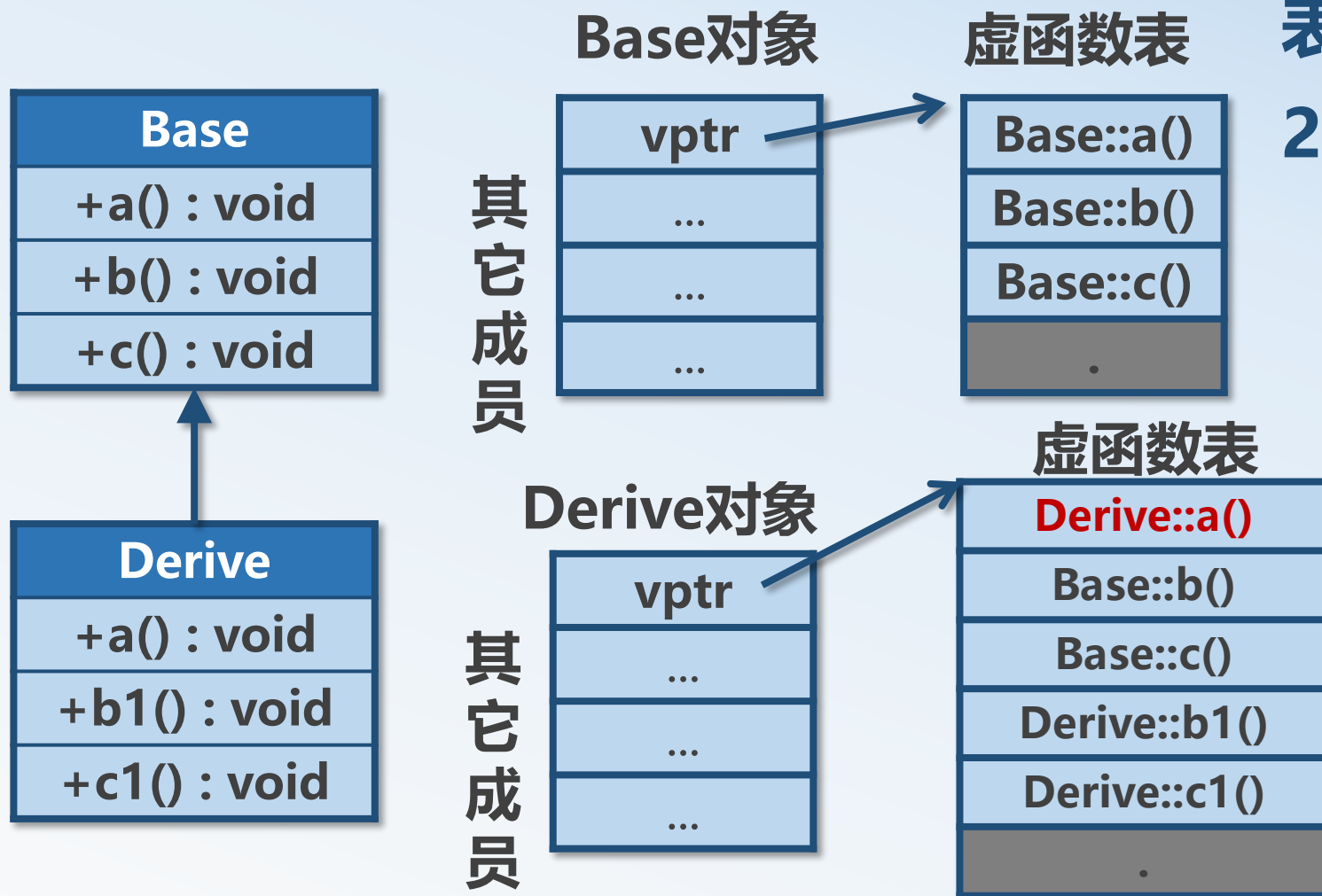


1) 虚函数按照其声明顺序放于表中。

2) 父类的虚函数在子类的虚函数前面。

# 动态多态的原理与本质

## ❖ "有覆盖" (单重继承):



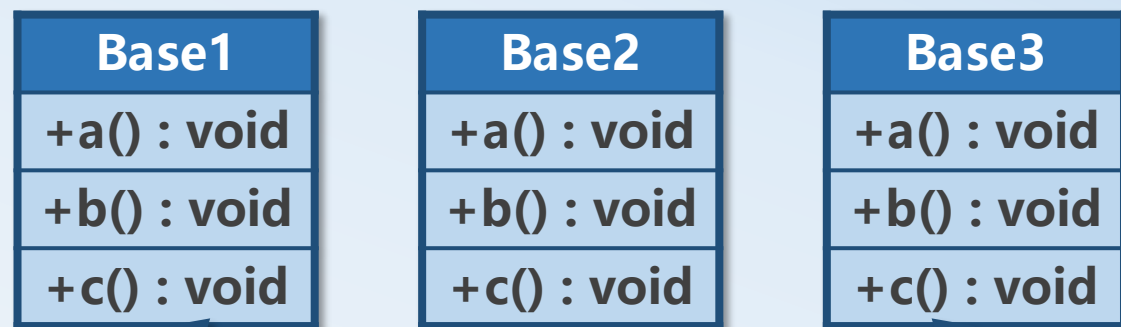
1) 覆盖的函数被放到了虚表中原来父类虚函数的位置。

2) 没有被覆盖的函数依旧。

```
Base *b = new Derive();  
b->a(); //b->Derive::a();
```

# 动态多态的原理与本质

❖ "多重继承" : ( 无虚函数覆盖 )

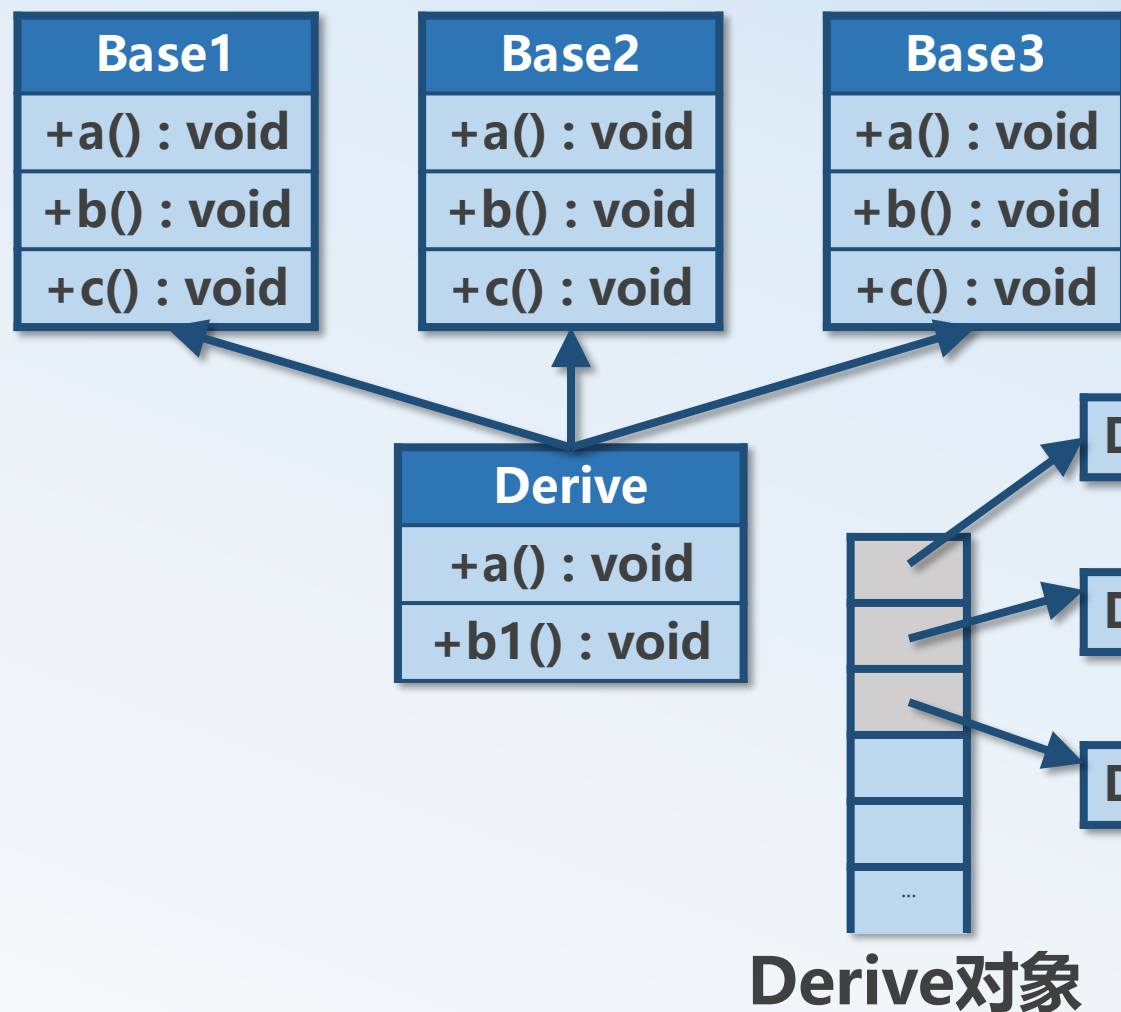


- 1) 每个父类都有自己的虚表。
- 2) 子类的成员函数被放到了第一个父类的虚函数表中

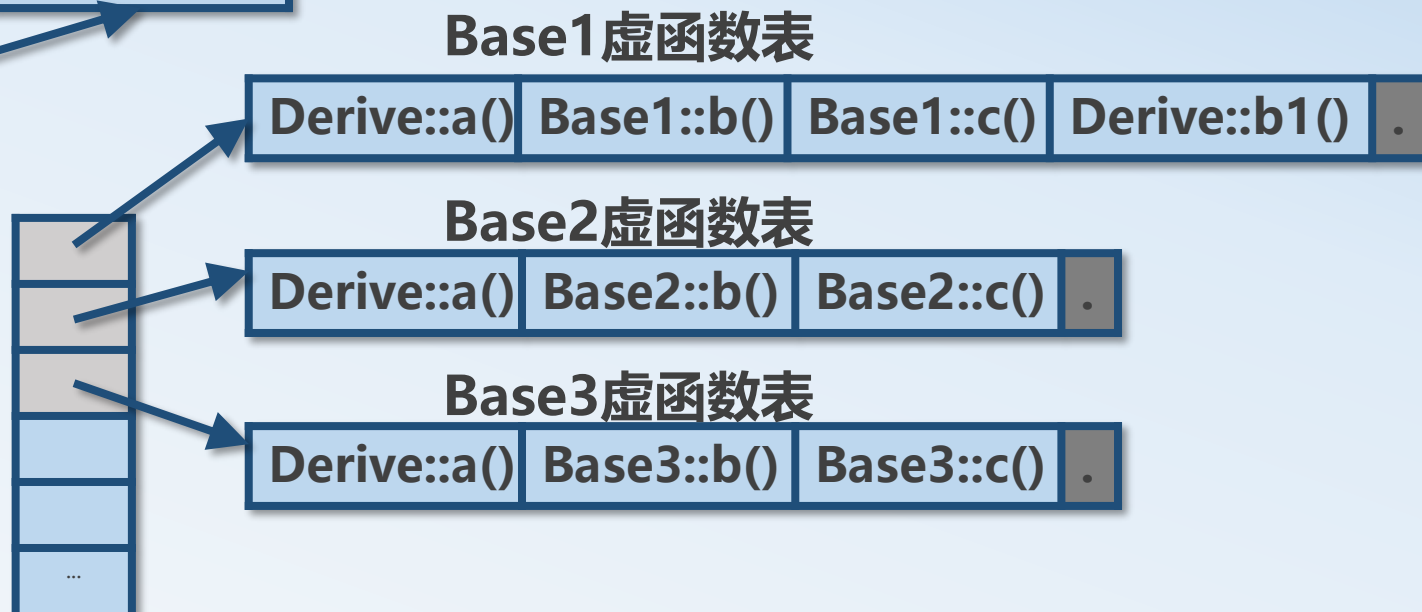


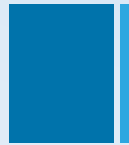
# 动态多态的原理与本质

❖ "多重继承" : ( 有虚函数覆盖 )



三个父类虚函数表中的f()的位置被替换成了子类的函数指针





# 本讲教学目标

- 掌握纯虚函数与抽象类
- 理解什么是接口类
- 理解动态多态的原理与本质



THANKS

