面向对象 OOP

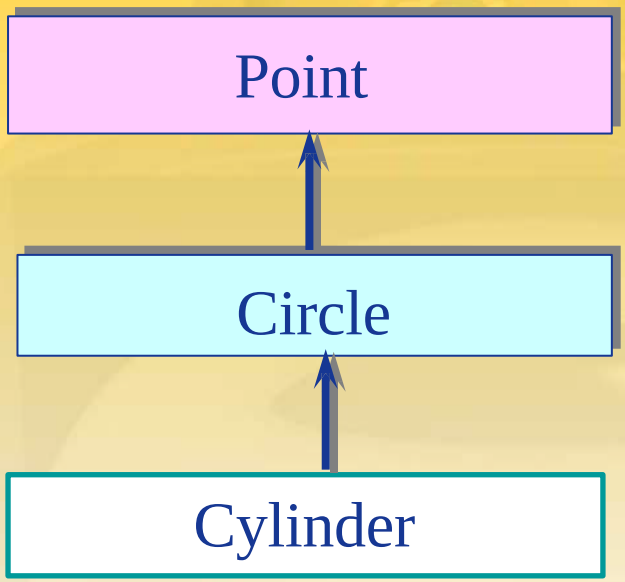
抽象(ion)

- 封装(encapsulate)
- 继承 (inheritance)
- 多态(polymorphism)

Ch5 继承

◆ 课堂练习

定义如下继承体系，能否使用一种容器，存储各种类对象？



第6章　多态

1. 理解多态
2. 多态的实现
3. 虚函数
4. 虚析构函数
5. 纯虚函数与抽象类

6.1 理解多态

6.1 理解多态

程序是客观世界的体现，在现实世界中多态现象比比皆是。

不同消息接收者产生不同行为

6.1 理解多态

➢ 多态性（Polymorphism）

所谓多态性是指同样的消息被不同类型的对象接收时产生不同行为的现象。

例如，鼠标单击，响应不同

- 向不同的对象发送同一个消息，不同的对象在接收时会产生不同的行为(方法)

同一名字，多种语义；

同个接口，多种方法；

```
Complex c1(1,2), c2(3,4), c3;
int i, j = 6;
c3 = c1 + c2 ;
i =  j + 2 ;
```

6.1 理解多态

- **多态性（Polymorphism）**

不同的类可以有同名的方法，
但其具体的实现和结果可以各不相同。

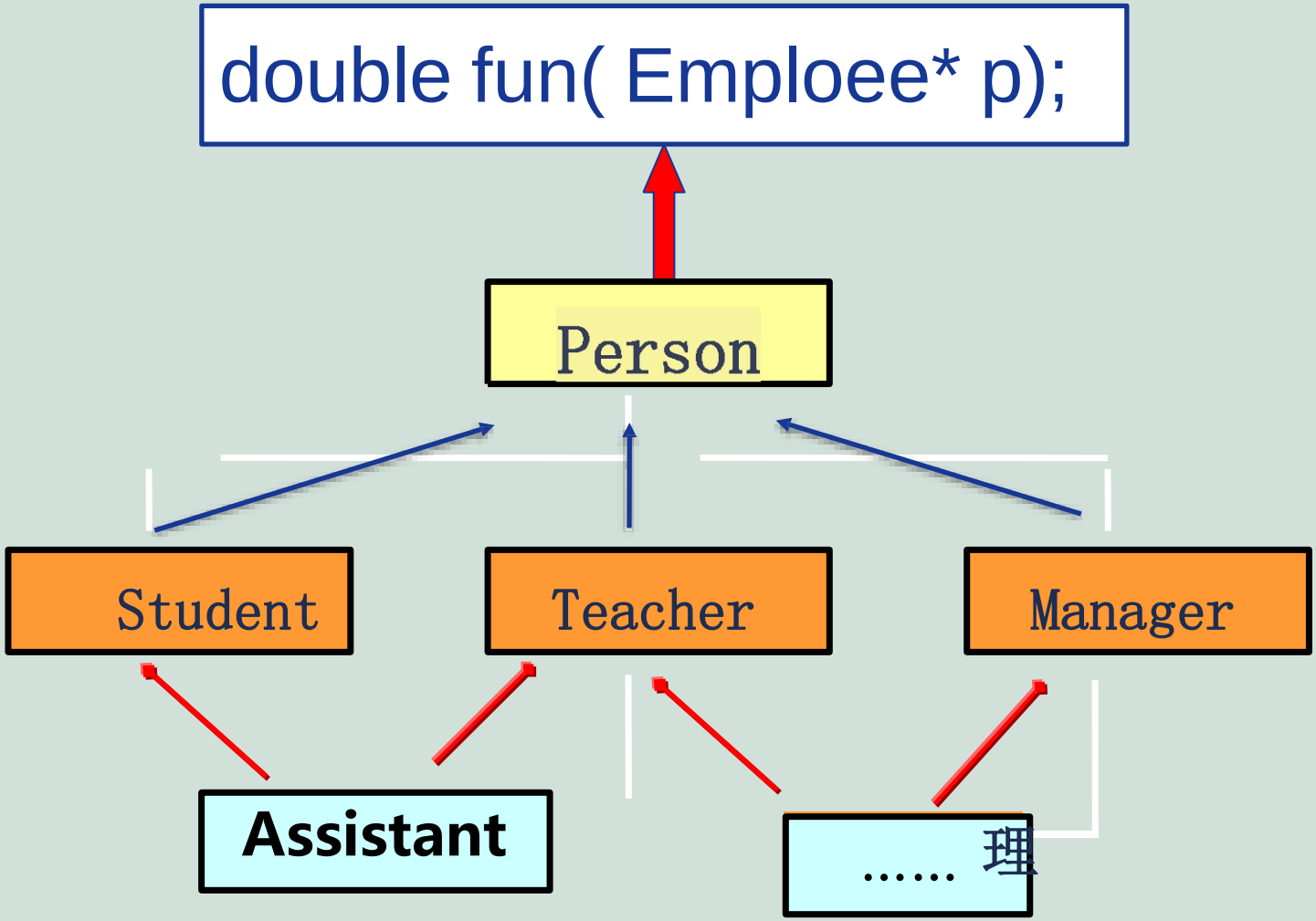
bush.roar()

orang.roar()

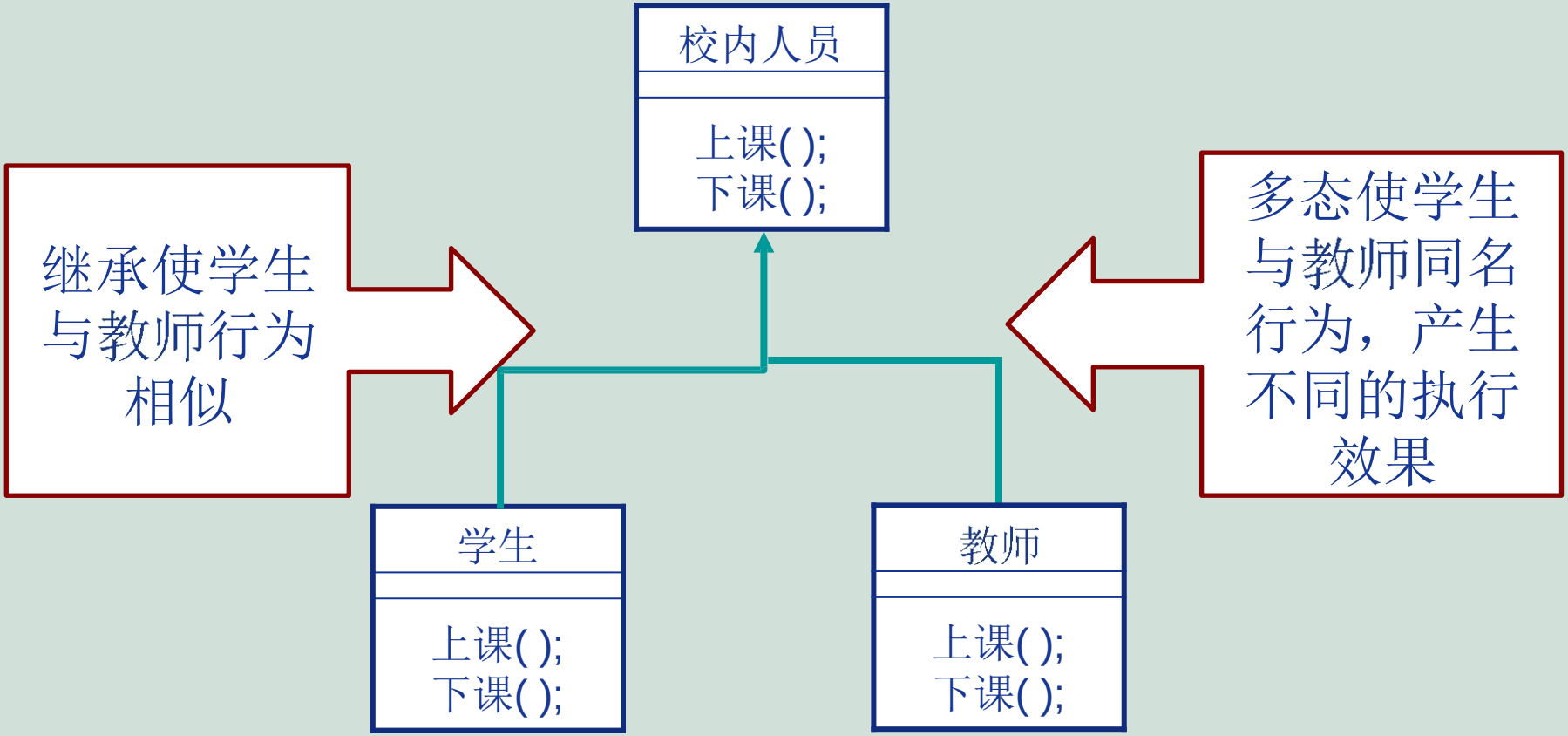
donald.roar()

客户以相同的方法请求，同一消息为不同的对象接受时可产生完全不同的结果。

- **多态性（Polymorphism）**



6.1 理解多态



本章介绍的多态性与第五章介绍的继承性相结合，
可以生成一系列虽彼此相似却又独一无二的类和对象。

6.2 多态的实现

- 多态分两类：**静态多态**与**动态多态**。
  - **静态多态**是指在程序编译时系统就能够确定要调用的是哪个函数，也被称为**编译时多态**。
  - 通过**函数的重载**来实现。

class A
int data
void set (char)
void set (int)
void show();

```
A objA;
objA.set( 'a' );
objA.set (5);
```

函数重载注意事项

- 不能仅靠函数的返回值来区别重载函数，必须从形式参数上区别开来。

- 不同参数传递方式也无法区别重载函数

```
void func(int value);
void func(int &value);
```

- 由typedef定义的类型别名并没有真正创建一个新的类型

-

```
typedef long size_t;
long function(long num);
size_t  function(size_t num);
```

静态多态

1. 类中重载成员函数

根据参数的特征加以区分

2. 派生类中的同名成员函数

- 使用“ :: ”加以区分
- 使用对象加以区分

静态关联（static binding在运行前进行的关联，
又称为**早期关联**。

```
class B :public A
char name[10];
A::Show ( )
Show ( )
```

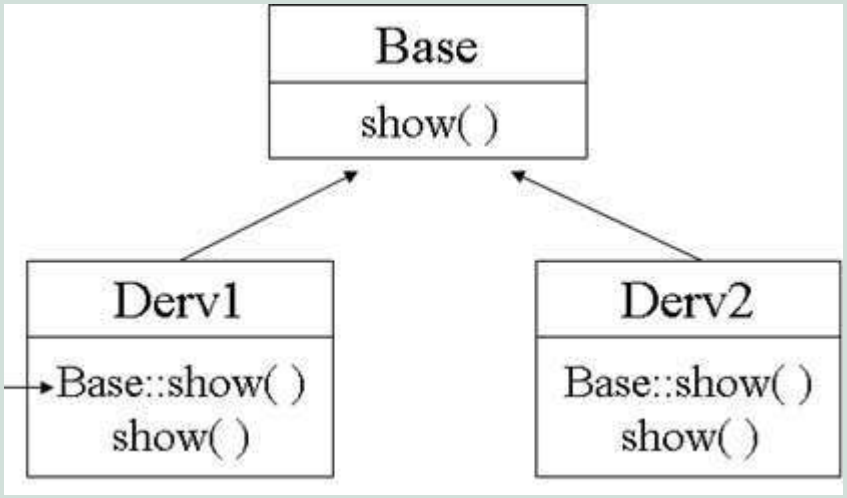
```
Aobj . Show ( );
Bobj . Show ( );
Bobj . A :: Show ( );
```

例　基类指针中调用同名成员函数

```
class Base
{   public:
      void show()
    { cout << "Base"<<endl ; }
};

class Derv1: public Base
{  public:
     void show(){ cout << "Derv1"<<endl ; }
};

class Derv2: public Base
{  public:
     void show(){ cout << "Derv2"<<endl ; }
};
```



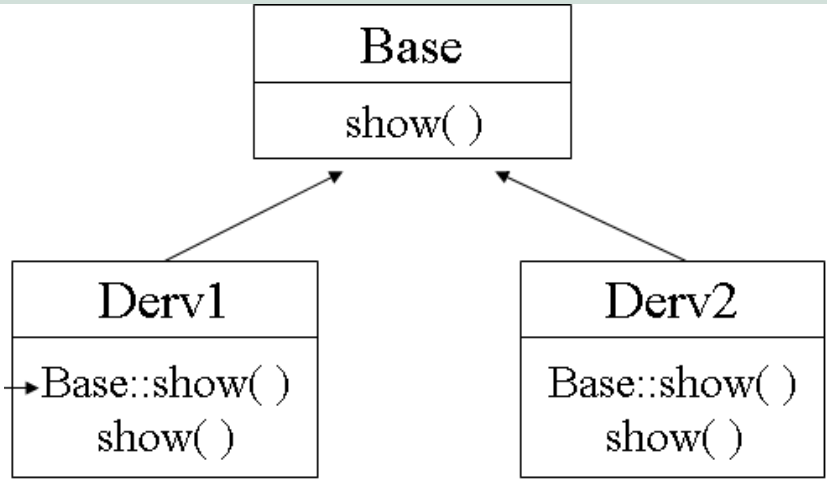
例　基类指针中调用同名成员函数

```
//eg1.cpp

Derv1   dv1;
Derv2   dv2;
dv1.show() ;
dv2.show() ;

Base* pBase;
pBase = &dv1;
pBase->show();

pBase = new Derv2();
pBase->show();
```



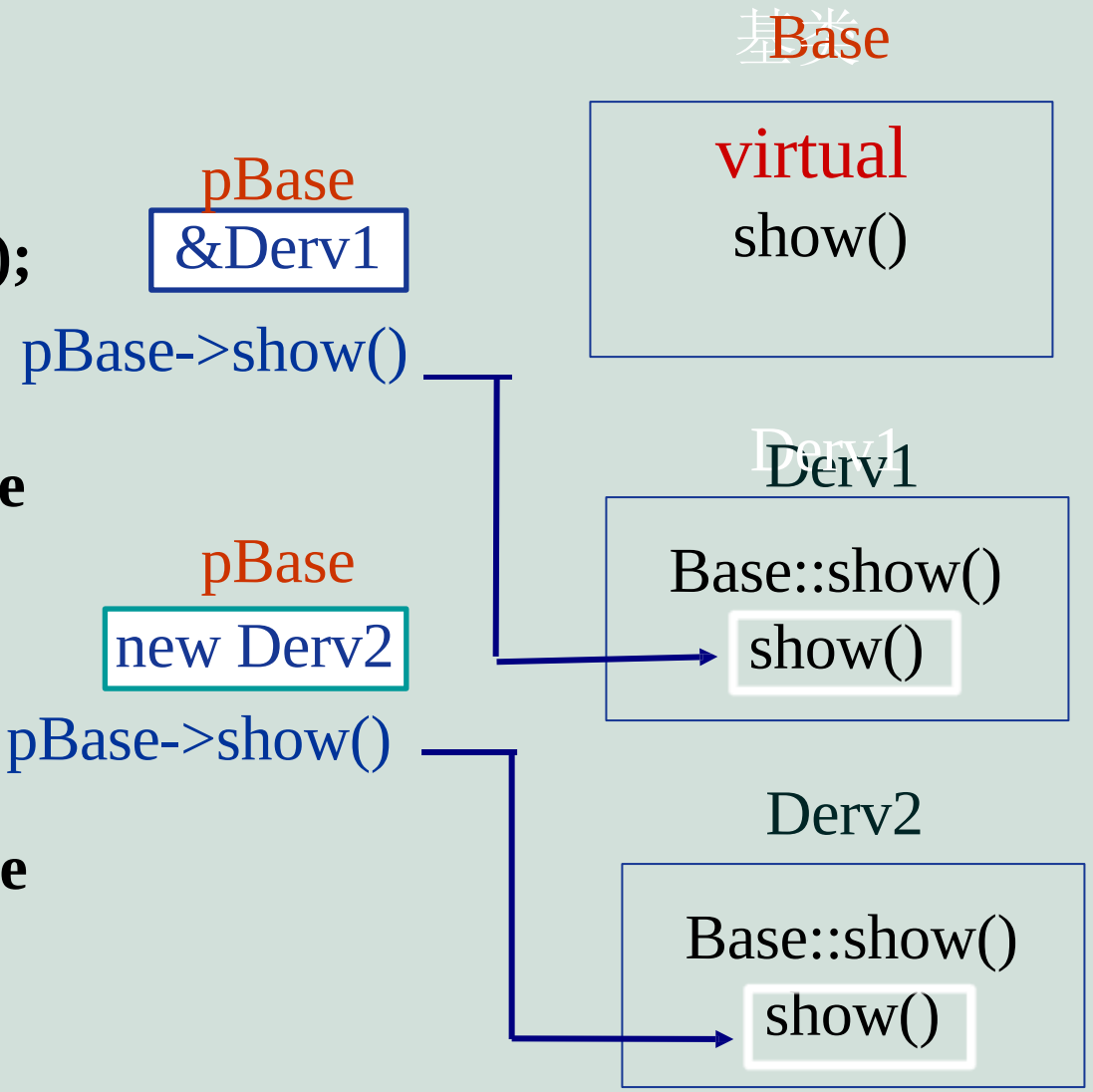
通过基类指针
只能访问从基类继承的成员

例　基类指针中调用虚函数

```
class Base
{   public:
      virtual   void show();
};

class Derv1: public Base
{  public:
      void show();
};

class Derv2: public Base
{  public:
      void show();
};
```



6.2 多态的实现

- **动态多态性**
  指程序在编译时并不能确定要调用的函数，直到运行时系统才能动态地确定操作所针对的具体对象，它又被称为**运行时多态。**
- 动态多态是通过**虚函数**（virtual function）来实现的。

6.3 虚函数

- C++中的虚函数的作用是允许在派生类中重新定义与基类同名的函数，并且可以**通过基类指针或者基类引用来访问这个同名函数**。虚函数成员声明的语法为：

```
virtual 函数类型  函数名（参数列表）;
```

注意以下两点：

1. irtual只能使用在类定义的函数**原型声明**中，不能在成员函数实现的时候使用，也不能用来限定类外的普通函数。
2. irtual具有继承性，在派生类覆盖基类虚成员函数时，既可以使用virtual，也可以不用virtual来限定，二者没有差别，默认派生类中的重写函数是具有virtual的。