

C++ Programming

Chapter 11 Inheritance and Derivation

Zheng Guibin
(郑贵滨)

目录

CONTENT

Inheritance and Derivation

继承与派生

- 继承概述
- 基类和派生类
- 派生类的构造与析构
- 多重继承

11. 继承与派生

◆ 面向对象程序设计的基本特点

- 抽象
- 封装
- 继承
- 多态

◆ 类与对象

- 类定义, 成员调用, 对象使用, 构造函数, 拷贝构造函数, 析构函数

11. 继承与派生

1 继承概述

2 基类和派生类

3 派生类的构造与析构

4 多重继承

11.1 继承概述

- ◆ 继承是不同事物之间存在的复杂关系的一种。
- ◆ 继承机制允许以原有的类为基础产生(派生)新的类。
- ◆ 优点: 增强代码的重用性和可扩充性。

通过C++语言中的继承机制, 一个新类既可以共享另一个类的操作和数据, 也可以在新类中定义已有类中没有的成员, 这样就能大大的节省程序开发的时间和资源。

11.1 继承概述

◆ 类之间的关系: *has-A* & *is-A*

- *has-A*

- 包含关系，用以描述一个类由多个“部件类”构成。
- 实现has-A关系用类成员表示，即一个类中的数据成员是另一种已经定义的类。

- *uses-A*

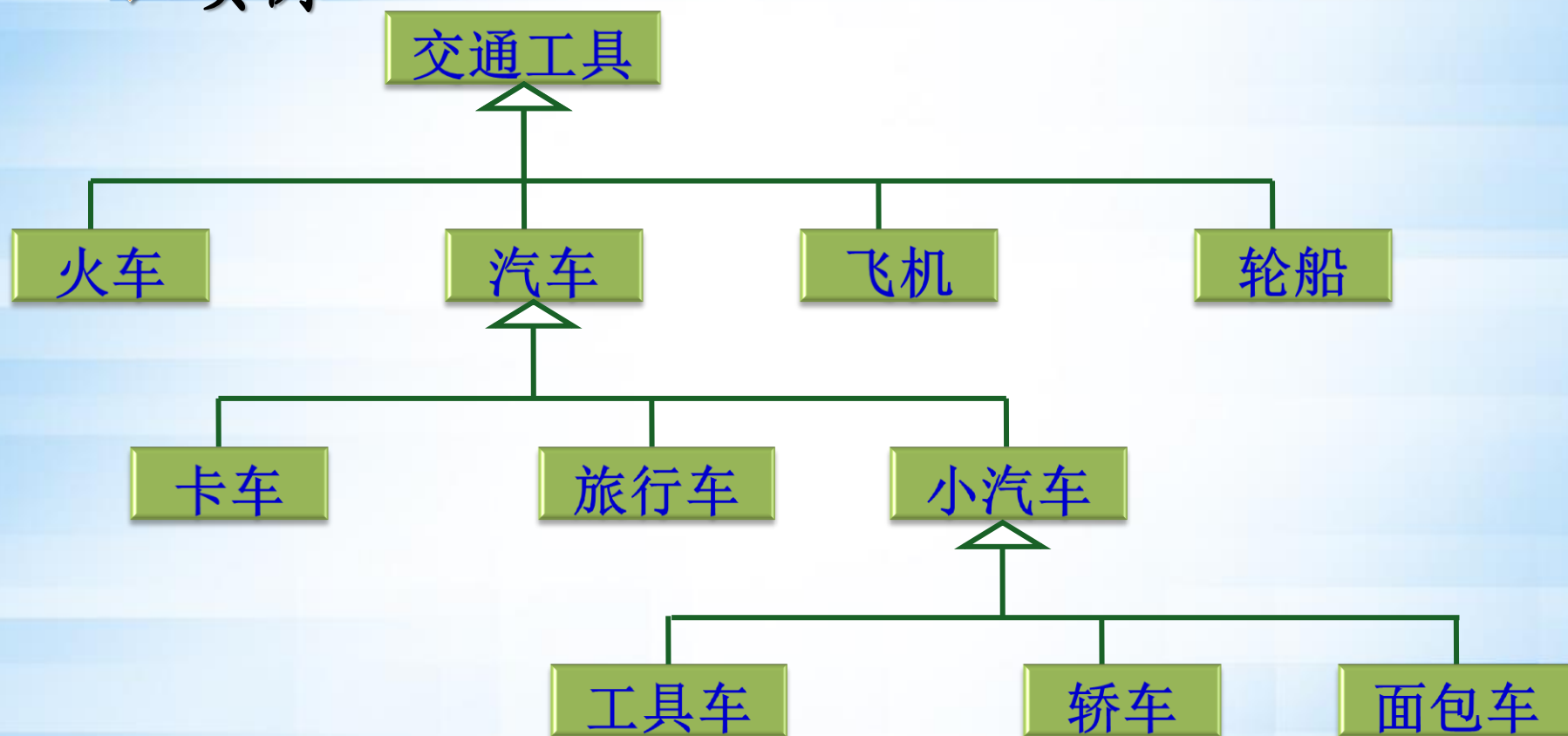
- 一个类部分地使用另一个类。通过类之间成员函数的相互联系，定义友员或对象参数传递实现。

- *is-A*

- 机制称为“继承”。
- 关系具有传递性,不具有对称性。

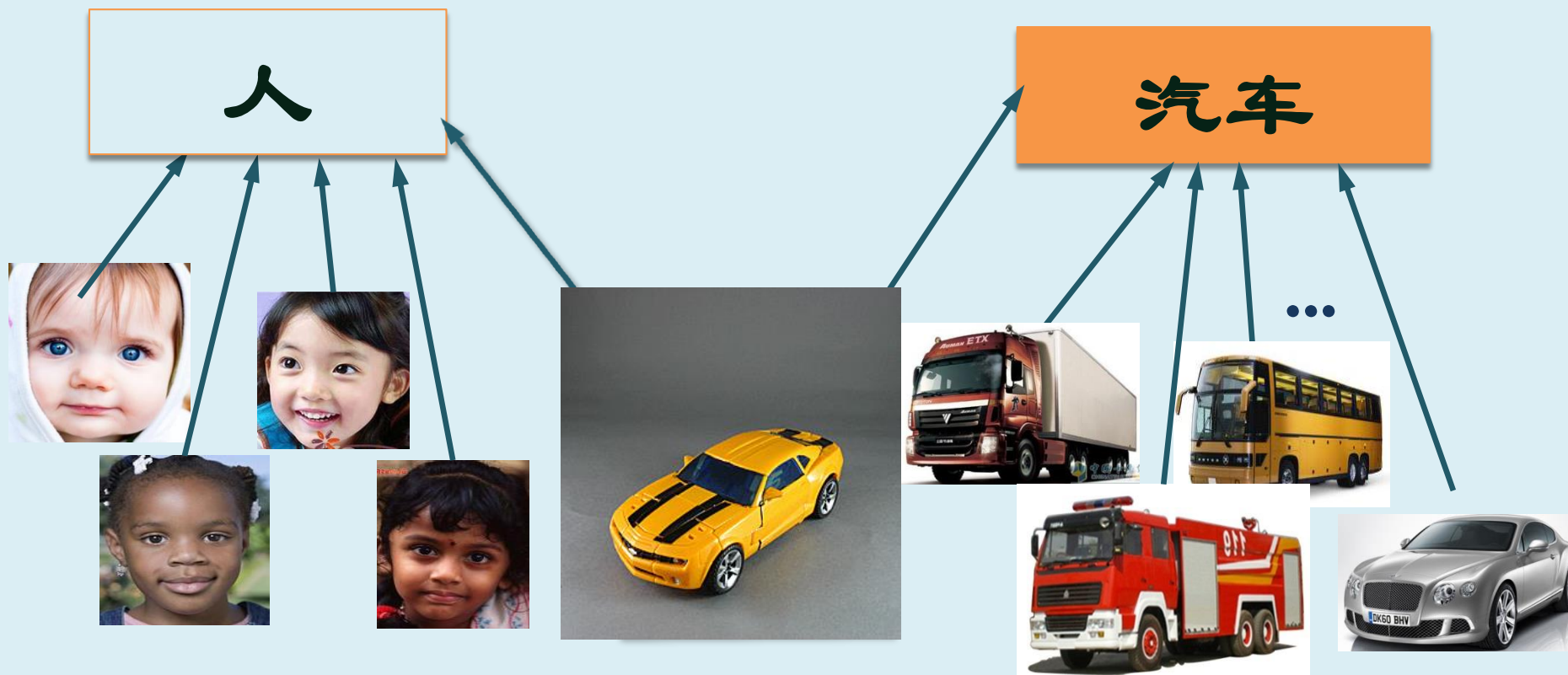
11.1 继承概述

实例



11.1 继承概述

实例



11.1 继承概述

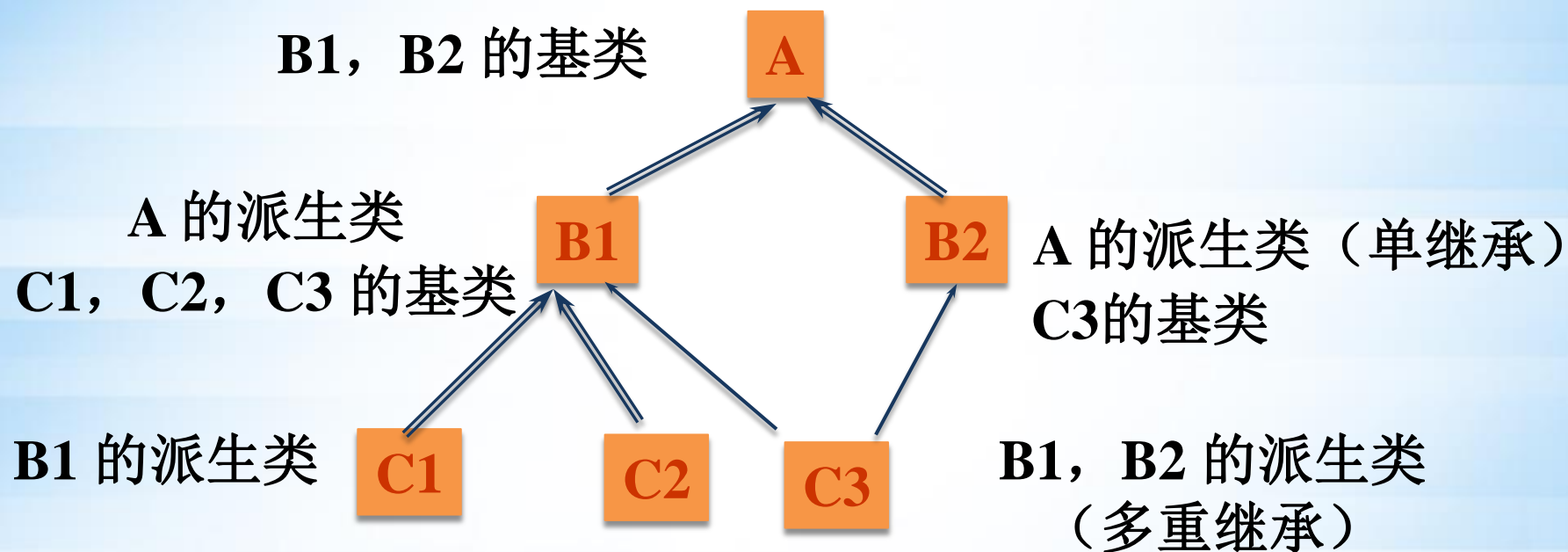
- ◆ **继承与派生是同一过程从不同的角度来看**
 - 保持已有类的特性而构造新类的过程称为继承。
 - 在已有类的基础上新增自己的特性而产生新类的过程称为派生。
- ◆ **继承允许以原有的类为基础产生(派生)新的类。**
 - 称已存在的用来派生新类的类为**基类**，又称为**父类**
 - 派生出的新类称为**派生类**，又称为**子类**。
 - 派生类包含基类的特征,共享基类的成员函数，使用基类的数据成员。
 - 还可以定义自己的新特性(数据成员和成员函数)。
 - 直接参与派生出某类的基类称为**直接基类**，基类的基类甚至更高层的基类称为**间接基类**。
- ◆ **继承的目的：实现代码重用。**

11.1 继承概述

◆ 继承关系的特点

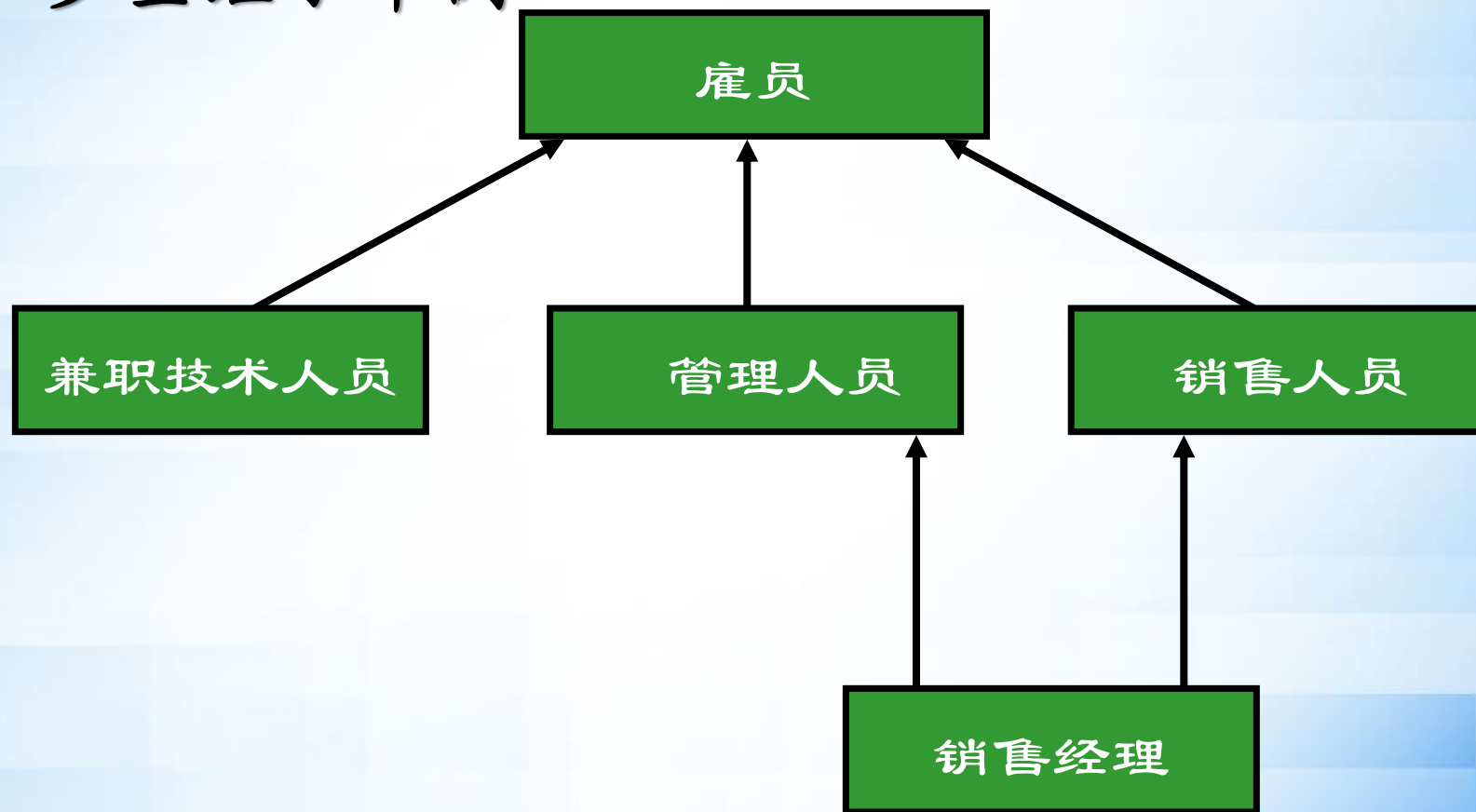
- 单继承: 一个派生类只有一个基类;
- 多重继承: 一个派生类有多个基类;
- 继承关系可以是多级: 即可以有类Y继承类X和类Z继承类Y同时存在。
- 不允许继承循环: 不能有类Y继承类X、类Z继承类Y和类X继承类Z同时存在。

11.1 继承概述



11.1 继承概述

◆ 多重继承举例



11.1 继承概述

◆ 派生类声明语法

单继承:

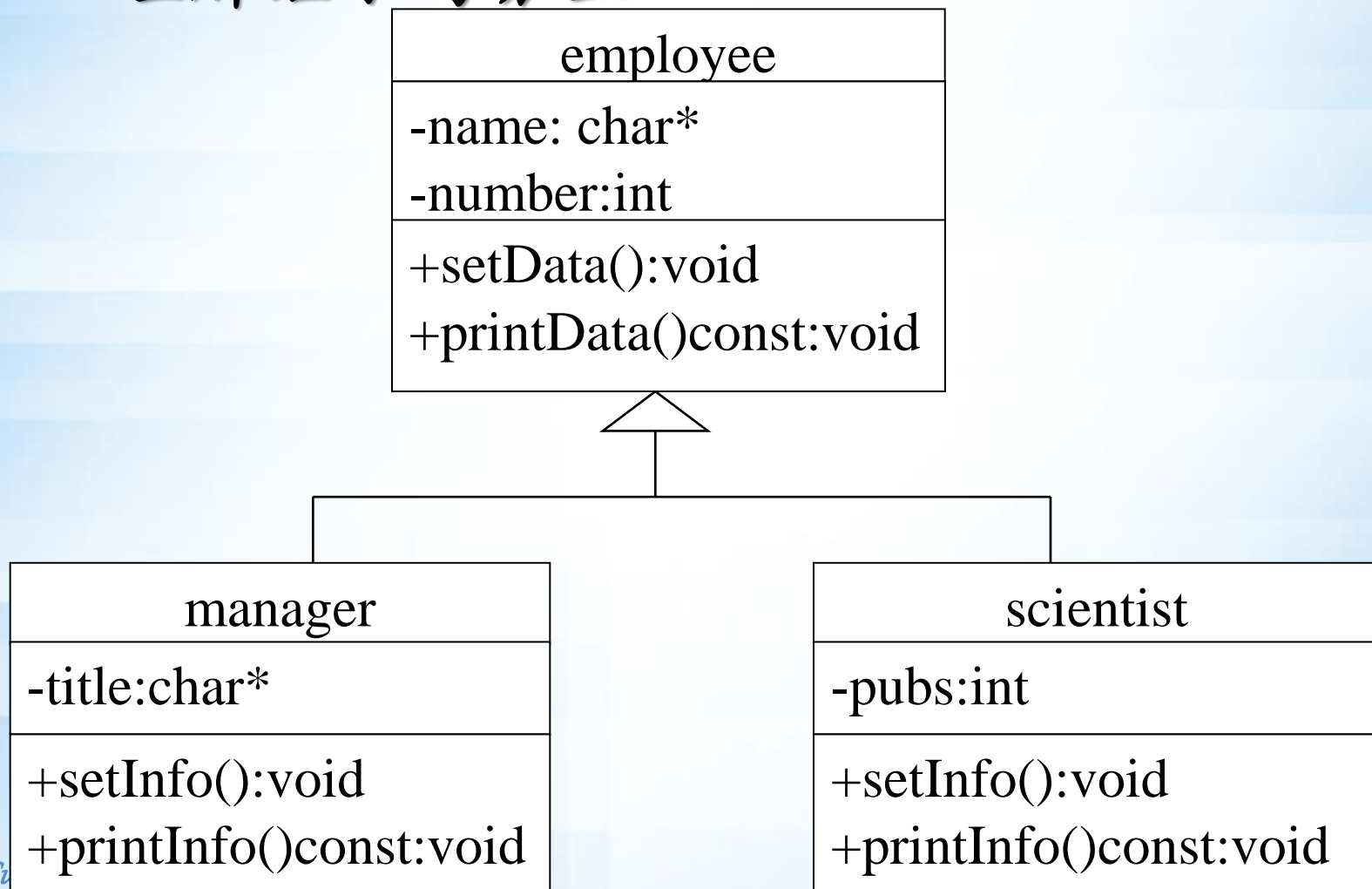
```
class 派生类名 : 访问控制/继承方式 基类名
{
    新增数据成员和成员函数声明;
};
```

多重继承:

```
class 派生类名 : 访问控制1/继承方式1 基类名1,
                ... , 访问控制n/继承方式n 类名n
{
    新增数据成员和成员函数声明;
};
```


11.1 继承概述

◆ 理解继承与派生



11.1 继承概述

```
class employee{    //..... };  
class manager : public employee{    //..... };
```

- 派生出的manager类具有以下成员：

manager
name: char* number:int -title:char*
+setData():int +printData()const:int +setInfo():void +printInfo()const:void

...练习

- ◆ 定义一个基类shape，在此基础上派生出Rectangle和Circle，二者都有GetArea()函数计算对象的面积。再使用Rectangle类创建一个派生类Square。

11. 继承与派生

1 继承概述

2 基类和派生类

3 派生类的构造与析构

4 多重继承

11.2 基类和派生类

- ◆ 继承方式与访问控制
- ◆ 重名成员
- ◆ 继承中的静态成员调用问题
- ◆ 类型转换规则

11.2 基类和派生类

- ◆ 一个基类A派生出类B，类B派生出类C，则类A为派生类C的间接基类。
 - 派生类是基类的具体化。
 - 派生类是基类定义的延续。
 - 派生类是基类的组合（多继承）。

- ◆ 派生类将其本身与基类区别开来的方法是添加数据成员和成员函数。

11.2 基类和派生类

◆ 类继承关系的语法形式

```
class 派生类名: 基类名表 {  
    数据成员和成员函数声明  
};
```

基类名表 构成

访问控制 基类名₁... , 访问控制 基类名_n

```
class employee  
{    //..... };  
class manager : public employee  
{    //..... };
```

访问控制

表示派生类对基类的继承方式

11.2 基类和派生类



(1) 继承方式

- 三种继承方式
 - 公有继承 public
 - 私有继承 private
 - 保护继承 protected
- 不同继承方式**不影响**
 - 派生类**成员**对**父类成员**的访问权限(子类**内**对父类非私有成员的访问)
- 不同继承方式**影响**
 - 通过派生类**对象**对**父类成员**的访问权限(子类**外**对父类非私有成员的访问)

11.2 基类和派生类

◆ (2) 访问控制——公有继承

基类	派生类
private 成员	隐藏
protected 成员	protected 成员
public 成员	public 成员
	private 成员
	protected 成员
	public 成员

11.2 基类和派生类

◆ (2) 访问控制——私有继承

基类	派生类
private 成员	隐藏
protected 成员	private 成员
public 成员	private 成员
	private 成员
	protected 成员
	public 成员

11.2 基类和派生类

◆ (2) 访问控制——保护继承

基类	派生类
private 成员	隐藏
protected 成员	protected 成员
public 成员	protected 成员
	private 成员
	protected 成员
	public 成员

11.2 基类和派生类

不论哪种方式继承基类，
派生类都不能直接使用
基类的私有成员

◆ (2) 访问控制

派生类对基类成员的使用，与继承访问控制和基类中成员性质有关

➤ 公有继承

基类的公有成员 → 派生类的公有成员

基类的保护成员 → 派生类的保护成员

➤ 私有继承

默认继承方式

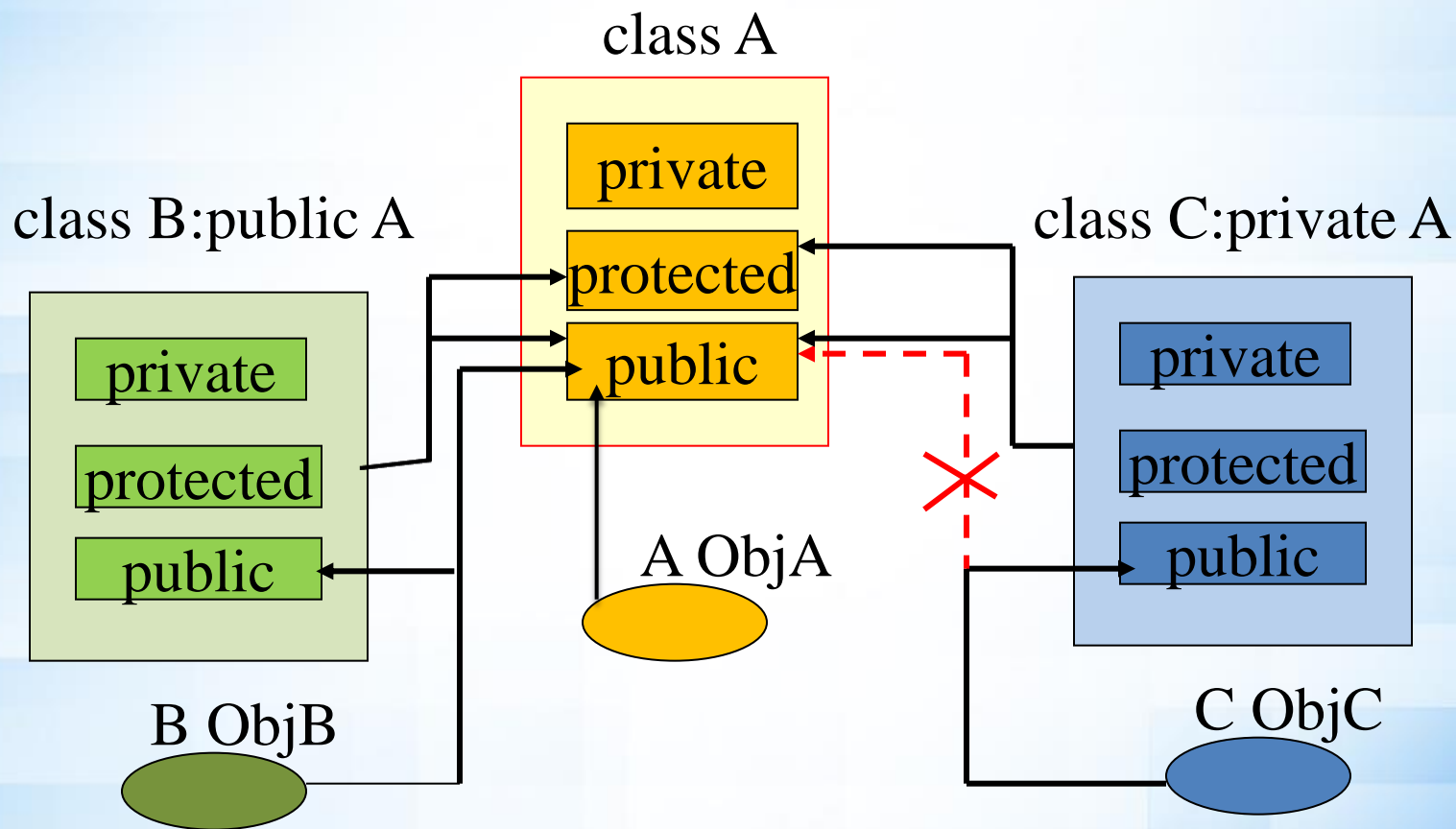
基类的公有成员和保护成员 → 派生类的私有成员

➤ 保护继承

基类的公有成员和保护成员 → 派生类的保护成员

11.2 基类和派生类

◆ (2) 访问控制



公有和私有的派生

11.2 基类和派生类

◆ (3) 重名成员

- 派生类定义了与基类同名的成员；
- 在派生类中访问同名成员时屏蔽了基类的同名成员；
- 在派生类中使用基类的同名成员，显式使用类名限定符：

类名 :: 成员

11.2 基类和派生类

◆ (3) 重名成员——成员函数的使用与覆盖

基本原则：派生类成员支配基类同名成员

1. 访问对象成员时，先访问本派生类对象中同名成员。
2. 如果派生类对象中没有该同名成员，则进而访问该对象的直接基类的同名成员。
3. 如果该对象的直接基类中仍然没有该同名成员，则不断上溯至该对象的间接基类中寻找，直至找到。

11.2 基类和派生类

◆ 理解重名数据成员使用

base a b

derived a b **b** **c**

derived obj



11.2 基类和派生类

//重名数据成员

例:

```
class base
{ public: int a, b; };
class derived: public base
{ public: int b, c; };
void f()
{ derived obj;
  obj.a = 1; //使用base 类的数据成员a
  obj.base::b = 2;
  obj.b = 3;
  obj.c = 4;
};
```

base a b

derived a b **b** **c**

derived obj

1



11.2 基类和派生类

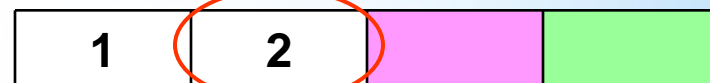
// 重名数据成员

```
class base
{ public: int a, b; };
class derived: public base
{ public: int b, c; };
void f()
{ derived o;
  o.a = 1;
  o.base::b = 2;
  o.b = 3;
  o.c = 4;
};
```

base	a	b
------	---	---

derived	a	b	b	c
---------	---	---	----------	----------

derived o



使用从base
类继承的 b



11.2 基类和派生类

// 重名数据成员

```
class base
{ public: int a, b; };
class derived: public base
{ public: int b, c; };
void f()
{ derived o;
  o.a = 1;
  o.base::b = 2;
  o.b = 3;
  o.c = 4;
};
```

base	a	b
------	---	---

derived	a	b	b	c
---------	---	---	----------	----------

derived o

1	2	3	
---	---	----------	--

使用derived 类
的数据成员b



11.2 基类和派生类

// 重名数据成员

```
class base
{ public: int a, b; };
class derived: public base
{ public: int b, c; };
void f()
{ derived o;
  o.a = 1;
  o.base::b = 2;
  o.b = 3;
  o.c = 4;
};
```

base	a	b
------	---	---

derived	a	b	b	c
---------	---	---	----------	----------

derived o

1	2	3	4
---	---	---	---

使用derived 类
的数据成员c



11.2 基类和派生类

//重名数据成员

```
class base
{ public :
    int a , b ;
};
class derived : public base
{ public :
    int b , c ;
};
void f ()
{ derived o ;
  o . a = 1 ;
  o . base :: b = 2 ;
  o . b = 3 ;
  o . c = 4 ;
};
```

base	a	b
------	---	---

derived	a	b	b	c
---------	---	---	----------	----------

derived o

1	2	3	4
---	---	---	---

- 基类成员的作用域延伸到所有派生类
- 派生类的重名成员屏蔽基类的同名成员



```
#include<iostream>
class A
{ public:
```

通过继承，类B具有两个同名成员函数

```
void A::print(); // void print( A * this );
```

```
void B::print(); // void print( B * this );
```

➤ 派生类中定义与基类同名的成员函数，称为重载成员函数

```
int a1, a2 ;
```

```
A( int i1=0, int i2=0 ) { a1 = i1; a2 = i2; }
```

```
void print()
```

```
{ cout << "a1=" << a1 << '\t' << "a2=" << a2 << endl ; }
```

```
};
```

```
class B : public A
```

```
{ public:
```

```
int b1, b2 ;
```

```
B( int j1=1, int j2=1 ) { b1 = j1; b2 = j2; }
```

```
void print() //定义同名函数
```

```
{ cout << "b1=" << b1 << '\t' << "b2=" << b2 << endl ; }
```

```
void printAB()
```

```
{ A::print() ; //派生类对象调用基类版本同名成员函数
```

```
print() ; //派生类对象调用自身的成员函数
```

```
}
```

```
};
```

```
void main()
```

```
{ B b ; b.A::print(); b.printAB(); }
```

(3) 继承中的静态成员调用

- ◆ 派生类不能访问基类的私有成员（隐藏）
- ◆ 基类定义的非私有静态成员，不受继承方式的影响，在整个类层次体系中都可以访问
- ◆ 派生类中访问基类静态成员，必须用以下形式显式说明：

类名::成员

- ◆ 定义与基类同名的成员

如果派生类定义了与基类同名的成员，称派生类的成员覆盖了基类的同名成员，若要在派生类中使用基类同名成员，可以显式地使用类名限定符：

类名::成员

例: (3) 继承中的静态成员调用

```
class A{
public:  static void f () {}
        void g () {}      };
```

改为private会怎样？

```
class B : private A { };
```

```
class C : public B { void h () ; };
```

```
void C :: h ()
```

```
{ A :: f (); // OK 不可见，但可以通过类名直接调用
```

f (); // Error! 不可见，若B继承A改为public方式就正确

B::g (); //Error, 不可见，若B继承A改为public方式就正确

g (); //Error, 不可见，若B继承A改为public方式就正确

```
} ;
```

(4) 类型转换规则

- ◆ 一个公有派生类的对象在使用上可以被当作基类的对象，反之则禁止。具体表现在：
 - 派生类的对象可以隐含转换为基类对象。
 - 派生类的对象可以初始化基类的引用。
 - 派生类的指针可以隐含转换为基类的指针。
- ◆ 通过基类对象名、指针只能使用从基类继承的成员


```
#include <iostream>
using namespace std;
class Base1 { //基类Base1 定义
public:
    void display() const {
        cout << "Base1::display()" << endl;
    }
};
class Base2: public Base1 { //公有派生类Base2 定义
public:
    void display() const {
        cout << "Base2::display()" << endl;
    }
};
class Derived: public Base2 { //公有派生类Derived 定义
public:
    void display() const {
        cout << "Derived::display()" << endl;
    }
};
```

(4) 类型转换规则

```

void fun(Base1 *ptr) {           //参数为指向基类对象的指针
    ptr->display();              //"对象指针->成员名"
}
int main() { //主函数
    Base1 base1;                //声明Base1类对象
    Base2 base2;                //声明Base2类对象
    Derived derived;            //声明Derived类对象

    fun(&base1);                //用Base1对象的指针调用fun函数
    fun(&base2);                //用Base2对象的指针调用fun函数
    fun(&derived);              //用Derived对象的指针调用fun函数

    return 0;
}

```

运行结果:

```

Base1::display()
Base1::display()
Base1::display()

```

绝对不要重新定义继承而来的非虚函数

(4) 类型转换规则

在公有派生的情况下，一个派生类的对象可用于基类对象适用的地方。赋值兼容规则有三种情况：

- (1) 派生类的对象可以赋值给基类的对象。

`base _Obj = derived _Obj;`

- (2) 派生类的对象可以初始化基类的引用。

`base& base_Obj = derived_obj;`

- (3) 派生类的对象的地址可以赋给指向基类的指针。

`base *pBase = &derived_obj;`

小结

- ◆ 掌握理解三种继承方式
- ◆ 掌握理解重名成员的调用规则
- ◆ 派生过程中对基类静态成员的使用
- ◆ 类型转换规则

练习

◆ 1.若类A和类B的定义如下：

```
class A{  
    int i,j;  
public:  
    void get();  
    //... };  
class B : A{  
    int k;  
public:  
    void make();  
    //...  
};  
void B::make()  
{ k=i*j; }
```

则上述定义中，（）是非法的表达式。并说明理由。

- A. void get();
- B. int k;
- C. void make();
- D. k=i*j;

练习

- ◆ 2. ____ 提供了类对外部的界面， ____ 只能被类的成员访问，而 ____ 不允许外界访问，但允许派生类的成员访问，这样既有一定的隐藏能力，又提供了开放的界面。

A 公有成员

B 私有成员

C 私有成员函数

D 保护成员

◆ 编写程序，

- 点类(Point): 有设置、获取坐标方法;
- 圆类(Circle): 有设置、获取半径、计算面积方法;
- 圆柱体(Cylinder): 有设置、获取高、计算体积方法;
- 三者继承关系。(采用泛化方法)
- 采用聚合方法(复杂类对象) 再做一次