

Lecture 9: 继承性

第十二讲 面向对象编程：继承

学习目标：

- 通过继承现有类来创建新类
- 基类和派生类之间的关系
- protected 成员的访问
- 继承层次中构造函数和析构函数的调用



面向对象的重要特征：继承性

继承性是面向对象程序设计的又一个重要特性。继承体现了类与类之间的一种特殊关系，即一般与特殊的关系。继承是指一个新的类拥有全部被继承类的属性和方法。继承机制使得新类不仅有自己特有的属性和方法，而且有被继承类的全部属性和方法。



继承:

- 是在一个已存在的类的基础上建立一个新的类，已存在的类称为“**基类**”或“**父类(超类)**”，新建立的类称为“**派生类**”或“**子类**”。
- 是保持已有类的特性而构造新类的过程
- 派生类显示继承的基类称为**直接基类**，经过两级或更多类层次继承的类称为**间接基类**。
- **继承的目的：实现代码重用**

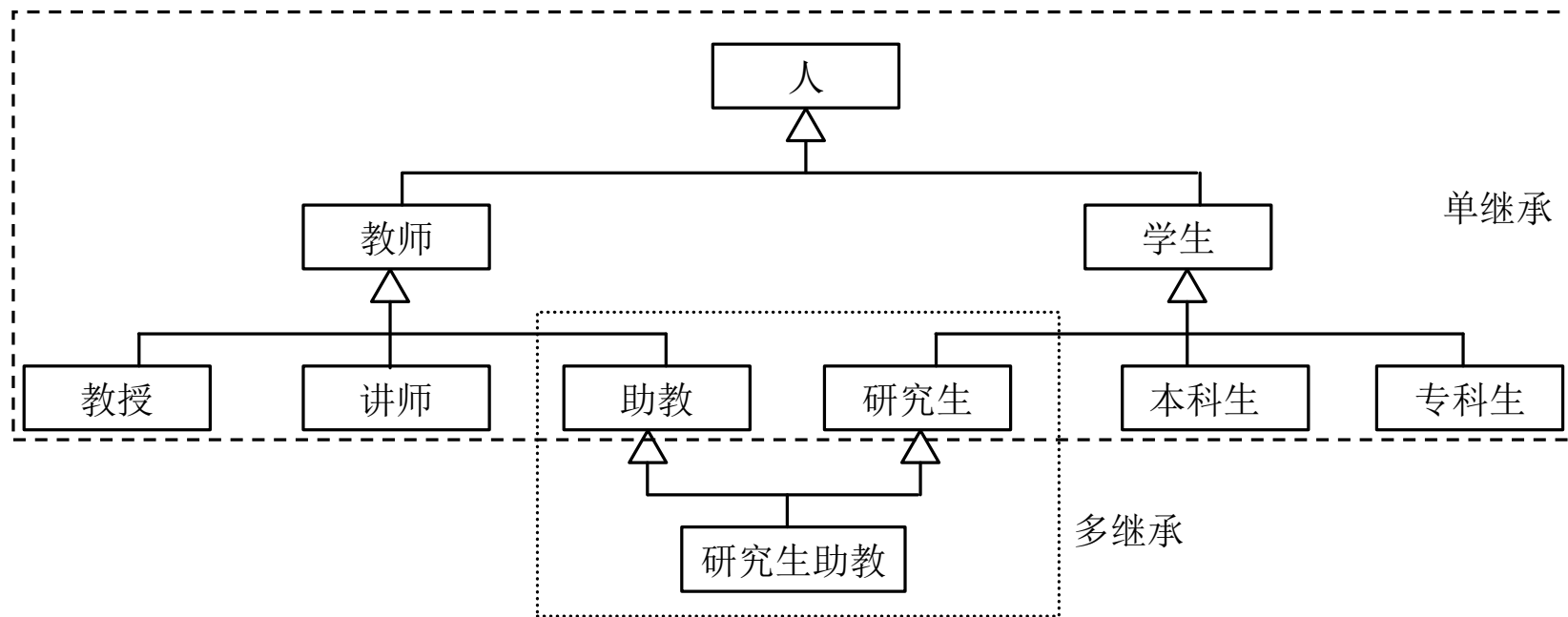


派生:

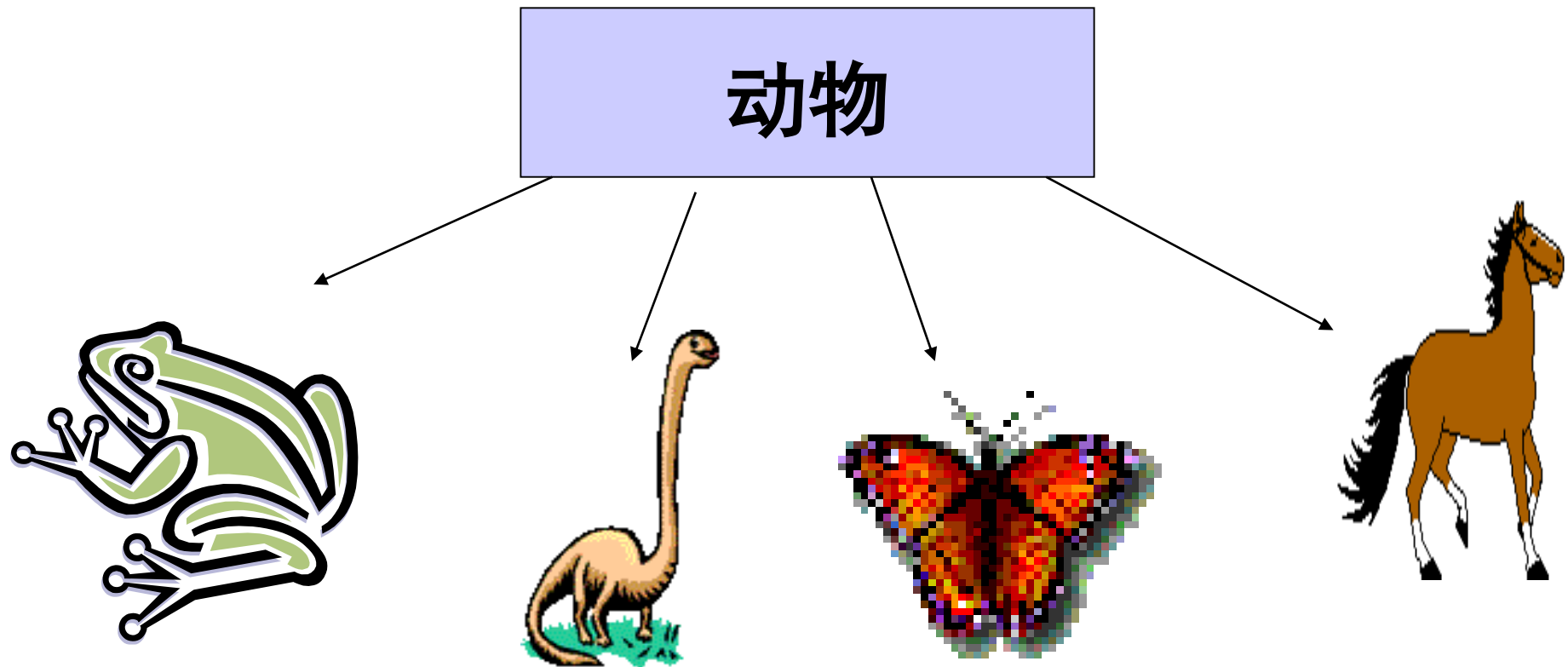
- 在已有类的基础上新增自己的特性而产生新类的过程
- 自然地表示现实世界, 是复杂的系统层次化, 提高代码的重用性, 增强语言功能, 提高软件开发效益.
- 派生类继承了基类的所有数据成员和成员函数, 并增加新的成员, 因而通常派生类比基类大得多。



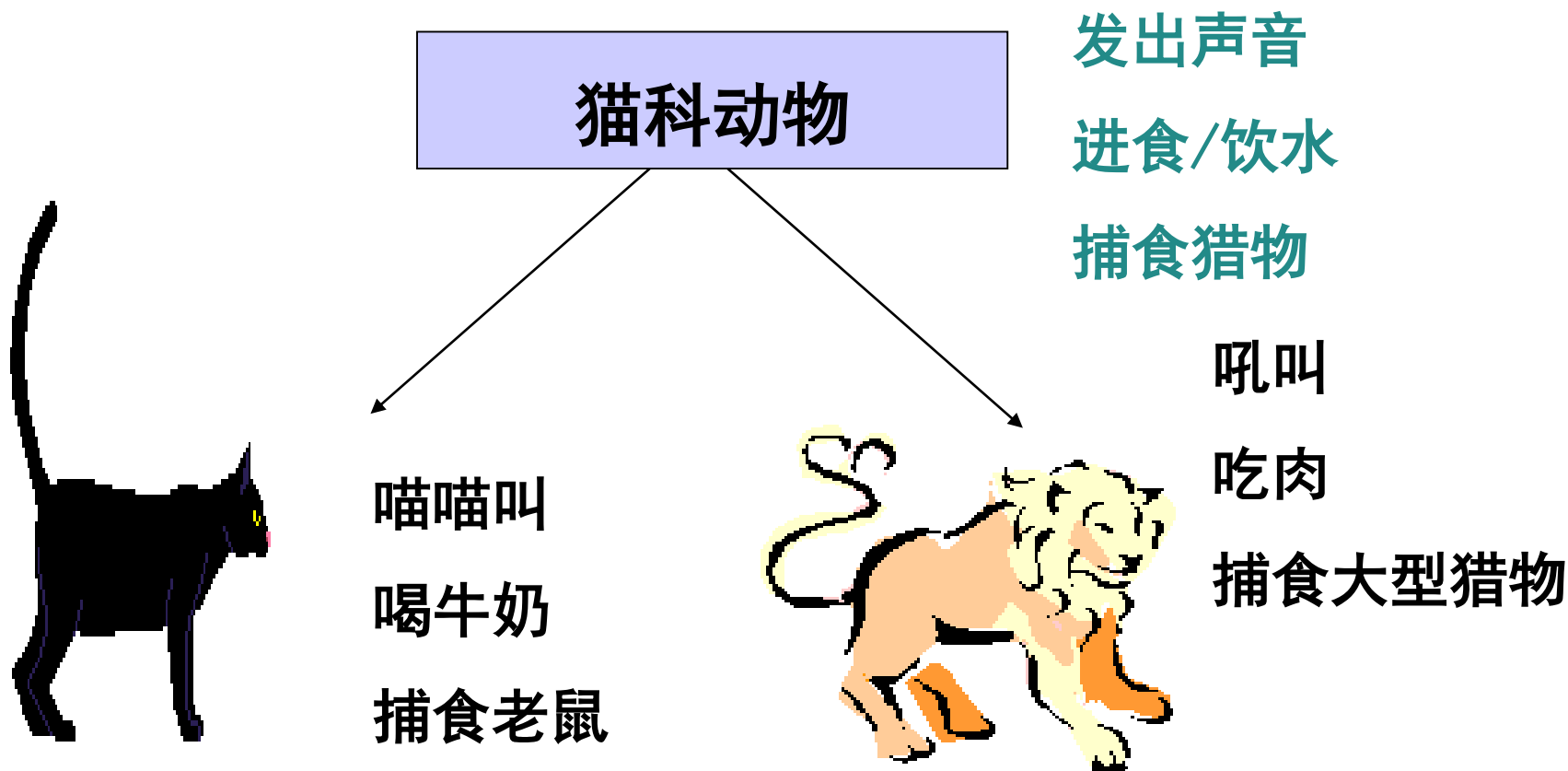
从基类产生派生类的方法一般分成两种：如果只允许一个派生类从一个惟一的基类继承产生则称做**单重继承**；如果允许一个派生类从两个或两个以上的基类继承产生，则称做**多重继承**。



1 Introduction



1 Introduction



1 Introduction

● 继承

- 软件重用的一种表现
- 在已有类的基础上创建新类
 - ✓ 使用已有类的数据和行为
 - ✓ 增加新的数据和行为



1 Introduction

● 继承

➤ 派生类从基类继承而来

✓ 派生类

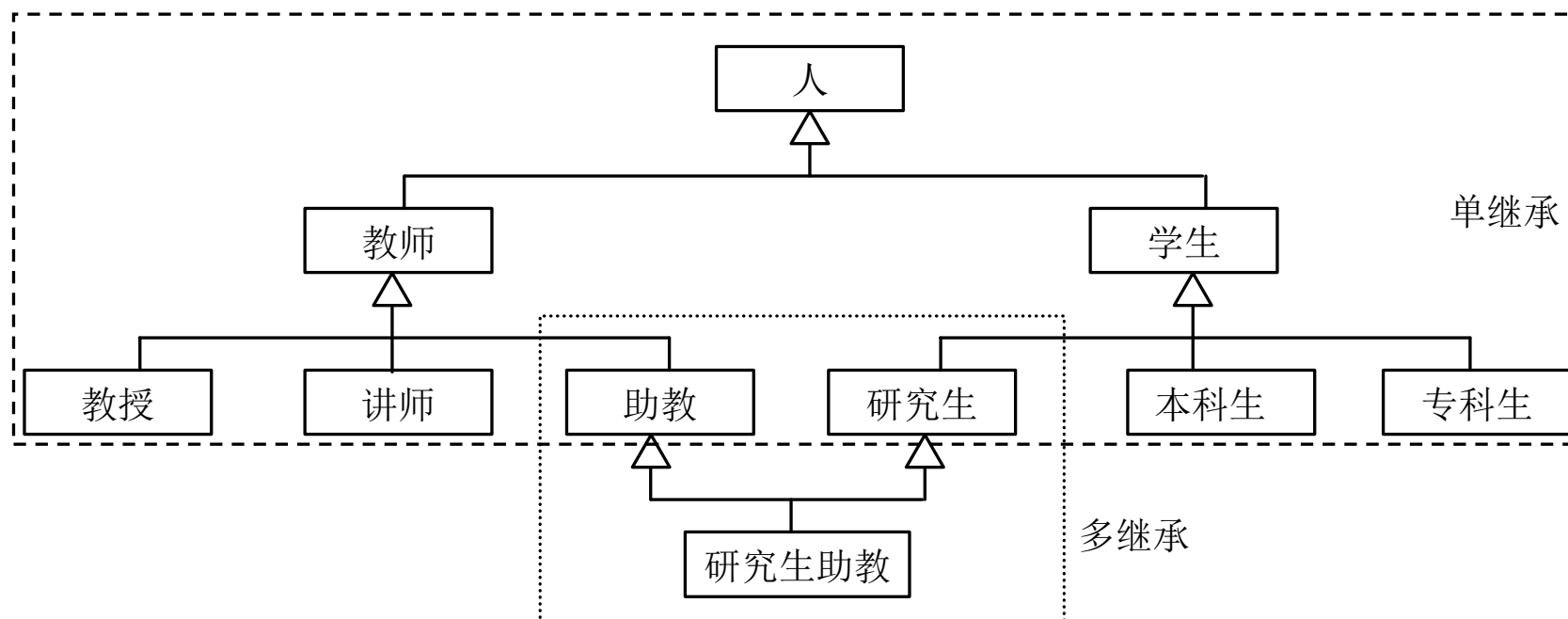
- ◇ 更加专业化的对象
- ◇ 从基类继承的行为：可以进一步定制
- ◇ 额外的行为



1 Introduction

- 类的层次结构

- 直接基类
- 间接基类
- 单继承
- 多继承



1 Introduction

● 三种类型的继承

- 继承方式指定了派生类成员以及类外对象对于从基类继承来的成员的访问权限。继承方式有三种：

public: 公有继承；

private: 私有继承；

protected: 保护继承。



1 Introduction

● “is a kind of” vs. “is a part of”

- **继承**描述的是一般类与特殊类的关系，类与类之间体现的是 “is a kind of”，即如果在逻辑上A是B的一种（is a kind of），则允许A继承B的功能和属性。例如汽车（automobile）是交通工具(vehicle) 的一种，小汽车（car）是汽车的一种。那么类automobile可以从类vehicle派生，类car可以从类automobile派生。



1 Introduction

● “is a kind of” vs. “is a part of”

- **组合**描述的是整体与部分的关系，类与类之间体现的是 “**is a part of**”，即如果在逻辑上A是B的一部分 (is a part of)，则允许A和其他数据成员组合为B。例如：发动机、车轮、电池、车门、方向盘、底盘都是小汽车的一部分，它们组合成汽车。而不能说发动机是汽车的一种。



1 Introduction



软件工程知识：派生类的成员函数**不能直接访问**其基类的private成员。

原因：如果派生类可以访问其基类的private成员，那么从该派生类继承的类也可以访问这些数据。但是这样将传递对private数据的访问权，使得信息不再隐藏。



2 Base Classes and Derived Classes

● 基类和派生类

➤ 基类通常代表了比派生类更广的范围

✓ 例如：

◇ 基类：交通工具

◇ 包括汽车，卡车，轮船，自行车等

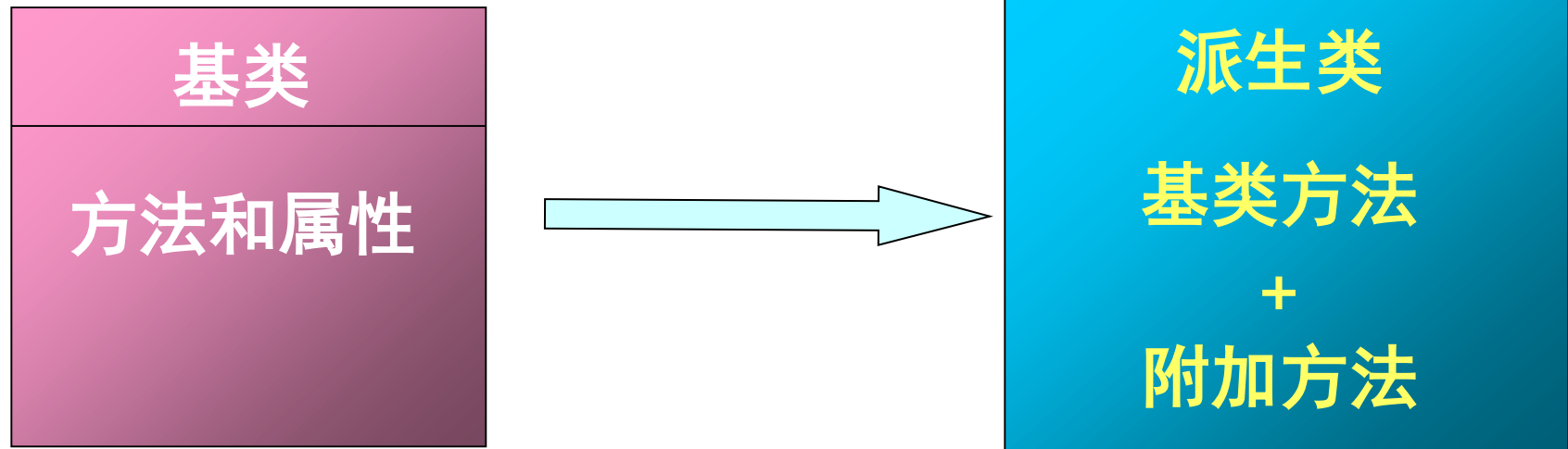
◇ 派生类：汽车

◇ 是交通工具的子集



2 Base Classes and Derived Classes

继承是允许重用现有类来构造新类的特性



2 Base Classes and Derived Classes

基类	派生类
学生	研究生；本科生
形状	圆；三角；矩形
贷款	汽车贷款；家庭贷款；抵押贷款
雇员	教职工；后勤人员
账户	支票账户；储蓄账户



派生类的定义

C++语言的派生类既可由单重继承产生，也可由多重继承产生。

C++语言派生类单重继承的定义格式如下：

```
class 派生类名: [继承方式] 基类名  
{  
    派生类新增加的成员函数  
    派生类新增加的数据成员  
};
```

“继承方式”可以是public（公有的）、protected（保护的）和private（私有的）。

```
class TwoDimensionalShape : public Shape
```



2 Base Classes and Derived Classes

直接基类和间接基类

- 直接基类

```
class A
```

```
{};
```

```
class B : public A
```

```
{};
```

// A是B的直接基类



2 Base Classes and Derived Classes

直接基类和间接基类

- 间接基类

```
class A
```

```
{};
```

```
class B : public A
```

```
{};
```

```
class C : public B
```

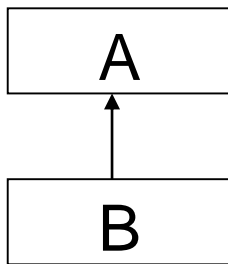
```
{};
```

// A是C的间接基类



2 Base Classes and Derived Classes

单一继承



```

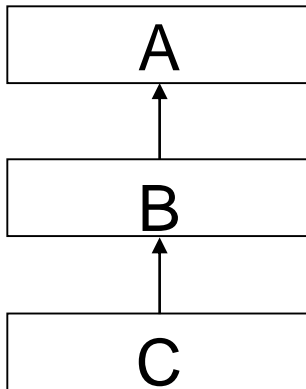
class A
{...};
class B :public A
{...};
    
```



中间为单性生殖的只有雌性个体的物种新墨西哥鞭尾蜥 (*Cnemidophorus neomexicanus*)，两边为有性生殖的物种，左为雄性小斑纹鞭尾蜥 (*C. inornatus*)，右为雌性西部鞭尾蜥 (*C. tigris*)。新墨西哥鞭尾蜥为二者自然杂交产生的物种。

2 Base Classes and Derived Classes

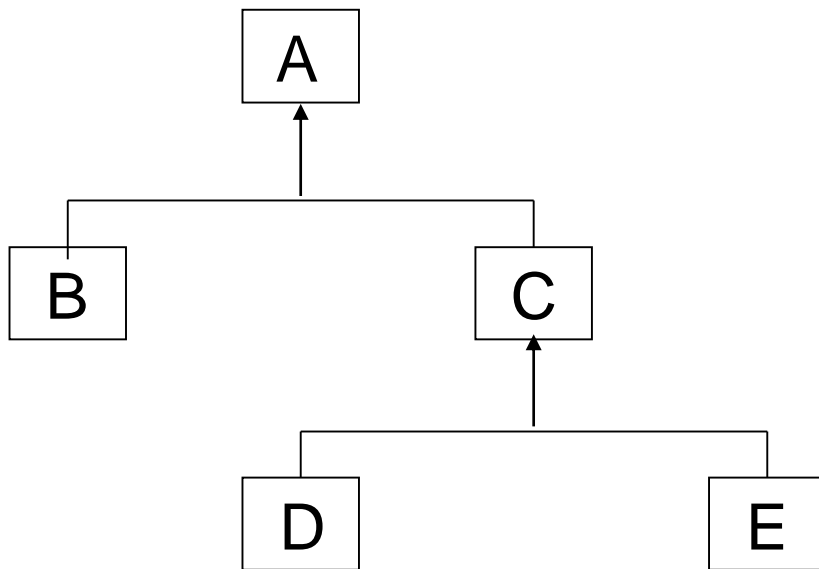
多级继承



```
class A
{...};
class B : public A
{...};
class C : public B
{...};
```

2 Base Classes and Derived Classes

层次继承

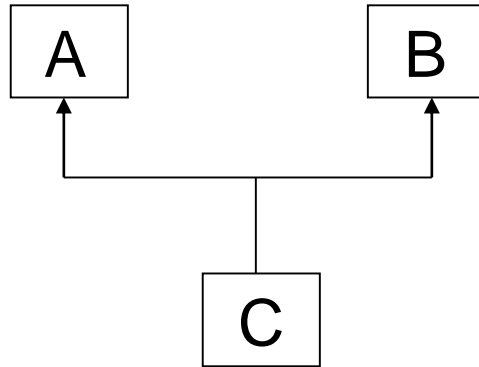


```
class A
{...};
class B :public A
{...};
class C :public A
{...};
class D :public C
{...};
class E :public C
{...};
```



2 Base Classes and Derived Classes

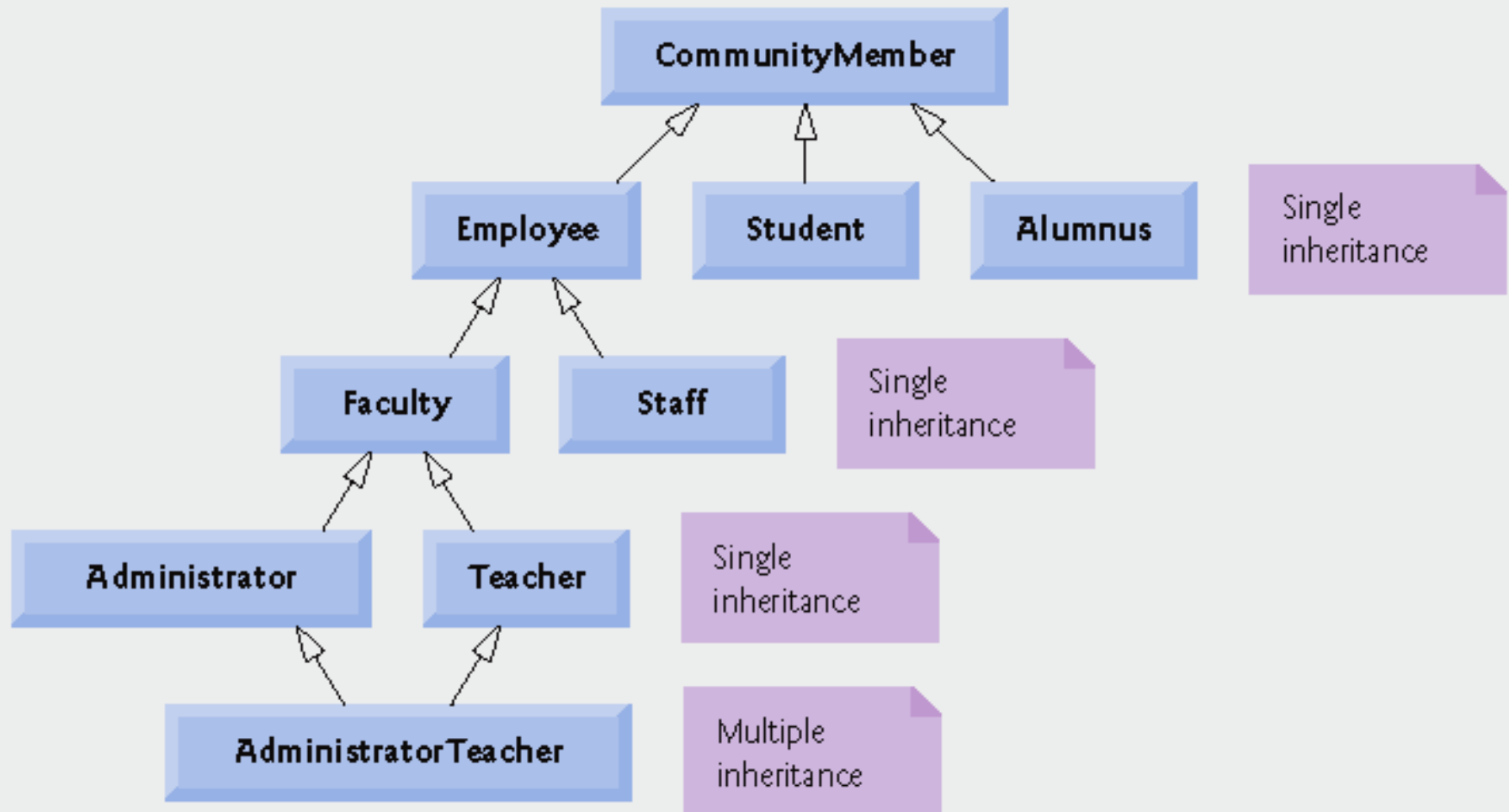
多重继承



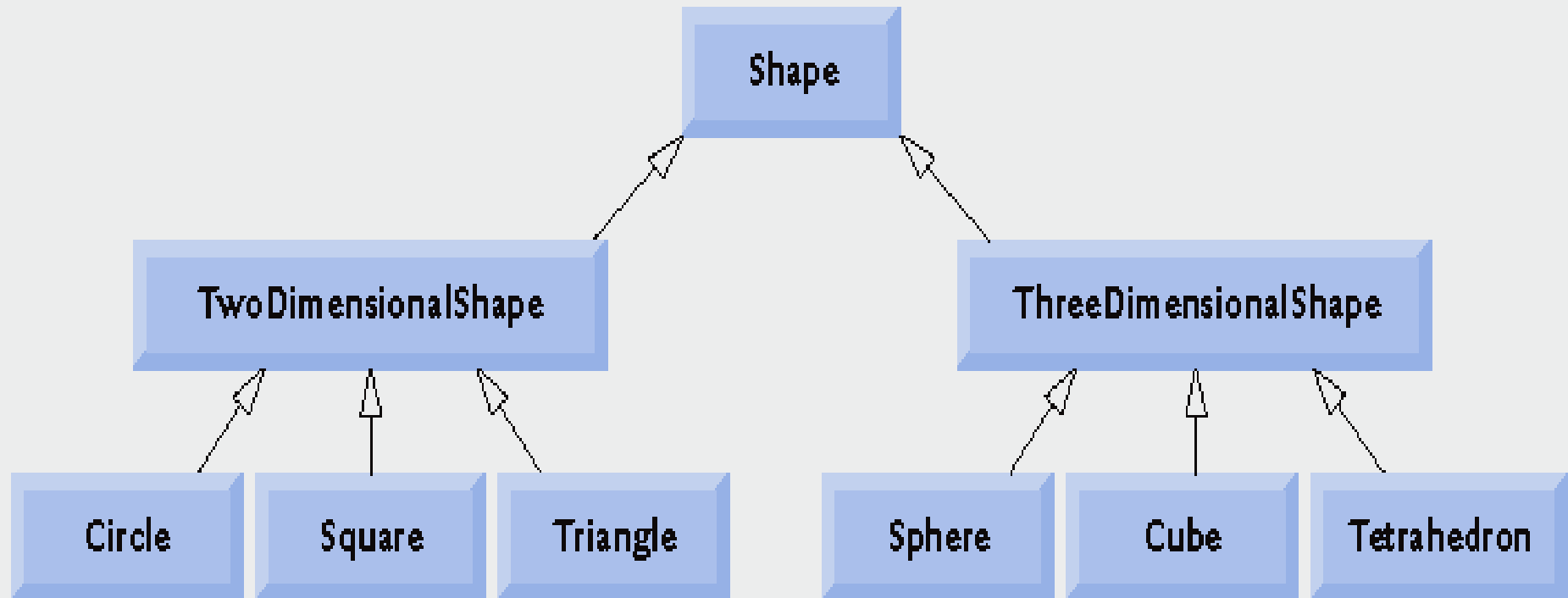
```
class A
{...};
class B
{...};
class C :public A, public B
{...};
```



2 Base Classes and Derived Classes



2 Base Classes and Derived Classes



2 Base Classes and Derived Classes

- public 继承方式时
 - 基类的私有成员：不能直接访问
 - ✓ 被派生类继承，通过公有成员函数来访问
 - 基类的 public 和 protected 成员
 - ✓ 按原来的访问方式被继承
 - 友元函数：不能被继承



2 Base Classes and Derived Classes

● protect 继承方式时

- 类的public成员能够被程序中所有函数访问
- 类的private成员只能被基类的成员函数和友元访问
- 类的protected访问是public访问和private访问之间的中间层次。
 - ✓ 基类的protected成员能被
 - ◇ 基类的成员和友元访问
 - ◇ 由基类所派生出的任何类的成员和友元访问。
 - ✓ 派生类成员简单地使用成员名就可以引用基类的public成员和protected成员。



由protected 声明的成员为受保护成员，受保护成员不能被外界引用（这点和私有成员类似），但它可以被派生类的成员函数引用（这点和公有成员类似）。

因此，受保护成员具有公有成员和私有成员的双重特性，对派生类的成员函数而言，它是公有成员。但对所在类之外定义的有关函数或类的对象而言，则是私有成员。



基类的私有数据成员被派生类（`public`派生或`private`派生）继承后变为“不可访问的成员”。如果在派生类中引用基类的私有数据成员，可以将基类的数据成员声明为`protected`。这样既不会破坏类的封装性，其数据成员还能被派生类的成员函数引用。

实现了数据隐藏，又方便继承，实现代码重用



三种继承方式下，基类成员在派生类中的访问控制属性总结如图：

基类属性 继承方式	public	protected	private
public	public	protected	不可访问
protected	protected	protected	不可访问
private	private	private	不可访问



- **protected 类型**

- 介于 public 和 private 之间
- 对protected 的访问，可以是
 - ✓ 基类成员
 - ✓ 基类友元
 - ✓ 派生类成员
 - ✓ 派生类友元



因此：

● 派生类成员

- 可以简单的使用成员名称来访问基类的 `public` 和 `protected` 成员
- 如果派生类中重新定义了基类的成员，可以通过 `:: + 基类成员名称`来访问基类成员



```
class A {  
    protected:  
        int x;  
}
```

```
int main()  
{  
    A a;  
    a.x=5;  //错误。此处 x 相当于私有成员  
}
```



4 Relationship between Base Classes and Derived Classes

- 后面的例子描述了基类和派生类之间的关系
 - 基类：CommissionEmployee（5个参数）
 - ✓ First name, last name, SSN, commission rate, gross sale amount
 - 派生类：BasePlusCommissionEmployee（增1参数）
 - ✓ First name, last name, SSN, commission rate, gross sale amount
 - ✓ **base salary**




```
1 // Fig. 12.4: CommissionEmployee.h
2 // CommissionEmployee class definition represents a commission employee.
3 #ifndef COMMISSION_H
4 #define COMMISSION_H
5
6 #include <string> // C++ standard string class
7 using std::string;
8
9 class CommissionEmployee
10 {
11 public:
12     CommissionEmployee( const string &, const string &, const string &,
13                        double = 0.0, double = 0.0 );
14
15     void setFirstName( const string & ); // set first name
16     string getFirstName() const; // return first name
17
18     void setLastName( const string & ); // set last name
19     string getLastName() const; // return last name
20
21     void setSocialSecurityNumber( const string & ); // set SSN
22     string getSocialSecurityNumber() const; // return SSN
23
24     void setGrossSales( double ); // set gross sales amount
25     double getGrossSales() const; // return gross sales amount
26
27     void setCommissionRate( double ); // set commission rate (percentage)
28     double getCommissionRate() const; // return commission rate
```

Class **CommissionEmployee** constructor



```
29
30 double earnings() const; // calculate earnings
31 void print() const; // print CommissionEmployee object
32 private:
33     string firstName;
34     string lastName;
35     string socialSecurityNumber;
36     double grossSales; // gross weekly sales
37     double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

Declare **private**
data members



```

1 // Fig. 12.5: CommissionEmployee.cpp
2 // Class CommissionEmployee member-function definitions.
3 #include <iostream>
4 using std::cout;
5
6 #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8 // constructor
9 CommissionEmployee::CommissionEmployee(
10     const string &first, const string &last, const string &ssn,
11     double sales, double rate )
12 {
13     firstName = first; // should validate
14     lastName = last;   // should validate
15     socialSecurityNumber = ssn; // should validate
16     setGrossSales( sales ); // validate and store gross sales
17     setCommissionRate( rate ); // validate and store commission rate
18 } // end CommissionEmployee constructor
19
20 // set first name
21 void CommissionEmployee::setFirstName( const string &first )
22 {
23     firstName = first; // should validate
24 } // end function setFirstName
25
26 // return first name
27 string CommissionEmployee::getFirstName() const
28 {
29     return firstName;
30 } // end function getFirstName

```

Initialize data members

能否写成:

grossSales = sales ;
commissionRate = rate ;



```

31
32 // set last name
33 void CommissionEmployee::setLastName( const string &last )
34 {
35     lastName = last; // should validate
36 } // end function setLastName
37
38 // return last name
39 string CommissionEmployee::getLastName() const
40 {
41     return lastName;
42 } // end function getLastName
43
44 // set social security number
45 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
46 {
47     socialSecurityNumber = ssn; // should validate
48 } // end function setSocialSecurityNumber
49
50 // return social security number
51 string CommissionEmployee::getSocialSecurityNumber() const
52 {
53     return socialSecurityNumber;
54 } // end function getSocialSecurityNumber
55
56 // set gross sales amount
57 void CommissionEmployee::setGrossSales( double sales )
58 {
59     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
60 } // end function setGrossSales

```

Function **setGrossSales**
validates gross sales amount

调用函数而不是直接赋值的原因
是需要检查值的合法性




```
61
62 // return gross sales amount
63 double CommissionEmployee::getGrossSales() const
64 {
65     return grossSales;
66 } // end function getGrossSales
67
68 // set commission rate
69 void CommissionEmployee::setCommissionRate( double rate )
70 {
71     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
72 } // end function setCommissionRate
73
74 // return commission rate
75 double CommissionEmployee::getCommissionRate() const
76 {
77     return commissionRate;
78 } // end function getCommissionRate
```

Function **setCommissionRate**
validates commission rate



79

```
80 // calculate earnings
```

```
81 double CommissionEmployee::earnings() const
```

```
82 {
```

```
83     return commissionRate * grossSales;
```

```
84 } // end function earnings
```

85

```
86 // print CommissionEmployee object
```

```
87 void CommissionEmployee::print() const
```

```
88 {
```

```
89     cout << "commission employee: " << firstName << ' ' << lastName
```

```
90         << "\nsocial security number: " << socialSecurityNumber
```

```
91         << "\ngross sales: " << grossSales
```

```
92         << "\ncommission rate: " << commissionRate;
```

```
93 } // end function print
```

Function **earnings**
calculates earnings

Function **print** displays
CommissionEmployee object



```

1 // Fig. 12.6: fig12_06.cpp
2 // Testing class CommissionEmployee.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6 using std::fixed;
7
8 #include <iomanip>
9 using std::setprecision;
10
11 #include "CommissionEmployee.h" // CommissionEmployee class definition
12
13 int main()
14 {
15     // instantiate a CommissionEmployee object
16     CommissionEmployee employee(
17         "Sue", "Jones", "222-22-2222", 10000, .06 );
18
19     // set floating-point output formatting
20     cout << fixed << setprecision( 2 );
21
22     // get commission employee data
23     cout << "Employee information obtained by get functions: \n"
24         << "\nFirst name is " << employee.getFirstName()
25         << "\nLast name is " << employee.getLastName()
26         << "\nSocial security number is "
27         << employee.getSocialSecurityNumber()
28         << "\nGross sales is " << employee.getGrossSales()
29         << "\nCommission rate is " << employee.getCommissionRate() << endl;

```

Instantiate **CommissionEmployee** object

Use **CommissionEmployee**'s *get functions* to retrieve the object's instance variable values



30

```

31 employee.setGrossSales( 8000 ); // set gross sales
32 employee.setCommissionRate( .1 ); // set commission rate

```

33

```

34 cout << "\nUpdated employee information output
35 << endl;

```

Use **CommissionEmployee**'s *set* functions to change the object's instance variable values

```

36 employee.print(); // display the new employee information

```

37

```

38 // display the employee's earnings

```

Call object's **print** function to display employee information

```

39 cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

```

40

```

41 return 0;

```

```

42 } // end main

```

Call object's **earnings** function to calculate earnings

Employee information obtained by get functions:

```

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales is 10000.00
Commission rate is 0.06

```

Updated employee information output by print function:

```

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 8000.00
commission rate: 0.10

```

Employee's earnings: \$800.00



6 Creating a BasePlusCommissionEmployee Class Without Using Inheritance

- 对于派生类 BasePlusCommissionEmployee
 - 大部分代码与 CommissionEmployee 相同
 - ✓ private 数据成员
 - ✓ public 成员函数
 - ✓ 构造函数
 - 只有额外的
 - ✓ private 数据成员 **baseSalary**
 - ✓ **setBaseSalary** 和 **getBaseSalary** 成员函数



```

1 // Fig. 12.7: BasePlusCommissionEmployee.h
2 // BasePlusCommissionEmployee class definition represents an employee
3 // that receives a base salary in addition to commission.
4 #ifndef BASEPLUS_H
5 #define BASEPLUS_H
6
7 #include <string> // C++ standard string class
8 using std::string;
9
10 class BasePlusCommissionEmployee
11 {
12 public:
13     BasePlusCommissionEmployee( const string &, const string &,
14                                const string &, double = 0.0, double = 0.0, double = 0.0 );
15
16     void setFirstName( const string & ); // set first name
17     string getFirstName() const; // return first name
18
19     void setLastName( const string & ); // set last name
20     string getLastName() const; // return last name
21
22     void setSocialSecurityNumber( const string & ); // set SSN
23     string getSocialSecurityNumber() const; // return SSN
24
25     void setGrossSales( double ); // set gross sales amount
26     double getGrossSales() const; // return gross sales amount
27
28     void setCommissionRate( double ); // set commission rate
29     double getCommissionRate() const; // return commission rate

```

Constructor takes one more argument, which specifies the base salary. 多一个参数



30

```
31 void setBaseSalary( double ); // set base salary  
32 double getBaseSalary() const; // return base salary
```

33

```
34 double earnings() const; // calculate earnings  
35 void print() const; // print BasePlusCommissionEmployee
```

Define *get* and *set* functions
for data member `baseSalary`

```
36 private:
```

```
37 string firstName;
```

```
38 string lastName;
```

```
39 string socialSecurityNumber;
```

```
40 double grossSales; // gross weekly sales
```

```
41 double commissionRate; // commission percentage
```

```
42 double baseSalary; // base salary
```

```
43 }; // end class BasePlusCommissionEmployee
```

44

```
45 #endif
```

Add data member `baseSalary`



```

1 // Fig. 12.8: BasePlusCommissionEmployee.cpp
2 // Class BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 using std::cout;
5
6 // BasePlusCommissionEmployee class definition
7 #include "BasePlusCommissionEmployee.h"
8
9 // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11     const string &first, const string &last, const string &ssn,
12     double sales, double rate, double salary )
13 {
14     firstName = first; // should validate
15     lastName = last; // should validate
16     socialSecurityNumber = ssn; // should validate
17     setGrossSales( sales ); // validate and store gross sales
18     setCommissionRate( rate ); // validate and store commission rate
19     setBaseSalary( salary ); // validate and store base salary
20 } // end BasePlusCommissionEmployee constructor
21
22 // set first name
23 void BasePlusCommissionEmployee::setFirstName( const string &first )
24 {
25     firstName = first; // should validate
26 } // end function setFirstName

```

Constructor takes one more argument,
which specifies the base salary

Use function setBaseSalary to validate data




```
27
28 // return first name
29 string BasePlusCommissionEmployee::getFirstName() const
30 {
31     return firstName;
32 } // end function getFirstName
33
34 // set last name
35 void BasePlusCommissionEmployee::setLastName( const string &last )
36 {
37     lastName = last; // should validate
38 } // end function setLastName
39
40 // return last name
41 string BasePlusCommissionEmployee::getLastName() const
42 {
43     return lastName;
44 } // end function getLastName
45
46 // set social security number
47 void BasePlusCommissionEmployee::setSocialSecurityNumber(
48     const string &ssn )
49 {
50     socialSecurityNumber = ssn; // should validate
51 } // end function setSocialSecurityNumber
52
```



```
53 // return social security number
54 string BasePlusCommissionEmployee::getSocialSecurityNumber() const
55 {
56     return socialSecurityNumber;
57 } // end function getSocialSecurityNumber
58
59 // set gross sales amount
60 void BasePlusCommissionEmployee::setGrossSales( double sales )
61 {
62     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
63 } // end function setGrossSales
64
65 // return gross sales amount
66 double BasePlusCommissionEmployee::getGrossSales() const
67 {
68     return grossSales;
69 } // end function getGrossSales
70
71 // set commission rate
72 void BasePlusCommissionEmployee::setCommissionRate( double rate )
73 {
74     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
75 } // end function setCommissionRate
76
77 // return commission rate
78 double BasePlusCommissionEmployee::getCommissionRate() const
79 {
80     return commissionRate;
81 } // end function getCommissionRate
82
```



```

83 // set base salary
84 void BasePlusCommissionEmployee::setBaseSalary( double salary )
85 {
86     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
87 } // end function setBaseSalary
88
89 // return base salary
90 double BasePlusCommissionEmployee::getBaseSalary() const
91 {
92     return baseSalary;
93 } // end function getBaseSalary
94
95 // calculate earnings
96 double BasePlusCommissionEmployee::earnings() const
97 {
98     return baseSalary + ( commissionRate * grossSales );
99 } // end function earnings
100
101 // print BasePlusCommissionEmployee object
102 void BasePlusCommissionEmployee::print() const
103 {
104     cout << "base-salaried commission employee: " << firstName << ' '
105         << lastName << "\nsocial security number: " << socialSecurityNumber
106         << "\ngross sales: " << grossSales
107         << "\ncommission rate: " << commissionRate
108         << "\nbase salary: " << baseSalary;
109 } // end function print

```

Function setBaseSalary validates data and sets instance variable baseSalary

Function getBaseSalary returns the value of instance variable baseSalary

Update function earnings to calculate the earnings of a base-salaried commission employee

Update function print to display base salary



```
1 // Fig. 12.9: fig12_09.cpp
2 // Testing class BasePlusCommissionEmployee.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6 using std::fixed;
7
8 #include <iomanip>
9 using std::setprecision;
10
11 // BasePlusCommissionEmployee class definition
12 #include "BasePlusCommissionEmployee.h"
13
14 int main()
15 {
16     // instantiate BasePlusCommissionEmployee object
17     BasePlusCommissionEmployee
18         employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20     // set floating-point output formatting
21     cout << fixed << setprecision( 2 );
22
```



```

23 // get commission employee data
24 cout << "Employee information obtained by get functions: \n"
25     << "\nFirst name is " << employee.getFirstName()
26     << "\nLast name is " << employee.getLastName()
27     << "\nSocial security number is "
28     << employee.getSocialSecurityNumber()
29     << "\nGross sales is " << employee.getGrossSales()
30     << "\nCommission rate is " << employee.getCommissionRate()
31     << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33 employee.setBaseSalary( 1000 ); // set base salary
34
35 cout << "\nUpdated employee information output by print function: \n"
36     << endl;
37 employee.print(); // display the new employee information
38
39 // display the employee's earnings
40 cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42 return 0;
43 } // end main

```



Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: \$1200.00



6 Creating a BasePlusCommissionEmployee Class Without Using Inheritance



软件工程知识：从一个类向另一个类复制粘贴相同代码，可能造成错误在多个源代码文件中扩散。当我们想要一个类复用别的类的数据成员和成员函数时，可以使用**继承**，而不是“复制粘贴”，从而避免代码错误的扩散。



6 Creating a BasePlusCommissionEmployee Class Without Using Inheritance



软件工程知识：使用继承时，类层次中所有类**共同的数据成员和成员函数在基类中声明**。当需要对这些共同特征进行修改时，程序员只需在基类中进行修改，于是派生类也就继承了相应的修改。如果不采用继承机制，则需要对所有包含代码副本的源代码文件进行修改。



7 Creating BasePlusCommissionEmployee by using Inheritance method

- 对于派生类 BasePlusCommissionEmployee
 - 从 CommissionEmployee 继承而来
 - 是一个 CommissionEmployee
 - 继承了所有 public 成员
 - 构造函数不能被继承
 - ✓ 使用基类初始化语法来初始化基类数据成员
 - 具有一个(新的)数据成员 baseSalary



```

1 // Fig. 12.10: BasePlusCommissionEmployee.h
2 // BasePlusCommissionEmployee class derived from class
3 // CommissionEmployee.
4 #ifndef BASEPLUS_H
5 #define BASEPLUS_H
6
7 #include <string> // C++ standard string class
8 using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15     BasePlusCommissionEmployee( const string &, const string &,
16                                const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18     void setBaseSalary( double ); // set base salary
19     double getBaseSalary() const; // return base salary
20
21     double earnings() const; // calculate earnings
22     void print() const; // print BasePlusCommissionEmployee object
23 private:
24     double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif

```

Include the base-class header file in the derived-class header file

Class BasePlusCommissionEmployee derives publicly from class CommissionEmployee

这两个函数在基类中已经定义过

```

30 double earnings() const; // calculate earnings
31 void print() const; // print CommissionEmployee object

```



```
1 // Fig. 12.11: BasePlusCommissionEmployee.cpp
2 // Class BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 using std::cout;
```

```
5
6 // BasePlusCommissionEmployee class definition
7 #include "BasePlusCommissionEmployee.h"
```

```
8
9 // constructor
```

```
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11     const string &first, const string &last, const string &ssn,
12     double sales, double rate, double salary )
```

```
13 // explicitly call base-class constructor 隐式地调用基类构造函数初始化private变量
14 : CommissionEmployee( first, last, ssn, sales, rate )
```

```
15 {
16     setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
```

```
18
19 // set base salary
```

```
20 void BasePlusCommissionEmployee::setBaseSalary( double salary )
```

```
21 {
22     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23 } // end function setBaseSalary
```

```
24
25 // return base salary
```

```
26 double BasePlusCommissionEmployee::getBaseSalary() const
```

```
27 {
28     return baseSalary;
29 } // end function getBaseSalary
```

Initialize base class data member by calling the base-class constructor using base-class initializer syntax



续

- 构造函数引入了基类初始化器
 - 使用成员初始化器将参数传给基类的构造函数
 - 派生类构造函数调用其基类的构造函数来初始化从基类继承来的数据成员
- 对于基类中的 `private` 数据成员
 - 派生类不能直接访问
 - 通过基类提供的 `private` 类型的 `get` 类函数间接访问



30

31 // calculate earnings

32 double BasePlusCommissionEmployee::earnings() const

33 {

34 // derived class cannot access the base class's private data

35 return baseSalary + (commissionRate * grossSales);

36 } // end function earnings

37

38 // print BasePlusCommissionEmployee object

39 void BasePlusCommissionEmployee::print() const

40 {

41 // derived class cannot access the base class's private data

42 cout << "base-salaried commission employee: " << firstName << ' '

43 << lastName << "\nsocial security number: " << socialSecurityNumber

44 << "\ngross sales: " << grossSales

45 << "\ncommission rate: " << commissionRate

46 << "\nbase salary: " << baseSalary;

47 } // end function print

Compiler generates errors because base class's data member commissionRate and grossSales are private

对于基类的数据成员 commissionRate 与 grossSales 是 private 类型的，因此，即使在其派生类中也不能直接访问，只能通过基类提供的 public 类型 **get** 函数进行访问。



```
C:\cpphttp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(35) :  
error C2248: 'CommissionEmployee::commissionRate' :  
cannot access private member declared in class 'CommissionEmployee'  
C:\cpphttp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(37) :  
see declaration of 'CommissionEmployee::commissionRate'  
C:\cpphttp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :  
see declaration of 'CommissionEmployee'  
  
C:\cpphttp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(35) :  
error C2248: 'CommissionEmployee::grossSales' :  
cannot access private member declared in class 'CommissionEmployee'  
C:\cpphttp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(36) :  
see declaration of 'CommissionEmployee::grossSales'  
C:\cpphttp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :  
see declaration of 'CommissionEmployee'  
  
C:\cpphttp5_examples\ch12\Fig12_10_11\BasePlusCommission-Employee.cpp(42) :  
error C2248: 'CommissionEmployee::firstName' :  
cannot access private member declared in class 'CommissionEmployee'  
C:\cpphttp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(33) :  
see declaration of 'CommissionEmployee::firstName'  
C:\cpphttp5_examples\ch12\Fig12_10_11\CommissionEmployee.h(10) :  
see declaration of 'CommissionEmployee'
```



- 如果派生类的构造函数没有显式调用基类的构造函数，编译器将会隐式调用基类的默认构造函数，如果该类没有默认构造函数，编译器将会发布错误信息。
 - 若类有显式的构造函数，那么编译器将不提供默认构造函数。
- 派生类构造函数调用其基类构造函数时，如果传递给基类构造函数的参数的个数和数据类型与基类构造函数的中的相应定义不符，将导致编译错误。



8 Using protected Data

- 使用 protected 数据成员

- 效果之一：

- ✓ 允许派生类BasePlusCommissionEmployee直接访问




```
1 // Fig. 12.12: CommissionEmployee.h
2 // CommissionEmployee class definition with protected data.
3 #ifndef COMMISSION_H
4 #define COMMISSION_H
5
6 #include <string> // C++ standard string class
7 using std::string;
8
9 class CommissionEmployee
10 {
11 public:
12     CommissionEmployee( const string &, const string &, const string &,
13         double = 0.0, double = 0.0 );
14
15     void setFirstName( const string & ); // set first name
16     string getFirstName() const; // return first name
17
18     void setLastName( const string & ); // set last name
19     string getLastName() const; // return last name
20
21     void setSocialSecurityNumber( const string & ); // set SSN
22     string getSocialSecurityNumber() const; // return SSN
23
```



```
24 void setGrossSales( double ); // set gross sales amount
25 double getGrossSales() const; // return gross sales amount
26
27 void setCommissionRate( double ); // set commission rate
28 double getCommissionRate() const; // return commission rate
29
30 double earnings() const; // calculate earnings
31 void print() const; // print CommissionEmployee object
```

```
32 protected:
```

```
33     string firstName;
34     string lastName;
35     string socialSecurityNumber;
36     double grossSales; // gross weekly sales
37     double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

Declare protected data



```

1 // Fig. 12.14: BasePlusCommissionEmployee.h
2 // BasePlusCommissionEmployee class derived from class
3 // CommissionEmployee.
4 #ifndef BASEPLUS_H
5 #define BASEPLUS_H
6
7 #include <string> // C++ standard string class
8 using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15     BasePlusCommissionEmployee( const string &, const string &,
16         const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18     void setBaseSalary( double ); // set base salary
19     double getBaseSalary() const; // return base salary
20
21     double earnings() const; // calculate earnings
22     void print() const; // print BasePlusCommissionEmployee object
23 private:
24     double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif

```

BasePlusCommissionEmployee still inherits publicly from CommissionEmployee

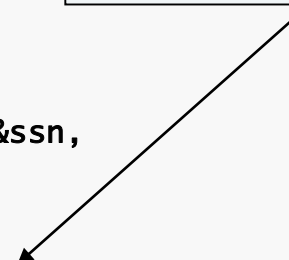



```

1 // Fig. 12.15: BasePlusCommissionEmployee.cpp
2 // Class BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 using std::cout;
5
6 // BasePlusCommissionEmployee class definition
7 #include "BasePlusCommissionEmployee.h"
8
9 // constructor
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11     const string &first, const string &last, const string &ssn,
12     double sales, double rate, double salary )
13 // explicitly call base-class constructor
14 : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16     setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
18
19 // set base salary
20 void BasePlusCommissionEmployee::setBaseSalary( double salary )
21 {
22     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23 } // end function setBaseSalary
24
25 // return base salary
26 double BasePlusCommissionEmployee::getBaseSalary() const
27 {
28     return baseSalary;
29 } // end function getBaseSalary

```

Call base-class constructor using
base-class initializer syntax



```

30
31 // calculate earnings
32 double BasePlusCommissionEmployee::earnings() const
33 {
34     // can access protected data of base class
35     return baseSalary + ( commissionRate * grossSales );
36 } // end function earnings
37
38 // print BasePlusCommissionEmployee object
39 void BasePlusCommissionEmployee::print() const
40 {
41     // can access protected data of base class
42     cout << "base-salaried commission employee: " << firstName << ' '
43     << lastName << "\nsocial security number: " << socialSecurityNumber
44     << "\ngross sales: " << grossSales
45     << "\ncommission rate: " << commissionRate
46     << "\nbase salary: " << baseSalary;
47 } // end function print

```

Directly access base class's protected data

在基类与派生类中都定义并实现了函数`earning()`和`print()`，那么在具体应用时该调用哪一个呢？



```
1  // Fig. 12.16: fig12_16.cpp
2  // Testing class BasePlusCommissionEmployee.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // BasePlusCommissionEmployee class definition
12 #include "BasePlusCommissionEmployee.h"
13
14 int main()
15 {
16     // instantiate BasePlusCommissionEmployee object
17     BasePlusCommissionEmployee
18         employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20     // set floating-point output formatting
21     cout << fixed << setprecision( 2 );
22
```



```
23  // get commission employee data
24  cout << "Employee information obtained by get functions: \n"
25      << "\nFirst name is " << employee.getFirstName()
26      << "\nLast name is " << employee.getLastName()
27      << "\nSocial security number is "
28      << employee.getSocialSecurityNumber()
29      << "\nGross sales is " << employee.getGrossSales()
30      << "\nCommission rate is " << employee.getCommissionRate()
31      << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33  employee.setBaseSalary( 1000 ); // set base salary
34
35  cout << "\nUpdated employee information output by print function: \n"
36      << endl;
37  employee.print(); // display the new employee information
38
39  // display the employee's earnings
40  cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42  return 0;
43 } // end main
```



Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: \$1200.00



8 Using protected Data

● 使用 protected 数据成员

➤ 优点

✓ 派生类可以直接访问基类数据成员

◇ 直接访问protected数据成员，可以使用避免 *set/get* 成员函数的调用开销，所以继承protected数据成员会使得程序的性能稍微有所提高。



8 Using protected Data

● 使用 protected 数据成员

➤ 缺点

- ✓ 无有效性检测：派生类可以赋非法值(set函数在基类中，派生类可以不使用它而直接赋值)
- ✓ 依赖于实现
 - ◇ 派生类依赖于基类的实现
 - ◇ 基类实现的改变会导致派生类的改变



8 Using protected Data



软件工程知识：即使可以直接修改数据成员的值，但可能的情况下，最好用成员函数修改和获取数据成员的值，set成员函数可以阻止给数据成员赋不合适的值，而get成员函数则有助于控制给客户的数据表达式。



8 Using protected Data



性能提示： 在一个派生类构造函数中，初始化成员对象和在成员初始化列表中**显式调用基类构造函数**可防止重复进行初始化。重复初始化会首先调用一个默认的构造函数，然后再在派生类构造函数内修改数据成员的值。



9 Using private Data

● 重新检查继承层次

➤ 使用良好的软件工程实践

- ✓ 将数据成员声明为 `private`

- ✓ 提供 `public` 类型的 `get` 和 `set` 函数

- ✓ 使用 `get` 获得数据成员的值



```
1 // Fig. 12.17: CommissionEmployee.h
9 class CommissionEmployee
10 {
11 public:
12     CommissionEmployee( const string &, const string &, const string &,
13         double = 0.0, double = 0.0 );
14
15     void setFirstName( const string & ); // set first name
16     string getFirstName() const; // return first name
17
18     void setLastName( const string & ); // set last name
19     string getLastName() const; // return last name
20
21     void setSocialSecurityNumber( const string & ); // set SSN
22     string getSocialSecurityNumber() const; // return SSN
23
24     void setGrossSales( double ); // set gross sales amount
25     double getGrossSales() const; // return gross sales amount
26
27     void setCommissionRate( double ); // set commission rate
28     double getCommissionRate() const; // return commission rate
```



```
29
30 double earnings() const; // calculate earnings
31 void print() const; // print CommissionEmployee object
32 private:
33     string firstName;
34     string lastName;
35     string socialSecurityNumber;
36     double grossSales; // gross weekly sales
37     double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
```



```
1  // Fig. 12.18: CommissionEmployee.cpp
9  CommissionEmployee::CommissionEmployee(
10     const string &first, const string &last, const string &ssn,
11     double sales, double rate )
12     : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
13 {
14     setGrossSales( sales ); // validate and store gross sales
15     setCommissionRate( rate ); // validate and store commission rate
16 } // end CommissionEmployee constructor
17
19 void CommissionEmployee::setFirstName( const string &first )
20 {
21     firstName = first; // should validate
22 } // end function setFirstName
23
25 string CommissionEmployee::getFirstName() const
26 {
27     return firstName;
28 } // end function getFirstName
```




```

31 void CommissionEmployee::setLastName( const string &last )
32 {
33     lastName = last; // should validate
34 } // end function setLastName
35
37 string CommissionEmployee::getLastName() const
38 {
39     return lastName;
40 } // end function getLastName
41 // =====
43 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
44 {
45     socialSecurityNumber = ssn; // should validate
46 } // end function setSocialSecurityNumber
47
49 string CommissionEmployee::getSocialSecurityNumber() const
50 {
51     return socialSecurityNumber;
52 } // end function getSocialSecurityNumber
53 // =====
55 void CommissionEmployee::setGrossSales( double sales )
56 {
57     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
58 } // end function setGrossSales

```



```

61 double CommissionEmployee::getGrossSales() const
62 {
63     return grossSales;
64 } // end function getGrossSales
65 //=====
67 void CommissionEmployee::setCommissionRate( double rate )
68 {
69     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
70 } // end function setCommissionRate
71
73 double CommissionEmployee::getCommissionRate() const
74 {
75     return commissionRate;
76 } // end function getCommissionRate
77 //=====
79 double CommissionEmployee::earnings() const
80 {
81     return getCommissionRate() * getGrossSales();
82 } // end function earnings

```

不是直接通过变量计算，而是通过public类型的函数调用，取其返回值进行计算

Use *get* functions to obtain the values of data members



```
84 // print CommissionEmployee object
85 void CommissionEmployee::print() const
86 {
87     cout << "commission employee: "
88         << getFirstName() << ' ' << getLastName()
89         << "\nsocial security number: " << getSocialSecurityNumber()
90         << "\ngross sales: " << getGrossSales()
91         << "\ncommission rate: " << getCommissionRate();
92 } // end function print
```



```
1  // Fig. 12.19: BasePlusCommissionEmployee.h
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15     BasePlusCommissionEmployee( const string &, const string &,
16         const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18     void setBaseSalary( double ); // set base salary
19     double getBaseSalary() const; // return base salary
20
21     double earnings() const; // calculate earnings
22     void print() const; // print BasePlusCommissionEmployee object
23 private:
24     double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
```

在 class **CommissionEmployee** 中已经定义:

```
30 double earnings() const; // calculate earnings
31 void print() const; // print CommissionEmployee object
```



```
1  // Fig. 12.20: BasePlusCommissionEmployee.cpp
10 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
11     const string &first, const string &last, const string &ssn,
12     double sales, double rate, double salary )
14     : CommissionEmployee( first, last, ssn, sales, rate )
15 {
16     setBaseSalary( salary ); // validate and store base salary
17 } // end BasePlusCommissionEmployee constructor
18
20 void BasePlusCommissionEmployee::setBaseSalary( double salary )
21 {
22     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23 } // end function setBaseSalary
24
26 double BasePlusCommissionEmployee::getBaseSalary() const
27 {
28     return baseSalary;
29 } // end function getBaseSalary
```



```

30
31 // calculate earnings
32 double BasePlusCommissionEmployee::earnings() const
33 {
34     return getBaseSalary() + CommissionEmployee::earnings();
35 } // end function earnings
36
37 // print BasePlusCommissionEmployee object
38 void BasePlusCommissionEmployee::print() const
39 {
40     cout << "base-salaried ";
41
42     // invoke CommissionEmployee's print function
43     CommissionEmployee::print();
44
45     cout << "\nbase salary: " << getBaseSalary();
46 } // end function print

```

Invoke base class' s earnings function。显式调用基类函数的 public 型函数，需写明基类的名字！

Invoke base class's print function

利用成员函数访问数据成员的值可能比直接访问这些数据稍微慢些，但是更符合软件工程规范。



在派生类中重定义基类成员

- 派生类可以通过提供同样签名(参数类型与参数个数)的新版本(如果签名不同,则是函数重载而不是函数重定义)重新定义基类成员函数。派生类引用该函数时会自动选择派生类中的版本。使用作用域运算符(::)可用来从派生类中访问基类的该成员函数的版本。
 - 派生类中重新定义基类的成员函数时,为完成某些附加工作,派生类版本通常要调用基类中的该函数版本。不使用作用域运算符(::)会由于派生类成员函数实际上调用了自身而引起无穷递归。这样会使系统用光内存,是致命的运行时错误。



```
1 // Fig. 12.21: fig12_21.cpp
2 // Testing class BasePlusCommissionEmployee.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6 using std::fixed;
7
8 #include <iomanip>
9 using std::setprecision;
10
11 // BasePlusCommissionEmployee class definition
12 #include "BasePlusCommissionEmployee.h"
13
```




```

14 int main()
15 {
16     // instantiate BasePlusCommissionEmployee object
17     BasePlusCommissionEmployee
18         employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
19
20     // set floating-point output formatting
21     cout << fixed << setprecision( 2 );
22
23     // get commission employee data
24     cout << "Employee information obtained by get functions: \n"
25         << "\nFirst name is " << employee.getFirstName()
26         << "\nLast name is " << employee.getLastName()
27         << "\nSocial security number is "
28         << employee.getSocialSecurityNumber()
29         << "\nGross sales is " << employee.getGrossSales()
30         << "\nCommission rate is " << employee.getCommissionRate()
31         << "\nBase salary is " << employee.getBaseSalary() << endl;
32
33     employee.setBaseSalary( 1000 ); // set base salary
34
35     cout << "\nUpdated employee information output by print function: \n"
36         << endl;
37     employee.print(); // display the new employee information
38
39     // display the employee's earnings
40     cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
41
42     return 0;
43 } // end main

```



Employee information obtained by get functions:

First name is Bob

Last name is Lewis

Social security number is 333-33-3333

Gross sales is 5000.00

Commission rate is 0.04

Base salary is 300.00

Updated employee information output by print function:

base-salaried commission employee: Bob Lewis

social security number: 333-33-3333

gross sales: 5000.00

commission rate: 0.04

base salary: 1000.00

Employee's earnings: \$1200.00



9 Using private Data



常见编程错误：如果在派生类中包含一个与基类中同名的但是有不同签名的成员函数，那么这个函数会**隐藏基类的同名函数**，如果通过派生类的对象的 public 接口试图调用基类的这个成员函数，将会产生编译错误。



常见编程错误：当派生类重新定义了基类的成员函数时，如果派生类需要调用基类的同名函数来做额外的工作，就需要在基类的同名成员函数在前面加上基类名称和二元作用域运算符(`::`)，否则将导致无限递归的错误。

10 Constructors and Destructors in Derived Classes

- 派生类的构造函数可以显式调用（利用了基类成员初始化器）或隐式（调用基类的默认构造函数）调用其直接基类的构造函数。
 - 如果该基类也是派生出来的，则该基类的构造函数需要调用它的基类的构造函数。
 - 在这个构造函数调用链中调用的**最后一个构造函数**是在继承层次中**最顶层的构造函数**，但是事实上它的函数体是最先执行完毕的，**最早调用的派生类构造函数最晚完成其函数体的执行**。
- 构造函数的级联调用
 - 初始化数据成员
 - ✓ **每个基类初始化自身的数据成员**



10 Constructors and Destructors in Derived Classes

- 析构函数的调用顺序和调用构造函数的顺序相反，因此派生类的析构函数在基类析构函数之前调用。
 - 当调用派生类对象的析构函数时，该析构函数执行其任务，然后调用继承层次中上一层基类的析构函数，这一过程重复进行，直到顶层的最后一个基类的析构函数被调用。之后，该对象从内存中删除。
- 销毁派生类对象
 - 析构函数的级联调用
 - ✓ 与构造函数调用的顺序相反
 - ✓ 派生类的析构函数先执行
 - ✓ 然后是上一级的析构函数
 - ◇ 直到基类的析构函数被调用



10 Constructors and Destructors in Derived Classes

- 假设生成派生类对象，基类和派生类都包含其他类的对象，则在建立派生类的对象时，
 - 首先执行基类成员对象的构造函数
 - 接着执行基类的构造函数
 - 以后执行派生类的成员对象的构造函数
 - 最后才执行派生类的构造函数。
- 析构函数的调用次序与调用构造函数的次序相反。
- 派生类不继承基类的构造函数、析构函数和重载的赋值运算符，但是派生类的构造函数、析构函数和重载的赋值运算符能调用基类的构造函数、析构函数和重载的赋值运算符。



10 Constructors and Destructors in Derived Classes



软件工程知识：假设创建一个派生类对象，其基类和派生类都包含其他类的对象，创建该派生类的对象后，首先执行基类成员对象的构造函数，然后执行基类构造函数，再执行派生类成员对象的构造函数，最后执行派生类的构造函数。析构函数的调用顺序与相应的构造函数的调用顺序相反。



```

1 // Fig. 12.22: CommissionEmployee.h
2 // CommissionEmployee class definition represents a commission employee.
3 #ifndef COMMISSION_H
4 #define COMMISSION_H
5
6 #include <string> // C++ standard string class
7 using std::string;
8
9 class CommissionEmployee
10 {
11 public:
12     CommissionEmployee( const string &, const string &, const string &,
13         double = 0.0, double = 0.0 );
14     ~CommissionEmployee(); // destructor
15
16     void setFirstName( const string & ); // set first name
17     string getFirstName() const; // return first name
18
19     void setLastName( const string & ); // set last name
20     string getLastName() const; // return last name
21
22     void setSocialSecurityNumber( const string & ); // set SSN
23     string getSocialSecurityNumber() const; // return SSN
24
25     void setGrossSales( double ); // set gross sales amount
26     double getGrossSales() const; // return gross sales amount
27
28     void setCommissionRate( double ); // set commission rate
29     double getCommissionRate() const; // return commission rate

```




```
30
31     double earnings() const; // calculate earnings
32     void print() const; // print CommissionEmployee object
33 private:
34     string firstName;
35     string lastName;
36     string socialSecurityNumber;
37     double grossSales; // gross weekly sales
38     double commissionRate; // commission percentage
39 }; // end class CommissionEmployee
40
41 #endif
```



```

1 // Fig. 12.23: CommissionEmployee.cpp
2 // Class CommissionEmployee member-function definitions.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "CommissionEmployee.h" // CommissionEmployee class definition
8
9 // constructor
10 CommissionEmployee::CommissionEmployee(
11     const string &first, const string &last, const string &ssn,
12     double sales, double rate )
13     : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
14 {
15     setGrossSales( sales ); // validate and store gross sales
16     setCommissionRate( rate ); // validate and store commission rate
17
18     cout << "CommissionEmployee constructor: " << endl;
19     print(); // 打印当前对象的数据成员值
20     cout << "\n\n";
21 } // end CommissionEmployee constructor
22
23 // destructor
24 CommissionEmployee::~CommissionEmployee()
25 {
26     cout << "CommissionEmployee destructor: " << endl;
27     print();
28     cout << "\n\n";
29 } // end CommissionEmployee destructor

```



```
30
31 // set first name
32 void CommissionEmployee::setFirstName( const string &first )
33 {
34     firstName = first; // should validate
35 } // end function setFirstName
36
37 // return first name
38 string CommissionEmployee::getFirstName() const
39 {
40     return firstName;
41 } // end function getFirstName
42
43 // set last name
44 void CommissionEmployee::setLastName( const string &last )
45 {
46     lastName = last; // should validate
47 } // end function setLastName
48
49 // return last name
50 string CommissionEmployee::getLastName() const
51 {
52     return lastName;
53 } // end function getLastName
54
55 // set social security number
56 void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
57 {
58     socialSecurityNumber = ssn; // should validate
59 } // end function setSocialSecurityNumber
```



```
60 // return social security number
61 string CommissionEmployee::getSocialSecurityNumber() const
62 {
63     return socialSecurityNumber;
64 } // end function getSocialSecurityNumber
65
66 // set gross sales amount
67 void CommissionEmployee::setGrossSales( double sales )
68 {
69     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
70 } // end function setGrossSales
71
72 // return gross sales amount
73 double CommissionEmployee::getGrossSales() const
74 {
75     return grossSales;
76 } // end function getGrossSales
77
78 // set commission rate
79 void CommissionEmployee::setCommissionRate( double rate )
80 {
81     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
82 } // end function setCommissionRate
83
84 // return commission rate
85 double CommissionEmployee::getCommissionRate() const
86 {
87     return commissionRate;
88 } // end function getCommissionRate
```



```
90
91 // calculate earnings
92 double CommissionEmployee::earnings() const
93 {
94     return getCommissionRate() * getGrossSales();
95 } // end function earnings
96
97 // print CommissionEmployee object
98 void CommissionEmployee::print() const
99 {
100     cout << "commission employee: "
101         << getFirstName() << ' ' << getLastName()
102         << "\nsocial security number: " << getSocialSecurityNumber()
103         << "\ngross sales: " << getGrossSales()
104         << "\ncommission rate: " << getCommissionRate();
105} // end function print
```



```

1 // Fig. 12.24: BasePlusCommissionEmployee.h
2 // BasePlusCommissionEmployee class derived from class
3 // CommissionEmployee.
4 #ifndef BASEPLUS_H
5 #define BASEPLUS_H
6
7 #include <string> // C++ standard string class
8 using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15     BasePlusCommissionEmployee( const string &, const string &,
16         const string &, double = 0.0, double = 0.0, double = 0.0 );
17     ~BasePlusCommissionEmployee(); // destructor
18
19     void setBaseSalary( double ); // set base salary
20     double getBaseSalary() const; // return base salary
21
22     double earnings() const; // calculate earnings
23     void print() const; // print BasePlusCommissionEmployee object
24 private:
25     double baseSalary; // base salary
26 }; // end class BasePlusCommissionEmployee
27
28 #endif

```



```

1 // Fig. 12.25: BasePlusCommissionEmployee.cpp
2 // Class BasePlusCommissionEmployee member-function definitions.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 // BasePlusCommissionEmployee class definition
8 #include "BasePlusCommissionEmployee.h"
9
10 // constructor
11 BasePlusCommissionEmployee::BasePlusCommissionEmployee(
12     const string &first, const string &last, const string &ssn,
13     double sales, double rate, double salary )
14     // explicitly call base-class constructor
15     : CommissionEmployee( first, last, ssn, sales, rate )
16 {
17     setBaseSalary( salary ); // validate and store base salary
18
19     cout << "BasePlusCommissionEmployee constructor: " << endl;
20     print();
21     cout << "\n\n";
22 } // end BasePlusCommissionEmployee constructor
23
24 // destructor
25 BasePlusCommissionEmployee::~BasePlusCommissionEmployee()
26 {
27     cout << "BasePlusCommissionEmployee destructor: " << endl;
28     print();
29     cout << "\n\n";
30 } // end BasePlusCommissionEmployee destructor

```



```

31
32 // set base salary
33 void BasePlusCommissionEmployee::setBaseSalary( double salary )
34 {
35     baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
36 } // end function setBaseSalary
37
38 // return base salary
39 double BasePlusCommissionEmployee::getBaseSalary() const
40 {
41     return baseSalary;
42 } // end function getBaseSalary
43
44 // calculate earnings
45 double BasePlusCommissionEmployee::earnings() const
46 {
47     return getBaseSalary() + CommissionEmployee::earnings();
48 } // end function earnings
49
50 // print BasePlusCommissionEmployee object
51 void BasePlusCommissionEmployee::print() const
52 {
53     cout << "base-salaried ";
54
55     // invoke CommissionEmployee's print function
56     CommissionEmployee::print();
57
58     cout << "\nbase salary: " << getBaseSalary();
59 } // end function print

```



C++ How to Program

```
1 // Fig. 12.26: fig12_26.cpp
2 // Display order in which base-class and derived-class constructors
3 // and destructors are called.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7 using std::fixed;
8
9 #include <iomanip>
10 using std::setprecision;
11
12 // BasePlusCommissionEmployee class definition
13 #include "BasePlusCommissionEmployee.h"
```



```
14
15 int main()
16 {
17     // set floating-point output formatting
18     cout << fixed << setprecision( 2 );
19
20     { // begin new scope    基类实例化成一个对象，需要注意的是该处加上括号与不加执行结果不同
21         CommissionEmployee employee1(
22             "Bob", "Lewis", "333-33-3333", 5000, .04 );
23     } // end scope
24     cout << endl;
25     // 派生类实例化成一个对象
26     BasePlusCommissionEmployee
27         employee2( "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );
28     cout << endl;
29     // 派生类再次实例化成一个对象
30     BasePlusCommissionEmployee
31         employee3( "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
32     cout << endl;
33     return 0;
34 } // end main
```



CommissionEmployee constructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04

CommissionEmployee destructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04

CommissionEmployee constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06

BasePlusCommissionEmployee constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 800.00

CommissionEmployee constructor:
commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15

```
20  { // begin new scope    基类实例化成一个对象
21      CommissionEmployee employee1(
22          "Bob", "Lewis", "333-33-3333", 5000, .04 );
23  } // end scope
```

```
26  BasePlusCommissionEmployee
27      employee2( "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );
```

```
30  BasePlusCommissionEmployee
31      employee3( "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
```

(continued at top of next slide...)



(... continued from bottom of previous slide)

BasePlusCommissionEmployee constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00

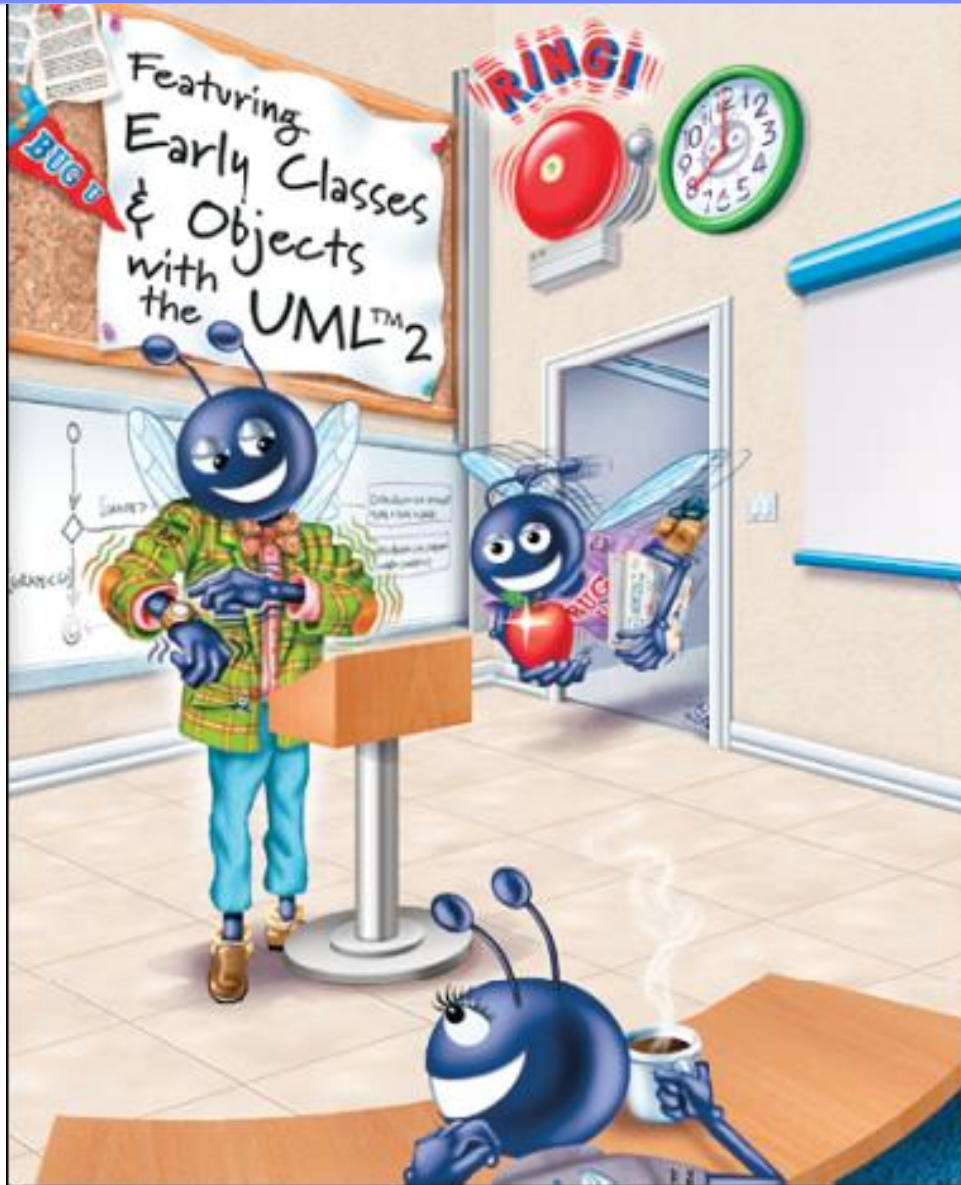
BasePlusCommissionEmployee destructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00

CommissionEmployee destructor:
commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15

BasePlusCommissionEmployee destructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 800.00

CommissionEmployee destructor:
commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06

C++ How to Program



Thank you!