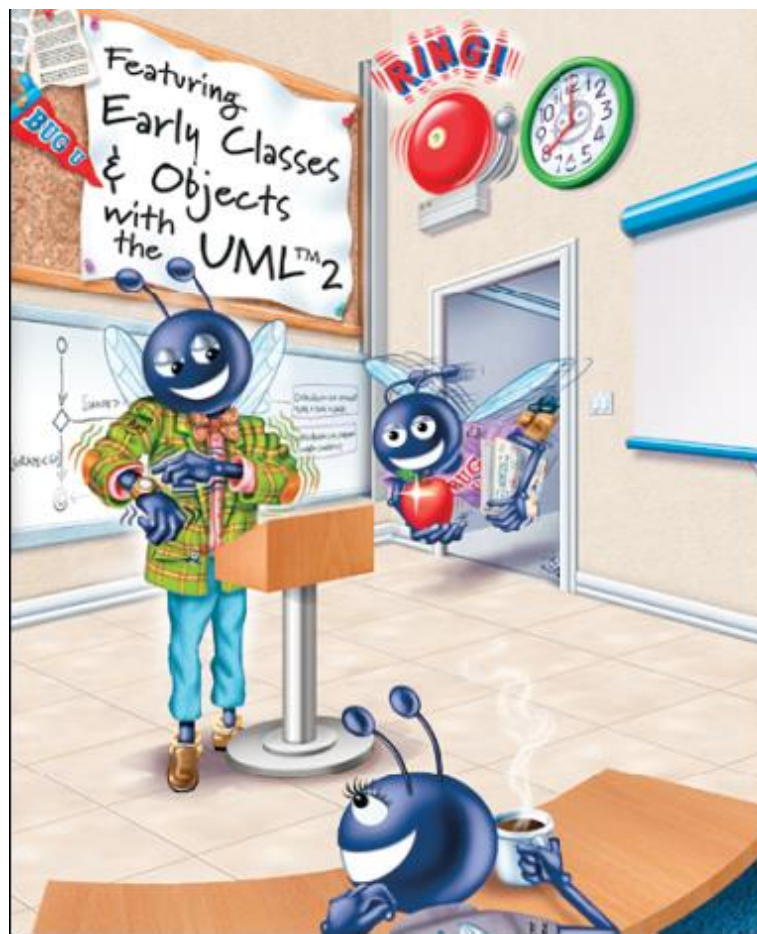


# C++程序设计



## 上节课内容回顾

1. 什么是运算符重载
2. 如何进行运算符重载
3. 类型转换
4. 重载 ++ 和 -- 运算符
5. 实例：Array类；String类；Date类

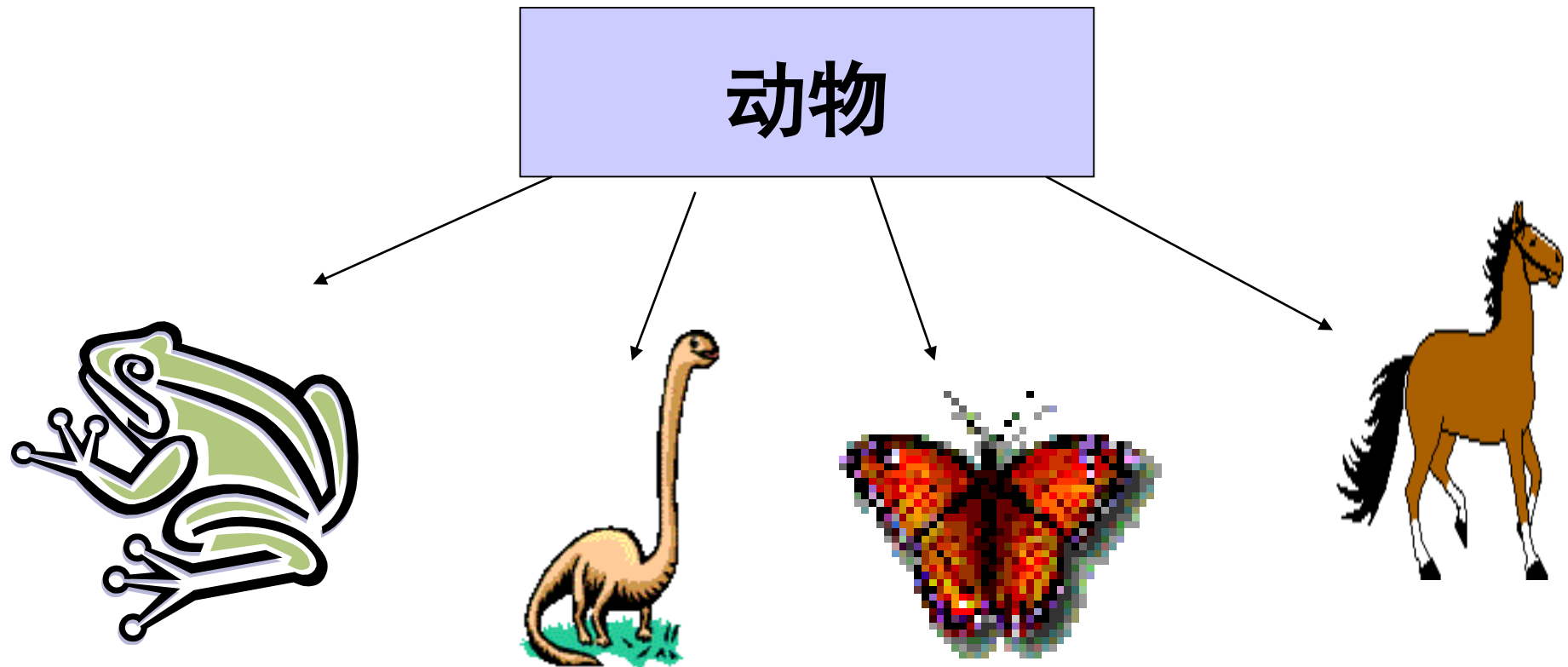
# 第十一讲 面向对象编程：继承

## 学习目标：

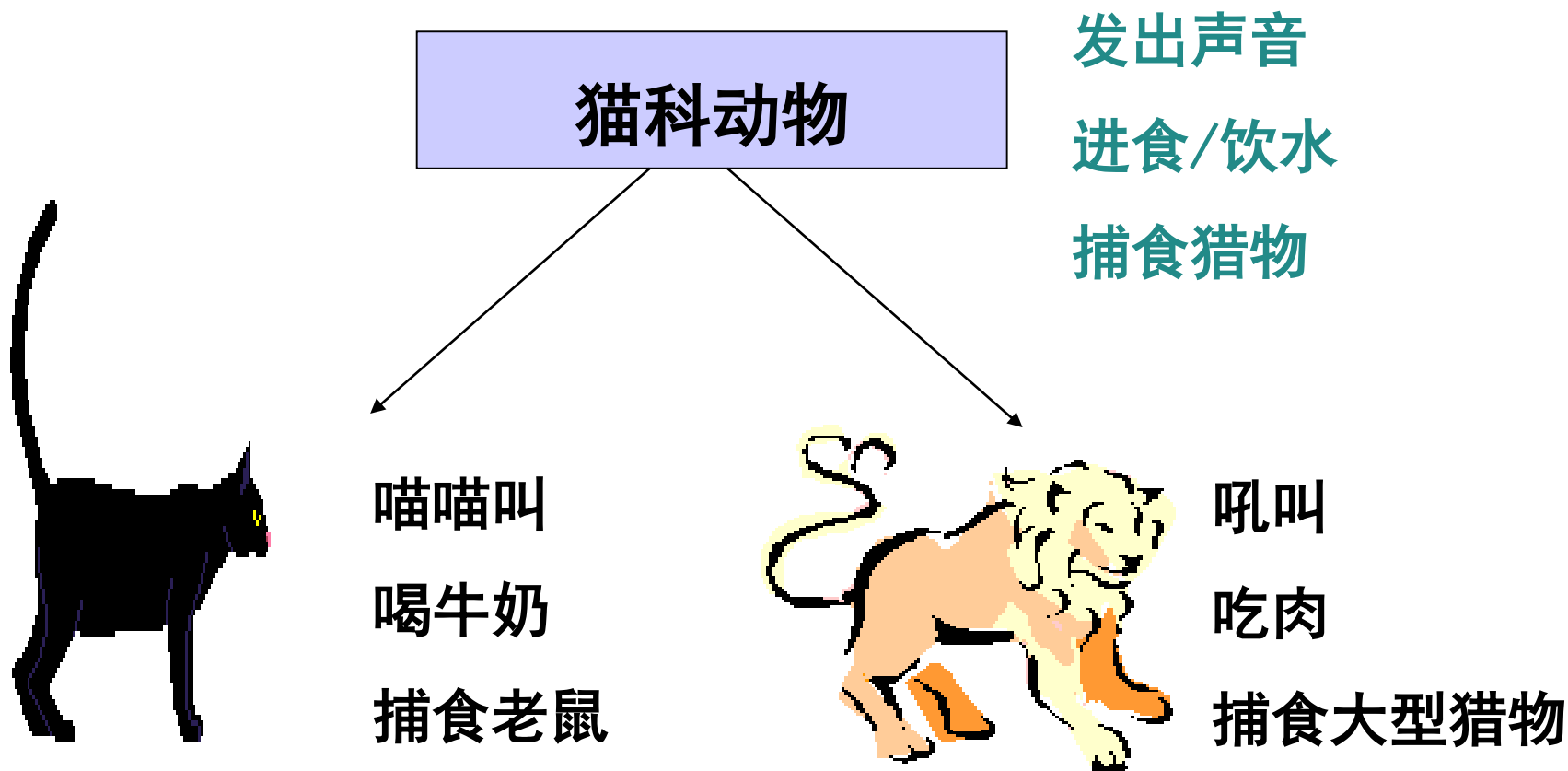
- 通过继承现有类来创建新类
- 基类和派生类之间的关系
- protected 成员的访问
- 继承层次中构造函数和析构函数的调用



# 1. Introduction



# 1. Introduction



# 1. Introduction

## ● 继承

- 软件重用
- 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

### ➤ public

◇ 每个派生类对象也是基类的对象

◇ 基类对象不是派生类对象

◇ 例如：所有汽车都是交通工具，但不是所有交通工具都是汽车

# 1. Introduction

## ● 三种类型的继承

### ➤ public

◇ 派生类可以访问基类的非私有成员

◇ 要访问基类的私有数据成员

◇ 派生类必须使用从基类继承的非私有成员函数

# 1. Introduction

- 三种类型的继承

- private

- ◆ 另一种组合

- protected

- ◆ 很少使用

# 1. Introduction

- “is-a” vs. “has-a”

- “is-a”

- ◇ 继承

- ◇ 派生类对象可以看作是基类对象

- ◇ 例如：汽车是交通工具

- ◇ 交通工具的属性和行为可以应用于汽车

# 1. Introduction

- “is-a” vs. “has-a”

- “has-a”

- ◇ 组合

- ◇ 对象包含一个或多个其他类的对象作为成员

- ◇ 例如：汽车有方向盘

# 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。



**软件工程知识：** 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

- 一个类的对象是另一个类的对象

- ◇ 例如：矩形是四边形

- ◇ 矩形类继承自四边形

- ◇ 四边形是基类

- ◇ 矩形是派生类

## 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

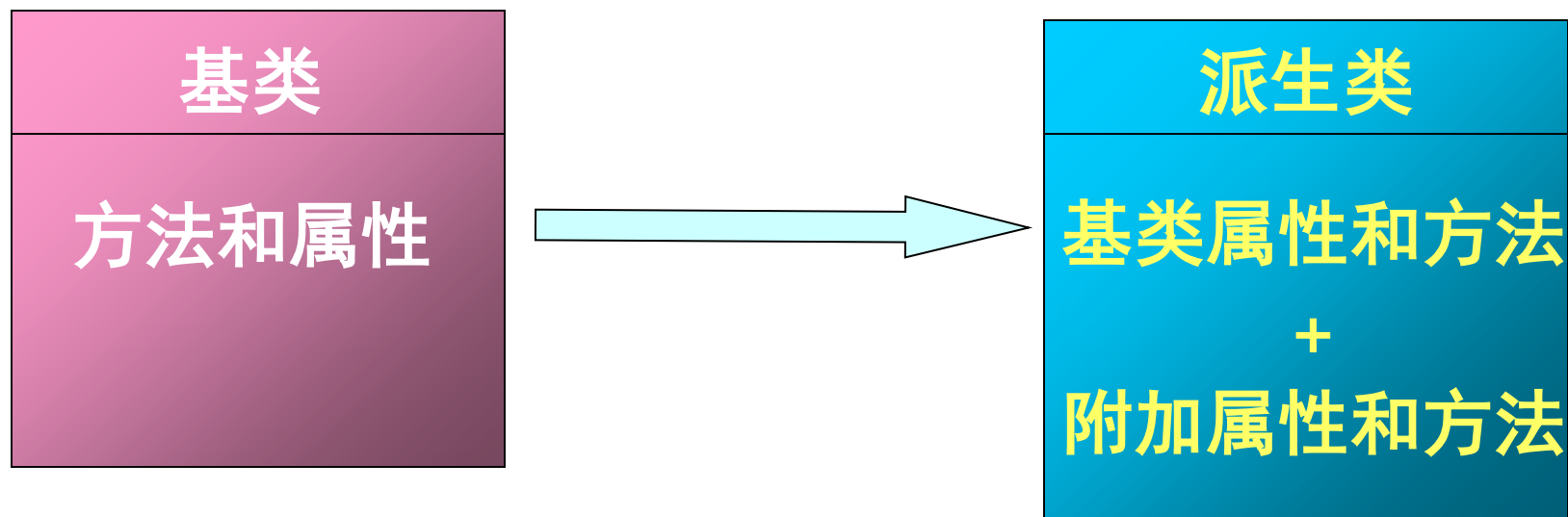
◇ 派生类：汽车

◇ 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

## 2. Base Classes and Derived Classes

### ● 继承层次

- 继承关系：树型层次结构
- 每个类成为
  - ◇ 基类：为其他类提供数据/行为  
或
  - ◇ 派生类：从其他类继承数据/行为

## 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

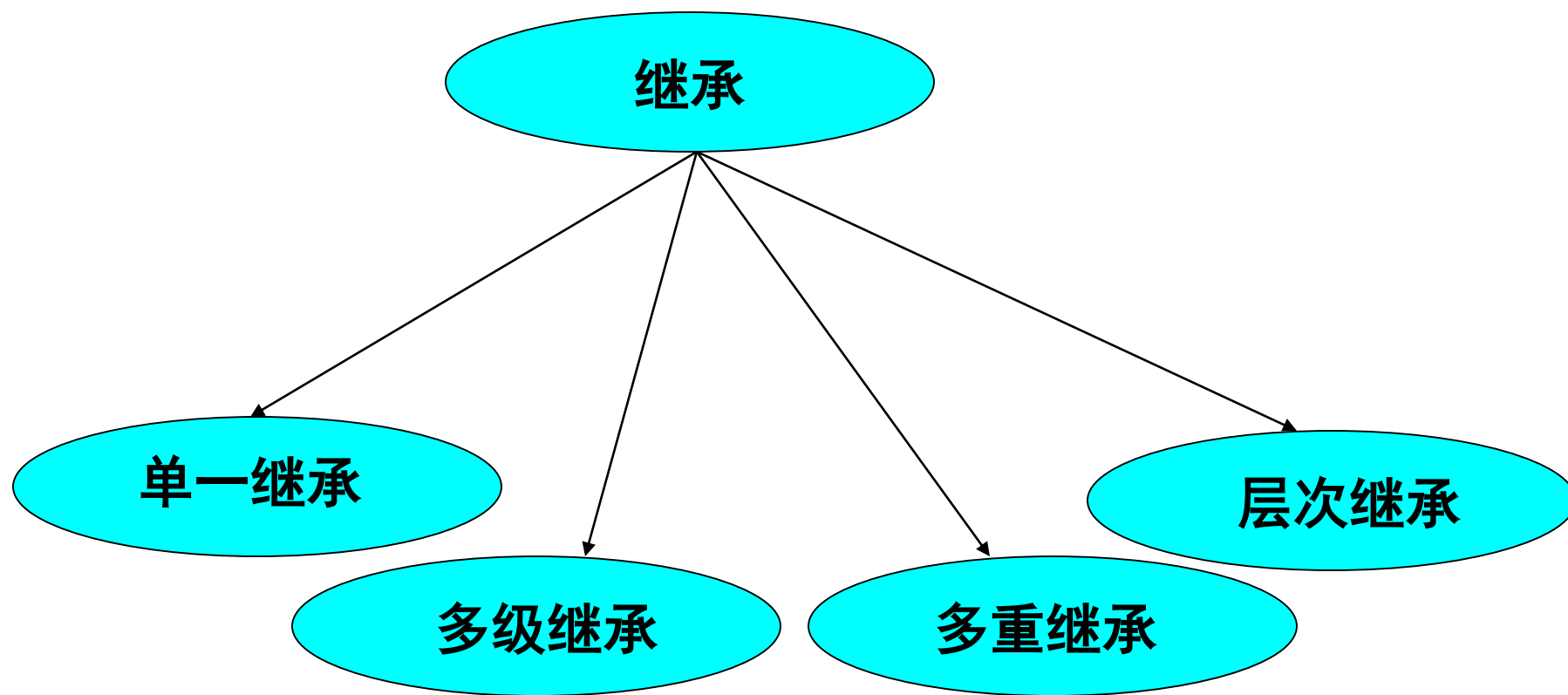
### 直接基类和间接基类

- 间接基类

```
class A
{ };
class B : public A
{ };
class C : public B    //A是C的间接基类
{ };
```

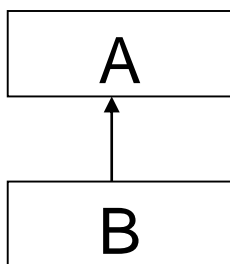
## 2. Base Classes and Derived Classes

### 继承的类型



## 2. Base Classes and Derived Classes

### 单一继承

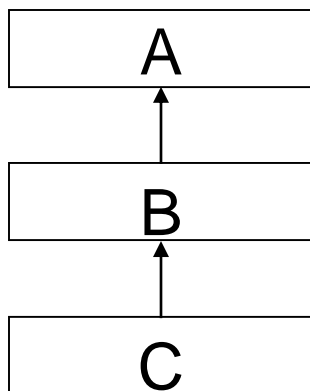


```
class A  
{...};  
class B : public A  
{...};
```



## 2. Base Classes and Derived Classes

### 多级继承

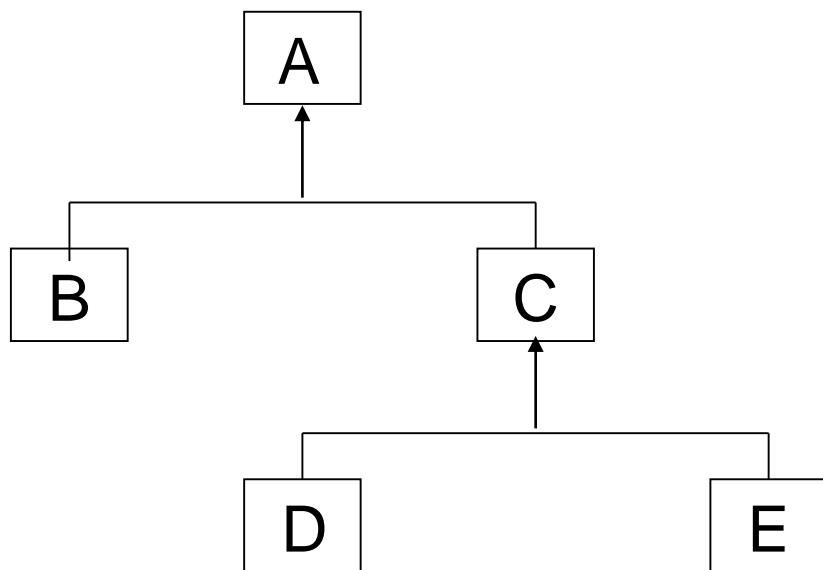


```
class A
{...};
class B : public A
{...};
class C : public B
{...};
```



## 2. Base Classes and Derived Classes

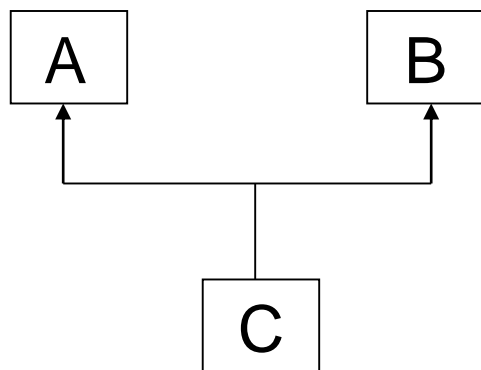
### 层次继承



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

## 2. Base Classes and Derived Classes

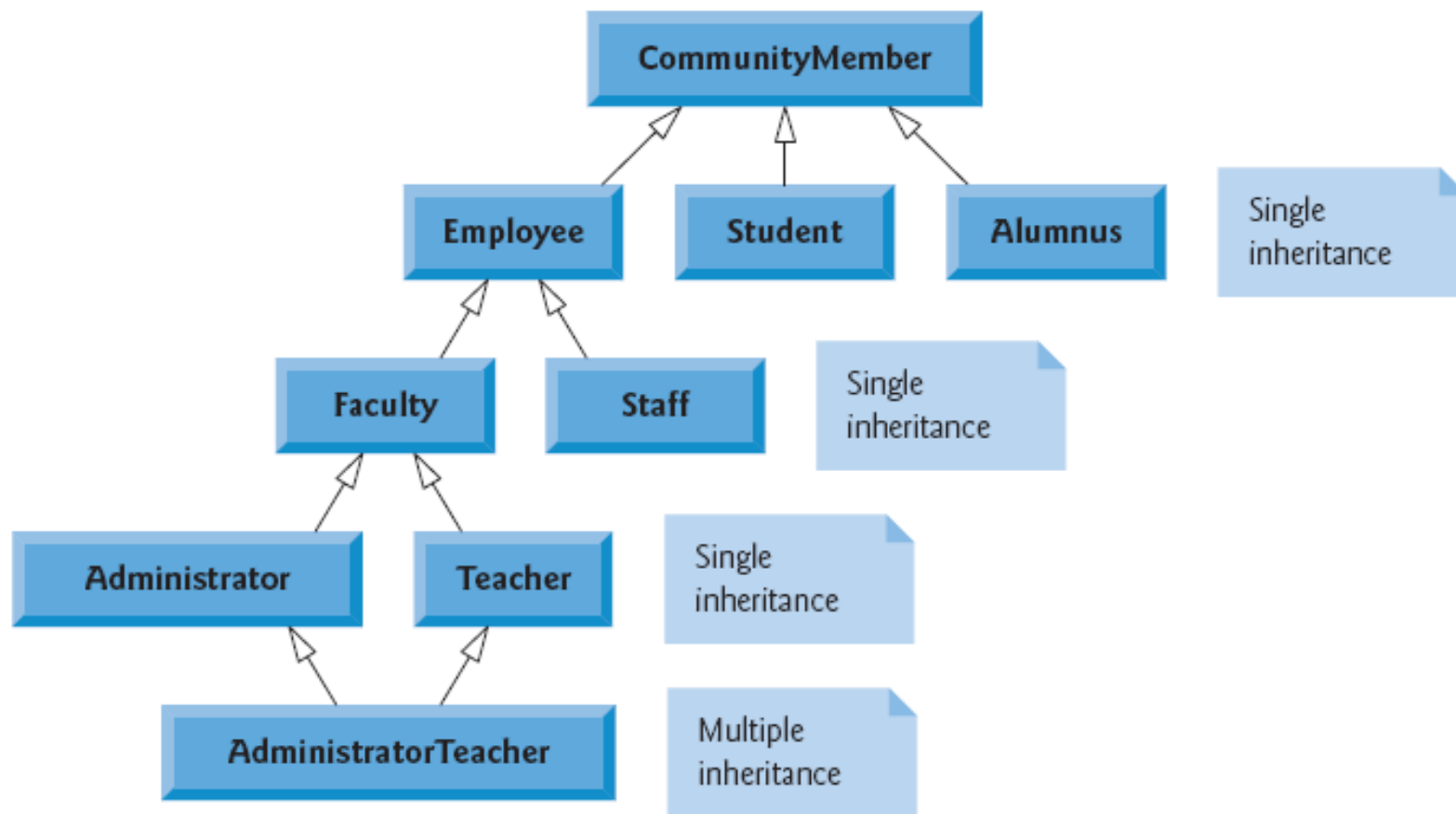
### 多重继承



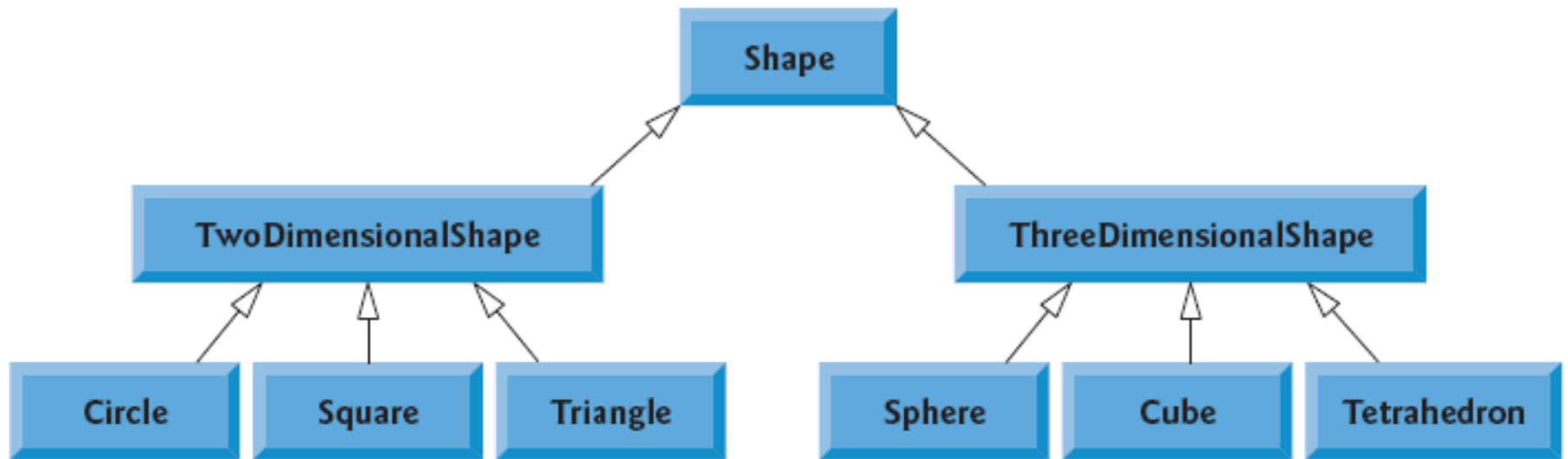
```
class A  
{...};  
class B  
{...};  
class C : public A, public B  
{...};
```



## 2. Base Classes and Derived Classes



## 2. Base Classes and Derived Classes



## 2. Base Classes and Derived Classes

### ● public 继承

- 例如：Class TwoDimensionalShape : public Shape
- 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

## 3. protected Members

- **protected access**

- 介于 public 和 private 之间

- protected 的访问

- ◇ 基类成员

- ◇ 基类友元

- ◇ 派生类成员

- ◇ 派生类友元

## 3. protected Members

### ● 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

## 3. protected Members

访问控制说明符	从自身类访问	从派生类访问	从类外访问
public	可以	可以	可以
protected	可以	可以	不可以
private	可以	不可以	不可以



## 4. Relationship between Base Classes and Derived Classes

- 基类和派生类之间的关系

- 例如：

- CommissionEmployee/BasePlusCommissionEmployee

- ◆ CommissionEmployee

- ◆ First name, last name, SSN, commission rate,  
gross sale amount

## 4. Relationship between Base Classes and Derived Classes

- 基类和派生类之间的关系

- 例如：

- CommissionEmployee/BasePlusCommissionEmployee

- ◆ BasePlusCommissionEmployee

- ◆ First name, last name, SSN, commission rate,  
gross sale amount

- ◆ And also: base salary

## 5. Creating and Using a CommissionEmployee Class

```
class CommissionEmployee
```

```
{
```

```
public:
```

```
    CommissionEmployee( const string &, const string &, const string &,  
                        double = 0.0, double = 0.0 );
```

Class CommissionEmployee constructor

```
.....
```

```
    double earnings() const; // calculate earnings
```

```
    void print() const; // print CommissionEmployee object
```

```
private:
```

```
    string firstName;
```

```
    string lastName;
```

```
    string socialSecurityNumber;
```

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

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

```
}; // end class CommissionEmployee
```

Declare private data members

// constructor

```
CommissionEmployee::CommissionEmployee(
```

```
    const string &first, const string &last, const string &ssn,  
    double sales, double rate )
```

```
{
```

```
    firstName = first; // should validate
```

```
    lastName = last; // should validate
```

```
    socialSecurityNumber = ssn; // should validate
```

```
    setGrossSales( sales ); // validate and store gross sales
```

```
    setCommissionRate( rate ); // validate and store commission rate
```

```
} // end CommissionEmployee constructor
```

Initialize data members



// calculate earnings

**double** CommissionEmployee::earnings() **const**

{

Function earnings calculates earnings



**return** commissionRate \* grossSales;

} // end function earnings

// print CommissionEmployee object

Function print displays  
CommissionEmployee object



**void** CommissionEmployee::print() **const**

{

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

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

<< "\ngross sales: " << grossSales

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

} // end function print

```
int main()
```

```
{
```

```
// instantiate a CommissionEmployee object
```

```
CommissionEmployee employee(  
    "Sue", "Jones", "222-22-2222", 10000, .06 );
```

Instantiate CommissionEmployee object

```
// set floating-point output formatting
```

```
cout << fixed << setprecision( 2 );
```

```
// get commission employee data
```

```
cout << "Employee information obtained by get functions: \n"
```

```
<< "\nFirst name is " << employee.getFirstName()
```

```
<< "\nLast name is " << employee.getLastName()
```

```
<< "\nSocial security number is "
```

```
<< employee.getSocialSecurityNumber()
```

```
<< "\nGross sales is " << employee.getGrossSales()
```

```
<< "\nCommission rate is " << employee.getCommissionRate()
```

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

```
employee.setGrossSales( 8000 ); // set gross sales
```

```
employee.setCommissionRate( 1 ); // set commission rate
```

```
cout << "\nUpdated empl
```

```
<< endl;
```

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

```
// display the employee's earnings
```

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

```
return 0;
```

```
} // end main
```

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

Call object's print function to display employee information

Call object's earnings function to calculate earnings

## 6. Creating a BasePlusCommissionEmployee Class Without Using Inheritance

### ● Class BasePlusCommissionEmployee

➤ 大部分代码与 CommissionEmployee 相同

◆ private 数据成员

◆ public 成员函数

◆ 构造函数



## 6. Creating a BasePlusCommissionEmployee Class Without Using Inheritance

- **Class BasePlusCommissionEmployee**

- 额外的

- ◆ private 数据成员 baseSalary

- ◆ setBaseSalary 和 getBaseSalary 成员函数

```
class BasePlusCommissionEmployee
```

```
{
```

```
public:
```

```
BasePlusCommissionEmployee( const string &, const string &,  
    const string &, double = 0.0, double = 0.0, double = 0.0 );
```

```
.....
```

```
void setBaseSalary( double ); // set base salary
```

```
double getBaseSalary() const; // return base salary
```

```
double earnings() const; // calculate earnings
```

```
void print() const; // print BasePlusCommissionEmployee object
```

```
private:
```

```
string firstName;
```

```
string lastName;
```

```
string socialSecurityNumber;
```

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

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

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

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

```
BasePlusCommissionEmployee::BasePlusCommissionEmployee(  
    const string &first, const string &last, const string &ssn,  
    double sales, double rate, double salary )  
{  
    firstName = first; // should validate  
    lastName = last; // should validate  
    socialSecurityNumber = ssn; // should validate  
    setGrossSales( sales ); // validate and store gross sales  
    setCommissionRate( rate ); // validate and store commission rate  
    setBaseSalary( salary ); // validate and store base salary  
} // end BasePlusCommissionEmployee constructor
```

// set base salary

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

```
{
```

```
    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
```

```
} // end function setBaseSalary
```

// return base salary

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

```
{
```

```
    return baseSalary;
```

```
} // end function getBaseSalary
```

// calculate earnings

```
double BasePlusCommissionEmployee::earnings() const
```

```
{
```

```
    return baseSalary + ( commissionRate * grossSales );
```

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

**// print BasePlusCommissionEmployee object**

**void BasePlusCommissionEmployee::print() const**

**{**

```
cout << "base-salaried commission employee: " << firstName << ' '
    << lastName << "\nsocial security number: " << socialSecurityNumber
    << "\ngross sales: " << grossSales
    << "\ncommission rate: " << commissionRate
    << "\nbase salary: " << baseSalary;
```

**} // end function print**

## 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 a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy

- 文件结构
- 构造函数
- 基类数据成员访问
- 调用基类构造函数
- protected访问修饰符
- 良好的软件工程原则
- 派生类调用基类同名函数
- 派生类覆盖基类同名函数



## 7. Creating a CommissionEmployee-BasePlusCommissionEmployee Inheritance Hierarchy

### ● Class 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
- 具有一个数据成员 baseSalary

```
#include "CommissionEmployee.h"
```

```
class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
    BasePlusCommissionEmployee( const string &, const string &,
        const string &, double = 0.0, double = 0.0, double = 0.0 );

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

    double earnings() const; // calculate earnings
    void print() const; // print BasePlusCommissionEmployee object
private:
    double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee
```

**// constructor**

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

**// explicitly call base-class constructor**

```
: CommissionEmployee( first, last, ssn, sales, rate )
```

```
{
```

```
    setBaseSalary( salary ); // validate and store base salary
```

```
} // end BasePlusCommissionEmployee constructor
```

```
// calculate earnings
```


```
double BasePlusCommissionEmployee::earnings() const
```

```
{
```

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

```
return baseSalary + ( commissionRate * grossSales );
```

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



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

**// print BasePlusCommissionEmployee object**

**void BasePlusCommissionEmployee::print() const**

**{**

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

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

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

**<< "\ngross sales: " << grossSales**

**<< "\ncommission rate: " << commissionRate**

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

**} // end function**

Compiler generates errors because the base class's data members `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` and `commissionRate` are private

## 8. Using protected Data

- 使用 protected 数据成员

- 使得 class BasePlusCommissionEmployee 可以直接访问基类数据成员

```
class CommissionEmployee
```

```
{
```

```
public:
```

```
    CommissionEmployee( const string &, const string &, const string &,  
                        double = 0.0, double = 0.0 );
```

```
    .....
```

```
    double earnings() const; // calculate earnings
```

```
    void print() const; // print CommissionEmployee object
```

```
protected:
```

```
    string firstName;
```

```
    string lastName;
```

```
    string socialSecurityNumber;
```

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

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

```
}; // end class CommissionEmployee
```



Declare protected data

```
class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
    BasePlusCommissionEmployee( const string &, const string &,
        const string &, double = 0.0, double = 0.0, double = 0.0 );

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

    double earnings() const; // calculate earnings
    void print() const; // print BasePlusCommissionEmployee object
private:
    double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee
```



**// constructor**

```
BasePlusCommissionEmployee::BasePlusCommissionEmployee(  
    const string &first, const string &last, const string &ssn,  
    double sales, double rate, double salary )  
    // explicitly call base-class constructor  
    : CommissionEmployee( first, last, ssn, sales, rate )  
{  
    setBaseSalary( salary ); // validate and store base salary  
} // end BasePlusCommissionEmployee constructor
```

// calculate earnings

**double** BasePlusCommissionEmployee::earnings() **const**

{

// can access protected data of base class

**return** baseSalary + ( commissionRate \* grossSales );

} // end function earnings

// print BasePlusCommissionEmployee object

**void** BasePlusCommissionEmployee::print() **const**

{

// can access protected data of base class

cout << "base-salaried commission employee: " << firstName << ' '  
<< lastName << "\nsocial security number: " << socialSecurityNumber  
<< "\ngross sales: " << grossSales  
<< "\ncommission rate: " << commissionRate  
<< "\nbase salary: " << baseSalary;

} // end function print



Directly access base  
class's protected data

## 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

- ◇ 避免了 *set/get* 成员函数的调用

## 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

- ◇ 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 get` 和 `set` 函数

- ◆ 使用 `get` 获得数据成员的值

```
class CommissionEmployee
{
public:
    CommissionEmployee( const string &, const string &, const string &,
        double = 0.0, double = 0.0 );

    .....

private:
    string firstName;
    string lastName;
    string socialSecurityNumber;
    double grossSales; // gross weekly sales
    double commissionRate; // commission percentage
}; // end class CommissionEmployee
```



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

Two arrows originate from the text box. One arrow points to the `getCommissionRate()` call in the `return` statement. The other arrow points to the `getGrossSales()` call in the same `return` statement.

```
double CommissionEmployee::earnings() const
{
    return getCommissionRate() * getGrossSales();
} // end function earnings
```

```
class BasePlusCommissionEmployee : public CommissionEmployee
```

```
{
```

```
public:
```

```
    BasePlusCommissionEmployee( const string &, const string &,
                                const string &, double = 0.0, double = 0.0, double = 0.0 );
```

```
    void setBaseSalary( double ); // set base salary
```

```
    double getBaseSalary() const; // return base salary
```

```
    double earnings() const; // calculate earnings
```

```
    void print() const; // print BasePlusCommissionEmployee object
```

```
private:
```

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

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

**// calculate earnings**

**double** BasePlusCommissionEmployee::earnings() **const**

{

**return** getBaseSalary() + CommissionEmployee::earnings();

**} // end function earnings**

**// print BasePlusCommissionEmployee object**

**void** BasePlusCommissionEmployee::print() **const**

{

cout << "base-salaried ";

**// invoke CommissionEmployee's print function**

CommissionEmployee::print();

cout << "\nbase salary: " << getBaseSalary();

**} // end function print**

## 9. Using private Data



**常见编程错误：**当派生类重新定义了基类的成员函数时，派生类常常需要调用基类的同名函数来做额外的工作，如果在调用基类的同名成员函数时忘了在前面加上基类名称和二元作用域运算符（::）将导致无限递归的错误。

## 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。

## 10. Constructors and Destructors in Derived Classes

- 实例化派生类

- 构造函数的级联调用

- ◇ 派生类的构造函数调用基类的构造函数

- ◇ 隐式的或显式的

## 10. Constructors and Destructors in Derived Classes

### ● 实例化派生类

#### ➤ 构造函数的级联调用

##### ◆ 基于继承层次

##### ◆ 例如：

CommissionEmployee/BasePlusCommissionEmployee  
hierarchy

◆ CommissionEmployee 构造函数最后被调用

◆ CommissionEmployee 构造函数最先被执行

## 10. Constructors and Destructors in Derived Classes

- 实例化派生类

- 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

### ● 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

- 基类构造函数，析构函数，重载的赋值运算符不能被派生类继承

## 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。

```
int main()
{
    // set floating-point output formatting
    cout << fixed << setprecision( 2 );
    { // begin new scope
        CommissionEmployee employee1(
            "Bob", "Lewis", "333-33-3333", 5000, .04 );
    } // end scope

    cout << endl;
    BasePlusCommissionEmployee
        employee2( "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );

    cout << endl;
    BasePlusCommissionEmployee
        employee3( "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
    cout << endl;
    return 0;
} // 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:**  
**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**

**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**

# 11. public, protected and private Inheritance

## ● public 继承

- 基类 public 成员 → 派生类 public 成员
- 基类 protected 成员 → 派生类 protected 成员
- 基类私有成员无法访问

# 11. public, protected and private Inheritance

- **protected 继承 (不是 *is-a* 关系)**

- 基类 public 和 protected 成员 → 派生类 protected 成员

- **private 继承 (不是 *is-a* 关系)**

- 基类 public 和 protected 成员 → 派生类 private 成员

# 11. public, protected and private Inheritance

Base-class member-access specifier	Type of inheritance		
	public inheritance	protected inheritance	private inheritance
public	<p>public in derived class.</p> <p>Can be accessed directly by member functions, friend functions and nonmember functions.</p>	<p>protected in derived class.</p> <p>Can be accessed directly by member functions and friend functions.</p>	<p>private in derived class.</p> <p>Can be accessed directly by member functions and friend functions.</p>
protected	<p>protected in derived class.</p> <p>Can be accessed directly by member functions and friend functions.</p>	<p>protected in derived class.</p> <p>Can be accessed directly by member functions and friend functions.</p>	<p>private in derived class.</p> <p>Can be accessed directly by member functions and friend functions.</p>
private	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and friend functions through public or protected member functions of the base class.</p>	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and friend functions through public or protected member functions of the base class.</p>	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and friend functions through public or protected member functions of the base class.</p>