Chapter 12

Object-Oriented Programming: Inheritance

OBJECTIVES

- To create classes by **inheriting** from existing classes.
- How inheritance promotes software reuse.
- The notions of **base classes** and **derived classes** and the relationships between them.
- The **protected** member access specifier.
- The use of constructors and destructors in inheritance hierarchies.
- The differences between **public**, **protected** and **private inheritance**.
- The use of inheritance to customize existing software.

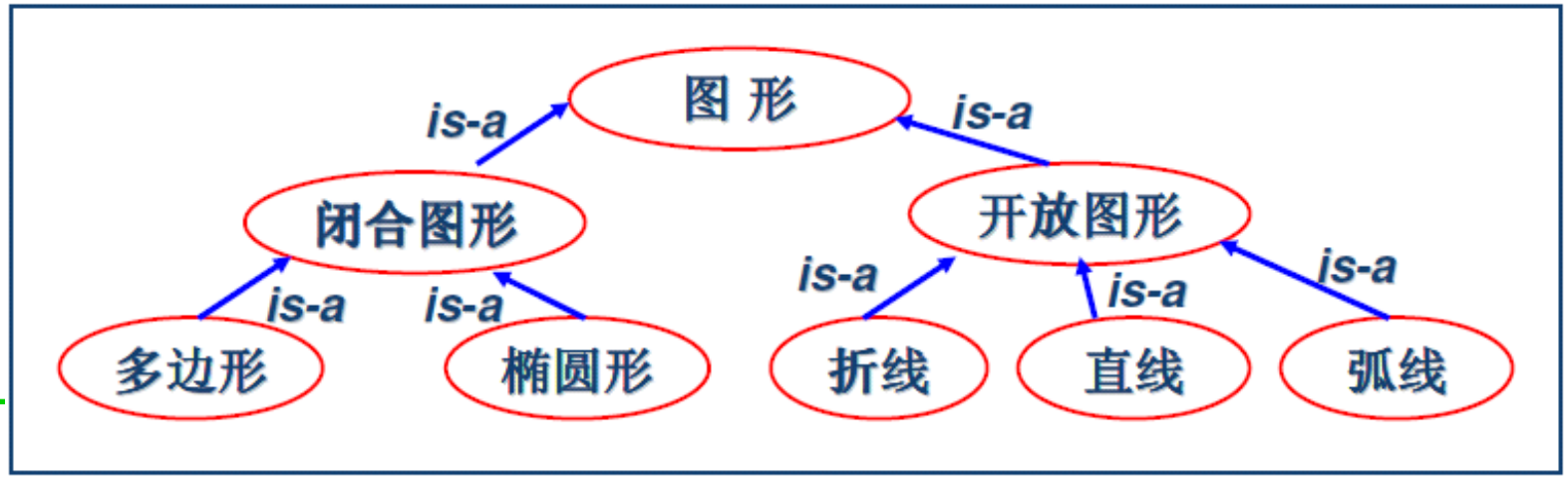
Topics

- **12.1 Introduction**
- 12.2 Base Classes and Derived Classes
- 12.3 protected Members
- 12.4 Relationship between Base Classes and Derived Classes
- 12.5 Constructors and Destructors in Derived Classes
- 12.6 public, protected and private Inheritance
- 12.7 Software Engineering with Inheritance

12.1 Introduction
--类之间的关系

- *has-a* relation
  Composition 组合
- *is-a* relation
  Inheritance 继承

- base class / derived class
  (基类 / 派生类)
- direct / indirect base class
  (直接 / 间接基类)
- single / multiple inheritance
- 3 kinds of inheritance

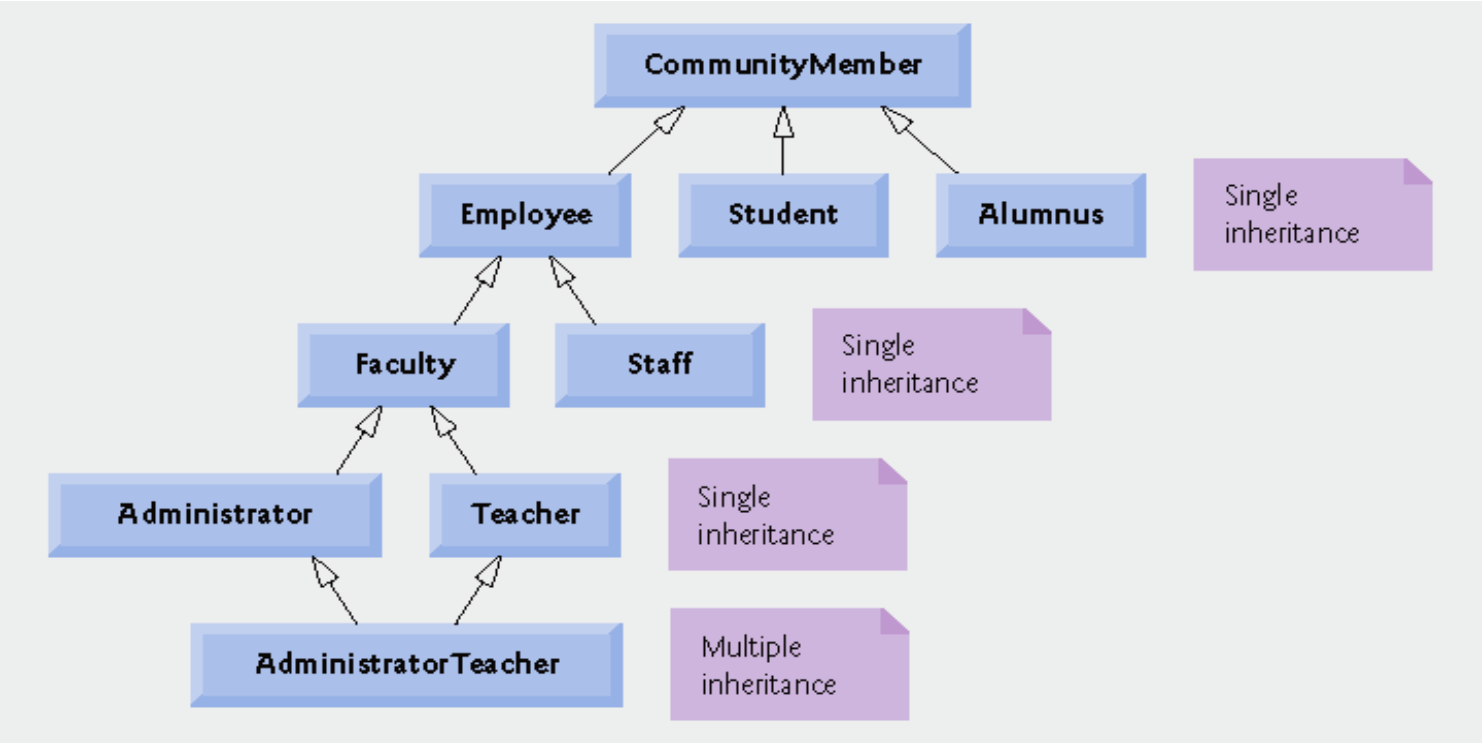


Topics

- 12.1 Introduction
- **12.2 Base Classes and Derived Classes**
- 12.3 protected Members
- 12.4 Relationship between Base Classes and Derived Classes
- 12.5 Constructors and Destructors in Derived Classes
- 12.6 public, protected and private Inheritance
- 12.7 Software Engineering with Inheritance

12.2 Base Classes and Derived Classes

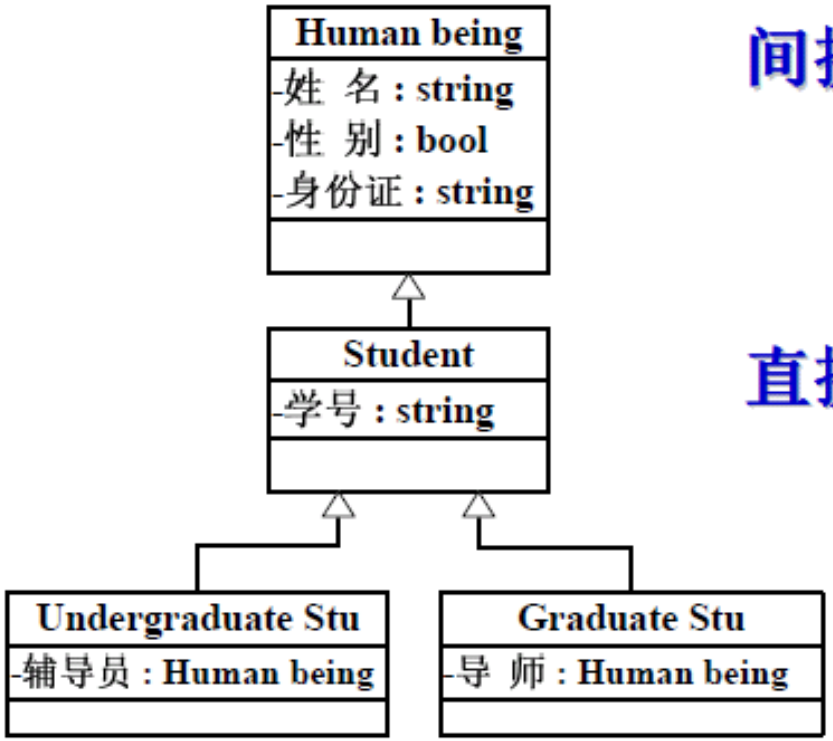
- 继承的层次关系(社区大学成员)



- 继承机制的主要作用: 软件复用, 支持软件的增量开发

12.2 Base Classes and Derived Classes

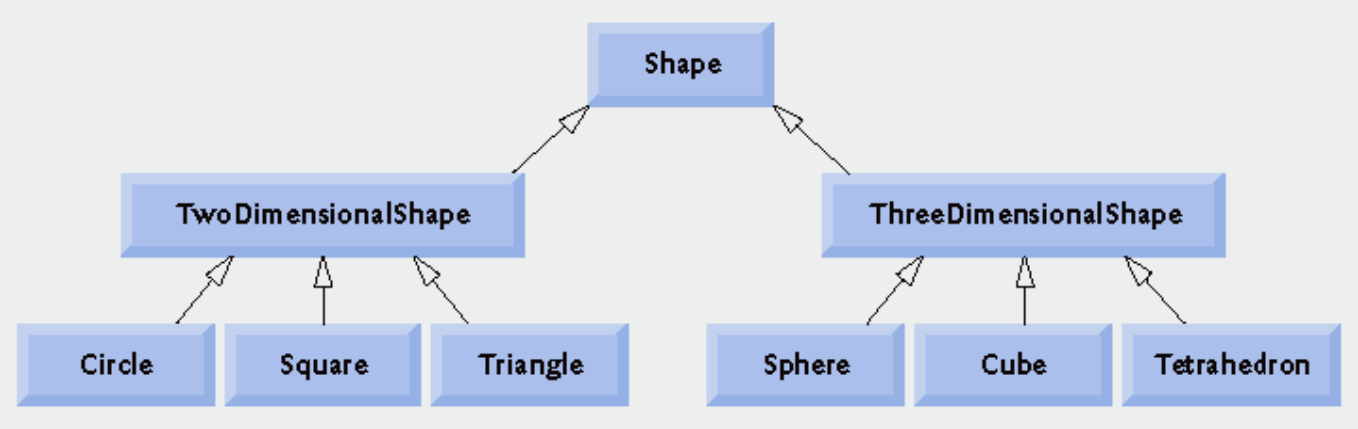
- **Base class**: 基类, 被继承的类
- **Derived class**: 派生类, 继承后得到类



Inheritance hierarchy 树形的层次关系图

12.2 Base Classes and Derived Classes

- 类继承的语法:
- class TwoDimensionalShape : public Shape



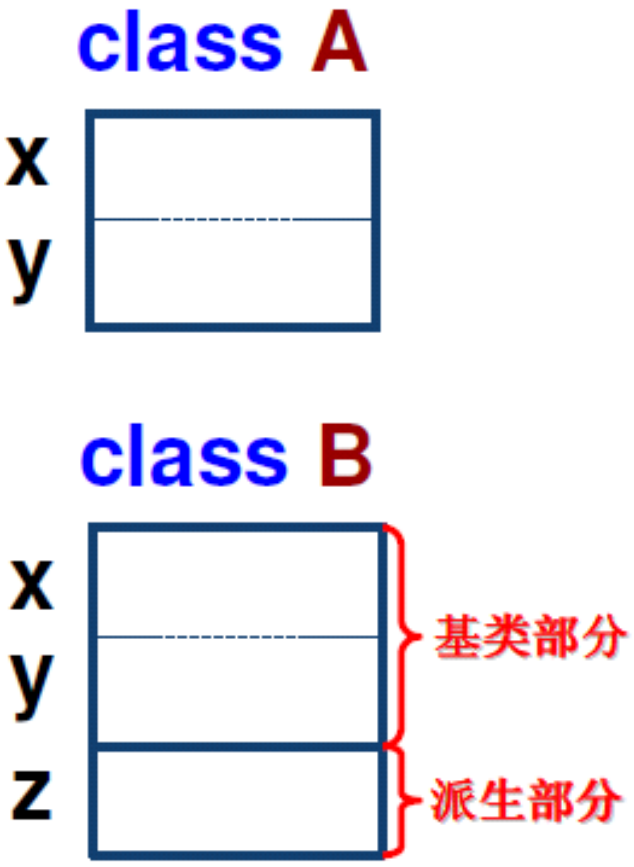
- With public inheritance(公有继承), all other base-class members retain their original member access when they become members of the derived class.
- Note: friend, constructor, destructor functions are not inherited.

12.2 Base Classes and Derived Classes

```
class A{
public:
   int x, y;
};
class B : public A{
public:
   int z;
};

int main()
{
   cout << "Size of A is " << sizeof(A) << endl
        << "Size of B is " << sizeof(B) << endl;
   return 0;
}
```

```
Size of A is 8
Size of B is 12
```



Topics

- 12.1 Introduction
- 12.2 Base Classes and Derived Classes
- **12.3 protected Members**
- 12.4 Relationship between Base Classes and Derived Classes
- 12.5 Constructors and Destructors in Derived Classes
- 12.6 public, protected and private Inheritance
- 12.7 Software Engineering with Inheritance

12.3 protected Members

- 解决问题: 派生类访问基类成员的权限控制
- 类有两种用户:
  - 对象(句柄): 对类进行实例化
  - 派生类: 在该类的基础上派生并设计新类

	类对象	派生类
public	/	/
protected	X	/
private	X	X

12.3 protected Members

- 基类的public成员能够被程序中所有函数访问
- 基类的private成员只能被基类的成员和友元函数访问
- 基类的protected成员只能被基类的成员和友元函数+ 派生类的成员和友元函数访问
- • 注意: 不能被类的对象访问

12.3 protected Members

- 派生类如何使用基类的成员？
- • 派生类可以直接通过成员名来使用基类的public成员和protected成员

- • 派生类可以重定义(redefine)基类的成员，并且依然可以通过以下方式访问基类的public/protected成员：

  base-class :: 成员名

- • 重新定义(redefine): 在派生类中给出基类的同名成员

Topics

- 12.1 Introduction
- 12.2 Base Classes and Derived Classes
- 12.3 protected Members
- **12.4 Relationship between Base Classes and Derived Classes**
- 12.5 Constructors and Destructors in Derived Classes
- 12.6 public, protected and private Inheritance
- 12.7 Software Engineering with Inheritance

12.4 Relationship between Base Classes and Derived Classes

- Commission Employee 佣金制雇员
- Base-salaried commission employees 带底薪的佣金制雇员

12.4.1 CommissionEmployee类

12.4.2 完全重写的 BasePlusCommissionEmployee类

12.4.3 继承+ 访问基类private成员

12.4.4 继承+ 访问基类protected成员

12.4.5 继承+ 通过public函数访问private数据

12.4 Relationship between Base Classes and Derived Classes

- class CommissionEmployee

  (first name, last name, social security number, commission rate and gross sales amount)

程序解读（P389）

12.4 Relationship between Base Classes and Derived Classes

- class BasePlusCommissionEmployee

  (first name, last name, social security number, commission rate, gross sales amount and base salary)

- class CommissionEmployee

  (first name, last name, social security number, commission rate, gross sales amount)

程序解读（P392）

12.4 Relationship between Base Classes and Derived Classes

- Define a new version of

  BasePlusCommissionEmployee class that **inherits directly** from class **CommissionEmployee**

程序解读（P397）

注意点

- 使用#include包含基类的头文件

- • 告诉编译器基类的存在(基类名)
- • 让编译器根据类定义确定对象大小以分配内存: 派生类的对象大小取决于派生类显式定义的数据成员和继承自基类(直接+间接)的数据成员
- • 让编译器能够判断派生类是否正确地使用了基类的成员

注意点

- 继承的语法

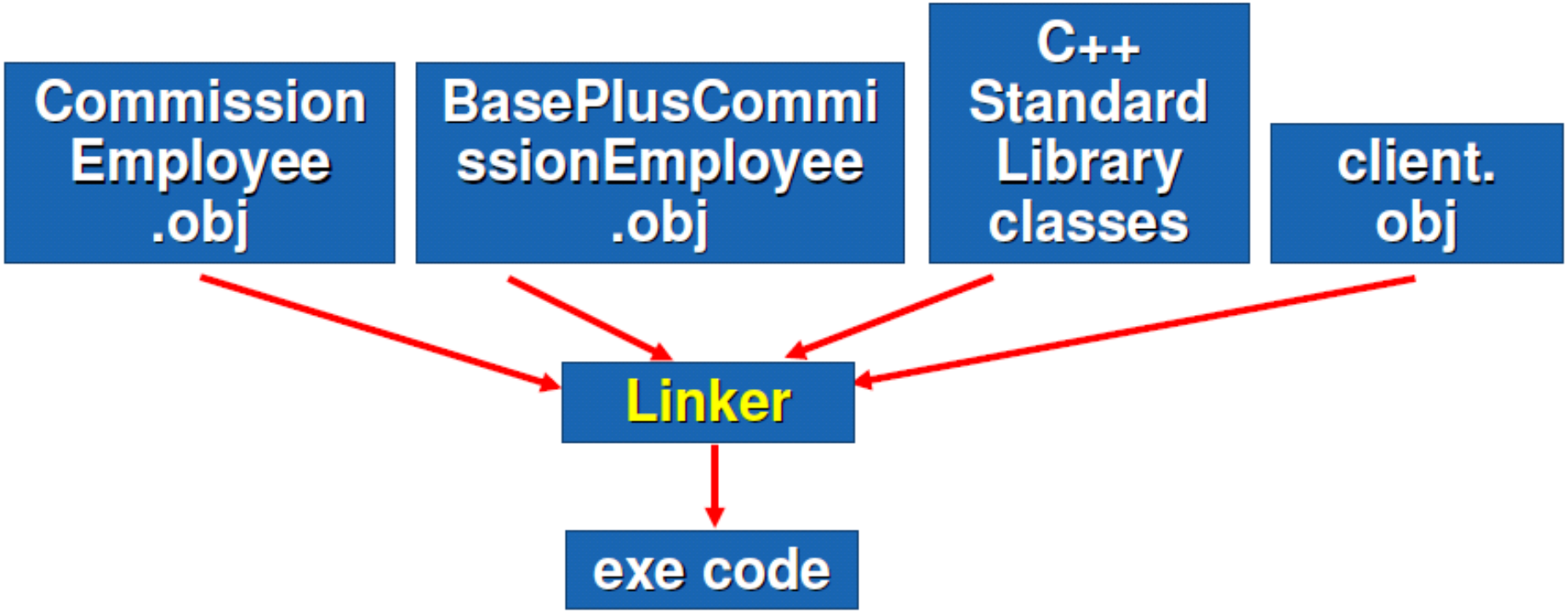
  class BasePlusCommissionEmployee :

  public CommissionEmployee

- 基类在派生类构造函数初始化列表中初始化
  - 一般应显式调用构造函数进行初始化
  - 如果未显式初始化, 则编译器隐性调用缺省构造函数

注意点

- 链接过程


Commission Employee .obj
BasePlusCommissionEmployee .obj
C++ Standard Library classes
client. obj
Linker
exe code

12.4 Relationship between Base Classes and Derived Classes

- 使用protected数据成员替代private数据成员

程序解读（P400）

12.4 Relationship between Base Classes and Derived Classes

- 影响数据的有效性检查
- • 在派生类中可以直接修改基类的数据成员
- 派生类依赖于基类的实现
- • 基类的数据成员发生改变有可能影响派生类的实现
- • 软件健壮性差

12.4 Relationship between Base Classes and Derived Classes

- 私有数据成员+ public函数接口

程序解读（P402）

注意点

- ① 通过调用基类的public成员函数来访问基类的私有数据成员
- ② 当功能相同时, 尽量调用成员函数, 以避免代码拷贝
- ③ 注意print()的重定义: 调用基类的print()成员函数时, 一定要使用“基类名::”, 否则会引起无限递归
- ④ 符合软件工程要求: 使用继承, 通过调用成员函数隐藏了数据, 保证了数据的一致性

redefine 与overload的区别

```
1.  class A
2.  {
3.  public:
4.      void f(){ cout << "A::f()" << endl; }
5.  };
6.  class B: public A
7.  {
8.  public:
9.      void f(){ cout << "B::f()" << endl; }
10.     void h(){
11.         f();
12.         A::f();
13.     }
14. };
15.
16. int main()
17. {
18.     B b;
19.     b.f();
20.     b.A::f();
21.     return 0;
22. }
```

```
B::f()
A::f()
```

redefine 与overload的区别

```
1.  class A
2.  {
3.  public:
4.      void f(){ cout << "A::f()" << endl; }
5.  };
6.  class B: public A
7.  {
8.  public:
9.      void f( int n){ cout << "B::f()" << endl; }
10.     void h(){
11.         f(0);
12.         A::f();
13.     }
14. };
15.
16. int main()
17. {
18.     B b;
19.     b.f();
20.
21.     return 0;
22. }
```

b.A::f();

error C2660: 'f' : function does not take 0 parameters

Topics

- 12.1 Introduction
- 12.2 Base Classes and Derived Classes
- 12.3 protected Members
- 12.4 Relationship between Base Classes and Derived Classes
- **12.5 Constructors and Destructors in Derived Classes**
- 12.6 public, protected and private Inheritance
- 12.7 Software Engineering with Inheritance

12.5 Constructors and Destructors in Derived Classes

- 基类先构造, 派生类后构造

A → B

- 派生类先析构, 基类后析构

B → A

12.5 Constructors and Destructors in Derived Classes

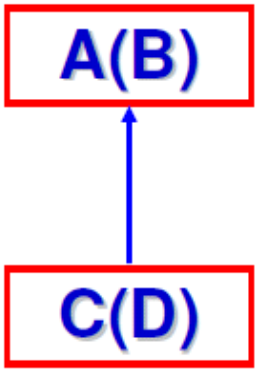
- 若类中含有其他类的对象(组合)

  构造: 先基类后派生类, 先被包含类后宿主类

  B → A → D → C

  析构: 与构造顺序相反

  C → D → A → B



12.5 Constructors and Destructors in Derived Classes

- 全局对象: 在任何函数(含main)执行前, 构造; 在程序结束时, 析构.
- 局部对象
  - 自动变量: 对象定义时, 构造; 块结束时, 析构.
  - 静态变量: 首次定义时, 构造; 程序结束时, 析构.
- 多个全局和静态对象(均为静态存储类别)析构顺序恰好与构造顺序相反.
- 特例1: 调用exit函数退出程序执行时, 不调用剩余自动对象的析构函数.
- 特例2: 调用abort函数退出程序执行时, 不调用任何剩余对象的析构函数.

```
1.  int main()
2.  {
3.     cout << fixed << setprecision( 2 );
4.     { // begin new scope
5.        CommissionEmployee employee1
6.           "Bob", "Lewis", "333-33-3333",
7.     } // end scope

8.     cout << endl;
9.     BasePlusCommissionEmployee
10.       employee2( "Lisa", "Jones", "555-55-5555",
11.
12.    cout << endl;
13.    BasePlusCommissionEmployee
14.       employee3( "Mark", "Sands", "888-88-8888"
15.    cout << endl;
16.    return 0;
17. } // end main
```

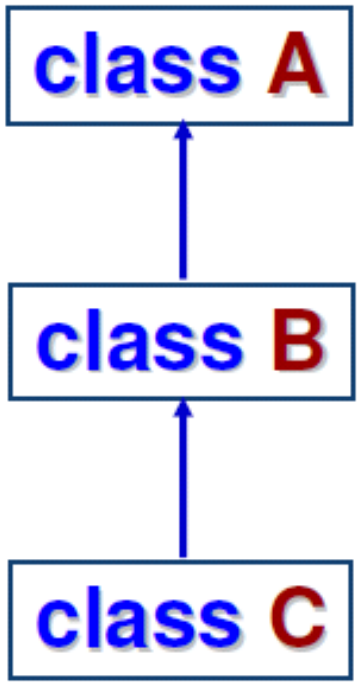
```
CommissionEmployee constructor:
CommissionEmployee destructor:
CommissionEmployee constructor:
BasePlueCommissionEmployee constructor:
CommissionEmployee constructor:
BasePlueCommissionEmployee constructor:
BasePlueCommissionEmployee destructor:
CommissionEmployee destructor:
BasePlueCommissionEmployee destructor:
CommissionEmployee destructor:
```

Topics

- 12.1 Introduction
- 12.2 Base Classes and Derived Classes
- 12.3 protected Members
- 12.4 Relationship between Base Classes and Derived Classes
- 12.5 Constructors and Destructors in Derived Classes
- **12.6 public, protected and private Inheritance**
- 12.7 Software Engineering with Inheritance

12.6 public, protected and private Inheritance

- B继承A, C继承B



- A的对象和B能否访问A的成员

  取决于A的成员的访问权限设置

- B的对象和C能否访问B中继承自A的成员

  取决于A的成员的访问权限设置和B继承A的类型, 即public / protected / private inheritance

12.6 public, protected and private Inheritance

继承方式	基 类(A)	派生类(B)
public	public成员 →	public成员
	protected成员 →	protected成员
	private成员 →	不可见
private	public成员 →	private成员
	protected成员 →	private成员
	private成员 →	不可见
protected	public成员 →	protedted成员
	protected成员 →	protected成员
	private成员 →	不可见

```
1.  class A{
2.  public:
3.     int x;
4.  protected:
5.     int y;
6.  private:
7.     int z;
8.  };
9.  class B : public A{
10. public:
11.    int m;
12.    int f(){ x=1;   public
13.             y=2;   protected
14.          X  z=3; } private
15. };
16. int main()
17. {
18.    B obj;
19.    obj.x = 10;
20.    obj.y = 20;   X
21.    obj.z = 30;   X
22.    obj.m = 40;
23. }
```

```
1.  class A{
2.  public:
3.     int x;
4.  protected:
5.     int y;
6.  private:
7.     int z;
8.  };
9.  class B : protected A{
10. protected:
11.    int m;
12.    int f(){ x=1;   protected
13.             y=2;   protected
14.          X  z=3; } private
15. };
16. int main()
17. {
18.    B obj;
19.    obj.x = 10;   X
20.    obj.y = 20;   X
21.    obj.z = 30;   X
22.    obj.m = 40;   X
23. }
```

```
1.  class A{
2.  public:
3.     int x;
4.  protected:
5.     int y;
6.  private:
7.     int z;
8.  };
9.  class B : private A{
10. private:
11.    int m;
12.   int f(){ x=1;   private
13.             y=2;   private
14.          X  z=3; } private
15. };
16. int main()
17. {
18.    B obj;
19.    obj.x = 10;   X
20.    obj.y = 20;   X
21.    obj.z = 30;   X
22.    obj.m = 40;   X
23. }
```

Topics

- 12.1 Introduction
- 12.2 Base Classes and Derived Classes
- 12.3 protected Members
- 12.4 Relationship between Base Classes and Derived Classes
- 12.5 Constructors and Destructors in Derived Classes
- 12.6 public, protected and private Inheritance
- **12.7 Software Engineering with Inheritance**

12.7 Software Engineering with Inheritance

- 派生类的程序员不需了解基类的源代码, 只需与目标代码连接即可
  - • ISV (Independent software vendors 独立软件供应商) 为目标代码提供头文件, 发放许可
  - • 继承实用的类库, 提高软件复用
- 软件工程提示
- 提取出共同的属性和行为并把它们封装在一个基类中, 然后通过继承生成派生类

Summary

- 基类和派生类的定义
- protected成员，派生类如何访问基类成员
- 继承关系中构造函数和析构函数顺序
- 三种继承

Homework

- 实验必选题目：

  12.10

- 实验任选题目：

- 作业题目：