

繼承概念的優點

- 類別再使用(程式碼再使用)
- 抽象化概念再使用
- 類別關係階層化

Q: 人、黑猩猩與猴子的
的有哪些共同屬性?

2-3 簡介繼承

靈長類

- 繼承

特徵(屬性)

- 手、足、脊椎、大拇指型態

行為:

- 育兒: 哺乳
- 使用工具

人類

黑猩猩

彌猴

- 人類是靈長類的一種
- 人類繼承了靈長類應有的特徵及行為
- 人類繼承了靈長類

繼承的概念

員工(employee)

[屬性]: 姓名、年齡、年資、年薪

[動作]: 計算健保扣除額、計算所得稅

繼承

工人

課員

課長

主任

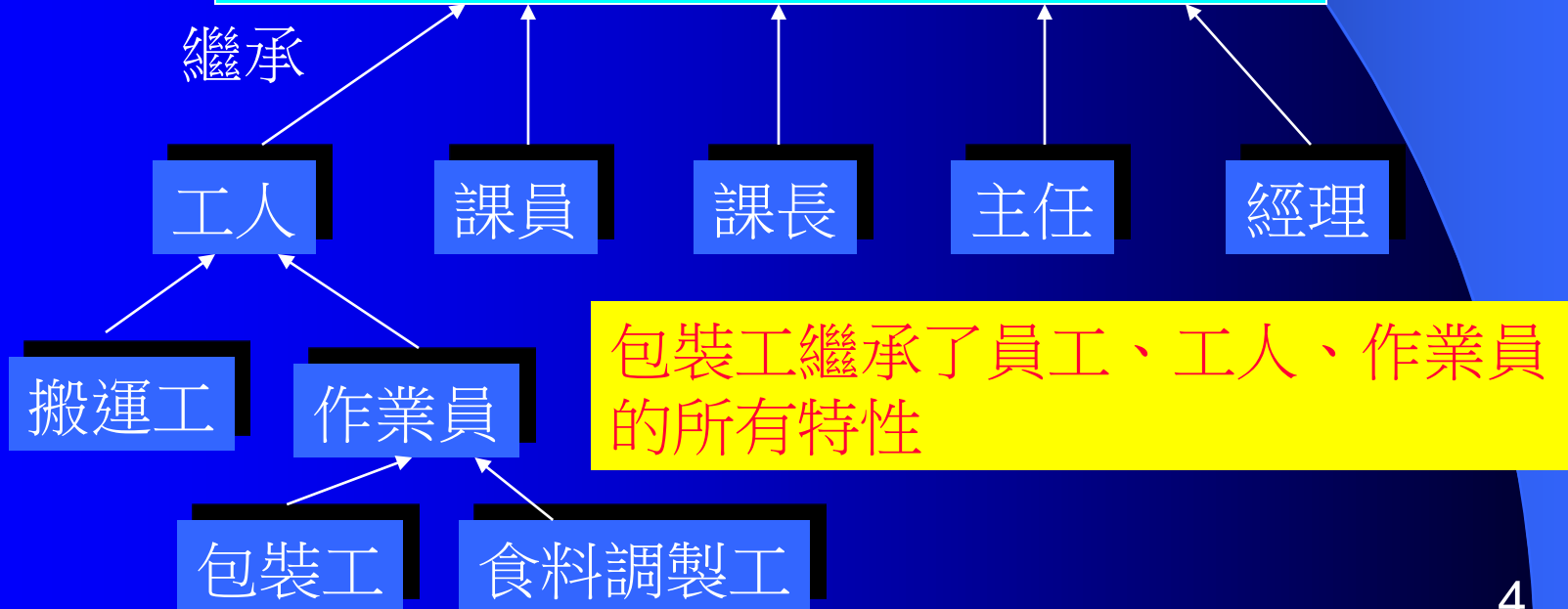
經理

繼承階層圖

員工(employee)

[屬性]: 姓名、年齡、年資、年薪

[動作]: 計算健保扣除額、計算所得稅



特性的再使用、新增與修改

員工(employee)

[屬性]: 姓名、年齡、年資、年薪

[動作]: 計算健保扣除額、計算所得稅

繼承

工人

[屬性] 加班時數

New

搬運工

作業員

[動作] 職業訓練, 計算所得稅

New

Override

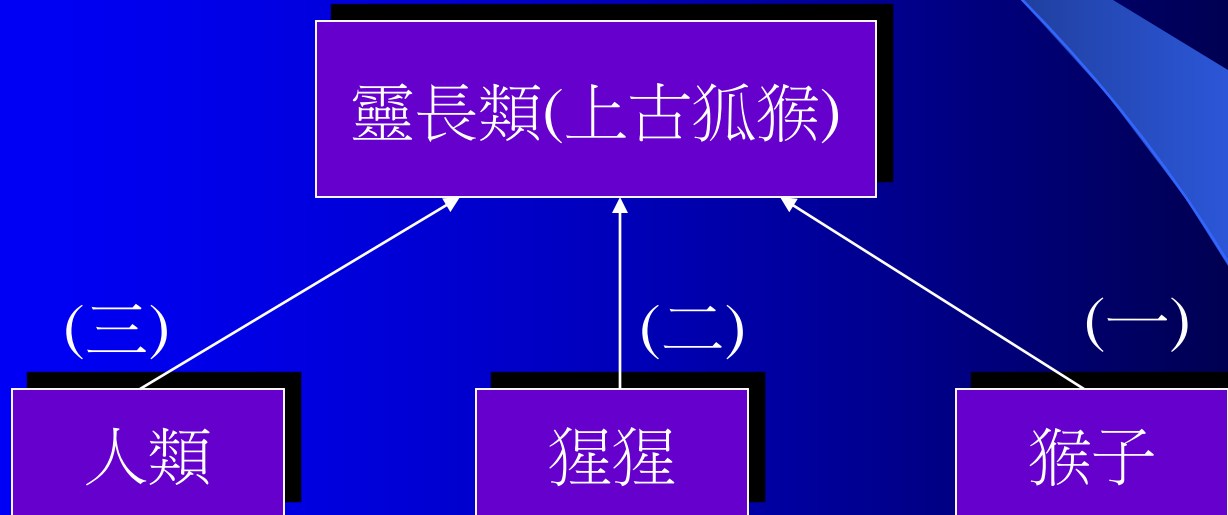
包裝工

食料調製工

請問包裝工的屬性與動作?

特性的再使用、新增與修改

- (1) 直接繼承 (2) 新增屬性或動作 (3) 修改屬性或動作

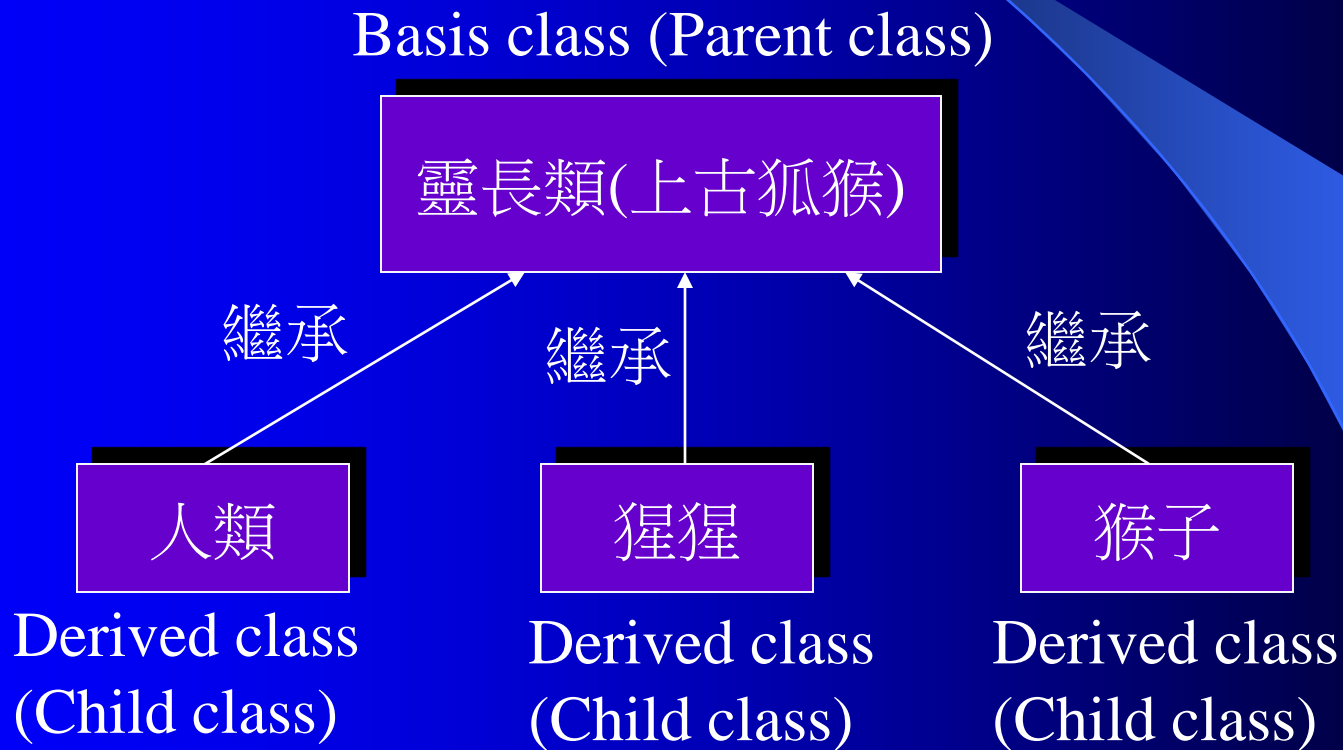


[屬性]: 膚色 *New*
[動作] 直立行走 *New*
 使用工具 *Override*

[動作] 手語 *New*

Nothing *New*

基底類別(basis class)與 衍生類別(derived class)

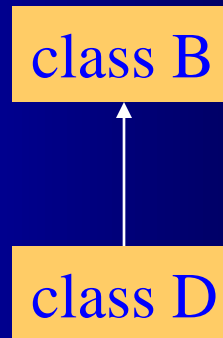


一、直接繼承

- 繼承的語法

```
class B {  
    int i; //屬性  
public: //動作  
    void set_i(int n) { i = n;}  
    int get_i() { return i; }  
};  
class D: public B { };
```

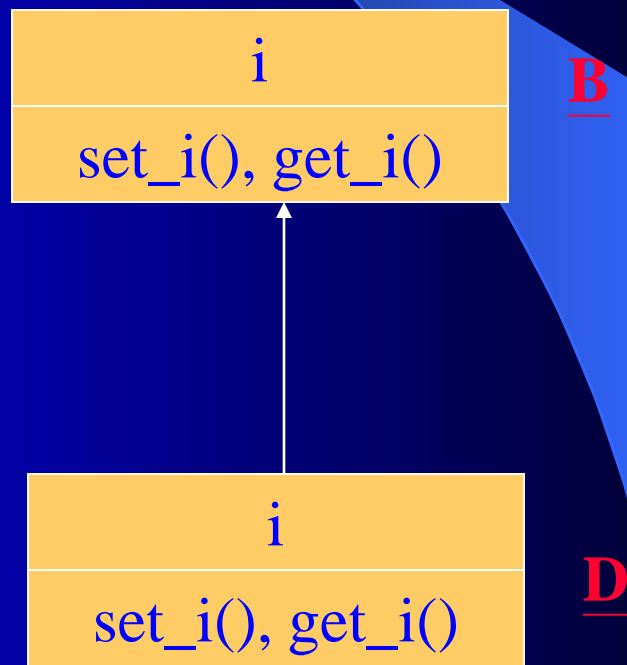
class D 繼承了 class B



直接繼承

```
class B {  
    int i; //屬性  
public: // 動作  
    void set_i(int n) { i = n;}  
    int get_i() { return i; }  
};  
class D: public B { };
```

Q: class D 有哪些資料
成員? 成員函數?

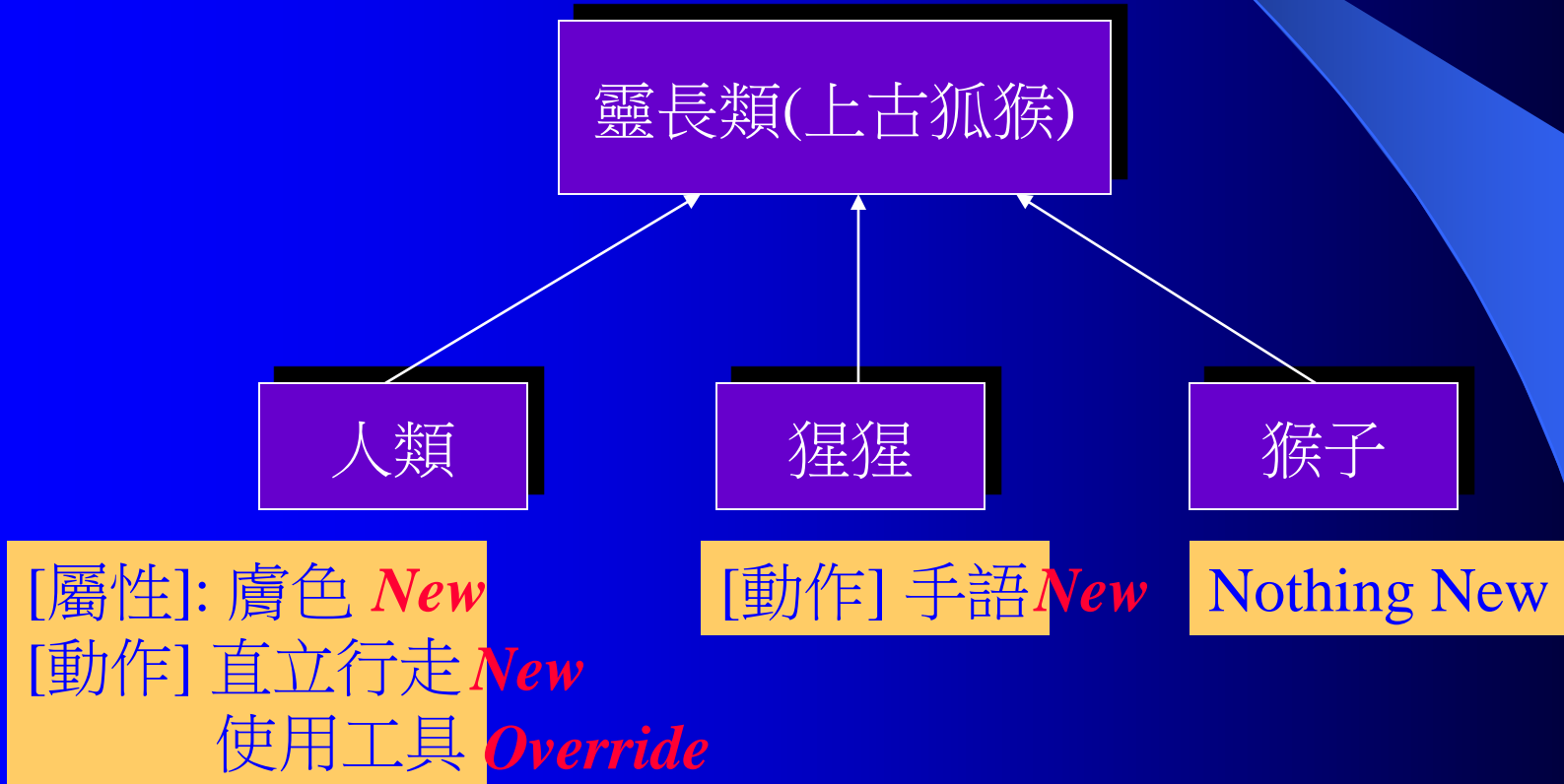


使用衍生類別

```
class B {....get_i(); ...set_i(int n); ..} ;  
class D: public B { } ;  
void main() {  
    D ob ;  
    ob.set_i(10) ;  
    cout << ob.get_i() <<endl ;  
}
```

衍生類別的內容(或任務)

- (1) 直接繼承 (2) 新增屬性或動作 (3) 修改屬性或動作



二、新增成員

```
class OneDim{  
    int x ;  
public:  
    void setx(int n) { x = n ;}  
    void showx() {cout << x <<endl ; }  
};
```

```
class TwoDim: public OneDim {
```

```
    int y ; //新增的data memeber
```

```
public:
```

```
    void sety(int n) { y = n ;} // 新增的member functions
```

```
    void showy() {cout << y << endl ; } //新增的functions
```

```
};
```

class TwoDim的成員

```
int x; int y ;  
setx(int n) ;  
showx() ;  
sety(int n) ;  
showy() ;
```

使用class TwoDim

```
void main() {  
    TwoDim td ;  
    td.setx(5) ; //基底類別  
    td.sety(10) ; //衍生類別  
    td.showx() ; //基底類別  
    td.showy() ; //衍生類別  
    // try, 可乎?  
    td.x = 10 ; td.y = 20 ;  
}
```

class TwoDim的成員

```
int x; int y ;  
setx(int n) ;  
showx() ;  
sety(int n) ;  
showy() ;
```

對外界(類別使用者)而言，TwoDim的成員中
哪些是 private? 哪些是 public?

在衍生類別中使用基底類別成員

```
class OneDim{
    int x ;
public:
    void setx(int n) { x = n ;}
    void showx() {cout << x <<endl ; }
};

class TwoDim: public OneDim {
    int y;
public:
    void setxy(int a, int b) { x= a; y = b ;}
    void showxy() {cout << x << “ “ << y << endl ; }
};
```

class TwoDim的成員

```
int x; int y ;
setx(int n) ;
showx() ;
setxy(int a, int b) ;
showxy() ;
```

Q: 子類別可以取用
父類別中的私有成員?

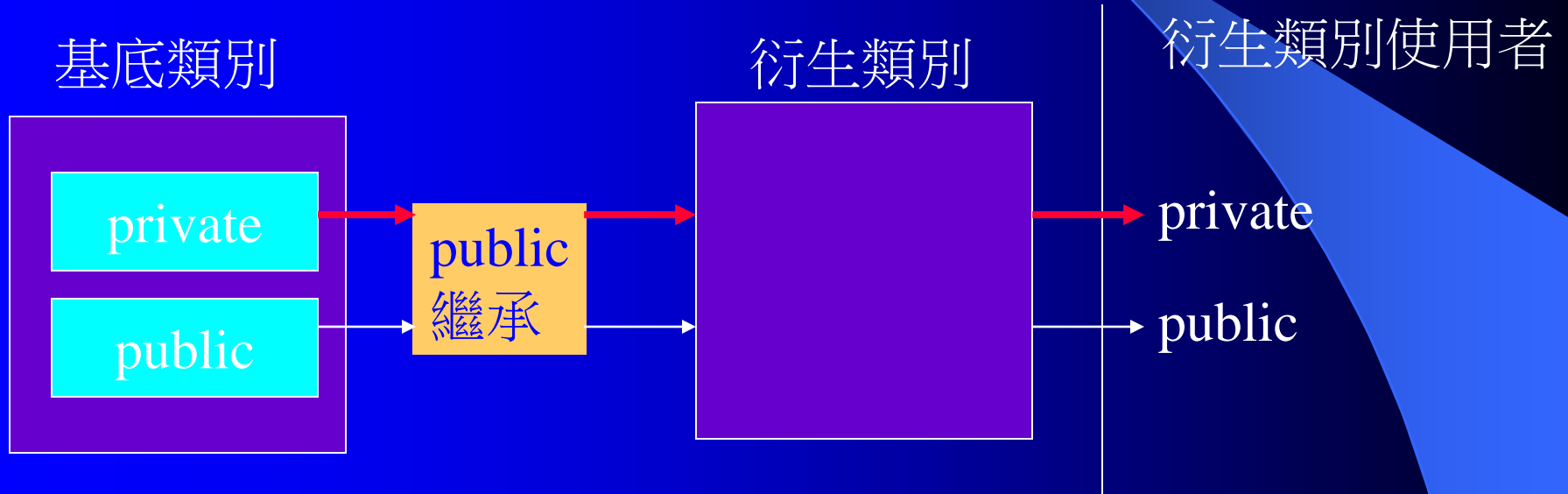
在衍生類別中使用基底類別成員

```
class OneDim{
    int x ;
public:
    void setx(int n) { x = n ;}
    void showx() {cout << x <<endl ; }
};

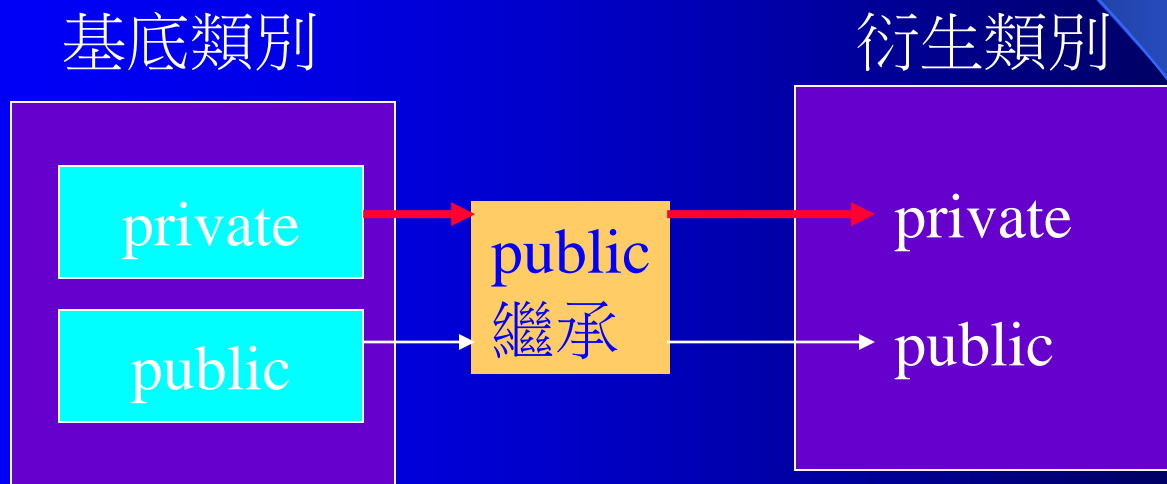
class TwoDim: public OneDim {
    int y;
public:
    void setxy(int a, int b) { setx(a); y = b ;}
    void showxy() { showx(); cout << y << endl ; }
};
```

仔細觀察與上例
有何差異?

父類別成員存取權



父類別成員存取權



EX: 完成以下程式

先寫出ThreeDim
有哪些成員?

```
class ThreeDim: public TwoDim {  
    int z ;  
public:  
    setxyz(int a, int b, int c) {....}  
    showxyz() {....}  
};
```

三、修改基底類別成員

```
class TwoDim {  
    int x, y ;  
public:  
    void setxy(int a, int b) { x = a; y = b; }  
    void show() { cout << x << " " << y ; }  
};
```

Q: 寫出ThreeDim的成員?

```
class ThreeDim :public TwoDim {  
    int z;  
public:  
    void setxyz(int a, int b, int c) { setxy(a, b); z = c ; }  
    void show() { TwoDim::show(); cout << " " << z ; }  
};
```

Override: 將父類別的函數重新定義一次
(函數名稱、回傳值均相同)

使用ThreeDim

```
void main() {  
    ThreeDim point3D ;  
    point3D.setxyz(1,2,3); //呼叫誰的set()  
    point3D.show();      //呼叫誰的show()  
}
```

EX: 回答以下問題

```
class TwoDim {
    int x, y ;
public:
    void set(int a, int b) { x = a; y = b; }
    void show() { cout << x << " " << y ;}
};
class ThreeDim :public TwoDim {
    int z;
public:
    void set(int a, int b, int c) {.....}
    void show() { TwoDim::show(); cout << " " << z ;}
};
```

Q1: 完成set(int a, int b, int c)

Q2: 寫出ThreeDim的成員?

Q3: set()算不算override?

2-3 範例一

fruit

name, color color: red, yellow, green, orange
getName()
getColor()
set(name, color)

Apple

cooking
set(name, color, cooking)
show()

Orange

juicy
set(name, color, juicy)
show()

main()

```
void main() {  
    Apple a; Orange o ;  
    a.set("Washington", red, yes) ; a.show();  
    o.set("Sunkys", orange, false) ; o.show() ;  
}
```

繼承的用處

- 描述物件間is-a-kind of的關係
- 提昇程式碼的再使用率
- 有利於類別的更新與修正

描述物件之間的關係

- has-a

- Each man has two legs

```
class man {  
    Leg legs[2];  
    .....  
};
```

- is-a-kind-of

- 工讀生是學校員工的一種
- 人是一種靈長類

增加程式碼的再使用率

```
class TwoDim {  
    int x, y ;  
public:  
    void setxy(int a, int b) { x = a; y = b; }  
    void show() { cout << x << " " << y ;}  
}  
class ThreeDim {  
    int z;  
public:  
    void setxyz(int a, int b, int c) { setxy(a, b); z = c ; }  
    void show() { TwoDim::show(); cout << " " << z ;}  
}
```

有利於類別的更新與修正

- 當你使用的類別庫設計不良或有所不足時，該如何？

// 你覺得 需要一個int sum();

// 你覺得 delete(int value)效率很差

```
class list {  
    ... // data member  
public:  
    void insert(int n) {...}  
    void delete(int value) {.....}  
    void show() {.....}  
};
```

後記: 回憶Window Programming

```
#include <afxwin.h>           //載入afxwin標頭檔

class MyApp : public CWinApp   //繼承CWinApp
{
public:
    BOOL InitInstance()       //程式進入點
    {
        CFrameWnd *Frame = new CFrameWnd(); //建立CFrameWnd
        物件(產生)
        m_pMainWnd = Frame;    //將m_pMainWnd設定為Frame

        Frame->Create(NULL,"Hello MFC");    //建立視窗(建立)
        Frame->ShowWindow(SW_SHOW);

        return true;
    }
};

MyApp a_app; //建立應用程式物件
```

7-2 使用保護成員

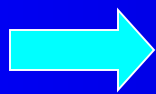
```
class OneDim {  
    int x;  
public:  
    void setx(int a) { x = a ; }  
    void getx() { return x ; }  
};  
class TwoDim:public OneDim {  
    int y ;  
    void distance(TwoDim& pt) {  
        return sqrt(pow(getx()-pt.getx(),2), pow(y-pt.y, 2)) ;  
    }  
    ...  
}
```

Q: 我可否直接在TwoDim
取用x?

Q: 缺點?

需求

- 我想在子類別中直接使用父類別的私有成員，但是外界仍不可直接存取這個成員。



將私有成員改成保護成員

範例一

```
class samp {  
    int a ;  
    protected:  
    int b ;  
    public:  
    int c ;  
    samp(int n, int m) { a = n; b=m; }  
};  
void main() {  
    samp ob(10, 20) ;  
    ob.a = 10;          ob.b = 20;          ob.c = 30 ;  
}
```

範例二

```
class base {
protected:
    int a, b ;
public:
    void setab(int n, int m) { a= n ; b=m;}
};
class derived: public base {
    int c ;
public:
    void setc(int n) {c = n ;}
    void show() { cout << a << ' ' << b << ' ' << c <<endl ;}
};

void main()
{
    derived d;
    d.setab(10,20);
    d.setc(30);
    d.show();
}
```


7-1 基底類別存取控制

```
class OneDim {...}
```

```
class TwoDim: public OneDim {...}
```

公有繼承(public)

```
class Base {  
private:  
    int a ;  
protected:  
    int b;  
public:  
    int c;  
};
```

```
class Derived: public Base {  
    // int a; private  
    // int b; protected  
    // int c; public  
};  
void main() {  
    Derived d ;  
    // a, b: private  
    // c: public  
}
```

私有繼承(private)

```
class Base {  
private:  
    int a ;  
protected:  
    int b;  
public:  
    int c;  
};
```

```
class Derived: private Base {  
    // int a; private  
    // int b; private  
    // int c; private  
};  
void main() {  
    Derived d ;  
    // a, b, c: private  
}
```

保護繼承(protected)

```
class Base {  
private:  
    int a ;  
protected:  
    int b;  
public:  
    int c;  
};
```

```
class Derived: protected Base {  
    // int a; private  
    // int b; protected  
    // int c; protected  
};  
void main() {  
    Derived d ;  
    // a, b, c: private  
}
```

7-3 建構子、解構子與繼承

- 以下成員不會被子類別繼承
 - (1) 建構子、解構子與operator=()
 - (2) 夥伴函數

```
class B {  
    int x ;  
public:  
    B(int n) { x = n ;}  
};  
class D: public B {} ;  
void main() { D d(10) ; .....}
```

物件欄位的生成消滅次序

```
class employee {  
    string name ;  
    int ID ;  
    .....  
};  
class worker: public employee {  
    float overtime ;  
    .....  
};
```

```
void main() {  
    worker w ;  
}
```



範例一：生成與消滅的次序

```
class base {  
    public:  
        base() { cout << “base constructing” <<endl;}  
        ~base() { cout << “base destructing” <<endl;}  
};  
class derived: public base {  
    public:  
        derived() { cout << “derived constructing” <<endl;}  
        ~derived() { cout << “derived destructing” <<endl;}  
};
```

範例二：傳引數給衍生類別建構子

```
class base {  
public:  
    base() { ... }  
    ~base() { ... }  
};  
class derived: public base {  
    int j ;  
public:  
    derived(int n) { j = n ; }  
    ~derived() { ... }  
};
```

```
void main() {  
    derived ob(10) ;  
}
```


成員的初始化: 使用Initializer

```
class point {  
    int x, y ;  
public  
    point(const point& p):x(p.x), y(p.y) { }  
};  
class triangle {  
    point pt1, pt2, pt3;  
public:  
    triangle(point p1, point p2, point p3):pt1(p1), pt2(p2), pt3(p3) { }  
};
```

範例三

```
class base {
    int i ;
public:
    base(int n) { i =n;}
};
// 接class derived
void main() {
    derived d1(10) ; // i=j=10
    derived d2(5, 3) ; //i=5,j=3
}
```

```
class derived: public base {
    int j ;
public:
    derived(int n):base(n){
        j = n ;
    }
    derived(int a, int b):base(a) {
        j = b;
    }
};
```