final, override

```
struct B {
    virtual void f1(int) const ;
    virtual void f2 ();
    void f3 () ;
    virtual void f5 (int) final;

};
struct D: B {
    void f1(int) const override ;  //正确：  f1与基类中的f1 匹配
    void f2(int) override ;        //错误：  B没有形如f2(int) 的函数。int f2()?
    void f3 () override ;          //错误：  f3不是虚函数
    void f4 () override ;          //错误：  B没有名为f4的函数
    void f5 (int) ;                //错误：  B已经将f5声明成final
}
```

虚函数

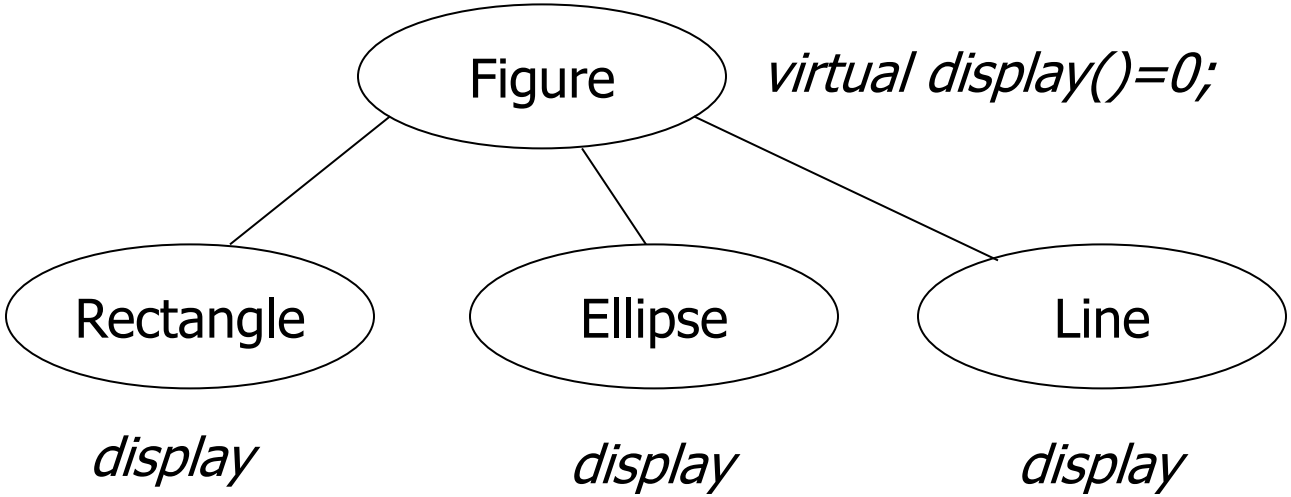
- 纯虚函数和抽象类
  - 纯虚函数
    - 声明时在函数原型后面加上 = 0
    - 往往只给出函数声明，不给出实现
  - 抽象类
    - 至少包含一个纯虚函数
    - 不能用于创建对象
    - 为派生类提供框架，派生类提供抽象基类的所有成员函数的实现

Means " not there"

```
virtual int f()=0;

class AbstractClass
{     ...
  public:
      virtual int f()=0;
};
```

_pure_virtual_called

虚函数




```
Figure *a[100];
a[0] = new Rectangle();
a[1] = new Ellipse();
a[2] = new Line();
...
for (int i=0; i<num_of_figures; i++)    a[i]->display();
```

```
AbstractFactory* fac;

case MAC:
    fac = new MacFactory;
case WIN:
    fac = new WinFactory;
```

WinButton
WinLabel
……

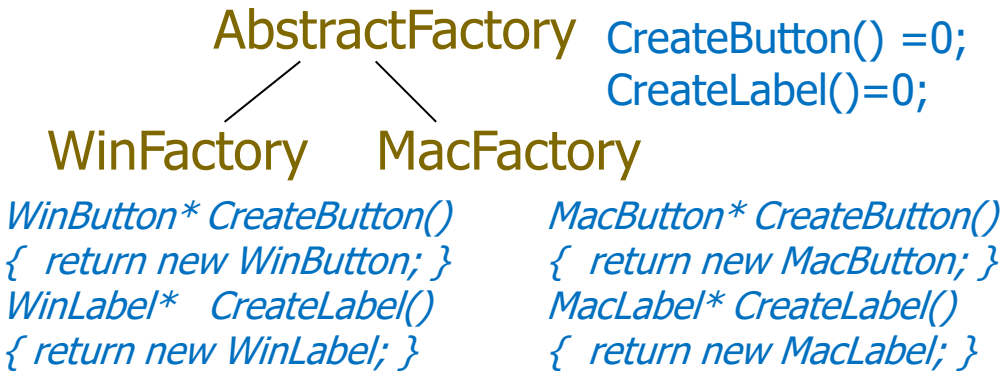
Step1:
提供Windows GUI类库

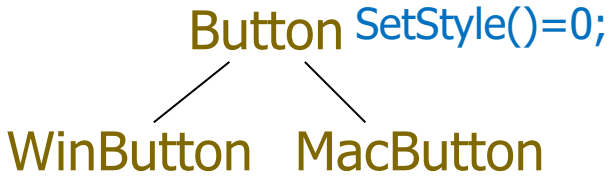
Step2:
增加对Mac的支持

MacButton
MacLabel
……

```
Button *pb= fac->CreateButton();
……
pb->SetStyle( … );

Label *pl=  fac->CreateLabel();
……
pl->SetText( … );
```

Step3:
增加对用户跨平台设计的支持





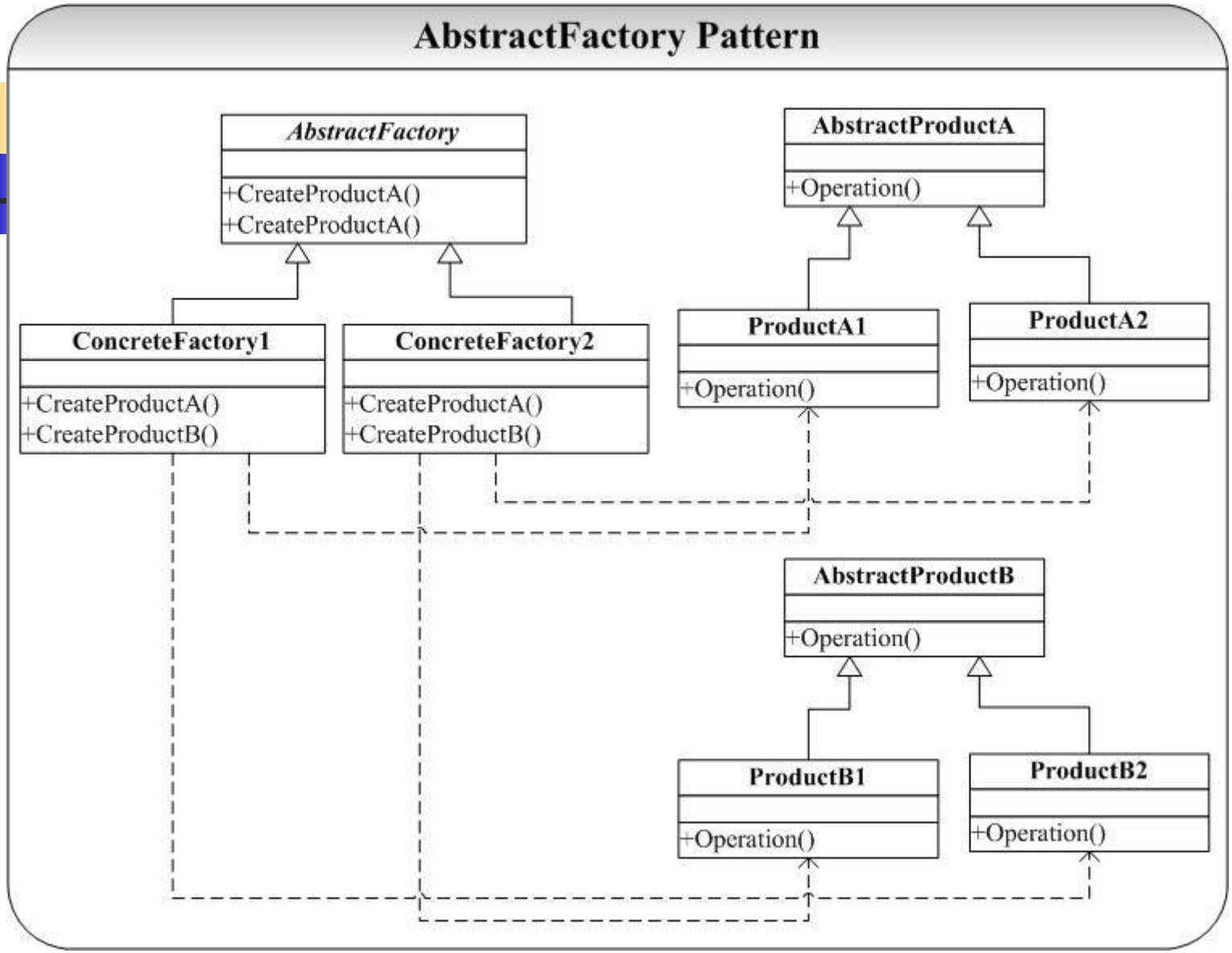
```
class Button; // Abstract Class
class MacButton: public Button {};
class WinButton: public Button {};
class Label; // Abstract Class
class MacLabel: public Label {};
class WinLabel: public Label {};
```

```
AbstractFactory* fac;
switch (style) {
case MAC:
    fac = new MacFactory;
    break;
case WIN:
    fac = new WinFactory;
    break;
}
Button* button = fac->CreateButton();
Label* Label = fac->CreateLabel();
```

```
class AbstractFactory {
public:
    virtual Button* CreateButton() =0;
    virtual Label* CreateLabel() =0;
};
class MacFactory: public AbstractFactory {
public:
    MacButton* CreateButton() { return new MacButton; }
    MacLabel* CreateLabel() {  return new MacLabel; }
};
class WinFactory: public AbstractFactory {
public:
    WinButton* CreateButton() { return new WinButton; }
    WinLabel*   CreateLabel() { return new WinLabel; }
};
```

抽象工厂模式

Abstact Factory


AbstractFactory Pattern
AbstractFactory
+CreateProductA()
+CreateProductA()
AbstractProductA
+Operation()
ConcreteFactory1
+CreateProductA()
+CreateProductB()
ConcreteFactory2
+CreateProductA()
+CreateProductB()
ProductA1
+Operation()
ProductA2
+Operation()
AbstractProductB
+Operation()
ProductB1
+Operation()
ProductB2
+Operation()

虚函数

- 虚析构函数

```
class B {...};
class D: public B{...};
B* p = new D;                    ?:  delete p;
```

```
class mystring {...}
class B {...}
class D: public B{
mystring name; ...}
B* p = new D;                    ?:  delete p;
```

虚函数

class FlyingBird
class NonFlyingBird

Penguin

virtual void fly() { error("Penguins can't fly!"); }

- 确定public inheritance,是真正意义的“is_a”关系
- 不要定义与继承而来的非虚成员函数同名的成员函数

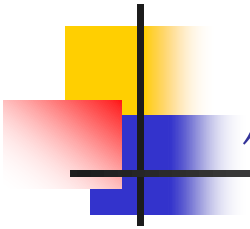
```
class Rectangle
{
    public:
    virtual void setHeight(int);
    virtual void setWidth(int);
            int height() const;
            int width() const;};

class Square: public Rectangle {
    public:
        void setLength (int);
    private:
        void setHeight(int );
        void setWidth(int );
... };
```

```
void Widen(Rectangle& r, int w)
{
    int oldHeight = r.height();
    r.setWidth(r.width() + w);
    assert(r.height() == oldHeight);
}
```

assert(s.width() == s.height());

```
Square s(1,1);
Rectangle *p = &s;
p->setHeight(10);
```

```
class B
{   public:
            void mf();
    ...    };
class D: public B {
   public:
            void mf();
... };

D x;
B* pB = &x;
pB->mf();     //B:mf
D* pD = &x;
pD->mf();    //D:mf
```



虚函数

- 明智地运用private Inheritance
  - Implemented-in-term-of
    - 需要使用Base Class中的protected成员，或重载virtual function
    - 不希望一个Base Class被client使用
  - 在设计层面无意义，只用于实现层面

```
class CHumanBeing { … };

class CStudent: private CHumanBeing { … };

void eat(const CHumanBeing& h)
{ … }

CHumanBeing a;  CStudent b;
eat(a);
eat(b);  //Error
```

虚函数

```
class Shape {
public:
    virtual void draw() const = 0;

    virtual void error(const string& msg);

    int objectID() const;
};
```

- 纯虚函数

  只有函数接口会被继承
  - 子类必须继承函数接口
  - （必须）提供实现代码

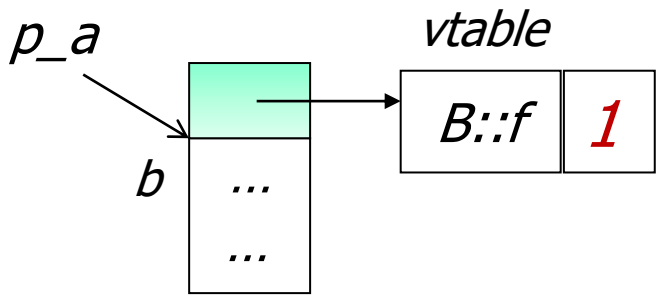
- 一般虚函数

  函数的接口及缺省实现代码都会被继承
  - 子类必须继承函数接口
  - 可以继承缺省实现代码

- 非虚函数

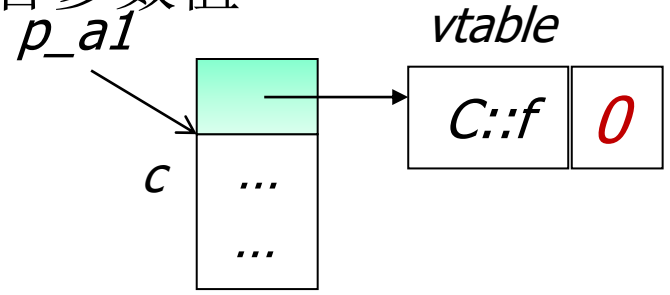
  函数的接口和其实现代码都会被继承
  - 必须同时继承接口和实现代码

```
(**((char *)p_a1 - 4))(p_a1)
char *q = *((char *)p_a1 - 4);
(*q)(p_a1, *q+4);
```



虚函数

- 绝对不要重新定义继承而来的缺省参数值
  - 静态绑定
  - 效率

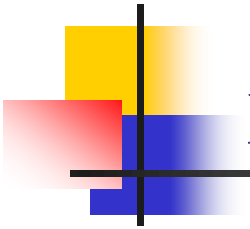



```
class A
{
public:
    virtual void f(int x=0) =0;
};

class B: public A
{
public:
    virtual void f(int x=1)
    { cout << x;}
};

class C: public A
{
public:
    virtual void f(int x) { cout<< x;}
};
```

```
A *p_a;
B b;
p_a = &b;
p_a->f();
```

```
A *p_a1;
C c;
p_a1 = &c;
p_a1->f();
```

对象中只记录虚函数的入口地址



多继承

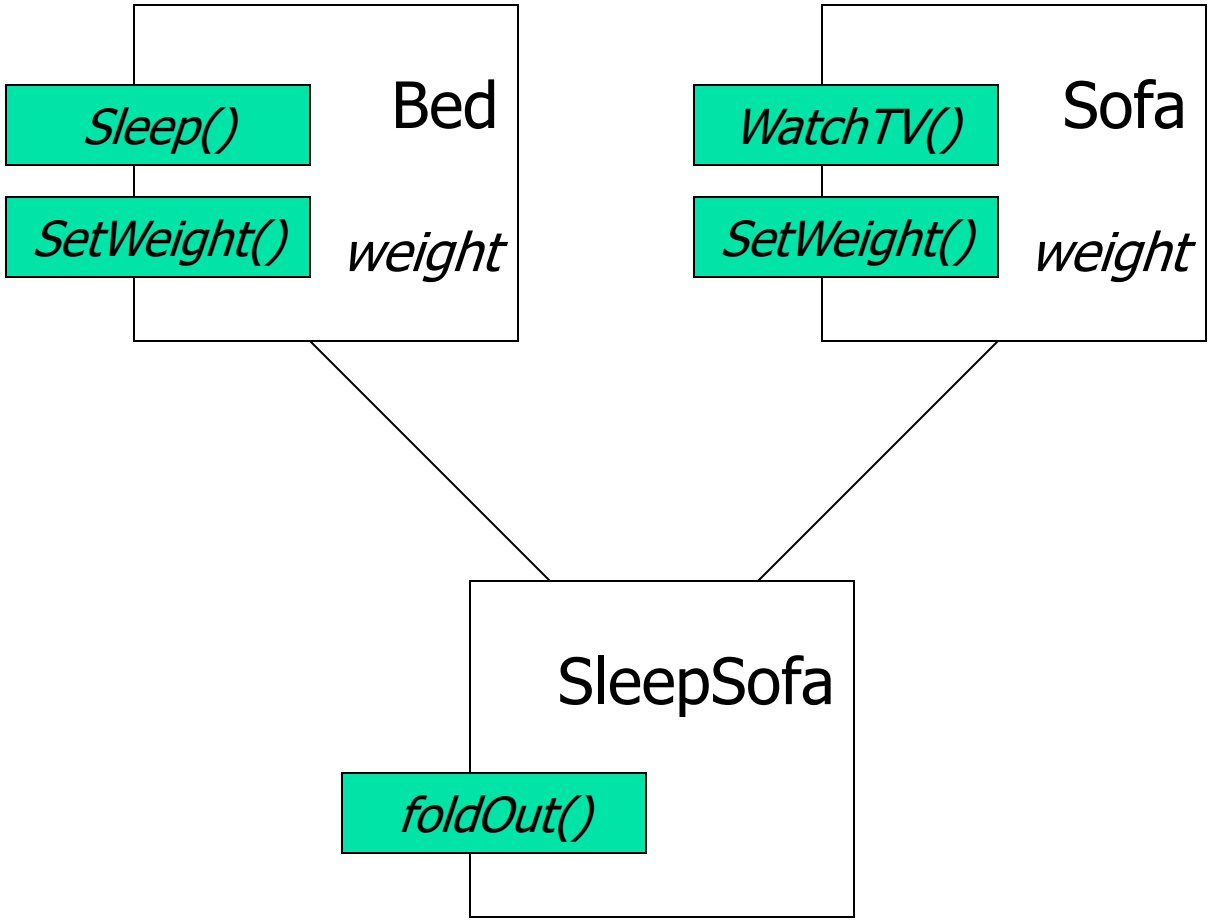
- 多继承
  - 定义

    class 〈派生类名〉：[〈继承方式〉]〈基类名1〉，
    [〈继承方式〉]〈基类名2〉，…

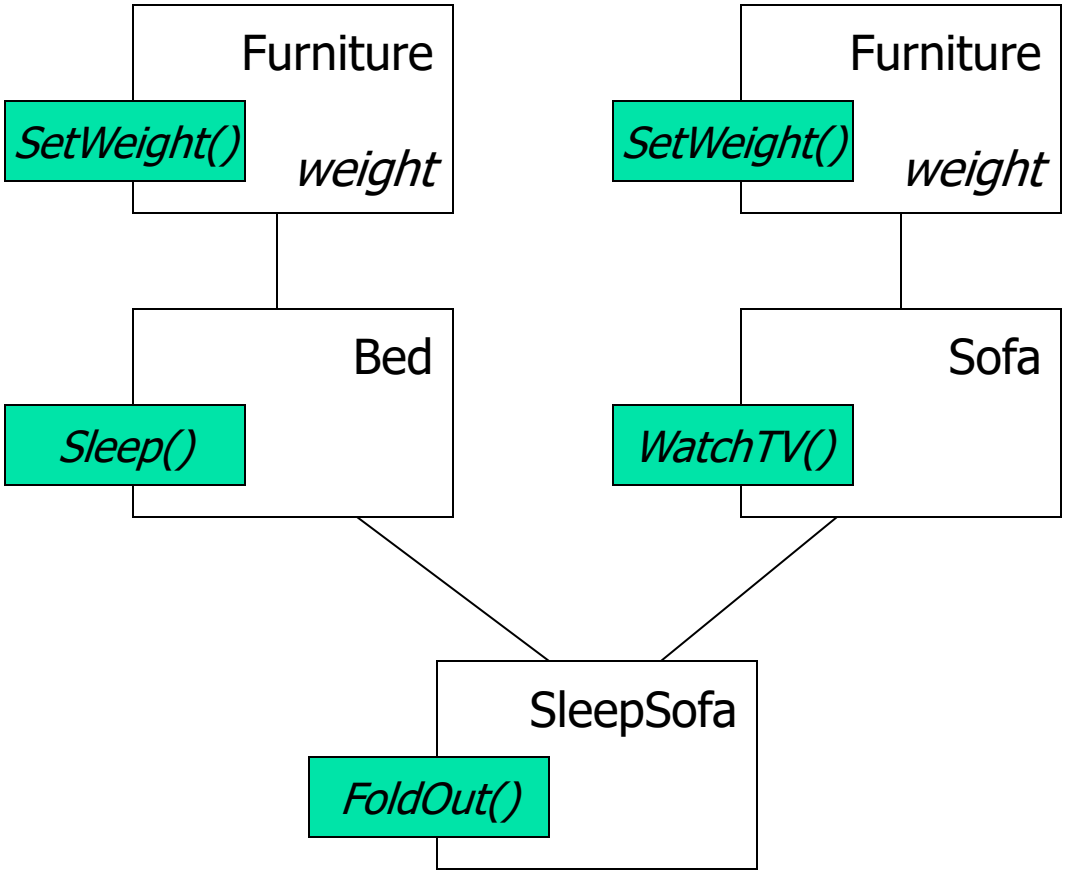
    { 〈成员表〉 *}*

    - 继承方式
      - *public*、*private* 、*protected*
    - 继承方式及访问控制的规定同单继承
    - 派生类拥有所有基类的所有成员

多继承

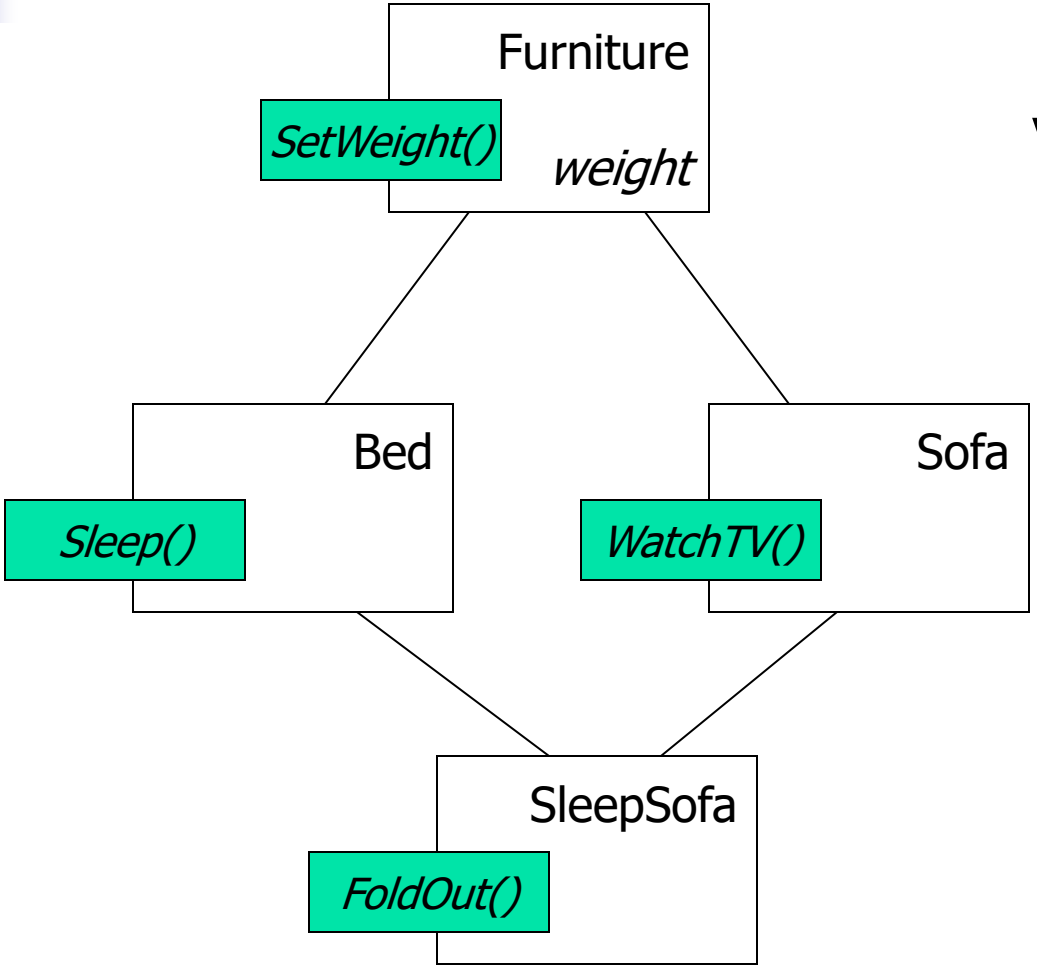

Bed
Sleep()
SetWeight()
weight
Sofa
WatchTV()
SetWeight()
weight
SleepSofa
foldOut()

多继承



Base-Class Decomposition

多继承



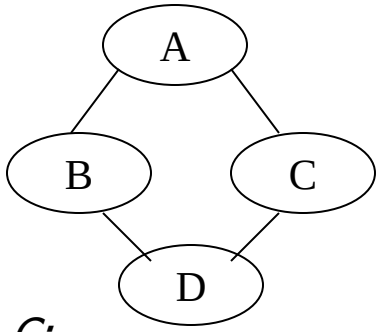
Virtual Inheritance

多继承

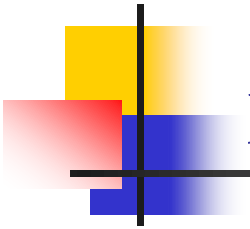
- 基类的声明次序决定：
  - 对基类构造函数/析构函数的调用次序
  - 对基类数据成员的存储安排

- 名冲突
  - <基类名>::<基类成员名>

```
class A
{  int x;
   ...
};
class B: A;
class C: A;
class D: B, C;
```



- 虚基类
  - 如果直接基类有公共的基类，则该公共基类中的成员变量在多继承的派生类中有多个副本



多继承

- 类D拥有两个x成员：B::x和C::x
- 虚基类
  - 合并

```
class A;
class B: virtual public A;
class C: public virtual A;
class D: B, C;
```

- 注意
  - 虚基类的构造函数由最新派生出的类的构造函数调用
  - 虚基类的构造函数优先非虚基类的构造函数执行

管理员: F:\Windows\System32\cmd.exe

```
class D size(28):
        +---
        | +--- (base class B1)
 0      | | {vfptr}
 4      | | x
        | +---
        | +--- (base class B2)
 8      | | {vfptr}
12      | | y
        | +---
        | +--- (base class B3)
16      | | {vfptr}
20      | | z
        | +---
24      | a
        +---

D::$vftable@B1@:
        | &D_meta
        |  0
 0      | &B1::v1
 1      | &D::vD

D::$vftable@B2@:
        | -8
 0      | &B2::v2

D::$vftable@B3@:
        | -16
 0      | &D::v3

D::v3 this adjustor: 16
D::vD this adjustor: 0
```