一、多态

1.多态的概念

多态:同一个事物，在不同场景下表现出的不同的状态

举例子来说明:见人说人话，见鬼说鬼话　学生买票的例子　自己想一些其他例子

2.多态的分类

静态多态(早绑定，静态联编): 在编译期间，根据所传递的实参类型或者实例化的类型,来确定到底应该调用那个函数即:在编译期间确定了函数的行为---函数重载、模板

动态多态(晚绑定,动态联编):在程序运行时，确定具体应该调用那个函数

3.动态多态的实现条件-- -在继承的体系中

>>虚函数&重写:基类中必须包含有虚函数(被virtual修饰的成员函数)，派生类必须要对基类的虚函数进行重写

>>关于虚函数调用:必须通过基类的指针或引用调用虚函数

体现:在程序运行时，基类的指针或引用指向那个子类的对象，就会调用那个子类的虚函数

4.重写

>> 1.基类中的函数一定是虚函数

>> 2.派生类虚函数必须与基类虚函数的原型一致：返回值类型 函数名字(参数列表)

例外:协变--基类虚函数返回值基类的指针或引用

派生类虚函数返回派生类的指针或引用基类虚函数和派生类虚函数的返回值类型可以不同

析构函数:如果将基类中析构函数设置成虚函数,派生类的析构函数提供，两个析构函数就可以构

成重写；两个析构函数名字不同

>>3.基类虚函数可以和派生类虚函数的访问权限不一-样

1>C+ +中构成重写的条件非常严格:虚函数 并且 原型一致

基类虚函数: virtual void TestFunc()

派生类虚函数: virtual void TetsFunc();//细节性的不同不容易发现，而导致重写失败

2>为了让编译器在编译期间帮助用户检测是否重写成功，C+ +11提供非常有用的关键字override:专门]让编译帮助用户检测派生类是否成功重写了基类的虚函数

如果重写成功:编译通过

如果重写失败:编译失败

final:如果用户不想要子类重写基类的虚函数,可以使用final修饰该关键字

3>函数重载、同名隐藏(重定义)、重写(覆盖)之间的区别?

函数重载概念:相同作用域、函数名字相同、参数列表不同(类型、个数、类型次序)

同名隐藏　　重写

相同点: 一个函数在基类中，一个函数在子类中

不同点:基类中函数是否为虚函数?

没有要求　　一定要是虚函数

函数原型是否相同?

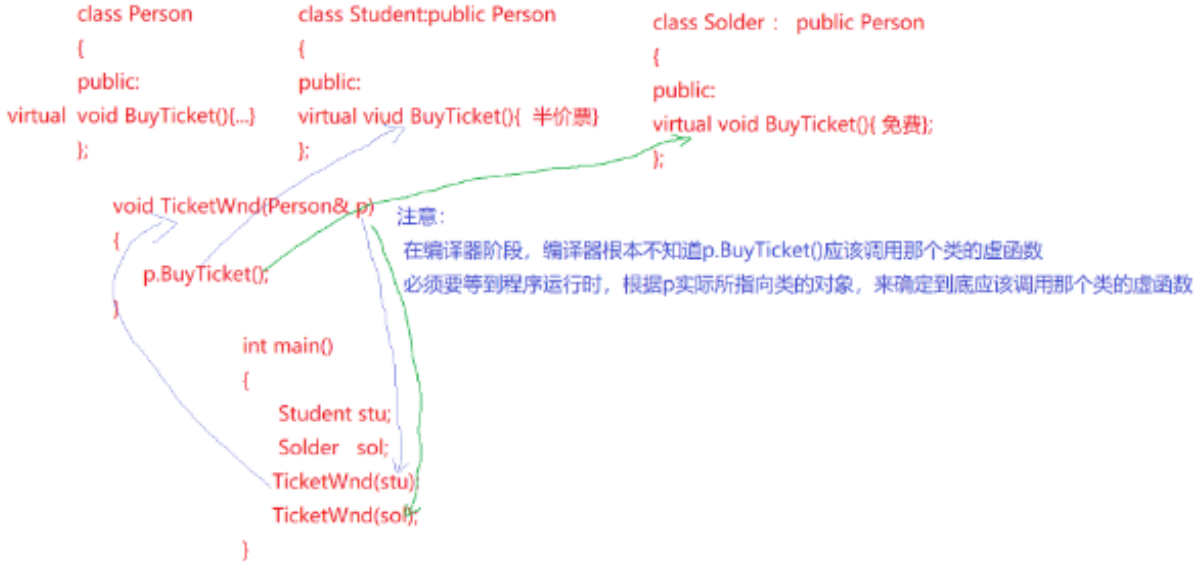
只要名字相同即可，其他没有要求　　原型必须要相同

重写的限制条件比同名隐藏更加的严格:

如果满则同名---但是没有满足重写的条件---定是同名隐藏

注意:同名隐藏和重写是两个不同的概念,不能说同名隐藏是一种特殊的重写

5、举例子说明

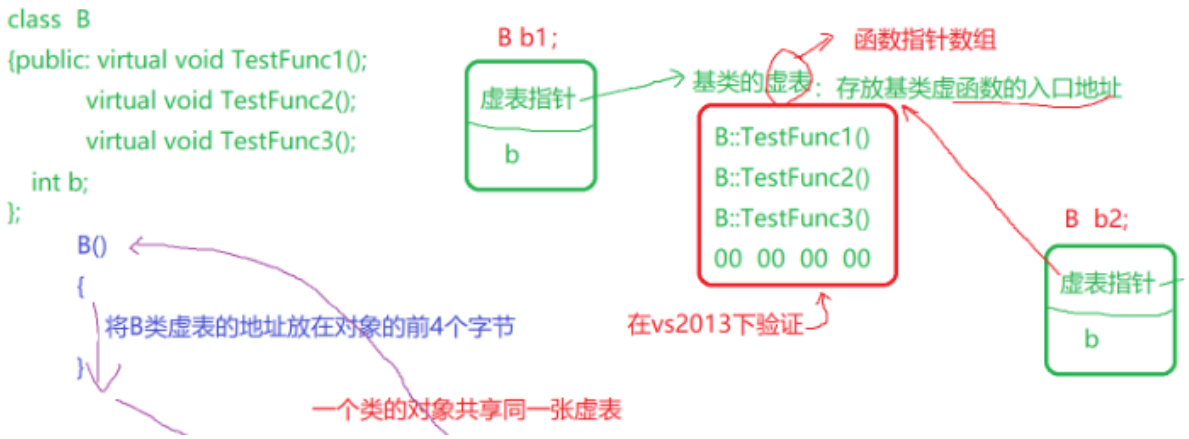


6、多态的实现原理

编译器在编译时会将类中的虚函数按照一定的规则存储在虚表中，在创建对象时，只需要将虚表的地址存储在对象的前四个字节

>>虚表构建过程

1、基类：编译器按照各个虚函数在类中声明的先后次序依次将虚函数保存在虚表中



2.子类虚表的构建过程

a.将基类虚表中内容拷贝一份到子类虚表中

b.如果子类重写了基类中那个虚函数，编译器会用子类自己虚函数的地址覆盖相同偏移量位置的基类虚函数地址

c.如果派生类新增加自己的虚函数，编译器会将派生类新增加的虚函数按照其在派生类中的声明次序依次放在虚表的最后