# 一、封装

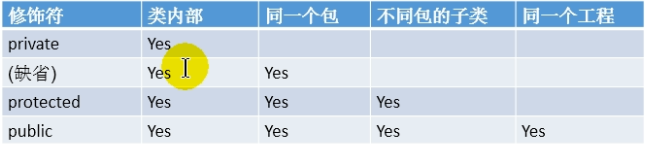
## 一、什么是封装

隐藏对象内部的复杂性，只对外公开简单的接口。便于外界调用，从而提

高系统的可扩展性、可维护性。通俗的说，把该隐藏的隐藏起来，该暴露

的暴露出来。这就是封装性的设计思想。

## 权限大小



# 二、继承

## 体现

一旦子类A继承父类B以后，子类A中就获取了父类B中声明的所有的属性和方法。  
特别的，父类中声明为private的属性或方法，子类继承父类以后，**仍然认为获取了父类中私有的结构**。  
**只有因为封装性的影响，使得子类不能直接调用父类的结构而已**。

## 2、子类的理解

子类继承父类以后，还可以声明自己特有的属性或方法：实现功能的拓展。  
子类和父类的关系，不同于子集和集合的关系。

子类要比父类强大，处理更多的事情

**extends：延展、扩展**

## 3、重写

重写的规定：  
方法的声明：权限修饰符 返回值类型 方法名(形参列表) throws 异常的类型{  
 //方法体  
}  
约定俗称：子类中的叫重写的方法，父类中的叫被重写的方法  
 ① 子类重写的方法的**方法名**和形参列表与父类被重写的方法的方法名和形参列表相同  
 ② 子类重写的方法的**权限修饰符**不小于父类被重写的方法的权限修饰符  
 >特殊情况：子类**不能**重写父类中声明为**private**权限的方法  
 ③ **返回值类型**：  
 >父类被重写的方法的返回值类型是void，则子类重写的方法的返回值 类型只能是void  
 >父类被重写的方法的返回值类型是A类型，则子类重写的方法的返回

值类型可以是A类或A类的**子类**  
 >父类被重写的方法的返回值类型是**基本数据类型**(比如：double)，则子 类重写的方法的返回值类型必须是**相同**的基本数据类型(必须也是

double)  
 ④ 子类重写的方法抛出的**异常类型**不大于父类被重写的方法抛出的异常 类型（具体放到异常处理时候讲）  
   
 子类和父类中的同名同参数的方法要么都声明为非static的（考虑重写），要么都声明为**static的（不是重写）**。Static 修饰的方法不能重写

## 子类对象实例化

1、当我们通过子类的构造器创建子类对象时，我们一定会直接或间接的调用其**父类的构造器**，进而调用父类的父类的构造器，...

2、直到调用了java.lang.Object类中空参的构造器为止。正因为**加载**过所有的父类的结构，所以才可以看到内存中有类中的结构，子类对象才可以考虑进行调用

3、明确：虽然创建子类对象时，调用了父类的构造器，但是自始至终就创建过**一个对象**，即为new的子类对象

# 多态

对象的多态性：父类的引用指向子类的对象

可以直接应用在抽象类和接口上

## 1、引用变量的两个类型

Java引用变量有两个类型：**编译时类型**和**运行时类型**。编译时类型由声明

该变量时使用的类型决定，运行时类型由实际赋给该变量的对象决定。简

称：**编译时，看左边；运行时，看右边**。

**若编译时类型和运行时类型不一致，就出现了对象的多态性(Polymorphism) ！！！**

多态情况下，

“看左边”：看的是父类的引用（父类中不具备子类特有的方法）

“看右边”：看的是子类的对象（实际运行的是子类重写父类的方法）

## 2、属性的多态

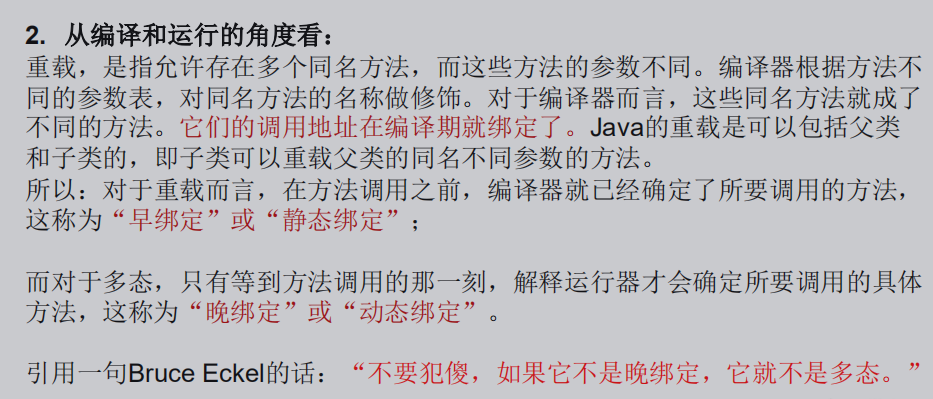
**·属性没有多态性**

一个引用类型变量如果声明为父类的类型，但实际引用的是子类

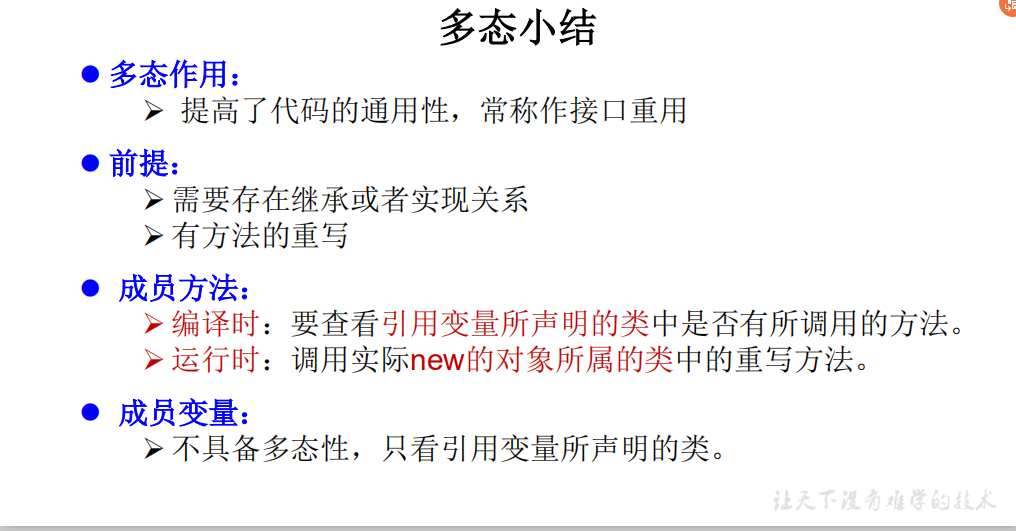
对象，那么该变量就不能再访问子类中添加的属性和方法

因为属性是在编译时确定的，编译时e为Person类型，没有school成员变量，因而编译错误。

## 3、动态绑定



## 4、小结



## 5、向下转型

