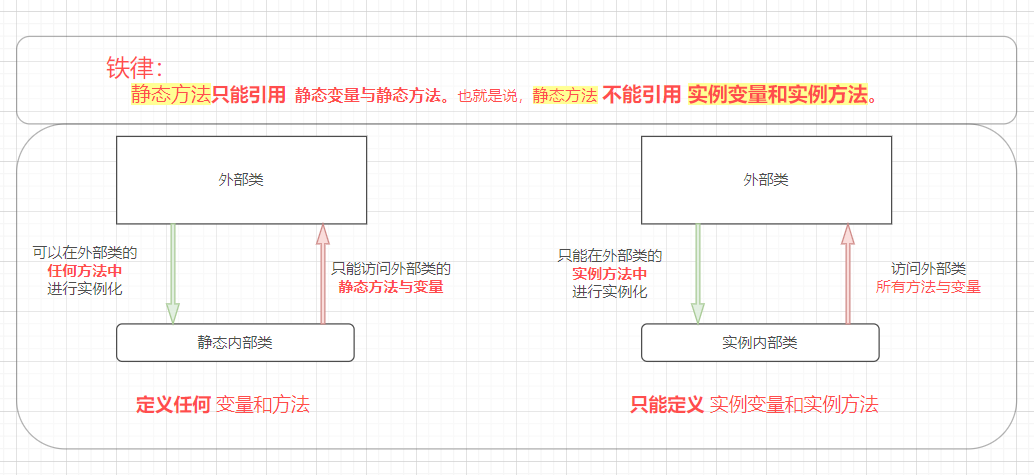
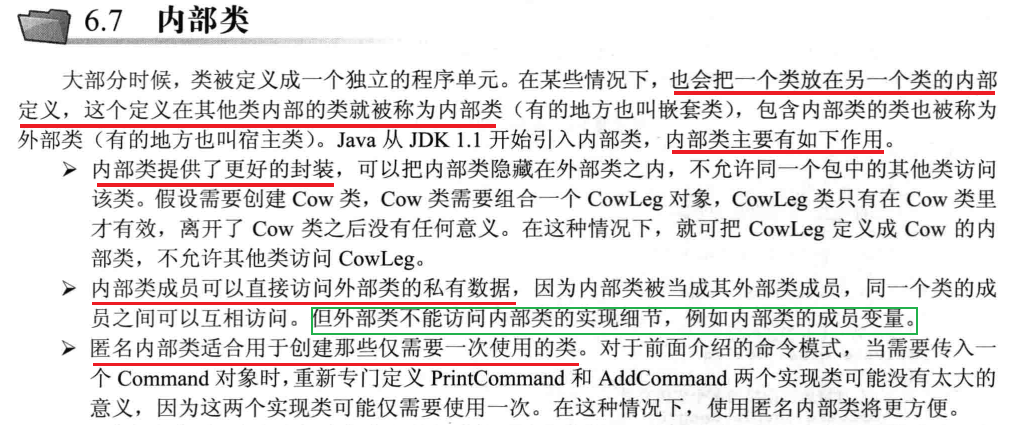
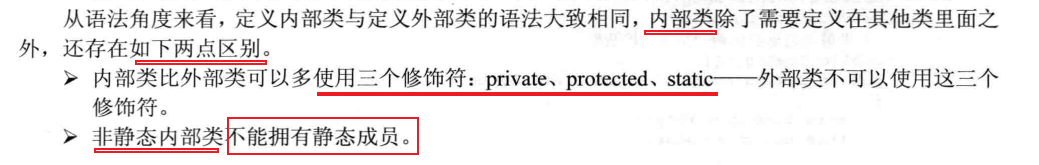
**Java 内部类**

## 思维导图

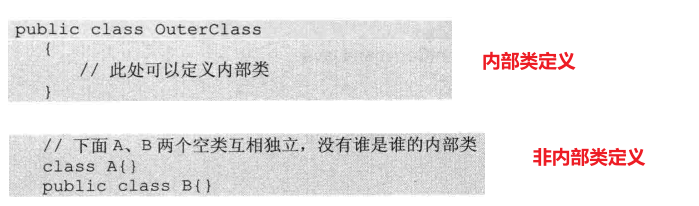


## 内部类概述



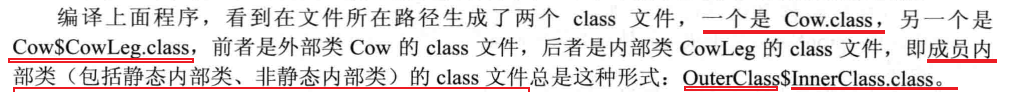


### 定义

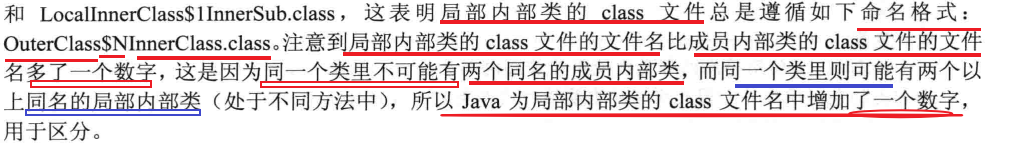


### 内部类Class命名格式

成员内部类

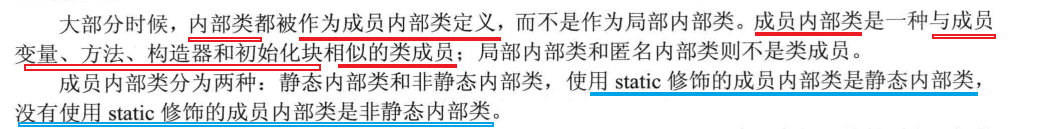


局部内部类



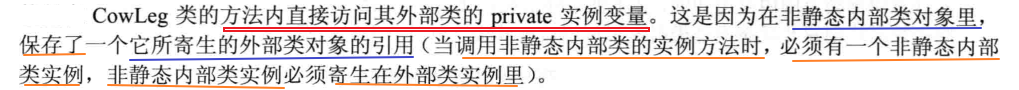
## 非静态内部类

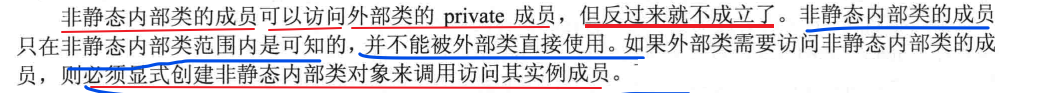
### 概述



### 使用内部类方法

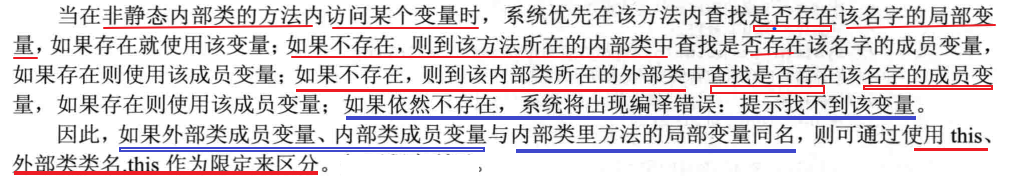
外部类方法调用内部类变量必须通过 **创建内部类实例方式调用**

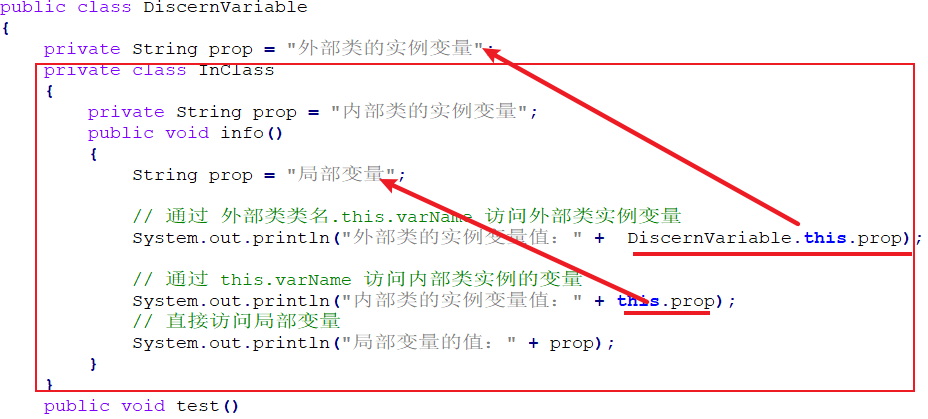




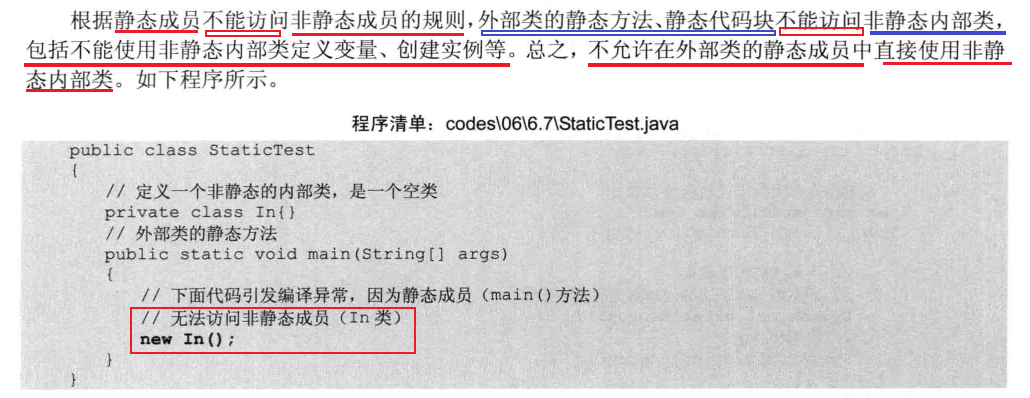
### 同名变量处理方式

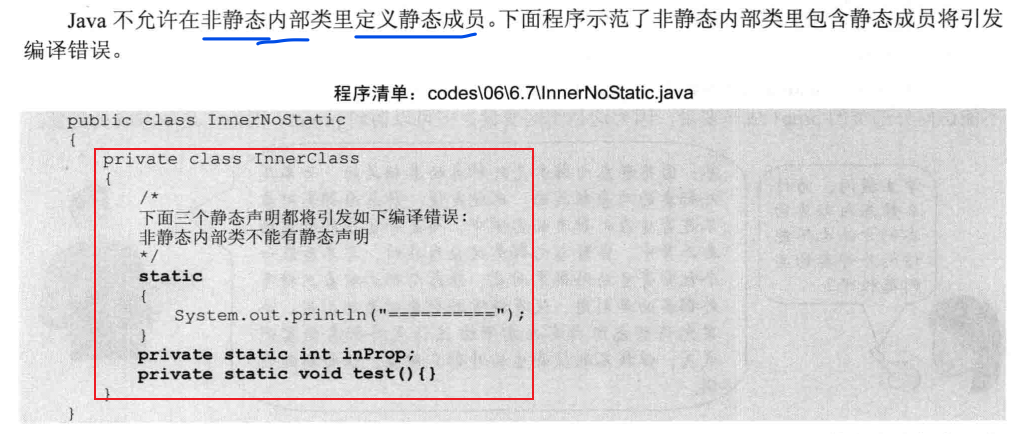
内部类获取变量的过程及同名变量的区分方式



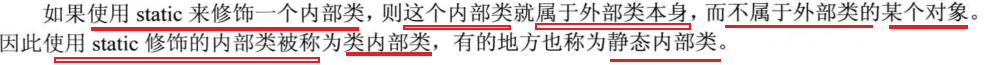


### 注意事项-重要

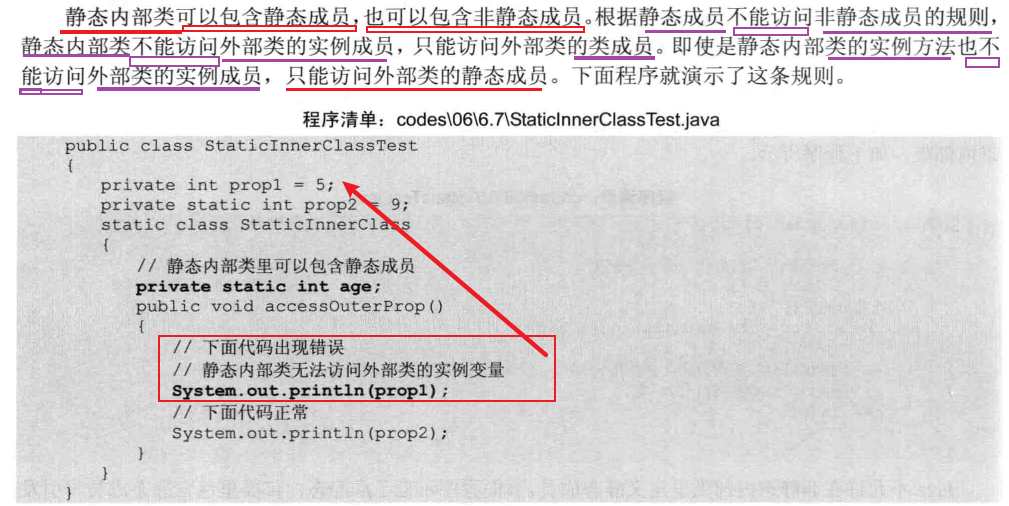




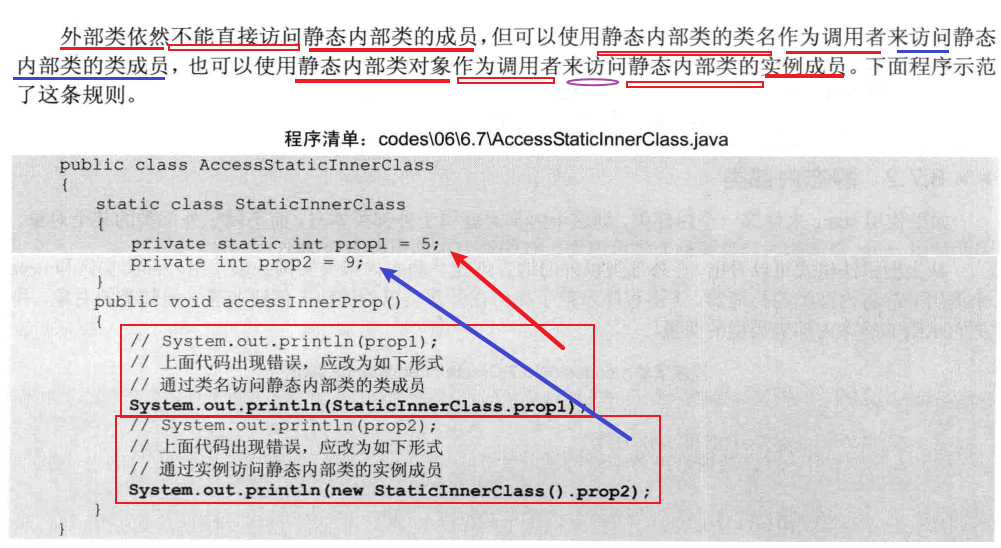
## 静态内部类



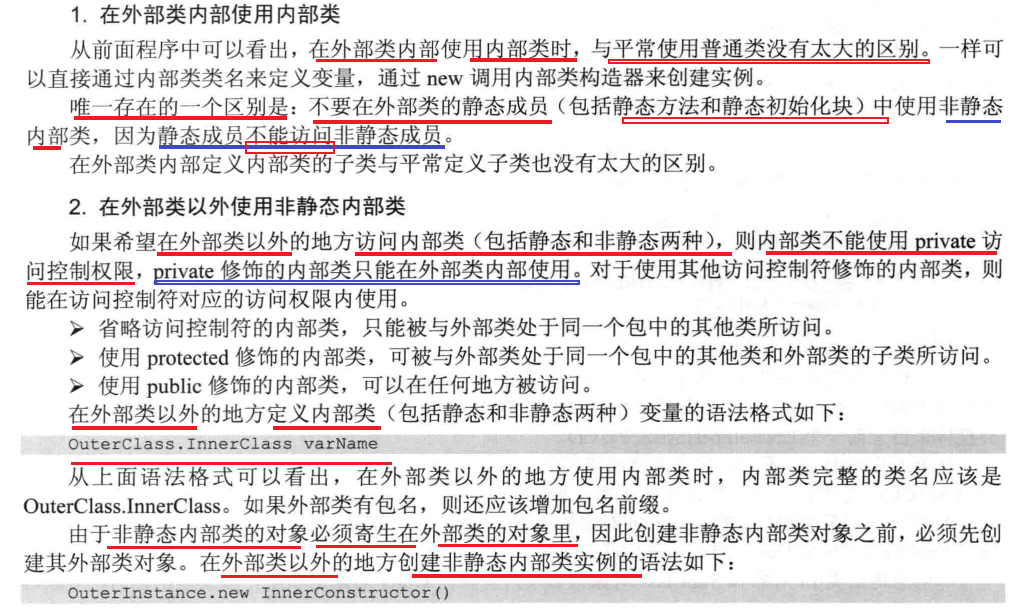
### 只能获取外部类静态成员

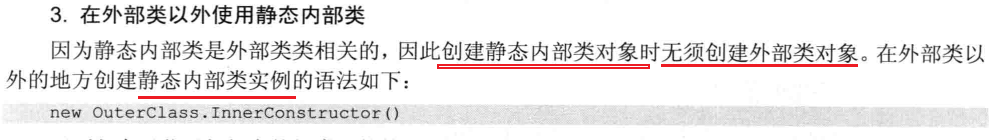


### 只能以类名方式获取内部类变量

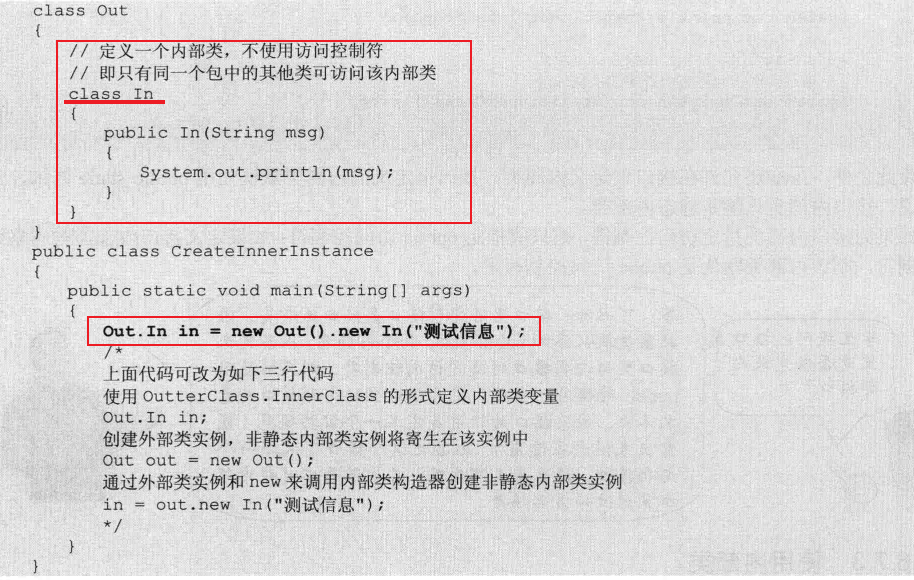


## 内部类使用

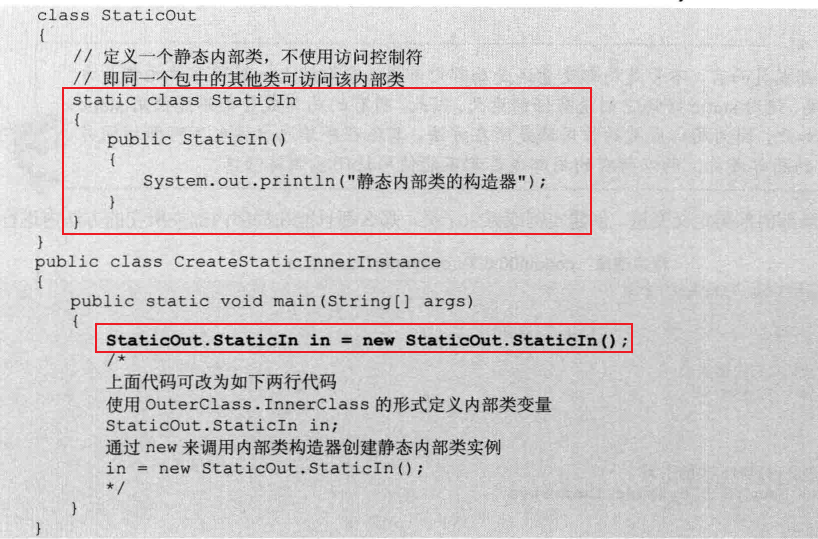




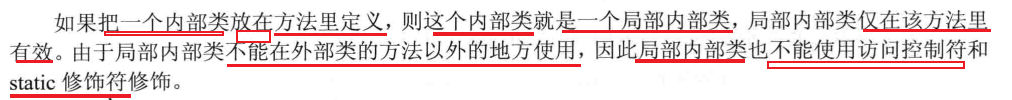
### 非静态内部类创建实例

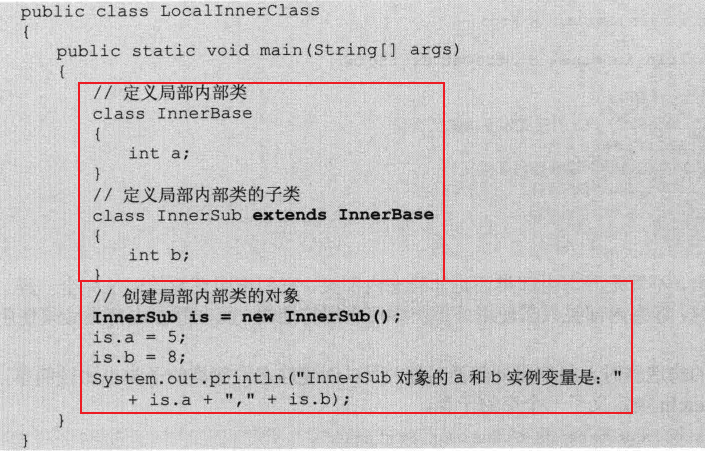


### 静态内部类创建实例

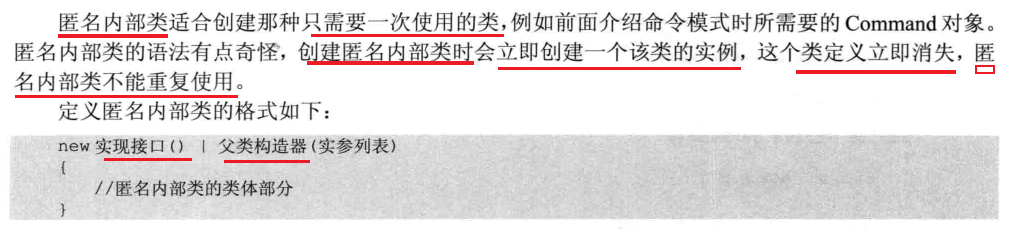


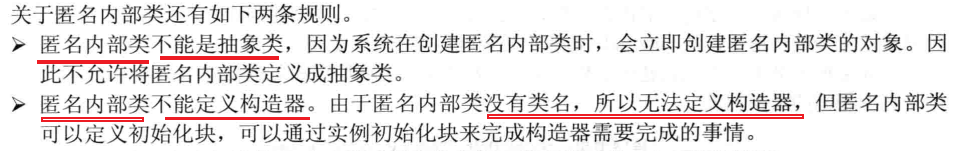
## 局部内部类



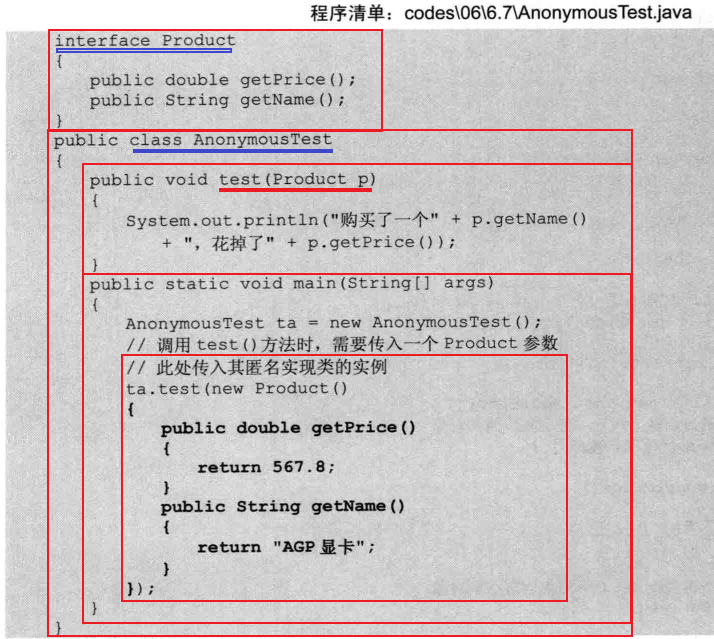


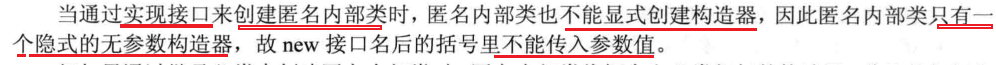
## 匿名内部类



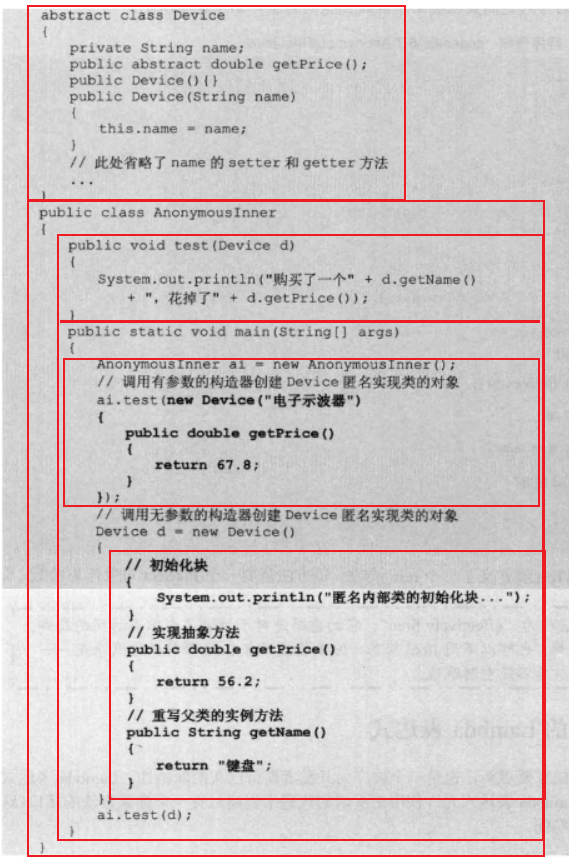


### 通过接口创建匿名内部类





### 通过继承父类创建匿名内部类



## 内容总结

1. 内部类的存在主要是为了进行更好的封装，如果不用内部类也可实现，只是对外会暴露一些信息而已。
2. 内部类是在 一个类中定义的，其作用相当于 类中的变量，可以使用相关修饰符，入private、pullic、static 来定义内部类。
3. 内部类的好处是可以方便的访问外部类的任何变量，包括私有变量，当然要符合 **静态成员 不能访问 非静态成员的规则。**
4. 匿名内部类，就没有名字的内部类，因为定义名字麻烦，而且该类只用一次，所以就使用匿名类的方式创建对象进行使用。实现接口或实现父类的类必须是要修名字的，要想定义出没有名字的类，**必须以此方式实现 接口类型 变量名=new 接口(){}，或 父类 变量名= new 父类(){}**。
5. 匿名类的的父类是抽象方法，否则使用匿名内部类就没有什么意义了。

## 使用总结-重要

前提：

**静态成员 不能访问 非静态成员的规则。**

内部类本质

内部类本质 相当于一个 外部类的一个变量[可以是静态的属于类的一部分，是非

静态的 属于对象的一部分]。不论内部类变量、方法 是否是 私有的，对于外部类而言

都可以访问(内部类本身就是外部类的一部分)。

**对象层面-属于外部类的某个对象**

**非静态内部类-可以访问任何外部类的变量和方法**

**1.不能定义任何static变量和方法。**但可以引用外部类的任何方法和变量， 包括static变量和方法。

2.**只能在外部类的实例方法中 进行new实例化。**

**不能定义static原因：**

非静态内部类不能定义任何静态变量和方法的原因是**因为它与外部类的**

**实例相关联**，而静态变量和方法是与类本身相关联的。如果非静态内部类定义了

静态变量和方法，那么它将**无法直接访问**外部类实例的方法和字段，这与非静态

内部类的设计初衷相违背。

**类层面-属于外部类本身**

**静态内部类-可以在外部类任何地方进行实例化**

1. 可以定义任何变量和方法(包括非静态变量变量和方法)。但**只能引用**

【静态变量和静态方法】。

1. 可以在外部类的 任何方法中 实例化对象。