# 什么是

乍一看，好些创建方式，挺复杂的吧？首先内部类是个啥东西？   
定义：**创建在一个类内部的类型**。根据创建位置的不同，分为**成员的、方法的、匿名的。接口中的内部类叫做接口内部类。**  
理解：在类的内部创建，那就是类的一部分，跟属性、方法同级。这也就解释了**为何能访问外部私有成员，我是你的一部分，我当然可以访问了**。

内部类（nested classes），面向对象程序设计中，**可以在一个类的内部定义另一个类**。嵌套类分为两种，即**静态嵌套类和非静态嵌套类**。静态嵌套类使用很少，最重要的是**非静态嵌套类，也即是被称作为内部类(inner)**。内部类是JAVA语言的主要附加部分。内部类几乎可以处于一个类内部任何位置，可以与实例变量处于同一级，或处于方法之内，甚至是一个[表达式](http://baike.baidu.com/view/420676.htm)的一部分。

　　内部类是[JAVA语言](http://baike.baidu.com/view/229611.htm)的主要附加部分。嵌套类从JDK1.1开始引入。其中inner类又可分为三种：

　　　　其一、在一个类（外部类）中直接定义的内部类；

　　　　其二、在一个方法（外部类的方法）中定义的内部类;

　　　　其三、匿名内部类。

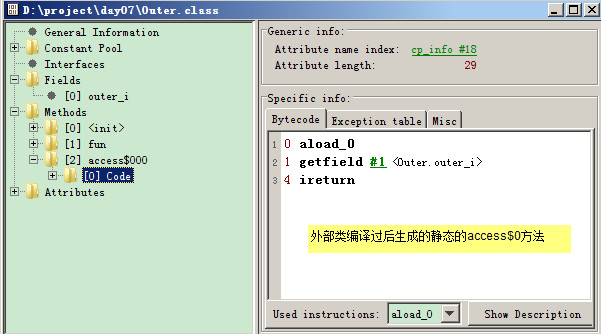
## 问题的引出<理解内部类>：

那问题来了，我是你的一部分，别人继承了**外部类**，会不会也把内部类也继承过去呢？这得从内部类设计的初衷与面向对象来探讨了。   
比如我们用Body类来描述人体。如果是描述人类的话，会去描述人类具有的属性跟功能。   
那现在我们描述人体，人体内部有**心肝脾胃肾**，这些再用属性来描述就不合适了吧？那怎么办呢？我们可以用内部类去描述。   
所以，无论内部类是公开的还是私有的，都不会被继承，因为他不是属性，也不是方法。而是一个**内部事务的描述，我们称之为内部类**。   
内部类可以更好地对内部事务进行封装，看例子：   
  
身体类中，有属于身体的各个器官，**各个器官有自己的功能与属性，于是我们把它封装成一个内部类去单独描述。**用来描述器官的**内部类是身体的一部分，所以可以去自由的访问身体的资源(属性与功能)，各个器官与身体相互协调完成运作。**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | public class Body {  　　private class Heart { // 心脏  　　　　public void bloodSupply() {  　　　　// 供血...        　　}       }  　　private class Hepar { // 肝脏  　　　　public void Metabolism() {  　　　　// 代谢...        　　}       }  　　private class Spleen { // 脾脏  　　　　public void storageBlood() {  　　　　// 储血...             }        }  　　private class Stomach { //胃部  　　　　public void digest() {  　　　　// 消化...             }        }  　　private class Kidney { // 肾脏  　　　　public void dischargePoison() {  　　　　// 排毒...             }        }  } |

## 问题的引出<内部类访问外部类成员方式>：

因为可以访问外部私有成员，那问题也就诞生了，他是怎么访问的呢？

1：对于成员内部类来说，他会持有一份外部类当前对象的引用，Outer.this。   
这样就可以调用外部类可见的方法跟成员变量，调用方法是通过持有的外部类引用去调用的。Outer.this.fun();   
那对于private成员是如何去访问的呢？是通过编译器在外部类中生成的静态的access$0()方法来访问的。Outer.this.access$0(Outer);   


  
2：对于方法内部类（匿名内部类）来说，因为内部类要访问所在方法中的局部变量，这时候用持有的外部类当前对象引用还能调用的到吗？   
调不到了，那Java语言的设计者是这么来解决这个问题的：将局部变量复制一份给内部类使用，怎么复制的？在内部类初始化的时候通过构造方法传值的方式。   
这样，内部类中会有一份复制的private修饰的成员变量。这样我就能访问了。但问题又来了，比如：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public void fun() {  　　int i = 1;  　　class Inner{  　　　　// int i = 1; 由编译器生成  　　　　publicvoid print() { i++ }      　}     　 System.out.println("i=" + i);  　　// 还是会输出i=1，我们明明在内部类方法中对此变量进行++了啊。抱歉，您++的是被复制的另一份。  } |

## 问题的引出<保持两个不同变量的一致性>：

好，问题就这么无情的出现了，怎么解决？Java又说了，要不然给局部变量加上final吧，这样就会保持值得一致性了。   
ok，问题解决，这就是为什么方法内部类访问的局部变量必须被final修饰的终极原因（为了约束两个不同变量的一致性）。设计问题，挺无奈的解决方案。   
  
来看看另一种说法：   
在Java中，方法的局部变量位于栈上，对象位于堆上。   
因为局部变量的范围被限制在该方法内，当一个方法结束时，栈结构被删除，该变量消失。   
但是，定义在这个类中的内部类对象仍然存活在堆上，所以内部类对象不能使用局部变量。除非这些局部变量被标识为最终的。    
  
  
这种说法是片面的，因为根本原因是内部类对象无法访问局部变量，才会去复制一份。   
为了保证两个变量的一致性，才去使用final关键字修饰局部变量。而不是因为栈生命周期与堆生命周期不一致的问题。三、最后来说说final关键字：

用final关键字修饰对象变量，只是不允许这个对象引用再指向其他的对象，但是这个引用所指向的对象的内容是可以改变的   
比如一个经典的例子：

|  |  |
| --- | --- |
| 1  2  3 | final StringBuffer sb = new StringBuffer("HelloWorld");  sb = new StringBuffer("Hello"); // 编译失败，不能修改sb引用的指向  sb.append("China"); // sb指向的对象可以被修改 |

其实从final的位置就看出来了，他是在修饰引用类型变量sb，而不是在修饰堆中的实例对象（new StringBuffer("HelloWorld");）

# 为什么要用

为什么需要内部类？

　　　　⒈ 内部类对象可以访问创建它的对象的实现，包括私有数据；

　　　　⒉ 内部类不为同一包的其他类所见，具有很好的封装性；

　　　　⒊ 使用内部类可以很方便的编写[事件驱动](http://baike.baidu.com/view/536048.htm)程序；

　　　　⒋ 匿名内部类可以方便的**定义运行时**[**回调**](http://baike.baidu.com/view/1305637.htm)；

　　　　5.内部类可以**方便的定义**

**每个内部类都能独立地继承自一个（接口的）实现**，所以无论外围类是否已经继承了某个（接口的）实现，对于内部类都没有影响。如果没有内部类提供的可以继承多个具体的或抽象的类的能力，一些设计与编程问题就很难解决。从这个角度看，内部类使得多重继承的解决方案变得完整。接口解决了部分问题，而**内部类有效地实现了“多重继承”。**

# 优点

# 缺点