封装性：

加上封装性，并不是不让访问，而是让别人按照自己的要求进行访问。而这个要求就是方法接口。

构造方法与匿名对象：

必须要有构造方法才能完成类的实例化，所以如果没有定义构造方法，每个类都有一个自定义的无参构造方法。

类名称：确定日后要调用的方法和属性的范围，

变量名：用于区分栈内存的唯一标识，可以指向堆内存。

匿名对象：只开辟堆内存空间但是没有一个显示的栈内存的名称指向它。所以它只能被使用一次，然后就会被作为垃圾。但是命名对象和匿名对象，自己按情况使用。

This对象：***（！！！！！！！这里我需要提醒一下，首先，一个类的方法是用来处理数据的，而这个数据包括传进的参数和类的属性，所以，如果一个类有多个实例化对象，他们是有各自的堆内存地址，但是方法调用都是使用的方法区中的Class对象的方法模板，那么当我们多个对象进行调用方法时，就需要进行区分，进行处理各自的数据）***

首先对象的创建过程：

在这里有必要总结一下对象的创建过程。请考虑一个名为Dog 的类：

(1) 类型为Dog 的一个对象首次创建时，或者Dog 类的static 方法／static 字段首次访问时，Java 解释器

必须找到Dog.class（在事先设好的类路径里搜索）。

(2) 找到Dog.class 后（它会创建一个Class 对象，这将在后面学到），它的所有static 初始化模块都会运

行。因此，static 初始化仅发生一次——在Class 对象首次载入的时候。

(3) 创建一个new Dog()时，Dog 对象的构建进程首先会在内存堆（Heap）里为一个Dog 对象分配足够多的存

储空间。

(4) 这种存储空间会清为零，将Dog 中的所有基本类型设为它们的默认值（零用于数字，以及boolean 和

char 的等价设定）。

(5) 进行字段定义时发生的所有初始化都会执行。

(6) 执行构建器。

 到(6)的时候， 这个对象其实已经在内存中存在了。所以this已经有引用了

基于此：

有说法1：

***内幕是：java对象的建立其实是在构造函数开始执行之前，就是说构造函数执行的时候，被构造的对象其实已经存在了，所以构造函数中可以用this引用当前被构造的对象进行处理（比如 this.name=“Jack”）。***

***其实构造方法不是在创建对象，而是在不停的复制对象到不同的地址中去。另外调用构造方法的其实还是这个this。***

***也不能说是复制，产生新对象并且放在内存里是虚拟机的工作，构造方法的功能其实是初始化这个新对象。  
“调用构造方法的其实还是这个this”我觉得这个说法好。***

***另有说法：***

**this指针是指向对象自己的内存，它存在于对象里，对象生存它就生存，对象消亡它就消亡。就好比c里的数组 int a[], a既是数组的名字（是数组本身），也是指向数组的首地址的指针常量，两者分不开。this也是一样，它既是对象本身，也是指向对象本身的指针。  
成员方法会被编译为\_\_thiscall的方式，该方式就是自动在参数里加一个this指针参数，这个this参数的值怎么来？就是通过上面的this指针来的。（总之this不是外部的一个引用，否则就不符合垃圾回收的准则。还有就是在还没有进行初始化前，已经有了堆中对象的实体，而这个this应当是在开辟出来堆内存后，调用构造函数之前，就将其的值给定了，super的原理类似）**

[***http://c.biancheng.net/cpp/biancheng/view/201.html***](http://c.biancheng.net/cpp/biancheng/view/201.html)

* this 是 const 指针，它的值是不能被修改的，一切企图修改该指针的操作，如赋值、递增、递减等都是不允许的。
* this 只能在成员函数内部使用，用在其他地方没有意义，也是非法的。
* 只有当对象被创建后 this 才有意义，因此不能在 static 成员函数中使用（后续会讲到 static 成员）。

**this 到底是什么**

this 实际上是成员函数的一个形参，在调用成员函数时将对象的地址作为实参传递给 this。不过 this 这个形参是隐式的，它并不出现在代码中，而是在编译阶段由编译器默默地将它添加到参数列表中。  
  
this 作为隐式形参，本质上是成员函数的局部变量，所以只能用在成员函数的内部，并且只有在通过对象调用成员函数时才给 this 赋值。

**构造方法也能进行互调用：**

**通过this.(参数，……)的方式进行构造函数的调用。但是这条语句必须放到第一行语句才能正常执行。**