一.面向对象思想

面向对象思想就是在计算机程序设计过程中，参照现实中的事物，将事物的属性特征、行为特征抽象出来，使用计算机语言将事物描述成一个类，通过由类实例化出来的各个对象的属性、行为来实现功能。 它区别于面向过程思想，强调的是通过调用对象的属性、行为来实现功能，而不是自己一步一步的去实现功能。在面向对象的语言中，包含了三大基本特征，即**封装**、**继承**和**多态**。

1.类的定义

类是对现实世界中一类 "事物" 的属性和行为的抽象，是一种抽象的引用数据类型。定义一个类的语法格式如下：

public class 类名 {

// 成员变量：对应事物的属性

// 成员方法：对应事物的行为

}

2.对象的创建与使用

通常情况下，一个类并不能直接使用，需要根据类创建一个对象才能使用。为类创建对象的过程，叫做实例化一个类，该对象属于类的一个具体实例，具有类中的所有属性和行为。即我们可以使用该对象访问类中的成员变量和成员方法，以此来获取该对象的属性和行为。

(1).对象的创建格式：

类名称 对象名 = new 类名();

(2).使用对象访问类中的成员：

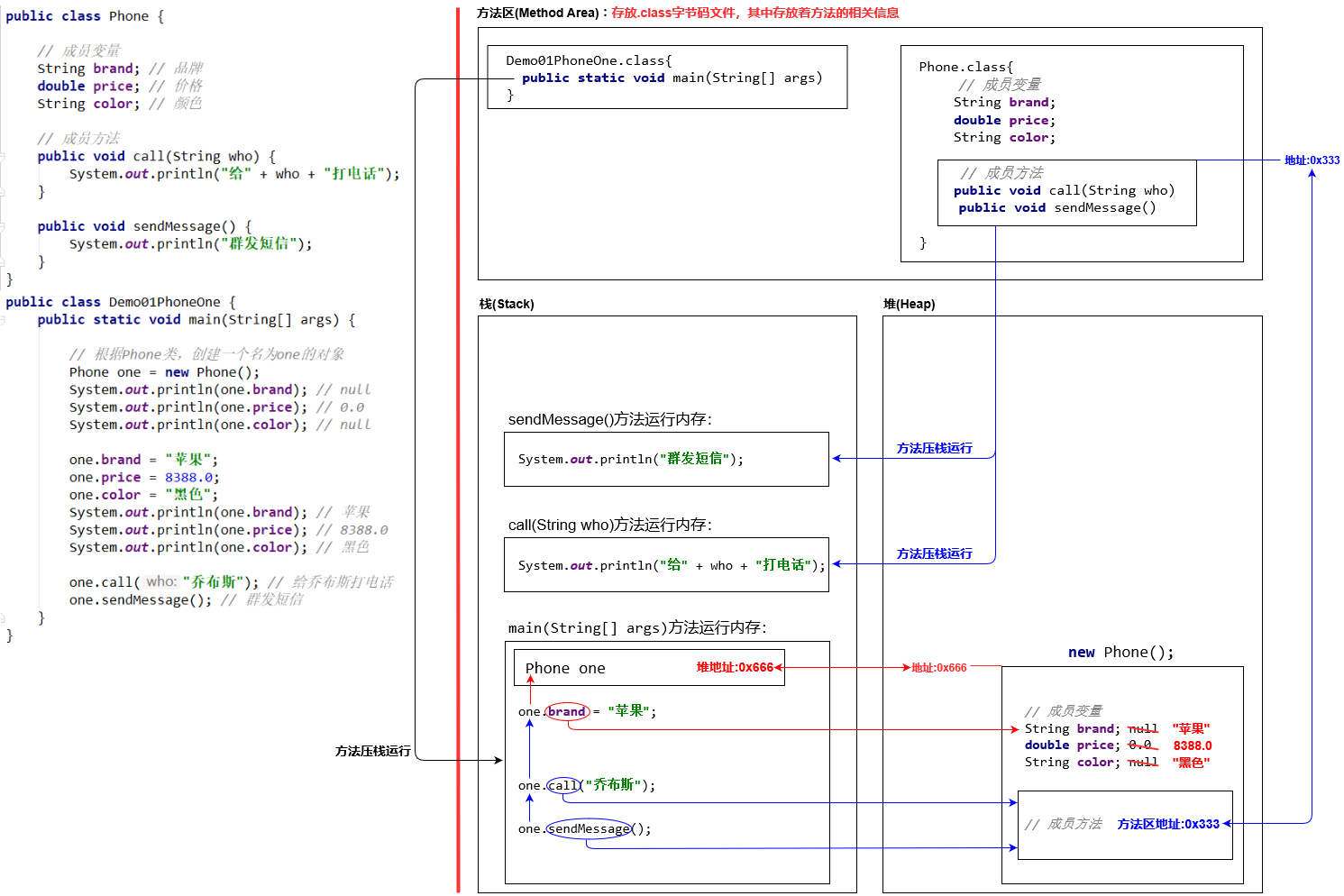
对象名.成员变量名;

对象名.成员方法名();

#.类与对象的理解：类是对现实生活中 "一类事物" 的抽象，是一个抽象的概念，并不能具体指到现实生活中哪个物体上去。比如：学生类，并不能具体到哪个学生上，只是将学生这一个群体归为一个类。而对象则是 "这一类事物" 所对应的具体物体，是一个实际存在的物体，能具体指到现实生活中哪个物体上去。比如：学生张三，能具体到这个学生对象，是现实生活中实际存在的一个物体。（类是虚无抽象的，对象是实际存在的！）

3.对象原理内存图

(1).一个对象的内存图



①.类文件通过编译生成.class字节码文件，JVM运行字节码文件时，将字节码文件存放在方法区。如图所示，成员方法在方法区内有自己的地址值。

②.方法必须要在栈中才能运行，所以main方法首先压栈运行，此时会在栈内存的底部开辟一个main方法的运行内存空间。所有方法都是以“压栈”的方式进入栈内存空间运行，并且方法运行结束时会立即释放自己的栈内存运行空间。

③.在方法中创建局部变量时，JVM会在main方法运行区中开辟一块空间用来存储变量。

Phone one 相当于以Phone类作为数据类型，创建一个名为one的变量，称为引用类型变量one。首先会在main方法运行区内，为该引用类型变量开辟一块空间。然后new Phone()会使JVM在堆内存中开辟空间用来存储新new对象的相关数据信息，并将对象的堆内存地址返回给引用类型变量one存储。则Phone one = new Phone(); 为将新new对象的堆内存地址赋值给在栈内存中的引用类型变量one。

④.one.brand = "苹果"; 在栈空间中根据对象名找到引用类型变量的内存区，再根据其中存放的地址在堆内存中找到对象，对相应的成员变量进行赋值操作。

⑤.对象在堆内存中的数据信息有：成员变量、成员方法。其中在成员方法的独立内存区域，保存的是成员方法的方法区地址。

one.call("乔布斯"); 在栈空间中根据对象名找到引用类型变量的内存区，再根据其中存放的地址在堆内存中找到对象。再根据成员方法的方法区地址在方法区内存中找到相应的成员方法，通知其压栈运行。call方法运行结束后会立马释放其栈内存运行空间，紧接着sendMessage方法压栈运行，运行结束后释放其栈内存运行空间。此时main方法运行结束，同样会释放其栈内存运行空间。

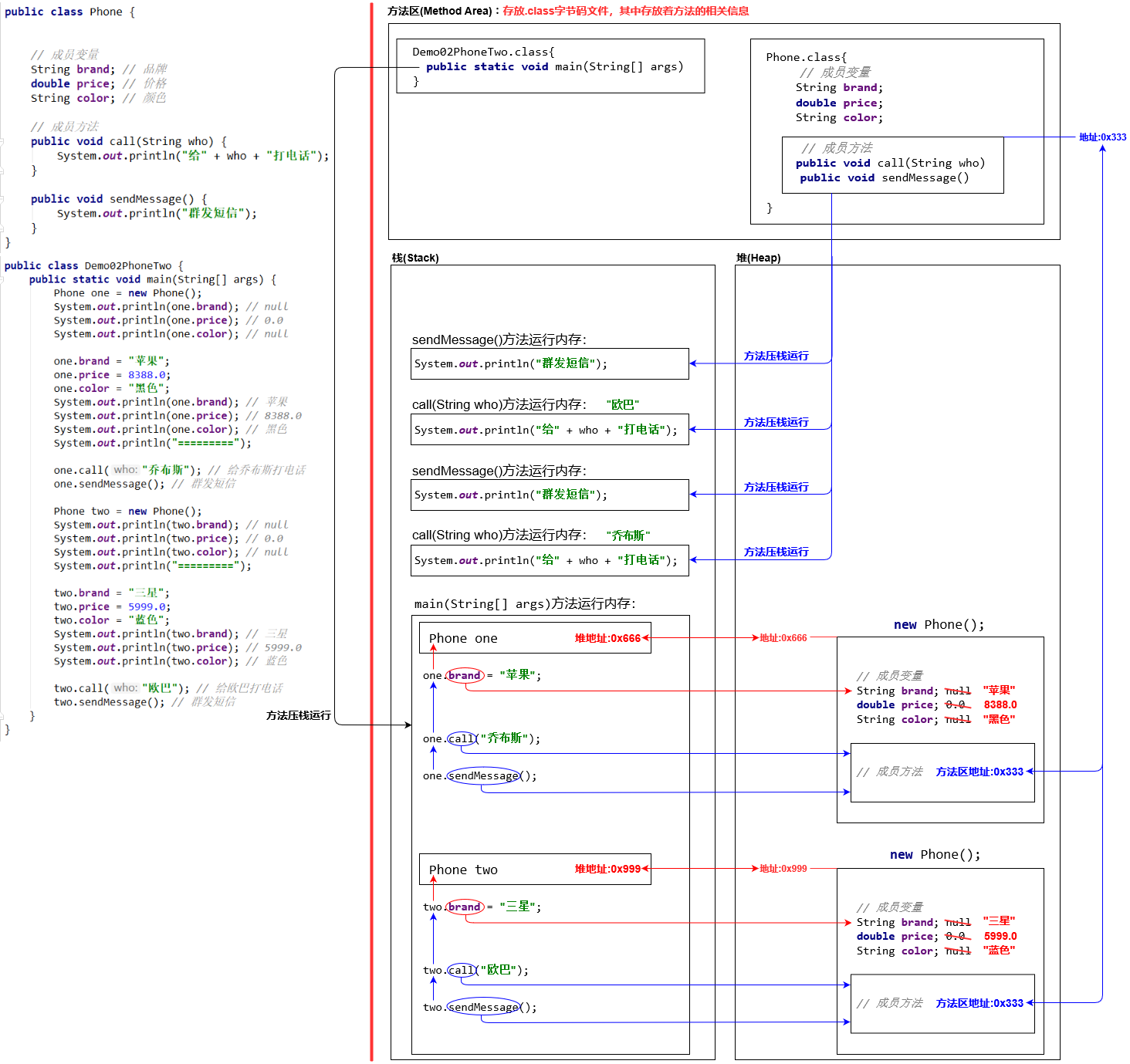
#.注意：

a.引用类型变量one的使用与其他普通变量没有差别，只不过其数据类型为"Phone类"，其中存储的是对象的堆地址。每次使用对象名访问成员变量、成员方法时，都要根据变量one中的地址值，去堆内存中寻址查找数据，相当于使用此变量去引用堆内存中的数据，故称之为引用类型变量，类也因此称为引用数据类型。

b.从数据类型的角度理解类：(类是引用数据类型的一种，与C语言中的结构体类型很类似，只不过多了个成员方法)

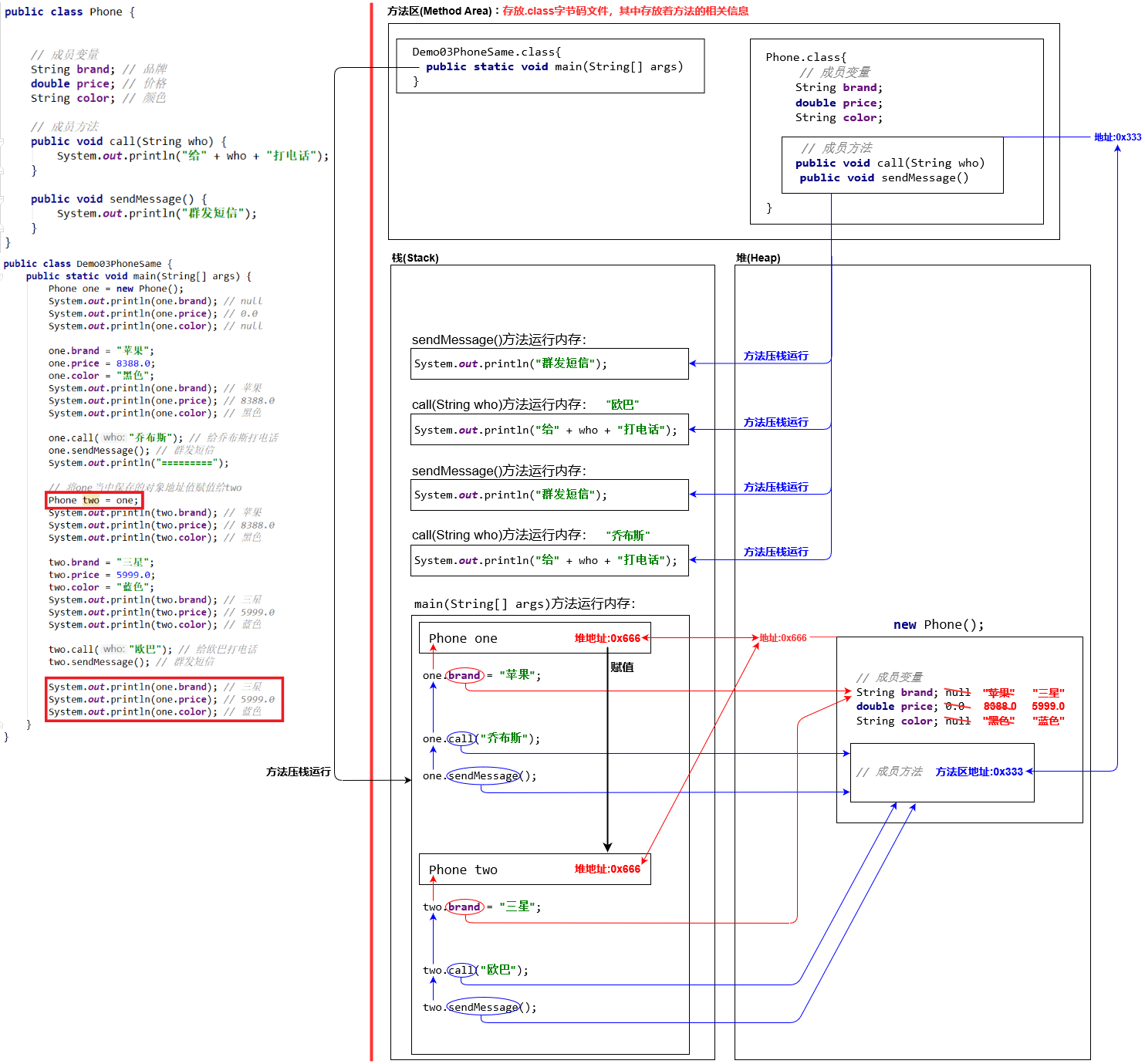
Phone类相当于一种自己定义的数据类型，并且属于引用数据类型，是一种抽象的数据类型。由Phone类new出来的对象one，就是"Phone类"这个数据类型的一个数据。

(2).两个对象的内存图



对象调用方法时，根据堆内存中成员方法的方法区地址，去方法区的字节码文件中寻找相应的成员方法信息。这样哪怕是new出来多个对象，就不用在每个对象的堆内存区内都保存一份方法信息，只在方法区内保存一份方法信息，节约了堆内存空间。

(3).两个引用类型变量指向同一个对象的内存图



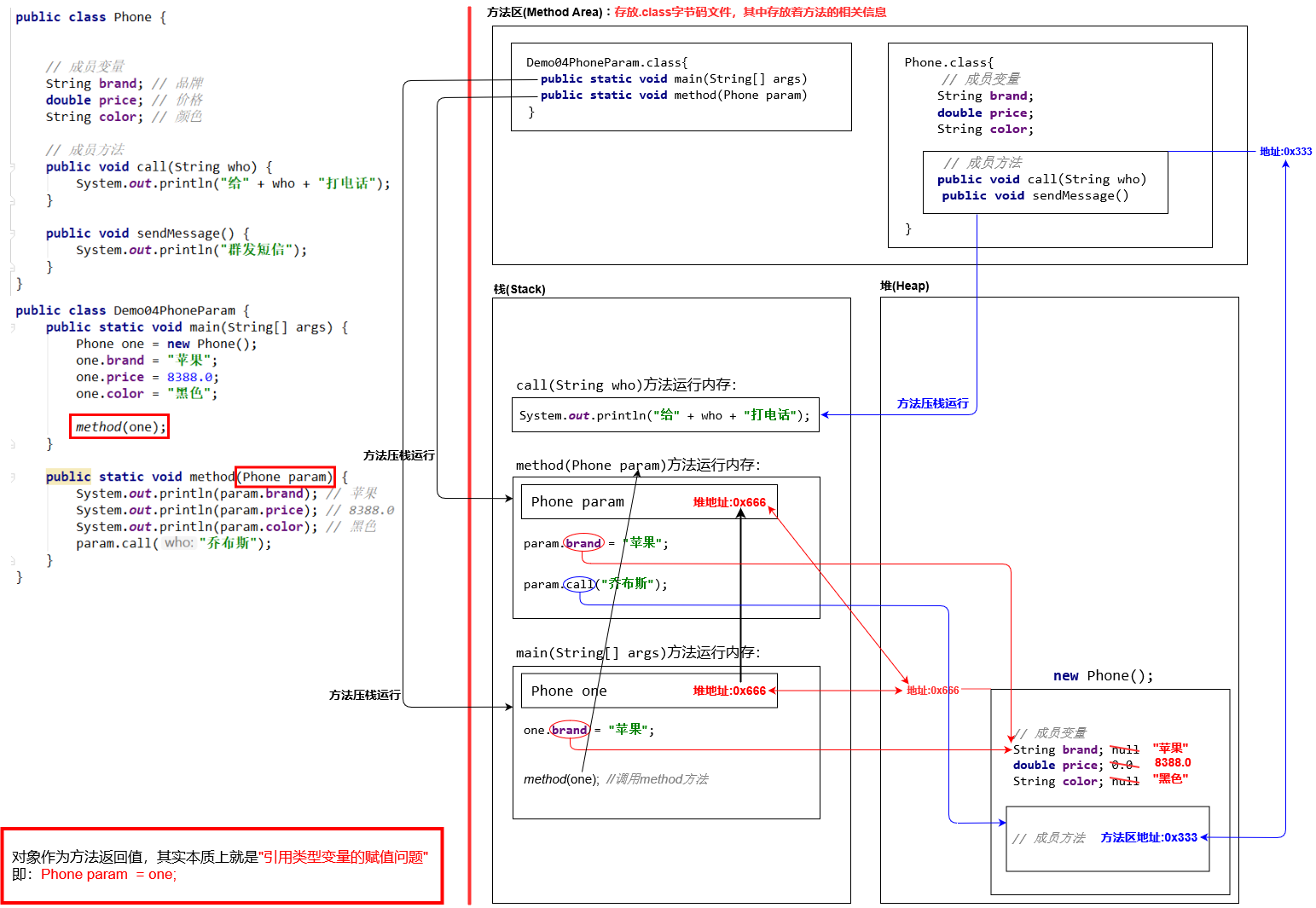
将对象的堆内存地址赋值给另一个引用类型变量，使两个引用类型变量都指向堆内存中的同一个对象。此时对两个不同对象的操作相当于对同一个对象的操作。

(4).对象作为方法的参数和返回值

在以前的学习中，方法的参数和返回值，使用的都是基本数据类型和数组。作为引用数据类型的对象也可作为方法的参数和返回值。对象作为方法的参数和返回值时，其实是以其引用类型变量来作为方法的参数和返回值。即方法的实参、形参和返回值，都是以 "类" 作为数据类型的引用类型变量，传递与返回的都是对象的堆内存地址。

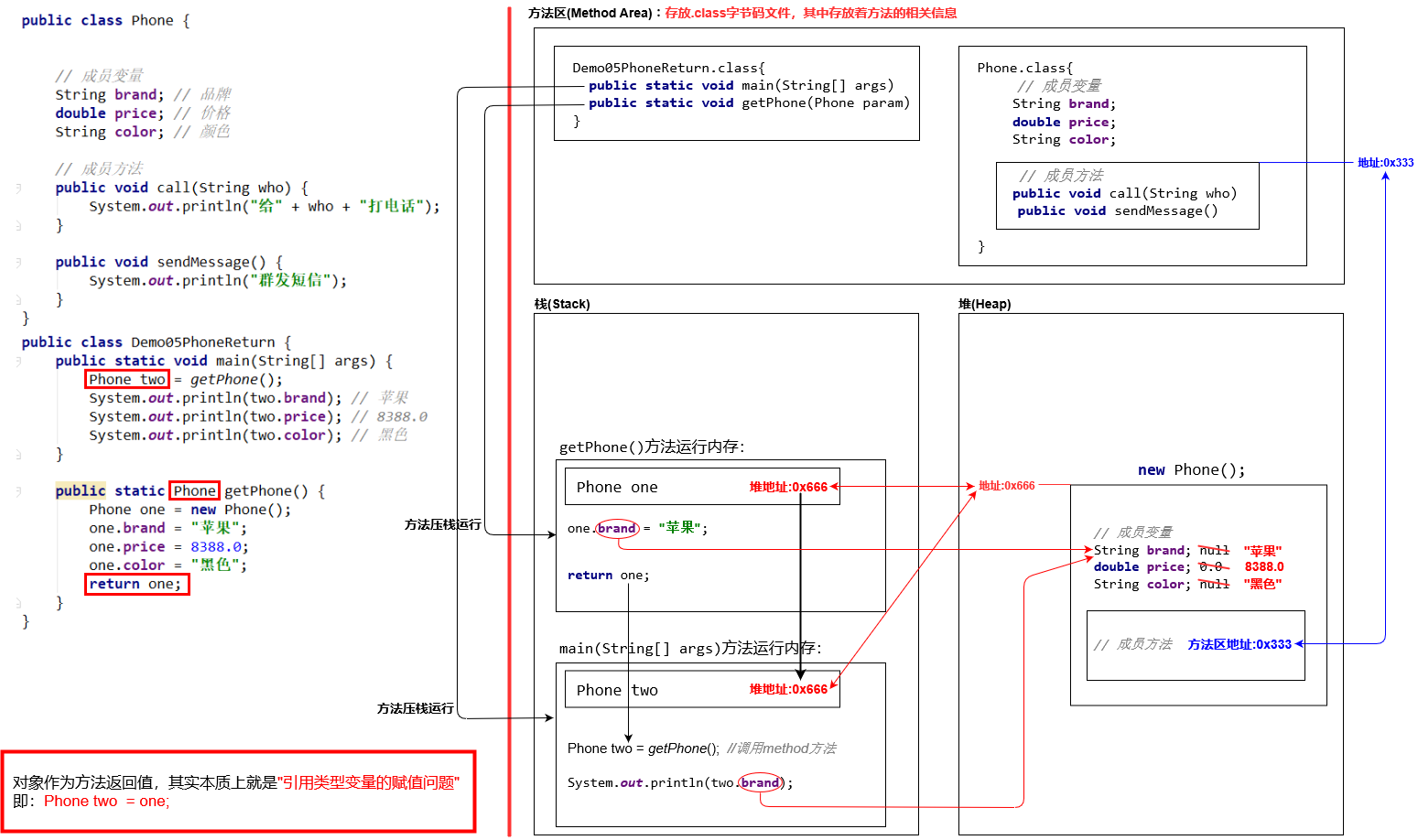
①.对象作为方法参数\_传递地址

对象作为方法参数进行传递，是以引用类型变量来作为方法的实参和形参，因此传递的是对象的堆内存地址。即实参、形参这两个引用类型变量此时指向同一个对象，因此在方法体内对形参对象的操作，就相当于对实参对象的操作。而对于基本数据类型的实参变量和形参变量，在方法体内改变形参变量并不会影响到实参变量，因为二者是不同的内存区域。



②.对象作为方法返回值\_返回地址

对象作为方法返回值，是以引用类型变量来作为方法的返回值，因此返回的是对象的堆内存地址。同时方法调用处要使用相同数据类型的引用类型变量来接收地址值，即返回值、接收值这两个引用类型变量此时指向同一个对象。

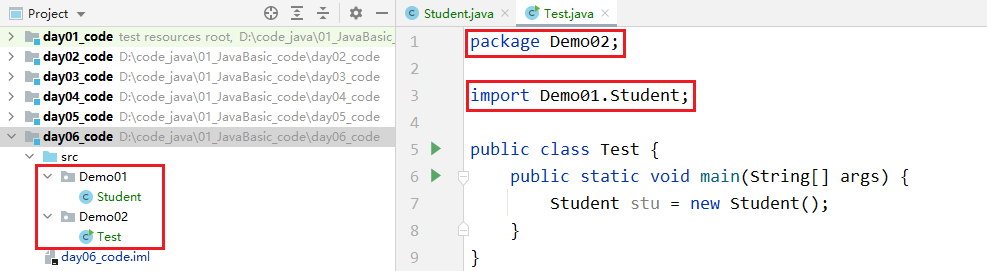


#.注意：

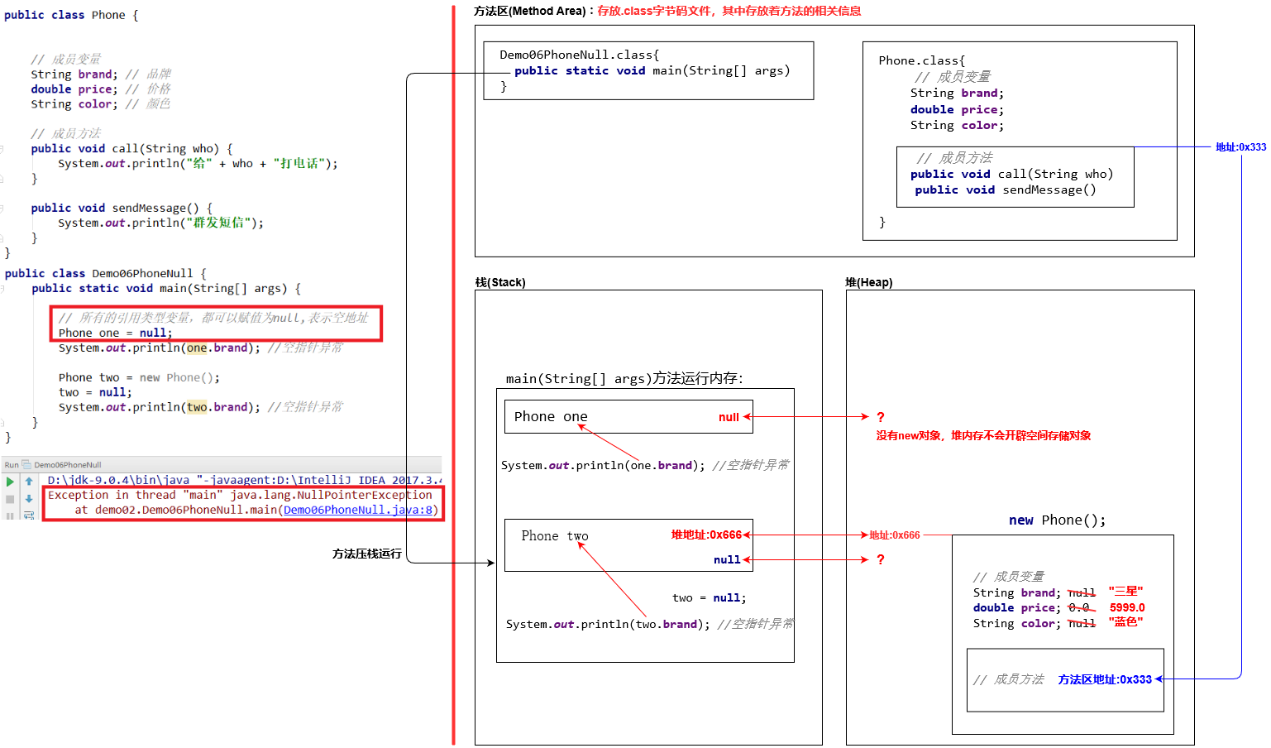
a.由于成员变量是存储在堆内存中的，所以成员变量在没有初始化赋值时，符合堆内存数据的默认值规则。

b.导入包的操作：

* package 语句：表明当前类文件是在哪个package下面（若包为多层的嵌套关系，则包名之间使用 . 相连）
* import 语句：若要在当前类文件中使用其他package里面自定义的类，则需要导入相应的类文件
  + 使用格式为：import 包名称.类名称; 其中包名称就是相对应package语句后面的包名
  + 对于和当前类属于同一个包的情况，可以省略导包语句不写（必须严格属于同一级package才无需导包，属于同一个父级package则不行！）

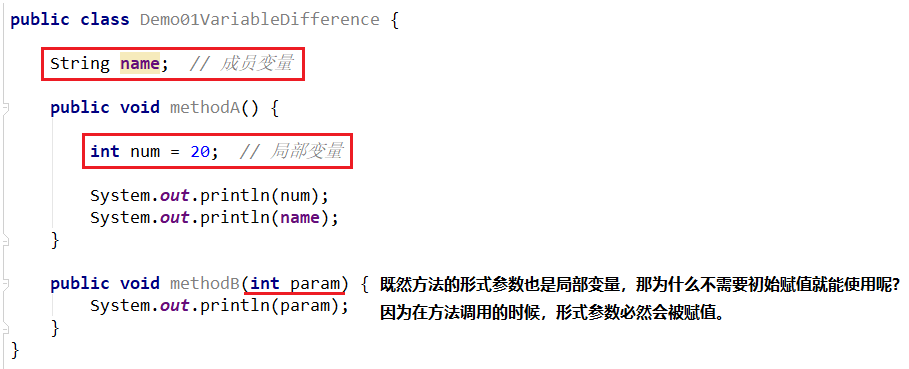


c.对象的空指针异常



一旦引用类型变量为null，此时我们再对对象数据进行操作，就会导致引用类型变量在堆内存空间中找寻不到地址，此时会抛出 java.lang.NullPointerException 空指针异常。

d.局部变量和成员变量的区别：



|  |  |  |
| --- | --- | --- |
|  | **局部变量** | **成员变量** |
| ①.定义的位置不一样 | 定义在方法中或者方法声明上(形式参数) | 定义在类中、方法外 |
| ②.作用范围不一样 | 仅限于方法内部使用 | 整个类中都可以使用 |
| ③.默认值不一样 | 没有默认值，必须先赋值再使用 | 有默认值，符合堆内存数据默认值规则 |
| ④.在内存中的存储位置不一样（了解） | 位于栈内存 | 位于堆内存 |
| ⑤.生命周期不一样（了解） | 随着方法进栈而诞生，随着方法出栈而消失 | 随着对象创建而诞生，随着对象被垃圾回收而消失 |

e.在成员方法中使用本类的成员变量：

在同一个类中，成员方法中的成员变量默认使用this关键字修饰，一般省略不写，如下图所示。想象一下，当show方法压栈运行时，this.name、this.age在内存图中是如何执行的：this关键字相当于在成员方法show()的栈运行区内，创建一个名为this的引用类型变量，这个this引用类型变量保存的是当前对象（当前调用该方法的对象）的堆内存地址。即谁调用该成员方法，方法中的this保存的就是谁的堆内存地址，则方法中的this引用就指向谁。因此对象在调用成员方法时，成员方法中的成员变量其实就是当前对象的成员变量。



二.封装

面向对象具有三大特征：封装、继承、多态。封装性在Java当中的体现：

* 方法就是一种封装：方法封装就是将一些细节信息隐藏起来，对于外界不可见。
* 关键字private也是一种封装：关键字private封装可以被认为是一个保护屏障，防止该类的代码和数据被其他类随意访问。要访问该类的数据，必须通过指定的方式进行访问。

1.使用private关键字进行封装操作

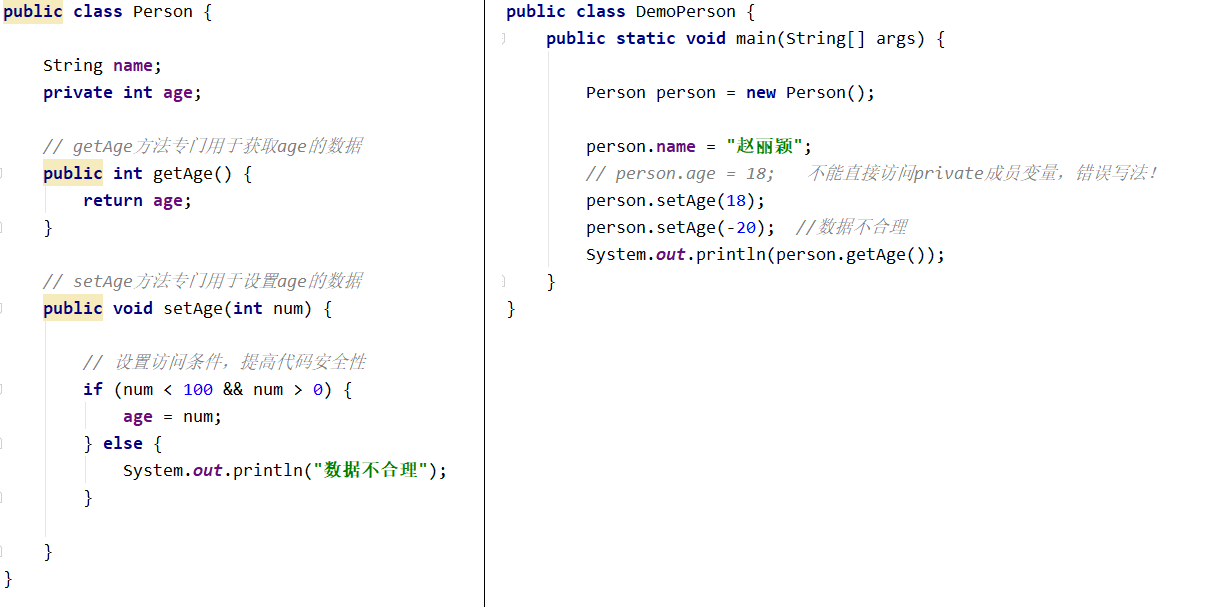
由上述代码可以看出，对于Person类的age属性，我们无法阻止不合理的数值被设置进来。这时我们就要对age属性进行封装保护的操作:

(1).使用private关键字来修饰成员变量

private是一个权限修饰符，代表私有权限。成员变量一旦使用了private进行修饰，就代表该成员变量为此类的一个私有属性。private成员变量在本类中仍然可以随意访问，但是超出本类范围之外就不能再直接访问了。即对象不能再像以前那样 "直接访问" 成员变量了(即 person.age = 18; 为错误写法)，要通过提供的公共方法来 "间接访问" private成员变量。

(2).为private关键字修饰的成员变量，提供一对公共的getXxx方法、setXxx 方法

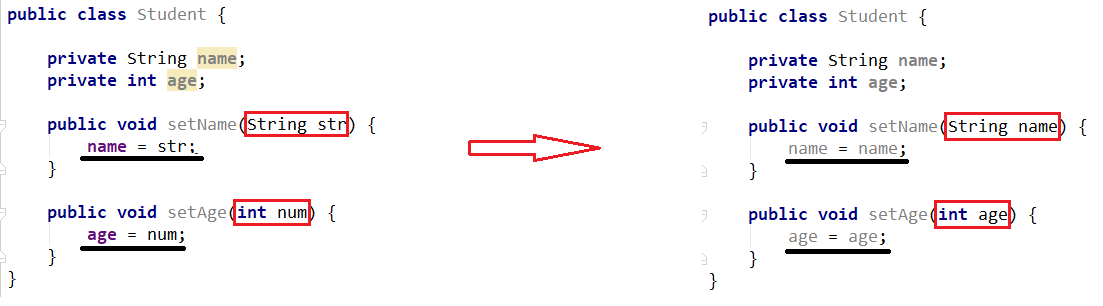
get、set方法与普通的成员方法一样，可以被对象直接访问。其中：get方法专门用于获取private成员变量的数据、set方法专门用于设置private成员变量的数据。对象通过访问公共的get、set方法，以此来间接访问private成员变量。



由此可见，private关键字封装就是：将某个属性隐藏封装起来变为私有属性，若需要访问某个私有属性，必须通过提供的公共方法进行间接访问。这样就可以防止该类的私有属性被其他类随意访问，以此来达到保护类中属性的效果。（我们还可以在公共方法中设置私有属性的访问条件，以此来提高代码的安全性）

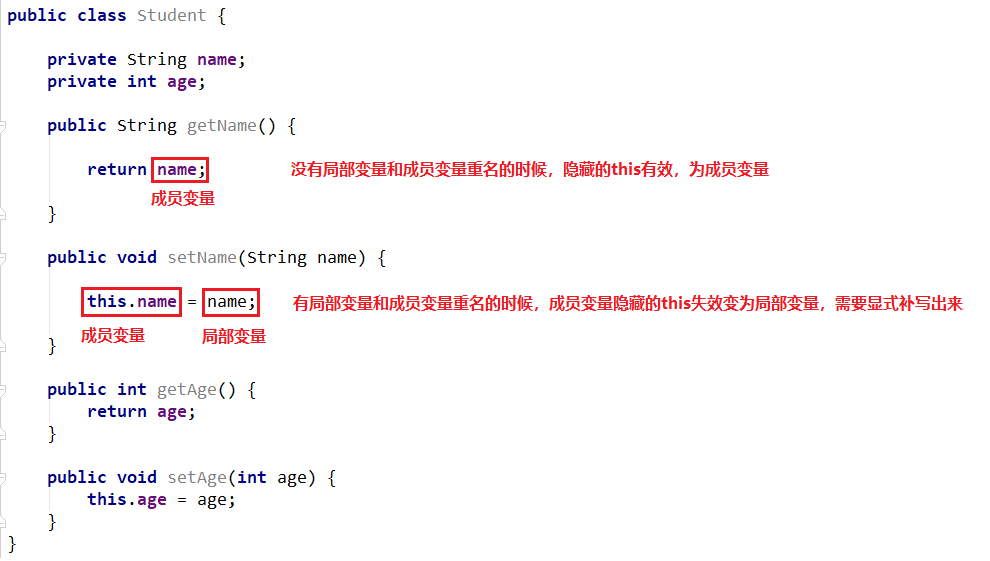
2.使用this关键字对private封装进行优化

我们发现setXxx方法中的形参名字并不符合见名知意的规定，要想见名知意，就要将其修改成相应的成员变量名，代码如下：



但是此时就会发现一个问题：既然方法中的局部变量和成员变量重名，那系统是把它们当作方法的局部变量还是类的成员变量呢？

当方法的局部变量和类的成员变量重名的时候，根据“就近原则”，二者均优先使用局部变量。即上述代码中的 name = name; 都是局部变量，此时set方法也就不能实现设置成员属性的效果了。相当于成员变量name前面隐藏的this失效了，name由成员变量转变为了局部变量，因此我们要将成员变量前面省略的this给显式补写出来，以此来告诉系统哪个是成员变量，哪个是局部变量。这就叫做：使用this关键字对private封装进行优化。具体实现代码如下：



3.构造方法

构造方法是类的一个成员方法。我们使用new Student()创建对象时，new代表在堆内存中创建一个对象，Student()代表此对象调用了类中的构造方法。构造方法在创建一个对象时被调用，一般用来初始化对象，给对象的成员变量赋初值。定义格式如下：

public 类名称 (参数类型 参数名称) {

方法体;

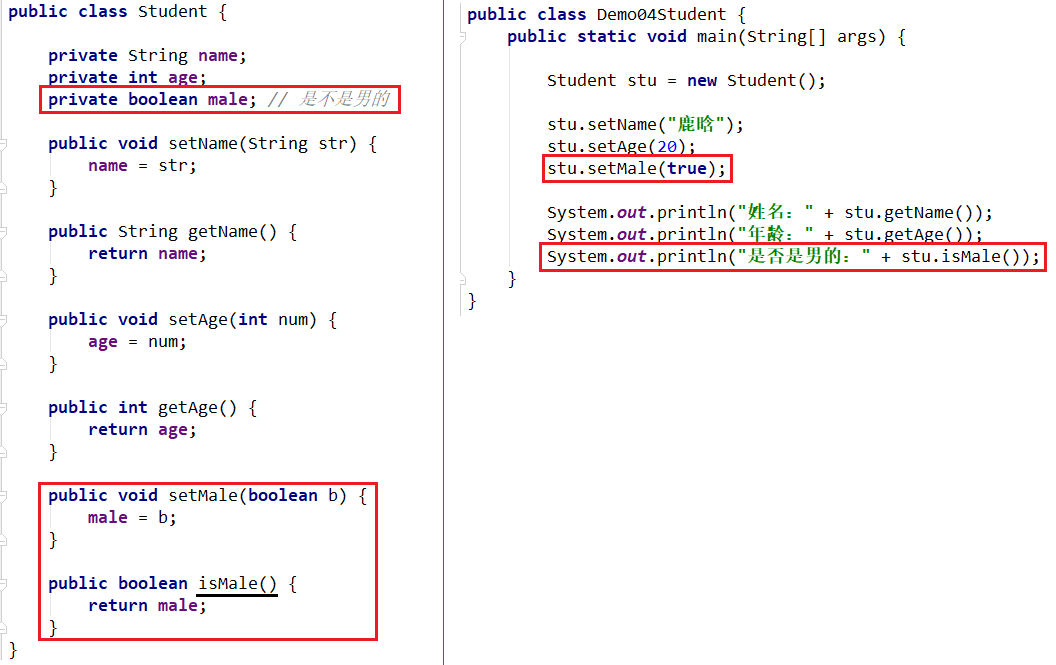
}

* 构造方法的名称必须和类名称完全一样，就连大小写也要一样
* 构造方法可以进行重载，一般使用有参构造方法来初始化对象
* 如果没有编写任何构造方法，那么编译器将会提供一个默认的隐式无参构造方法。一旦编写了至少一个构造方法，那么编译器提供的默认构造方法将会失效
* 从有参构造方法可以看出，this关键字可以使用在任何成员方法中，用来解决局部变量和成员变量重名的问题，而不仅仅局限于get、set方法

#.注意：

a.公共方法必须是getXxx、setXxx式的命名规则。

b.若private修饰的成员变量是boolean类型的，那么getXxx方法一定要写成isXxx的形式，而setXxx方法不变。



三.标准代码---JavaBean

JavaBean是Java语言编写类的一种标准规范。符合JavaBean的类，要求类必须是具体的、公共的，而且一个标准的类通常要有以下四个组成部分：

a.所有的成员变量都要使用private关键字修饰

b.为每一个private成员变量编写一对儿Getter、Setter方法

c.编写一个无参数的构造方法

d.编写一个全参数的构造方法，用来初始化对象的全部成员变量