# day06

## 面向对象思想

1. 面向对象 : 就是一种代码设计思想, 设计思维
2. 面向过程与面向对象比较:

a : 面向过程 : 现在有问题需要解决, 面向过程思维方式, 我怎么实现, 强调对于问题的解决过程, 分析思路

b : 面向对象 : 现在有问题需要解决, 面向对象思维方式, 我让谁帮我实现, 强调谁具有对于数据的把控和调遣权利

举例 : 教室打扫卫生需求问题需要解决:

面向过程 : 摆桌子和椅子, 扫地, 拖地, 关灯关空调

面向对象 : 让高利打扫卫生, 我监督

面向过程是面向对象基础, 面向对象基于面向过程实现, 面向对象属于比较高级设计思想

1. 面向对象优势:
2. 由代码的实施者, 变成代码指挥者
3. 将复杂问题相对简化
4. 更符合人类思考方式, 遇到问题先想找谁, 让谁帮我实现
5. 面向对象三大特征:
6. 封装
7. 继承
8. 多态

## 类与对象

### 类与对象的关系

1. 类 : 类型, 对于一个事物的抽象表示形式

举例 : 人物玩具模型

1. 对象 : 对于一个事物具体表现形式

举例 : 葫芦娃, 超级玛丽, 阿凡达...

### 类的组成

1. 类的组成 : 就是一系列属性和行为的集合
2. 属性 : 表示事物的特征, 以成员变量的形式定义

成员变量(全局变量) : 与以前定义变量的方式一致, 只不过, 定义位置在类中方法外

1. 行为 : 表示方法功能, 以前定义方法修饰符使用public static , 现在讲到面向对象, 方法修饰符可以去掉static

(如果本类中方法在main方法中直接调用, 修饰符添加static; 如果使用对象调用方法, 那么方法可以没有static关键字)

举例 : 人类(Person)

属性 : 姓名 , 年龄 , 性别...

行为 : 睡觉, 吃饭, 工作...

代码

|  |
| --- |
| package com.ujiuye.object;  public class Person {  // 1. 成员变量(全局变量)  // 姓名  String name;  // 年龄  int age;  // 性别  String sex;    // 2. 定义出方法功能  public void eat() {  System.out.println(name + "正在吃饭");  }    public void sleep() {  System.out.println(name + "目前已经睡觉" + age + "年");  }    public void work() {  System.out.println(name + "是" + sex + ", 工作年龄为 : " + age);  }  } |

### 对象的创建和使用

1. 创建对象语法结构:

数据类型 变量名 = new 数据类型();

类名 对象名 = new 类名();

1. 说明 :
2. 类名 : 表示对象属于什么类型, 类是引用数据类型
3. 对象名 : 符合标识符规范, 小驼峰原则, 从第二个英文单词开始, 首字母大写
4. = : 赋值运算符, 将对象在内存中占有的空间地址值赋值给对象名
5. new : 关键字, 表示新建, 为对象在堆内存中开辟空间存储数据
6. 类名 : 与前面类型保持一致即可
7. () : 构造方法的调用

举例 : Person p = new Person();

1. 对象使用:

用 对象名.使用类中的属性和功能

a : 给对象中的成员变量进行赋值 : 对象名.属性 = 值; p.name = “张三”;

b : 获取对象中的成员变量的值 : 变量 = 对象名.属性; String name = p.name;

c : 调用方法功能 : 对象名.方法名(实际参数); p.eat();

代码

|  |
| --- |
| package com.ujiuye.object;  public class TestPerson {  public static void main(String[] args) {  // 1. 创建出一个人类(Person)对象  // 类名 对象名 = new 类名();  Person p = new Person();  // 2. 给p人类对象进行成员变量赋值  p.name = "张三";  p.age = 20;  p.sex = "男";    // 3. 获取对象p中的成员变量的数据  String name = p.name;  System.out.println(name);    System.out.println(p.age);  System.out.println(p.sex);    // 4. 调用person类型中方法功能  p.eat();  p.sleep();  p.work();  }  } |

### 练习

案例 : 定义出一个小汽车类, 小汽车具有轮胎个数(num) 和 颜色(color) , 也具有跑的功能(run) , 设计代码创建出两个小汽车对象, 并且运行run功能; 要求 : run功能可以将小汽车的属性展现出来

代码

|  |
| --- |
| package com.ujiuye.object;  public class Car {  // 轮胎个数(num)  int num;  // 颜色(color)  String color;    public void run() {  System.out.println(color +"颜色的"  + num + "个轮胎个数的汽车正在运行");  }  } |

|  |
| --- |
| package com.ujiuye.object;  public class TestCar {  public static void main(String[] args) {  // 1. 创建出一个汽车对象  Car c = new Car();  System.out.println(c);  c.color = "红";  c.num = 4;  c.run();    // 2. 创建出另外一个汽车对象  Car c1 = new Car();  System.out.println(c1);  c1.color = "军绿色";  c1.num = 8;  c1.run();    // 3. 两个car类型的对象引用指向同一块内存空间  Car c2 = c1;  System.out.println(c2.num);// 8    Car c3 = new Car();  System.out.println(c3.num);// 0  System.out.println(c3.color);// null  }  } |

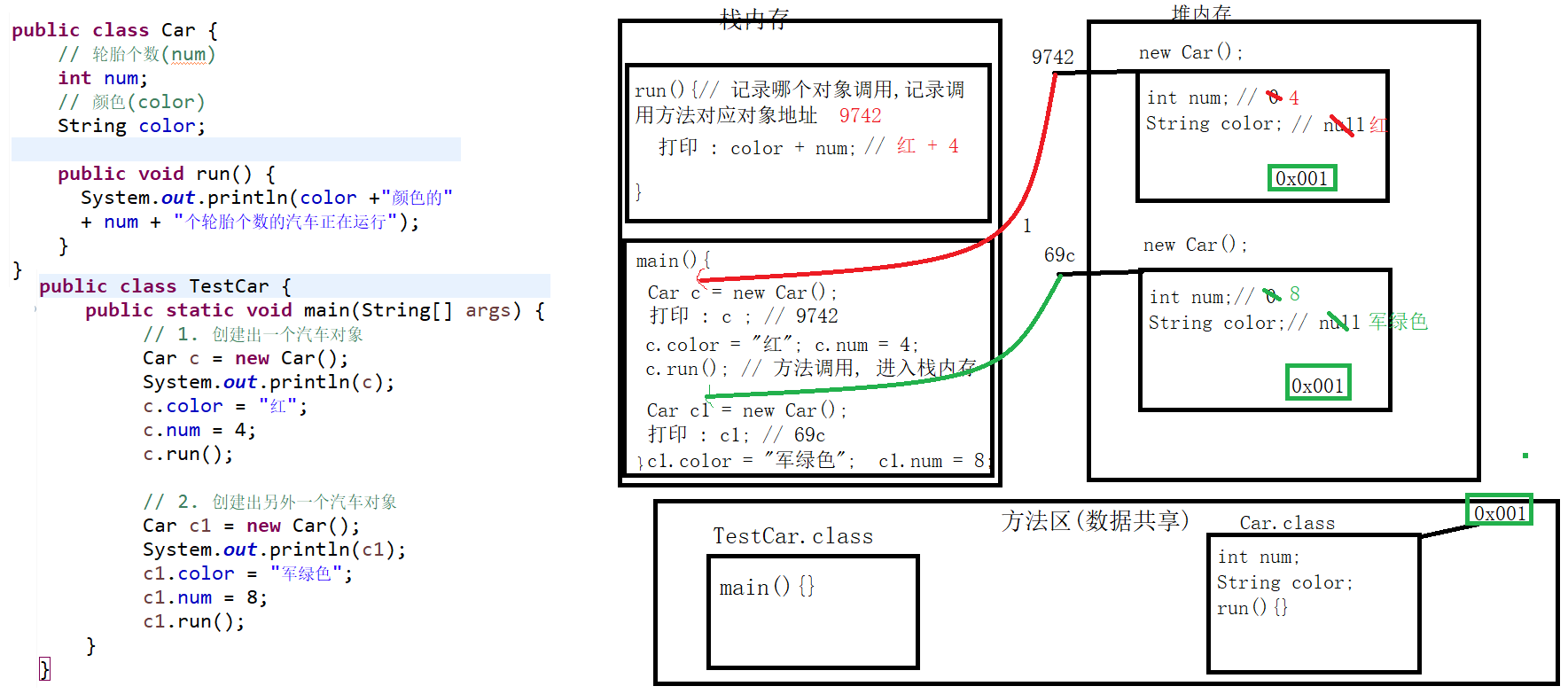
### 对象在内存中理解

栈内存 : 方法调用, 进到栈内存中占有空间运行, 方法运行完毕, 弹出栈内存, 释放空间

堆内存 : 引用数据类型通过new关键字开辟空间, 创建对象, 需要在堆内存中占有空间

方法区(数据共享) : 一个类型对应的.class字节码文件, static静态修饰成员, 字符串常量

1. 两个对象在内存中理解



a : 将需要操作类型对应的.class字节码文件加载到方法区中, Car.class进入到方法区中, 有一个内存地址

b : main方法作为程序唯一执行入口, 被JVM虚拟机主动调用, main方法进去到栈内存执行

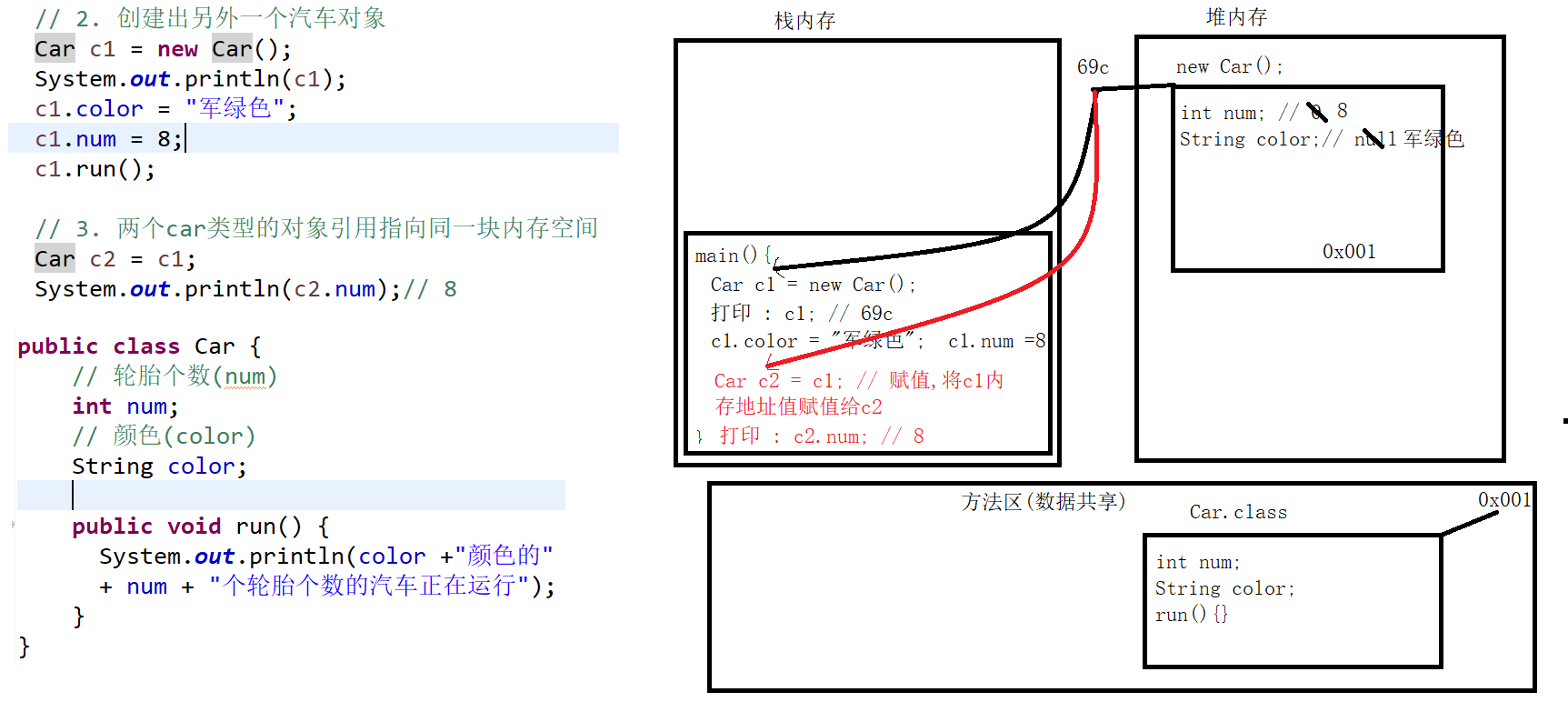
c : 创建出一个Car类型对象, 需要在堆内存中开辟出一块空间, 记录对象所属类型(Car.class)在方法区中地址, 将类型中所有非静态成员变量加载进入到对象所在堆内存区域中, JVM虚拟机会为每一个成员变量进行默认的赋初值动作

d : 将对象内存地址返回到栈内存的变量引用处

e : 在栈内存中使用对象, 就相当于在操作堆内存中空间地址中数据

注意 : 方法调用时, 在方法中, 记录, 哪个对象调用方法, 方法记下这个对象的地址, 方法中使用的变量从记录的地址中获取

1. 两个变量引用指向同一块内存空间



### 匿名对象

1. 有名对象 : Person p = new Person();
2. 匿名对象 : new Person();
3. 匿名对象使用 : 只能使用一次, 因为匿名对象在堆内存中的地址值, 没有变量记录, 所以使用一次结束后, 无法找到这个匿名对象地址值, 也就只能使用一次

new Person().属性;

new Person().方法(实际参数);

1. 匿名对象存在意义 : 如果一个变量(对象)在代码中无法使用, 那么这个变量占有的内存空间区域就会变成垃圾, 等待被回收掉, 匿名对象在内存中停留的时间是短暂的, 当一次属性或者方法调用完毕, 匿名对象在内存中就没有意义, 马上变成垃圾等待回收

1. 匿名对象使用场景 : 思考, 如果一个类型中的成员变量或者某方法只调用一次就够了, 可以使用匿名对象

代码

|  |
| --- |
| package com.ujiuye.object;  public class Demo01\_匿名对象使用 {  public static void main(String[] args) {  // 匿名对象  int age = new Person().age;  System.out.println(age);// 0    new Person().eat();// null正在吃饭  }  } |

## 封装

### 3.1封装的概述

1. 封装 : 面向对象中一大特征
2. 封装 : 隐藏属性或者事物的实现细节, 对外提供公共的访问方式
3. 封装好处 :

a : 因为隐藏实现细节, 提高代码安全性

b : 封装实际代码中体现很多, 类和方法都属于封装的表现形式, 于是可以提高代码的复用性

### 3.2 private关键字

1. private : 关键字, 修饰符, 表示私有的, 私密的.

2. private 可以修饰

1) 成员变量(实际开发中使用最多的修饰场景)

2) 修饰方法

3) 修饰构造方法

4) 修饰内部类

3. 使用private修饰的成员只能在当前类型中使用(只能在本类中使用)

4. 私有private关键字只是封装的一种表现形式

### 3.3 Setter和Getter方法

提供对外的,公共的访问私有成员变量的方式

1. Setter 方法 : set方法, 用于给类型中的私有成员变量进行赋值; 私有成员变量在外类中, 只能通过Setter方法进行赋值, 因此方法中限制如何赋值只能遵守, 提高代码的安全性

2. Getter 方法 : get方法, 用于获取类型中私有成员变量的值

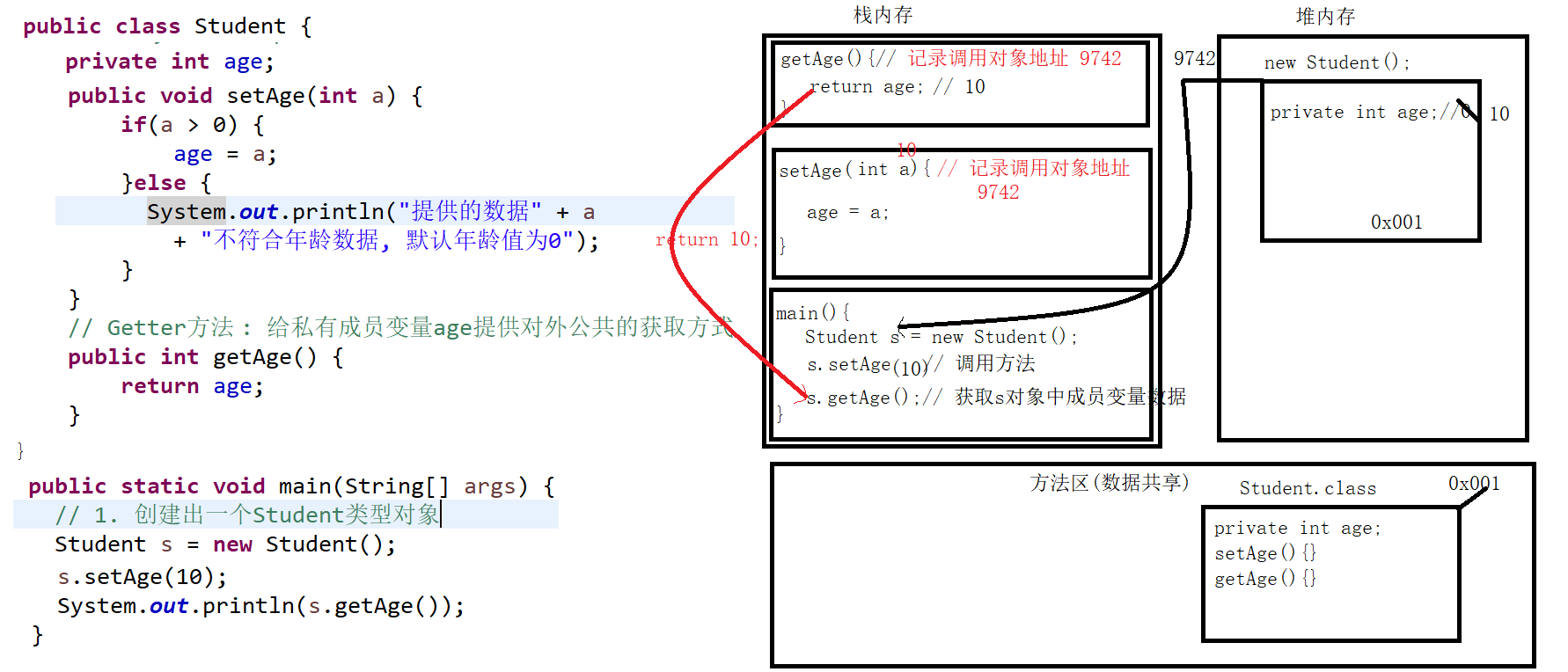
3. 总结 : 实际开发中, 基本上所有的实体类(现实世界中存在的事物类型, 客户, 手机, 汽车, 产品...), 属性都是使用private修饰的, 再提供对应的set和get方法进行访问

代码

|  |
| --- |
| package com.ujiuye.object;  public class Student {  // 姓名  String name;  // 班级号  String classNo;  // 学号  String id;  // 年龄, 因为使用private修饰, age只能在Student类中使用, 其他所有外类不能直接使用和访问  private int age;    // Setter方法 : 给age私有成员变量提供对外的, 公共的访问age方式, 给age进行赋值  public void setAge(int a) {  if(a > 0) {  age = a;  }else {  System.out.println("提供的数据" + a + "不符合年龄数据, 默认年龄值为0");  }  }    // Getter方法 : 给私有成员变量age提供对外公共的获取方式  public int getAge() {  return age;  }    public void study() {  System.out.println("学生一直在努力的学习");  }  } |

|  |
| --- |
| package com.ujiuye.object;  public class TestStudent {  public static void main(String[] args) {  // 1. 创建出一个Student类型对象  Student s = new Student();  s.name = "大美丽";  s.classNo = "3年2班";  s.id = "001";  // 想法 : 不想让直接给age进行赋值, 不好把控赋值是否正确, 可以使用private私有修饰age  // The field Student.age is not visible  // s.age = 10;  s.setAge(10);    System.out.println(s.getAge());  }  } |

### 3.4 Setter和Getter方法在内存中理解



### 3.5成员变量和局部变量的比较

1. 定义位置不同:
2. 成员变量 : 定义在类中方法外的变量, 称为成员变量(全局变量)
3. 局部变量 : 定义在方法中, 或者是定义在参数列表中变量, 称为局部变量
4. 在内存中存储区域不同:
5. 成员变量 : 跟着对象存储在堆内存中
6. 局部变量 : 跟着方法存储在栈内存中
7. 生命周期不同:
8. 成员变量 : 跟着对象创建存在有值, 随着对象消亡而消失
9. 局部变量 : 跟着方法调用进栈, 存在有值, 随着方法运行完毕, 局部变量跟着方法消亡
10. JVM初始化值不同
11. 成员变量 : JVM虚拟机会有默认赋初值动作, 根据成员变量所属数据类型不同决定初值是多少
12. 局部变量 : JVM虚拟机不会主动给局部变量赋值, 于是局部变量没有手动赋值, 不能使用

### 3.6变量的就近访问原则和this关键字

1. 变量就近访问原则 : 当使用某一个变量时, 使用的就是最近一次变量定义结果;

现象 : 如果有局部变量, 优先使用局部变量; 如果没有局部变量, 使用成员变量.

1. this : 关键字, 表示当前类型对象的引用(表示是一个对象)

a : this关键字第一种使用方式 : 可以区分成员变量和局部变量重名问题, 带有this.关键字表示成员变量的使用

b : 哪个对象调用带有this关键字的方法, this关键字就表示哪个对象

代码

|  |
| --- |
| package com.ujiuye.object;  public class Demo02\_VariableUse {  // 成员变量 : 跟着对象存储在堆内存中  int i = 10;    public void useI() {  // 局部变量 : 跟着useI方法存储在栈内存中  int i = 20;  // 1. i值结果是局部变量20, 表现出变量就近访问原则  System.out.println(i);// 20  // 2. 问题 : 如果就需要在useI方法中调用成员变量i  // 于是 : 思考,今天讲类中的成员变量和方法调用  System.out.println(new Demo02\_VariableUse().i);// 10  // 3. 另外一种调用成员变量i方式 : this关键字  System.out.println(this.i);// 10  System.out.println("this---" + this);  }    public static void main(String[] args) {  Demo02\_VariableUse demo2 = new Demo02\_VariableUse();  System.out.println("demo2-----"+demo2);  demo2.useI();  System.out.println("-------------");    Demo02\_VariableUse demo3 = new Demo02\_VariableUse();  System.out.println("demo3----"+demo3);  demo3.useI();  }  } |

### this关键字和变量访问原则在Setter方法中的使用练习

代码

|  |
| --- |
| package com.ujiuye.object;  public class Product {  // 属性 : 产品编号  private String id;  // 产品名称  private String name;  // 产品单价  private double price;    // 可以使用快捷键 : alt + shift + s 自动生成私有成员变量对应的set和get方法  public String getId() {  return id;  }  public void setId(String id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public double getPrice() {  return price;  }  public void setPrice(double price) {  this.price = price;  }  } |