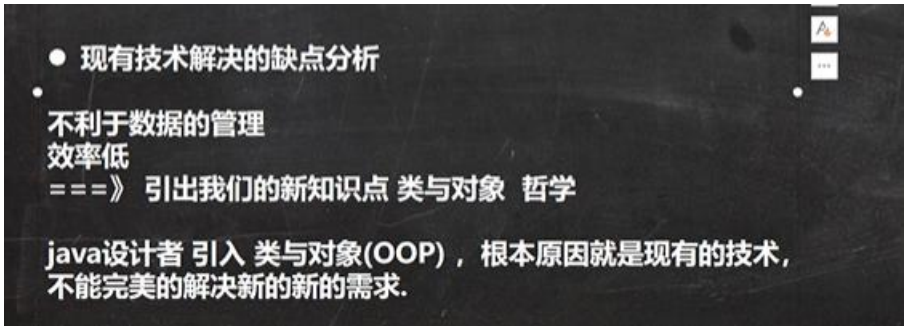
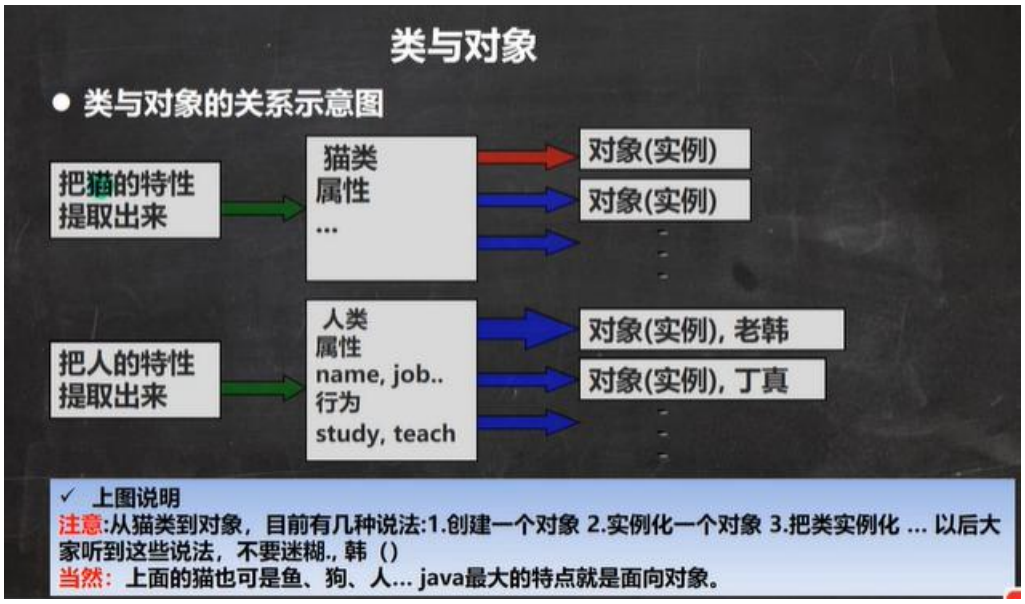
类与对象

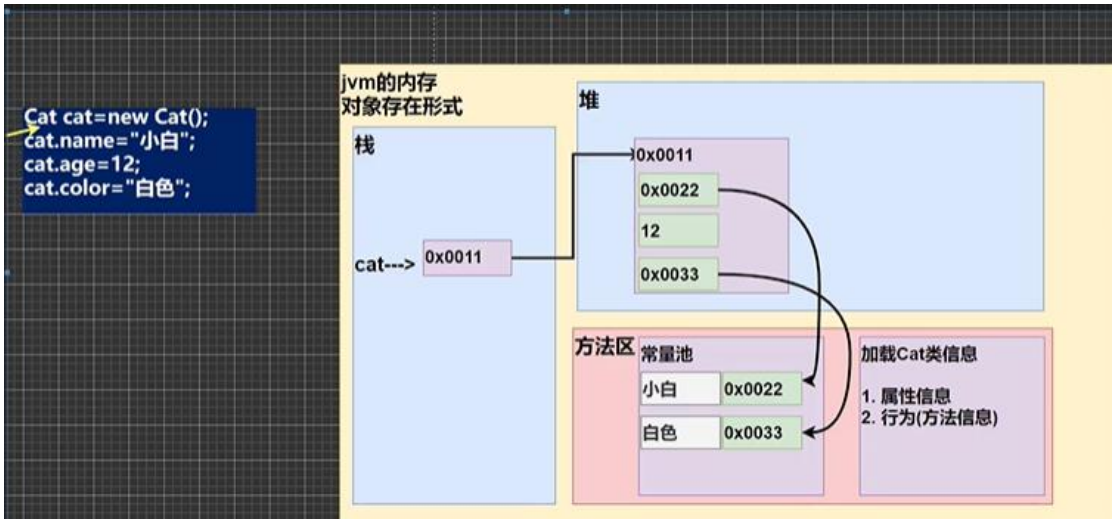


关系示意图（猫举例）

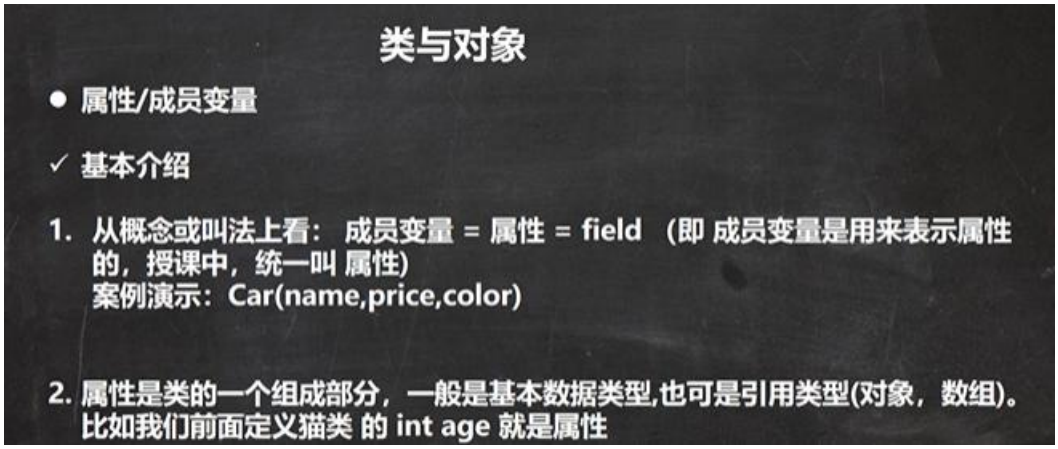


对象存在形式

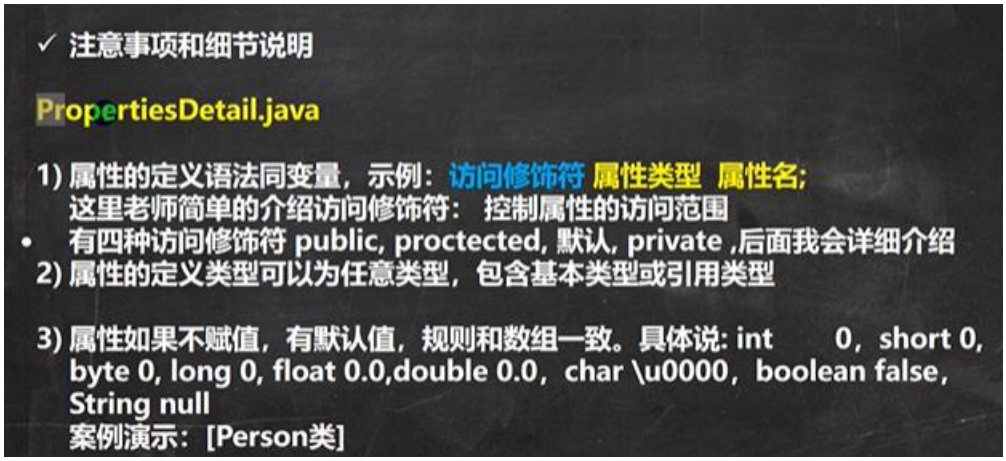
先在栈中创建数据空间，再在堆中开辟空间存储对象



属性 （成员变量）



细节

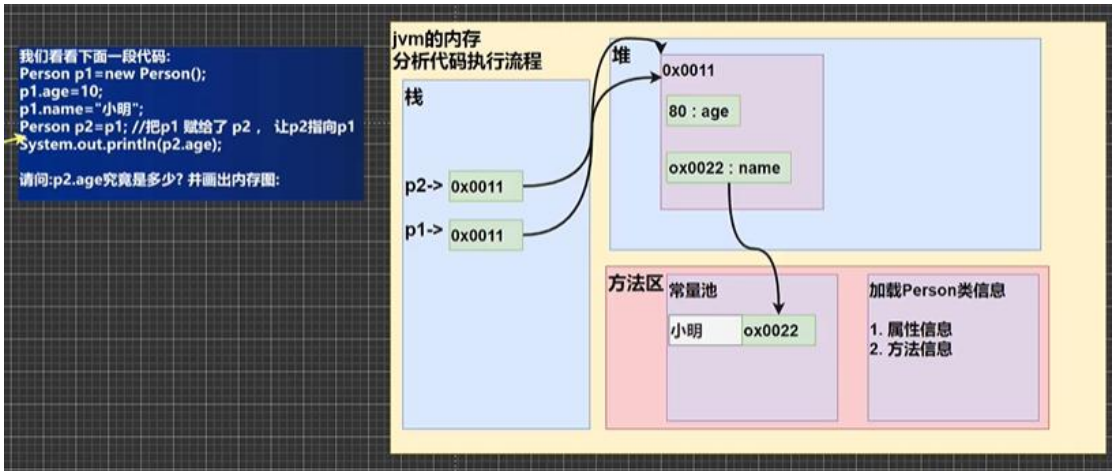


Person p1=new Person();

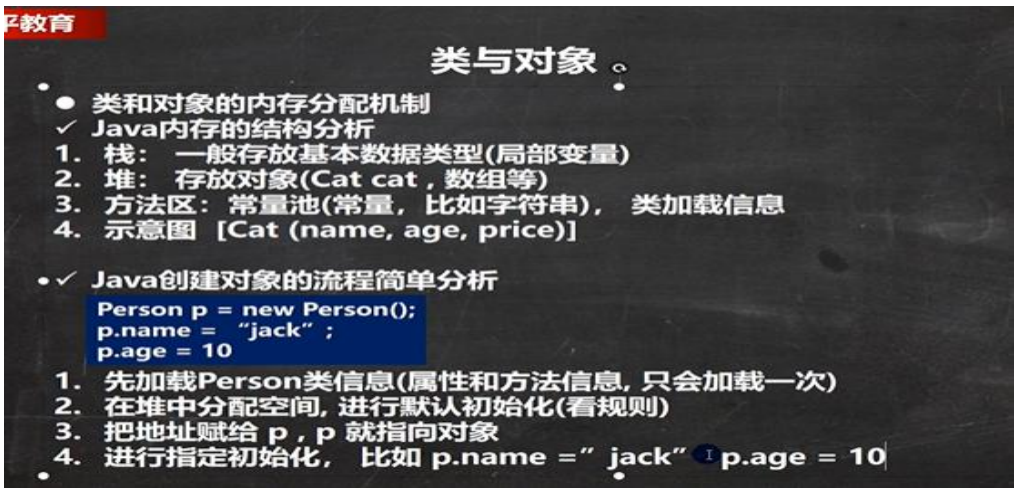
P1.age=10;

Person p2=p1;

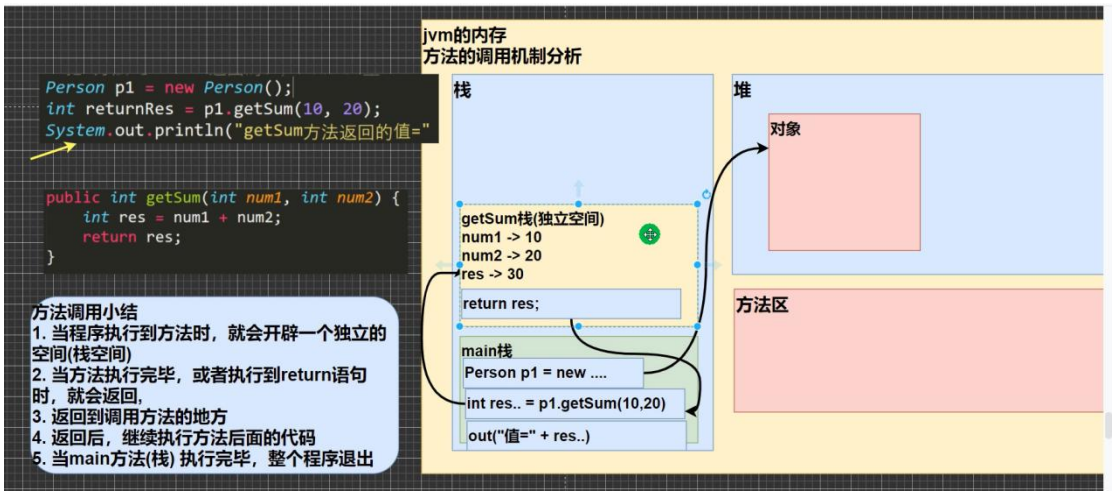
p2.age=?



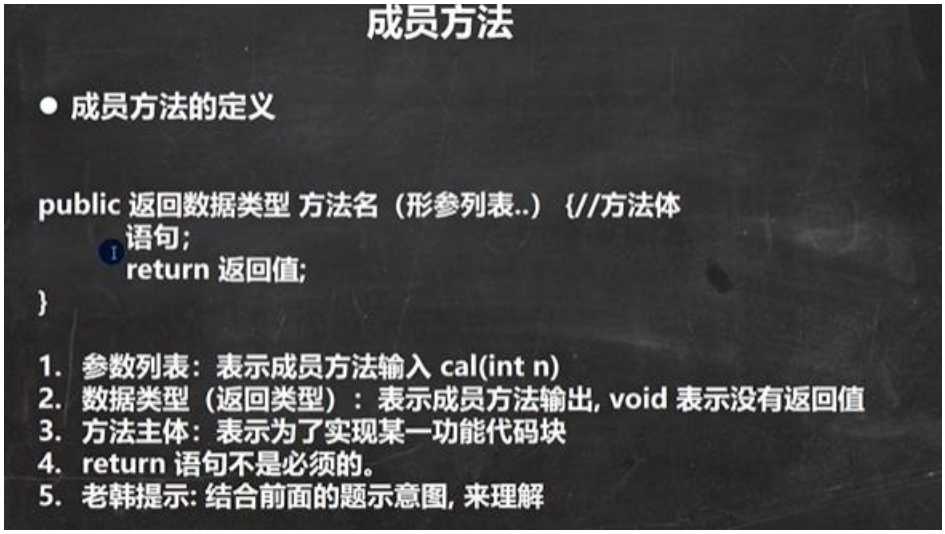
创建对象简单流程



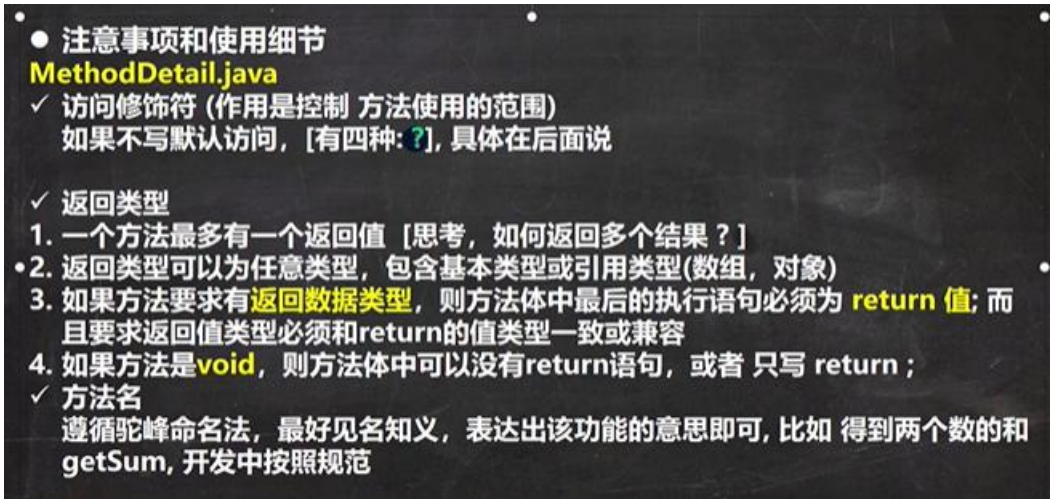
Return 调用得到的结果返回到栈中的空间（谁调用返回给谁）

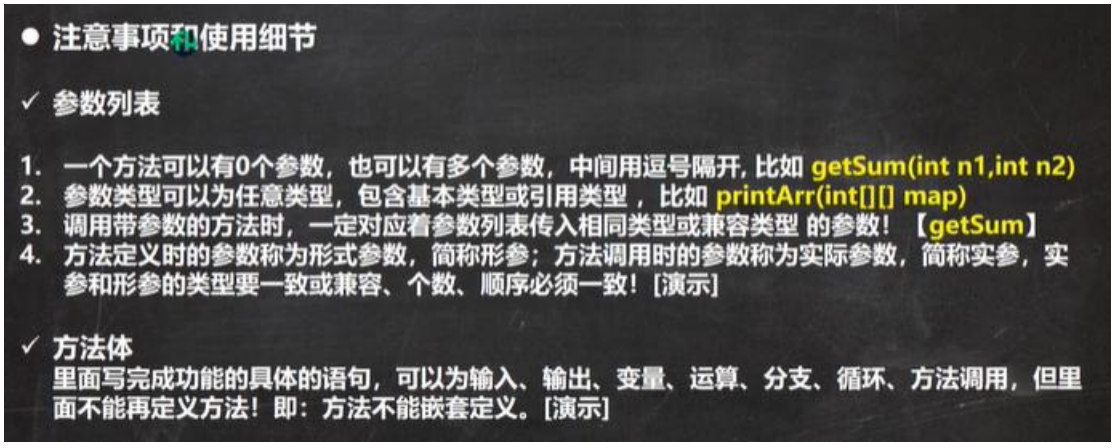


方法定义



方法细节



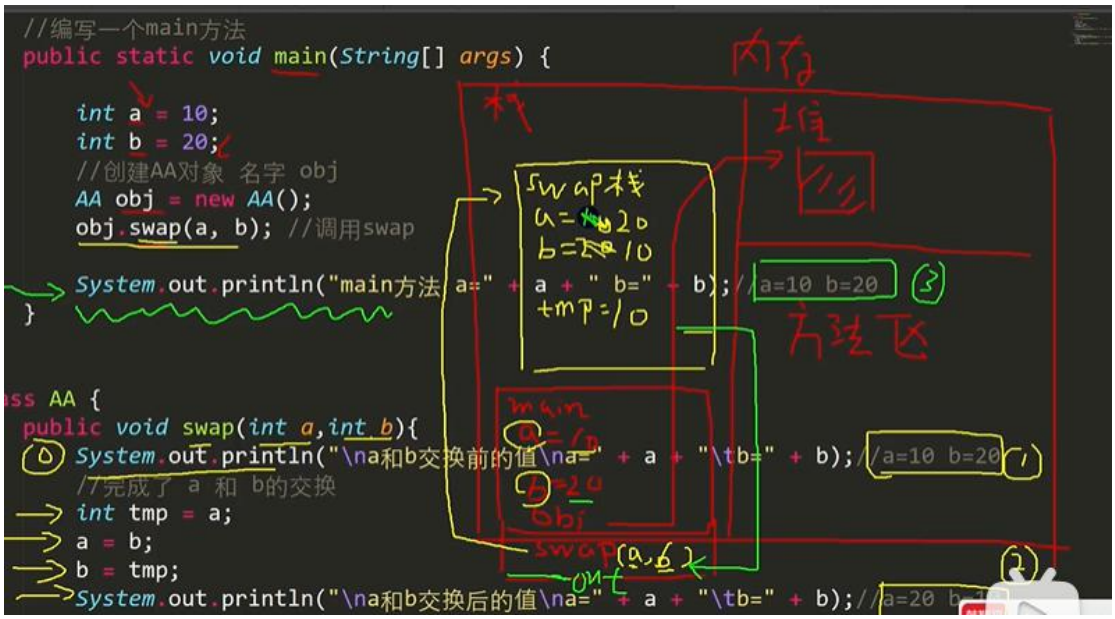


第 3 点兼容类型的参数指的是 可以自动转换的类型 比如形参设置 int 类型 传进去的实参是 byte 变量也可以

另外 方法不能嵌套定义

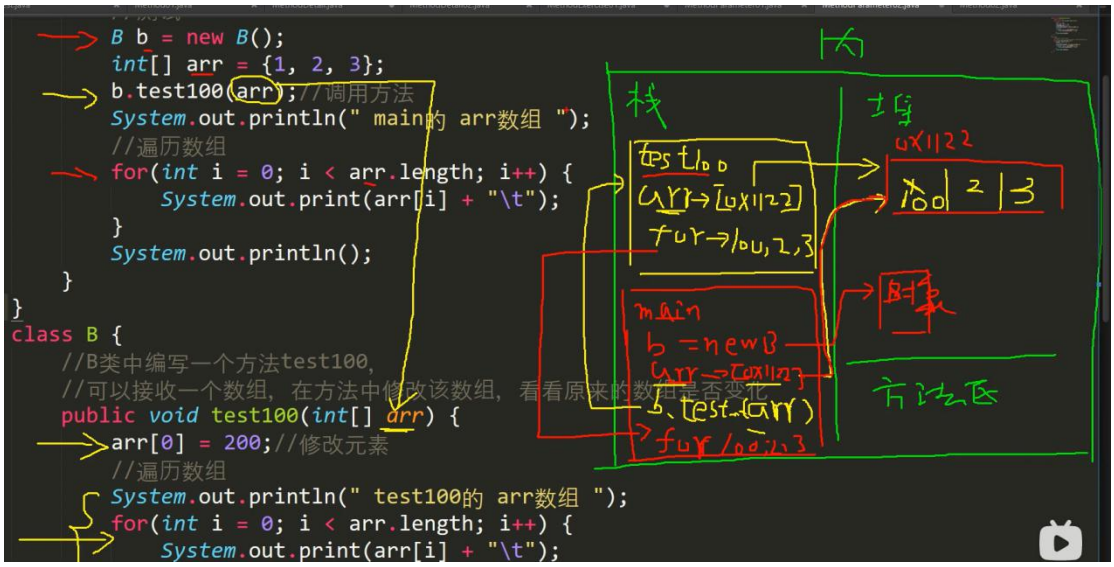
第 2 点指的是 A 类中方法里要调用 b 类方法 就需要先声明 b 类对象 然后再调用

参数传递机制 1（值参数传递）



方法中调用之后，会在栈中创建方法的独立空间 在这个独立空间中 a 和 b 发生了交换
但在 main 方法中 main 空间 a 和 b 并未交换

参数传递机制 1（引用参数传递）

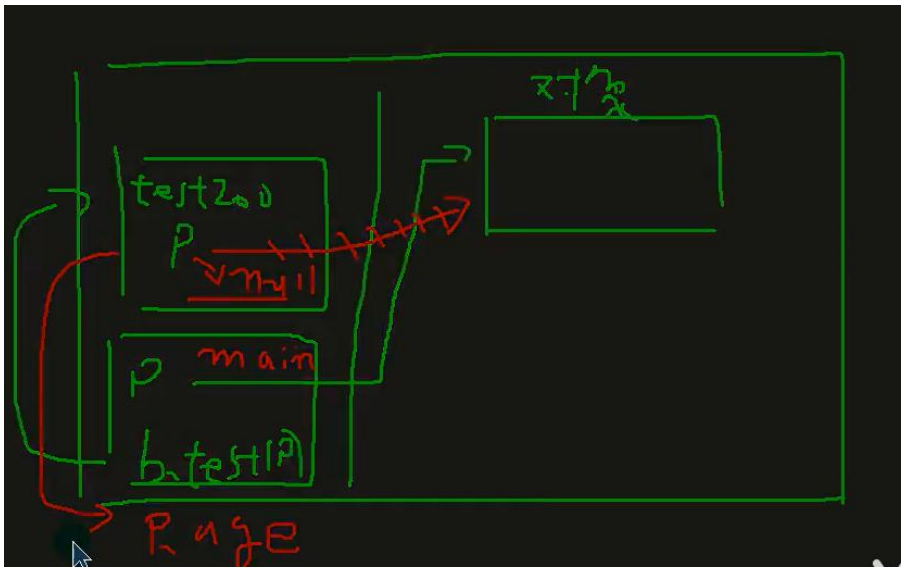


引用参数 指向的是地址 方法中对其更改之后，就彻底更改了

对象 数组 都是引用类型

```
        p.age = 10;

        b.test200(p);
        //测试题，如果 test200 执行的是 p = null ,下面的结果是 10
        System.out.println("main 的p.age=" + p.age);//10000
    }
}
class Person {
    String name;
    int age;
}
class B {

    public void test200(Person p) {
        //p.age = 10000; //修改对象属性
        //思考
        p = null;
    }
```

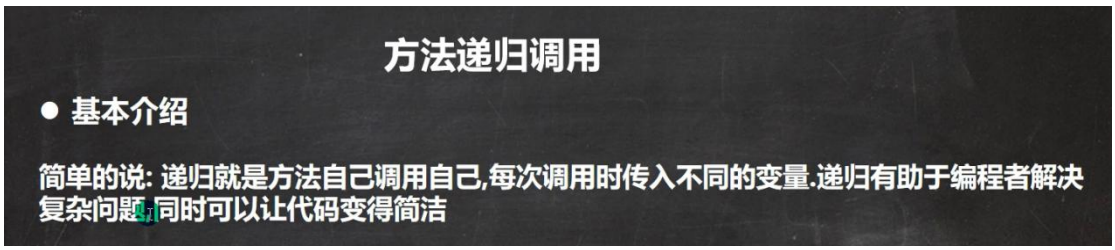
这个结果为啥还是 10 呢？



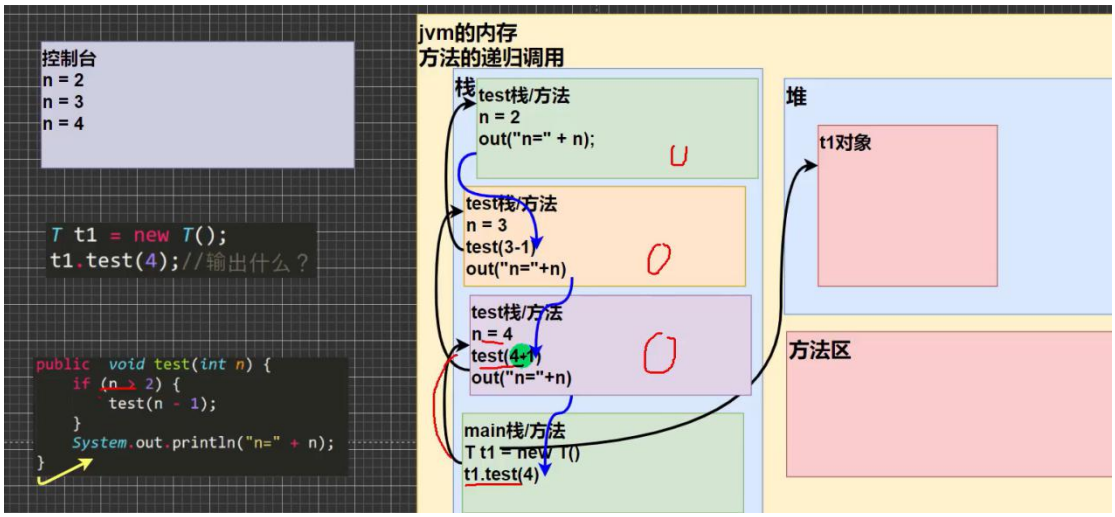
因为 p 置空时意思是从指向对象变成指向空 但对 main 的变量并未更改 所以返回 main 中变量值

```
public void test200(Person p) {
    //p.age = 10000; //修改对象属性
    //思考
    p = new Person();
    p.name = "tom";
    p.age = 99;
```

还是返回 10 因为 new 给创建新对象在堆里开辟了新空间 还是不影响 main 中的变量

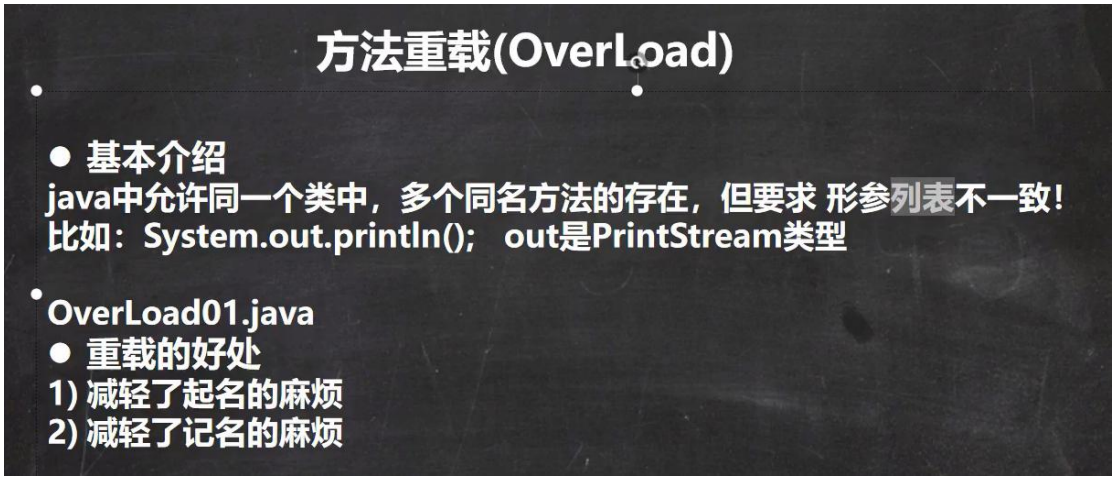


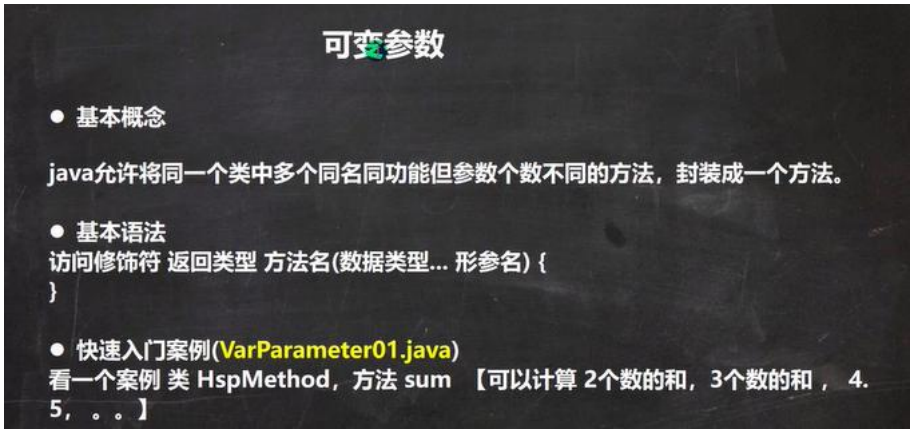
递归机制 1



方法递归的存储方式
每递归一次就执行一个方法 开辟一个栈空间

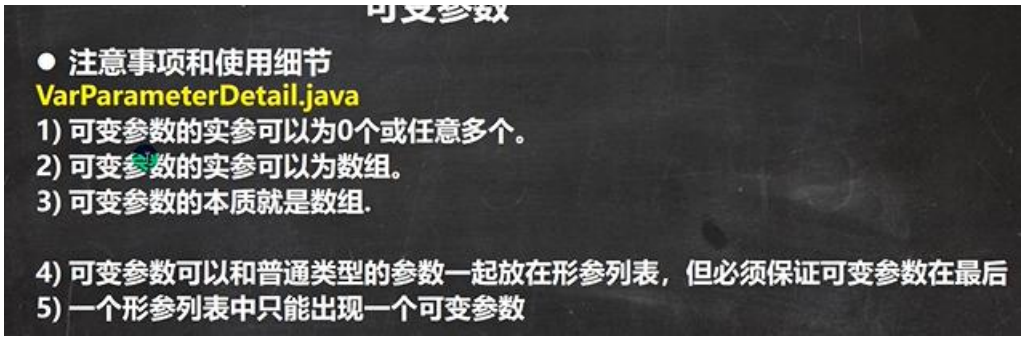
方法重载



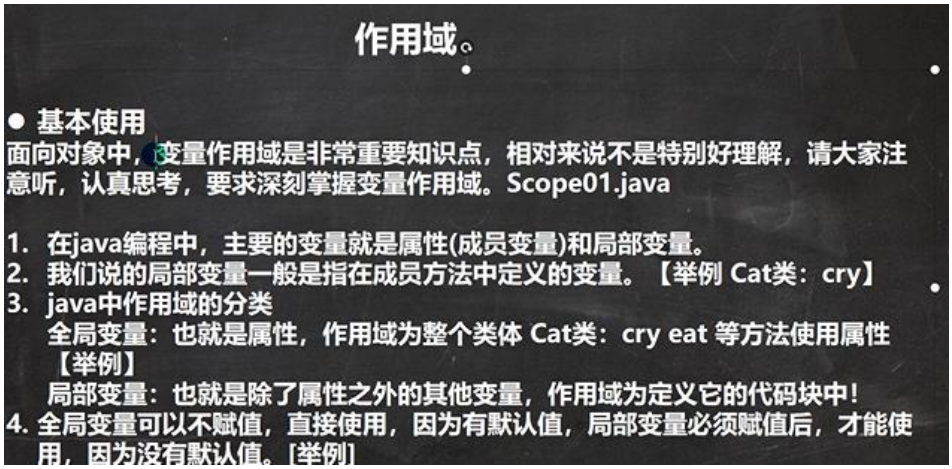


例子 double ... 变成了一个集合

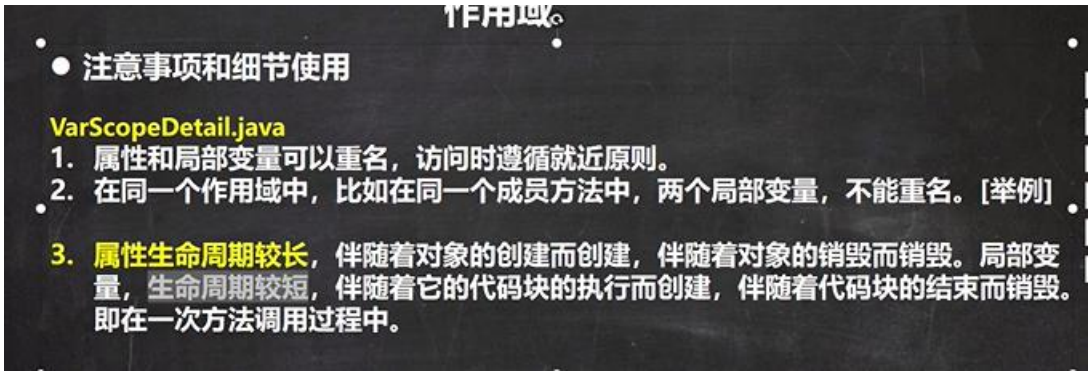
```
    public static void main(String[] args){
        HspMethod md = new HspMethod();
        System.out.println(md.showScore("李四",89,99.5));
        System.out.println(md.showScore("周三",79,89.5,67.7));
        System.out.println(md.showScore("王五",79,89.5,67.7,89,99.5));
    }


ass HspMethod{
    public String showScore(String name,double... scores){
        double totalScore = 0;
        for(int i = 0;i < scores.length;i++){
            totalScore += scores[i];
        }
        return name + "的" + scores.length + "门课总分为" + totalScore;
    }
```



作用域：



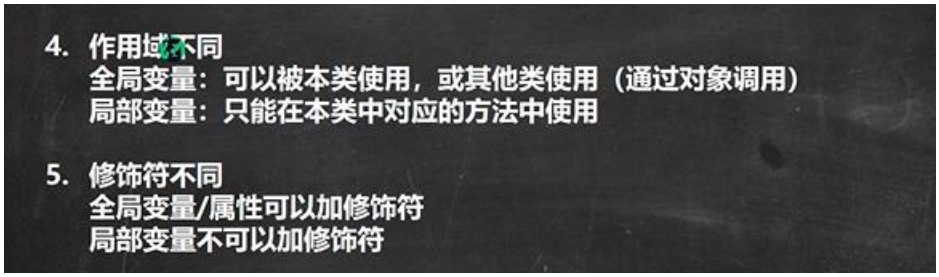
其中第 4 点就是 全局变量有默认值 未给初值也可以直接用
局部变量未给初值不能直接用



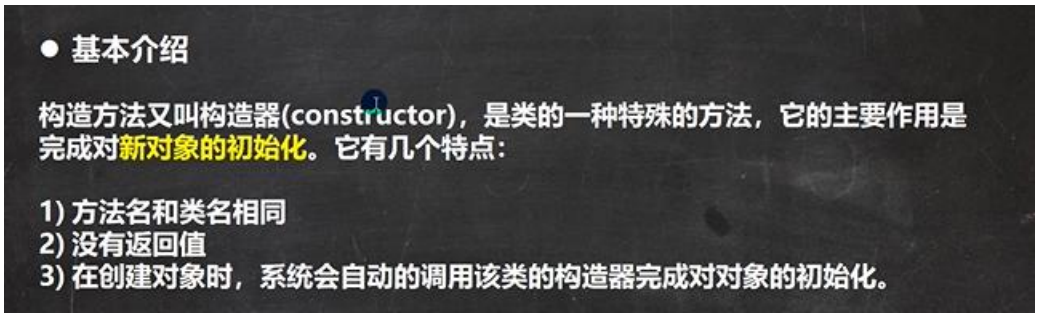
就近原则举例

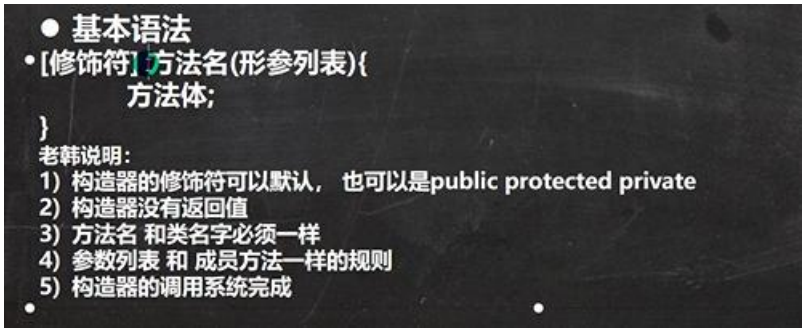
```
String name = "jack";

public void say() {
    String name = "king";
    System.out.println("say() name=" + name);
}
```

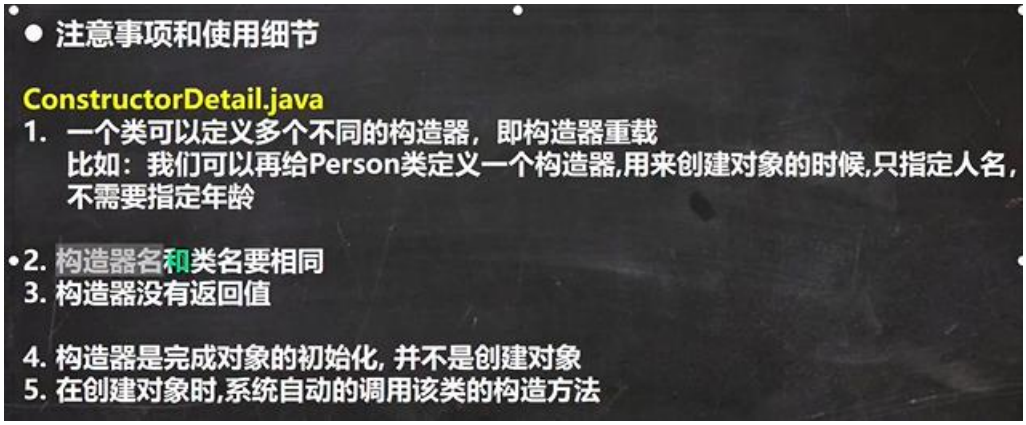
输出 king

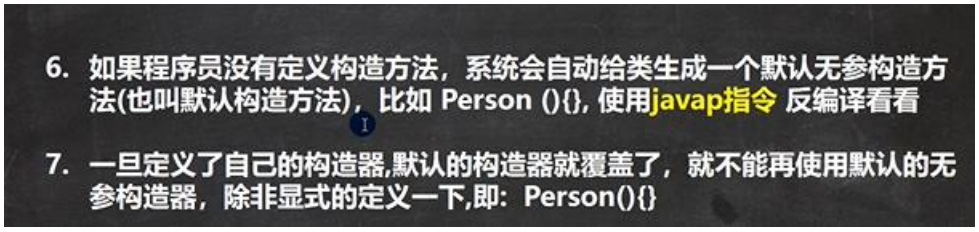


构造器/构造方法









Javap 反编译

1) javap 是 JDK 提供的一个命令行工具,javap 能对给定的 class 文件提供的字节代码进行反编译。

2) 通过它，可以对照源代码和字节码，从而了解很多编译器内部的工作，对更深入地理解如何提高程序执行的效率等问题有极大的帮助。

3) 使用格式

```
javap <options> <classes>
```

常用: javap -c -v 类名

-help --help -?	输出此用法消息
-version	版本信息
-v -verbose	输出附加信息
-l	输出行号和本地变量表
-public	仅显示公共类和成员

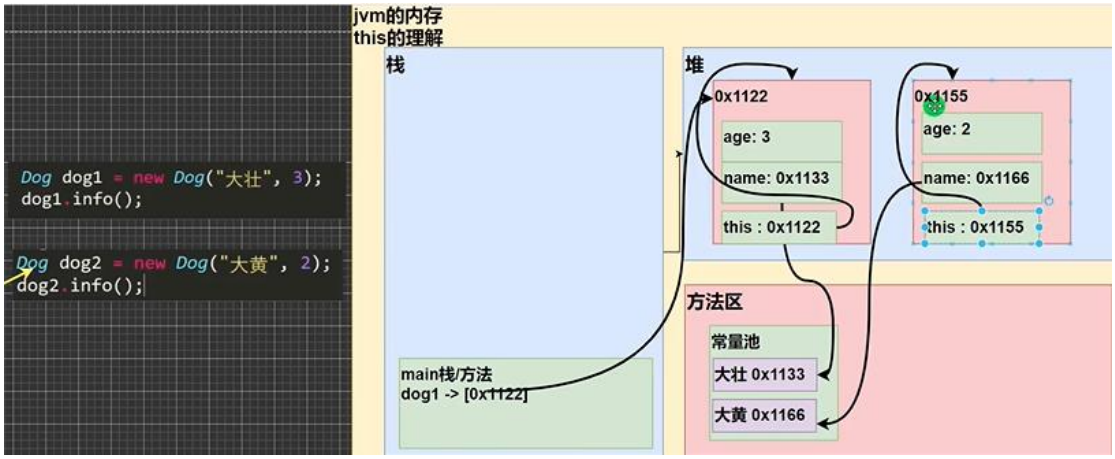
Person p=new Person（）；这个叫对象引用在栈中创空间
p.age=9;//这个才给对象创建堆空间

This

```
String name;
int age;
```

```
public Dog(String name, int  age){//构造器
    //this.name 就是当前对象的属性
    this.name = name;
    //this.age 就是当前对象的
    this.age = age;
}
```

This 指的是当前对象的（this 指向对象本身）



Hashcode 的使用　对象.hashCode()与 this.hashCode()的输出的哈希码值一样

hashCode

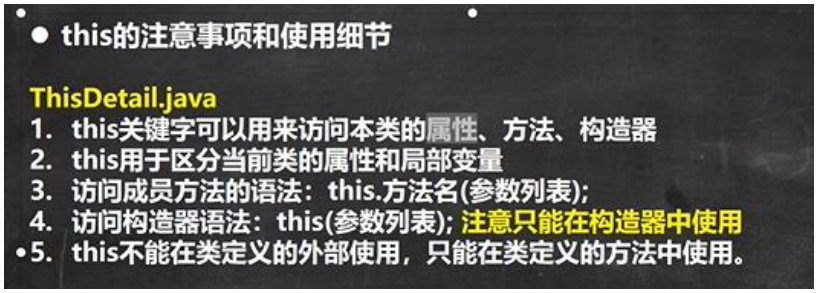
public int **hashCode**()

返回该对象的哈希码值。支持此方法是为了提高哈希表（例如 java.util.Hashtable 提供的哈希表）的性能。

hashCode 的常规协定是：

- 在 Java 应用程序执行期间，在对同一对象多次调用 hashCode 方法时，必须一致地返回相同的整数，前提是将对象进行 equals 比较时所用的信息没有被修改。从某一应用程序的一次执行到同一应用程序的另一次执行，该整数无需保持一致。
- 如果根据 equals(Object) 方法，两个对象是相等的，那么对这两个对象中的每个对象调用 hashCode 方法都必须生成相同的整数结果。
- 如果根据 equals(java.lang.Object) 方法，两个对象不相等，那么对这两个对象中的任一对象上调用 hashCode 方法*不* 要求一定生成不同的整数结果。但是，程序员应该意识到，为不相等的对象生成不同整数结果可以提高哈希表的性能。

实际上，由 Object 类定义的 hashCode 方法确实会针对不同的对象返回不同的整数。（这一般是通过将该对象的内部地址转换成一个整数来实现的，但是 Java™ 编程语言不需要这种实现技巧。）



同类下一个构造器去访问另一个构造器用 this（复用构造器）只能复用一个构造器

```
public T() {
    System.out.println("T() 构造器");
    //这里去访问 T(String name, int age)
    this("jack", 100);
}

public T(String name, int age) {

    System.out.println("T(String name, int age) 构造器");
}
```

上图错误，对 this 的调用必须是构造器中的第一条语句，所以应把 this 和 Sys。。。调换