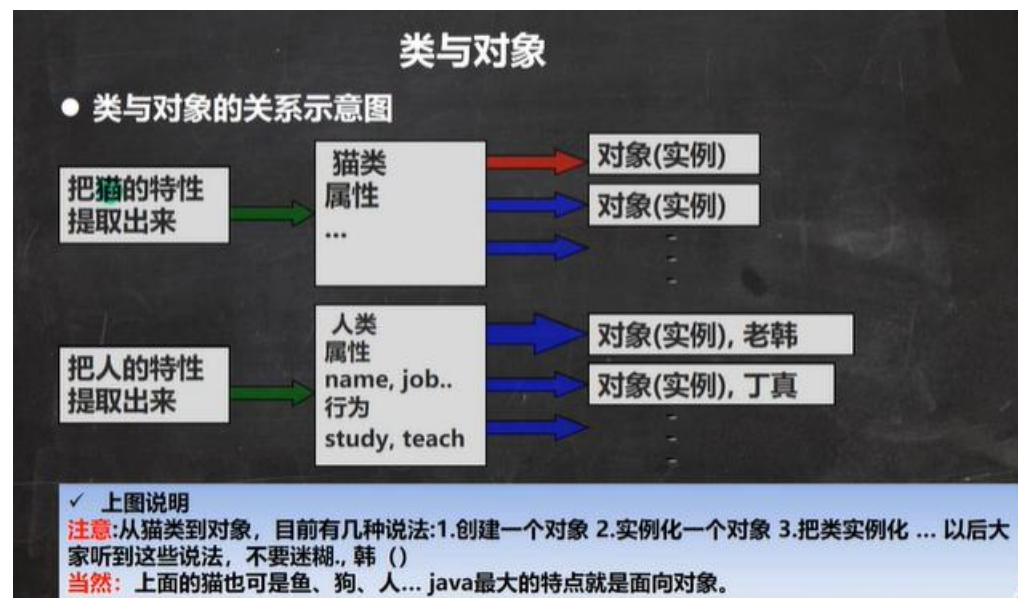


类与对象

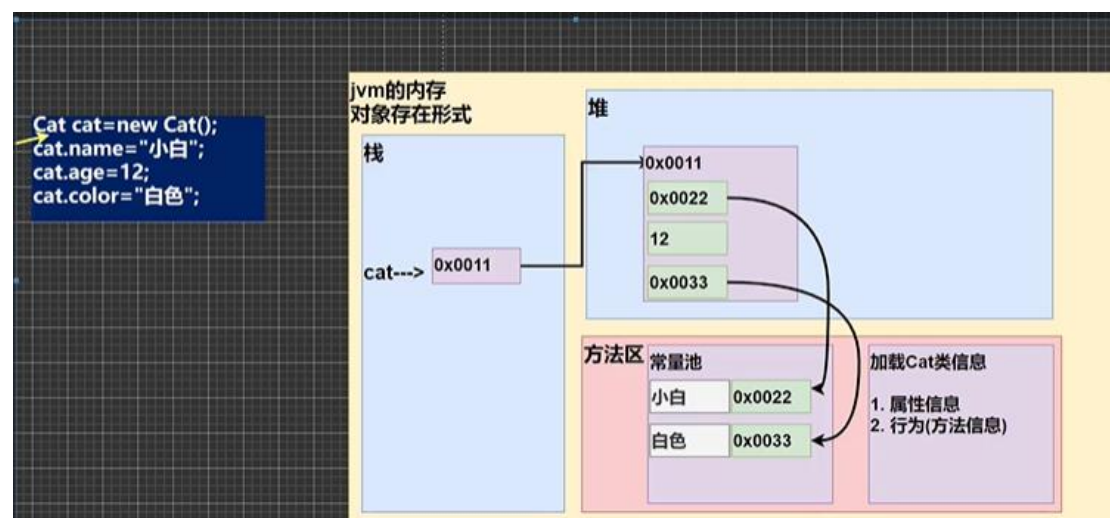
- 现有技术解决的缺点分析
 - 不利于数据的管理
 - 效率低
- ===》引出我们的新知识点 类与对象 哲学
- java设计者 引入 类与对象(OOP) , 根本原因就是现有的技术, 不能完美的解决新的新的需求.

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）

类与对象

- 属性/成员变量
- ✓ 基本介绍

1. 从概念或叫法上看：成员变量 = 属性 = field（即成员变量是用来表示属性的，授课中，统一叫属性）
案例演示：Car(name,price,color)
2. 属性是类的一个组成部分，一般是基本数据类型,也可能是引用类型(对象，数组)。比如我们前面定义猫类的 int age 就是属性

细节

- ✓ 注意事项和细节说明

PropertiesDetail.java

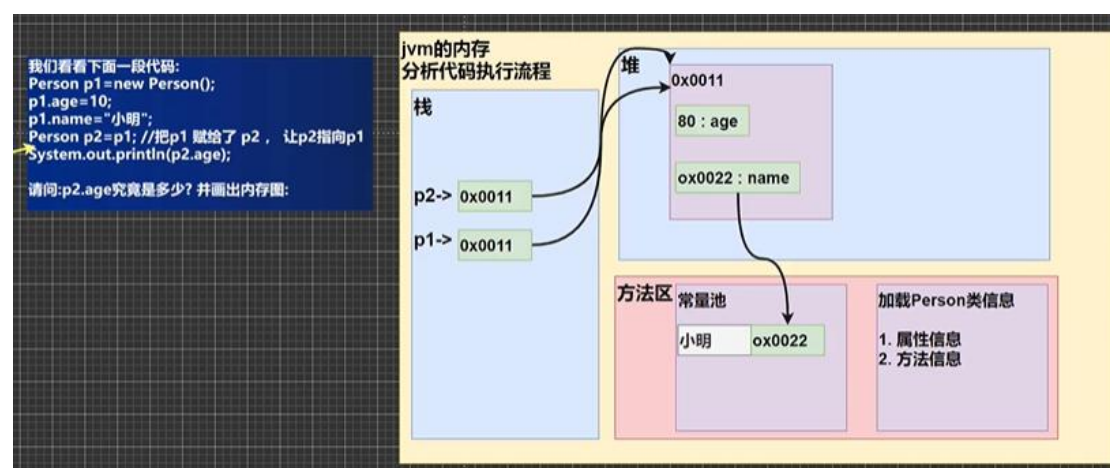
- 1) 属性的定义语法同变量，示例：**访问修饰符 属性类型 属性名**；
这里老师简单的介绍访问修饰符：控制属性的访问范围
- 有四种访问修饰符 public, protected, 默认, private ,后面我会详细介绍
- 2) 属性的定义类型可以为任意类型，包含基本类型或引用类型
- 3) 属性如果不赋值，有默认值，规则和数组一致。具体说: int 0, short 0, byte 0, long 0, float 0.0, double 0.0, char \u0000, boolean false, String null
案例演示：[Person类]

```
Person p1=new Person();
```

```
p1.age=10;
```

```
Person p2=p1;
```

```
p2.age=?
```



创建对象简单流程

教育

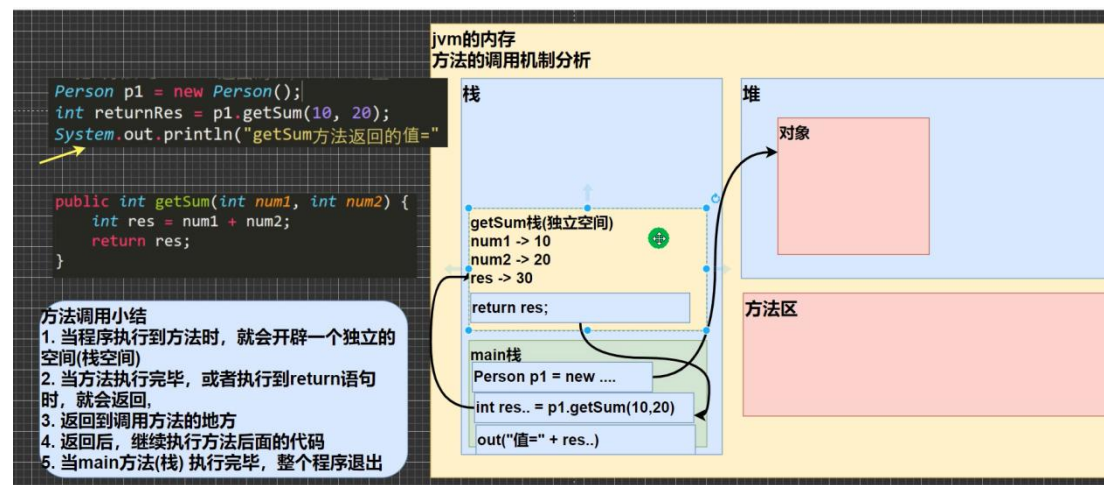
类与对象

- 类和对象的内存分配机制
 - ✓ Java内存的结构分析
 1. 栈：一般存放基本数据类型(局部变量)
 2. 堆：存放对象(Cat cat, 数组等)
 3. 方法区：常量池(常量, 比如字符串), 类加载信息
 4. 示意图 [Cat (name, age, price)]
- ✓ Java创建对象的流程简单分析

```
Person p = new Person();
p.name = "jack";
p.age = 10
```

 1. 先加载Person类信息(属性和方法信息, 只会加载一次)
 2. 在堆中分配空间, 进行默认初始化(看规则)
 3. 把地址赋给 p, p 就指向对象
 4. 进行指定初始化, 比如 p.name = "jack" p.age = 10

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

成员方法

- 成员方法的定义

```
public 返回数据类型 方法名 (形参列表..) { //方法体
    语句;
    return 返回值;
}
```

1. 参数列表：表示成员方法输入 cal(int n)
2. 数据类型 (返回类型)：表示成员方法输出, void 表示没有返回值
3. 方法主体：表示为了实现某一功能代码块
4. return 语句不是必须的。
5. 老韩提示：结合前面的题示意图, 来理解

方法细节

● 注意事项和使用细节

MethodDetail.java

✓ 访问修饰符 (作用是控制 方法使用的范围)

如果不写默认访问, [有四种: ?], 具体在后面说

✓ 返回类型

1. 一个方法最多有一个返回值 [思考, 如何返回多个结果?]

2. 返回类型可以为任意类型, 包含基本类型或引用类型(数组, 对象)

3. 如果方法要求有**返回数据类型**, 则方法体中最后的执行语句必须为 **return 值**; 而且要求返回值类型必须和return的值类型一致或兼容

4. 如果方法是**void**, 则方法体中可以没有return语句, 或者 只写 return ;

✓ 方法名

遵循驼峰命名法, 最好见名知义, 表达出该功能的意思即可, 比如 得到两个数的和 getSum, 开发中按照规范

● 注意事项和使用细节

✓ 参数列表

1. 一个方法可以有0个参数, 也可以有多个参数, 中间用逗号隔开, 比如 **getSum(int n1,int n2)**

2. 参数类型可以为任意类型, 包含基本类型或引用类型, 比如 **printArr(int[][] map)**

3. 调用带参数的方法时, 一定对应着参数列表传入相同类型或兼容类型 的参数! **【getSum】**

4. 方法定义时的参数称为形式参数, 简称形参; 方法调用时的参数称为实际参数, 简称实参, 实参和形参的类型要一致或兼容、个数、顺序必须一致! [演示]

✓ 方法体

里面写成功能的具体的语句, 可以为输入、输出、变量、运算、分支、循环、方法调用, 但里面不能再定义方法! 即: 方法不能嵌套定义。[演示]

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

● 注意事项和使用细节

✓ 方法细节调用说明

1. 同一个类中的方法调用: 直接调用即可。比如 **print(参数);**

案例演示: A类 sayOk 调用 print()

2. 跨类中的方法A类调用B类方法: 需要通过对象名调用。比如 **对象名.方法名(参数);** 案例演示: B类 sayHello 调用 print()

3. 特别说明一下: 跨类的方法调用和方法的访问修饰符相关, 先暂时这么提一下, 后面我们讲到访问修饰符时, 还要再细说。

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）

The diagram shows the memory state during a value parameter pass. It is divided into two main regions: **main** (main method) and **swap** (swap method).

- main method:** Contains variables `a = 10` and `b = 20`. A call to `obj.swap(a, b)` is shown. The state is labeled (1) with `a=10 b=20`.
- swap method:** Created when `swap` is called. It has its own local variables `a` and `b`. The state is labeled (2) with `a=20 b=10`. It performs the swap logic using a temporary variable `tmp`.
- Return:** After the swap method finishes, it returns control to the main method. The state in main is now labeled (3) with `a=10 b=20`, showing that the original values are restored.

Handwritten notes in Chinese: "内存" (Memory), "堆" (Heap), "方法区" (Method Area), "栈" (Stack).

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）

The diagram shows the memory state during a reference parameter pass. It is divided into two main regions: **main** (main method) and **test100** (test100 method).

- main method:** Contains a reference variable `arr` pointing to an array object in memory (address 0x1122). The state is labeled (1) with `arr -> [0x1122]`.
- test100 method:** Created when `test100` is called. It receives the reference `arr` and points to the same array object in memory. The state is labeled (2) with `arr -> [0x1122]`.
- Modification:** The `test100` method modifies the array element at index 0 (e.g., `arr[0] = 200`). This change is reflected in the array object in memory.
- Return:** After the `test100` method finishes, it returns control to the main method. The state in main is now labeled (3) with `arr -> [0x1122]`, showing that the array has been modified.

Handwritten notes in Chinese: "内存" (Memory), "堆" (Heap), "方法区" (Method Area), "栈" (Stack).

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

```

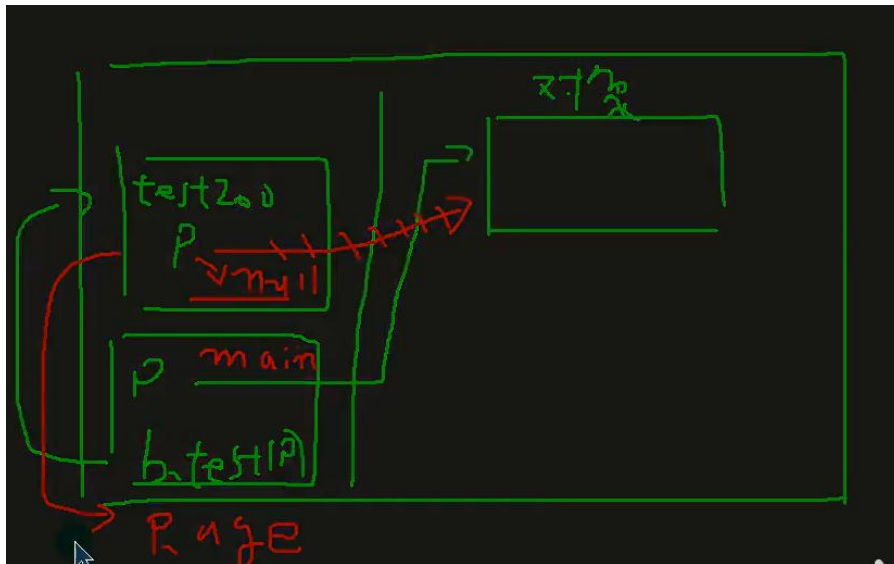
        p.name = "jack";
        p.age = 10;

        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;
}

```

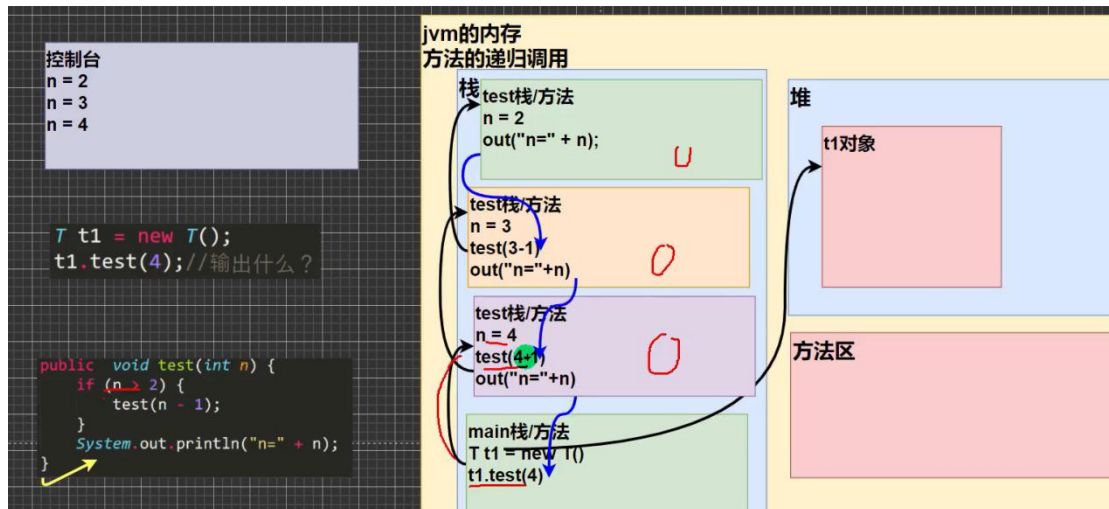
还是返回 10 因为 `new` 给创建新对象在堆里开辟了新空间 还是不影响 `main` 中的变量

方法递归调用

- 基本介绍

简单的说: 递归就是方法自己调用自己,每次调用时传入不同的变量.递归有助于编程者解决复杂问题,同时可以让代码变得简洁

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

方法重载(OverLoad)

- 基本介绍

java中允许同一个类中, 多个同名方法的存在, 但要求 形参列表不一致!
比如: `System.out.println()`; `out`是`PrintStream`类型

- OverLoad01.java

- 重载的好处

- 1) 减轻了起名的麻烦
- 2) 减轻了记名的麻烦

方法重载(OverLoad)

- 注意事项和使用细节

- 1) 方法名: 必须相同
- 2) 参数列表: 必须不同 (参数类型或个数或顺序, 至少有一样不同, 参数名无要求)
- 3) 返回类型: 无要求

可变参数

- 基本概念

java允许将同一个类中多个同名同功能但参数个数不同的方法，封装成一个方法。

- 基本语法

访问修饰符 返回类型 方法名(数据类型... 形参名) {
}

- 快速入门案例(VarParameter01.java)

看一个案例 类 HspMethod, 方法 sum 【可以计算 2个数的和, 3个数的和, 4. 5, ...】

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));  
}  
  
class 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;  
    }  
}
```

可变参数

- 注意事项和使用细节

VarParameterDetail.java

- 1) 可变参数的实参可以为0个或任意多个。
- 2) 可变参数的实参可以为数组。
- 3) 可变参数的本质就是数组。
- 4) 可变参数可以和普通类型的参数一起放在形参列表，但必须保证可变参数在最后
- 5) 一个形参列表中只能出现一个可变参数

作用域:

作用域。

- 基本使用

面向对象中，变量作用域是非常重要知识点，相对来说不是特别好理解，请大家注意听，认真思考，要求深刻掌握变量作用域。Scope01.java

1. 在java编程中，主要的变量就是属性(成员变量)和局部变量。
2. 我们说的局部变量一般是指在成员方法中定义的变量。【举例 Cat类: cry】
3. java中作用域的分类
全局变量：也就是属性，作用域为整个类体 Cat类: cry eat 等方法使用属性
【举例】
局部变量：也就是除了属性之外的其他变量，作用域为定义它的代码块中！
4. 全局变量可以不赋值，直接使用，因为有默认值，局部变量必须赋值后，才能使用，因为没有默认值。【举例】

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

作用域

- 注意事项和细节使用

VarScopeDetail.java

1. 属性和局部变量可以重名，访问时遵循就近原则。
2. 在同一个作用域中，比如在同一个成员方法中，两个局部变量，不能重名。[举例]
3. **属性生命周期较长**，伴随着对象的创建而创建，伴随着对象的销毁而销毁。局部变量，生命周期较短，伴随着它的代码块的执行而创建，伴随着代码块的结束而销毁。即在一次方法调用过程中。

就近原则举例

```
String name = "jack";

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

输出 king

4. **作用域不同**
全局变量：可以被本类使用，或其他类使用（通过对象调用）
局部变量：只能在本类中对应的方法中使用
5. **修饰符不同**
全局变量/属性可以加修饰符
局部变量不可以加修饰符

构造器/构造方法

- 基本介绍

构造方法又叫构造器(constructor)，是类的一种特殊的方法，它的主要作用是完成对**新对象的初始化**。它有几个特点：

- 1) 方法名和类名相同
- 2) 没有返回值
- 3) 在创建对象时，系统会自动的调用该类的构造器完成对对象的初始化。

- 基本语法
- [修饰符] 方法名(形参列表){
 方法体;
}

老韩说明：

- 1) 构造器的修饰符可以默认，也可以是public protected private
- 2) 构造器没有返回值
- 3) 方法名 和类名字必须一样
- 4) 参数列表 和 成员方法一样的规则
- 5) 构造器的调用系统完成

● 注意事项和使用细节

ConstructorDetail.java

1. 一个类可以定义多个不同的构造器, 即构造器重载
比如: 我们可以再给Person类定义一个构造器, 用来创建对象的时候, 只指定人名, 不需要指定年龄
2. 构造器名和类名要相同
3. 构造器没有返回值
4. 构造器是完成对象的初始化, 并不是创建对象
5. 在创建对象时, 系统自动的调用该类的构造方法
6. 如果程序员没有定义构造方法, 系统会自动给类生成一个默认无参构造方法(也叫默认构造方法), 比如 `Person (){}` , 使用javap指令 反编译看看
7. 一旦定义了自己的构造器, 默认的构造器就覆盖了, 就不能再使用默认的了, 除非显式的定义一下, 即: `Person(){}`

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。）

this的注意事项和使用细节

ThisDetail.java

1. this关键字可以用来访问本类的属性、方法、构造器
2. this用于区分当前类的属性和局部变量
3. 访问成员方法的语法：this.方法名(参数列表);
4. 访问构造器语法：this(参数列表); **注意只能在构造器中使用**
5. this不能在类定义的外部使用，只能在类定义的方法中使用。

同类下一个构造器去访问另一个构造器用 this（复用构造器）只能复用一個构造器

```
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。。调換