进制

对于整数有四种表达方式：

1. 二进制：0,1，满 2 进 1.以 0b 或 0B 开头；
1. 十进制：0-9，满 10 进 1；
2. 八进制：0-7，满 8 进 1.以数字 0 开头；
3. 十六进制：0-9 及 A(10)-F(15)，满 16 进 1.以 0x 或 0X 开头表示。此处的 A-F 不区分大小写。

十进制	十六进制	八进制	二进制
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111
16	10	20	10000
17	11	21	10001

进制的转换：

1.二进制转十进制：从最低位（右边）开始，将每个位上的数提取出来，乘以 2 的（位数-1）次方，然后求和。

2.八进制转十进制：从最低位（右边）开始，将每个位上的数提取出来，乘以 8 的（位数-1）次方，然后求和。

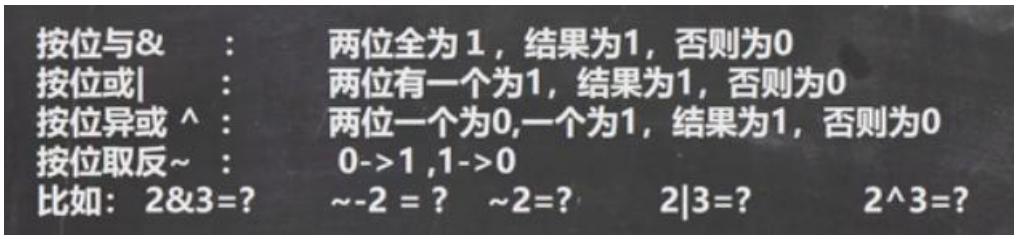
3.十六进制转十进制：从最低位（右边）开始，将每个位上的数提取出来，乘以

16 的（位数-1）次方，然后求和。

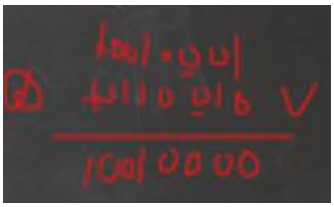
4.十进制转二进制：将该数不断除以 2，直到商为 1 为止，然后将每一步得到的余数倒过来就是对应的二进制。
5.十进制转八进制：将该数不断除以 8，直到商为 1 为止，然后将每一步得到的余数倒过来就是对应的二进制。
6.十进制转十六进制：将该数不断除以 16，直到商为 1 为止，然后将每一步得到的余数倒过来就是对应的二进制。
7.二进制转八进制：从低位开始，将二进制数每三位一组，转成八进制数即可。
0b11(3)010(2)101(5)=>0325　　0b11(3)100(4)101(5)=>0345
8.二进制转为十六进制：从低位开始，将二进制数每四位一组，转成八进制数即可。
0b1101(13=>D)0101(5)=>D5　0b11(3)1001(9)0110(6)=>0x396
9.八进制转为二进制：将八进制数每 1 位，转成对应的一个 3 位的二进制数即可。
02(010)3(011)7(111)=>0b010011111
10.十六进制转为二进制：将十六进制数每 1 位，转成对应的一个 4 位的二进制数即可。
0x2(0010)3(0011)B(1011)=>0b001000111011

原码、反码、补码（重难点）
对于有符号的数而言：
1. 二进制的最高位是符号位：0 表示正数，1 表示负数；
2. 正数的原码、反码、补码都一样（三码合一）；
3. 负数的反码 = 它的原码符号位不变，其他位取反（0->1,1->0）；
4. 负数的补码 = 负数的反码 + 1　，　负数的反码 = 负数的补码 - 1；
5. 0 的反码、补码都 0；
6. java 没有无符号数，换言之，java 中的所有数都是有符号的；
7. 在计算机运算时，都是以补码的方式来运算的；（因为补码把正数和负数统一起来了）
8. 当我们看运算结果时，要看他的原码（重点！）。

位运算符：java 中有 7 个位运算符（&、|、^、~、>>、<<、和>>>）
分别是：按位与&、按位或|、按位异或^、按位取反~，其运算规则为：



按位与&：相应的位置上对应的数都为 1 结果为 1；反之为 0。其他的类似。



例如：2&3 的结果：（第一位为符号位，符号位为 0 则为正数，为 1 则为负数）

```
public class BitOperator{
    public static void main(String[] args){
        //1.先得到 2 的补码 => 2 的原码为：00000000 00000000 00000000 00000010
        //  2 是正数所以补码和原码一样，2 的补码为：00000000 00000000 00000000 00000010
        //2.3 的补码 => 3 的原码为：00000000 00000000 00000000 00000011
        //  3 是正数所以补码和原码一样，3 的补码为：00000000 00000000 00000000 00000011
        //3.按位&
        //  00000000 00000000 00000000 00000010
        //  00000000 00000000 00000000 00000011
        //  00000000 00000000 00000000 00000010 &运算后的结果(补码)
        //  由于是正数则运算后的结果的原码也为：00000000 00000000 00000000 00000010
        //  则对应的十进制结果是：2
        System.out.println(2&3);//2
    }
}
```

~-2 的结果：

```
//~-2
        //1.先得到-2 的原码  10000000 00000000 00000000 00000010
        //2.再得到-2 的反码  11111111 11111111 11111111 11111101
        //3.再得到-2 的补码  11111111 11111111 11111111 11111110
        //4.~-2 操作          00000000 00000000 00000000 00000001 运算后的结
```

果(补码)

//由于是正数则运算后的结果的原码也为：00000000 00000000 00000000 00000001

//则对应的十进制结果是：1

System.out.println(~-2);//1

~2 的结果：

//~2

//1.先得到 2 的原码=补码 00000000 00000000 00000000 00000010

//2.再进行~2 操作得到 11111111 11111111 11111111 11111101 运算后的结果(补码)

//3.将补码转为反码 11111111 11111111 11111111 11111100 (补码-1)

//4.将反码转为原码 10000000 00000000 00000000 00000011 (符号位不变，其他位取反)

//5.所以结果为-3

System.out.println(~2);//-3

2|3 的结果：

先得到 2 的补码=原码：00000000 00000000 00000000 00000010

再得到 3 的补码=原码：00000000 00000000 00000000 00000011

再进行运算： 00000000 00000000 00000000 00000011（该结果为补码）

由于结果为正数则补码=原码：00000000 00000000 00000000 00000011

结果为 3

2^3 的结果：

先得到 2 的补码=原码：00000000 00000000 00000000 00000010

再得到 3 的补码=原码：00000000 00000000 00000000 00000011

再进行运算： 00000000 00000000 00000000 00000001（该结果为补码）

由于结果为正数则补码=原码：00000000 00000000 00000000 00000001

结果为 1

~-5 的结果：

先得到-5 的原码：10000000 00000000 00000000 00000101

再得到-5 的反码：11111111 11111111 11111111 11111010

再得到-5 的补码：11111111 11111111 11111111 11111011

再进行运算： 00000000 00000000 00000000 00000100（该结果为补码）

由于结果为正数则补码=原码：00000000 00000000 00000000 00000100

结果为 4

13&7 的结果：

先得到 13 的原码=补码：00000000 00000000 00000000 00001101

再得到 7 的原码=补码： 00000000 00000000 00000000 00000111

再进行运算： 00000000 00000000 00000000 00000101

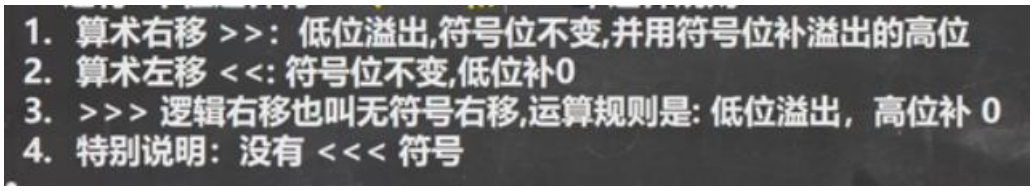
由于结果为正数则补码=原码：00000000 00000000 00000000 00000101

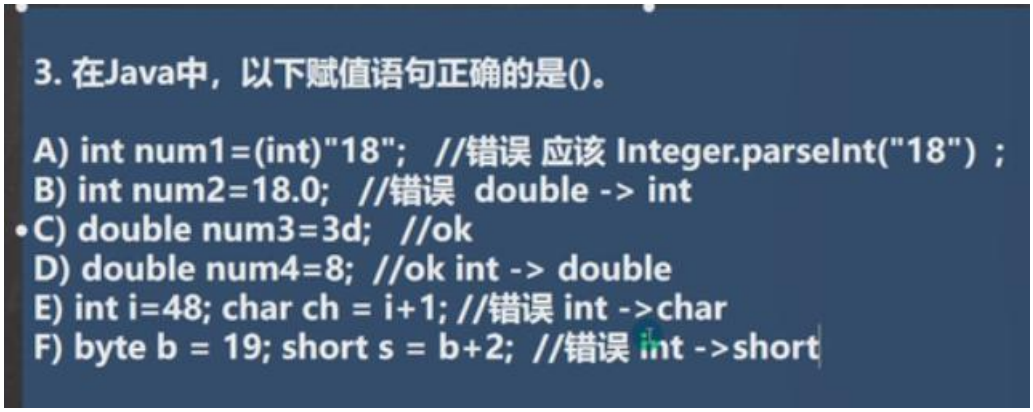
结果为 5

5|4 的结果：
先得到 5 的原码=补码：00000000 00000000 00000000 00000101
再得到 4 的原码=补码：00000000 00000000 00000000 00000100
再进行运算：　　00000000 00000000 00000000 00000101
由于结果为正数则补码=原码：00000000 00000000 00000000 00000101
结果为 5

-3^3 的结果：
先得到-3 的原码：　　10000000 00000000 00000000 00000011
再得到-3 的反码：　　11111111 11111111 11111111 11111100
再得到-3 的补码：　　11111111 11111111 11111111 11111101
再得到 3 的原码=补码：00000000 00000000 00000000 00000011
再进行运算：　　11111111 11111111 11111111 11111110（该结果为补码）
由于结果为负数，则补码对应的反码为：11111111 11111111 11111111 11111101
反码对应的原码为：10000000 00000000 00000000 00000010
则结果为-2

位运算另外三个运算符>>、<<、和>>>，运算规则：





第四章 程序控制结构

顺序控制：程序从上到下逐行执行，中间没有任何的判断和跳转
Java 中定义变量时采用合法的前向引用。

分支控制 if-else：
1）单分支
基本语法：
```
if(条件表达式){
        执行代码块：(可以有多条语句)
}
```
说明：当条件表达式为 true 时，就会执行{}内的代码。如果为 false，就不执行。（注：如果{}中只用一条语句，则可以不用{}）

2）双分支
基本语法：
```
if(条件表达式){
        执行代码块 1：(可以有多条语句)
}
else{
        执行代码块 2;
}
```

3）多分支
基本语法：
```
if(条件表达式 1){
        执行代码块 1：(可以有多条语句)
}
else if(条件表达式 2){
        执行代码块 2;
}
.....
else{
        执行代码块 n;
}
```
多分支可以没有 else，如果所有的表达式都不成立，则一个执行入口都没有。

嵌套分支：在一个分支结构中又完整地嵌套了另一个完整的分支结构，里面的分支结构成为内层分支外面的分支结构称为外层分支。（最好不要超过 3 层，可读性不强）

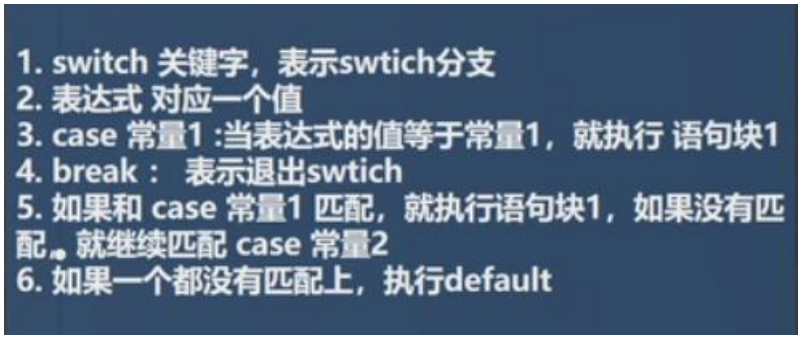
基本语法：
```
if(){
    if(){
```

```
            //if-else.....
        }else{
            //if-else
        }
}
```

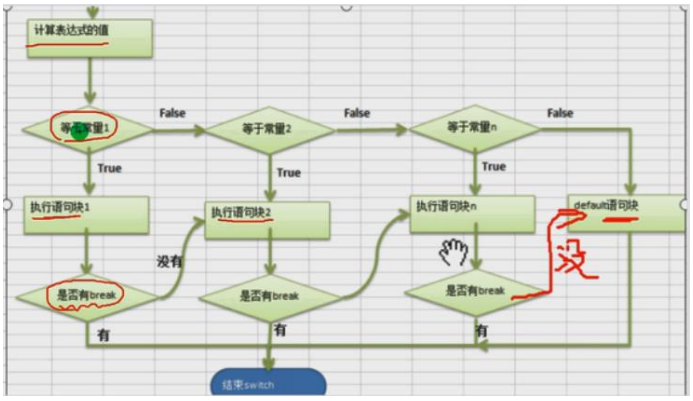
switch 分支结构
基本语法：

```
switch(表达式){
        case  常量 1: //当...
        语句块 1;
        break;//结束 switch
        case  常量 2: //当...
        语句块 2;
        break;
        ...
        case  常量 n: //当...
        语句块 n;
        break;
        default;//之前的都没有匹配上就执行 default
        default 语句块;
        break;
}
```

注意：



switch 语句执行流程图：



switch 注意事项和细节：
1. 表达式数据类型，应该和 case 后的常量类型一致，或者是可以自动转成可以相互比较的类型，比如输入的是字符串，而常量是 int；
2. switch(表达式)中表达式的返回值必须是：byte,short,int,char,enum[枚举],String；
3. case 子句中的值必须是常量，而不能是变量；
4. default 子句是可选的，当没有匹配的 case 时，执行 default；
5. break 语句用来在执行完一个 case 分支后使得程序跳出 switch 语句块；如果没有写 break，程序会顺序执行到 switch 结尾，除非遇到 break。

使用穿透：

```
//使用穿透
//3,4,5 春季，6,7,8 夏季，9,10,11 秋季，12,1,2 冬季
System.out.println("请输入月份：  ");
int month = myscanner.nextInt();
switch(month){
    case 3:
    case 4:
    case 5:
        System.out.println("春季");
        break;
    case 6:
    case 7:
    case 8:
        System.out.println("夏季");
        break;
    case 9:
    case 10:
    case 11:
        System.out.println("秋季");
        break;
    case 12:
    case 1:
    case 2:
        System.out.println("冬季");
        break;
    default:
        System.out.println("输入有误");
        break;
}
```

switch 语句和 if 语句的对比：
1. 如果判断的具体数值不多，而且符合 byte,short,int,char,enum[枚举],String 这六种类型。虽然两个语句都可以使用，但建议使用 switch 语句；

2. 其他情况：对区间判断，对结果为 boolean 类型判断，使用 if。

for 循环控制：
基本语法：
for(循环变量初始化;循环条件;循环变量迭代){
　　循环操作（语句）;
}
注：
1. for 关键字，表示循环控制；
2. for 有四要素：（1）循环变量初始化（2）循环条件（3）循环操作（4）循环变量迭代；
3. 循环操作：这里可以有多条语句，也就是要执行的代码；
4. 如果循环操作（语句）只有一条语句，可以省略{}。（但尽量别省）

for 循环执行流程：
```
for(int i=1;i<=80;i++){
　　System.out.println(“nihao”);
}
```

for 循环的注意事项和细节说明：
1）循环条件是返回一个布尔值的表达式；
2）for(;循环判断条件;)中的初始化和变量迭代可以写到其他地方，但是两边的分号不能省略；
3）循环初始值可以有多条初始化语句，但要求类型一样，并且中间用逗号隔开。

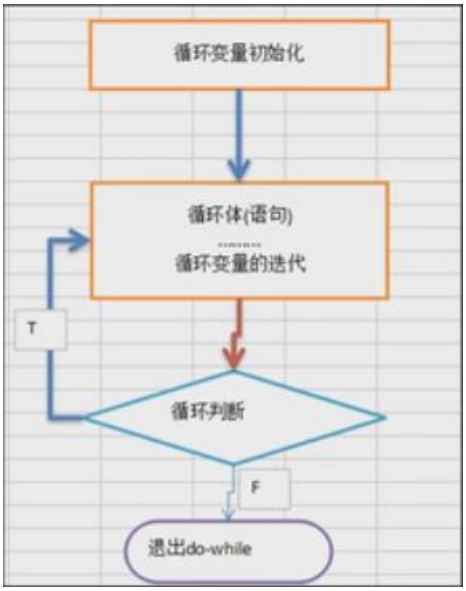
while 循环控制：
基本语法：
```
while(循环条件){
　　循环体(语句);
　　循环变量迭代;
```

}
说明：1）while 循环也有四要素；2）只是四要素的位置和 for 不一样。

注意：
1. 循环条件是返回一个布尔值的表达式；
2. while 循环是先判断再执行语句。

do..while 循环控制：
基本语法：
do{
　　循环体（语句）;
　　循环变量迭代;
}while(循环条件);

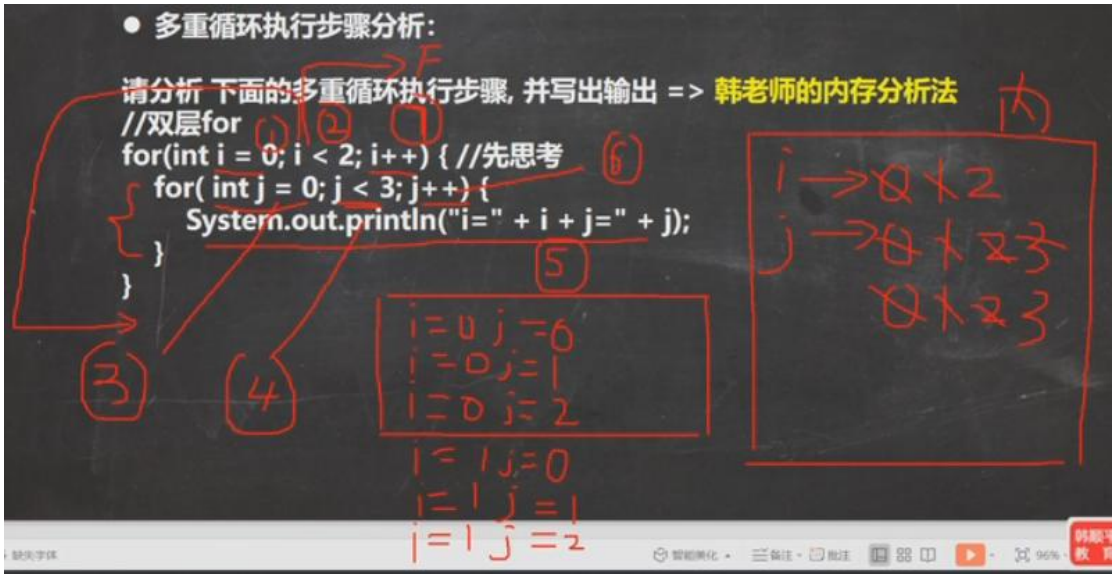
注：
1. do...while 是关键字；
2. 也有循环四要素，只是位置不一样；
3. 先执行，再判断；也就是说一定会执行一次；
4. 最后只有一个分号；
5. while 和 do..while 区别：while 是先判断再执行，而 do..while 是先执行再判断。



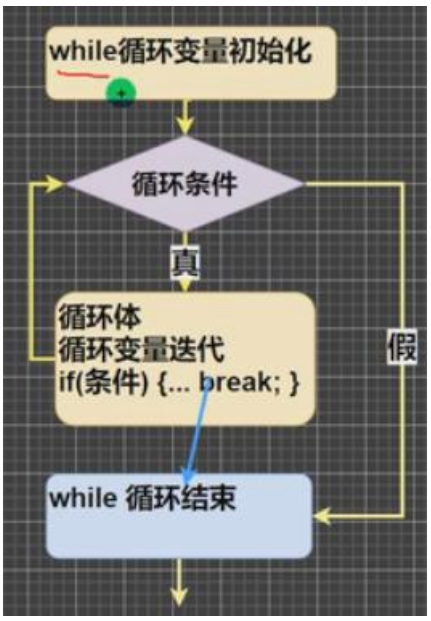
多重循环控制（重难点）

实质上，嵌套循环就是把内层循环当成外层循环的循环体。当只有内层循环的循环条件为 false 时，才会完全跳出内层循环，才结束外层的当次循环，开始下一次的循环。

当外层循环次数为 m 次，内层为 n 次，则内层循环体实际上需要执行 m*n 次。



跳转控制语句-break
通过循环来控制，但是不知道要循环多少次，使用 break，当某个条件满足时，终止循环，通过该需求可以说明其他流程控制的必要性。



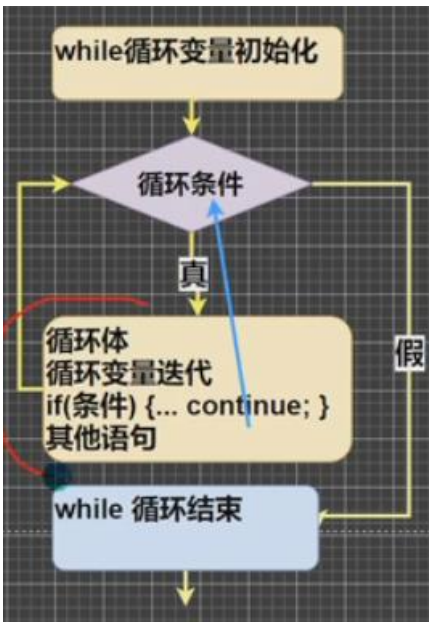
break 注意事项和细节：
1. break 语句出现在多层嵌套的语句块时，可以通过标签指明要终止的是那一层语句块
2. 标签的基本使用：

```
label1:{    ...
label2:         {     ...
label3:                   {         ...
                                    break;
                                ...
                          }
                 }
        }
```

（1）break 语句可以指定退出哪一层；
（2）label1 是标签，名字可以自己指定；
（3）break 后指定到哪个标签就退出到哪里；
（4）但在实际开发中尽量不使用；
（5）如果没有指定 break，默认退出到最近的循环体。

字符串（String）的内容比较方法：equals 方法："".equals(name)【可以避免空指针】

跳转控制语句-continue
1）continue 语句用于结束本次循环，继续执行下一次循环
2）continue 语句出现在多层嵌套的循环语句体中时，可以通过标签指明要跳过的是哪一层循环，这个和前面的标签使用的规则一样
3）基本语法：{...continue...}




```
label1:
for(int j = 0; j < 4; j++){
        label2:
        for(int i = 0; i < 10; i++){
                if(i == 2){
                        //看看分别输出什么值，并分析
                        //continue ;
                        //continue label2;
                        continue label1;
                }
                System.out.println("i = " + i);
        }
}
```

跳转控制语句-return

表示跳出所在的方法（注意：如果 return 写在 main 方法则会直接退出程序，不再执行其下面的语句）

Homework01

```
public static void main(String[] args) {
    /*
    某人有100,000元,每经过一次路口, 需要交费,规则如下:
    1) 当现金>50000时,每次交5%
    2) 当现金<=50000时,每次交1000
    编程计算该人可以经过多少次路口, 要求: 使用 while + break方式完成

    思路分析
    1. 定义 double money 保存 100000
    2. 根据题的要求, 我们分析出来有三种情况
        money > 50000
        money >=1000 && money <= 50000
        money < 1000
    3. 使用多分支 if-elseif-else
    4. while+break[money < 1000], 同时使用一个变量count来保存通过路口
    代码实现
     */
```

第五章 数组、排序

数组：可以存放多个同一类型的数据，数组也是一种数据类型，是引用类型（数组就是一组数据）

Array01.java

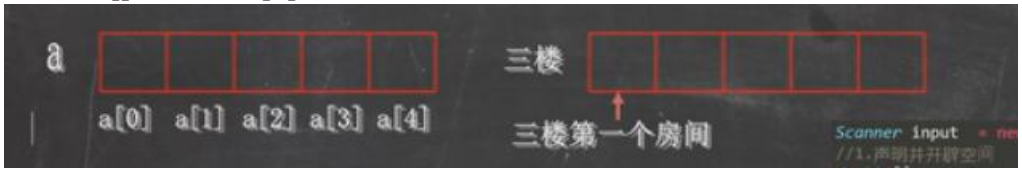
```
public class Array01{
    public static void main(String[] args){
        //定义一个数组
        //1. double[] 表示是 double 类型的数组，数组名为 hens
        //2. {3,4,1,3.4,2,50} 表示数组的值/元素，依次表示是数组的第几个元素
        double[] hens = {3,4,1,3.4,2,50};

        //遍历数组得到数组的所有元素的和，使用 for
        //1. 可以通过 hens[下标] 来访问数组的元素
        //   下标是从 0 开始编号的，比如第一个元素是 hens[0]
        //2. 通过 for 就可以循环得访问 数组的元素/值
        //3. 使用一个变量 totalWeight 将各个元素累积
        //4. 注意可以通过 数组名.length 来得到数组的大小/长度
        double totalWeight = 0;
        System.out.println("数组的长度= " + hens.length );
        for(int i=1;i<hens.length;i++){
            System.out.println("第"+ (i+1) + "个元素的值=" + hens[i]);
            totalWeight = totalWeight + hens[i];
        }
        System.out.println("总体重=" + totalWeight + "平均体重=" + (totalWeight / hens.length));
        }
    }
```

数组的使用

1. 使用方式 1-动态初始化

数组的定义：数组名[]=new 数据类型[大小]

比如：int a[] = new int[5] // 表明创建了一个数组，名字为 a，存放 5 个 int



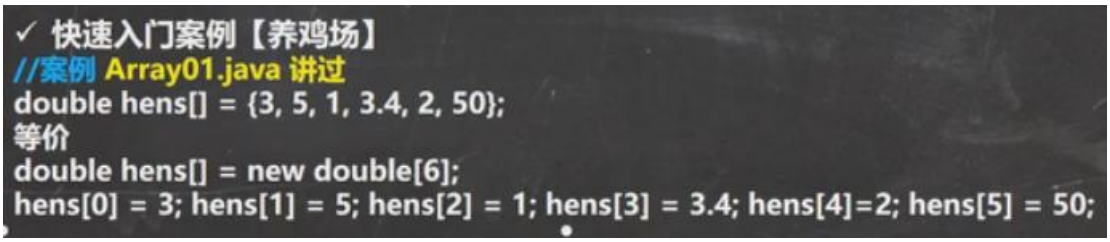
Array02.java

```
import java.util.Scanner;
public class Array02{
    public static void main(String[] args){
        //创建一个 double 数组，大小为 5
        double totalgrade[] = new double[5];
```

```
            Scanner myscanner = new Scanner(System.in);
            //循环输入
            for(int i=0 ; i < totalgrade.length ; i++){
                System.out.println("请输入第"+(i+1)+"个元素的值");
                totalgrade[i] = myscanner.nextDouble();
            }
            System.out.println("输出数组的值");
            for(int i=0 ; i < totalgrade.length ; i++){
                System.out.println("第"+(i+1)+"个元素的值= "+totalgrade[i]);
            }
        }
}
```

2. 使用方式 2-动态初始化
先声明数组：数据类型 数组名[]；也可以 数据类型[] 数组名；
int a[]; 或者 int[] a;
创建数组：数组名 = new 数据类型[大小];
a = new int[10];

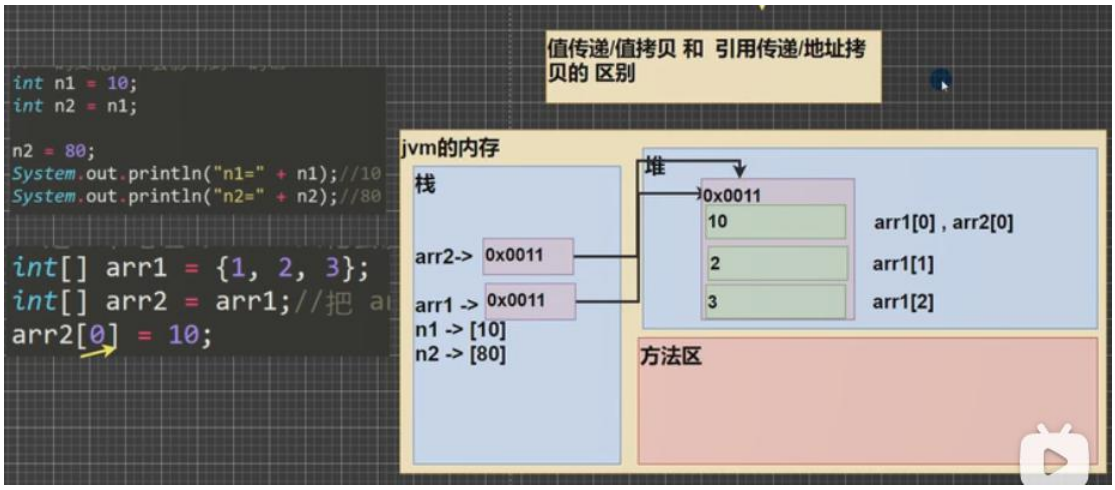
3. 使用方法 3-静态初始化
初始化数组：数据类型 数组名[]={元素值,元素值...}
int a[] = {2,5,6,7,8,89,98,53,43}，如果知道数组有多少元素，具体见上面的用法，
相当于：int a[] = new int[9];



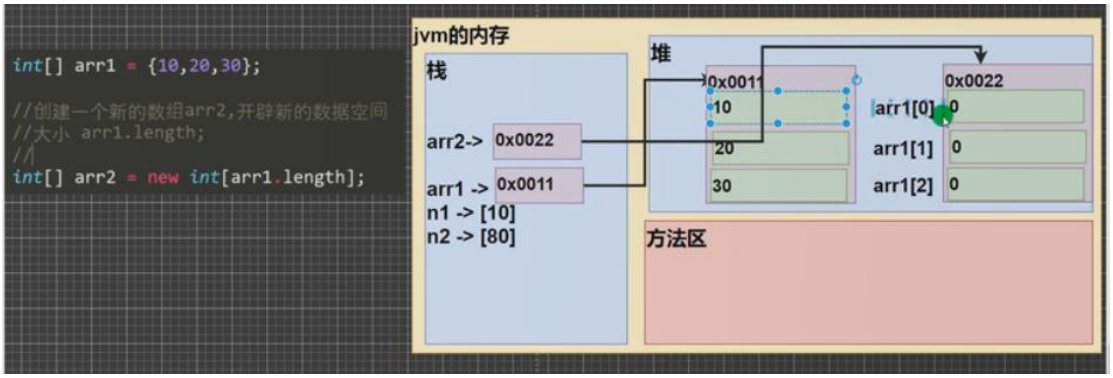
数组使用注意事项和细节：
1. 数组是多个相同类型数据的组合，实现对这些数据的统一管理；
2. 数组中的元素可以是任何数据类型，包括基本类型和引用类型，但不能混用；
3. 数组创建后，如果没有赋值，有默认值 int 0，short 0，byte 0，long 0，float 0.0，double 0.0，char \u0000，boolean false，String null；
4. 使用数组的步骤：（1）声明数组并开辟空间；（2）给数组各个元素赋值；（3）使用数组；
5. 数组的下标是从 0 开始的；
6. 数组的下标必须在指定范围内使用，否者会报错：下标越界异常，比如 int [] arr = new int[5];则有效下标为 0-4；
7. 数组属引用类型，数组型数据是对象（object）。

数组赋值机制：
1. 基本数据类型赋值，这个值就是具体的数据，而且相互不影响

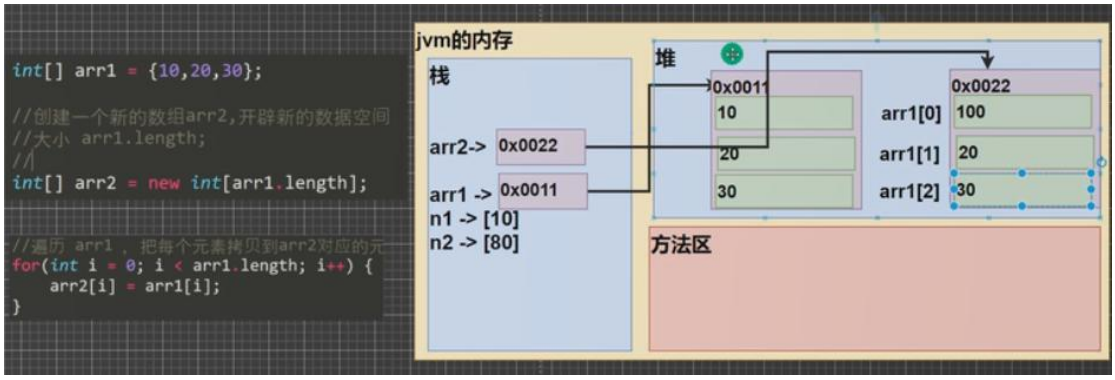
2. 数组在默认情况下是引用传递，赋的值是地址（引用传递/拷贝）
int n1 = 10; int n2 = n1; n2 = 80=>n1=10,n2=80（n2 的变化不会影响到 n1）
Int[] arr1 = {1,2,3}; int[] arr2 = arr1; arr2[0] = 10=>arr1[]={10,2,3},arr2[]={10,2,3}
（arr2[]的变化会影响到 arr1[]）



数组拷贝



```java
//遍历 arr1 ， 把每个元素拷贝到arr2对应的元素位置
for(int i = 0; i < arr1.length; i++) {
    arr2[i] = arr1[i];
}

//老师修改 arr2
arr2[0] = 100;

//输出arr1
System.out.println("====arr1的元素====");
for(int i = 0; i < arr1.length; i++) {
    System.out.println(arr1[i]);//10,20,30
}


System.out.println("====arr2的元素====");
for(int i = 0; i < arr2.length; i++) {
    System.out.println(arr2[i]);//
}
```



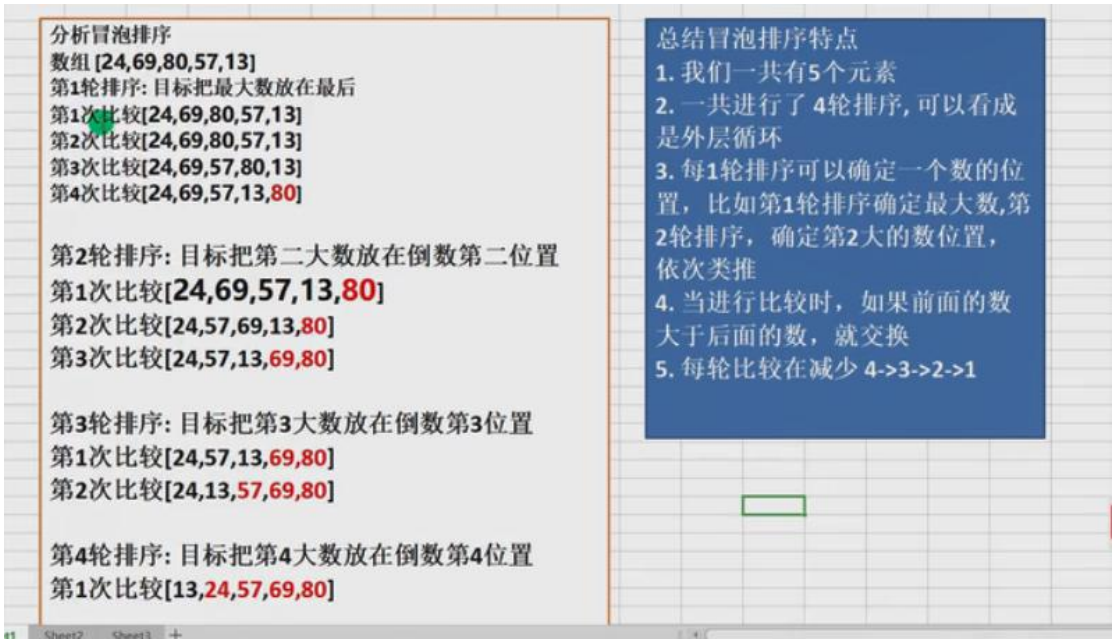
数组反转

```
//数组反转
//arr[11,22,33,44,55,66]->arr[66,55,44,33,22,11]
public class ArrayReverse{
    public static void main(String[] args){
        int chars[] = {11,22,33,44,55,66};
        //设置中间变量
        int middle = 0;
        for(int i=0;i<(chars.length/2);i++){
            middle = chars[(chars.length)-1-i];
            chars[(chars.length)-1-i] = chars[i];
            chars[i] = middle;
```

```
        }
        //输出数组的时候需要用循环
        System.out.println("==========");
        for(int i=0;i<chars.length;i++){
            System.out.println("反转后的数组 1：  " + chars[i]);
        }
        //使用逆序赋值的方法
        //
        int chars1[] = {11,22,33,44,55,66};
        int chars2[] = new int[chars.length];
        for(int j=0;j<chars1.length;j++){
            chars2[j] = chars1[chars1.length-1-j];
        }
        //输出数组的时候需要用循环
        System.out.println("==========");
        for(int j=0;j<chars2.length;j++){
            System.out.println("反转后的数组 2：  " + chars2[j]);
        }
    }
}
```

数组的排序介绍：
1. 内部排序：指将需要处理的所有数据都加载到内部存储器中进行排序，包括（交换排序法、选择式排序法和插入式排序法）；
2. 外部排序法：数据量过大，无法全部加载到内存中，需要借助外部存储进行排序，包括（合并排序法和直接合并排序法）。

冒泡排序法（Bubble Sorting）的基本思想：通过对待排序序列从后向前（从下标较大的元素开始），依次比较相邻元素的值，若发现逆序则交换，使值较大的元素逐渐从前移向后部，就像水底下的旗袍一样逐渐向上冒。



BubbleSort.java

```
public class BubbleSort{
    public static void main(String[] args){
        int arr[] = {24,69,80,57,13};
        int temp = 0;
        for(int i=0;i<arr.length-1;i++){
            for(int j=0 ; j<(arr.length-1-i) ; j++){
                //如果前面的数大于后面的数就交换位置
                if(arr[j] > arr[j+1]){
                    temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
            System.out.println("====第" + (i+1) + "轮====");
            for(int j=0;j<arr.length;j++){
                System.out.println(arr[j] + "\t");
```

```
            }
        }


    }
}
```

查找
在 java 中常用的查找方式有两种：
1. 顺序查找
2. 二分法查找

多维数组--二维数组

```
public class TwoDimensionalArray01{
    public static void main(String[] args){
        //二维数组：
        //1. 从定义形式上看 int[][]
        //2. 是原来的一维数组的每个元素是一维数组，就构成二维数组
        int[][] arr = {{0,0,0,0,0,0},{0,0,1,0,0,0},{0,2,0,3,0,0},{0,0,0,0,0,0}};
        //关于二维数组的关键概念：
        //(1)每一个一维数组就是一个二维数组的一个元素，
        //    所以这里的二维数组一共有 4 个元素
        System.out.println("二维数组的元素个数= " + arr.length);//4 个
        //(2)二维数组的每个元素是一维数组，
        //    所以如果需要得到每一个一维数组的值就需要再循环遍历
        //输出二维数组图形
        for(int i=0;i<arr.length;i++){
            //遍历二维数组的每个元素(数组)
            //1. arr[i]表示二维数组的第 i 个元素
            //2. arr[i].length 得到 对应的 每一个一维数组的长度
            for(int j=0;j<arr[i].length;j++){
                System.out.print(arr[i][j] + "\t");//输出一维数组
            }
            System.out.println();//换行
        }
    }
}
```

二维数组的使用方式：
1. 动态初始化
（1）语法：类型[][] 数组名 = new 类型[大小][大小]
（2）比如：int a[][] = new int[2][3]

2. 动态初始化（第二种）
（1）先声明：类型 数组名[][]
（2）再定义（开辟空间） 数组名 = new 类型[大小][大小]
（3）赋值（有默认值，比如 int 类型的就是 0）

```
        一个有三个一维数组，每个一维数组的元素是不一样的
        */

        int[][] arr = new int[3][]; //创建 二维数组，但是只是确定一维数组的个数
        for(int i = 0; i < arr.length; i++) {//遍历arr每个一维数组
            //给每个一维数组开空间 new
            //如果没有给一维数组 new ，那么 arr[i]就是null
            arr[i] = new int[i + 1];

            //遍历一维数组，并给一维数组的每个元素赋值
            for(int j = 0;  j < arr[i].length; j++) {
                arr[i][j] = i + 1;//赋值
            }

        }
```

3. 静态初始化

定义：类型 数组名[][] = {{值 1,值 2...},{值 1,值 2...}...{值 1,值 2...}}

TwoDimensionalArray02.java

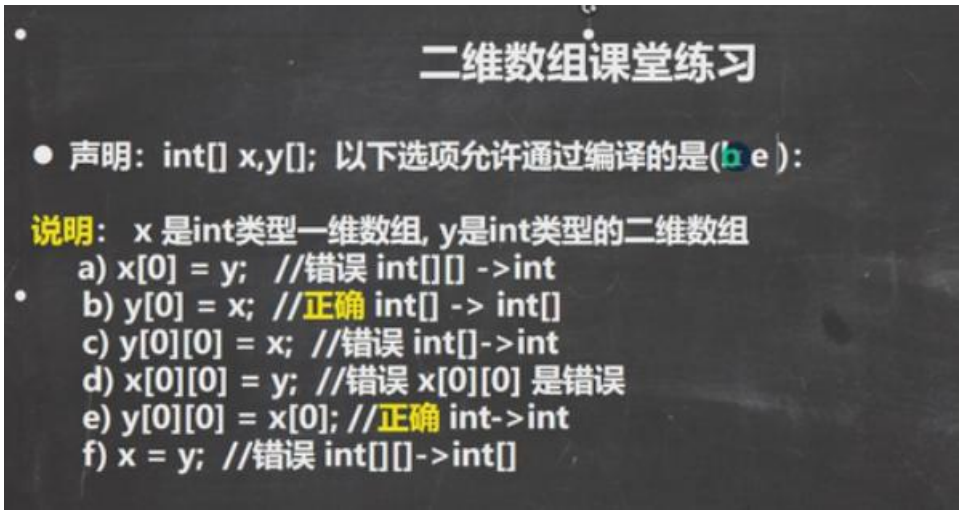
```
//遍历二维数组 arr[][] = {{4,6},{1,4,5,7},{-2}};
//并得到和
//
public class TwoDimensionalArray02{
    public static void main(String[] args){
        int arr[][] = {{4,6},{1,4,5,7},{-2}};
        int sum = 0;
        for(int i=0;i<arr.length;i++){
            for(int j=0;j<arr[i].length;j++){
                sum += arr[i][j];
            }
        }
        System.out.println("数组的元素之和为：  " + sum );
    }
}
```

杨辉三角

```
//打印杨辉三角
//1
//1 1
//1 2 1
//1 3 3 1
//1 4 6 4 1
//1 5 10 10 5 1
//1 6 15 20 15 6 1
//...
import java.util.Scanner;
public class YangHui{
    public static void main(String[] args){
        /*
        1. 第 1 行有 1 个元素，第 n 行有 n 个元素
        2. 每一行的首位的数都是 1
        3. 从第 3 行开始，对于非第一个元素和最后一个元素的值
            arr[i][j] = arr[i-1][j] + arr[i-1][j-1]
         */
        Scanner myscanner = new Scanner(System.in);
        //杨辉三角的层数
        System.out.println("请输入需要的杨辉三角的层数：  ");
        int floor = myscanner.nextInt();
        //定义二维数组
        int yanghui[][] = new int[floor][];
        for(int i=0 ; i<yanghui.length ; i++){
            //给每个一维数组(行)开空间
            yanghui[i] = new int[i+1];
            //给每个一维数组(行)赋值
            for(int j=0 ; j<yanghui[i].length ; j++){
                if( j==0 || j==yanghui[i].length-1){
                    yanghui[i][j] = 1;
                }else{
                    yanghui[i][j] = yanghui[i-1][j] + yanghui[i-1][j-1];
                }
            }
        }
        //遍历输出杨辉三角
        for(int i=0; i<yanghui.length ; i++){
            for(int j=0; j<yanghui[i].length ; j++){
                System.out.print(yanghui[i][j] + " ");
            }
            //每一行打印完换行
```

```
            System.out.println();
        }
    }
}
```

```
D:\javacode\chapter06>javac YangHui.java

D:\javacode\chapter06>java YangHui
请输入需要的杨辉三角的层数:
13
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
```



1. 下面数组定义正确的有__BD___ Homework01.java

A. String strs[] = { 'a' ,'b', 'c'}; //error, char ->String

B. String[] strs = {"a", "b", "c"}; //ok

C. String[] strs = new String{"a" "b" "c"};//error

D. String strs[] = new String[]{"a", "b", "c"};//ok

E. String[] strs = new String[3]{"a", "b", "c"};//error ,编译不通过

2. 写出结果 Homework02.java

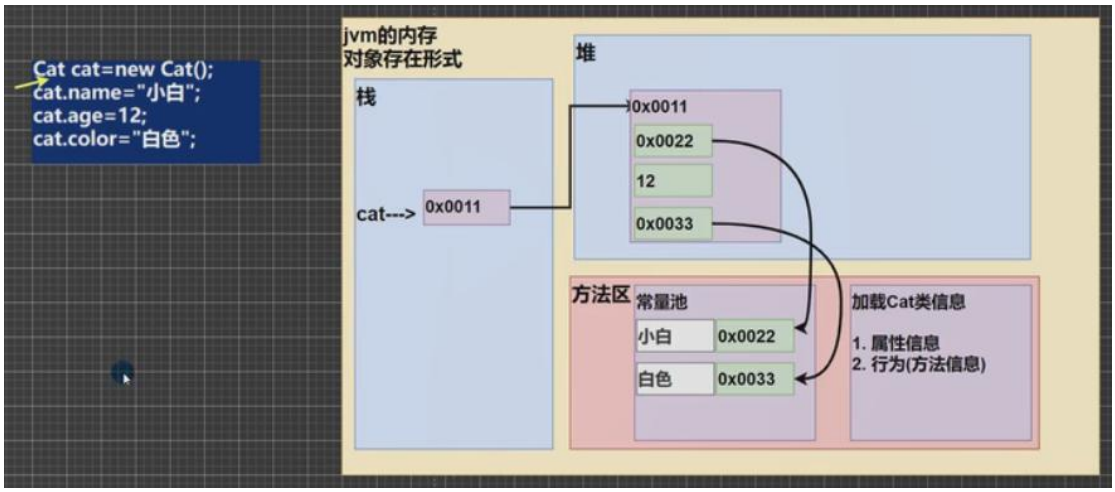
```
String foo="blue";
boolean[] bar=new boolean[2]; //bar[0]默认false bar[1] false
if(bar[0]){
        foo="green";
}
System.out.println(foo);
```

第六章 面向对象编程（基础部分）

注意：从类到对象的几种说法：1.创建一个对象；2.实例化一个对象；3.把对象实例化。

类与对象的区别与联系：
（1）类是抽象的、概念的，代表一类事物，比如人类，猫类...即它是数据类型；
（2）对象是具体的、实际的，代表一个具体的事物，即是实例；
（3）类是对象的模板，对象是类的一个个体，对应一个实例。

对象在内存中的存在形式



属性/成员变量：
1. 从概念或叫法上看：成员变量 = 属性 = field 字段（字段）（即成员变量是用来表示属性的）；-----> 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，也可以是引用对象（对象、数组）。比如之前定义的猫类 的 int age 就是属性。

注意事项：
1. 属性的定义语法同变量，比如：访问修饰符 属性类型 属性名；（访问修饰符为控制属性的访问范围；有 4 种访问修饰符：public，protected,默认,private）
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，其规则和数组一致。

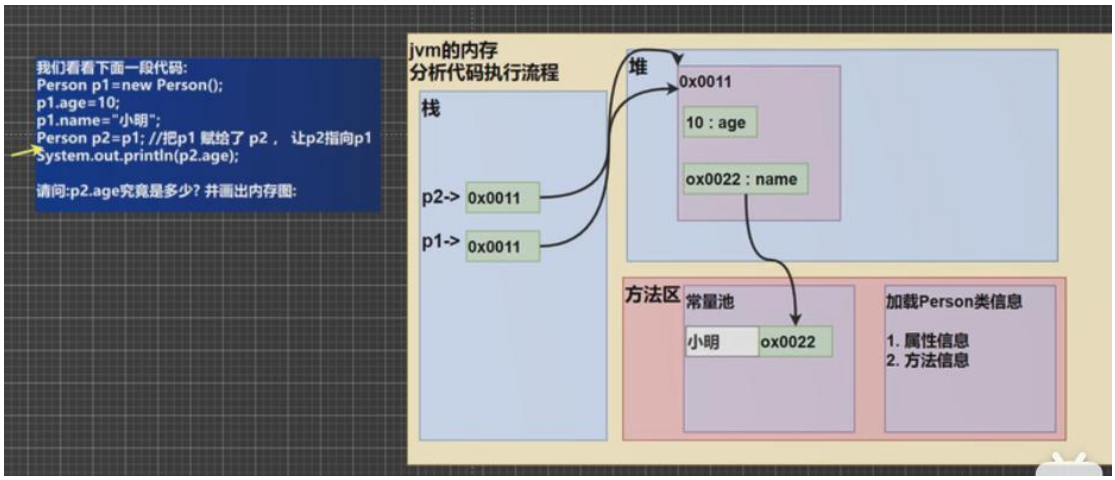
如何创建对象：
1. 先声明再创建：Cat cat;//声明一个对象为 cat | cat = new Cat();//创建
2. 直接创建：Cat cat = new Cat()

如何访问属性：
基本语法：对象名.属性名

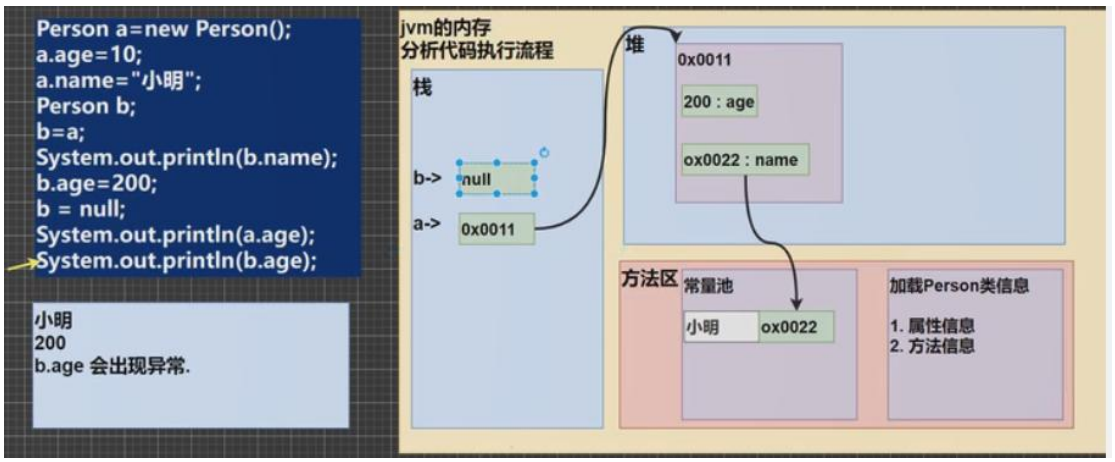
类与对象的内存分配机制

java 内存结构分析：

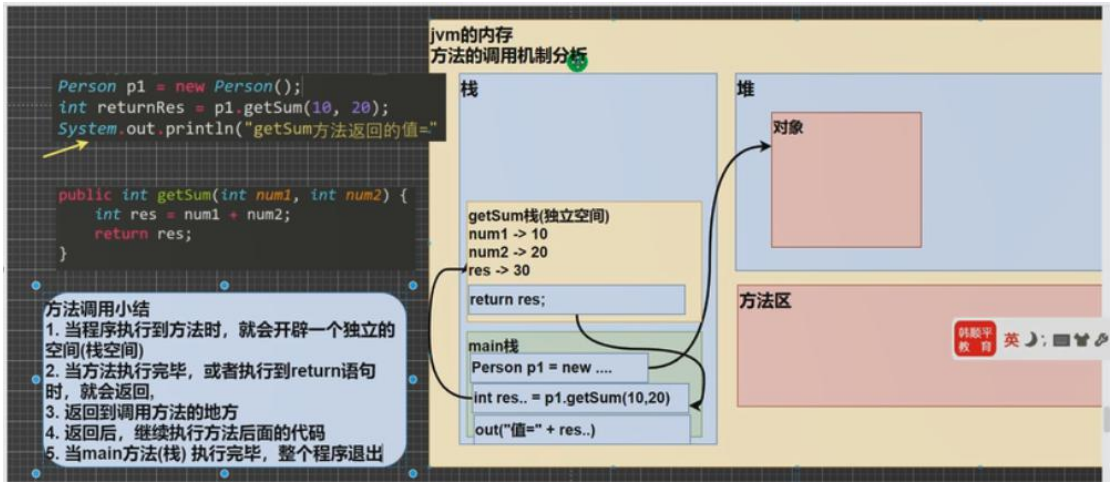
1. 栈：一般存放基本数据类型（局部变量）；
2. 堆：存放对象（Cat cat，数组等）；
3. 方法区：常量池（常量，比如字符串），类加载信息；
4. 示意图：[Cat (name,age,price)]。



b = null -> 代表 b 不是一个对象了



成员方法：
在某些情况下，我们需要定义成员方法（简称方法）。比如人类：除了有一些属性外（年龄，姓名..）人类还有一些行为比如：可以说话、跑步..，通过学习，还可以做学术题。这时就要用成员方法才能完成，需要对 person 类进行完善。



方法的好处：
1. 提高代码的复用性；
2. 可以将实现的细节封装起来，然后供其他用户来调用即可。

成员方法的定义：
public（访问修饰符） 返回数据类型 方法名 （参数列表...） {//方法体
语句;
return 返回值;
}

1. 参数列表：表示成员方法输出 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：表示未来实现某一功能代码块；
4. return 语句不是必须的；

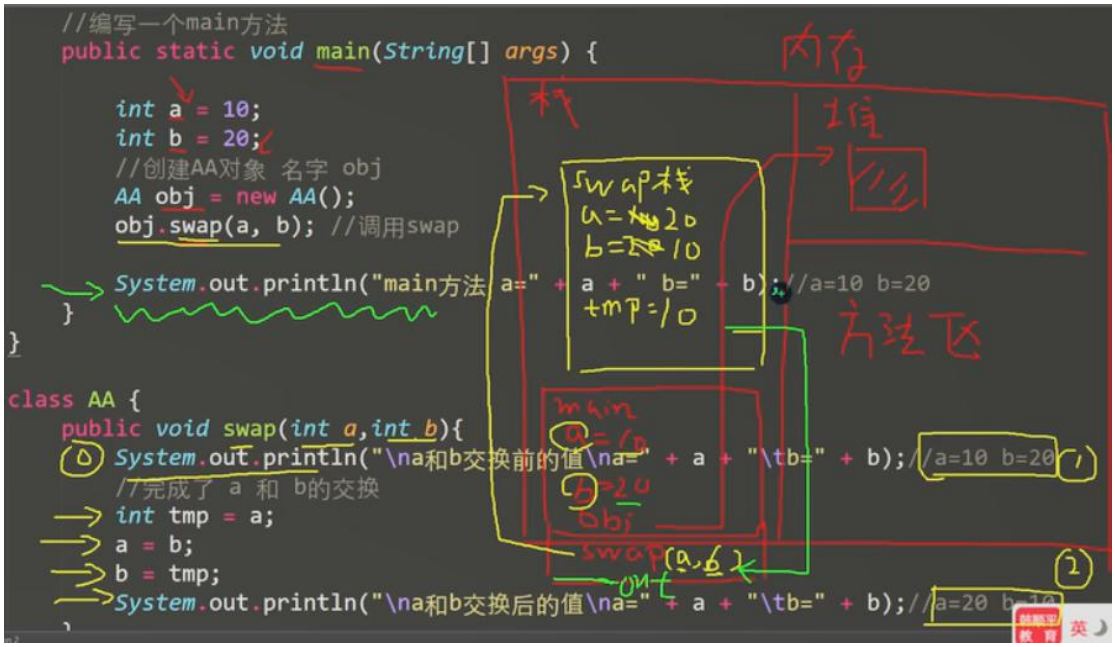
方法的注意事项和细节：
访问修饰符（public）的作用是控制方法使用的范围（如果不写，就是默认访问修饰符），有四种：public/protected/private/默认。
返回数据类型:
1. 一个方法最多有一个返回值，如果有多个返回值就返回数组；
2. 返回的类型可以为任意类型，包含基本类型或引用类型（数组、对象）；
3. 如果方法要求有返回数据类型，则方法体中最后的执行语句必须为 return 值；而且要求返回值类型必须和 return 的值类型一致或兼容；
4. 如果方法是 void，则方法体中可以没有 return 语句，或者只写 return；

方法名：
遵循驼峰命名法，最好见名知意，表达出该功能的意思即可。

形参列表：
1. 一个方法可以有一个参数也可以有多个参数，中间用逗号间隔就行；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定一致或兼容、个数、顺序必须一致

方法体：
里面写完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。

方法细节调用说明：
1. 同一类中的方法调用：直接调用即可。（不需要创建对象调用）比如 print(参数)；
2. 跨类（不同类）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（参数）；
3. 特别说明：跨类的方法调用和方法的访问修饰符相关。

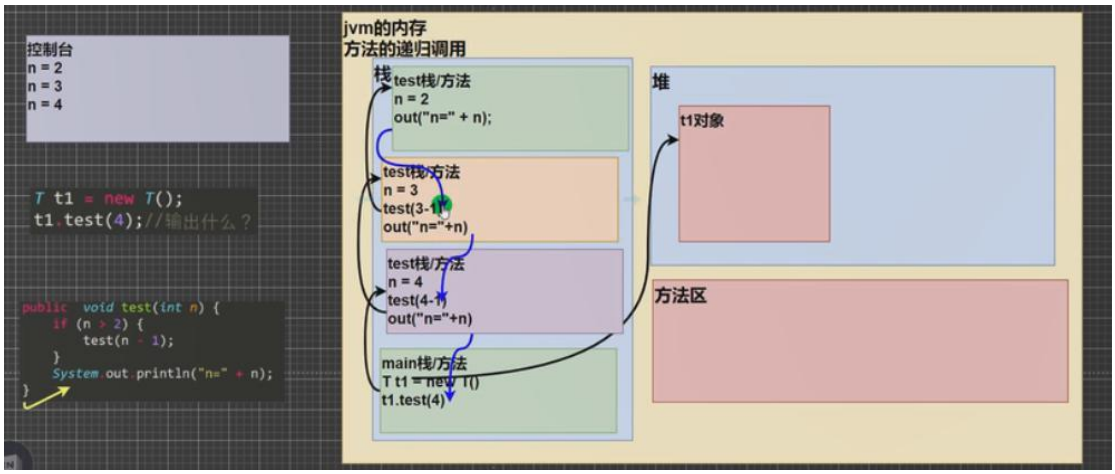


对于基本数据类型，传递的是值（值拷贝），形参的任何改变不影响实参。
引用类型传递的是地址（传递的也是值，但是值是地址），可以通过形参影响实参。

克隆对象：
//1)编写一个 MyTool 类

```
//   编写一个方法可以打印二维数组的数据
//2)编写一个方法 copyPerson,可以复制一个 Person 对象，返回复制的对象
//   克隆对象，注意要求得到新对象和原来的对象是两个独立的对象
public class MethodExercise03{
    public static void main(String[] args){
        MyTool mytool = new MyTool();
        int arr[][] = {{1,2,3},{4,5,6},{7,8,9}};
        mytool.print(arr);

        Person p = new Person();
        p.name = "gan";
        p.age = 100;

        MyTool tool = new MyTool();
        Person p2 = tool.copyPerson(p);

        //到此 p 和 p2 都是两个独立的 Person 对象，属性相同
        System.out.println("p  的属性  age = " + p.age +"  名字= " + p.name);
        System.out.println("p2 的属性  age = " + p2.age +"  名字= " + p2.name);
    }
}

class Person{
    String name;
    int age;
}

class MyTool{
    public void print(int[][] arr){
        for(int i=0;i<arr.length;i++){
            for(int j=0;j<arr[i].length;j++){
                System.out.println(arr[i][j] + " ");
            }
        }
    }

    public Person copyPerson(Person p){
        Person p2 = new Person();
        p2.name = p.name;//把原来对象的名字赋给 p2.name
        p2.age = p.age;//把原来对象的年龄赋给 p2.age

        return p2;
    }
}
```

递归调用



阶乘
递归调用

```
//Fibonacci 数列的打印
//
public class Fibonacci{
	public static void main(String[] args){
		int n = 10;
		T fabe = new T();
		int fabe1 = fabe.f(n);
		System.out.println("n= "+n+"对应的斐波拉契数为："+fabe1);
	}
}

class T{
	public int f(int n ){
		if( n==1 || n==2 ){
			return 1;
		}else if(n>2){
			return f(n-1) + f(n-2);
		}else{
			System.out.println("应该输入大于 0 的数");
			return -1;
		}
	}
}
```

//迷宫问题

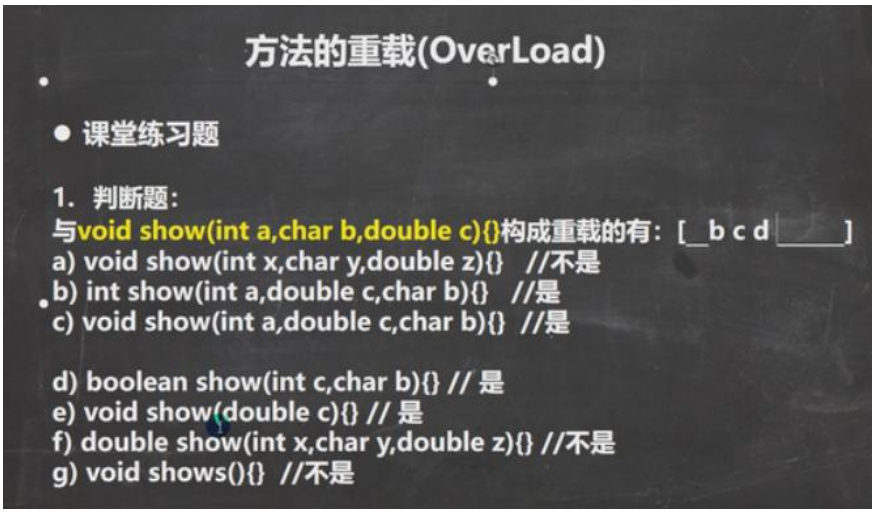
```
//
public class MiGong{
    public static void main(String[] args){
        //    思路
        //1. 先创建迷宫，用二维数组表示：  int[][] map = new int[8][9]
        //2. 再规定  map  数组的元素值：0  表示可以走; 1  表示障碍物

        int[][] map = new int[8][7];
        //3. 将最上面的一行和最下面的一行全部设置为 1
        for(int i = 0; i<7 ; i++){
            map[0][i] = 1;
            map[7][i] = 1;
        }
        //4. 将最右边的一列和最左边的一类全部设置为 1
        for(int i=0;i<8;i++){
            map[i][0]=1;
            map[i][6]=1;
        }
        //障碍物的设置
        map[3][1] = 1;
        map[3][2] = 1;
        map[2][2] = 1;
        //输出当前地图
        System.out.println("====当前地图情况====");
        for(int i=0; i<map.length ; i++){
            for(int j=0 ; j<map[i].length;j++){
                System.out.print(map[i][j] + " ");//输出一行
            }
            System.out.println();
        }

        //使用 findWay 给老鼠找路
        T t1 = new T();
        t1.findWay(map,1,1);

        System.out.println("====找路的情况如下====");
        for(int i=0; i<map.length ; i++){
            for(int j=0 ; j<map[i].length;j++){
                System.out.print(map[i][j] + " ");//输出一行
            }
            System.out.println();
        }
    }
}
```

```
class T{
    //使用递归回溯的思想来解决老鼠出迷宫问题
    //1. findWay 方法是用来专门找出迷宫的路径
    //2. 如果找到就返回 true，没有找到就返回 false
    //3. map 代表二维数组即迷宫
    //4. i 和 j 代表老鼠的位置，初始化的位置为(1,1)点
    //5. 由于使用递归找路，所以规定 map 数组每一个值的含义
    //    0 表示可以走;1 表示障碍物;2 表示可以走;3 表示走过但是是死路
    //6. 当 map[6][5] = 2 就说明找到通路,就可以结束,否则就继续找
    //7. 先确定老鼠的找路策略 下->右->上->左
    public boolean findWay(int[][] map, int i, int j){
        if(map[6][5] == 2){
            return true;
        }else {
            if(map[i][j] == 0){//当前这个位置为 0，表示可以走
                //假定可以走通
                map[i][j] = 2;
                //使用找路策略类确定该位置是否真的可以走通
                //下->右->上->左
                if(findWay(map,i+1,j)){//先走下
                    return true;
                }else if(findWay(map,i,j+1)){//右边
                    return true;
                }else if(findWay(map,i-1,j)){//上
                    return true;
                }else if(findWay(map,i,j-1)){//左
                    return true;
                }else{
                    map[i][j] = 3;
                    return false;
                }
            }else{ //map[i][j] = 1,2,3
                return false;
            }
        }

    }
    //修改找路策略，看看路径是否变化
    //下->右->上->左 ===> 上->右->下->左
        public boolean findWay2(int[][] map, int i, int j){
        if(map[6][5] == 2){
            return true;
        }else {
```

```
                if(map[i][j] == 0){//当前这个位置为 0，表示可以走
                    //假定可以走通
                    map[i][j] = 2;
                    //使用找路策略类确定该位置是否真的可以走通
                    //上->右->下->左
                    if(findWay2(map,i-1,j)){//先走上
                        return true;
                    }else if(findWay2(map,i,j+1)){//右边
                        return true;
                    }else if(findWay2(map,i+1,j)){//下
                        return true;
                    }else if(findWay2(map,i,j+1)){//左
                        return true;
                    }else{
                        map[i][j] = 3;
                        return false;
                    }
                }else{ //map[i][j] = 1,2,3
                    return false;
                }
            }

        }
    }
```

方法重载（OverLoad）
java 中允许同一个类中，有多个同名方法的存在，但是要求形参列表不一致！
重载减轻了命名的麻烦

注意：（1）方法名必须相同；（2）形参列表必须不同（形参的类型或者个数或者顺序，至少有一样不同，但是参数名无要求）；（3）返回类型无要求。



```java
class Methods {
    //分析
    //1 方法名 m
    //2 形参 (int)
    //3.void
    public void m(int n) {
        System.out.println("平方=" + (n * n));
    }

    //1 方法名 m
    //2 形参 (int, int)
    //3.void
    public void m(int n1, int n2) {
        System.out.println("相乘=" + (n1 * n2));
    }

    //1 方法名 m
    //2 形参 (String)
    //3.void
    public void m(String str) {
        System.out.println("传入的str=" + str);
```

可变参数：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。
其基本语法为：
访问修饰符 返回类型 方法名(数据类型... 形参名){
}

注意事项：
（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，但是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：
1. 在 java 编程中，主要的变量就是属性（成员变量=全局变量）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；
3. java 中作用域的分类：全局变量：也就是属性，作用域为整个整体；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（属性）可以不赋值，可以直接使用。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。

注意事项：
1. 属性（全局变量）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；
3. 属性的生命周期较长，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死亡；
4. 作用域范围不同：全局变量（属性）：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。

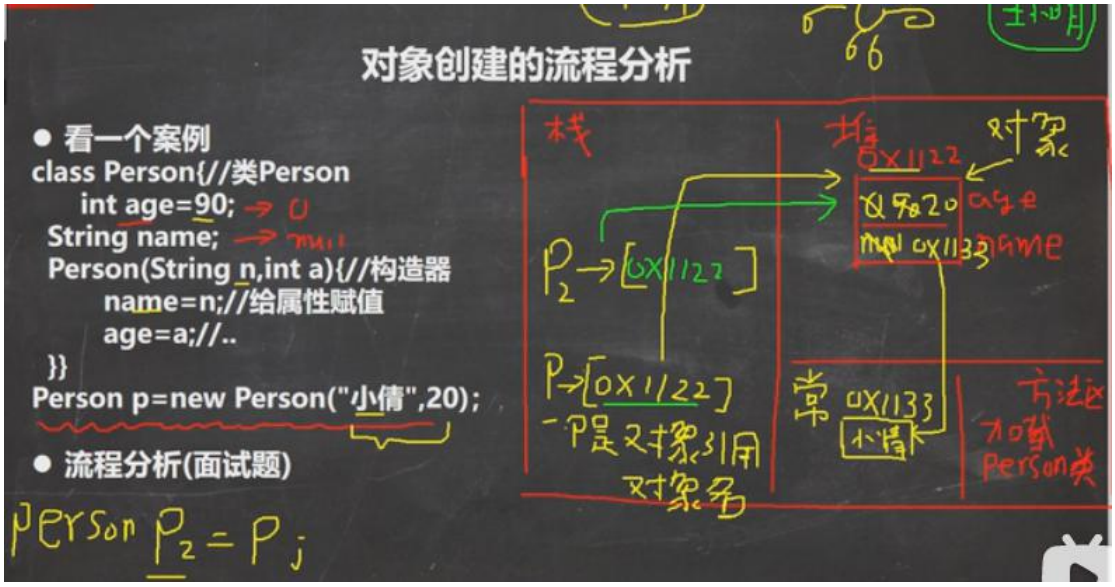
构造方法/构造器：
构造方法又叫构造器（constructor），是类的一种特殊的方法，它的主要作用是完成对新对象的初始化，有几个特点：
1）方法和类名相同；
2）没有返回值；
3）在创建对象时，系统会自动的调用该类的构造器完成对对象的初始化。

基本语法：
[修饰符] 方法名（形参列表）{
　　方法体;
}
说明：
（1）构造器的修饰符可以默认；
（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）构造器的调用由系统完成。

注意事项：
1. 一个类可以定义多个不同的构造器，即构造器重载；
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器，系统会自动给类生成一个默认无参构造器（也叫默认构造器），比如 Person(){};可以使用 javap 反编译.class 文件
7. 一旦定义了自己的构造器，默认的无参数构造器就被覆盖了，就不能再使用默认的无参构造器了，除非显式地定义一下，即 Dog(){}

```
//构造器
//
//
public class Constructor01{
    public static void main(String[] args){
```

```
        //当我们 new  一个对象时，直接通过构造器
        Person p1 = new Person("smith", 80);
        System.out.println("p1 的信息如下");
        System.out.println("p1 对象  name= " + p1.name);
        System.out.println("p1 对象  age = " + p1.age);
    }
}

class Person{
    String name;
    int age;
    //构造器
    //1. 构造器没有返回值，也不能写 void
    //2. 构造器的名称和类 Person 一样
    //3. (String pName, int pAge)  是构造器形参列表，规则和成员方法一样
    public Person(String pName, int pAge){
        System.out.println("构造器被调用~~ 完成对象的属性初始化");
        name = pName;
        age = pAge;
    }
    public Person(String pName){
        name = pName;
    }
}
```



流程分析：
1. 加载 person 类信息（Person.class），只会加载一次；
2. 在堆中分配空间（地址）；
3. 完成对象初始化[(1)默认初始化 age = 0 name = null，(2)再进行显式初始化 age

= 90 name = null ，(3)构造器的初始化 age = 20 name = 小倩]；
4. 把对象在堆中的地址返回给 p（p 为对象名或叫做对象的引用）。

this 关键字

```
class Dog{ //类

    String name;
    int age;
    // public Dog(String dName, int  dAge){//构造器
    //   name = dName;
    //   age = dAge;
    // }
    //如果我们构造器的形参，能够直接写成属性名，就更好了
    //但是出现了一个问题，根据变量的作用域原则
    //构造器的name 是局部变量，而不是属性
    //构造器的age  是局部变量，而不是属性
    public Dog(String name, int  age){//构造器
        name = name;
        age = age;
    }

    public void info(){//成员方法,输出属性x信息
        System.out.println(name + "\t" + age + "\t");
    }
```

局部变量和属性会进行重叠，进而导致混乱

Java 虚拟机会给每个对象分配 this，代表当前对象。

```
    //如果我们构造器的形参，能够直接写成属性名，就更好了
    //但是出现了一个问题，根据变量的作用域原则
    //构造器的name 是局部变量，而不是属性
    //构造器的age  是局部变量，而不是属性
    //==> 引出this关键字来解决
    public Dog(String name, int  age){//构造器
        //this.name 就是当前对象的属性name
        this.name = name;
        //this.age 就是当前对象的属性age
        this.age = age;
    }

    public void info(){//成员方法,输出属性x信息
        System.out.println(name + "\t" + age + "\t");
    }
}
```

简单来说：哪个对象调用，this 就代表哪个对象

This 使用注意事项：

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。