Java SE

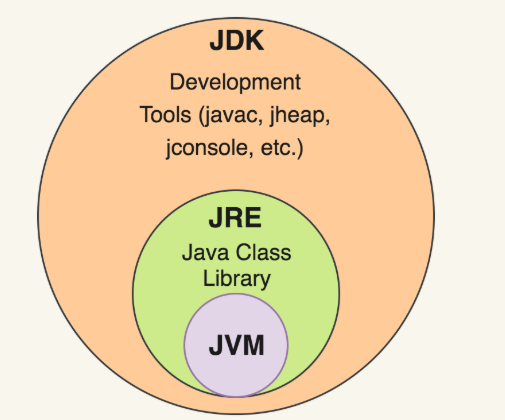
# Java语言描述

## 1.基础知识

JDK(Java Development Kit)--java开发工具

JRE(Java Runtime Enviornment)--java运行环境

JVM（Java Virtual Machine）--java虚拟机



* 若要运行如Hello.java文件，则先通过cd进入文在所在目录
* Javac 文件名.java ---将源文件编译为.class文件（字节码文件/二进制文件）
* Javac -encoding utf-8 文件名.java--可以解决中文乱码的问题
* Java 文件名 ---将编译好的文件放到JVM上运行

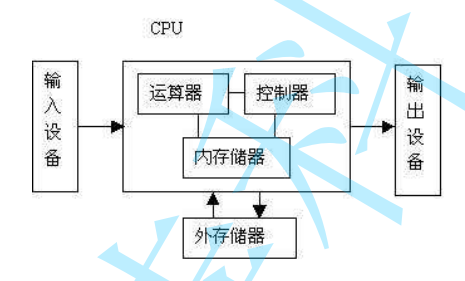
注：Java程序最后都是在java虚拟机上运行的，即一次编译，多次运行，是跨平台的，与操作系统无关，只与是否装了jdk有关

## 写java程序时应注意的点

1. 一个java文件里面，最多只能有一个public的类，即只能有一个public class，而且这个public class后面的类名必须和文件名相同
2. 每次修改代码进行编译之后，都会覆盖掉之前的class文件。但是，如果这次修改把代码全部删除了，那么这次编译不会覆盖掉之前的class文件，运行后也会是上次的结果
3. 类名一定要注意使用大驼峰的形式（如HelloWorld）
4. 变量名一定要使用小驼峰的格式（如maxNumAdd）
5. java中定义数组的时候是用（数组类型 数组名称）这样的方式定义的（如int[] a表示定义一个类型为int[]的数组，又如public static void main(String[] args)里面的String[] args表示一个名为args的字符串数组(args的类型为String[])）
6. **String类型不是Java的包装类，像Long，Character那些才是**
7. **标识符由字母、数字、下划线“\_”、美元符号“$”或者人民币符号“￥”组成,并且首字母不能是数字。**

# 数据类型和运算符

题外话：冯诺依曼计算机结构



* 手机里面的8+256里面的8表示内存，256表示外存
* 程序在运行的时候，里面的变量是存储在内存里面的

## 1.基础知识

1.变量命名不能用数字开头，同时最好不要以下划线开头

2.注释使用ctrl+/ 怎么注释怎么取消

3.块注释使用ctrl+shift+/

4.int a = 10

System.out.println(a)--换行输出

System.out.printf(“%d\n”,a)--和C语言一样格式化输System.out.print(a)--不换行输出

## 2.整型变量(Class Integer)

1.在java中，整型没有64位和32位的概念，都是4个字节

4个字节（byte）也就是32位（bit）表示的范围为-2^31->2^31-1（这里为什么正数要-1呢，因为带有带有负号的数最小是-1，不带负号的数最小是0，简单说就是在计算机中0属于正数那边)，大概是-21亿到+21亿

2.

-1 0

-214...48 214...47

整型的数值范围图

左半边是负数，从-1到-214...48，右半边是正数，从0到214...47，如果发生溢出，如214...47+1，它的值会是-214...48，同理-214...48-1=214...47，就像图所示的圆一样，首尾相连。

3.但是如果int a =214...48，在Java里面无法通过编译（C语言可以），会直接报错，这就是Java的安全性（一些C语言能通过编译的小瑕疵在Java里面就不能通过编译）

## 2.长整型变量(Class Long)

1.占8个字节，从-2^63-2^63-1。

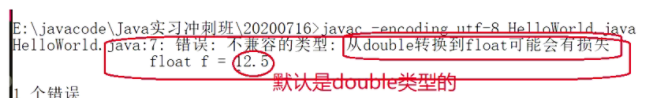
2.定义的时候可以Long a =10L,可以在10后面加个L，表示是一个长整型。

Tips:包装类的好处：可以更好的让我们面向对象---比如Integer可以直接int a =Integer.parseInt(s)把字符串直接转换成数字，但是如果是普通的int类型，就需要自己写一个方法去实现这个转换。

## 双精度浮点类型变量(Class Double)

1.占8个字节，一般使用小数的时候，只用double

2.Java的安全性



## 字符串变量（Class Character）

1.占2个字节，所以可以表示一个汉字（char a =’任’）

2.Java里面没有ascii码表，只有unicode表，它能表示2^16个信息，包括了255个ascii表中内容，同时它还有中文

3.char不能赋值负数！！！！！！，char a = 65,输出a的值为A

## 字节类型变量（Class Byte）

1.占一个字节，Java特有，占一个字节（范围从-128到127）

## 短整型变量（Class Short）

1. 占两个字节

## 布尔类型变量（Class Short）

1. 只有两个值，true和false，不能用1和0
2. 没有明确说占几个字节，有的说上一个字节，有的两个字节，JVM上没有明确规定

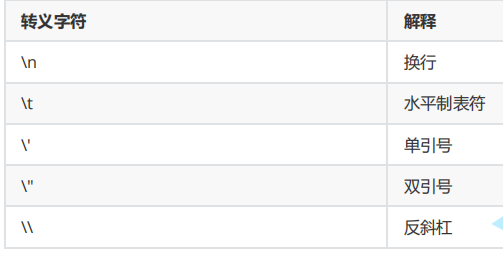
## 字符串类型变量（Class String）

1. 存在常量池中

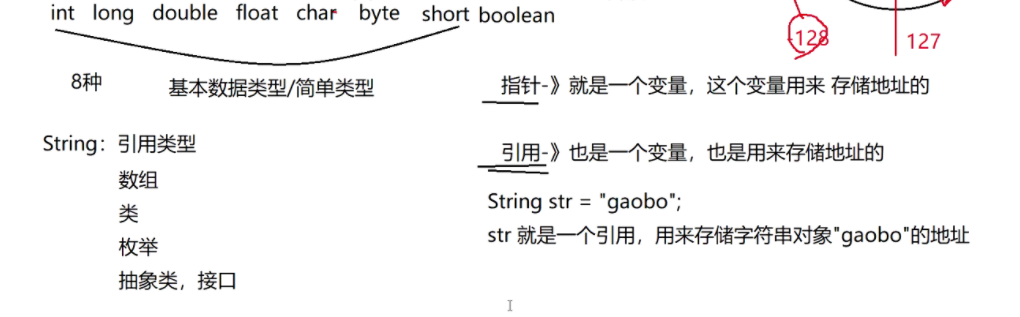
2.一旦表达式中出现字符串，+（拼接）以后一定是字符串



3.在字符串中想要输出单引号，双引号，反斜杠，前面都要加上反斜杠



## 总结



八种基本数据类型：byte(1) short(2) char(2) int(4) long(8) float(4) double(4) boolean(?)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

01结束

## 常量

1.在Java中，我们使用final关键字，来定义一个常量，

如int a = 20是变量，final int a = 20是常量

1. 变量：运行期间可以改变的量，并且只有在运行的时候，才能知道它的值。

常量：在编译期间，就能确定它的值，并且在运行期间不可以进行改变。

Int a = 20;

Final int b = 20;

Int c = 80+a; //这里在编译的时候，并不知道a是多少

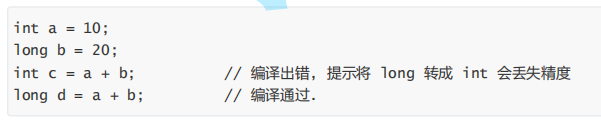
Int c = 80+b; //这里在编译的时候，就已经知道b是多少，后面的80+b会被替换为100

1. 常量一旦被定义，就不能修改。一定要初始化，才能使用。具体是不是在定义的时候初始化，没有关系。

## 理解类型转换

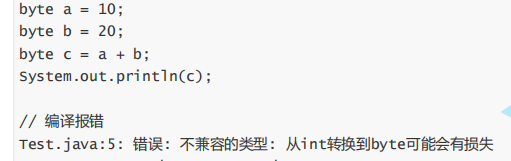
1.把大类型（8字节）给小类型（小字节）的时候，一定要使用强制类型转换，不然编译会出错（只要他觉得精度会丢失，就不会让你通过编译，这就是Java安全性的体现）

2.



当 int 和 long 混合运算的时候, int 会提升成 long, 得到的结果仍然是 long 类型, 需要使用 long 类型的变量来接收结果. 如果非要用 int 来接收结果, 就需要使用强制类型转换.

3.

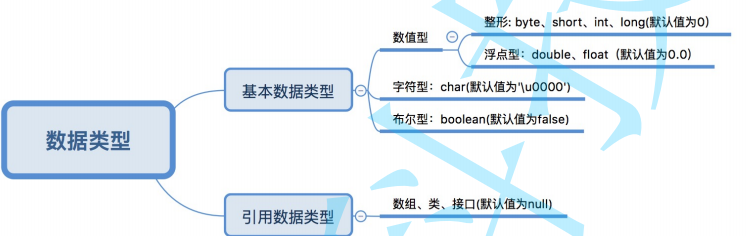


byte 和 byte 都是相同类型, 但是出现编译报错. 原因是, 虽然 a 和 b 都是 byte, 但是计算 a + b 会先将 a 和 b 都 提升成 int, 再进行计算, 得到的结果也是 int, 这是赋给 c, 就会出现上述错误.

由于计算机的 CPU 通常是按照 4 个字节为单位从内存中读写数据. 为了硬件上实现方便, 诸如 byte 和 short 这种低于 4 个字节的类型, 会先提升成 int, 再参与计算。

4.int转String---int a = String.valueof(num)

String转int---String b = Integer.parseInt(“123”)

5.

在Java中，变量在定以后，如果要使用，那么使用之前一定要进行初始化，否则会编译出错。这里面写的默认值，在后面类和对象的时候会提到

## 运算符

1.**负数取余运算**：所有语言和计算器都遵循了尽量让商尽量靠近0的原则，即5%(-3) 的结果为2而不是-1，(-5)%3的结果是-2而不是1

在编译器中，两个异号的数取余之后的结果取决于分子的符号。负数%负数，编译器会将分母的负数自动转换为正整数，然后再将分子负数的负号提取出来，将两个正整数取余，最后的结果加上负号就好了。

10 % 3 = 1

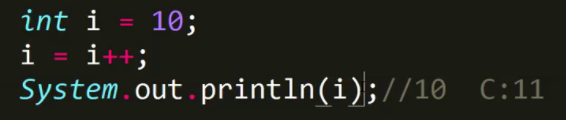
10 % -3 = 1

-10 % 3 = -1（原式 = -（10 % 3））

-10 % -3 = -1（原式 = -（10 % 3））

2.Java中的模可以取小数，如10.5 % 2.0 = 0.5

3.

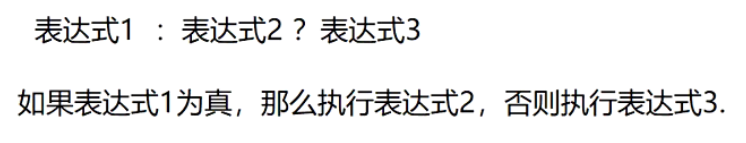


记一下就行，在C里面是11，在Java里面是10

1. **关系运算符**的返回值都是boolean类型（a==b--true）
2. !只能作用于布尔表达式
3. **移位运算**：同学：如何将一个数快速扩大8倍？（左移3位）



1. **条件运算符：**



8.**运算符的优先级**：没必要记，想先算哪个，就加上括号

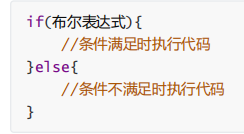
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

02结束

# 程序逻辑控制

## 1.分支结构

1. 在Java里面if else这种后面的大括号不用换行。

2.不管if后面有几条语句，哪怕只有一条，也必须加上花括号

1. 一般分支大于3个了，采用switch语句
2. 能作为switch参数的数据类型有哪些？

Int,byte,char ，string可以作为switch参数

Long,float,double,boolean不能作为switch参数

5.用switch时候，要注意不要遗漏break,switch一般在三种分支以上的时候采用，但是它无法表达相对复杂的条件。

## 循环结构

1. break:结束整个循环

Continue：结束本趟循环

1. for循环的第二个条件如果没有写，那么其实就是一个死循环

## 输入（Scanner）

1. nextline()能读入字符串，遇到空格不会结束，会读入上一个回车
2. next（）能读入字符串，遇到空格就结束。

**\*:最大公因数：**

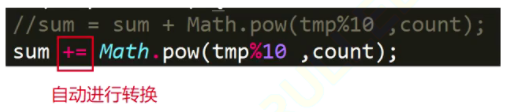
1. **最大公因数可以是1，且一定存在**
2. **0和一个数的最大公约数就是那个数本身**
3. **辗转相除法：不管大小a%b得到余数，然后a=b;b=余数；就这样循环，当b也就是除数为0，循环结束，此时被除数a就是最大公约数。**
4. **最大公倍数 = a \* b / 最大公约数**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

03结束

Some tips:1.String里面，要比较两个String的值相等不相等，要用.equals，而不是用==，==是用来比较地址是否相等。

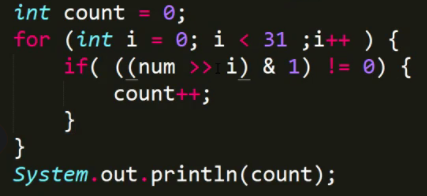
1.



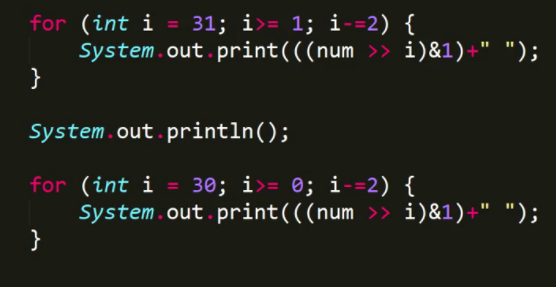
这里Math.pow返回的是double类型的数据，但是sum是int，按理说会出问题，但是+=这个符合运算符会自动进行强制类型转换

2.要学会通过按位&去获得一个二进制数里面有几个1，一个二进制的奇数位和偶数位分别都是多少。

**有几个1:**每次右移i位，然后和0000 0001相&，如果结果是1，表明第i位上面是1，所以依次把每一位都移到最前面和0000 0001进行&运算，如果是1，count++就行。



**奇数位和偶数位分别是多少：**



要输出偶数位的数，进行右移奇数位（i），然后和0000 0001进行&运算，结果即是第i位上的值

**总结：要取得二进制数上某一位的值，对其进行适当地右移操作，把当前位移到第一位，然后和0000 0001进行&运算，其结果就是该位上的值。**

# 方法的使用

## 方法使用的注意事项

1.目前来说，只要在main函数中调用的方法，要使用public static

2.方法命名：见名思意，采用小驼峰

3.Java不需要进行函数声明

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

04结束

### 作业题：

有一组数据，只有一个数字是出现一次，其他是两次，请找出这个数字。

解题思路：采用二进制的异或（^）来进行求解，a^a = 0,a^0 = a，由于一组数据中，一样的数据^会变成0,然后0和剩下的一个数进行^会变成那个数。

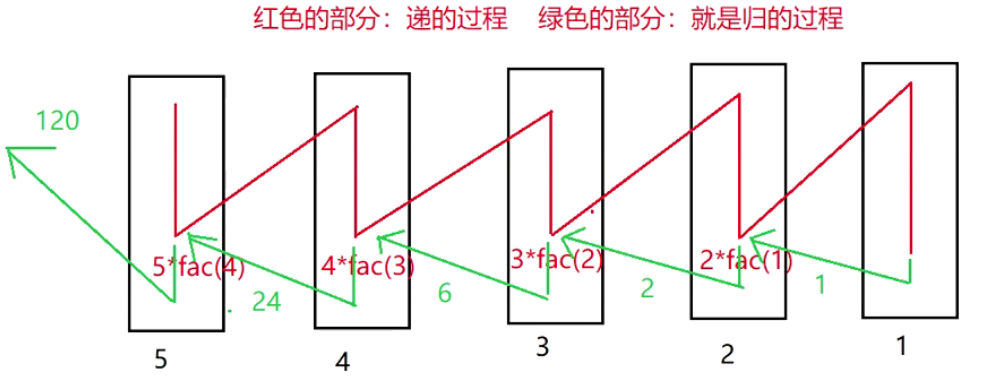
## 方法的重载

1. 方法名要相同
2. 参数列表不同：（1）个数不同；（2）类型不同；（3）参数顺序不相同
3. 返回值不做要求

**\*：重载和重写的区别？？？？**

## 递归

1. 要注意设置递归结束的条件
2. 求斐波那契数列的时候，尽量使用循环的方法，不要用递归





**栈溢出异常：终止条件错了**

栈的增长方向：从高地址到低地址 （栈底是高地址）

堆的增长方向：从低地址到高地址

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

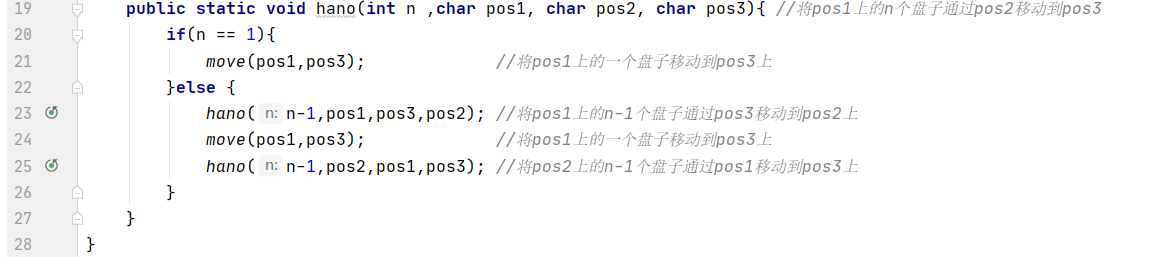
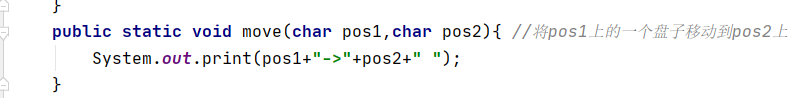
05结束

**汉诺塔问题：if(n=1) A->C**

**Else 将A上的n-1个通过C挪到B上 n-1 A C B**

**将A上剩余的最大的那一个挪到C上**

**将B上的n-1个通过A挪到C上 n-1 B A C**



见idea中的代码

# 数组

## 数组的定义

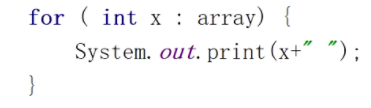


1. 如果数组没有进行初始化，那么默认值为0

## 数组的打印方式

1.for

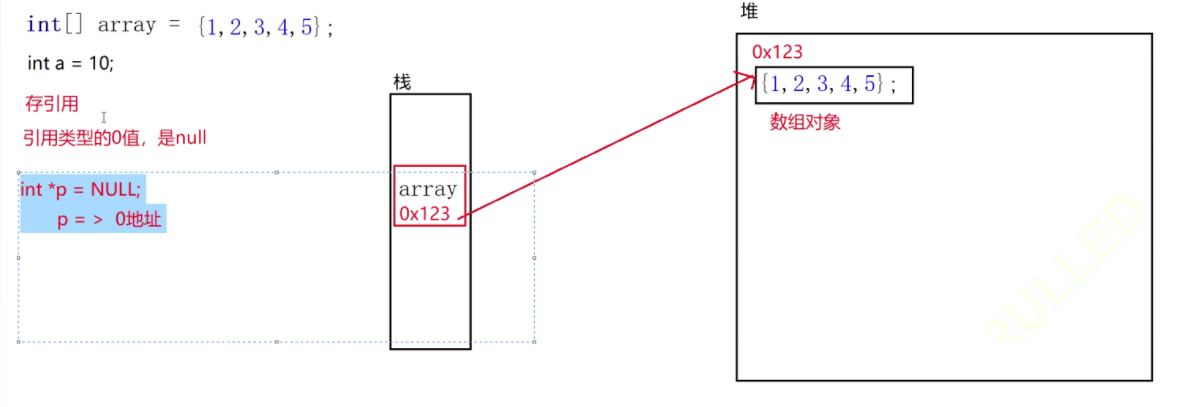
2.

for each循环

3.[1, 2, 3, 4, 5]

## 理解引用类型（重点）

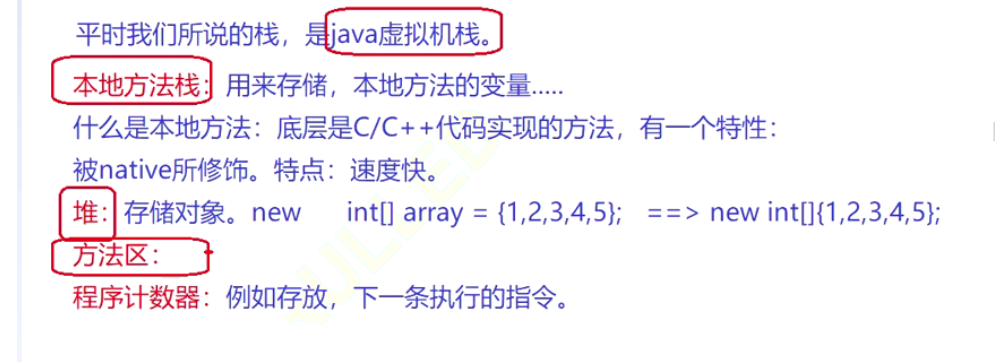
1. 数组也属于引用类型



* 堆不占用内存，栈需要占用内存，array存在栈中，其中存放的是数组对象的地址0x123，而数组对象存在堆中
* Null也是一个对象，如果尝试去访问null.length，会出现空指针异常
* 当Array=null，置空的时候，原来堆里面的对象会被自动回收，不需要像C里面那样free

## 初识 JVM 内存区域划分(重点)

1.



## 数组作为方法的返回值

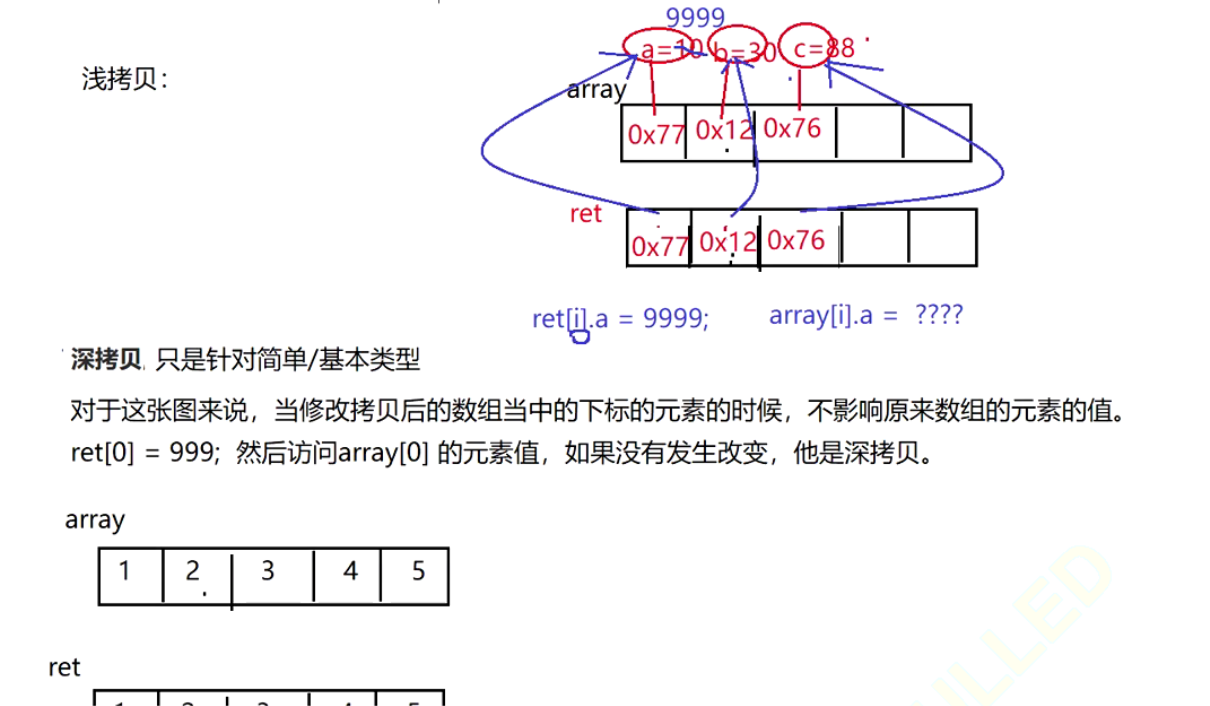
1. 在写题的时候，对方法传入的参数要进行特殊判断

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

06结束

## 数组拷贝（重点）

1. for循环拷贝
2. Int [] ret = Arrays.copyOf(a,a.length); //使用Arrays类里面的方法
3. System.arraycopy(a,ret,0,a.length); //使用System里面的方法
4. Int [] ret = a.clone(); //产生需要克隆对象的一个副本



* 浅拷贝：第一个数组中存放的都是地址，拷贝到第二个数组中的也只是地址，其所指的对象没有发生改变，如果改变ret[0].a的值，相对应的array[0].a的值也会发生相应的改变。
* 深拷贝：见图

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

07结束

## 数组一些相关的题目

1. 二分查找：在进行二分查找之前一定要确保数组是有序的，可以使用Arrays.sort来对数组进行排序
2. 冒泡排序：总共进行n-1轮，每一轮都可以使最大的一个数冒泡到最后面，

每经过一轮，有序序列的个数就+1。

------- i表示趟数，j在无序序列里面两两比较

i = 0 ;i < a.length-1

J = 0; j<a.length-1-i

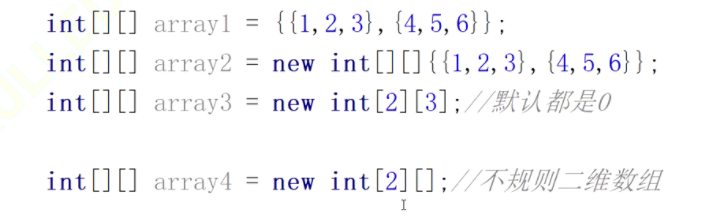
冒泡排序还可以用flag来进行优化，如果没有一轮i循环没有发生比较，那表示数组已经有序，后续的循环也就没有必要了。

1. 随机数random的用法：

Random random = new Random()

Int a = random.nextInt(bound) //(0,bound]

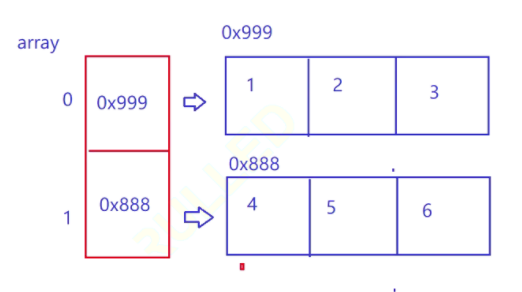
## 二维数组

1.二维数组的定义方法

2.二维数组的遍历和输出

一定要注意二维数组是用a.length和a[i].length这种形式的方法去进行数组遍历的

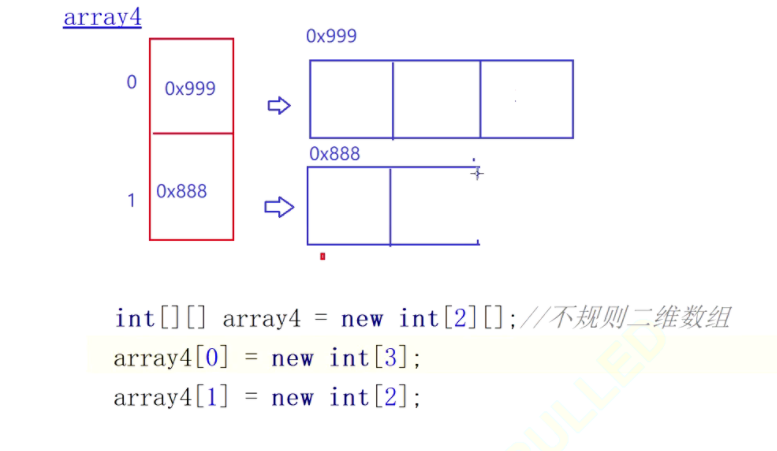
因为二维数组的本质实际上是如下图所示的





注：除了循环遍历打印以外，还可以使用Arrays.deepToString方法来进行打印

3.不规则的二维数组



# 类和对象

## 类和对象

1.类名采用大驼峰的形式（如Person）

1. 字段：类里面，方法外定义的变量，叫做字段，也叫作成员变量，成员属性
2. 基本属性类型的默认值：

引用类型是null，

简单类型是0，其中特别的，boolean是false，char是’\u0000’(其实就是’ ’)

## 2.static关键字（重要）

1.静态的成员变量/方法，静态的成员变量/方法只有一份，它不属于对象，存放在方法区（这里指的静态方法是方法的地址）

1. 静态成员方法中不能调用非静态方法（相反可以），因为非静态方法是要用对象的引用来访问的，而静态方法是不依赖对象存在的，所以两者发生矛盾。

3.成员方法内部不能定义静态成员变量，只能访问。因为静态变量是存在方法区里面的，要是随便哪里都能定义的话，那不是乱了吗。

4.写题目的时候看仔细一点，谢谢

5.在Java中，若局部变量没有进行初始化，则不能通过编译，切记

但是，若成员变量没有进行初始化，则创建对象后，会有默认值，可以访问。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

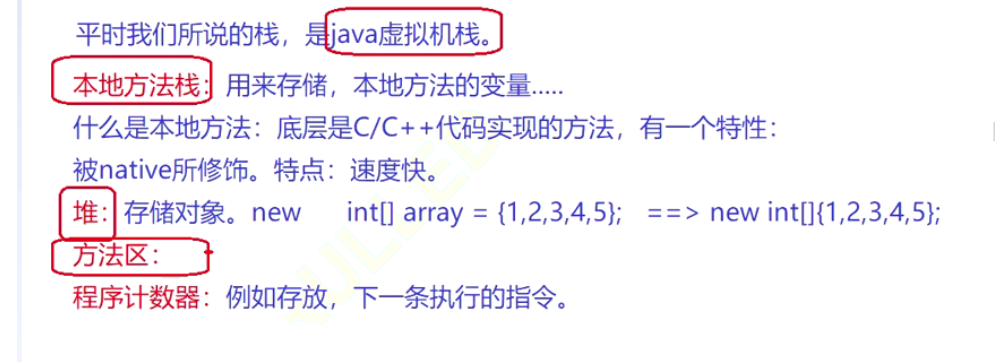
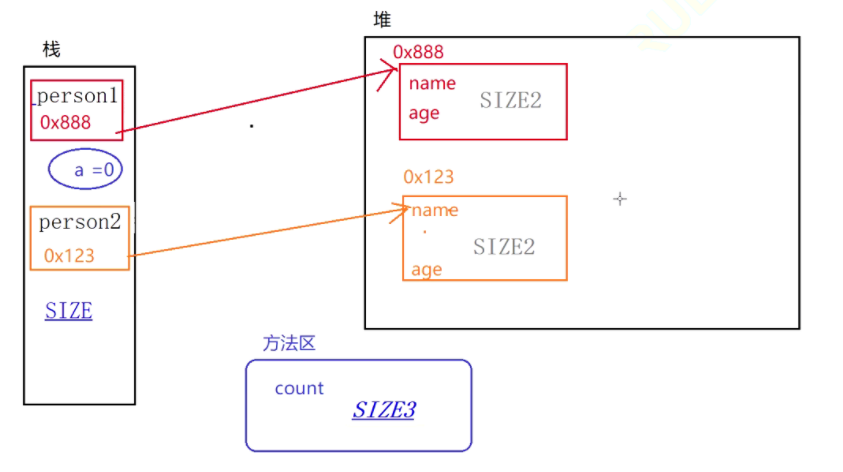
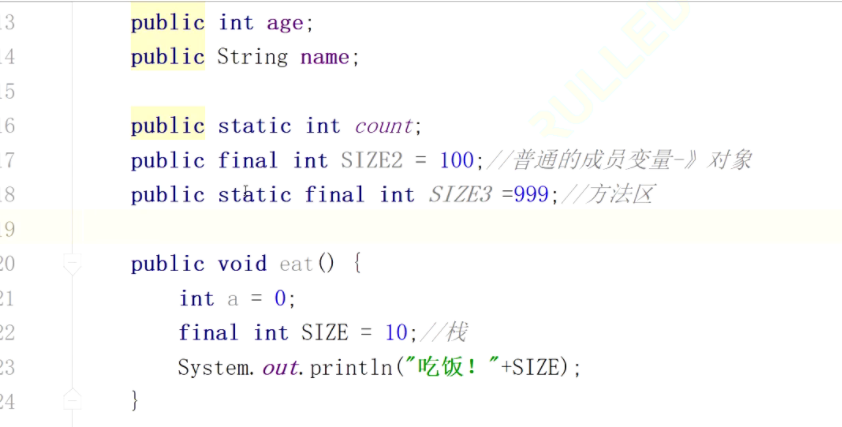
08结束

* 注意，float a = 1.0 ; /会报错，因为1.0默认是double类型

义的变量

* 从内存存储上来说，若该变量被static修饰，则存放在方法区，其他的，

若是局部变量，则存储在栈上；若是普通成员变量，则存储在堆上

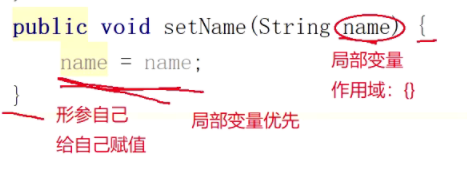


## 封装

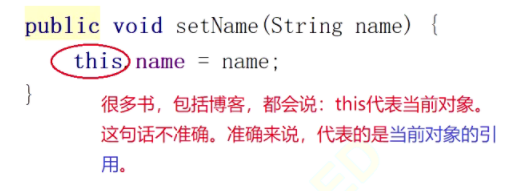
* 每一个类会对应生成一个字节码文件（.class文件）。在new这个类的时候才会去调用相应类的.class文件
* 写简历的时候不要写什么熟悉Java三大特征啥的（首先不只是java其次也不是只有三大特征），要写熟悉/了解面向对象语言的一些基本特征，例如继承，封装，多态等

1. 怎么实现封装？用private关键字修饰成员变量
2. private是私有的。被其修饰之后，只能在当前类当中使用
3. 每个类都会预留公有的接口的。一旦你把这个方法设置为public，那么就不要轻易修改他的属性为其他属性。
4. 使用private封装可以降低类的使用者对类的学习成本，会提高开发效率
5. 将属性设置为private以后，需要提供一系列的get和set方法

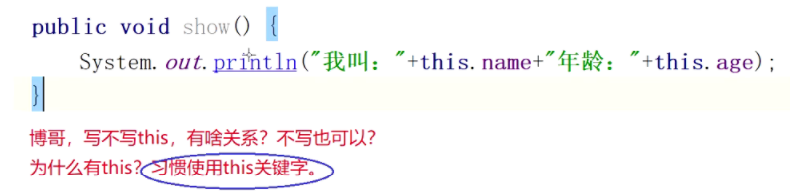
6.this关键字的使用



原来的name没有被赋值，这里是形参自己给自己赋值



所以这里使用this关键字，this表示当前对象的引用



一定要习惯使用this关键字

## 构造方法

1. 没有返回值，且方法名和类名是一样的（public Student）

* 一个对象的产生，需要分为几步

1.为对象分配内存

2.调用合适的构造方法（构造方法有好几个）

3.调用完构造方法后，对象才真正的存在

1. 面试官问你this是啥。你就说我一开始以为this表示的是对象，但是后来想了想，一个对象要产生必须先调用它里面的构造方法，而构造方法里面能使用this关键字。如this表示对象，那不就是矛盾了，变成了一个对象在产生之前就存在了。后来经过了解，我明白了this实际上指的是对象的地址
2. 当你在类当中写了其他的构造方法，那么编译器就不会为你自动生成默认的构造方法了（无参数）。
3. 也可以使用快捷生成构造方法
4. 构造方法可以发生重载（原因：（1）.方法名不同；（2）.参数列表不同；（3）.返回值不做要求；（4）.在一个类当中）
5. 在一个构造方法中调用其他的构造方法，要使用this关键字，且this关键字必须写在第一行，不然会报错



1. this的调用方法

* this.成员变量----成员变量一定是普通的成员变量，不能是静态的成员变量。
* this.成员方法
* this()----调用构造方法

## 代码块

* 普通代码块：在函数内部的代码块
* 实例代码块
* 静态代码块



**执行顺序：静态代码块->实例代码块->构造方法（就算new了两个对象，静态代码块只会执行一次）**

**注意：同是static的情况下，看定义的先后顺序**

## 补充说明

1.toString方法：将对象转换为字符串这样的操作称为**序列化**

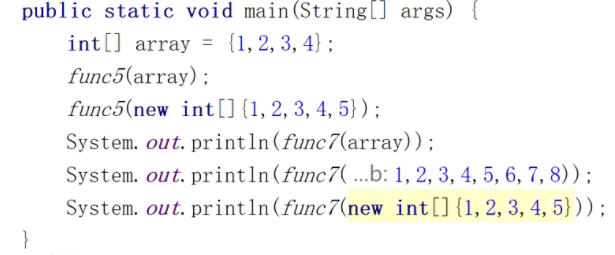
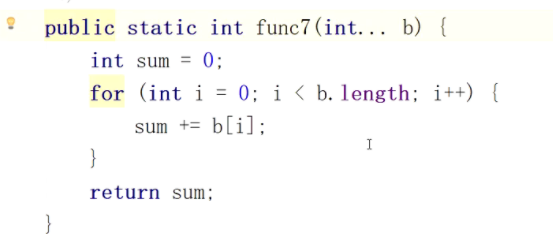
**反序列化**：序列化反过来

* 匿名对象：new Student().a;

优点：只使用一次；缺点：每次用的时候都要new对象

* 匿名数组:return new int[]{1,2,3,4,5}

1. 可变参数编程



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

09结束

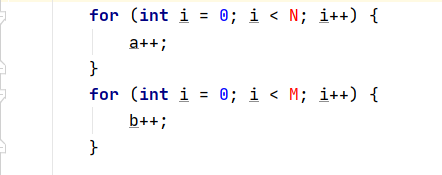
* 私有变量只能在当前类中调用
* 看那种类里面有mian函数的，直接从main函数那里开始看

# 复杂度

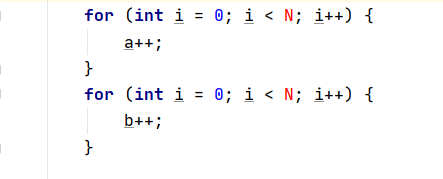
## 时间复杂度

1.

* 平时我们所说的时间复杂度是**最坏情况下**的时间复杂度



时间复杂度：O(N+M)



时间复杂度：O(N^2)

1. 递归的时间复杂度：要看递归执行的次数

# 顺序表和链表

## 线性表

1. 线性表的种类：顺序表，链表，栈，队列，字符串等
2. 在逻辑上是线性结构，但是物理地址上不一定是连续的

## 线性表

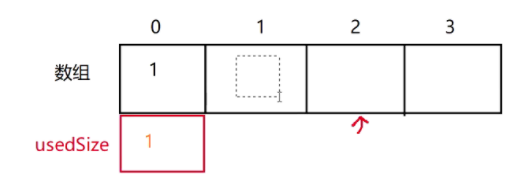
1. 物理地址连续，逻辑是线性的。

## 3.线性表的接口实现

一. // 在 pos 位置新增元素

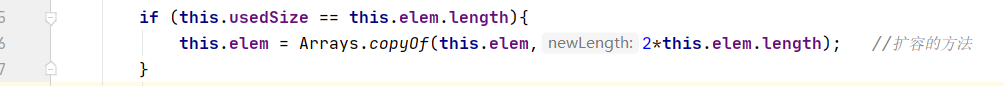
public void add(int pos, int data) { }

1. 判断pos位置是否合法，如图所示，若1号位置无元素，此时无法向2号位置放入元素



1. 如果插入的位置已经有元素，则将此元素以及其之后的元素全部后移
2. 当usedSize == elem.length的时候就要进行扩容

扩容的方法：使用Arrays.copyOf即可



// 判定是否包含某个元素

public boolean contains(int toFind) { return true; }

// 查找某个元素对应的位置

public int search(int toFind) { return -1; }

// 获取 pos 位置的元素**（注意对pos位置的元素进行特判，小于0和大于等于usedSize）**

public int getPos(int pos) { return -1; }

// 给 pos 位置的元素设为 value **（注意对pos位置的元素进行特判，小于0和大于等于usedSize）**

public void setPos(int pos, int value) { }

//删除第一次出现的关键字key

public void remove(int toRemove) { }

// 获取顺序表长度

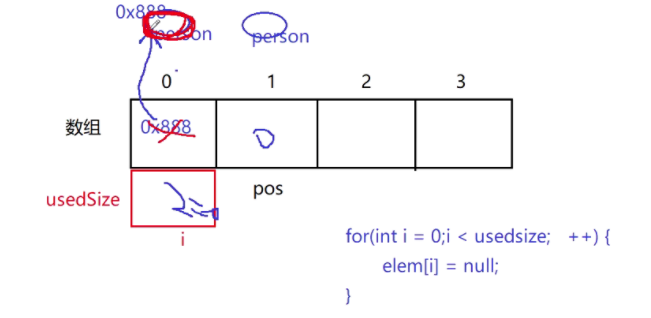
public int size() { return 0; }

// 清空顺序表

public void clear() { }

}

最后一个清空顺序表，如果数组中存放的是引用类型的变量，那不只是把usedSize置0就可以了，还需要把数组中的元素都置为空。不然的话上面那些残留的Person会引起内存泄漏



**Switch类型**



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10结束

## 链表

1.物理上不一定连续，逻辑上连续

2.顺序表相当于集合中的ArrayList，链表相当于集合中的LinkedList

3.在对通过参数传入的pos,index等，需要对其进行特判，比如index,pos>MaxSize,这里要尤其注意别忘了index,pos<0的情况

**无头结点的非循环单链表**

//头插法

public void addFirst(int data);

就是在最前插入，注意对链表为空的特判

//尾插法

public void addLast(int data);

就是在最后面插入，注意对链表为空的特判

//任意位置插入,第一个数据节点为0号下标

public boolean addIndex(int index,int data);

就主要是对index的特判，如果index不合法（<0或者>length），返回false,若index为0，那就是头插，若Index为length，那就是尾插，其他位置写一个找到index所对应结点的前一个结点的函数searchPrev(index)，轻松解决。

//查找是否包含关键字key是否在单链表当中

public boolean contains(int key);

//删除第一次出现关键字为key的节点

public void remove(int key);

主要还是一些特判，1.链表为空；2.key值不存在；3.key值为头结点上的data;注意特判里面要return，以为只需要删除第一次出现的结点。

其他的位置，写一个找到data值为key的结点的前一个结点的函数searchPreNode(key)，轻松解决。

//删除所有值为key的节点

public void removeAllKey(int key);

<https://blog.csdn.net/qq_24016309/article/details/121004625>

//得到单链表的长度

public int size();

public void display();

public void clear();

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

11结束

## 内存泄漏的查看（重要）

如果链表不调用clear消除，在程序运行过程中会发生内存泄漏

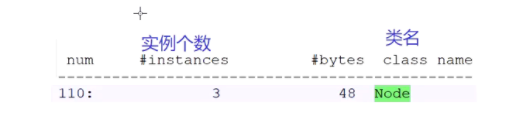
1. 在程序最后一行补个sout，然后在这打个断点，调试。
2. 打开cmd，输入jps,找到在执行的类的进程号



1. jmap -histo:live 10564 > e:log.txt（>其实表示的是重定向，如果不重定向，东西会显示在cmd里面）

4.回车后cmd里面的光标会一直闪烁，此时回到idea调试点击下一步，然后文件里面就有东西了。

5.去打开这个文件，查找Node（因为是泄漏的对象是Node类型的）



如图说明泄漏了3个Node对象

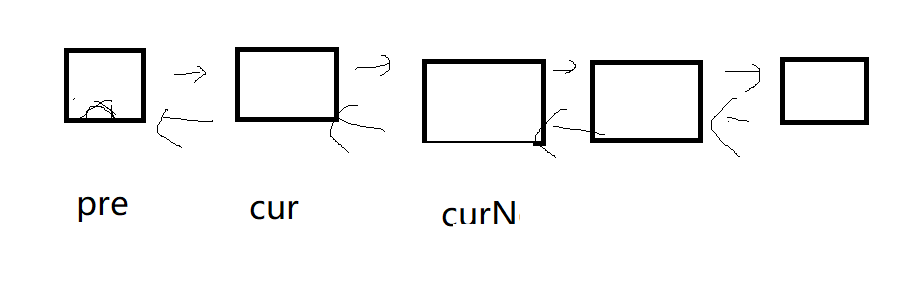
6.调用了clear以后，再重新一遍上面四步，会发现找不到Node了。

* javac：编译java文件，生成.class
* java：执行java文件
* javap：反汇编
* jps：查看进程号
* jamp：查看内存泄漏（你当时怎么去测内存泄漏的？用jmap）

## 链表面试题目（重要）

### (1).leetcode\_206链表反转

多指针法自己去画图，去尝试



### (2).leetcode\_876链表的中间结点

快慢指针法，这里要注意fast==null要放在前面，因为要是fast为空，调用fast.next会出现空指针异常。

Fast==null || fast.next == null

Return slow;

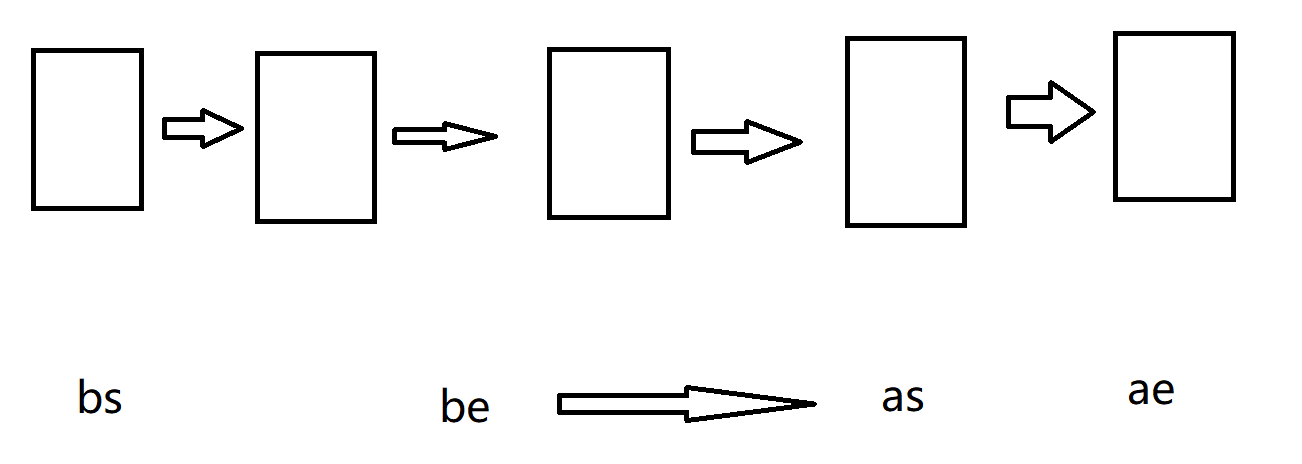
### **(3).输入一个链表，输出该链表中倒数第k个结点**。 [OJ链接]

https://www.nowcoder.com/practice/529d3ae5a407492994ad2a246518148a?

双指针法，让fast先走k步，注意在fast先走的时候要判断fast当前是否为空，防止出现空指针异常。

### **(4) .编写代码，以给定值x为基准将链表分割成两部分，所有小于x的结点排在大于或等于x的结点之前** 。[OJ链接]

https://www.nowcoder.com/practice/0e27e0b064de4eacac178676ef9c9d70?



四指针法，把比12小的放在bs->be区间里面，比12大的放在as->ae区间里面，最后be.next = as就行，

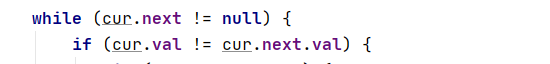
**这里还要注意几个细节：**

* **如果be==null，直接return as;**
* **最后要把ae.next = null;**
* **要把ae.next=null，要先判断ae是否==null**

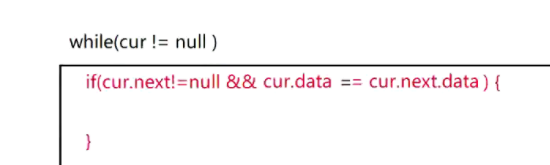
### (5).在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 [OJ链接]

<https://www.nowcoder.com/practice/fc533c45b73a41b0b44ccba763f866ef?>

**不要只会这种逻辑，用cur.next不为空来保证没有空指针异常**



**要学会用下面这种逻辑，就是循环的条件是cur!=null，把对cur.next的使用和cur.next!=null绑定在一起，防止空指针异常**



这道题目也还是用上一题的在原链表的基础上生成一个新链表有newHead和newEnd

### (6). 链表的回文结构。[OJ链接]

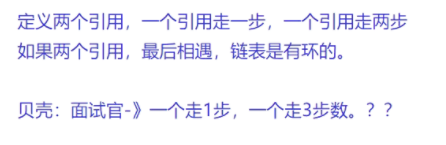
<https://www.nowcoder.com/practice/d281619e4b3e4a60a2cc66ea32855bfa?>

* 首先要想到使用**快慢指针找到链表中间的结点**，用了一个isOdd变量来判断链表个数的奇偶
* slow指针在走的时候要使用**三指针法把前半段链表逆置**
* 最后从slowpre往前，一个从slow往后判断即可

也可以先找到后半段的结点，然后单独写一个函数把后半段链表逆置

### (7).leetcode\_141\_

给定一个链表，判断链表中是否有环。https://leetcode-cn.com/problems/linked-list-cycle/



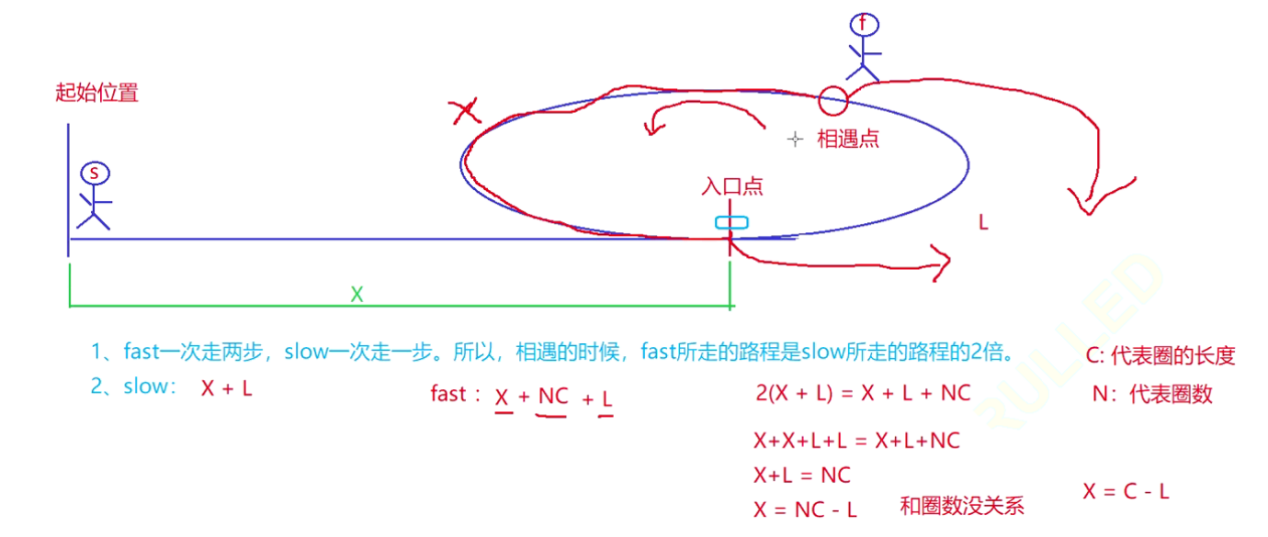
一个走一步一个走三步是不行的，他们可能要更晚相遇或者相遇不了

一个走一步一个走两步一定是最好的最快相遇的，而且一定会相遇，因为走一步那个人跨度是两步，走两步的人不可能一次跨过这两步

### (8).leetcode\_142

给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null

<https://leetcode-cn.com/problems/linked-list-cycle-ii/>



所以当fast和slow相遇的时候，我们只需要将slow = head，然后让fast和slow一步一步往前走，当他们再次相遇的时候，就是环的入口点。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

13结束

### (9).剑指offer 52\_输入两个链表，找出它们的第一个公共结点

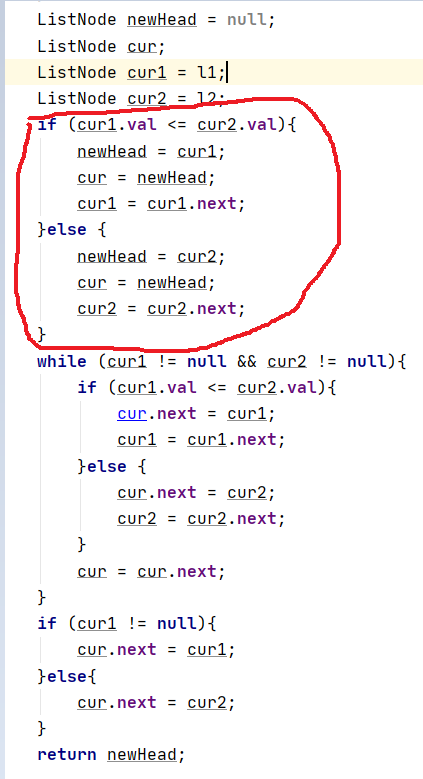
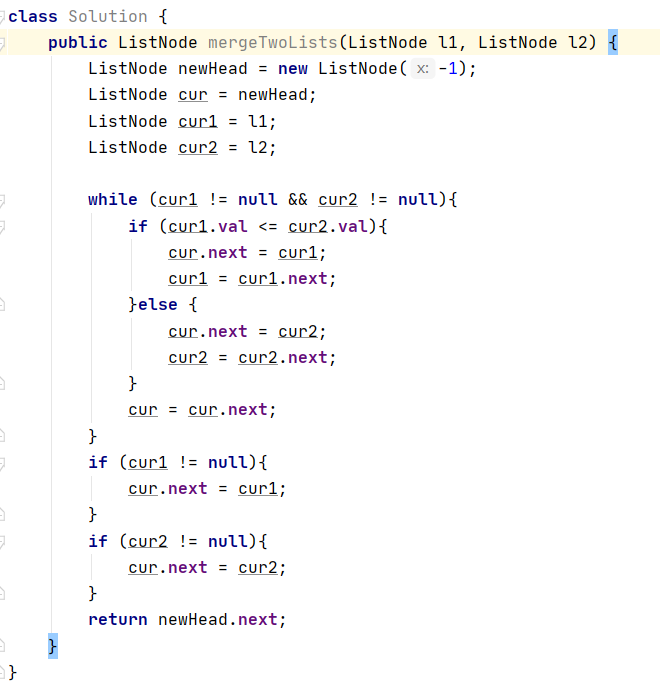
<https://leetcode-cn.com/problems/liang-ge-lian-biao-de-di-yi-ge-gong-gong-jie-dian-lcof/>

1. 分别求出两个链表的长度
2. 让最长的链表走两个链表的差值步
3. 两个引用同时开始走，如果他两相遇了，那么说明两个链表是相交的

### (10).leetcode\_21\_将两个有序链表合并为一个新的有序链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的

<https://leetcode-cn.com/problems/merge-two-sorted-lists/>

**如果你加上了傀儡结点，那么圈出来的那部分判断便可以省去了，说白了，傀儡结点的好处就是可以使得对第一个结点的操作和后面的结点一样。包括前面的一大片的特殊判断也可以全部省去。傀儡结点yyds**



解题思路的话就是弄个newHead，然后同时遍历两个链表，哪个更小往后面插就行了。最后把剩下的那段插到最后面。

**还是一句话，傀儡结点yyds（new ListNode(-1)）**

## 顺序表和链表的区别

## (数组和链表的区别)

## (ArrayList和LinkedList区别)

* 顺序表（ArrayList）：

优点 ：1.空间连续，支持随机访问

缺点 ：1.插入或者删除的时间复杂度为O(N)，

2.增容的代价比较大（增容后是原来的两倍，会浪费很多空间）

* 链表：（LinkedList）:

缺点 ：1.以结点为单位存储，不支持随机访问

优点 ：1.任意位置插入删除的时间复杂度为O(1)

2.没有增容问题，插入一个开辟一个空间

**​**

## 双向链表（LinkList底层就是双向链表）

<https://blog.csdn.net/qq_24016309/article/details/121163375>

//头插法

public void addFirst(int data);

//尾插法

public void addLast(int data);

//任意位置插入,第一个数据节点为0号下标

public boolean addIndex(int index,int data);

//查找是否包含关键字key是否在单链表当中

public boolean contains(int key);

//删除第一次出现关键字为key的节点

public void remove(int key);

//删除所有值为key的节点

public void removeAllKey(int key);

//得到单链表的长度

public int size();

public void display();

public void clear();

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

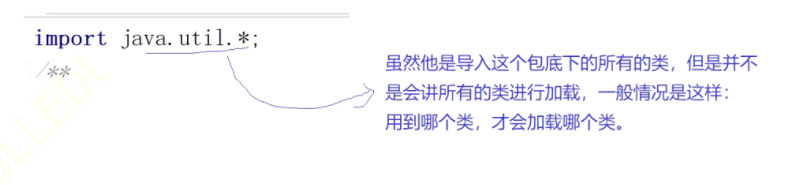
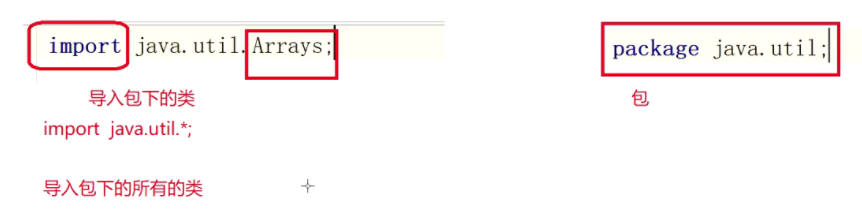
14结束

# **​**面向对象编程

## 包

1.包 (package) 是组织类的一种方式.

使用包的主要目的是保证类的唯一性



1. import表示的是导入包里面的类。但是这里要注意，lang包是不需要导入的，编译器会帮我们导入
2. 一些基本规则

* 在文件的最上方加上一个 package 语句指定该代码在哪个包中.
* 包名需要尽量指定成唯一的名字, 通常会用公司的域名的颠倒形式(例如 com.bit.demo1 ).
* 如果一个类没有 package 语句, 则该类被放到一个默认包中（src）

1. **包的访问权限**

public:一切通用

private:仅限类的内部使用

默认：包访问权限，只能在包的内部访问。

比如在包demo1里面有一个类A，A里面 int val = 0;为classA的成员变量

在包demo1里面还有一个类B，在B里面A a = new A，可以使用a.val访问

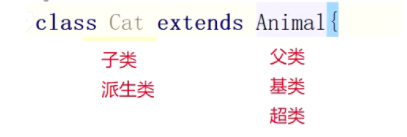
在包demo2里面有一个类C，在C里面能A a = new A，但是不能用a.val访问

## 继承

### (1).基本概念

1.当有A is a B这样的形式的时候，可以使用A extends B

**2.弄清楚子类，派生类，父类，基类，超类这些概念**



3.继承的意义

* 代码的复用
* 降低了代码冗余

**4.面试：子类继承了父类的什么？**

--除了构造方法以外所有的东西

1. 对于父类的private的字段和方法，子类是无法访问的（**但是确实都被继承了，只是无法访问**）

### (2).面试：this和super关键字的区别（重要）

1. this代表的是当前对象的引用，super代表父类的引用
2. this.data:访问当前对象的成员变量

this.fun():访问当前对象的成员方法

this():调用当前类的构造方法

super.data:访问父类的成员变量

super.fun():访问父类的成员方法

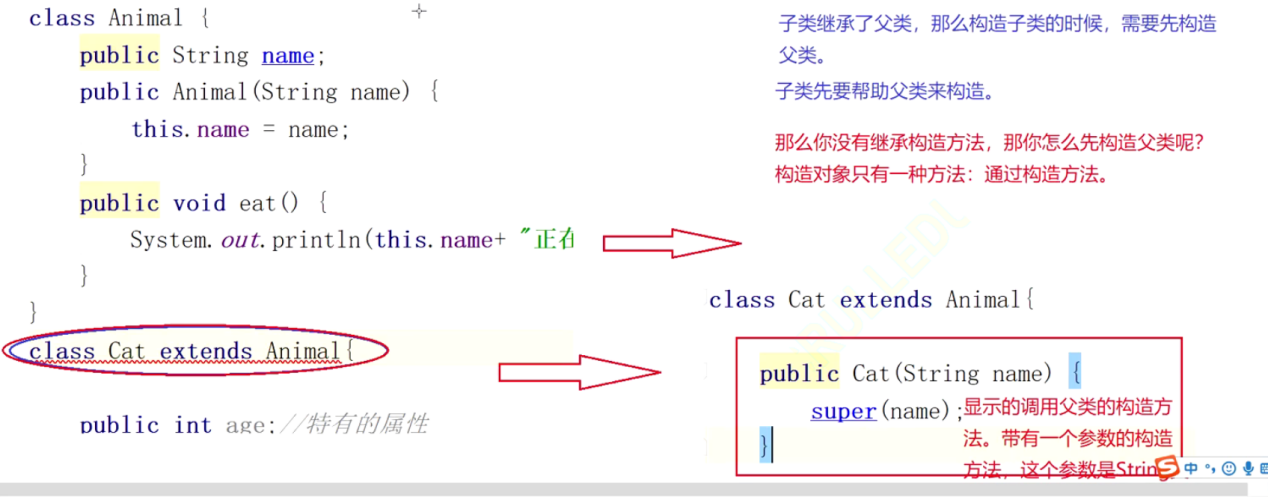
super():调用父类的构造方法

### (3).super关键字的使用

子类继承了父类，那么构造子类的时候，需要先构造父类

子类要先帮助父类构造

所以如果父类里面自己写了一个构造方法，原来的编译器自带的构造方法失效，在子类里面**必须写一个构造方法来显示调用父类的构造方法**，这个构造方法里面写super(参数)来帮助父类构造，**且super()这句话必须放在第一行**



super.data:访问父类的成员变量

super.fun():访问父类的成员方法

super():调用父类的构造方法

### (4).Protceted关键字（为继承而生）

1. 如果把字段设为 private, 子类不能访问. 但是设成 public, 又违背了我们 "封装" 的初衷。两全其美的办法就是 protected 关键字

2.



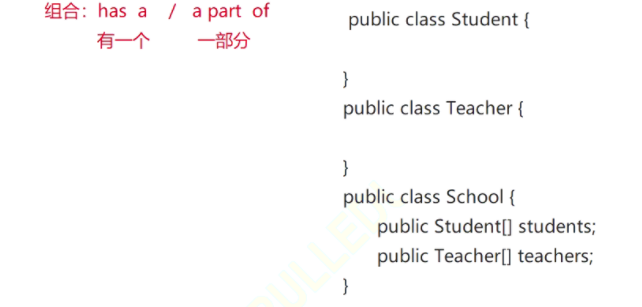
叫做不同包当中的子类呢？就是classA在包1中，classB在包2中，B继承了A，B就是A的子类，且他们在不同的包中，**此时B中要访问A中的protected类型的变量要使用super.**

具体使用方法：字段尽量private，方法用public

### (5).fianl关键字

1. final可以修饰变量--》 fianl int SIZE = 10 --》只能初始化一次
2. fianl也可以修饰类--》 **final class Animal表示不能被继承**（密封类）
3. fianl也可以修饰方法--》当前修饰的方法不能被重写。我学过重载，后面会讲到（密封方法）

## 组合

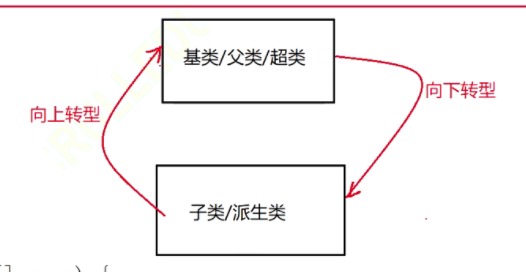


组合并没有涉及到特殊的语法(诸如 extends 这样的关键字), 仅仅是将一个类的实例作为另外一个类的字段。这是我们设计类的一种常用方式之一。

## 多态

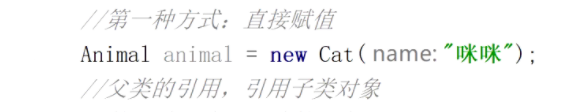
### (1).向上转型

1.向上转型

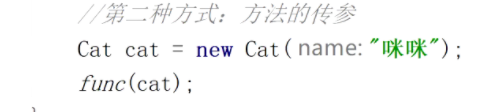
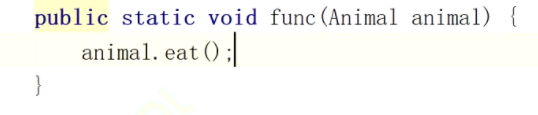


**2.向上转型的表现形式（cat给animal）**

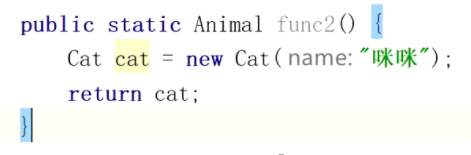
1. 直接赋值



1. 方法的传参



1. 返回值



### .动态绑定（重要）

1. 重写（在继承时出现）

* 方法名相同
* 参数列表相同
* 返回值也要相同

注意：

被private修饰的方法，不能被重写

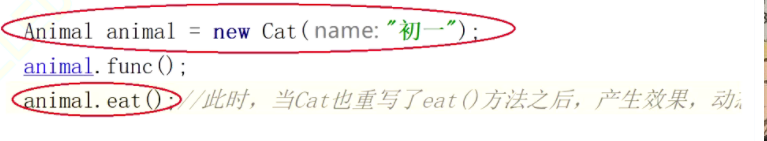
被final修饰的方法，不能被重写

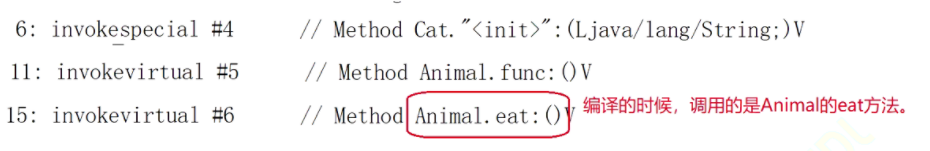
被static修饰的方法，不能被重写

需要重写的方法的访问修饰限定符，子类的访问修饰限定符（如public）一定要大于等于父类（如protected）的访问修饰符

1. **什么情况下，会发生动态绑定：**
2. **父类的引用，引用子类的对象（向上转型）**
3. **通过父类的引用，调用子类和父类重写的那个方法**

**（注意这里只能调用子类重写的那个方法，人家编译的时候调用的都是父类的方法，调用不了子类的方法，成员变量是静态绑定，所以成员变量的值在new子类对象的时候就已经确定了，动态绑定是针对一个个函数和方法而言的）**





**编译的时候，调用的还是Animal的eat方法**

**但是最后运行的时候，调用的是cat的eat方法，这就叫动态绑定，也叫运行时多态**

（静态绑定：在程序执行前，也就是编译的时候，方法已经被绑定）

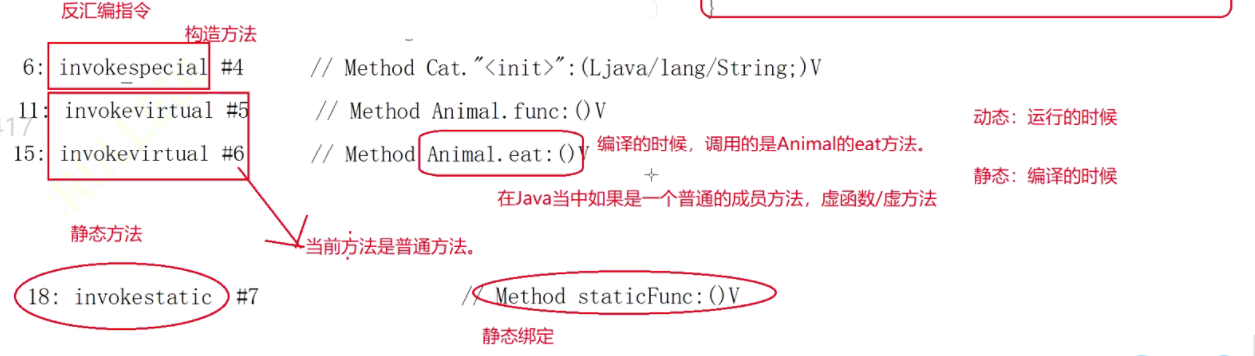
**成员变量**，fianl,static,private修饰的方法都是静态绑定

**拓展：反汇编后**

Invokespecial--------构造方法

Invokevirtual---------普通方法

Invokestatic----------静态方法



注意：在父类构造方法里面调用重写过的方法也会发生动态绑定

**面试问题：重载(overload)和重写(overwrite)的区别？**

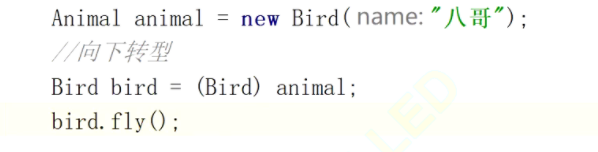


**面试问题：什么是多态？**

**首先要发生向上转型，也就是父类引用引用子类对象，无论通过是直接赋值，函数参数传递啊，返回值啊，总之要让父类引用子类对象。而且父类和子类有同名的覆盖方法，就是子类重写过父类的方法，然后通过父类的引用来调用这个同名的方法的时候，它有可能会表现出不同的行为，这个过程就是多态。**

### .向下转型

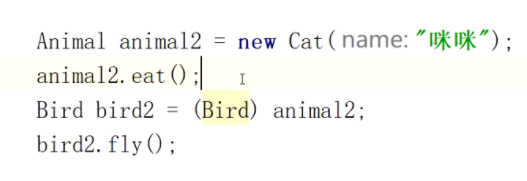
1.

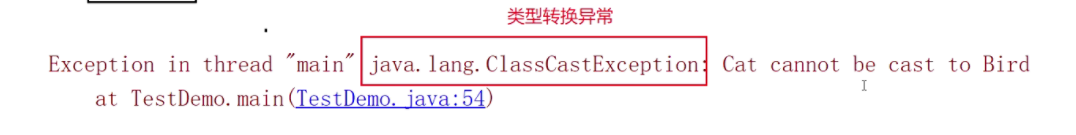


父类的引用强转为子类，然后赋值给子类。

但是这种操作基本不用，因为不安全

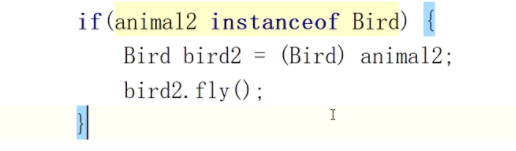
2.





1能强制转换赋值是因为原来就是动态绑定了，但是如果一开始动态绑定的不是bird，是cat后面就不能强转赋值给bird，会提示cat不能转换成brid的异常

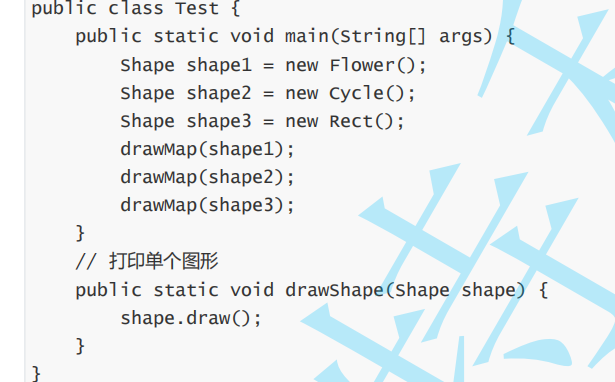
3.



所以如果要向下转型，一定要判断animal2是否能转换成bird

### .理解多态

我们可以写一些只关注父类的代码, 就能够同时兼容各种子类的情况



在这个代码中,分割线上方的代码是类的实现者编写的, 分割线下方的代码是类的调用者编写的。当类的调用者在编写 drawMap 这个方法的时候, 参数类型为 Shape (父类), 此时在该方法内部并不知道, 也不关注当前的 shape 引用指向的是哪个类型(哪个子类)的实例。此时 shape 这个引用调用 draw 方法可能会有多种不同的表现(和 shape 对应的实例相关), 这种行为就称为多态。

**封装和多态**

* 封装是让类的调用者不需要知道类的实现细节.
* 多态能让类的调用者连这个类的类型是什么都不必知道, 只需要知道这个对象具有某个方法即可。（调用者写方法的时候，只要把参数类型设置为父类shape，不需要管以后是哪个子类对象会被当做参数传进来）

因此, 多态可以理解成是封装的更进一步, 让类调用者对类的使用成本进一步降低

### .多态的好处

1. 类调用者对类的使用成本进一步降低
2. 能够降低代码的 "圈复杂度", 避免使用大量的 if - else

（的计算一段代码中条件语句和循环语句出现的个数, 这个个数就称为 "圈复杂度"）

1. 可扩展能力更强（如果要再加一种形状就很好加）

## 抽象类

### .概念

1. 抽象方法：abstract修饰的方法
2. 抽象类：包含抽象方法的类（只要有抽象方法就要是抽象类）

抽象类用法：反正你这个类不用实例化，方法也会被重写，那就定义成抽象类（如Shape）

### .注意点

1. 抽象类是不能被实例化的
2. 抽象类和普通类的区别就在于：多了一个抽象方法，不能被实例化。
3. 如果一个普通类继承了一个抽象类，那么这个普通类一定要重写抽象类当中的抽象方法
4. 抽象方法是一定要被重写的，此时也包含了重写的规则（子类的访问修饰限定符一定要大于父类的）
5. 抽象类中可以包含其他的非抽象方法,也可以包含字段。这个非抽象方法和普通方法的规则都是一样的, 可以被重写,也可以被子类直接调用。
6. 抽象方法不能是 private 的
7. 如果一个类（A）继承了抽象类，且不想重写抽象类方法，那么这个类（A）就只能定义为抽象类了。最后，一旦有其他类继承了这个抽象类，那么一定要重写这个抽象方法。---》**出来混总是要还的**

**8.抽象类也可以发生向上转型和多态**

### .抽象类的作用

1. 抽象类存在的最大意义是为了被继承
2. 抽象类本身不能被实例化, 要想使用, 只能创建该抽象类的子类. 然后让子类重写抽象类中的抽象方法
3. 抽象类的出现，其实也是方便程序员进行检查代码（一继承就知道了）

## 接口（interface）

### .基本概念和注意点

1.接口里面的方法默认都是public abstract

2.所有的成员变量一定是public static final的

（以上1,2两种在接口里面都是默认的，一般都不写）

3.接口也是不可以被实例化的

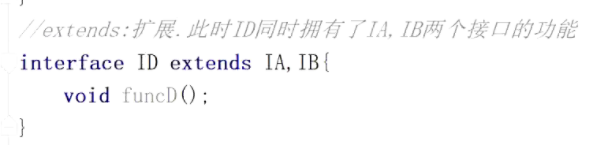
4.类和接口的关系是implements的关系

5.接口也是可以发生向上转型的，也就是可以发生多态

6.一个类可以实现多个接口



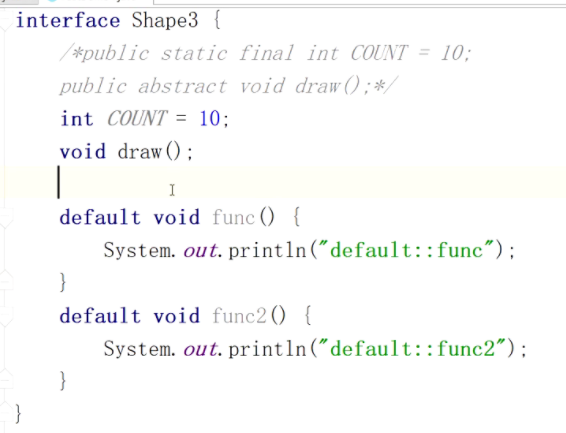
1. 接口之间可以互相扩展

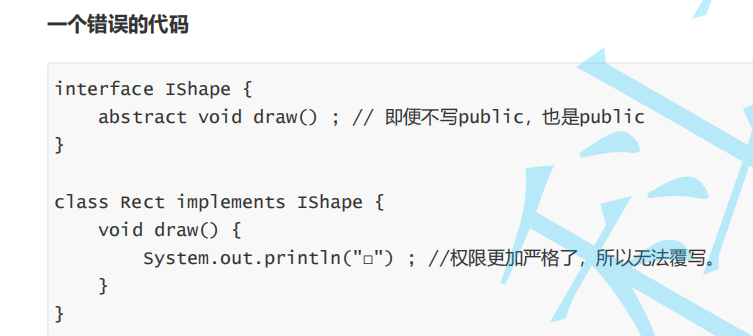


一旦有类实现了接口ID，那么这个类需要重写的就是IA,IB,ID三个接口的抽象方法

8.接口的出现，弥补了java的缺陷，可以进行多继承了

**注：jdk1.8的新特性：在接口中，如果加上default，接口就可以有具体的实现**





在接口里面的方法默认是public abstract的，所以可以不写，但是当有类来实现接口的时候，重写里面的方法一定要加上public，不然编译器默认是包访问权限，小于public，固不能重写

### .实现多个接口

1. 如有一个父类Animal,然后三个接口run,swim,fly

对于Cat,他就可以继承Animal,然后实现run这个接口

（Cat extends Animal implements run）

对于Duck，他就可以继承Animal，然后实现run和swim这两个接口

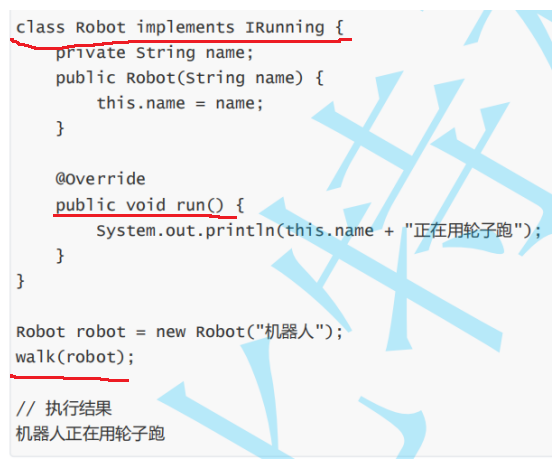
（Duck extends Animal implements run,swim）

如果要把这些都写到动物里面，但是不是所有的动物都会飞，跑，游泳，所以继承Animal只是为了让其拿到所有Animal的公有属性，而各种动物不同的行为就可以通过接口来实现

### .接口的好处



**甚至参数可以不用是动物，只要会跑就行（如robot）**



**让程序猿忘记类型. 有了接口之后, 类的使用者就不必关注具体类型, 而只关注某个类是否具备某种能力。**

像在这个写这个walk方法的时候，参数定义为IRunning接口，里面在使用这个参数的时候，不需要知道传进来的具体是什么，只需要知道它是会跑的就行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**封装**让我们忘记类的实现细节

**继承**让我们忘记传进来的是什么类型，只要是子类就行（Animal animal，传入动物的子类）

**接口**让我们忘记所有，只需要知道它有这种特性就行（IRunning irunning，传入会跑的东西，也就是能实现这个接口的东西）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### .接口的使用（重要）



Comparable