## 1.常见的DOS命令

dir---展现指定目录下的所有的子文件和子目录---directory

mkdir---新建目录---md

rmdir---删除目录---从计算机中彻底移除，不可撤销---rd---要求删除的目录必须为空目录

del---删除文件

cls---清空屏幕---clear screen

exit---退出命令提示符

notepad---打开记事本

calc---打开计算器

mspaint---打开画图

## Java

1995年SUN推出的

Oak---James Gosling---爪哇岛---JAVABean NetBean

1.0->1.1->1.2(GUI)--1.3--1.4--1.5/5.0---6---7.0---1.8(十大特性)---1.9---10（18.3）---18.9---19.3

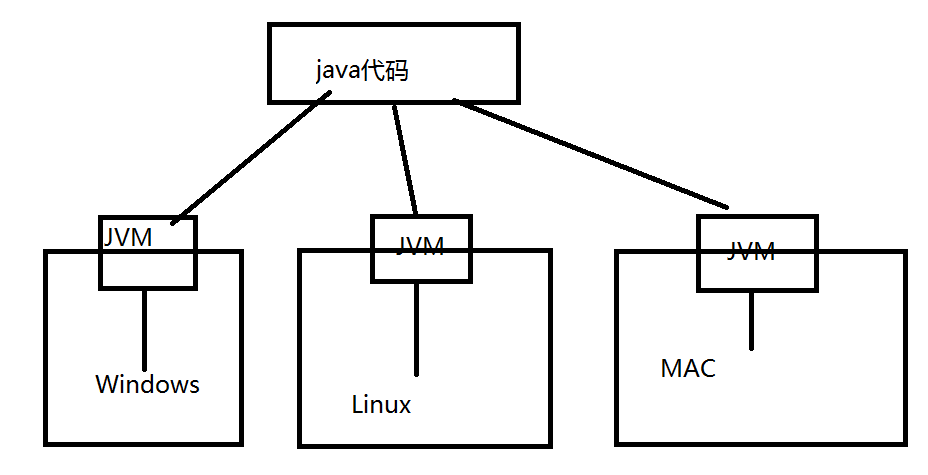
J2SE/JAVASE --- Standard Edition---标准版/基础版

J2EE/JAVAEE --- Enterprise Edition--- 企业版/商务版

J2ME/JAVAME --- Micro Edition --- 微型版/移动版 ----Android---Kotlin

### 跨平台

JVM（Java Virtual Machine）---Java虚拟机---将Java代码转化为对应的操作系统能够理解的指令。---不同的操作系统有不同的虚拟机与之对应，同一段代码交给虚拟机之后，虚拟机再转化给操作系统。---JAVA语言能够跨平台的前提是JVM---JVM不是跨平台的。



### JDK、JRE、JVM

JVM---Java Virtual Machine---是Java语言能够跨平台的条件

JRE---Java Runtime Environment --- Java运行时环境 --- JVM+核心类库

JDK---Java Development Kit --- Java开发工具包 --- 开发工具+JRE

[www.oracle.com](http://www.oracle.com)

注意：安装路径中不要出现空格和中文

将Java代码翻译成当前操作系统所理解的指令---编译---编译完成之后会产生一个class文件，这个文件称之为字节码文件---这个字节码文件就是给当前操作系统看的指令文件

### 入门程序

class Demo{

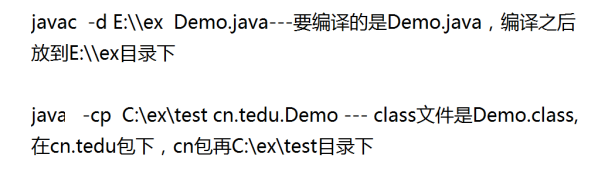
public static void main(String[] args){

System.out.println("Hello World !");

}

}

1. 主函数public static void main(String[] args){}---是程序的入口
2. class文件名和类名对应的---每一个类在编译完成之后会产生一个class文件
3. 如果一个类用public来修饰（公共类），那么要求这个类名和Java 文件名要一致
4. 一个Java文件中可以定义多个类，但是只能有一个公共类
5. 如果添加了包名，在运行的时候需要先将class文件放到对应的包的目录中，然后再java 包名.类名
6. javac -d 编译之后的包存放的目录 要编译的Java文件
7. java -cp class文件存放的路径 包名.类名



. 表示当前目录

环境变量---给当前的操作系统指定命令的运行路径的

JAVA\_HOME=D:\Java\jdk1.7.0\_75

Path=%JAVA\_HOME%\bin;

D:\Java\jdk1.7.0\_75\bin;

### 关键字

有特殊含义的单词

在Java中一共有53个关键字---其中有2个关键字目前没有使用：goto，const---保留字

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 用于定义数据类型的关键字 | | | | | | | |
| class | interface | byte | | short | int | | long |
| float | double | char | | boolean | void | | enum |
| 用于定义数据类型值的关键字 | | | | | | | |
| true | false | null | |  |  | |  |
| 用于定义流程控制的关键字 | | | | | | | |
| if | else | switch | | case | default | | while |
| do | for | break | | continue | return | |  |
| 用于定义访问权限修饰符的关键字 | | | | | | | |
| private | protected | public | |  |  | |  |
| 用于定义类、函数、变量修饰符的关键字 | | | | | | | |
| abstract | final | static | | synchronized | |  |  |
| 用于定义类与类之间关系的关键字 | | | | | | | |
| extends | implements | |  |  |  | |  |
| 用于定义建立实例、判断实例的关键字 | | | | | | | |
| new | this | super | | instanceof |  | |  |
| 用于异常处理的关键字 | | | | | | | |
| try | catch | finally | | throw | throws | |  |
| 用于包的关键字 | | | | | | | |
| package | import |  | |  |  | |  |
| 其他修饰符关键字 | | | | | | | |
| native | strictfp | transient | | volatile | assert | |  |

### 标识符

在程序中自定义的名称

#### 规则

由字母、数字、\_、$组成,但是不建议使用$

Java支持中文命名，但是不建议

数字不能开头

不能使用关键字

尽量见名知意

#### 驼峰命名法

类名/接口名：如果由多个单词组成，要求每一个单词的首字母大写 --- PlayBasketball,Demo

变量名/方法名：如果由多个单词组成，第一个单词的首字母小写，其余单词的首字母大写 --- playBasketball,work

常量名：所有单词全部大写，每一个单词之间用\_隔开 --- PLAY\_BASKETBALL ,SAM

包名：所有单词全部小写，每一层包之间用.隔开 --- cn.tedu

复习：

1. Java技术结构：j2se,j2ee,j2me
2. Java的跨平台：JVM是Java能够跨平台的前提---Java语言是跨平台，JVM不是跨平台的
3. 环境变量的配置：JAVA\_HOME=JDK的安装目录 Path=%JAVA\_HOME%\bin;
4. javac -d 路径 要编译的Java文件--编译
5. java -cp 路径 包名.类名---运行
6. 关键字：53个---const，goto---关键字都是小写---System,String
7. 标识符：字母、数字、\_、$组成，数字不能开头，不能使用关键字，见名知意，大小写敏感---Windows系统区分大小写么？---不区分

## 2.计算机常量,变量,进制,数据类型

### 注释

在代码中解释程序的文字

// 注释文字 --- 单行注释

/\*注释文字\*/ --- 多行注释

/\*\*注释文字\*/ --- 文档注释---可以利用javadoc将注释内容来提取处出来形成文档---只能放在类或者函数上，javadoc命令只能用于提取公共类

### 计算机常量

本身不可改变的量

整数常量：所有的整数 3，4，100

小数常量：所有的小数 1.02,5.36，8.7209

字符常量：将一个符号用 ‘’ 标志起来就构成一个字符常量 ‘a’ ‘+’ ‘1’ ‘ ’

字符串常量：将多个字符用 “”标志起来就构成了一个字符串常量 “abc” “12w” “a” “”

布尔常量：只有两个值---true/false---用于表示逻辑值

空常量：null

考虑：2--整数, 2.0---小数 ‘2’--字符, “2”---字符串, ‘2.0’--非法, “2.0”--字符串 分别是什么常量？

### 进制

计数方式

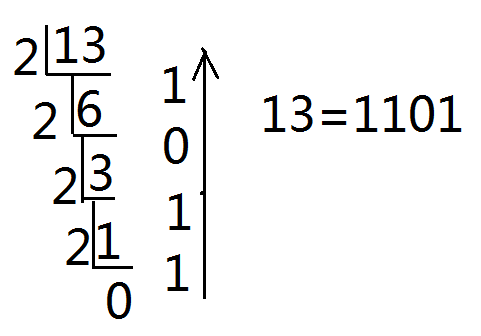
二进制:0-1，满2进1 在JDK1.7以前Java代码中不允许直接表示二进制数字，从jdk1.7 开始允许在代码中使用二进制数字，要求以0b/0B开头 1+1=10 11+1=100 0b100100 0B10111

八进制：0-7，满8进1 以0开头标志八进制数字 7+1=10 16+1=17 17+1=20 015 07 012 027

十进制：0-9，满10进1

十六进制：0-9，A-F/a-f，满16进1 以0x/0X开头标志十六进制数字 9+1=a a+1=b f+1=10 0x23 0xa 19+1=1a 1f+1=20

十进制转换成二进制：不断地除以2取余数，将余数倒排



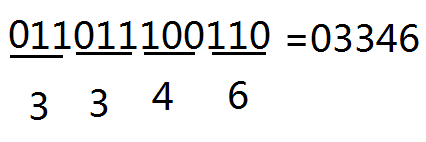
二进制转化成十进制：从低位次开始，每一位乘以2的位次之幂，然后将积求和



十进制转化为其他进制：就是除以对应的进制然后取余倒排

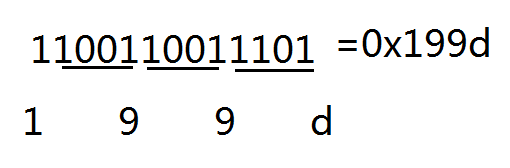
其他进制转化为十进制：乘以对应进制的位次次幂然后求和

二进制转换成八进制：从低位次开始，每三位化为一组，产生一个八进制数字，最高位如果不足三位，则补0---三变一



八进制转换为二进制：一变三

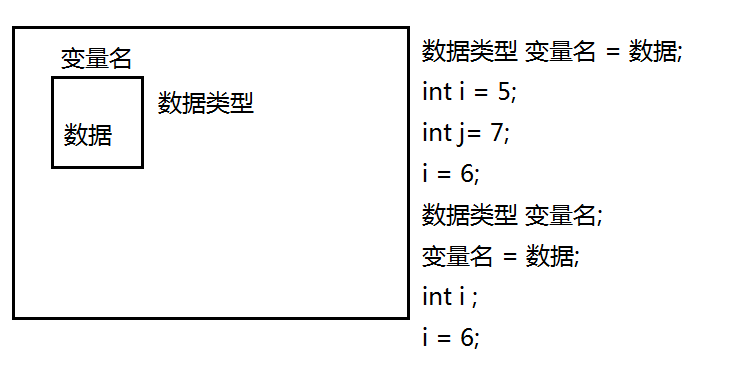
二进制转化为十六进制：四变一



十六进制转化为二进制：一变四

### 变量

存储数据的



变量名不能重复。

int i, j,k;---允许一次性定义多个同一类型的变量，用,隔开

int i;

double j;

变量先定义后使用，先给值后使用，在哪儿定义在哪儿用

扩展：每一个二进制数字称之为一位(bit---b)--->字节（Byte---B）---1B=8b 1KB=1024B 1MB=1024KB -> GB -> TB

## 数据类型

### 基本数据类型

数值型

整数型

byte---字节型---1个字节 --- -27~27-1 - -128~127

byte b = 25; byte b2 = 128;---false

short---短整型---2个字节 --- -215~215-1 --- -32768~32767

short s = 3; short s = -109;---几乎不用

int---整型---4个字节--- -231~231-1 --- 2.1\*1010~2.1\*1010---整数默认为int类型

System.out.println(6); int i = 129;从JDK1.7开始int j = 3\_186\_485\_318; ->编译完成之后就是int j = 3186485318;

long---长整型---8个字节--- -263~263-1 --- -9\*1018~9\*1018---后边添加L/l标志是一个长整型

long l = 31864853188L;

浮点型

float---单精度---4个字节 --- -1038~1038---要求必须以F/f标志float类型的数字

float f = 3.23f;

double---双精度---8个字节--- -10308~10308 ---小数默认为double

System.out.println(3.4); double d = 4.2; double d2 = 5.6D; double d3 = 6.01d;

double d = 3e3; --- 科学计数法 3\*103=3000.0

double d = 0x3p3;---科学计数法 3\*23=24.0

char---字符型---2个字节--- 0-216~1 --- 0~65535

char c = ‘中’; char c2 = ‘\u52fa’;

‘a’ --- 97 ‘A’---65 ‘0’---48

‘中’ gbk---2 utf-8--3 utf-16---2

‘a’ iso---1 gbk---1 u8-1 u16-1

#### 转义字符

‘\t’ ---table 制表符 ‘\r’---return 回车 ‘\n’---next/newLine 换行

‘\\’ ---\ ‘\’’ --- 单引号 ‘\”’ --- 双引号

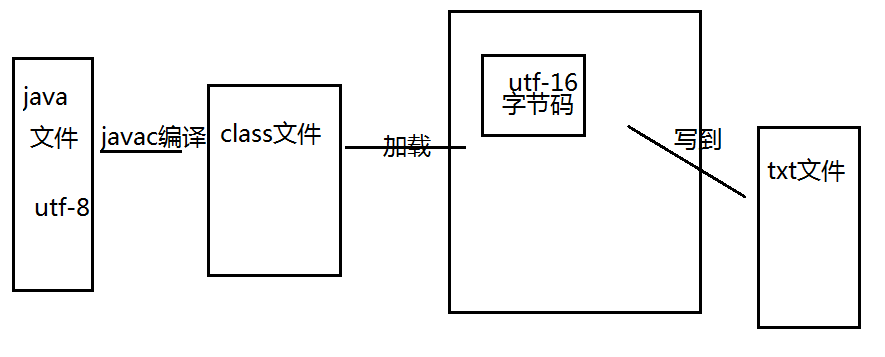
编码：将数据按照指定的规则转化为数字---编码表---码表---ASCII

规定所有的码表默认兼容西欧码表

ISO-8859-1---西欧码表---1个字符1个字节

gb2312---国标码---2个字节一个字符---收录了常见的简体汉字以及一部分的常见繁体汉字---gbk

Unicode编码体系---收录了常见语言的常见字符---Java采用了Unicode编码体系--- utf-8---三个字节一个字符，Java文件默认是使用utf-8；Java文件编译完成之后会产生对应的class文件，class文件加载到内存之后是以utf-16形式存储的。utf-16规定一个字符2个字节



boolean ---布尔型---true/false---内存大小根据jdk版本和操作系统来确定

boolean b = true;

引用数据类型

数组[] 类class 接口interface

### 数据类型的转换

#### 自动类型转换/隐式转换

byte b = 120;

int i = b;

float f = 3.2f;

double d = f;

规律一：小的类型能够自动转化为大的类型

int i = 6;

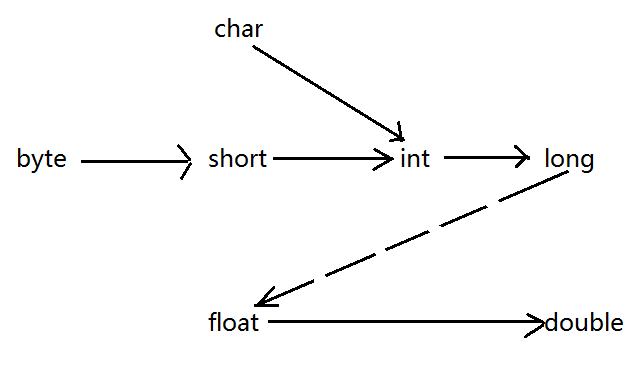
float f = i;

规律二：整数能够自动转化为小数，但是可能会有精度损失

char c = ‘a’;

int i = c;

规律三：字符可以自动转化为整数



float f = 23;---true---23是整数，默认为int

float f = ‘a’;---true

long l = 231;---true

short s = ‘a’;---true--- ‘a’是一个常量，因此在编译的时候值是确定的，所以在编译的时候jvm会去检查 ‘a’是否在short的取值范围内，如果范围符合，则可以转换

char c = 97;---true---97是一个常量，因此在编译的时候取值也是确定的，所以在编译的时候jvm会发现97在char类型的范围内，则可以转换

short s = 97;

char c = s;---false---s是一个变量，在编译的时候只能确定一件事儿---只能确定s的类型short类型，就只能比较short和char类型的范围是否符合---由于char不能完全包含short的取值范围，所以会产生范围的冲突---不允许转换，编译不通过

char c = ‘a’;

short s = c;---false

#### 强制类型转换/显式转换

double d = 3;

int i = (int)d;

double d = -3.9;

int i = (int)d; --- i = -3;

注意：小数强转为整数的时候会舍弃小数部分

int i = 128;

byte b = (byte)i;

### 数据的原反补三码

任意一个数据都有原码、反码、补码这么三种形式。

数据在计算机中是以补码形式来存储的，因此实际上计算的也是数据的补码。

对于正数，原反补三码一致。

int i = 5;

原码：00000000 00000000 00000000 00000101

反码：00000000 00000000 00000000 00000101

补码：00000000 00000000 00000000 00000101

对于负数，反码是在原码的基础上最高位不变，其余位1->0,0->1;补码在反码的基础上+1---最高位是符号位，0表示正数，1表示负数

int i = -3;

原码：10000000 00000000 00000000 00000011

反码：11111111 11111111 11111111 11111100

补码：11111111 11111111 11111111 11111101

## 3.运算符

### 算术运算符

+ - \* / % ++ --

注意：

1. byte/short/char在运算的时候会自动的提升为int类型
2. 整数运算完成之后的结果一定是整数
3. 当小类型和大类型一起运算的时候结果一定是大的类型
4. byte b = 4 + 1; --- 4 和1都是常量，值确定，所以在编译的时候进行优化，编译完成之后结果就是byte b = 5;
5. 整数/0 -- ArithmeticException---算术异常 任意一个非零小数/0;任意一个非零数字/0.0---Infinity 0.0/0 0/0.0 0.0/0.0---NaN---Not a Number---非数字

% 取模---取余

7%3=1 2%5=2

-18%5=-3 -56%11=-1

18%-5=3 56%-11=1

-18%-5=-3 --- 对于负数的取余，先忽略符号按照正数取余来运算，运算完成之后看%左边数字的符号，如果左边为正，结果为正，左边为负，结果为负。

3%1.2=0.6 4.73%1.54=0.11

注意：小数无法精确运算---绝大部分小数在内存中无法精确存储

++/-- 自增/自减

如果++/--在前，先自增/自减然后参与后续运算

如果++/--在后，先将值取出来参与后续运算然后再自增/自减

int i = 5;

int j = ++i; -> 先将i自增为6，然后将6赋值给j，所以j的值也是6；

int i = 5;

int j = i++;-> 先将i的值5取出来，然后i自增为6，最后将5赋值给j

int i = 3;

int j = ++i + 1; -> 先将i自增为4，然后将i的值4取出来参与+1，最后将4+1赋值给j，所以j的值为5

int i =3;

int j = i++ + 1; -> 先将i的值3取出来参与+1运算，i再自增为4，然后将3+1的值赋值给j，所以j的值是4

int i = 5;

int j = ++i \* 2; -> j = 12

int i = 5;

int j = i++ \* 2; -> j = 10

int i = 9;

int j = i++ + ++i; 9 + 11 -> j = 20

int i = 9;

int j = ++i + i++; 10+10->j=20

注意：byte/short/char可以参与++/--运算，结果依然是原来的类型---底层做了强制转换

### 赋值运算符

= += -= \*= /= %= &= |= ^= <<= >>= >>>=

int i = 8;

i += 2; -> i = i + 2; -> i = 10

int i = 6;

i -= 4; -> i = 2;

注意：

1. 除了=以外，其余的符号要求变量必须先有值才能使用
2. byte/short/char可以参与赋值运算

int i = 5;

i += i -= i\*= i++; -> i=-15

i = 5 + ( 5 - (5 \* 5));

int i = 5;

i = i++; ---先将i的值5取出来，然后i自增为6，最后将5赋值给i -> i = 5

复习：

1. 注释：在程序中解释说明的文字 // /\* \*/ /\*\* \*/
2. 计算机常量：整数、小数、字符、字符串、布尔、空
3. 进制：二进制--0b，八进制--0，十六进制---0x；要求掌握进制的换算
4. 变量：先定义后使用，先赋值后使用，在哪儿定义在哪儿使用
5. 数据类型：基本类型---byte/short/int/long/float/double/char/boolean，引用类型---数组、类、接口;记住byte的取值范围；整数默认为int，小数默认为double；
6. 数据的转换：自动转换（小转大，整转小）、强制转换
7. 运算符：算术运算符、赋值运算符

### 关系运算符

==等于 !=不等 > < >= <=

3 == 4 --false

3 != 4 --true

instanceof --- 关键字 --- 判断对象和类的关系的---不能判断基本类型

System.out.println(“abc” instanceof String); --- true

System.out.println(5 instanceof int); ---错误

### 逻辑运算符

&与---And |或---Or !非---Not ^异或---Xor &&--短路与 ||--短路或

true&true=true true&false=false false&true=false false&false=false

true|true=true true|false=true false|true=true false|false=false

!true=false !false=true

true^true=false true^false=true false^true=true false^false==false

&&：如果&&左边的结果为false，那么&&右边的表达式就不再运算---短路

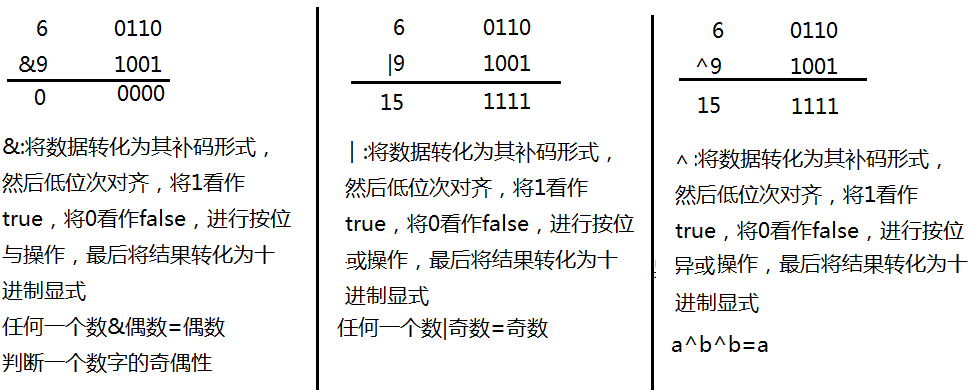
||：如果||左边的结果为true，那么||右边的表达式就不再运算

注意：||能够把&&给短路掉，但是&&不能把||短路掉

### 位运算符

注意：位运算符针对的整数的补码

&与 |或 ^异或 <<左移 >>右移 >>>无符号右移 ~取反



#### 交换两个变量的值

int i = 3, j = 7;

第一种方式：利用第三方变量---追尾法

int temp = i;

i = j;

j = temp;

第二种方式：加减法

i = i + j;

j = i - j; -> j = i + j - j; -> j = i;

i = i - j; -> i = i + j - i; -> i = j;

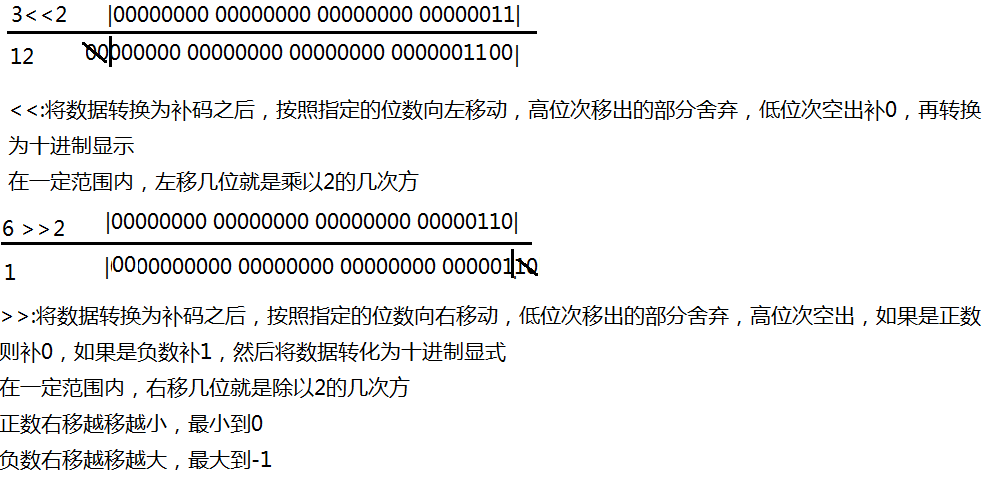
第三种方式：异或法

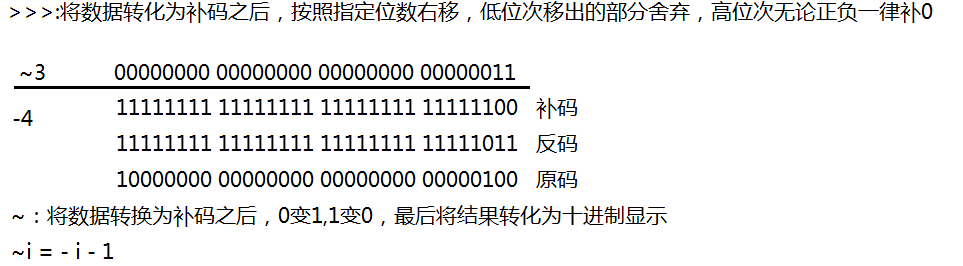
i = i ^ j;

j = i ^ j; -> j = i ^ j ^ j; -> j = i;

i = i ^ j; -> i = i ^ j ^ i; -> i = j;

i ^= j; j ^= i; i ^= j;





### 三元运算符

逻辑值? 表达式1:表达式2

执行顺序：如果逻辑值为true，执行表达式1；反之，执行表达式2

三元运算一定有结果，结果可以用变量来接住

double d = a > b ? 3 : 5.2;

double d2 = i > j ? 1.2 : ‘b’;

m > n ? true : 3; --- 不可以

两个表达式结果的类型要么一致，要么其中一个能够自动转化为另一个

i, j, k

int max = i > j ? (i > k ? i : k) : (j > k ? j : k);---三元表达式的嵌套

练习： >=90-A >=80-B >=70 -C >=60-D <60-E

#### 从控制台输入数据

import java.util.Scanner;

Scanner sc = new Scanner(System.in);

int n = sc.nextInt(); // 输入整数

double d = sc.nextDouble(); // 输入小数

String str = sc.next(); // 输入字符串

String str2 =sc.nextLine(); // 输入字符串

## 流程控制

复习：

1. 关系运算符： ==
2. 逻辑运算符：^ && ||---短路特性
3. 位运算符：& | ^ << >> >>> ~ --- 针对的是整数的补码
4. 三元运算符： 逻辑值?表达式1 :表达式2;

总结：运算符的优先级

() ++ -- ~ ！ 算术 << >> >>> 关系 逻辑 & | ^ 三元 赋值

1. 判断结构：if,if-else, if-else if
2. 选择结构：switch-case --- byte/short/char/int,String；break

### 4.循环结构,数组

#### while循环

while(逻辑值){

Code;

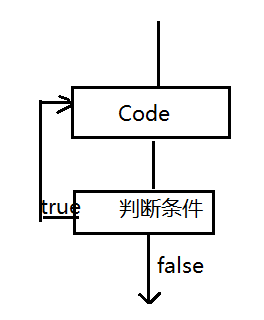
}

do{

Code;

} while(逻辑值);

无论如何代码块都会执行一次。



#### for循环

定义循环变量;

while(判断循环变量是否成立){

代码块;

改变循环变量;

}

for(定义循环变量;循环条件;改变循环变量){

代码块;

}

#### break和continue

break：可以在选择和循环结构中使用。表示终止一层循环

continue：只能用在循环结构中。表示跳出本次一层循环继续下次的循环

在Java中，break和continue支持标号形式

练习：输入一个数字，判断这个数字是否是一个质数---除了1和本身没有别的因子

int num = s.nextInt();

2~num -1

练习：打印1000以内的所有的质数

1. 先获取1-1000这1000个数字
2. 获取到这个数字之后要判断这个数字是否一个质数

练习：打印九九乘法表

1\*1=1

1\*2=2 2\*2=4

1\*3=3 2\*3=6 3\*3=9

思考：如果用一个循环打印九九乘法表？

练习：输入一个数字，分解数字的质因数

120 = 2 \* 2 \* 2 \* 3 \* 5

复习：

1. 循环结构：while，for ---循环的嵌套
2. break和continue：break可以用于循环和选择结构continue只能用于循环结构

## 5.数组

存储多个同一类型的数据的容器---大小是固定的

会对数组中的元素自动的进行编号，编号从0开始的---下标

### 定义格式

数据类型[] 数组名 = new 数据类型[数组的大小/长度]; --- 动态初始化

int[] arr = new int[5]; --- 表示定义了一个能存储5个整型元素的数组

arr[2] = 5; --- 表示向arr数组中下标为2的位置放入元素5

System.out.println(arr[2]);

数据类型[] 数组名 = new 数据类型[]{元素1, 元素2, ... , 元素n}; --- 静态初始化

int[] arr = new int[]{2,5,1,6,4,8};---同时规定了数组的长度就是6

arr[3] = 7;

数据类型[] 数组名 = {元素1, 元素2, ... , 元素n}; --- 不能分开定义

int[] arr = {2,1,6,4,3,1,8};

注意：[]在前，紧跟数据类型的；[]在后紧跟具体的变量的

### 内存

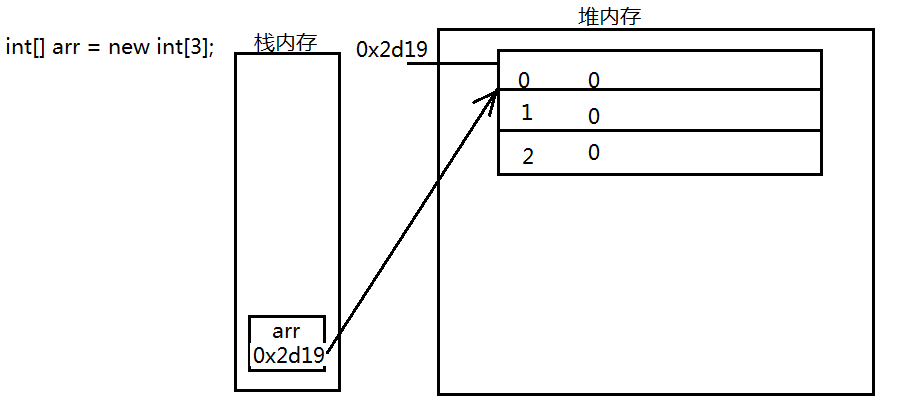
Java将内存分为了5块：栈内存、堆内存、方法区、本地方法栈、寄存器

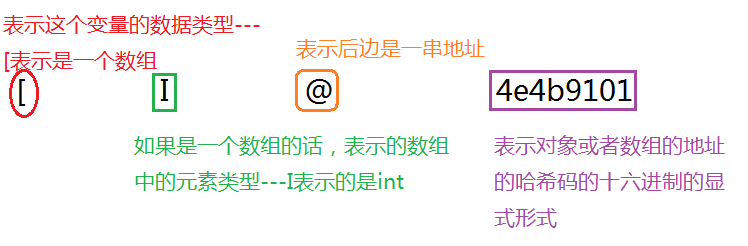
栈内存：存储变量的。栈内存不会自动给变量赋值。而且变量用完之后会从栈内存中立即移除以释放内存空间

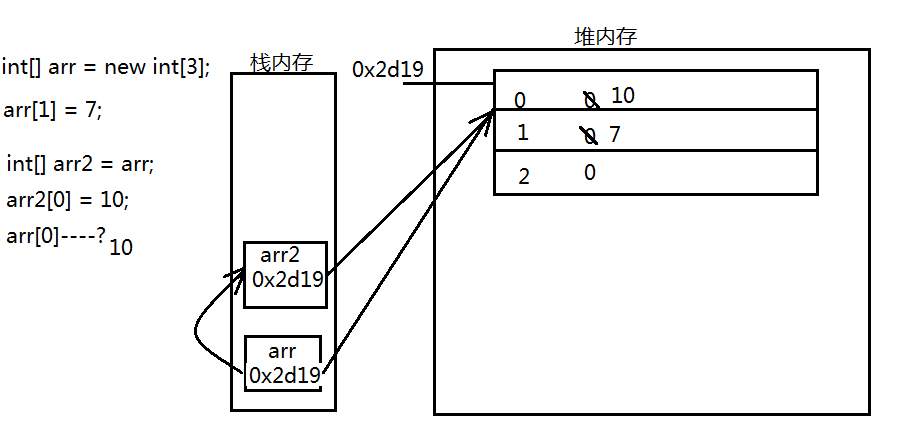
堆内存：存储的是数组或者对象。碰到new关键字的时候会到堆内存中开辟空间。堆内存会对存入其中的元素自动的赋予一个默认值---byte/short/int-0, long-0L, float-0.0f, double-0.0, char- ‘\u0000’, boolean-false, 除了这八种基本类型以外的所有的类型的默认值都是null。 堆内存中的数据用完之后会在不定的某个时刻被回收。

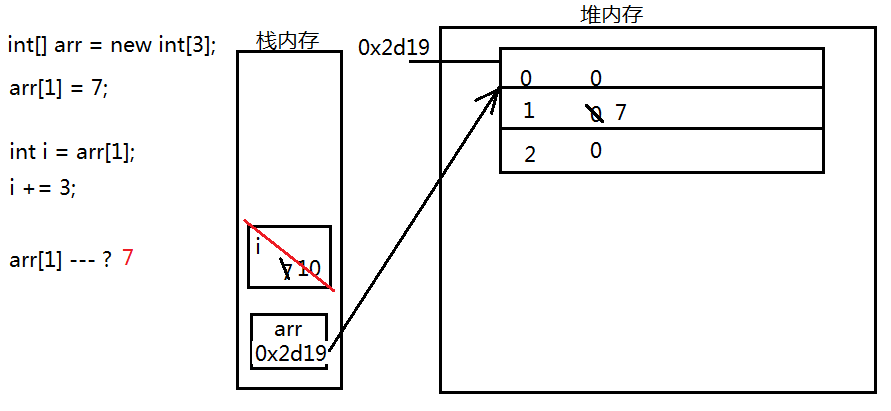
boolean[] bs = new boolean[3]; bs[0] --->false

String[] ss = new String[5]; ss[0] ---> null









如果操作的下标超过了最大下标，则会出现ArrayIndexOutOfBoundsException

### 数组的应用

1. 获取数组的元素---数组名[下标]
2. 获取数组的长度
3. 遍历数组

// 利用普通的for循环来遍历数组

for(int i = 0; i < arr.length; i++){

System.out.println(arr[i]);

}

// 增强for循环

for(int i : arr){

System.out.println(i);

}

Arrays.toString(arr);

1. 获取数组中的最值（最大值/最小值）

// 定义一个变量来记录最大值

int max = arr[0];

// 拿着max依次和其他的元素比较

for(int i = 1; i < arr.length; i++){

if(max < arr[i])

max = arr[i];

}

System.out.println(max);

// 定义一个变量记录最大值的下标

int max = 0;

for(int i = 1; i < arr.length; i++){

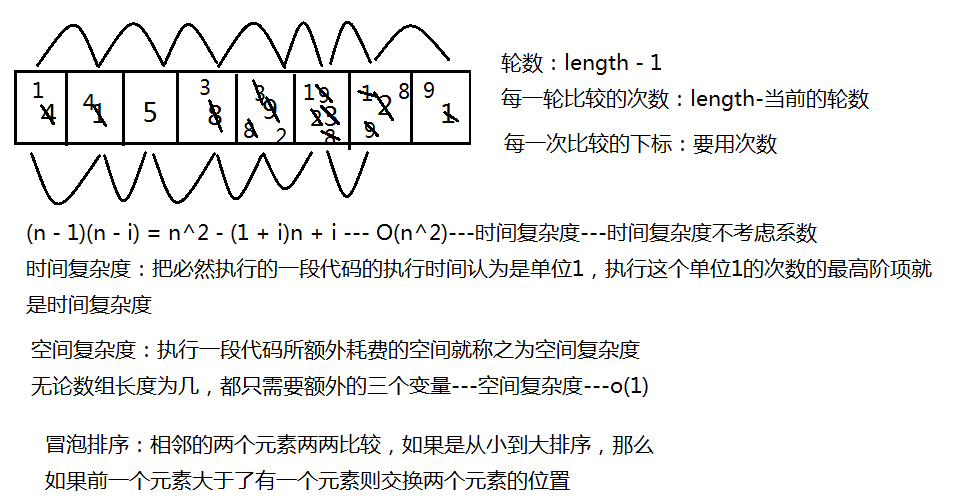
if(arr[max] < arr[i])

max = i;

}

System.out.println(arr[max]);

1. 数组排序



// 冒泡排序

for(int i = 1; i < arr.length; i++){ // 控制轮数

for(int j = 1; j <= arr.length - i; j++){ // 控制次数和下标

if(arr[j - 1] > arr[j]){

int temp = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = temp;

}

}

}

System.out.println(Arrays.toString(arr));

## 数组

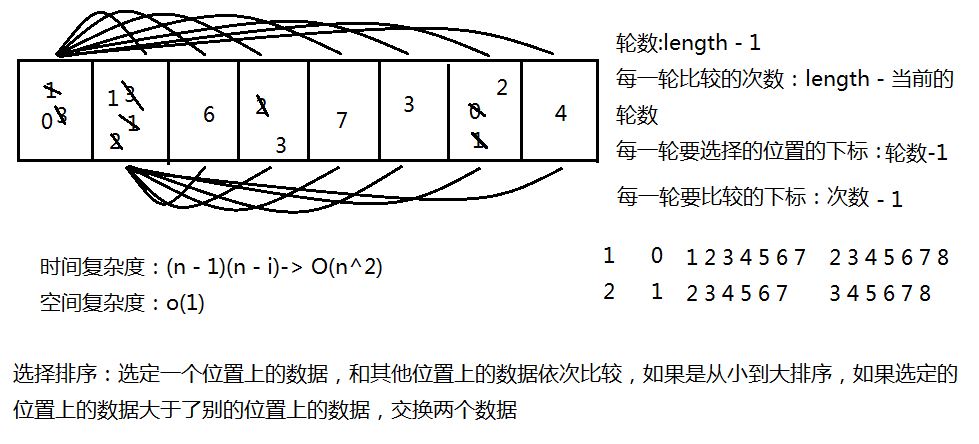
定义格式---3种

数组的内存---Java将内存分为了5块：栈内存、堆内存、方法区、本地方法栈、寄存器---数组是存在堆内存中。会自动的赋予默认值

### 数组的应用

1. 获取数组的元素
2. 数组的长度
3. 遍历数组
4. 获取最值
5. 数组的排序

冒泡排序：时间复杂度---O（n2）,空间复杂度---o(1)



// 选择排序

for(int i = 1; i < arr.length; i++){ // 控制轮数和选择的下标

for(int j = 1 + i; j <= arr.length ; j++){ // 控制次数和比较的下标

if(arr[i - 1] > arr[j - 1]){

int temp = arr[i - 1];

arr[i - 1] = arr[j - 1];

arr[j - 1] = temp;

}

}

}

System.out.println(Arrays.toString(arr));

Arrays.sort(数组);---只能升序排列，时间复杂度是O(nlogn)

1. 数组的反转

// 首尾互换

for(int start = 0, end = arr.length - 1; start < end; start++, end--){

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

}

1. 数组元素的查找

int[] arr = {1,3,5,9,14,25,53,72};

int number = 58;

// 折半查找

// 时间复杂度是O(logn)

// 空间复杂度是o(1)

int min = 0;

int max = arr.length - 1;

int mid = (min + max) / 2;

while(arr[mid] != number){

if(arr[mid] > number){

max = mid - 1;

} else {

min = mid + 1;

}

if(min > max){

mid = -1;

break;

}

mid = (min + max) / 2;

}

System.out.println(mid);

1. 数组的复制

System.arraycopy(要复制的数组,要复制的起始下标,要存放的数组,要存放的起始下标,要复制的元素的个数);

数组的扩容/缩容---本质上就是数组的复制，所以扩容之后地址必然发生了改变

数组 = Arrays.copyOf(要扩容的数组, 扩容之后的长度);

### 二维数组

存储数组的数组---二维数组---二维数组中的每一个元素实际上是一个一维数组

#### 定义格式

数据类型[][] 数组名 = new 数据类型[二维数组的长度][包含的一维数组的长度];

int[][] arr = new int[3][5]; --- 定义包含3个一维的整型数组，每个一维数组能存储5个整数。

数据类型[][] 数组名 = new 数据类型[二维数组的长度][];

int[][] arr = new int[3][]; --- 定义了包含3个一维的整型数组，没有规定每一个一维数组的大小

arr[0] = new int[3];

arr[1] = new int[7];

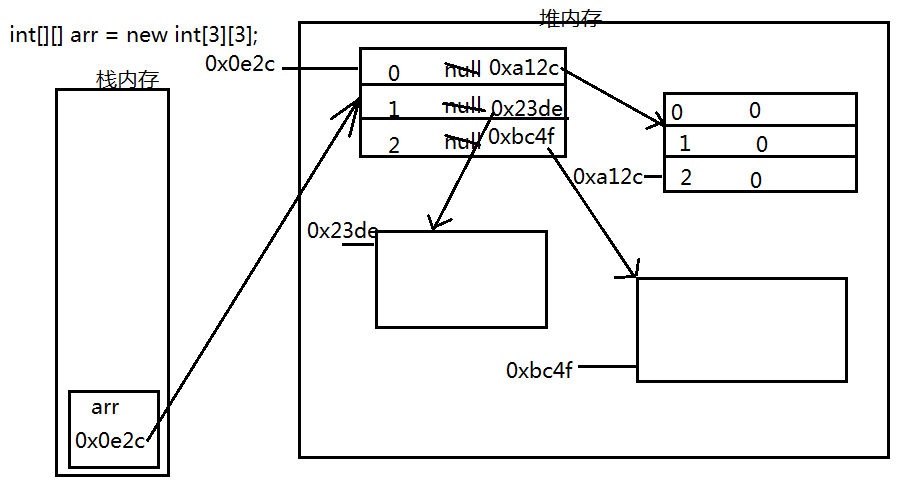
arr[2] = new int[4];

数据类型[][] 数组名 = {{一维数组1},{一维数组2},{一维数组3}, ... ,{一维数组n}};

int[][] arr = {{2,4}, {1,2,3,5,7}, {8}, {8,4,3,0}}; --包含了4个一维数组

System.out.println(arr[1][2]); --- 3

System.out.println(arr[1]);---打印对应的一维数组的地址



如果没有给二维数组中对应的一维数组定义大小，那么针对这个一维数组操作会出现空指针异常---NullPointerException

面试题：对于int[] x[], y均以初始化，下列各项中正确的是：---BC

1. x[0][0] = y;
2. x[0] = y;
3. x[0][0] = y[0];
4. x[0] = y[0];
5. x = y;

练习：杨辉三角

1

1. 1
2. 2 1
3. 3 3 1
4. 4 6 4 1
5. 5 10 10 5 1

arr[i][j] = arr[i - 1][j] + arr[i - 1][j - 1];

输入行数，然后输出对应的前n行

## 方法

将一段代码/逻辑提取出来进行包装，这种包装形式就称之为方法/函数。---为了方便重复利用这段代码

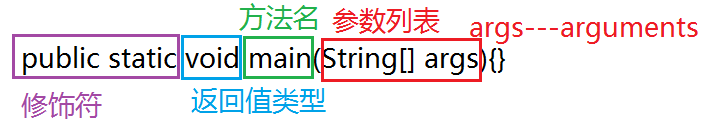
#### 定义格式

修饰符 返回值类型 方法名(参数列表){

代码块;

return 返回值;

}



// 求2个整数的和

// 明确方法的返回值类型---求2个整数的和---结果一定是一个int类型---返回值类型是int

// 明确方法是否需要未知量---这2个整数对于当前方法而言是2个未知量，需要以参数的形式体现

// 形式参数---形参

public static int add(int i, int j){

int sum = i + j;

// 将结果返回给用方法的人

return sum;

}

// 打印m行n列的\*组成的矩形

// 明确返回值类型 --- 打印矩形 --- 没有结果 --- void

// 明确是否有未知量---m行n列，m和n是两个未知量，需要以参数形式来体现

public static void printStar(int m, int n){

for(int i = 0; i < m; i++){

for(int j = 0; j < n; j++){

System.out.print("\*");

}

System.out.println();

}

}

练习：

1. 哥德巴赫猜想：任意一个大于等于6的偶数，都能分解成两个质数之和

18 = 7 + 11

18 = 5 + 13

14 = 3 + 11

14 = 7 + 7

输入一个大于等于6的偶数，将其进行分解

2. 亲密数：如果A的所有因子（包含1而不包含本身）之和等于B，并且B的所有因子（包含1而不包含本身）之和等于A，A和B是一对亲密数---打印3000以内的所有的亲密数

12 : 1 2 3 4 6 12

1+2+3+4+6=16

16：1 2 4 8 16

1+2+4+8=15

### 方法的递归

public static void main(String[] args){

// StackOverFlowError ---栈溢出错误

// 方法在栈中执行

// 当方法执行的时候需要在栈中占用一部分空间，当递归的次数比较多的时候，会耗费大量的内存。如果栈内存不够用了就会出现栈溢出错误

// 循环的效率要高于递归---所以能使用循环的而时候减少使用递归

System.out.println(sum(10500));

}

public static int sum(int n){

if(n == 1)

return 1;

// 在方法中调用了自己本身 --- 方法的递归

return n + sum(n - 1);

}

练习：求n的阶乘 ---> 5! = 5 \* 4 \* 3 \* 2 \* 1

智能开发工具

JCreator--现在不用了

Eclipse --- 日食 --- 绿色的、开源的、免费的、基于插件 -- 要求必须安装了JDK

Luna --- Mars(jdk1.7) --- Neon.2(jdk1.8) Neon.3 ---- Oxygen(jdk1.8)

MyEclipse --- 适合于Web开发 --- 收费

IntelliJ ---IDEA ---适合于Web，Maven管理和代码重构性比MyEclipse要好很多 -- 收费

Alt+/ 提示键

Ctrl+D 删除一行

Ctrl+Shift+o 自动导包

方法的传值和方法的重载

复习：

1. 数组：排序---冒泡、选择排序---时间复杂度O(n2)，空间复杂度o(1);Arrays.sort()---时间复杂度O(nlogn);查找---如果元素有序，可以使用二分查找---时间复杂度O(logn);反转---首尾交换；复制---本质上产生了一个新的数组---延伸出了数组的扩容机制

2. 二维数组：元素类型是一维数组。---二维数组的每一个位置上存放的是对应的一维数组的地址。---如果只定义了二维数组的大小，而没有定义包含的一维数组的大小，那么这时候每一个位置上存放的是一个null---如果针对null进行操作会出现NullPointerException

3. 方法/函数：为了使某段代码能够重复利用，所以将这段代码提取成方法。void--没有返回值；形参和实参；返回值---有了具体的返回执行类型就一定确保方法有返回值

public static boolean m(int number){ // 返回值类型是boolean类型，也就意味着方法必须有一个boolean类型的返回值

if(number < 4){ // 由于在返回之前需要进行判断，这个时候就意味着return语句不一定执行

return true;

}

return false;

}

public static boolean m2(int number){

for(int i = 0; i < 5; i++){ // 对于 JVM而言，i只是一个变量，具体值是未知的，所以无法判断变量i与5的大小是否一定成立，因此JVM认为return语句不一定会执行

// 凡是出现了变量参与的判断，jvm在编译的时候都不确保后续的代码是否一定执行

return number \* 2 > 3;

}

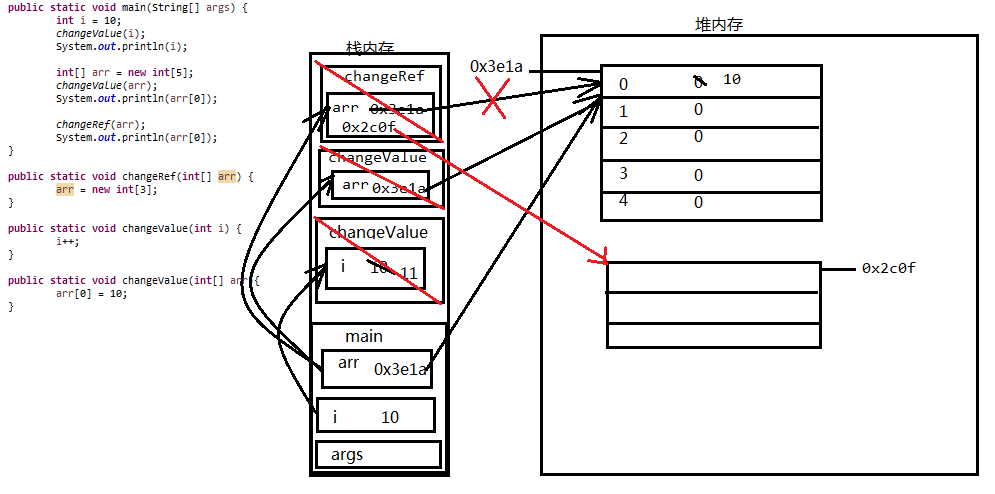
}

方法的递归---形式是方法调用自己本身。---方法是在栈中执行

### 方法的重载

在同一个类中，方法名一致而参数列表不同---方法的重载。---重点关注的就是参数列表

### 方法的传值

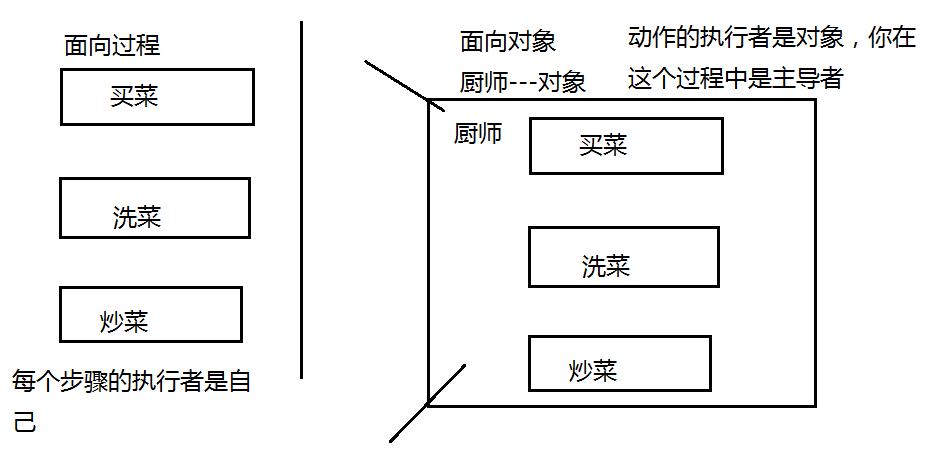


在方法传参的时候，基本类型传递的是实际的数据，所以新方法中数据发生改变不影响原方法中的数据；数组传递的是地址，所以在新方法中如果不改变地址，则会影响原来的方法中的数组；如果在新方法中改变了地址，则不会影响原来方法中的数组

## 6.面向对象

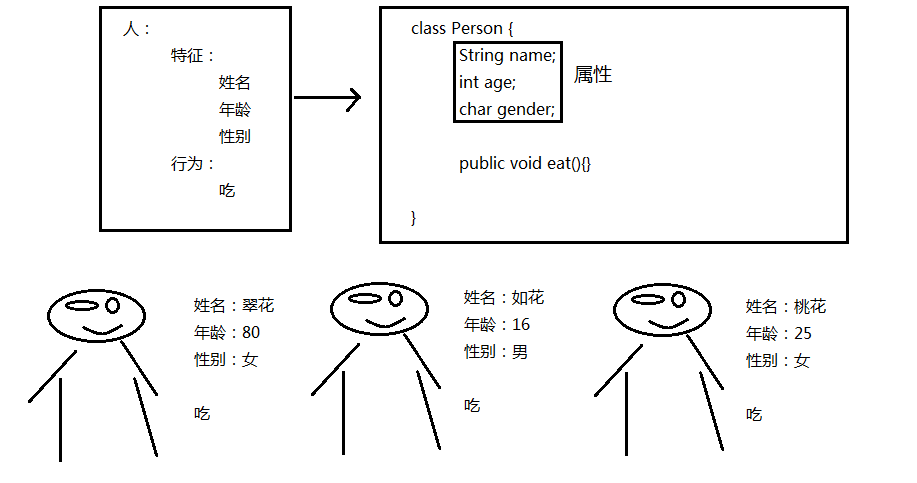
面向过程：注重的过程，强调的动作

面向对象：所关注的是对象。只要有了对应的对象，就能拥有这个对象所具有的一切功能

面向对象是基于面向过程的

面向对象一定优于面向过程吗？---不一定---当处理一些相对简单的事务的时候，建议使用面向过程；如果处理的事务比较复杂，建议使用面向对象。

### 类和对象的关系



根据一类对象的共有的特征和行为进行抽取，将这类事物用类来表示，将特征抽取成了属性，将行为抽取成方法---类是对象的抽取/概括

利用new关键字来创建对应的类的对象/实例---对象是类的具体化

复习：

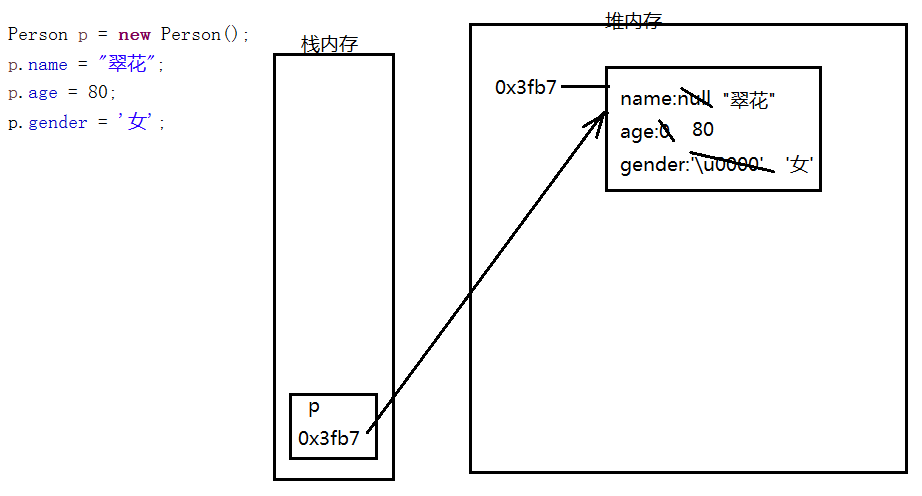
1. 方法的重载：同一个类中，方法名相同，参数列表不同---对应位置上的数据类型不同--- 方法签名：方法名+参数列表 --- add(int, int)

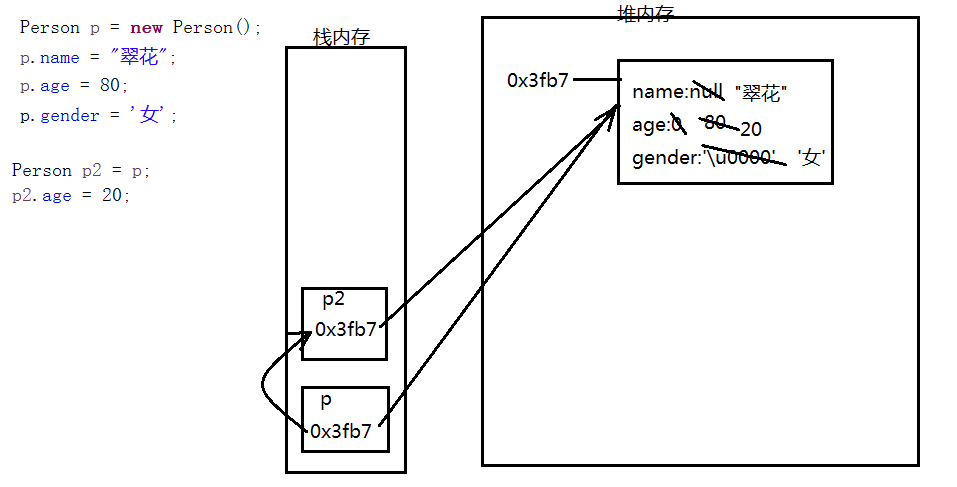
2. 方法的传参：对于基本类型，传递的是实际值；对于引用类型，传递的是地址---传递后在地址不变的情况下，影响原来的对象；如果地址改变，则不会影响原来的对象

3. 面向对象和面向过程：如果需要自己动手，就是面向过程；如果找别的人来动手，就是面向对象。面向过程注重的动作；面向对象强调的是对象。面向对象是基于面向过程。---如果处理简单的事务，使用面向过程；如果处理复杂的事务，使用面向对象。

4. 类和对象的关系：类是对象的抽取；对象是类的具体化

### 对象在内存的存储





### 成员变量（member）和局部变量(local variable)

1. 定义位置：成员变量（属性）是指定义在类内方法外的变量；定义在了方法或者语句内的变量就是局部变量。

2. 作用范围：成员变量是作用在整个类内；局部变量只能作用于对应的方法或者语句中。

3. 内存位置：成员变量是存储在堆内存中，并且自动的赋予了一个默认值；局部变量存在栈内存中，不会自动给默认值。

4. 生命周期：成员变量随着对象的创建而存储到堆内存中，随着对象的回收而被销毁；局部变量在方法或者语句执行完成之后立即回收。

### 构造方法

在类中与类同名而没有返回值类型的方法---构造方法---创建对象

在类中，如果没有手动添加构造方法，那么Java会在编译的时候添加一个默认的无参构造，如果手动添加了构造方法，Java就不再添加默认的构造方法了。

构造方法可以重载。

### this关键字

注意：在Java中，所有的非静态方法和非静态属性都是通过对象来调用的。

this代表当前在活动的对象---可以认为是一个虚拟对象---用于在类内调用本类中的非静态方法和非静态属性的。

this语句表示在本类中调用本类其他形式的构造函数

### 构造代码块

在类内方法外用{}括起来的一段代码---构造代码块/初始化代码块---在创建对象的时候先于构造方法执行

### 局部代码块

在方法或者语句内部用{}括起来的一段代码---局部代码块---限制变量的使用范围和生命周期，以提高内存的利用率

## 面向对象的特征

封装、继承、多态（抽象）

### 封装

体现形式：方法，属性的私有化---将类中的属性用private修饰，限制属性的使用范围，对外提供操作和获取这个的属性的公共方法，在这些方法中可以根据场景来限制属性的值，使之更符合场景要求；内部类

优点：提高代码的复用性，保证数据的合法性

### 继承

根据一些类中的共有特征和共有的行为进行抽取，这些行为和特征抽取到一个新的类中，让原来的类通过extends关键字与新的类产生关系，这种关系称之为继承。新抽取出来的类是父类（超类/基类），原来的类就是子类（派生类）。

子类通过继承父类，就可以使用父类中的一部分方法和属性

在Java中，类和类之间支持的是单继承---一个子类只能有一个父类，但是一个父类可以有多个子类。支持多层继承。---继承结构树

单继承一定比多继承好吗？---不一定

多继承在代码的复用性上要优于单继承

class A {

public int m(){return 0;}

}

class B {

public int m(){return 3;}

}

// 如果允许多继承

class C extends A, B {}

C c = new C();

int i = c.m();

对于多继承而言，如果多个父类中存在了方法签名一致的方法，子类在调用的时候会产生混乱---单继承能够很好的避免方法调用的混乱

作用：提高了代码的复用性，避免方法调用的混乱

### super关键字

在子类中表示父类对象的引用---代替了父类对象---虚拟对象---在子类调用父类中的方法或者属性。

在Java中，要创建子类对象必然先创建一个父类对象---父类对象是通过super语句来创建的。---如果子类的构造函数中没有手动提供super语句，那么默认调用父类的无参构造---super语句表示调用父类中对应形式的构造函数---如果父类只提供了含参构造，那么子类的构造函数中就得手动提供对应形式的super语句---super语句必须放在子类构造函数的第一行

### 权限修饰符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 本类中 | 子类中 | 同包类中 | 其他类中 |
| public | 可以 | 可以 | 可以 | 可以 |
| protected | 可以 | 可以 | 可以 | 不可以 |
| 默认 | 可以 | 同包子类可以 | 可以 | 不可以 |
| private | 可以 | 不可以 | 不可以 | 不可以 |

范围：public > protected > 默认 > private

### 方法的重写

在父子类中存在了方法签名完全一致的方法---方法的重写/覆盖

#### 遵循原则

1. 方法签名相同

2. 如果父类方法的返回值类型是基本类型/void，那么子类重写的方法的返回值类型必须一致

class A{

public double m(){}

}

class B extends A {

public double m(){}

}

3. 子类重写的方法的权限修饰符的范围必须要大于等于父类权限修饰符的范围

class A{

protected double m(){}

}

class B extends A {

public double m(){}

}

4. 如果父类方法的返回值类型是引用类型，那么子类重写的方法的返回值类型和父类方法返回值类型一致，或者子类方法的返回值类型是父类方法返回值类型的子类。

class A{}

class B1 extends A{}

class B2 extends A{}

class C {

public A m(){}

}

class D extends C {

public B2 m(){}

}

### 多态

编译时多态：方法的重载

add(2,3); --- add(int , int)

add(1,4,6);---add(int , int , int)

运行时多态：方法的重写，向上造型---运行时多态是基于继承的

注意：对于使用向上造型创建的对象而言，能干什么看的是父类；怎么干看的是子类。---对于使用向上造型创建的对象，不能去调用子类中独有的方法

#### 重写的理解

1. 子类重写的方法的权限修饰符的范围要大于等于父类方法的权限修饰符的范围

class A {

public void m(){}

}

class B extends A {

protected void m(){} // 错的

}

A a = new B(); // a对象是A类来声明的，A类就会告诉a对象有一个m方法可以使用，而且这个m方法可以在任意的地方使用；a对象实际是用B类创建的，m方法的执行就要看B类，B类告诉a对象m方法不能在其他的类中使用。---声明和使用过程会产生矛盾

2. 子类重写的方法的返回值类型是父类方法返回值类型的子类或者是其本身。

class A{}

class B extends A {

public void m(){}

}

class C {

public B mc(){return new B();}

}

class D extends C {

public A mc(){} // 错的

}

C c = new D(); // 创建了一个c对象，c对象是用C类声明的，C类就会告诉C对象有一个mc方法可以使用，而且这个mc方法的返回值类型是B;实际创建类型是D，在执行的时候，D类告诉c对象返回值类型应该是A

B b = c.mc(); // 定义了一个B类型的变量来接住返回的结果，B类告诉b对象有一个m方法可以使用---这儿就用子类接住了一个父类对象 --- B b = new A();--因为父类不一定含有子类中的方法，所以用子类对象接住父类对象是不可行的

b.m();

注意：基本类型的大小是指的取值范围的大小而不是说基本类型之间有继承关系---基本类型之间是平等的

作用：为了实现代码的灵活调用---解耦

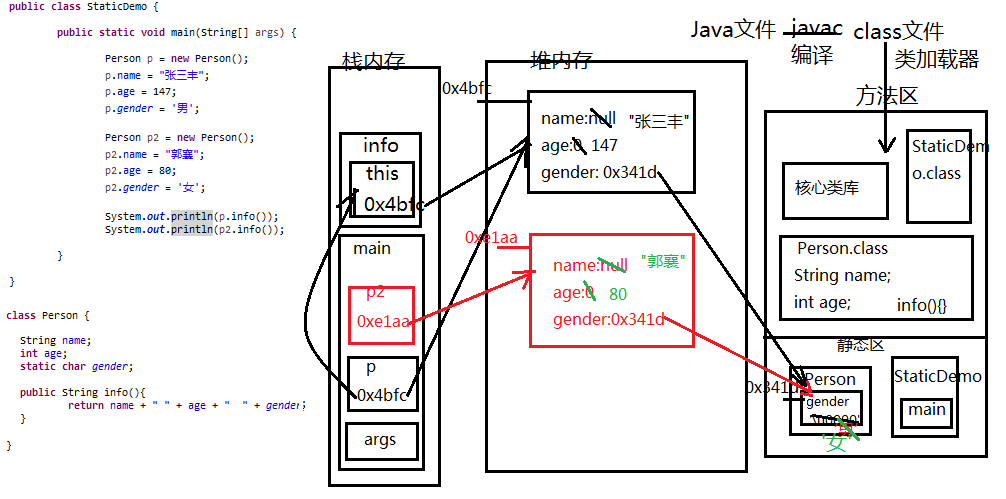
练习：定义一个代表形状的类---Shape,为其提供两个子类---圆形--Circle，矩形---Rectangle---正方形---Square,在形状类中定义获取周长（girth）和面积(area)的方法，在子类中做不同的实现

## static

是一个修饰符---可以修饰变量、方法、内部类、代码块

### 静态变量

static修饰变量--静态变量



复习：

1. 成员变量和局部变量：定义位置、作用范围、内存位置、生命周期

2. 构造方法：与类同名没有返回值类型的方法---在创建对象的时候，需要调用不同形式的构造方法---可以被重载

3. this关键字：代表当前在活动的对象的引用---可以认为是一个虚拟对象，用于在类内调用本类中的方法或者属性；this语句---this(参数)，表示调用本类中对应形式的构造方法---必须放在构造函数的第一行

4. 代码块：构造代码块---在类内方法外用{}来定义---在创建对象的时候，先于构造方法执行；每创建一个对象，都会执行一次；局部代码块---在方法内用{}定义的---限制变量的生命周期，从而提高内存的利用率

5. 面向对象的特征：封装、继承、多态

6. 封装：体现形式---函数（为了能够重复利用某段逻辑或者功能，将其进行提取，这种提取形式就是方法）、属性的私有化（将属性用private修饰，然后提供对外的操作或者读取的方法，可以在方法中对属性值做限定使之更加符合应用场景）、内部类 --- 提高了代码的复用性/降低了代码的冗余度，保证数据的合法性

7. 继承：extends---Java支持的是类与类之间的继承；多继承和单继承的比较；---提高代码的复用性，保证调用的方法不会产生歧义

8. super关键字：在子类中表示父类对象的引用---在调用父类中的方法或者属性的时候，需要通过父类对象来调用，super就表示这个隐含的父类对象---虚拟对象；super语句---定义在子类的构造函数中，表示调用父类对应形式的构造函数---子类的构造函数中都会直接或者间接的创建父类对象---super语句必须定义在子类构造函数的首行

class A {}

class B extends A {

public B(){ // 如果不手动提供super，会有一个默认的无参super语句

super();

}

public B(int i){

this();

}

}

9. 权限修饰符：public > protected > 默认 > private --- 明确子类中使用和子类对象使用不是一会事

10. 多态：编译时多态---方法的重载，运行时多态---向上造型，方法重写---基于继承；行为多态---方法的重载和重写，对象多态---向上造型---父类声明，子类创建---父类定义行为的集合，子类实现行为的细节

11. 方法的重写：在父子类中方法签名一致的非静态方法---方法签名，返回值类型（基本类型/void）,权限修饰符，返回值类型（引用）

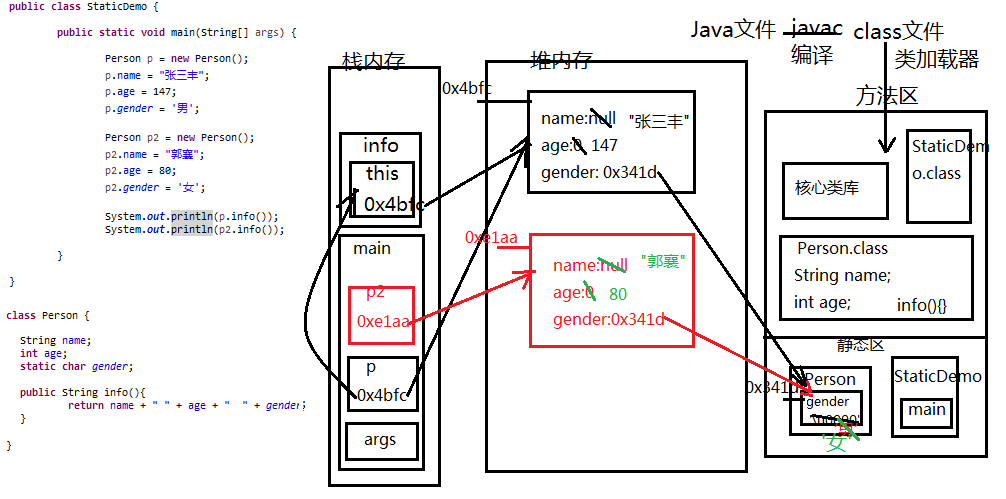
## static

static修饰的变量、方法、内部类、代码块

### 静态变量

用static修饰的变量称之为静态变量/类变量。在类加载的时候加载到了方法区中的静态区，并在静态区中自动的赋予了默认值。静态变量是先于对象而存在的，所以它可以不通过对象而是通过类名来调用，也可以通过对象来调用，但是习惯上建议使用类名调用。这个类的每一个对象存储的是静态变量在静态区中的地址，所以这些对象是共享这个静态变量的。也因此，只要有一个对象对这个静态变量的值做了改变，其他对象获取的值也会改变。---如果一些对象共享了某个特征，可以设置为静态的；如果每一个对象的某个属性各不一样，那么使用非静态的。

类加载到了方法区中的静态常量池（存储类信息），类只加载一次，不再移除。除了核心类库以外，其他的类在第一次使用的时候才加载



静态变量可以定义在方法中吗？--不可以---静态变量随着类的加载而加载并初始化；方法在被调用的时候才执行

静态变量可以在构造代码块中定义吗？---不可以

注意：静态变量只能定义在类内，不能定义到方法或者代码块中

System.out

System.in

### 静态方法

static修饰的方法就叫静态方法/类方法。随着类的加载而加载到方法区中的静态区，只存储在了静态区没有执行。在方法被调用的时候，到栈内存中执行。静态方法先于对象出现的，意味着可以通过类名来调用，也可以通过对象来调用的

main

System.arraycopy()

Arrays.sort/toString/copyOf

Math.pow/sprt

System.out.println();---out是一个静态对象，println方法不是静态方法

静态方法可以重载吗？---可以

静态方法可以被继承吗？---可以

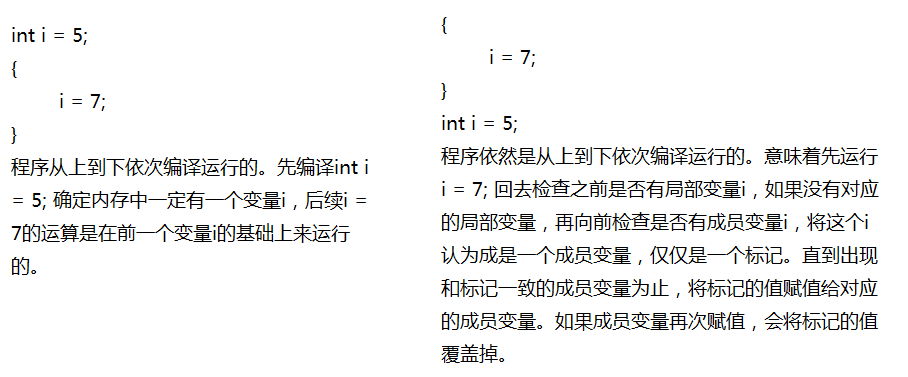
静态方法可以被重写吗？---不能

在父子类中可以存在方法签名一致的静态方法吗？---可以---如果父子类中存在了方法签名一致的方法，要么都是静态方法，要么都是非静态方法

### 静态代码块

用static修饰的代码块---只在类加载的时候执行一次---在类第一次真正使用（创建对象或者调用方法）的时候执行一次

执行顺序：先加载父类---父类静态，然后加载子类---子类静态，其次创建父类对象---父类的构造，最后创建子类---子类的构造



## final

修饰数据、方法、类

final修饰的数据---常量---值不可改变---对于基本类型而言指的是实际值不可变，对于引用类型而言，指的是地址不可改变，而属性值可以改变

public class A {

final int i;

{

this.i = 10;

}

public A(){}

public A(int i){

this.i = i; // 二次赋值

}

}

A a = new A(5);

final修饰的方法---最终方法---不能够被重写---可以被重载，能被继承

考虑：在子类继承父类的时候，父类的什么东西对子类不可见？---被隐藏

private、代码块、构造函数

注意：父类的所有的数据和方法子类都可以继承

final修饰的类---最终类---不能被继承---System

扩展：可以把一个类的构造方法来私有化,这个类可以被继承吗？---可以

总结：

1. 在不考虑内部类的情况下，可以修饰类的修饰符：public、默认、abstract、final

2. 可以修饰方法的修饰符：权限修饰符、static、final、abstract、strictfp、synchronized、native

## abstract

如果子类对父类的某一个或者多个方法做了重写，父类中对应的方法没有定义方法体的必要，那么就可以省略这个方法的方法体，用abstract关键字把这个方法定义为抽象方法---抽象方法所在的类一定是抽象类，但是抽象类中不一定有抽象方法

抽象类不允许实例化---不能用Java代码实例化，底层是用C语言来构建的这个对象

子类继承抽象类之后必须重写其中的抽象方法，除非这个子类本身也是抽象类

抽象方法可以进行重载吗？---可以

能够用static/final/private修饰抽象方法吗？--- 不能

抽象方法可以是默认的权限修饰符吗？---可以，要求父子类同包

抽象方法所在的类一定是抽象类，一定不是最终类

### 接口

用interface来定义接口，用implements关键字让类和接口之间差生联系，这种联系称之为实现。

接口不能实例化，也不能定义构造方法

Java中支持单继承，多实现---类和类之间是单继承，类与接口之间是多实现的---一个类可以实现多个接口

interface A {

public abstract int m();

}

interface B {

public abstract String m();

}

class C implements A, B {} // 多实现不能提高代码的复用性，反而导致实现的时候出现方法的混乱---接口不是为了提高复用性，而是为了统一方法结构，方便同一的调用

接口可以继承接口，而且是多继承

在JDK1.7及其以前，接口中只能定义抽象方法

接口中的方法默认用public abstract修饰

接口中的属性默认用public static final修饰

interface A {

void m();

}

class B implements A {

void m(){} // 接口中的方法默认public

}

复习：

1. static：可以修饰变量、方法、内部类、代码块。

静态变量---随着类的加载而加载到方法区中的静态区，并赋予了默认值。静态变量先于对象存在。所有的对象存储的静态变量所对应的地址

可以用this调用静态变量吗？---可以

静态方法---在类加载的时候加载到了方法区中的静态区，只是存储在了静态区没有执行，在方法被调用的时候到栈内存中执行。静态方法先于对象存在。静态可以重载，不可以被重写，但是在父子类中可以存在方法签名完全一致的静态方法(隐藏)。---如果父子类中存在了方法签名完全一致的方法，要么都是静态，要么都是非静态。

可以在静态方法中使用this吗？---不能

可以在静态方法中直接调用本类的非静态方法吗？---不可以

public void m1(){}

public static void m2(){

m1(); // 错--- 在Java中，所有的非静态方法和非静态属性都需要通过对象来调用

}

静态代码块---在类第一次被真正使用的时候---创建对象，调用方法

执行顺序：父类静态-子类静态-父类构造-子类构造

2. final：修饰数据、方法、类

常量---定义好之后值不可改变---基本类型是实际值不可变，引用类型是地址不可变，其中的属性值或者元素值可以改变的。---如果定义了一个成员常量，需要在对象创建完成之前给值；如果是静态常量，需要在类加载完成之前给值。

最终方法---不能被重写---可以被重载，可以被继承

子类可以继承父类的所有的属性和方法，有一部分父类的功能对子类不可见

子类通过继承可以使用父类中一部分的方法和属性

最终类---不能被继承

3. abstract：修饰类、方法

抽象类---不能被实例化，但是有构造方法，抽象类中可以定义一切属性和方法；子类在继承了抽象类之后必须重写抽象类中的所有抽象方法，除非这个子类也是抽象类。抽象类一定不是最终类

抽象方法---可以被重载，必然被重写---static/final/private不能修饰抽象方法。如果一个抽象方法是使用了默认权限修饰符来修饰的，那么这个时候要求父子类同包。

4. interface：用implements关键字与类产生实现关系---多实现。接口可以继承多个接口---类实现了接口之后必须重写接口中的所有的抽象方法。接口不能定义构造方法。---接口中所有的方法都是抽象方法，默认用public abstract修饰。接口中定义的属性默认是用public static final修饰

## 内部类

### 方法内部类

定义在方法中的类

Outer1$1$Inner1.class---外部类$编号$内部类---习惯上，如果class文件名中出现了$,一般会认为这是一个内部类

### 成员内部类

定义在类内方法外的类

Outer2.Inner2 oi2 = new Outer2().new Inner2();

### 静态内部类

用static修饰的内部类

Outer3.Inner3 oi3 = new Outer3.Inner3();

### 匿名内部类

没有名字的内部类

任何一个可以被继承的类以及任何一个接口都可以存在匿名内部类的形式---匿名内部类本质上是继承了对应的类或者是实现了对应的接口

## 包

用package声明包，用import导包

\* 表示一个通配符---导入当前包下的所有的类，但是不包括子包下的类---不推荐使用

java---Java小组提供的原生包

javax---Java的扩展包

org---第三方厂商提供的包

java.lang---核心包---包含了Java程序要运行的基本类---在JVM启动的时候就自动加载到内存中了---所以这个包中的所有的类在使用的时候不需要手动导包

考虑：什么情况下可以不用手动导包？---java.lang， 同包类

java.util---工具包---提供了很多便捷的操作数据的类

java.math---和数学运算有关的包

java.text---和格式化有关的类

java.io---和数据传输有关的包

java.sql---对数据库进行操作的包

java.net---和网络通信有关的包

java.security---和数据安全有关的包

java.nio---支持高并发的包

java.applet---GUI操作相关的包---现在几乎不用

javax.swing---GUI的扩展包

## 垃圾分代回收机制

回收针对的是堆内存。

将堆内存划分为了2块：新生代和老生代。新生代有划分为伊甸园区和幸存区。对象刚创建的时候是放在新生代的伊甸园区，在伊甸园区经过一次回收之后如果这个对象依然存在，则会挪到幸存区。如果在幸存区经过多次回收，会挪到老生代。老生代的扫描频率是远低于新生代。老生代的对象一旦回收，往往会导致程序的卡顿甚至于崩溃。

发生在新生代的回收---初代回收(minor gc --- garbage collector)

发生在老生代的回收---完全回收(full gc)

如果一个对象在新生代存放不开，会试图直接放入老生代---OutOfMemoryError

### 扩展：类加载机制

加载（将class文件转换为字节码放入内存）---校验（检查这段字节码是否合法）---准备（为所有的类变量分配空间并默认初始化）---解析（引用替换过程）---初始化（类变量赋值，静态代码块执行）

类加载器---ClassLoader

启动/引导类加载器 --- Bootstrap ClassLoader---加载Java的原生类

扩展类加载器 --- Extension ClassLoader --- 加载扩展类

应用类加载器 --- Application ClassLoader --- 加载自定义类

在Java中，加载类的时候双亲委派机制

API --- Application Programming Interfaces --- 应用程序接口

Object String Pattern 数学类 包装类 日期类

异常 集合 泛型 映射

File IO 线程 套接字 反射 注解

JDK1.8的特性，（有时间就扩展）jvm的运行参数

复习：

1. 内部类：方法/局部内部类、成员内部类、静态内部类---匿名内部类。--- 除了静态内部类以外，其余的内部类中都不允许定义静态变量和静态方法；除了静态内部类以外，其余的内部类都允许使用当前外部类的成员属性和成员方法；方法内部类如果使用当前方法中的数据，这个数据需要是一个常量。---类比方法先加载

2. 包：package声明，import导包。---java.lang，java.util

3. 垃圾分代回收机制：针对堆内存---新生代（伊甸园区，幸存区），老生代---初代回收，完全回收

4. 类加载机制：加载（先去加载外部类，然后再加载内部类）-校验-准备(为类变量分配空间并初始化)-解析（引用替换---常量和符号进行直接替换）-初始化;类加载器---启动、扩展、应用---双亲委派机制

## Object

Object是Java中的顶级父类---也是Java中唯一一个没有父类的类

### 重要方法

clone()---克隆出一个和原有对象的属性值一样的新对象---如果一个类产生的对象想要被克隆，那么这个类必须一个接口Cloneable---这个接口仅仅标志这个类的对象可以被克隆

finalize()---这个方法通知GC回收垃圾---但是GC不一定运行---System.gc()

getClass()---获取对象的实际类型而不是声明类型

toString()---将对象转化为字符串，默认输出的是地址---当打印一个对象的时候实际上是默认调用了这个对象的toString方法

equals(Object o)---比较两个对象是否相等---默认比较的两个对象的地址是否一致，但是实际开发过程中，往往会手动重写equals方法---地址、参数、类型、属性值---要求掌握两到三个属性的判断

练习：重写Students类中的equals方法

class Student {

private String name;

private String no;

}

## String

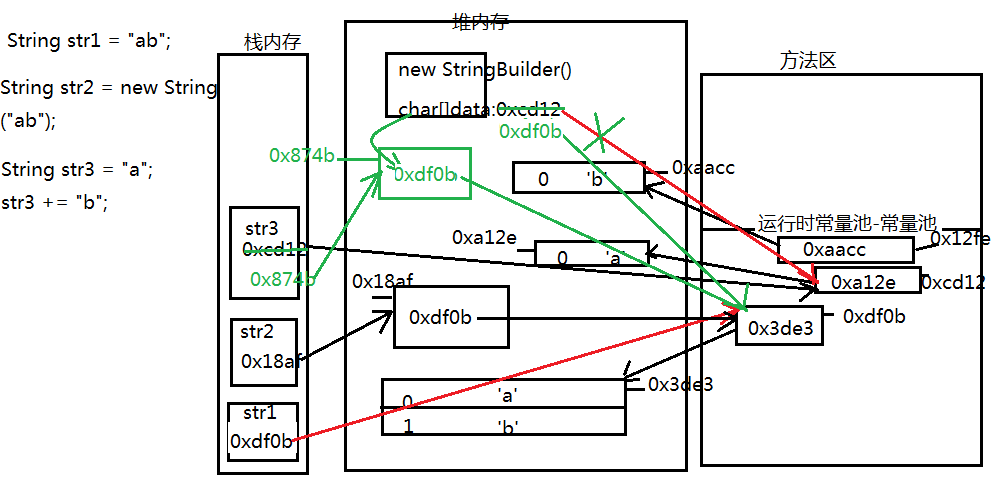
代表字符串的类，最终类---字符串是一个常量，定义好之后不能改变；因为字符串是常量，所以可以被共享。

字符串在Java底层是以数组形式存储的。

String str = “abc” -> final char[] data = {‘a’, ‘b’, ‘c’};

String str = “a”; str += “b”;

-> str = new StringBuilder(str).append(“b”).toString(); ->str = new String(“ab”);



String str = “abc”; --- 1

String str = new String(“abc”); --- 2

String str = “a”; str += “b”;---5

// 已经定义好了一个字符串数组arr，包含了100个字符串

// 如果需要把这100个字符串拼接成1个字符串

// 使用+会额外产生多少个对象？

String result = “”; // 1

for(String str : arr){

result += str; // 每拼接一次，就要额外产生3个对象。 100 --- 300

}

// 整个过程中，要额外产生301个对象

// 如果使用StringBuilder直接拼接结果，额外产生多少个对象？

StringBuilder sb = new StringBuilder(); // 1

for(String str : arr){

sb.append(str); // append一次，产生1个对象 100 --- 100

}

String str = sb.toString(); // 1

// 整个过程中，要额外产生102个对象

如果拼接多个字符串的话，建议使用StringBuilder

StringBuilder是一个线程不安全的类，StringBuffer是一个线程安全的类

练习：

1. 输入字符串，输出这个字符串中字母、数字以及其他符号的个数

2. 输入字符串，输出这个字符串中的所有的数字之和 --- dn323nnds92ndkn -> 3+2+3+9+2=19

3. 输入字符串，提取这个字符串中的数字并进行升序排序 --- dsikn38dsgbs9244b10 -> 38924410 -> 01234489

4. 输入一个字符串，输出每个字符出现的次数 --- dsosnasdb -> d:2 s:3 o:1 n:1 a:1 b:1

注意：String类中提供了一系列的操作字符串但是不改变原串的方法，都是产生一个新的字符串

## String

提供了一系列的操作而不改变的原串。

西欧码表：1个字符1个字节

GBK:1个字符2个字节

UTF-8：1个字符3个字节

UTF-16:1个字符2个字节

练习：输入一个字符串和一个数字，这个数字表示的是字节个数，按照指定的字节个数来截取字符串---不允许出现半个汉字的情况

达内BigData 7 -> 达内Big --- 3 -> 达

练习：输入一个字符串和子字符串，打印这个子字符串在字符串中出现的所有的位置

class Test {

public static void main(String[] args){

String str = “abc”;

change(str);

System.out.println(str); // abc

}

public static String change(String str){

str = “def”;

return str;

}

}

## REGEX --- Pattern

regular expression---正则表达式---本身表示规则，用正则表达式来指定规则，然后利用这个规则对其他事物做判断、匹配、筛选等

[\\d](file:///\\d) 表示数字

[\\.](file:///\\.) 表示 .

+ >= 1 ? <= 1 \* >= 0

练习：输入字符串，判断这个字符串是否是一个小数

3.13 3.00 .5---false 00.21---false

练习：

1. 匹配邮箱

@qq.com @126.com @sina.com.cn @tedu.cn

[字母/数字@数字/字母.com](mailto:字母/数字@数字/字母.com)\.cn\.com.cn

2. 匹配密码：8-12位 大写字母/小写字母/数字/空格中至少出现2种

练习：

1. 替换叠字 --- 我我我我我我爱爱爱爱爱爱爱爱学学学学学学学学学学学习习习习习习习习习习习习习习习 -》 我爱学习 (.)\\1+

2. 输入一个字符串，求字符串的平均碎片长度

扩展：疯狂值

5

20 14 72 31 10

疯狂值：相邻两个数字的差的绝对值之和

|20-14| +|14-72| + |72-31| + |31 - 10| = 126 ---Math.abs(int i)

求最大疯狂值

复习：

1. String：操作字符串但是不改变原字符串

2. Pattern：指定规则

str.replaceAll(“(.)\\1+”, “$1”);

## 包装类

针对每一种基本类型都提供对应的类类型，然后利用这些类产生的对象来便捷的操作的数据。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| byte | short | int | long | float | double | char | boolean | void |
| Byte | Short | Integer | Long | Float | Double | Character | Boolean | Void |

注意：Void本身是一个最终类，不能创建对象

将基本类型转化为对应的引用类型的对象---封箱

将基本类型直接赋值给引用类型---自动封箱，底层实际是调用了对应类身上的valueOf方法

将引用类型直接赋值给基本类型---自动拆箱，底层实际上调用了对应对象身上的\*\*\*Value方法

自动封箱/拆箱是JDK1.5的特性之一

考虑：如何将字符串转化为对应的字符？

“c” -> ‘c’ char ch = “c”.charAt(0); char ch = “c”.toCharArray()[0];

注意：八种包装类型对应的对象的哈希码是固定不变的

## 数学类

Math---最终类，只针对基本类型提供了基本的数学运算，本身构造方法私有了，提供了大量的静态方法，本身是作为一个完全的工具类来使用的。

BigDecimal---用于精确运算的类，需要将参数以字符串形式传递

## 日期类

Date---表示日期的对象，需要掌握将字符串转化为日期的方法---SimpleDateFormat 中的parse方法，在转换的时候需要指定的转化格式

Calendar --- 仅作了解

## JDK8的部分内容

### 接口中的默认方法

从JDK1.8开始，允许在接口中定义的普通的实体方法，但是必须用default来修饰。

也允许在接口中定义静态方法

### Lambda表达式

(参数) -> {代码块};

用于去重写接口中的抽象方法。

注意：要求能够使用Lambda表达式的接口中只能有一个抽象方法

如果接口中只有一个抽象方法，这个接口称之为函数式接口

不能用Object对象来接住Lambda表达式

### 时间包

对时间体系重新划分，划分出来一个全新的时间包---java.time

复习：

1. 包装类：基本类型所对应的类形式。自动封箱-valueOf，自动拆箱-\*\*\*Value, ---JDK1.5出现的；包装类的对象的哈希码是固定不变的。

public class Test {

public static void main(String[] args){

Integet i1 = 25;

Integer i2 = 25;

System.out.println(i1 == i2);

int i3 = 25;

System.out.println(i1 == i3); // true

}

}

2. 数学类：Math---最终类，构造函数私有化，提供了很多的静态方法来操作基本类型的数据；BigDecimal---用于精确运算，要求参数必须是字符串

3. 日期类：Date---掌握字符串和日期对象的转换，SimpleDateFormat, parse-将字符串转化为日期， format-将日期转化为字符串

4. JDK8的部分特性：

接口中的默认方法：允许接口存在实体方法，必须用default修饰；或者可以定义静态方法，但是不能用default修饰---接口中方法的权限修饰符默认是public

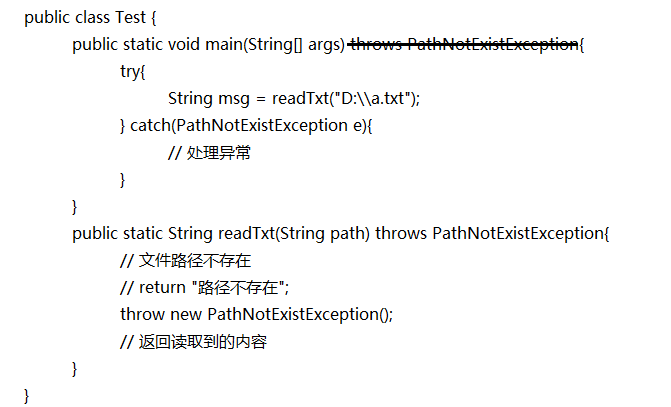
Lambda表达式：重写接口中的抽象方法---要求这个接口只能有一个抽象方法。参数类型可以省略，Java会根据程序的上文自动推导。任意一个Lambda表达式都不能用Object来接住。

方法的传递：方法体只有一句，直接操作参数（参数没有产生变形），直接调用已有类或者是已有对象身上的方法---参数一一对应的

非显式常量：在JDK1.8中，方法内部类使用当前方法中的数据的时候，这个数据可以不用显式声明为常量

时间包：java.time包---LocalDate，只含日期而不含时间；LocalTime,只含时间而不含日期

## 异常



异常是Java中一套用于问题的反馈和处理的机制。

### Throwable

Error --- 错误，出现之后不应该捕获处理 --- VirtualMachineError：[OutOfMemoryError](mk:@MSITStore:C:\Users\Administrator\Desktop\开班软件\JDK_API_1_6_zh_CN.CHM::/java/lang/../../java/lang/OutOfMemoryError.html), [StackOverflowError](mk:@MSITStore:C:\Users\Administrator\Desktop\开班软件\JDK_API_1_6_zh_CN.CHM::/java/lang/../../java/lang/StackOverflowError.html)

Exception---异常，出现之后可以处理。

编译时异常：在编译的时候就出现要求处理的异常---处理的方式：继续向上抛出，try-catch来进行捕获

CloneNotSupportedException---克隆不支持异常

ParseException---转换异常

运行时异常：在编译的时候没有出现而到了运行的时候出现。出现之后可以处理，也可以不处理---RuntimeException

ArithmeticException---算术异常

NullPointerException---空指针异常

IndexOutOfBoundsException---下标越界异常

ClassCastException---类转换异常

PatternSyntaxException---正则格式异常

NumberFormatException---数字格式化异常

自定义的异常如果不是继承了RuntimeException或者是其子类，那么就是编译时异常

#### 异常捕获的方式：

1. 如果多个异常的处理方式各不一样，多个catch分别捕获分别处理

2. 如果多个异常的处理方式都一样，捕获这多个异常的父类

3. 如果多个异常需要进行分组处理，那么同一组的异常之间可以用|隔开 --- JDK1.7及以后

编译时异常必须在出现之后才能捕获，但是运行时可以随意抛出随意捕获

异常对方法的重载没有任何的影响

总结：重载和重写有什么区别？

重载：在同一个类中存在方法名一致而参数列表不同的方法，构成了重载。重载只和方法签名有关，和修饰符、返回值类型以及异常没有关系。重载是一种编译时多态。

重写：在父子类中存在了方法签名完全一致的非静态方法，构成了重写。在方法重写的时候，子类方法的权限修饰符的范围要比父类权限修饰符的范围大，至少保持一致。如果父类方法的返回值是基本类型/void/最终类，那么子类的返回值类型要保持一致。如果父类方法的返回值类型是引用类型，子类方法的返回值类型要是父类方法返回值类型的子类。最后，子类重写的方法的异常范围不能超过父类方法的异常的范围。重写是一种运行时多态。

finally---结合try使用，完成善后的工作---无论是否出现异常，finally都会来执行一次

在开发阶段，如果出现异常，捕获异常并处理，停止项目。

在项目上线阶段，如果出现异常，记录错误日志，跳转错误提示页面。

## 断言

根据已有条件对结果做准确性的判断---断言

在Java中，断言没有默认开启

assert 断言条件 : 提示信息;

### 内部接口

类中可以定义类，类中可以定义接口，接口中可以定义接口，接口中可以定义类

类中定义的接口以及接口中定义的类和接口都是静态的。

## Map<K,V>---映射

两组变量---键，值。每一个键必然对应一个值，键不可重复---一个映射实际上是由多个键值对组成的。

键值对所对应的接口---Entry，它是Map中的内部接口---每一个键值对都是一个Entry对象，那么实际上Map是由多个Entry对象来组成的

Map是集合吗？---不是，是集合框架的一员。

集合框架：和集合、数组、映射相关的接口和类都是集合框架的成员 --- Map/Collection/Collections/Iterator/Comparator/Comparable/Arrays

如何遍历一个映射？

1. 先获取到映射中的所有的键，然后遍历这些键，再根据键来获取值

2. 将映射中所有的键值对放入到一个集合中

HashMap:允许键或者值为null。默认初始容量是16，加载因子是0.75f，默认增加一倍。是一个异步式线程不安全的映射

Hashtable:不允许键或者值为null。默认初始容量是11，加载因子是0.75f。是一个同步式线程安全的映射---仅仅在笔试中出现

ConcurrentHashMap---异步式线程安全的映射

## File

代表文件或者目录的类。

练习：删除目录

写成一个方法：获取指定目录下的子文件和子目录(listFiles()),如果是子文件（isFile），直接删除（delete()）；如果是子目录(isDirectory)，需要再次获取---和当前方法的功能一致---递归

练习：统计工作空间中Java文件和class文件的个数

写成一个方法：获取所有的子文件和子目录；如果是子目录依然递归；如果是子文件，判断是否是一个Java文件还是一个class文件

### 路径

绝对路径：以盘符或者/开头的路径。和当前路径没有任何关系

相对路径：不以盘符或者/开头的路径。以当前路径为基准进行计算 --- .. 表示上一层目录

目录分隔符：在Windows用\分隔目录，但是在Linux下，用的是/分隔

路径分隔符：在Window用;分隔不同的路径，在Linux下用的:来分隔

复习：

1. Map：映射---以键值对形式存在的。键不能重复，值可以重复。---Entry，每一个Entry对象代表一个键值对，一个map对象实际上是由多个Entry对象组成

HashMap：键或者值都可以是null。默认初始容量是16，默认加载因子是0.75f，每次默认增加一倍。异步式线程不安全的映射

Hashtable：键或者值都不允许为null。默认如是容量是11，加载因子是0.75f。同步式线程安全的映射

掌握遍历Map方式---keySet(), entrySet()

2. File：代表文件或者目录的对象。创建好file对象的时候，file对象代表的文件或者目录不一定存在。在后续操作文件的时候，判断文件是否存在或者不判断而是直接创建。

FileFilter --- 文件过滤器---直接操作文件

FilenameFilter --- 文件名过滤器---操作文件名

new File(“D:\\”);

## IO流

用于传输数据的一套API。

数据的来源/目的地：硬盘、内存、硬件设备、网络

IO --- Input Output Stream --- 输入输出流 --- 输入流是指数据从外部流向程序，输出流是指数据从程序流向外部。

读取文件---数据从文件读取到程序中---输入流

向文件中写入一段数据---数据从程序流向文件---输出流

根据数据的传输方向：输入流、输出流

根据数据的传输形式：字符流、字节流

|  |  |  |
| --- | --- | --- |
|  | 输入 | 输出 |
| 字节 | InputStream | OutputStream |
| 字符 | Reader | Writer |

四个基本流都是抽象类

向TXT文件中写入一个字符串 --- 字符流、输出流 FileWriter

### 流的异常处理

1. 将流对象放到try之外定义并且赋值为null，放到try之内初始化

2. 在关流之前需要判断流对象是否初始化成功---判断流对象是否为null

3. 关流之后需要将流对象置为null

4. 需要在写完数据数据之后手动冲刷缓冲区以防关流失败有数据死在缓冲区中

从TXT文件读取数据 --- 字符流、输入流 FileReader

练习：复制TXT文件---先输入流指向源文件，再有输出流指向新文件，然后将读取到的数据写到新文件中

### 缓冲流

在构建BufferedReader对象的时候需要传入一个Reader对象，底层读取数据靠的是传入的Reader对象，BufferedReader提供了缓冲区。---这种方式称之为装饰设计模式

练习：统计Java代码的行数

注意：字符流只能读取纯字符类的文件 .txt .java .html

注意：字符流不能读取Office组件---POI

### 字节流

在读取的时候是以字节形式读取的。

练习：复制一个压缩/视频文件---统计复制时间

System.currenTimeMillis();

### 合并流

SequenceInputStream --- 将多个输入流合并为一个输入流进行操作---只有输入没有输出---合并流的构建需要一个Enumeration（本质上是一个迭代器，Java中最古老的迭代器）对象，需要利用Vector来获取Enumeration对象---elements()

复习：

1. IO流：用于传输数据的API。

根据传输方向：输入流和输出力。

根据传输形式：字符流和字节流。

---InputStream OutputStream Reader Writer---都是抽象类。

数据的来源：硬盘、输入设备、内存、网络

文件字符流：操作字符类文件。字符输出流在输出的时候有缓冲区存在的。字符输入流没有缓冲区。

writer.close(); writer = null;

缓冲流：BufferdReader BufferedWriter 在创建对象的时候都需要传入一个流对象，实际读取数据靠的传入的流对象，缓冲流起的作用是提供一个缓冲区。---装饰者模式 --- Decorate Pattern

文件字节流：FileOutputStream, FileInputStream，在操作的时候没有缓冲区。

合并流：SequenceInputStream ，用于合并多个字节流的。在构建合并流对象的时候，需要传入Enumeration对象，利用了Vector中的elements()来获取这个Enumeration对象。

### 转换流

提供了字符流和字节流之间的相互转换。

OutputStreamWriter：将字符流转化为字节流。

InputStreamReader：将字节流转化为字符流

### 系统流/标准流

系统流都是字节流

System.in 标准输入流

System.out 标准输出流

System.err 标准错误流

练习：从控制台来读取一行数据

System.in -- InputStreamReader -- BufferedReader

### 打印流

PrintStream/PrintWriter--- 能够便捷的向外输出数据

### 序列化/反序列化流

将对象进行存储的过程就称之为序列化 --- 将对象保存到硬盘上就叫持久化

将对象还原回来的过程就称之为反序列化

注意：

1. 一个对象如果想要被序列化，那么要求这个对象对应的类必须实现一个接口---Serailizable --- 仅仅用于标志实现这个接口的类的对象允许被序列化

2. 被static/transient修饰的属性不能被序列化

3. 需要在类中手动指定一个属性---serialVersionUID---版本号。当一个对象序列化出去的时候会携带这个类的版本号，在反序列化的时候会检查对象携带的版本号和类中定义的版本号是否一致，如果一致则允许被反序列化。如果一个类中没有手动指定版本号，那么Java在编译的时候会根据当前类中的属性和方法自动计算产生一个版本号。也就意味着类中的属性或者方法产生变动，版本号就会改变。---因此需要手动指定，一旦手动指定就不再变化了。---版本号默认用private static final long 修饰限定

4. 在Java中数组和集合中的元素以及大部分的映射不能随着集合或者映射一起序列化

序列化和反序列化的加密机制---盒子---锁---钥匙---生产---了解即可

### Properties

是一个可以持久化的映射。键或者值只能是字符串。

必须要存储到.properties文件中---Properties文件的默认编码是西欧编码，因此不能存储中文---一旦放入中文就会出现乱码或者是转化为对应的Unicode编码

Properties经常作为配置文件使用

## JDK1.5的部分特性

自动封箱拆箱、增强for循环、静态导入、可变参数、枚举、泛型、反射、内省、动态代理、注解

### 静态导入/静态导包

import static 包名.类名.方法名; --- 表示导入指定包中的指定的静态方法

优点：能够一定程度的提高使用效率

缺点：书写不便；降低程序的可阅读性；如果在本类中存在了方法签名一致的静态方法，会导致静态导入无效

### 可变参数

用...定义一个可变参数---可变参数本质上就是一个数组，所以可以用操作数组的方式来操作一个可变参数。---可以传入任意多个参数也可以传入数组。---可以只传入一个参数也可以不传参数---对参数个数不做限定

可变参数只能定义一个，并且必须定义到参数列表的末尾。

### 枚举

取值情况相对固定并且能够一一列举---枚举

用enum定义枚举类。枚举常量之间用 , 隔开---枚举常量必须定义在枚举类的首行。

枚举类中的构造方法默认是私有的只能是私有的---构造方法可以重载

枚举类中可以定义一切的方法和属性，甚至包括抽象方法

在Java中，所有的枚举的顶级父类是java.lang.Enum

switch-case的()需要一个表达式，表达式的值可以是byte/short/char/int, jdk1.7允许使用String， jdk1.5开始允许使用枚举常量

### Junit测试---单元测试

不是Java原生提供的而是第三方厂商提供的现在被广泛应用的测试手段

@Test ---在要测试的方法上添加此注解 --- 在使用的时候要求 “三无”：无参无返回值非静态

### Debug---调试模式

断点调试

F6---执行到下一行

F5---钻入方法

F7---返回上一行

F8---执行到下一个断点

复习：

1. 转换流：转换输出流是将字符转化为字节，转换输入流是将字节转化为字符 ---最底层都是以字节形式来读取或者写出数据，表现形式都是以字符形式来展现---转换流都是字符流

2. 系统流：System.in，System.out，System.err---都是字节流---系统流本质上是静态对象---一旦关闭就无法使用---系统流在使用的时候一般不用关闭

3. 打印流：PrintStream, PrintWriter---允许便捷的打印数据

4. 序列化/反序列化

序列化：将对象的信息进行完整保存 --- 持久化

一个接口---Serializable，两个关键字---static/transient，版本号--- serialVersionUID --- 保证在类产生微小变动的情况下已经序列化的出去的对象能够反序列化回来，集合/Map---元素不能随着集合一起序列化出去

5. 静态导入：import static 包名.类名.方法名 --- 表示导入某个已有类中的指定的静态方法

6. 可变参数：用 ... 来定义，本质上是一个数组。参数的个数不做限定。可变参数只能定义一个而且必须放在参数列表的末尾

7. 枚举：用enum来定义枚举。枚举常量必须放在第一行。枚举中的构造方法默认是私有的。枚举可以定义方法和属性，包括抽象方法。枚举的父类是Enum。从jdk1.5开始，允许在switch-case中使用枚举常量

8. 单元测试：需要单独导入测试库---Junit4，在要测试的方法上加上@Test，要求测试的方法不能有参数，无返回值，是一个非静态方法。@Before---一般用于完成一些初始化的操作，@After---用于善后

## 线程

进程：计算机中执行的任务---服务一般没有界面

线程：进程中的一个小任务 QQ/微信、下载软件、记事本、JVM。

注意：计算机的每一个核在每一个时刻都只能处理一个进程中的一个线程---宏观上是并行，微观上是串行

扩展：Windows中默认是用一个核处理，Linux下是有几个核就用几个核

### 定义线程

1. 继承Thread，重写run方法，将逻辑放到run方法中，开启线程用的是start方法

2. 实现Runnable，重写run方法，需要利用Thread对象来启动这个线程

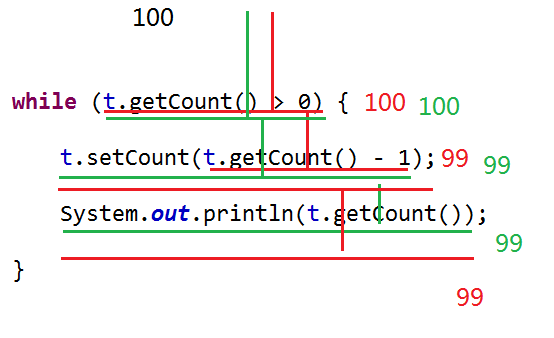
3. 实现Callable<T>，重写call方法 --- 现阶段知道即可

### 多线程的并发安全问题

因为多个线程之间相互抢占而导致出现不符合常理的现象就称之为多线程的并发安全问题

线程的执行是相互抢占资源的。---谁抢到谁就执行。

线程的抢占是发生在线程执行的每一步过程中。



同步代码块---synchronized---意味着一个线程在执行了其他线程不能执行---需要锁对象---锁对象要求被所有的线程都认识：共享资源、类的字节码、this

同步：一段逻辑在一个时间段内只允许一个线程执行

异步：一段逻辑在一个时间段内只允许多个线程执行

同步一定安全---是

不安全一定异步---是

安全不一定同步---宏观角度---微观上安全一定同步

### 死锁

由于锁导致多个线程相互等着致使程序无法继续往下执行

导致的原因：多线程、较多的共享资源、锁的嵌套

避免死锁的方法：减少线程数量、减少共享资源、统一锁对象、避免锁嵌套

### 等待唤醒机制

利用的wait和notify来控制线程的执行顺序---等待唤醒机制

练习：生产消费模型

一个线程表示生产者，另一个线程表示消费者。生产一次然后消费一次

生产者在生产的时候要求本次生产的商品数量和上次剩余的商品数量的和不能超过1000，本次的消费数量不超过提供的商品数量

0 500 500

500 420 80

80 800 880

880 10 870

870 130 1000

1000 0 1000

1000 0 1000

线程在等待的时候是去线程池中等待的。线程池本质上是一个存储线程的队列。---遵循先进先出的原则

如果出现了多个线程，只唤醒一个线程就不够用了，所以用notifyAll唤醒所有的线程

注意：等待唤醒机制必须结合锁对象来使用---等待唤醒和锁是对应的

总结：sleep和wait有什么区别？

sleep需要休眠时间，到点自然醒。会释放执行权，没有释放锁。设计在了Thread类上，是一个静态方法。

wait可以不指定等待时间，也可以指定等待时间。如果没有指定等待时间需要手动唤醒。wait必须结合锁使用。释放执行权，释放锁。设计在了Object上，是一个成员方法。

### 守护线程

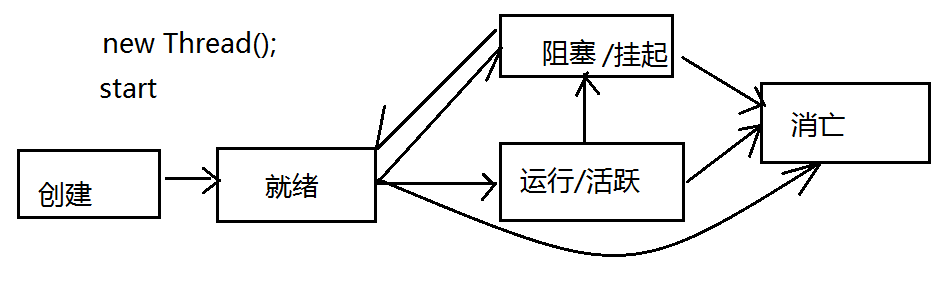
守护其他线程，被守护的线程结束的时候，守护线程无论完成与否都要结束

一个线程要么是守护线程要么是被守护的线程---当最后一个被守护的线程结束才会导致所有的守护线程结束---GC

### 线程的优先级

优先级一共有1-10这十个等级。理论上数字越大优先级越高，也就意味着线程抢到资源的几率就越大。实际中相邻两个优先级的差别很不明显。至少相差5个优先级及以上才会相对明显

### 线程的状态



考虑：引入多线程模型的意义在哪儿？

减少响应时间，提高CPU的利用率

1 0.2 0.8

2 0.36 0.8\*0.8=0.64

3 0.8\*0.8\*0.8

## 单例模式

在全局过程中只有一个实例。

对象定义的时候就初始化叫饿汉式

对象在真正使用的时候才初始化叫懒汉式

饿汉式会增加加载过程，懒汉式能够避免这个问题。

懒汉式会出现线程安全问题，饿汉式不会出现线程安全问题

扩展：掌握单例模式的七种方式

复习：

1. 线程：任务

线程的定义：继承Thread类，实现Runnable接口，实现Callable接口

多线程的并发安全问题：多个线程相互抢占，抢占是发生在了每一步，导致出现了一些不符合常理的数据。

同步锁机制：synchronized---锁对象：要求被所有的线程都认识---共享资源、类的字节码、this

死锁：多个线程的锁不同且形成了嵌套导致程序无法继续往下执行---统一锁对象、减少锁嵌套

等待唤醒机制：利用wait和notify、notifyAll

守护线程：被守护的线程结束，守护线程随之结束

线程的优先级：1-10

线程的状态：就绪、阻塞、运行

2. 单例模式：全局过程中只存在了一个实例。

## 网络编程

基于网络进行数据传输---IO

网络模型：

物理层 数据链路层 网络层--IP 传输层---UDP/TCP 会话层 表示层 应用层 --- http,ftp,POP3,SMTP...

IP地址：IPv4---用了4组数字表示一个地址，每一组数字的大小是0-255---IPv6

端口号：0-65535，0-1024在开发过程中不要使用

域名：每一级域名之间用 . 隔开

DNS解析服务器：将域名和地址进行对应

### UDP

基于流的。不建立连接，不可靠，传输速率相对较快。需要对数据进行封包，每个包不超过64K。---DatagramSocket, DatagramPacket --- 适用于要求传输速度不要求可靠性的场景， 视频聊天

发送端:

1. 创建套接字对象

2. 准备数据包，将数据放到数据包中并且要绑定发送的地址

3. 发送数据

4. 关流

接收端：

1. 创建套接字对象，绑定要监听的接收端口

2. 准备数据包

3. 接收数据

4. 关流

5. 将数据从数据包进行解析

练习：单人聊天---一个线程表示发送端，另一个线程表示接收端

255.255.255.255在UDP中表示的是一个广播地址

### TCP

基于流的。需要建立连接，经过三次握手，可靠，传输速率相对较低。理论上不对数据的大小进行限制。---适用于要求可靠性而不要求传输速率的场景 文件传输、文字聊天

客户端：---Socket

1. 创建客户端的套接字对象

2. 发起连接，绑定地址

3. 获取输出流对象，写出数据

4. 通知服务器端数据写完

5. 关流

服务器端：---ServerSocket

1. 创建服务器端的套接字对象，并且绑定监听的端口号

2. 接收连接，获取到一个Socket对象

3. 获取输入流，读取数据

4. 通知客户端数据已经读完

5. 关流

注意：receive/connect/accept/read/write都会产生阻塞

扩展：BIO --- Blocking IO --- 同步式阻塞式IO

NIO --- New IO --- NonBlocking IO --- 同步式非阻塞式IO --- JDK1.4

AIO --- Asynchronous IO --- 异步式非阻塞式IO --- jdk1.8

练习：文件的上传

网络编程.docx

扩展：如何传输一个目录？

扩展：利用TCP实现客户端之间的聊天

class Message implements Serializable {

String hostname; // 本机

String toAdrress; // 对方IP

byte type; // 确定请求类型

// 0 文字

// 1 语音

byte[] data; // 数据

int port; // 端口号

// 根据请求类型确定对应的端口号

// 文字 2001-3000

// 语音 3001-4000

}

## 注解

解释程序---给机器看的

用@interface来定义注解---jdk1.5出现的

@Override

@SuppressWarnings---压制警告

元注解：修饰注解的注解

@Target---限制注解的使用范围的--- 如果注解没有指定范围，则表示任意一个范围都可以使用

@Retention --- 限制注解出现的时间的

@Documented --- 表示当前这个注解能够生成到文档中

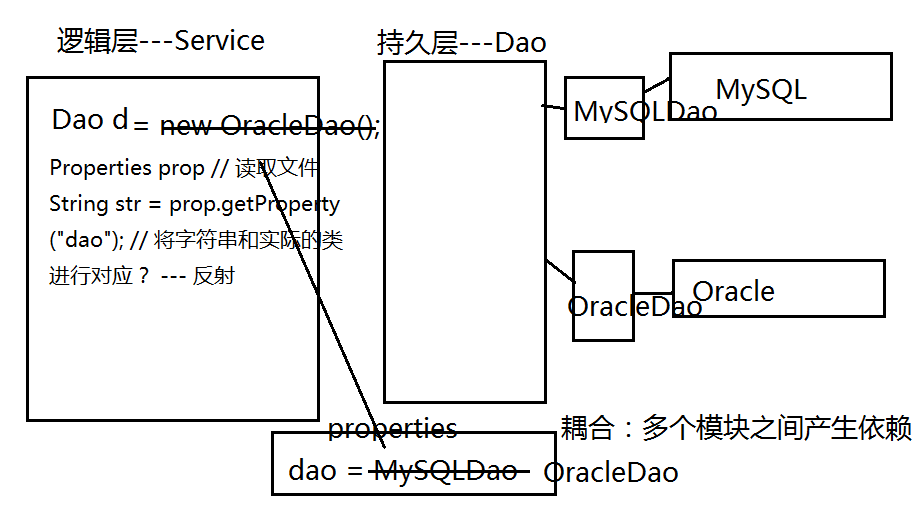
@Inherited --- 表示此注解能够限制子类

复习：

1. UDP：基于流的。不建立连接，不可靠，传输速率相对较快。在发送数据的时候需要对象数据进行封包，每个包不超过64K --- 发送端和接收端 --- DatagramSocket。

2. TCP：基于流的。建立连接，经过三次握手，可靠，传输速率相对较慢。不限制数据的大小 --- 客户端(Socket)和服务器端(ServerSocket)

## 反射



能够实现解耦操作

Class --- 代表字节码的类 --- 代表类的类

Field --- 代表属性的类

Constructor --- 代表构造方法的类

Method --- 代表方法的类

Package --- 代表包的类

解剖一个类，然后分析这个类的字节码以获取字节码对象以及对应的实例对象

### 如何去获取一个Class对象？

1. 通过类名.class方式来获取一个字节码对象

2. 通过对象.getClass()方式来获取这个对象的实际类型对应的字节码

3. 通过Class.forName(类的全路径名)方式来获取指定类的字节码

### 获取实例对象

1. 如果这个类中提供了无参构造，那么可以直接利用字节码对象.newInstance()来获取对应的实例对象

2. 如果想使用这个类中含参构造来创建对应的实例对象，那么需要先获取这个类中对应形式的构造函数，然后再调用newInstance(Object...)来创建实例对象

如果在获取这个类中的属性/方法/构造方法的时候，这些属性/方法是非公有的，那么在执行之前需要进行暴力破解---setAccessiable

## JVM的一些运行参数

栈内存(Stack)：执行代码块---计算---每一个线程独有

堆内存(Heap)：存储对象 --- 被所有线程共享的

新生代(Young Generation)：伊甸园区(eden)和幸存区(Survivor)

老生代(Old Generation)

持久代(Permanent)/方法区(Method Area)：存储---被所有线程所共享的

静态常量池：存储类信息 --- 字节码

运行时常量池：字面量(数值、字符、字符串、逻辑值和null)和自定义常量

本地方法栈(Native Stack)：执行本地方法（native）---每一个线程独有的

寄存器/PC计数器(programming counter)：用于存储和调度线程中的指令---每一个线程都都有一个PC计数器

所有的栈所占用的内存不能超过物理内存的1/3或者是2G

堆内存所占用的大小不能超过物理内存的1/4或者是2G

-... 标准参数

-X... 非标准参数

-Xss 用于限制栈内存大小的

-Xss128K 表示栈内存是128K

-Xms 表示堆内存分配的初始大小

-Xms5M 表示堆内存初始大小是5M

-Xmx 限制堆内存的最大占用大小

-Xmx10M 表示堆内存最大是10M

-Xmn 限制新生代的大小

扩展参数 -XX:

-XX:+PrintGC

-XX:+PrintGCDetails

### 回收机制

Mark-Cleaning：先找到无用的对象，然后将对象标记，最后清理标记的对象---清理相对较快，但是会产生内存的碎片化

Copying：将内存分为两块，使用其中的一块，如果要回收，就将不需要回收的对象挪到另一块去，然后重建要回收的这一块区域 --- 不会产生内存的碎片化，但是耗费内存并且耗费时间

Mark-Sweeping：既能保证清理速度又能保证内存不会产生碎片化

jdk1.7采取的回收器是G1回收，G1回收算法针对新生代和老生代采取不同的回收机制。