[一、java基础知识： 3](#_Toc2067765)

[一、计算机基础： 3](#_Toc2067766)

[1、计算机存储单位： 3](#_Toc2067767)

[2、Java程序运行过程 3](#_Toc2067768)

[3、Java的安装环境的搭建 4](#_Toc2067769)

[4、进制转换 5](#_Toc2067770)

[（1）、十进制----二进制转换 5](#_Toc2067771)

[（2）、十进制转八进制： 5](#_Toc2067772)

[（3）、十进制转十六进制： 6](#_Toc2067773)

[5、计算机中的运算规则 7](#_Toc2067774)

[二、语法变量 8](#_Toc2067775)

[1、类型转换： 9](#_Toc2067776)

[三、运算符 9](#_Toc2067777)

[1、比较运算符 9](#_Toc2067778)

[2、 9](#_Toc2067779)

[3、位运算符右移后补1 9](#_Toc2067780)

[4、任何数与一个指定的数异或两次，其结果为自身 10](#_Toc2067781)

[【经典案例1】文档加密解密 10](#_Toc2067782)

[5、移位运算符 10](#_Toc2067783)

[例一：iC3,vQgqBLt< 10](#_Toc2067784)

[例二： 11](#_Toc2067785)

[【经典案例2】：用移位的方式实现十进制转十六进制 11](#_Toc2067786)

[四、程序的流程控制 12](#_Toc2067787)

[1、顺序结构 12](#_Toc2067788)

[2、选择结构 12](#_Toc2067789)

[3、循环结构 14](#_Toc2067790)

[五、一维数组 15](#_Toc2067791)

[1、数组空指针异常 15](#_Toc2067792)

[2、遍历数组 15](#_Toc2067793)

[3、数组排序 15](#_Toc2067794)

[（1）、选择排序 16](#_Toc2067795)

[（2）冒泡排序 17](#_Toc2067796)

[（3）插入排序 18](#_Toc2067797)

[六、二维数组 19](#_Toc2067798)

[七、练习 20](#_Toc2067799)

[八、递归 21](#_Toc2067800)

[九、Collection \ Set \ Properties学习 23](#_Toc2067801)

[十、Calendar日期对象学习 24](#_Toc2067802)

[二、面向对象 25](#_Toc2067803)

[1、this关键字 25](#_Toc2067804)

[三、IO流 26](#_Toc2067805)

[1、文件夹及内容的拷贝 26](#_Toc2067806)

[2、文件夹的删除 27](#_Toc2067807)

[java的261个问题： 28](#_Toc2067808)

[1、重写方法需要注意的问题： 28](#_Toc2067809)

[2、构造方法可以被重载，但是不能被继承重写 28](#_Toc2067810)

[3、static修饰的方法 28](#_Toc2067811)

[4、重写toString()方法 29](#_Toc2067812)

[5、为什么重写equals方法的时候需要重写hashcode方法 29](#_Toc2067813)

[6、创建类的对象时候，类中各成员的执行顺序： 30](#_Toc2067814)

[7、调用内部类方法 31](#_Toc2067815)

[8、内部类和外部类属性相同的时候区分： 31](#_Toc2067816)

[9、匿名内部类访问外部方法的局部变量或参数 32](#_Toc2067817)

[10、异常处理机制 32](#_Toc2067818)

[11、Comparor排序 33](#_Toc2067819)

[12、io流 35](#_Toc2067820)

[13、文件的创建、删除、移动 36](#_Toc2067821)

[14、文件夹的创建、删除、移动 37](#_Toc2067822)

[15、流 38](#_Toc2067823)

[16、事务 39](#_Toc2067824)

[17、/u编码格式转换成为文字 40](#_Toc2067825)

[18、servlet 41](#_Toc2067826)

[19、获取当前系统的物理内存 42](#_Toc2067827)

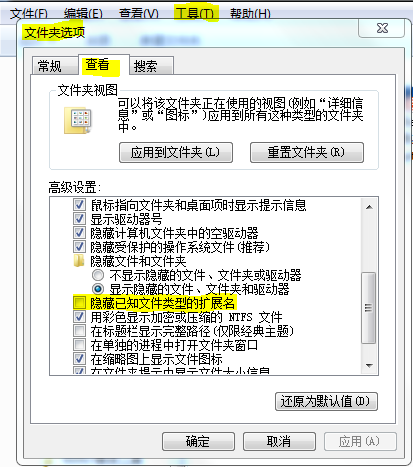
[20、组合 43](#_Toc2067828)

[21、利用反射解耦 44](#_Toc2067829)

[22、可变参数 48](#_Toc2067830)

# 一、java基础知识：

显示文件扩展名：



## 一、计算机基础：

### 1、计算机存储单位：

1(b)字节 = 8bit;

1kb = 2^10b = 1024b ;

1mb = 1024k;

1G = 1024M;

1T = 1024G;

2^64 = 1024\*1024\*1024\*1024\*1024\*1024\*16

计算5+9=14：

00000101+00001001

java：一次编译，到处运行

### 2、Java程序运行过程

（.java源程序 => 字节码中间文件.class => 安装JVM虚拟机（.exe文件）（解释字节码文件））

控制台命令：cmd => f: => cd ctrl+v

javac FirstJavaDemo.java

java FirstJavaDemo

### 3、Java的安装环境的搭建

1、jdk:开发环境 java\javac源文件\及基础核心类

jre:运行环境，java及基础核心类

2、java的应用环境划分：

j2se:标准版

j2ee:企业版

j2me:小型版

3、配置环境变量：

可执行程序都在bin目录下面，源文件在src.zip下面，lib下的rt.jar是核心类库

4、更改javac的path路径

set path=%path%;C:program...\java\jdk1.5.0\bin

set path

（2）我电脑=>高级：系统变量

用户变量会覆盖系统变量；

查看path：set path=%path%;

path环境变量：当我们运行一个exe可执行文件时候，系统会首先在当前目录下面找，如果没有去注册表找，如果还没有就去path环境变量中找。

需要成功执行的javac.exe和java.exe命令 ，就应该将这两个命令所在的目录追加到path环境变量中。

想启动什么只需要把目录到bin加到path里面，之后控制台直接输入qq

5、类加载时候，java虚拟机会在classpath指定的目录下找对应的.class文件

编译的时候可以用绝对路径：javac +绝对路径（认windows目录）

运行时候之后能是类名：java 绝对路径（不认Windows目录）

classpath环境变量：虚拟机在运行一个程序的时候，需要首先加载字节码文件，虚拟机不认操作系统的路径，它只会到classpath环境变量配置的目录下去找是否存在指定的.class文件。

查看classpath：set classpath

set classpath=.或者将目录整个拷贝过来

set classpath=F:\java\day在任何目录都可以运行

set classpath=abc

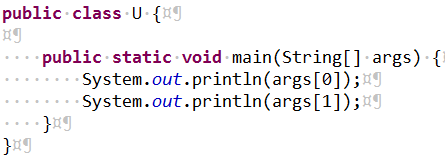
java Test

6、cd .. ：退出一级目录

cd 目录：进入目录

cd\ :退出全部目录

7、给mian方法赋初始值

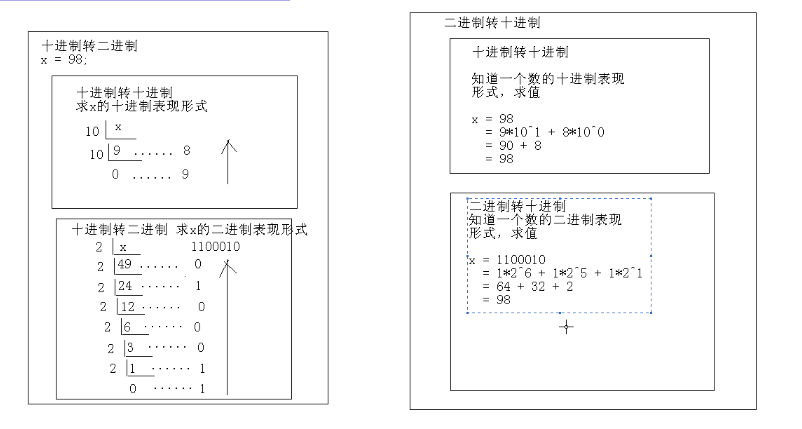


javac U.java ---> java U tome rose ---> 输出 ：tome rose

### 4、进制转换

（1）、十进制----二进制转换

（二进制转十进制: 每一位\*2的对应位数次幂 10=1\*2^1+0\*2^0）



读取键盘：

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.*in*));

System.*out*.println("本程序提供十进制转化二进制");

String line = br.readLine();

**int** num = Integer.*parseInt*(line);

StringBuffer str = new StringBuffer();

while (num > 0) {

int yushu = num % 2;

str.append(yushu);

num = num / 2;

}

str.reverse();

System.out.println(str);

#### （2）、十进制转八进制：

（8进制转10进制，每一位\*8的对应位数次幂 17=1\*8^1+7\*8^0）(十六进制相同)

**public** **static** **void** main(String[] args) **throws** IOException {

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.*in*));

System.*out*.println("本程序提供十进制转化八进制");

String line = br.readLine();

**int** num = Integer.*parseInt*(line);

StringBuffer str = **new** StringBuffer();

**while** (num > 0) {

**int** yushu = num % 8;

str.append(yushu);

num = num / 8;

}

str.reverse();

System.*out*.println(str);

}

注意：16禁止必须以0x开头，8禁止必须以0开头；

#### （3）、十进制转十六进制：

（注意abcdef的代替）

第一种：比较麻烦：

**public** **static** **void** main(String[] args) **throws** IOException {

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.*in*));

System.*out*.println("本程序提供十进制转化八进制");

String line = br.readLine();

**int** num = Integer.*parseInt*(line);

StringBuffer str = **new** StringBuffer();

**while** (num > 0) {

**int** yushu = num % 16;

String yushuNow = *replace16Binary*(yushu);

str.append(yushuNow);

num = num / 16;

}

str.reverse();

System.*out*.println(str);

}

**public** **static** String replace16Binary(**int** yushu) {

String yushuNow = String.*valueOf*(yushu);

**if** (yushu <= 15 && yushu >= 10) {

yushuNow = yushuNow.replace("10", "a").replace("11", "b")

.replace("12", "c")

.replace("13", "d").replace("14", "e").replace("15", "f");

}

**return** yushuNow;

}

第二种：比较简单

**public** **static** **void** main(String[] args) **throws** IOException {

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.*in*));

System.*out*.println("本程序提供十进制转化八进制");

String line = br.readLine();

**int** num = Integer.*parseInt*(line);

StringBuffer str = **new** StringBuffer();

**while** (num > 0) {

**int** yushu = num % 16;

**if** (yushu > 9) {

str.append((**char**) (yushu - 10 + 'A'));

} **else** {

str.append(yushu);

}

num = num / 16;

}

str.reverse();

System.*out*.println(str);

}

第三种：特殊

**public** **static** **void** main(String[] args) {

**int** num = 198;

StringBuffer sb = **new** StringBuffer();

**char**[] buf = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a',

'b', 'c', 'd', 'e', 'f' };

**for** (**int** i = 0; i < 8; i++) {

**int** temp = num & 0x0000000f;

// sb.append(buf[temp]); //查表法

sb.append(Character.*forDigit*(temp, 16)); // 调用Character类的静态方法，将数组转成制定基数的字符 将数字temp转换成16进制的字符

num = num >> 4;

}

sb.reverse();

System.*out*.println(sb);

}

### 5、计算机中的运算规则

（1）所有的运算都演变成通过累加器对二进制进行加法运算

计算机负数表示：

5-5 ---> 5+（-5）

1、获得源码，出去符号位的数字转成二进制

5 ---> 00000101

2、获得反码:将源码的每一分取反

11111010

3、获得补码：将反码加1

11111011(-5)（这个就是-5的表示方法）

+ 00000101

1 00000000 （1代表溢出，没有意义）

（2）ASCii码表格

a:97 A:65 最大是126

注意：回车占2个字节：因为敲击回车的时候，相当于按了回车和换行两个字节。

/\*\*

\* AscII码表：键盘上所有的键在词表中都对应一个字符，有一个特殊的，enter键对应两个（\r\n）

\* 每个字符占一个字节

\* gb2312码表：英文字符占一个字节（正数)，汉子占两个字节（负数）

\* gbk码表（增容）：英文字符占一个字节（正数)，汉子占两个字节（第一个为负数，第二个字节可正可负）

\* Unicode码表：国际码表，收罗了世界上大部分常用文字，每一个字符占两个字节，前两个字节为特征码

\* UTF-8码表：国际码表，每个字符占1~3个字节，实验证明英文占一个，汉字占三个，前三个为特征码

\*/

**public** **static** **void** main(String[] args) {

/\*\*

char c = 'A';

System.out.println(c + 1); 输出66

\*/

/\*\*

int num = 98;

System.out.println((char) num); //输出b

\*/

System.*out*.println("aaaaa\r\nggg");

}

## 二、语法变量

严格区分大小写

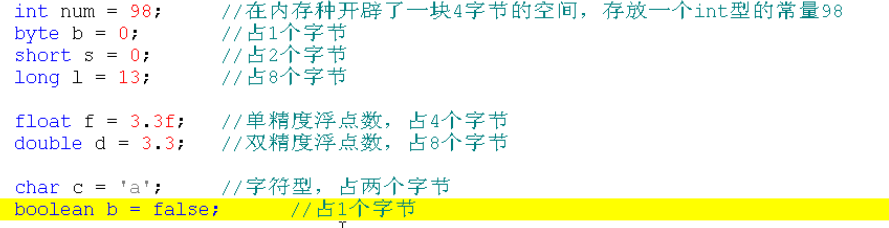
查找帮助文档：c:\\programs\\java\jdk\src.zip\java\long\ 注释就是用法

字符常量："\t"制表符 table键

转义字符 "\b"退格键 Back Space键

" \" " 双引号 '\'单引号

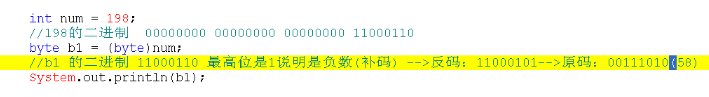
"\\"是一个斜杠



### 1、类型转换：

（1）byte、short、char 类型在运算过程在都会类型提升为int 型，这叫做表达式类型自动提升，也叫隐式类型转换。

（2）我们可以将一个int型强转为byte、short、char类型，但是有时候会丢失精度，这叫做强制类型转换，也叫显式类型转换



任何数和字符串相加，都为一个字符串。

变量的生命周期：在方法内定义的变量，当方法被调用的时候，变量被定义，当方法调用结束，变量会随之释放，这期间，我们成为变量的生命周期，也是变量的作用域。变量的作用域内，不允许重复定义，超出变量的作用域，变量将不能被访问。

函数：

函数的形参：方法定义的形参相等于在方法第一行定义的局部变量，当方法被调用的时候就被定义了，并完成了初始化，初始值为传入的实参。

在取模运算中，模数的符号会被忽略 5%-1 = 1 被模数的符号不会被忽略-5%-2 =-1

经典例题1：一个x个学生，每个房间可以住6人，需要几个房间？ 【return (x+5)/6】

经典例题2：假设你要让x的值在0--9之间循环变化 int x = 0; while(true){x = (x+1)%10}

## 三、运算符

### 1、比较运算符

instanceof,检测是否是类的对象，"hello" instanceof tring :true

Person p = new Person();

p instanceof Person ===> true

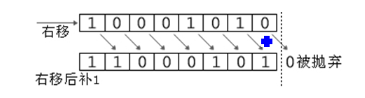
2、String str = null ;【防止出现空指针异常，如果此处为空，就不再判断.length==0，短路了】

if( str!=null && str.length==0 ){

str="abc";

}

### 3、位运算符右移后补1





4、任何数与一个指定的数异或两次，其结果为自身，文件加密

【经典案例】：不用第三方的变量交换两个数的值

第一种：

int x=1;int y=2

x = x+y;

y = x-y;

x = x-y;

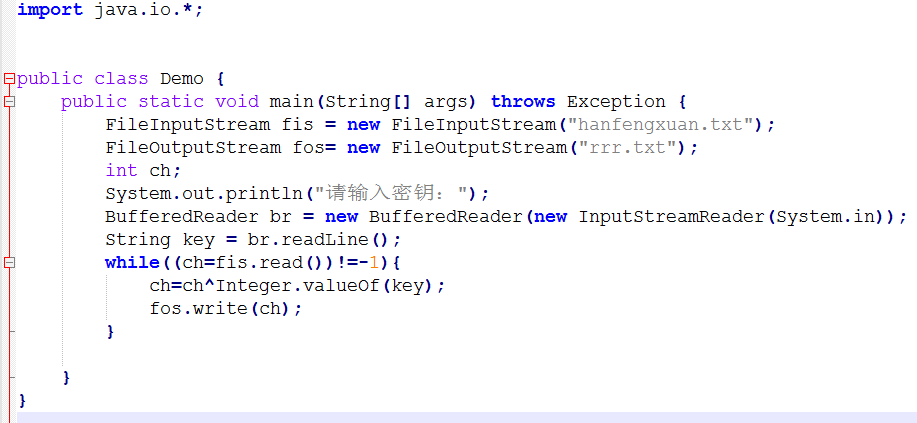
第二种：

x=x^y;

y=x^y;

x=x^y;

#### 【经典案例1】文档加密解密



### 5、移位运算符

#### 例一：iC3,vQgqBLt<

01001001001

<<4

10010010000

>>4（因为01001001001是以0开头，所有占4个0）

00000100100

#### 例二：

1001011101

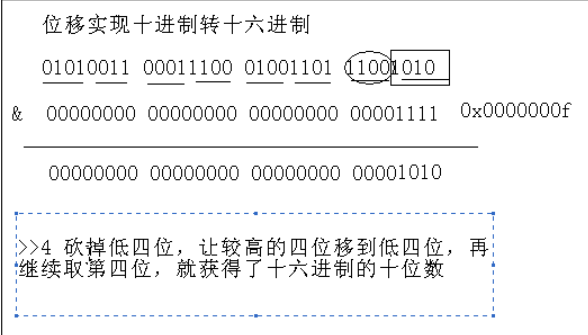
>>4（因为1001011101是以1开头，所有补4个1）

1111100101

>>>4(无符号右移，不管是以0开头，还是以1开头，都补0）

0000100101

#### 【经典案例2】：用移位的方式实现十进制转十六进制



**public** **class** Test {

**public** **static** **void** main(String[] args) {

**int** num = 198;

*ten2Hex*(num);

}

/\*

\* 十进制转十六进制 思路：

\* 1、定义一个容器StringBuffer,存十六进制的每一位

\* 2、让num&0x0000000f,获得石窟禁止的最低位

\* 3、存入StringBuffer

\* 4、让num>>4,砍掉低四位

\* 5、循环2--4步，用循环，循环8次，用for循环比较方便

\* 6、将容器反向打印输出

\*/

**static** **void** ten2Hex(**int** num) {

StringBuffer sb = **new** StringBuffer();

**for** (**int** i = 0; i < 8; i++) {

**int** temp = num & 0x0000000f;

**if** (temp > 9) {

sb.append((**char**) (temp - 10 + 'A'));

} **else** {

sb.append(temp);

}

num = num >> 4;

}

sb.reverse();

System.*out*.println(sb);

}

}

位移运算时，首先将会移动的位数对该类型占的位数进行取余，例如 a>>33相当于 a>>1

x >> 2 相当于 x / 22

x << 2 相当于 x \* 22

例题：

/\*\*

\* 实现2的n次方

\* **@author** Administrator

\*/

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**int** n = 8;

System.*out*.println(1 << n);

}

}

## 四、程序的流程控制

### 1、顺序结构

### 2、选择结构

switch函数中，byte\short\char\int可以作为参数，原因是自动类型提升

int x=3；

int y=1;

switch(x)

{

case 1;

x++;

break;

case 2:

y++;

break;

default:(当别的条件都不满足的时候走，但是也要加break；)

y--;

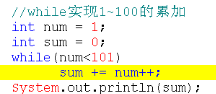
break;

}

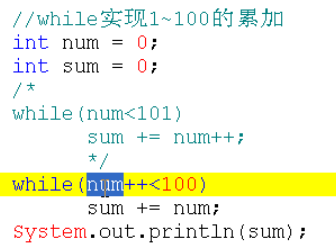
system.out.println(y)

案例：while后面不允许加：冒号

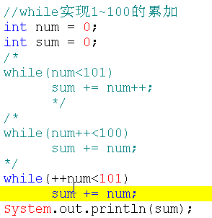
实现方法1:



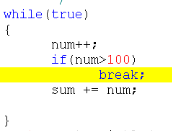
实现方法2：



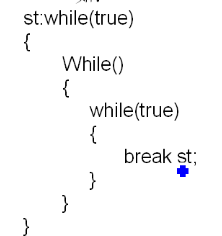
实现方法3：

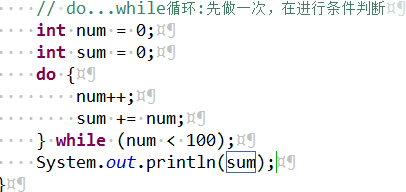


实现方法4:

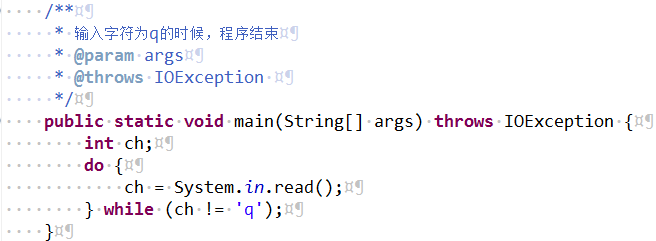


while用法：





案例2：



### 3、循环结构

// 打印等腰三角形，键盘输入

// 行数：0--n

// n-1 n-2..空格、1---n\*

**int** n = System.*in*.read() - 48;

System.*out*.println(n);

**for** (**int** i = 0; i < n; i++) {

**for** (**int** x = 0; x < n - i - 1; x++) {

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

}

**for** (**int** y = 0; y < 2 \* i + 1; y++) {

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

}

System.*out*.println();

}

## 五、一维数组

内存：

栈内存：存放的都是临时变量、随时存随时释放的变量eg:方法里面定义的局部变量

堆内存：持久对象，常量、对象

对象的首地址0xC000

1、数组空指针异常

如果数组为null，那么在访问时会报NUllPointerException

int arr[] =null; arr.lenth报空指针异常

int arr[]=new int[5]; arr[5] 报角标越界异常0~(length-1)，超过报此异常

### 2、遍历数组

**public** **static** **void** main(String[] args) {

**int** num[] = { 1, 2, 3, 4, 5 };

StringBuffer str = **new** StringBuffer();

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

str.append(num[i] + ",");

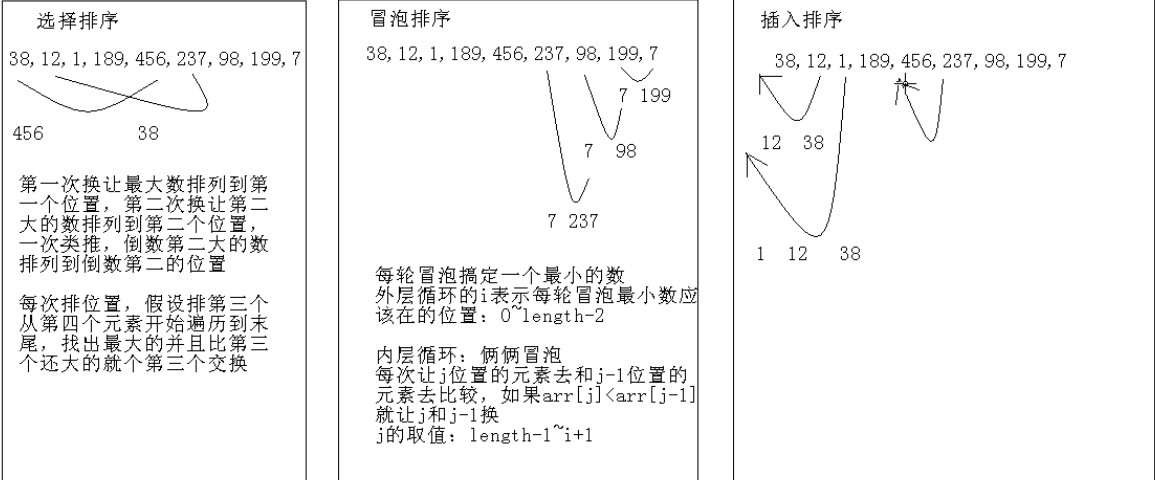
}

System.*out*.println(str.deleteCharAt(str.length() - 1));

System.*out*.println(str.delete(str.length(), str.length() + 1));

}

### 3、数组排序



#### （1）选择排序

**public** **static** **void** main(String[] args) {

**int** num[] = { 11, 88, 45, 198, 5 };

*listArr*(num);

System.*out*.println("-------------");

*selectSort*(num);

*listArr*(num);

}

/\*\*

\* 选择排序思路

\* 1、定义一个外层循环，控制要排列的次数，每次搞定一个最大数

\* 假设用i来代表每次最大数摆放的位置，i的取值 0 ~ length-2

\* 2、定义一个变量pos记住最大数所在的位置，定义一个变量max记住最大数

\* max初始值为i位置的元素,pos初始值为i

\* 3、定义顶层循环，用于遍历数组找最大数

\* 假设i 位置的最大，找出 i+1 ~ length-1里面最大的比i还大的

\* 遍历数组，如果有比max大的将值给max,同时用pos记住元素的角标

\* 4、让最大数和i位置的元素交换

\*/

**static** **void** selectSort(**int**[] arr) {

// { 198, 88, 45, 11, 5 }

**for** (**int** i = 0; i < arr.length - 1; i++) { // i=1

**int** pos = i; // pos=1

**int** max = arr[i]; // max=88

// 找最大数

**for** (**int** j = i + 1; j < arr.length; j++) { // j=2

**if** (arr[j] > max) { // 45>88

max = arr[j];

pos = j;

}

}

*exchange*(arr, i, pos);

}

}

**static** **void** exchange(**int**[] arr, **int** pos1, **int** pos2) {

**int** temp = arr[pos1];

arr[pos1] = arr[pos2];

arr[pos2] = temp;

}

**static** **void** listArr(**int**[] arr) {

StringBuffer sb = **new** StringBuffer();

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

sb.append(arr[i] + ",");

}

sb.delete(sb.length() - 1, sb.length());

System.*out*.println(sb);

}

#### （2）冒泡排序

每轮冒泡搞定一个最轻的数

外层循环的i表示每轮冒泡最小数应该在的位置，0~length-2

内层循环：俩俩冒泡

每次让j位置的元素去和j-1位置的元素去比较，如果arr[j]<arr[j-1]就让j和j-1换

j的取值，length-1~i+ 1

**public** **static** **void** main(String[] args) {

**int** num[] = { 11, 2, 7, 31, 4, 15 };

*listArr*(num);

*bubbleSort*(num);

*listArr*(num);

}

**static** **void** listArr(**int**[] arr) {

StringBuffer sb = **new** StringBuffer();

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

sb.append(arr[i] + ",");

}

System.*out*.println(sb.delete(sb.length() - 1, sb.length()));

}

/\*\*

\* 冒泡排序

\* 思路：

\* 1、定义一个外层循环，每次搞定一个最小数，i为最小数的位置，i的取值，0~length-2

\* 2、定义一个内层循环，从数组的末尾开始向前开始冒泡，知道i+1的位置和i位置的元素进行完冒泡

\* j的取值范围 ：length-1~i+1

\*/

**static** **void** bubbleSort(**int**[] arr) {

// { 2, 4, 11, 7, 15,31 }

**for** (**int** i = 0; i < arr.length - 1; i++) { //i=1

**for** (**int** j = arr.length - 1; j > i; j--) { //j=1

**if** (arr[j] < arr[j - 1]) {

*changeList*(arr, j, j - 1);

}

}

}

}

/\*\*

\* 交换位置

\*/

**static** **void** changeList(**int** arr[], **int** pos, **int** i) {

**int** temp = arr[i];

arr[i] = arr[pos];

arr[pos] = temp;

}

}

#### （3）插入排序

**public** **static** **void** main(String[] args) {

**int** num[] = { 11, 2, 7, 31, 4, 15 };

*listArr*(num);

*insertSort*(num);

*listArr*(num);

}

**static** **void** listArr(**int**[] arr) {

StringBuffer sb = **new** StringBuffer();

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

sb.append(arr[i] + ",");

}

System.*out*.println(sb.delete(sb.length() - 1, sb.length()));

}

/\*\*

\* 插入排序

\* 思路：外层循环每次搞定一个数的正确位置，将第i个元素插入到它应该在位置

\* i的范围 1~length-1

\* 内存循环：将第i个数不断的和前面比，小就换，直到遇上比它还小的数就停

\* 次数不定while

\* 如果次元素被换到了第0个位置，也要停

\*/

**static** **void** insertSort(**int**[] arr) {

// { 2, 4, 11, 7, 15,31 }

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

**int** pos = i; //pos记住被插入元素的角标

**while** (pos > 0 && arr[pos] < arr[pos - 1]) {

*changeList*(arr, pos, pos - 1);

pos--;

}

}

}

/\*\*

\* 交换位置

\*/

**static** **void** changeList(**int** arr[], **int** pos, **int** i) {

**int** temp = arr[i];

arr[i] = arr[pos];

arr[pos] = temp;

}

## 六、二维数组

（1）定义

// 二维数组定义

**int** arr1[][] = **new** **int**[5][10];

**int**[] arr2[] = **new** **int**[5][10];

**int** arr3[][] = **new** **int**[3][];

arr3[0] = **new** **int**[4];

arr3[1] = **new** **int**[3];

arr3[2] = **new** **int**[8];

// arr3[4] = new int[8]; 数组越界

System.*out*.println(arr1.length); // 长度为5

（2）遍历

/\*\*

\* 二维数组：数组的每一个元素又是一个数组

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

**int**[][] arr = { { 1, 2, 3 }, { 4, 5 }, { 6, 7, 8, 9 } };

System.*out*.println(arr.length); //3

// 遍历所有的元素

// 遍历整个数组

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

// 遍历每一个元素

**for** (**int** j = 0; j < arr[i].length; j++) {

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

}

}

（3）与数组相关的函数

**public** **static** **void** main(String[] args) {

**int**[] arr1 = { 15, 2, 4, 3, 11, 6, 43 };

**int**[] arr2 = { 13, 21, 6, 11, 12, 40 };

// 源数组，拷贝的起始位置，目标数组，目标数组的位置，拷贝的长度

System.*arraycopy*(arr1, 1, arr2, 2, 3);

*listArr*(arr2);

System.*out*.println("排序后");

Arrays.*sort*(arr2);

*listArr*(arr2);

}

**static** **void** listArr(**int**[] arr) {

**for** (**int** num : arr) {

System.*out*.print(num + " ");

}

}

## 七、练习

1、 键盘输入三个数字，输出最大数字

（1）办法1

**public** **static** **void** main(String[] args) **throws** IOException {

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.*in*));

**int** arr[] = **new** **int**[3];

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

System.*out*.println("请输入第" + (i + 1) + "个数字");

String line = br.readLine();

**int** num = Integer.*parseInt*(line);

arr[i] = num;

}

Arrays.*sort*(arr);

System.*out*.println("最大的数字为" + arr[arr.length - 1]);

}

（2）

**public** **static** **void** main(String[] args) **throws** IOException {

**int** x = Integer.*parseInt*(args[0]);

**int** y = Integer.*parseInt*(args[1]);

**int** z = Integer.*parseInt*(args[2]);

**int** max = *getMaxNum*(x, y, z);

System.*out*.println("最大数是" + max);

}

**static** **int** getMaxNum(**int** x, **int** y, **int** z) {

**int** max = x > y ? x : y; //二元表达式

**return** max > z ? max : z;

}

## 八、递归

**public** **static** **void** main(String[] args) {

/\*\*

\* 函数的递归：就是函数自己调用自己，必须要有结束条件

\*/

**int** n = 8;

**int** sum = *getSum*(n);

**int** fibo = *getFibo*(n);

*System.out.println(sum);*

*System.out.println(fibo);*

*ten2Binary(98);*

}

// 设计一个方法，求1~n的和

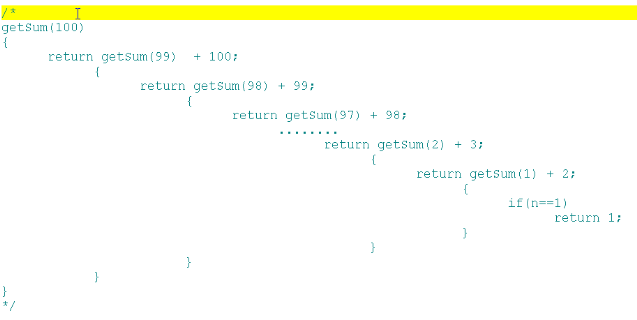
**static** **int** getSum(**int** n){

**if** (n == 1)

**return** 1;

**return** *getSum*(n - 1) + n;

}



// 设计一个方法 1 ，1，2，3，5，8，13...,求第n个数是什么

**static** **int** getFibo(**int** n) {

**if** (n == 1)

**return** 1;

**if** (n == 2)

**return** 1;

**return** *getFibo*(n - 1) + *getFibo*(n - 2);

}

// 设计十进制转二进制

**static** **void** ten2Binary(**int** num) {

**if** (num == 0)

**return**;

**int** temp = num % 2;

num = num / 2;

*ten2Binary*(num);

System.*out*.print(temp);

}

// 汉诺塔

**public** **static** **void** main(String[] args) {

*hano*(3, "a", "b", "c");

}

**public** **static** **void** hano(**int** n, String a, String b, String c) {

**if** (n == 1) {

System.*out*.println(a + "-->" + c);

**return**; 千万不要忘记返回，否则报错

}

**if** (n == 2) {

System.*out*.println(a + "-->" + b);

System.*out*.println(a + "-->" + c);

System.*out*.println(b + "-->" + c);

**return**;

}

*hano*((n - 1), a, c, b);

System.*out*.println(a + "-->" + c);

*hano*((n - 1), b, a, c);

}

## 九、Collection \ Set \ Properties学习

**package** day12;

**import** java.util.Comparator;

**import** java.util.Map.Entry;

**import** java.util.Set;

**import** java.util.TreeMap;

**import** java.util.TreeSet;

/\*\*

\* abaac 输出 a(3) b(1) c(1)

\* **@author** Administrator

\*

\*/

**public** **class** Test {

**public** **static** **void** main(String[] args) {

String str = "aaaabbbccccc";

*run*(str);

}

**private** **static** **void** run(String str) {

**char**[] c = str.toCharArray();

String s = "";

TreeMap<String, Integer> tm = **new** TreeMap<String, Integer>();

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

**if** (!s.contains(String.*valueOf*(c[i]))) {

s += c[i];

tm.put(String.*valueOf*(c[i]), 1);

} **else** {

**int** value = tm.get(String.*valueOf*(c[i])) + 1;

tm.put(String.*valueOf*(c[i]), value);

}

}

Set<Entry<String, Integer>> entrys = tm.entrySet();

TreeSet<Entry<String, Integer>> ts = **new** TreeSet<Entry<String, Integer>>(

**new** Comparator<Entry<String, Integer>>() {

@Override

**public** **int** compare(Entry<String, Integer> e1,

Entry<String, Integer> e2) {

**int** num = e1.getValue() - e2.getValue();

**if** (num != 0) {

**return** -num;

}

**return** e1.getKey().compareTo(e2.getKey());

}

});

**for** (Entry<String, Integer> entry : entrys) {

ts.add(entry);

}

**for** (Entry<String, Integer> entry : ts) {

System.*out*.println(entry.getKey() + "=====" + entry.getValue());

}

}

}

## 十、Calendar日期对象学习

**package** day12;

**import** java.util.Calendar;

**public** **class** CalendarTest {

**public** **static** **void** main(String[] args) {

Calendar c = Calendar.*getInstance*();

**int** nu1m = c.get(Calendar.*DAY\_OF\_MONTH*);

// 计算某年的二月有多少天

**int** num = 2008;

*getNum*(num);

Calendar start = Calendar.*getInstance*();

// 100天完工，开始今天，周末休息，10一五一放一天假

Calendar finish = *getFinishDate*(start);

**int** cha = finish.get(finish.*DAY\_OF\_YEAR*) - c.get(c.*DAY\_OF\_YEAR*);

System.*out*.println(cha + "天");

}

/\*\*

\* 100天完工，开始今天，周末休息，10一五一放一天假

\* **@param** c

\* **@return**

\*/

**private** **static** Calendar getFinishDate(Calendar c) {

**for** (**int** i = 0; i < 100; i++) {

**if** (c.get(c.*DAY\_OF\_WEEK*) == 1 || c.get(c.*DAY\_OF\_WEEK*) == 7) {

i--;

}

**if** ((c.get(c.*MONTH*) == 4 || c.get(c.*MONTH*) == 9)

&& (c.get(c.*DAY\_OF\_MONTH*) == 1)) {

i--;

}

c.add(c.*DAY\_OF\_YEAR*, 1);

}

System.*out*.println(c.get(c.*YEAR*) + "年" + (c.get(c.*MONTH*) + 1) + "月"

+ c.get(c.*DAY\_OF\_MONTH*) + "日");

**return** c;

}

/\*\*

\* 计算某年的二月有多少天

\* **@param** num

\*/

**private** **static** **void** getNum(**int** num) {

Calendar c = Calendar.*getInstance*();

c.set(num, 2, 1);

c.add(Calendar.*DAY\_OF\_YEAR*, -1);

System.*out*.println(c.get(Calendar.*DAY\_OF\_MONTH*));

}

}

# 二、面向对象

什么是面相对象：程序中用对象代表现实中的事物，对象存在着唯一性

## 1、this关键字

是对一个对象的引用，this出现在方法体内，具体引用哪个对象，就看调用方法的哪个对象一句话，谁调用我，我就指向谁。

三、包、类

编译带包的源文件命令：在生成class的时候同时生成包文件

编译命令：javac -d <目录> 源文件名.java 【目录：就是你想在哪编译的位置，如果是当前目录：就是"."】

# 三、IO流

## 1、自定义MyBufferedReader类

**import java.io.IOException;**

**import java.io.Reader;**

**public class MyBufferedReader extends Reader {**

**private Reader r;**

**public MyBufferedReader(Reader r) {**

**this.r = r;**

**}**

**private int len;**

**private int pos;**

**private char[] ch = new char[1024];**

**public int read() throws IOException {**

**if (len == 0) {**

**len = r.read(ch);**

**pos = 0;**

**}**

**if (len == -1) {**

**return -1;**

**}**

**len--;**

**return ch[pos++];**

**}**

**public String readLine() throws IOException {**

**StringBuilder sb = new StringBuilder();**

**int ch;**

**while ((ch = read()) != -1) {**

**if (ch == '\r') {**

**continue;**

**}**

**if (ch == '\n') {**

**break;**

**}**

**sb = sb.append((char) ch);**

**}**

**if (ch == -1 && sb.length() == 0) {**

**return null;**

**} else {**

**return sb.toString();**

**}**

**}**

**@Override**

**public int read(char[] cbuf, int off, int len) throws IOException {**

**int count = 0;**

**for (int i = off; i < off + len; i++) {**

**int ch = read();**

**if (ch == -1) {**

**break;**

**}**

**count++;**

**cbuf[i] = (char) ch;**

**}**

**return count;**

**}**

**@Override**

**public void close() throws IOException {**

**r.close();**

**}**

**}**

## 2、自定义MyLineNumberReader类

**package test;**

**import java.io.IOException;**

**import java.io.Reader;**

**public class MyLineNumberReader extends MyBufferedReader{**

**public MyLineNumberReader(Reader r) {**

**super(r);**

**}**

**public String readLine() throws IOException {**

**lineNumber++;**

**return super.readLine();**

**}**

**private int lineNumber;**

**public int getLineNumber() {**

**return lineNumber;**

**}**

**public void setLineNumber(int lineNumber) {**

**this.lineNumber = lineNumber;**

**}**

**}**

## 3、自定义MyBufferedWriter

**package test;**

**import java.io.IOException;**

**import java.io.Writer;**

**public class MyBufferedWriter extends Writer {**

**private Writer w;**

**public MyBufferedWriter(Writer w) {**

**this.w = w;**

**}**

**private char[] buf = new char[1024];**

**private int pos;**

**public void write(int ch) throws IOException {**

**if (pos == 1024) {**

**this.flush();**

**pos = 0;**

**}**

**buf[pos++] = (char) ch;**

**}**

**public void write(String str) throws IOException {**

**char[] c = str.toCharArray();**

**for (char cha : c) {**

**this.write(cha);**

**}**

**}**

**public void newLine() throws IOException {**

**this.write("\r\n");**

**}**

**@Override**

**public void write(char[] cbuf, int off, int len) throws IOException {**

**for (int i = off; i < off + len; i++) {**

**this.write(buf[i]);**

**}**

**}**

**@Override**

**public void flush() throws IOException {**

**w.write(buf, 0, pos);**

**}**

**@Override**

**public void close() throws IOException {**

**this.flush();**

**w.close();**

**}**

**}**

## 4、文件夹及内容的拷贝

**package** filestudy;

**import** java.io.BufferedInputStream;

**import** java.io.BufferedOutputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**public** **class** CopyDir {

**public** **static** **void** main(String[] args) {

File source = **new** File("E:/source");

File target = **new** File("E:/target");

**try** {

*copyDir*(source, target);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**private** **static** **void** copyDir(File source, File target) **throws** IOException {

**if** (source.isDirectory()) {

File newDir = **new** File(target, source.getName());

newDir.mkdir();

File[] files = source.listFiles();

**for** (File file : files) {

*copyDir*(file, newDir);

}

} **else** **if** (source.isFile()) {

*copyFile*(source, target);

}

}

**private** **static** **void** copyFile(File source, File target) **throws** IOException {

File newFile = **new** File(target, source.getName());

newFile.createNewFile();

BufferedInputStream bis = **null**;

BufferedOutputStream bos = **null**;

**try** {

bis = **new** BufferedInputStream(**new** FileInputStream(

source));

bos = **new** BufferedOutputStream(

**new** FileOutputStream(newFile));

**int** ch;

**while** ((ch = bis.read()) != -1) {

bos.write(ch);

}

} **finally** {

**try** {

**if** (bis != **null**) {

bis.close();

}

} **finally** {

**if** (bos != **null**) {

bos.close();

}

}

}

}

}

## 5、自定义MyBufferedInputStream

**package baozhuang;**

**import java.io.IOException;**

**import java.io.InputStream;**

**public class MyBufferedInputStream extends InputStream {**

**private InputStream in;**

**private byte[] but = new byte[1024];**

**private int len;**

**private int pos;**

**public MyBufferedInputStream(InputStream in) {**

**this.in = in;**

**}**

**public void close() throws IOException {**

**in.close();**

**}**

**@Override**

**public int read() throws IOException {**

**if (len == 0) {**

**len = in.read(but);**

**pos = 0;**

**}**

**if (len == -1) {**

**return -1;**

**}**

**len--;**

**return but[pos++] & 0xff;//buf[pos++]&0xff**

**}**

**}**

## 6、自定义MyBufferedOutputStream类

**import java.io.IOException;**

**import java.io.OutputStream;**

**public class MyBufferedOutputStream extends OutputStream {**

**private OutputStream out;**

**public MyBufferedOutputStream(OutputStream out) {**

**this.out = out;**

**}**

**private int pos;**

**private byte[] but = new byte[1024];**

**@Override**

**public void write(int b) throws IOException {**

**if (pos == 1024) {**

**this.flush();**

**pos = 0;**

**}**

**but[pos++] = (byte) b;**

**}**

**public void write(String str) throws IOException {**

**char[] chs = str.toCharArray();**

**for (char c : chs) {**

**this.write(c);**

**}**

**}**

**public void newLine() throws IOException {**

**this.write("\r\n");**

**}**

**public void flush() throws IOException {**

**out.write(but, 0, pos);**

**}**

**public void close() throws IOException {**

**this.flush();**

**out.close();**

**}**

**}**

## 7、文件夹的删除

**package** filestudy;

**import** java.io.File;

**public** **class** DeleteDir {

**public** **static** **void** main(String[] args) {

File file = **new** File("e:/target");

*deleteDir*(file);

}

**private** **static** **void** deleteDir(File file) {

**if** (file.isDirectory()) {

File[] files = file.listFiles();

**for** (File f : files) {

*deleteDir*(f);

}

}

file.delete();

}

}

## 8、列出某个目录的全部文件的绝对路径名称

**package** test;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.util.ArrayList;

/\*\*

\*

/\*★★★★

====第四题====================================

编写一个程序，当用户输入一个目录时，

该程序能列出该目录下的所有子目录和文件。将目录和文件的绝对路径输出打印

思路：

1，获取用户输入的目录，封装成文件对象。

a.获取输入可以有两种形式，一种通过System.in。另一种通过给主函数传值的形式。

b.把输入的字符串封装成File对象。对其进行判断。该文件夹是否存在。

2，通过递归的形式列出目录所有文件夹和文件

a.定义一个功能方法，遍历该文件夹对象，如果遍历到的File对象是文件夹就再次调用该功能方法（递归）；

\*/

**public** **class** Test3 {

**public** **static** **void** main(String[] args) {

**try** {

File targetFile = *getFile*();

ArrayList<File> as = **new** ArrayList<File>();

*getListFile*(targetFile, as);

*systemList*(as);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**private** **static** **void** systemList(ArrayList<File> as) {

**for** (File f : as) {

System.*out*.println(f.getAbsolutePath());

}

}

**private** **static** **void** getListFile(File targetFile, ArrayList<File> as) {

File fs[] = targetFile.listFiles();

**if** (fs == **null**) {

**return**;

}

**for** (File f : fs) {

**if** (f.isDirectory()) {

*getListFile*(f, as);

} **else**{

as.add(f);

}

}

}

**private** **static** File getFile() **throws** IOException {

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.*in*));

String filaname = "";

**try** {

**while** (**true**) {

filaname = br.readLine();

File f = **new** File(filaname);

**break**;

}

} **finally** {

br.close();

}

**return** **new** File(filaname);

}

}

9、列出某个目录的树状结构图

**package filestudy;**

**import java.io.BufferedWriter;**

**import java.io.File;**

**import java.io.FileWriter;**

**import java.io.FilenameFilter;**

**import java.io.IOException;**

**public class FileBasic {**

**public static void main(String[] args) {**

**File file = new File("E:\\delete");**

**BufferedWriter bw = null;**

**try {**

**bw = new BufferedWriter(new FileWriter("day15/a.txt"));**

**listAllFiles(file, bw, 0);**

**} catch (IOException e) {**

**e.printStackTrace();**

**} finally {**

**try {**

**if (bw != null) {**

**bw.close();**

**}**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**private static void listAllFiles(File file, BufferedWriter bw, int i) throws IOException {**

**bw.write(getOrder(i) + file.getName());**

**bw.newLine();**

**if (file.isDirectory()) {**

**File[] files = file.listFiles();**

**for (File f : files) {**

**listAllFiles(f, bw, i + 1);**

**}**

**}**

**}**

**private static String getOrder(int order) {**

**String listStr = "|-";**

**for (int i = 0; i < order; i++) {**

**listStr += "--";**

**}**

**return listStr;**

**}**

**}**

**class MyFileNameFilter implements FilenameFilter {**

**public boolean accept(File dir, String name) {**

**if (name.endsWith(".java"))**

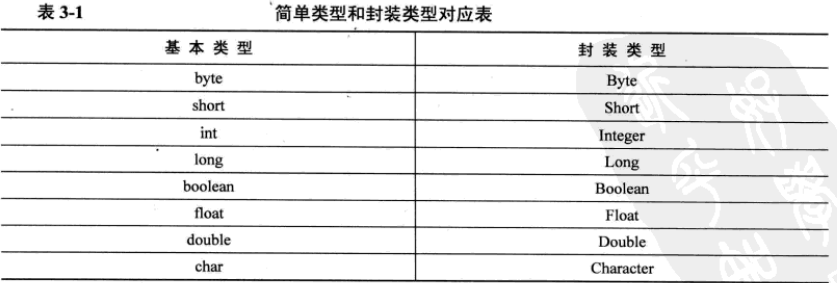
**return true;**

**return false;**

**}**

**}**

# java的261个问题：



## 1、重写方法需要注意的问题：

如果父类中有throws抛出的异常，那么子类抛出的异常可以和父类抛出的异常对象相同或者是负累异常的子类。

## 2、构造方法可以被重载，但是不能被继承重写

## 3、static修饰的方法

static修饰的方法在子类中不可以被重写，因为static关键字修饰的方法和属性，表示该属性和方法与类相关。当JVM运行static修饰的方法时，解释器不会检查当前变量指向哪个类型的变量，而是直接调用当前变量类型的方法。

例如：

public class A{

public static void show(){

System.out,println("A");

}

}

public class B extends A{

public static void show(){

System.out,println("B");

}

public static void main(String []args){

A a=new B();

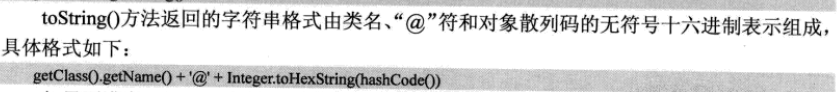
a.show();

}

}

输出结果为：A

## 4、重写toString()方法



## 5、为什么重写equals方法的时候需要重写hashcode方法

重写equals方法是为了方便比较两个对象的内容是否相等，hashcode方法用于返回调用该方法的对象的散列码值，此方法将返回整数形式的散列码值。如果两个数相等，那么返回的散列码值一定相等，但是两个数的散列码值相等，两个数不一定equals，两个不相等的数的散列码值不一定不相等。

@Override

**public** **boolean** equals(Object obj) {

**if** (obj == **null**)

**return** **false**;

**if** (**this** == obj)

**return** **true**;

**if** (**this**.getClass() != obj.getClass())

**return** **false**;

**if** (obj **instanceof** ZhzjEvidenceFactor) {

ZhzjEvidenceFactor zhzjEvidenceFactor = (ZhzjEvidenceFactor) obj;

**if** (**this**.getId().equals(zhzjEvidenceFactor.getId())

&& **this**.getCaseId().equals(zhzjEvidenceFactor.getCaseId())

&& **this**.getEvidenceId().equals(

zhzjEvidenceFactor.getEvidenceId())

&& **this**.getFactorType().equals(

zhzjEvidenceFactor.getFactorType())

&& **this**.getFactorValue().equals(

zhzjEvidenceFactor.getFactorValue())

&& **this**.getSelected().equals(

zhzjEvidenceFactor.getSelected())

&& **this**.getIgnore().equals(zhzjEvidenceFactor.getIgnore())

&& **this**.getIgnoreReason().equals(

zhzjEvidenceFactor.getIgnoreReason())) {

**return** **true**;

} **else** {

**return** **false**;

}

}

**return** **false**;

}

@Override

**public** **int** hashCode() {

**int** hash= 17;

hash = hash \* 31 + getId().hashCode();

hash = hash \* 31 + getCaseId().hashCode();

hash = hash \* 31 + getEvidenceId().hashCode();

hash = hash \* 31 + getFactorType().hashCode();

hash = hash \* 31 + getFactorValue().hashCode();

hash = hash \* 31 + getSelected().hashCode();

hash = hash \* 31 + getIgnore().hashCode();

hash = hash \* 31 + getIgnoreReason().hashCode();

**return** hash;

}

}

## 6、创建类的对象时候，类中各成员的执行顺序：

（1）父类静态成员和静态块

（2）子类静态成员和静态块

（3）父类普通块

（4）父类构造方法

（5）子类普通块

（6）子类构造方法

## 7、调用内部类方法

（1）在定义的类内部调用内部方法

**package** innerClass;

**public** **class** Outer {

**private** **int** size;

**public** **int** getSize() {

**return** size;

}

**public** **void** setSize(**int** size) {

**this**.size = size;

}

**public** **class** Inner {

**public** **void** getSize() {

++size;

}

}

**public void testInner() {**

**Inner inner = new Inner();**

**inner.getSize();**

**}**

**public** **static** **void** main(String[] args) {

Outer tset = **new** Outer();

tset.setSize(15);

tset.testInner();

System.*out*.println(tset.getSize());

}

}

（2）在其他类调用内部类方法

**public** **static** **void** main(String[] args) {

// 第一种

Outer.Inner in = **new** Outer().**new** Inner();

in.getSize();

// 第二种

Outer t = **new** Outer();

Outer.Inner in1 = t.**new** Inner();

in1.getSize();

}

## 8、内部类和外部类属性相同的时候区分：

**public** **class** Outer {

**private** **int** i = 100;

**class** B {

**private** **int** i = 10;

**public** **void** show() {

System.*out*.println(i);

System.*out*.println(Outer.**this**.i);

}

}

**public** **void** getShow() {

B b = **new** B();

b.show();

}

**public** **static** **void** main(String[] args) {

Outer outer = **new** Outer();

outer.getShow();

}

}

输出：10，100

## 9、匿名内部类访问外部方法的局部变量或参数

访问的变量或者参数必须使用final修饰

**public** **class** Outer {

**private** **int** m;

**public** **void** go(**int** x, **final** **int** y) {

**int** a;

**final** **int** b = 10;

**new** Thread(){

@Override

**public** **void** run() {

System.*out*.println("m=" + m);

// System.out.println("x=" + x);

System.*out*.println(y);

// System.out.println(a);

System.*out*.println(b);

**super**.run();

}

}.start();

}

}

## 10、异常处理机制

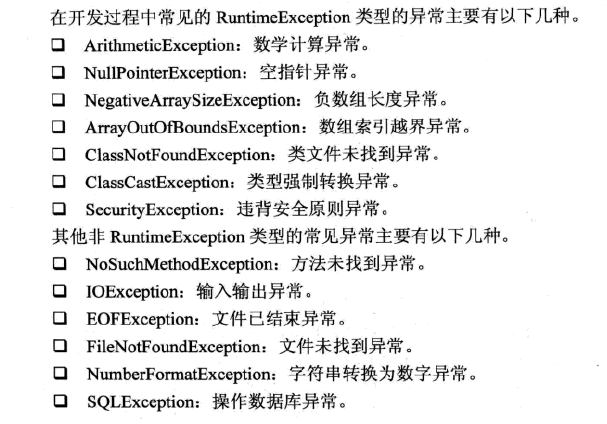
Error是指Java虚拟机内部发生的错误，由虚拟机生成并抛出，例如资源耗尽等情况。这些错误Java程序没有办法处理。

error和exception都是Throwable的子类。

（1）常见的运行时异常 RuntimeException

RuntimeException是Exception的子类：ArithmaticException【例如除数为0】

【数组越界异常】ArrayIndexOutOfBoundsException，其他异常就不是运行时异常，如【输入输出异常】IOException



（2）try catch是必须的语句块，catch可以有多个

（3）自定义异常

## 11、Comparor排序

**package** file;

**import** java.net.SocketException;

**import** java.net.UnknownHostException;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.LinkedHashMap;

**import** java.util.List;

**import** java.util.Map;

**public** **class** MapSort {

**public** **static** **void** main(String[] args) **throws** UnknownHostException, SocketException {

List<Map<String, Object>> listResult = **new** ArrayList<Map<String, Object>>();

Map<String, Object> map1 = **new** LinkedHashMap<String, Object>();

map1.put("count", 2);

map1.put("name", "abc");

map1.put("key", "acv");

listResult.add(map1);

Map<String, Object> map2 = **new** LinkedHashMap<String, Object>();

map2.put("count", 3);

map2.put("name", "bbc");

map2.put("key", "bcv");

listResult.add(map2);

Map<String, Object> map3 = **new** LinkedHashMap<String, Object>();

map3.put("count", 1);

map3.put("name", "cbc");

map3.put("key", "ccv");

listResult.add(map3);

Map<String, Object> map4 = **new** LinkedHashMap<String, Object>();

map4.put("count", 4);

map4.put("name", "cbc");

map4.put("key", "ccv");

listResult.add(map4);

System.*out*.println("排序前：");

**for** (Map<String, Object> map : listResult) {

System.*out*.println("count:" + map.get("count") + " name:"

+ map.get("name") + " key:" + map.get("key"));

}

Collections.*sort*(listResult, **new** MapComparatorDesc());

System.*out*.println("降序：");

**for** (Map<String, Object> map : listResult) {

System.*out*.println("count:" + map.get("count") + " name:"

+ map.get("name") + " key:" + map.get("key"));

}

Collections.*sort*(listResult, **new** MapComparatorAsc());

System.*out*.println("升序：");

**for** (Map<String, Object> map : listResult) {

System.*out*.println("count:" + map.get("count") + " name:"

+ map.get("name") + " key:" + map.get("key"));

}

}

**static** **class** MapComparatorDesc **implements** Comparator<Map<String, Object>> {

@Override

**public** **int** compare(Map<String, Object> m1, Map<String, Object> m2) {

Integer v1 = Integer.*valueOf*(m1.get("count").toString());

Integer v2 = Integer.*valueOf*(m2.get("count").toString());

**if** (v2 != **null**) {

**return** v2.compareTo(v1);

}

**return** 0;

}

}

**static** **class** MapComparatorAsc **implements** Comparator<Map<String, Object>> {

@Override

**public** **int** compare(Map<String, Object> m1, Map<String, Object> m2) {

Integer v1 = Integer.*valueOf*(m1.get("count").toString());

Integer v2 = Integer.*valueOf*(m2.get("count").toString());

**if** (v1 != **null**) {

**return** v1.compareTo(v2);

}

**return** 0;

}

}

}

## 12、io流

File file = **new** File("D:\\hello.txt");

System.*out*.println("\*\*\*\*\*\*文件属性测试\*\*\*\*\*\*");

System.*out*.println("是否存在:" + file.exists());

System.*out*.println("文件名：" + file.getName());

System.*out*.println("上级目录：" + file.getParent());

System.*out*.println("是否可读：" + file.canRead());

System.*out*.println("是否可写：" + file.canWrite());

System.*out*.println("绝对路径：" + file.getAbsolutePath());

System.*out*.println("相对路径：" + file.getPath());

System.*out*.println("是否为绝对路径：" + file.isAbsolute());

System.*out*.println("是否为目录：" + file.isDirectory());

System.*out*.println("是否为文件：" + file.isFile());

System.*out*.println("是否为隐藏文件：" + file.isHidden());

System.*out*.println("最后修改时间：" + **new** Date(file.lastModified()));

System.*out*.println("文件长度：" + file.length());

输出结果：

\*\*\*\*\*\*文件属性测试\*\*\*\*\*\*

是否存在:true

文件名：hello.txt

上级目录：D:\

是否可读：true

是否可写：true

绝对路径：D:\hello.txt

相对路径：D:\hello.txt

是否为绝对路径：true

是否为目录：false

是否为文件：true

是否为隐藏文件：false

最后修改时间：Wed Nov 07 10:22:07 CST 2018

文件长度：20

## 13、文件的创建、删除、移动

**（1）文件的创建**

**public** **static** **void** main(String[] args) **throws** IOException {

File file = **new** File("D:\\hfx.txt");

**if** (!file.exists()) {

file.createNewFile();

}

File.*createTempFile*("tmp", ".log", **new** File("D:\\"));

}

（2）文件的删除

File file = **new** File("D:\\hfx.txt");

**if** (file.exists()) {

file.delete();

}

（3）文件的移动

**public** **static** **void** main(String[] args) **throws** IOException {

File file = **new** File("D:\\hfx.txt");

**if** (!file.exists()) {

file.createNewFile();

}

File dest = **new** File("C:\\a.txt");

file.renameTo(dest);

}

## 14、文件夹的创建、删除、移动

（1）创建文件夹

**public** **static** **void** main(String[] args) {

File file = **new** File("D:\\test\\soft");

// mkdir()方法早D:\\test目录下不存在时将发生异常

file.mkdir();

// mkdir()方法早D:\\test目录下不存在时将发生异常

file.mkdirs();

}

（2）删除文件夹

**public** **static** **void** main(String[] args) {

File file = **new** File("D:\\test");

*deleteFile*(file);

}

**public** **static** **void** deleteFile(File file) {

**if** (file.exists()) {

**if** (file.isFile()) {

file.delete();

} **else** **if** (file.isDirectory()) {

File files[] = file.listFiles();

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

System.*out*.println(files[i]);

*deleteFile*(files[i]);

}

}

file.delete();

} **else** {

System.*out*.println("无可删除的" + "\n");

}

}

}

（3）遍历目录中的所以文件

**public** **static** **void** main(String[] args) {

File file = **new** File("D:\\test");

*fileList*(file);

}

**public** **static** **void** fileList(File file) {

**if** (file.isFile()) {

System.*out*.println("文件------->" + file.getName());

}

**else** **if** (file.isDirectory()) {

System.*out*.println("文件夹目录是------>" + file.getName());

File files[] = file.listFiles();

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

// 递归用法

*fileList*(files[i]);

}

}

}

}

（4）查看一个文件夹的大小

**public** **static** **void** main(String[] args) {

File file = **new** File("D:\\test");

System.*out*.println(*getDirSize*(file));

}

**static** **long** *size* = 0;

**public** **static** Long getDirSize(File file) {

**if** (file.isFile()) {

*size* += file.length();

} **else** **if** (file.isDirectory()) {

File[] fi = file.listFiles();

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

*getDirSize*(fi[i]);

}

}

**return** *size*;

}

}

## 15、流

流是一个很形象的概念，大部分的应用程序都接收某种形式的数据输入，并产生跟某种形式的数据输出。当程序需要读取数据的时候，就会开启一个通向数据源的流，这个数据源可以是文件，内存或者是网络连接。与读取过程想死，当程序员需要写入数据的时候，就会开启一个通向目的地的流，在程序读取和洗耳液数据的过程中数据就像流动一样。

Java把不同的输入和输出源抽象表述为流。

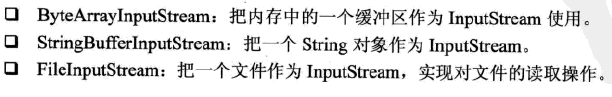
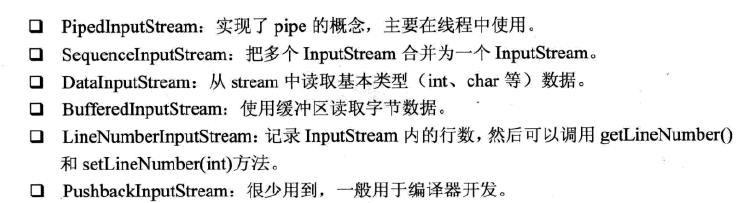
按照流向不同，可以分为输入流和输出流

按照处理单位不同，可以分为字节流和字符流。

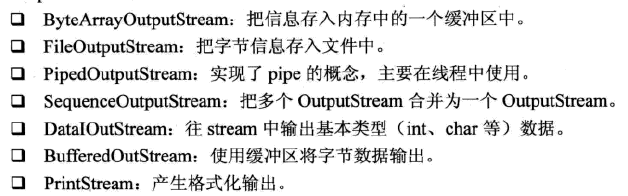
（1）字节流

字节流在数据读取和写入时以字节为单位，包含InputStream和OutputStream两个基础类。

inputScream用于按字节读取数据，子类有:

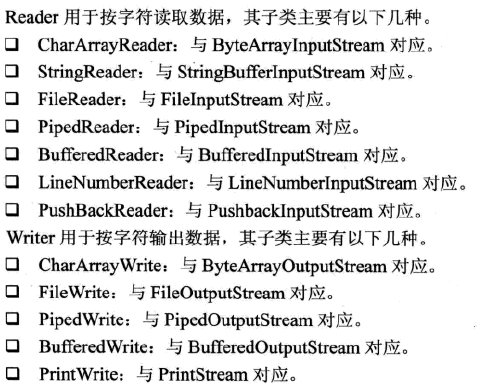


outputstream用于按字节输出数据，子类有：



（2）字符流

包含Reader和Writer两个基础类



## 16、事务

比如银行存钱，你给小明转账1000元，首先银行会在你的账户扣去1000元，之后再小明的账户增加1000元，相当于对数据库进行了两次操作，而事务就是为了保持数据库的一致性，所以应该这两个步骤都完成了之后再提交，这个时候我们再daoimpl层，也就是最后调用数据库mapper层，写上这么一句话：

@Transactional(propagation = Propagation.*REQUIRED*, isolation = Isolation.*READ\_COMMITTED*, timeout = 3, rollbackFor = Exception.**class**)

## 17、/u编码格式转换成为文字

**package** com.thunisoft;

/\*\* ChineseCharacterTransform.java: ----- 2016-9-20 下午3:49:51 wangzhongyuan \*/

**public** **class** ChineseCharacterTransform {

**public** **static** **void** main(String[] args) {

String str = "U字符";

String Ustr = *to\_U\_Str*(str);

System.*out*.println(Ustr); // 转化为U字符

Ustr = "\u662f\u5426\u53ea\u540c\u6b65\u5355\u4eba\u5355\u7f6a\u5f00\u5173";

System.*out*.println(*UStr\_2\_Str*(Ustr)); // 转化为表示的文本

}

/\* 将字符串转化为\\u形式的字符串，如： "U字符" -> "\\u55\\u5b57\\u7b26" ;U字符为字符串中每个字符的16进制信息 \*/

**public** **static** String to\_U\_Str(String str) {

String tmp = "";

**for** (**char** C : str.toCharArray())

// 获取所有字符

tmp += "\\u" + Integer.*toHexString*(C); // 将每个字符的的值，转化为16进制字符串

**return** tmp;

}

/\* 将U字符转化为其表示的字符串, 如： "\\u55\\u5b57\\u7b26" -> "U字符" ;按\\u分割，依次转化为对应字符 \*/

**public** **static** String UStr\_2\_Str(String Ustr0) {

String Ustr = Ustr0;

**int** S = 0, E = 0;

String C = "", Value = "";

**while** (Ustr.contains("\\u")) {

S = Ustr.indexOf("\\u") + "\\u".length();

E = Ustr.indexOf("\\u", S);

**if** (E == -1)

E = Ustr.length();

**if** (E > S) {

C = Ustr.substring(S, E);

**if** (C.length() > 4)

C = C.substring(0, 4);

Value = (**char**) Integer.*parseInt*(C, 16) + "";

Ustr = Ustr.replace("\\u" + C, Value);

}

}

**return** Ustr;

}

}

## 18、servlet

配置访问的路径：

（1）直接在servlet类的类名上面配置，访问时候：

http://localhost/FirstJavaEE/HelloServlet

@WebServlet("/HelloServlet")配置servlet路径

（2）在web.xml中配置：

<servlet>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>com.learn.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

<url-pattern>/servlet/HelloServlet</url-pattern>

</servlet-mapping>

此时的访问路径为

http://localhost:8080/FirstJavaEE/servlet/HelloServlet

## 19、获取当前系统的物理内存

**package** com.thunisoft.monitor.os;

**import** java.lang.management.ManagementFactory;

**import** com.sun.management.OperatingSystemMXBean;

/\*\*

\* 監控工具類

\* **@author** Administrator

\*

\*/

**public** **class** Test {

**public** **static** **void** main(String[] args) {

// 虚拟机级内存情况查询

**long** vmFree = 0;

**long** vmUse = 0;

**long** vmTotal = 0;

**long** vmMax = 0;

**int** byteToMb = 1024 \* 1024;

Runtime rt = Runtime.*getRuntime*();

vmTotal = rt.totalMemory() / byteToMb;

vmFree = rt.freeMemory() / byteToMb;

vmMax = rt.maxMemory() / byteToMb;

vmUse = vmTotal - vmFree;

System.*out*.println("JVM内存已用的空间为：" + vmUse + " MB");

System.*out*.println("JVM内存的空闲空间为：" + vmFree + " MB");

System.*out*.println("JVM总内存空间为：" + vmTotal + " MB");

System.*out*.println("JVM总内存空间为：" + vmMax + " MB");

System.*out*.println("======================================");

// 操作系统级内存情况查询

OperatingSystemMXBean osmxb = (OperatingSystemMXBean) ManagementFactory

.*getOperatingSystemMXBean*();

String os = System.*getProperty*("os.name");

**long** physicalFree = osmxb.getFreePhysicalMemorySize() / byteToMb;

**long** physicalTotal = osmxb.getTotalPhysicalMemorySize() / byteToMb;

**long** physicalUse = physicalTotal - physicalFree;

System.*out*.println("操作系统的版本：" + os);

System.*out*.println("操作系统物理内存已用的空间为：" + physicalFree + " MB");

System.*out*.println("操作系统物理内存的空闲空间为：" + physicalUse + " MB");

System.*out*.println("操作系统总物理内存：" + physicalTotal + " MB");

// 获得线程总数

ThreadGroup parentThread;

**int** totalThread = 0;

**for** (parentThread = Thread.*currentThread*().getThreadGroup(); parentThread

.getParent() != **null**; parentThread = parentThread.getParent()) {

totalThread = parentThread.activeCount();

}

System.*out*.println("获得线程总数:" + totalThread);

**int** kb = 1024;

OperatingSystemMXBean osmxb1 = (OperatingSystemMXBean) ManagementFactory

.*getOperatingSystemMXBean*();

// 操作系统

String osName = System.*getProperty*("os.name");

// 总的物理内存

**long** totalMemorySize = osmxb1.getTotalPhysicalMemorySize() / kb;

// 剩余的物理内存

**long** freePhysicalMemorySize = osmxb1.getFreePhysicalMemorySize() / kb;

// 已使用的物理内存

**long** usedMemory = (osmxb1.getTotalPhysicalMemorySize() - osmxb1

.getFreePhysicalMemorySize()) / kb;

System.*out*.println("==========================" + osmxb1);

}

}

## 20、组合

**package** 组合;

**class** Card {

**public** **void** spend() {

System.*out*.println("400元");

}

}

**class** Person {

**private** Card card;

**public** Person(Card card) {

**this**.card = card;

}

**public** **void** shopping() {

card.spend();

}

}

**class** Chinese **extends** Person {

**public** Chinese(Card card) {

**super**(card);

}

**public** **void** shopping() {

System.*out*.println("讨价还价");

**super**.shopping();

}

}

**public** **class** Demo1 {

**public** **static** **void** main(String[] args) {

Chinese ch = **new** Chinese(**new** Card());

ch.shopping();

}

}

## 21、利用反射解耦

package interfaceStudy;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.lang.reflect.Constructor;

interface Database{

int[] getDatebase();

}

interface Generator{

int[] generator();

}

interface DataHandler{

int[] handlerData();

}

**// 甲君写的代码，数据库层**

class MySql implements Database {

public int[] getDatebase() {

int[] arr = { 11, 33, 55, 212, 23, 21, 87, 2111 };

return arr;

}

}

**// 甲1君写的代码，数据库层**

class Oracle implements Database {

public int[] getDatebase() {

int[] arr = { 99, 999, 111, 222, 9999, 344 };

return arr;

}

}

**// 乙君写的代码，数据产生器**

class OneGenerator implements Generator {

private Database database;

public OneGenerator(Database database) {

this.database = database;

}

public int[] generator() {

int data[] = database.getDatebase();

for (int i = 0; i < data.length - 1; i++) {

for (int j = data.length - 1; j > i; j--) {

if (data[j] < data[j - 1]) {

int temp = data[j];

data[j] = data[j - 1];

data[j - 1] = temp;

}

}

}

return data;

}

}

**// 乙2君写的代码，数据产生器**

class TwoGenerator implements Generator {

private Database database;

public TwoGenerator(Database database) {

this.database = database;

}

public int[] generator() {

int data[] = database.getDatebase();

for (int i = 0; i < data.length - 1; i++) {

int pos = i;

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

if (data[j] > data[pos]) {

pos = j;

}

}

int temp = data[pos];

data[pos] = data[i];

data[i] = temp;

}

return data;

}

}

**//丙君写的代码，数据处理器**

class OneDataHandler implements DataHandler {

private Generator generator;

public OneDataHandler(Generator generator) {

this.generator = generator;

}

public int[] handlerData() {

int data[] = generator.generator();

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

data[i]++;

}

return data;

}

**}**

**//丙2君写的代码，数据处理器**

class TwoDataHandler implements DataHandler {

private Generator generator;

public TwoDataHandler(Generator generator) {

this.generator = generator;

}

public int[] handlerData() {

int data[] = generator.generator();

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

data[i]--;

}

return data;

}

}

public class Demo4 {

public static void main(String[] args) throws Exception {

// Database database = new DataBaseImpl();

// Generator generator = new OneGenerator(database);

// DataHandler datahandler = new OneDataHandler(generator);

// int data[] = datahandler.handlerData();

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

// System.out.print(data[i] + " ");

// }

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

**// 1.通过反射获得数据库对象**

System.out.println("请输出你的数据库：");

String databasrClassName = br.readLine();

**// 1-1根据类名反射出类**

Class databaseClazz = Class.forName(databasrClassName);

**// 1-2根据类创造出类的实例对象**

Database database = (Database) databaseClazz.newInstance();

**// 2.通过反射获得数据产生器**

System.out.println("请输出你的数据产生器：");

String generatorClassName = br.readLine();

**// 2-1根据类名反射出类**

Class generatorClazz = Class.forName(generatorClassName);

**// 2-2获得该类指定的构造器**

Constructor generatorConstructor = generatorClazz

.getConstructor(Database.class);

**// 2-3根据构造器创造出类的实例对象**

Generator generator = (Generator) generatorConstructor

.newInstance(database);

**// 3.通过反射获得数据处理器**

System.out.println("请输出你的数据处理层：");

String dataHandlerClassName = br.readLine();

**// 3-1根据类名反射出类**

Class dataHandlerClazz = Class.forName(dataHandlerClassName);

**// 3-2获得该类指定的构造器**

Constructor dataHandlerConstructor = dataHandlerClazz

.getConstructor(Generator.class);

**// 3-3根据构造器创造出类的实例对象**

DataHandler dataHandler = (DataHandler) dataHandlerConstructor

.newInstance(generator);

int data[] = dataHandler.handlerData();

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

System.out.print(data[i] + " ");

}

}

}

请输出你的数据库：

interfaceStudy.Oracle

请输出你的数据产生器：

interfaceStudy.TwoGenerator

请输出你的数据处理层：

interfaceStudy.TwoDataHandler

9998 998 343 221 110 98

## 22、可变参数

**package** canChangeParam;

**public** **class** CanChangeParam {

**public** **static** **void** main(String[] args) {

*run*(1, 122, 23123);

}

**public** **static** **void** run(**int**... num) {

System.*out*.println(num.length);

}

}

只有传入的是int类型的值就可以

