

● java中有7个位运算(&、|、^、~、>>、<<和>>>)

✓ 分别是 按位与&、按位或|、按位异或^、按位取反~ ,它们的运算规则是:

按位与&	:	两位全为1, 结果为1, 否则为0
按位或	:	两位有一个为1, 结果为1, 否则为0
按位异或^	:	两位一个为0,一个为1, 结果为1, 否则为0
按位取反~	:	0->1, 1->0

位运算符

● 还有3个位运算符 >>、<< 和 >>>, 运算规则:

1. 算术右移 >>: 低位溢出,符号位不变,并用符号位补溢出的高位
2. 算术左移 <<: 符号位不变,低位补0
3. >>> 逻辑右移也叫无符号右移,运算规则是: 低位溢出, 高位补 0
4. 特别说明: 没有 <<< 符号

*原码, 反码, 补码

- 1.对于有符号的而言:二进制的最高位是符号位 :0 表示正 ,1 表示 负
- 2.正数的原码 ,反码,补码都一样
3. 负数的反码=原码符号位不变 ,其他位取反 (0 -> 1 , 1 ->0)
4. 负数的补码=反 码 + 1, 负 数 的 反 码 = 补 码 - 1
5. 0 的反码,补码都是 0
6. java 没有无符号数 ,java 中所有数都有符号
- 7.在计算机运算的时候都是以补码的方式运算的
- 8.当我们看运算结果时要看原码

**位运算符 运算时化作二进制数 (4 个字节 8 位) 内容看练习

>>算数右移: 低位溢出符号位不变, 并用符号位补溢出的高位 本质是 $a \gg b \text{ } a/2^b$

<<算数左移: 符号位不变, 低位补 0 本质是 $a \ll b \text{ } a * 2^b$

>>>逻辑右移(无符号右移):低位溢出, 高位补 0

If 语句直接练习

单支 if

双分支 if else

多分支 if ... else if ...else if ... else

嵌套

```
// 接收字符方法: 获得字符串的第一个字符  
char gender = myScanner.next().charAt(0);
```

Switch

switch分支结构

● 基本语法

```
switch(表达式){  
    case 常量1: //当...  
        语句块1;  
        break;  
    case 常量2;  
    I 语句块2;  
        break;  
    ...  
    case 常量n;  
        语句块n;  
        break;  
    default:  
        default语句块;  
        break;
```

1. switch 关键字, 表示switch分支
2. 表达式 对应一个值
3. case 常量1 :当表达式的值等于常量1, 就执行 语句块1
4. break : 表示退出switch
5. 如果和 case 常量1 匹配, 就执行语句块1, 如果没有匹配, 就继续匹配 case 常量2
6. 如果一个都没有匹配上, 执行default

注意:

细节 1: 表达式数据类型, 要和 case 后的数据类型一致, 或者可以自动转成可以比较的类型

细节 2: case 后面的常量不能重复

```
char c = 'b';  
switch(c){  
    case 'a':  
        System.out.println("ok1");  
        break;  
    case 98: // 细节1: 表达式数据类型, 要和case后的数据类型一致, 或者可以自动转成可以比较的类型  
        System.out.println("ok2");  
        break;  
    case 'b': //细节2: case后面的常量不能重复  
        System.out.println("ok3");  
        break;
```

```
C:\Users\Administrator\Desktop\baoyan\lyc\第二周\视频中练习文件>javac SwitchDetail.java  
SwitchDetail.java:12: 错误: case 标签重复  
    case 'b': //细节2: case后面的常量不能重复
```

1 个错误

Switch 穿透现象

```
case 3:  
case 4:  
case 5:  
    System.out.println("春季");  
    break;  
case 6:  
case 7:  
case 8:  
    System.out.println("夏季");  
    break;
```

当多个常量同时判断为一种情况时可以进行穿透操作, 方便。

注意: Switch 中表达式的返回值 必须是 byte short int Char String enum(枚举) 类

型

Switch 和 if 比较

- switch和if的比较
- 1. 如果判断的具体数值不多, 而且符合byte、short、int、char、enum[枚举], String这6种类型。虽然两个语句都可以使用, 建议使用switch语句。
- 2. 其他情况: 对区间判断, 对结果为boolean类型判断, 使用if, if的使用范围更广

For 循环

for循环控制

- 基本语法

```
for (循环变量初始化; 循环条件; 循环变量迭代) {  
    循环操作(可以多条语句);  
}
```

- 老韩说明

1. for 关键字, 表示循环控制
2. for有四要素: (1)循环变量初始化(2)循环条件(3)循环操作(4)循环变量迭代
3. 循环操作, 这里可以有多条语句, 也就是我们要循环执行的代码
4. 如果 循环操作(语句) 只有一条语句, 可以省略 {}, 建议不要省略

For 循环注意事项

1) 循环条件时返回一个布尔值的表达式

2) for(;循环判断条件;)种初始化和变量迭代可以写其他地方但“;”不能省略(可扩大变量作用范围)

举例: int i=0;

For(;i<10;){

System.out.println("aaa");

i++;//这一定要加上

}

3) 循环初始值可以有多条初始化语句, 但要求类型一样中间用逗号隔开, 循环变量迭代也可以有多条变量迭代语句, 中间用逗号隔开

While 用法

while循环控制

- 基本语法

```
循环变量初始化;  
while(循环条件){  
    循环体(语句);  
    循环变量迭代;  
}
```

- 说明

- 1) while 循环也有四要素
- 2) 只是四要素放的位置, 不一样.

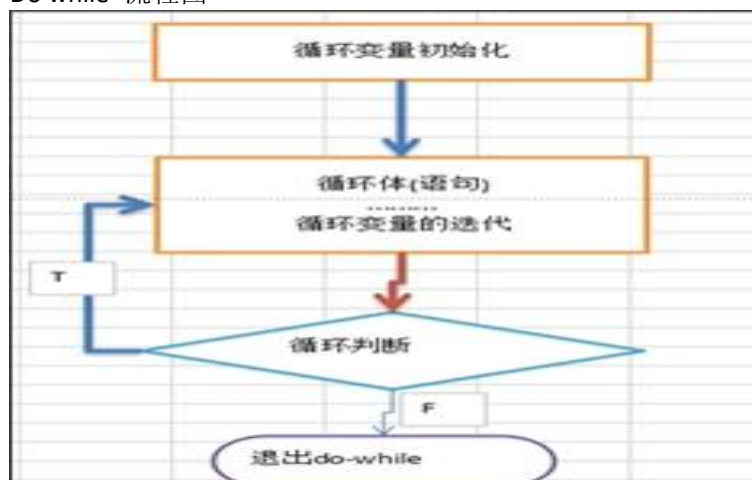
具体看练习程序

Do while 相比 while 在显示菜单时用

do..while循环控制

- 基本语法
循环变量初始化;
do{
 循环体(语句);
 循环变量迭代;
}while(循环条件);
- 说明:
 1. do while 是关键字
 1. 也有循环四要素, 只是位置不一样
 2. 先执行, 再判断, 也就是说, 一定会执行一次
 3. 最后 有一个 分号 ;
 4. while 和 do..while 区别举例: 要账

Do while 流程图



多重循环

多重循环控制(难点! 重点!)

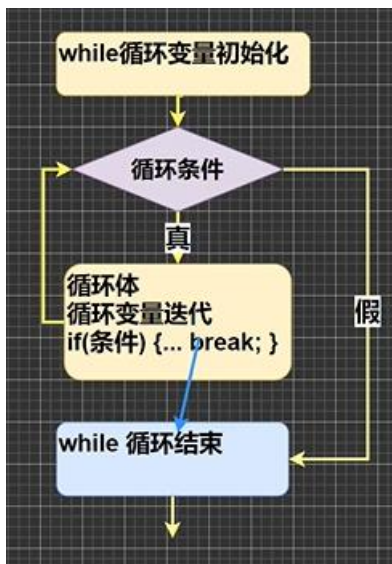
- 介绍
 1. 将一个循环放在另一个循环体内, 就形成了嵌套循环。其中, for ,while ,do...while均可以作为外层循环和内层循环。【建议一般使用两层, 最多不要超过3层, 否则, 代码的可读性很差】
 2. 实质上, 嵌套循环就是把内层循环当成外层循环的循环体。当只有内层循环的循环条件为 false时, 才会完全跳出内层循环, 才可结束外层的当次循环, 开始下一轮的循环[听不懂, 走案例]。
 3. 设外层循环次数为m次, 内层为n次, 则内层循环体实际上需要执行m*n次。

Break 使用

跳转控制语句-break

- 基本介绍:
break语句用于终止某个语句块的执行, 一般使用在switch或者循环[for , while , do-while]中
- 基本语法:

```
{  
    .....  
    break;  
    .....  
}
```
- 以while使用break为例,画出示意图



当 if 条件为真 执行 break 跳转到循环结束这一步

Break 提前结束循环 并未退出程序

跳转控制语句 continue

跳转控制语句-continue

● 基本介绍:

- 1) continue语句用于结束本次循环，继续执行下一次循环。
- 2) continue语句出现在多层嵌套的循环语句体中时，可以通过标签指明要跳过的是哪一层循环，这个和前面的标签的使用的规则一样。

● 基本语法:

```

{
    .....
    continue;
    .....
}

```

● 以while使用continue为例,画出示意图

其中 continue 默认跳转到的是最近得循环

可以指定跳转到哪个循环 就是在 continue 后面加上 标签号

```

// 编写一个main方法
public static void main(String[] args) {

    label1:
    for(int j = 0; j < 2; j++){
        label2:
        for(int i = 0; i < 10; i++){
            if(i == 2){
                // 看看分别输出什么值，并分析
                // continue ; // 等价于 continue label2
                // continue label2; // 等价 continue;
                continue label1;
            }
            System.out.println("i = " + i); // 输出2次[0,1,3,4,5,6,7,8,9]
        }
    }
}

```

Handwritten notes and diagrams:

- Next to the code, there are handwritten notes: $j=0$, $i=0$, and $i=1$.
- On the right, there is a red box containing the handwritten text: $j=0$, $i=0$, and $i=0+2$.
- A red arrow points from the `continue label1;` line in the code to the `label1:` line, illustrating the jump.

跳转控制语句 return

return使用在方法，表示跳出所在的方法，在讲解方法的时候，会详细的介绍，这里我们简单的提一下。注意：如果 return 写在 main方法，退出程序..

数组细节

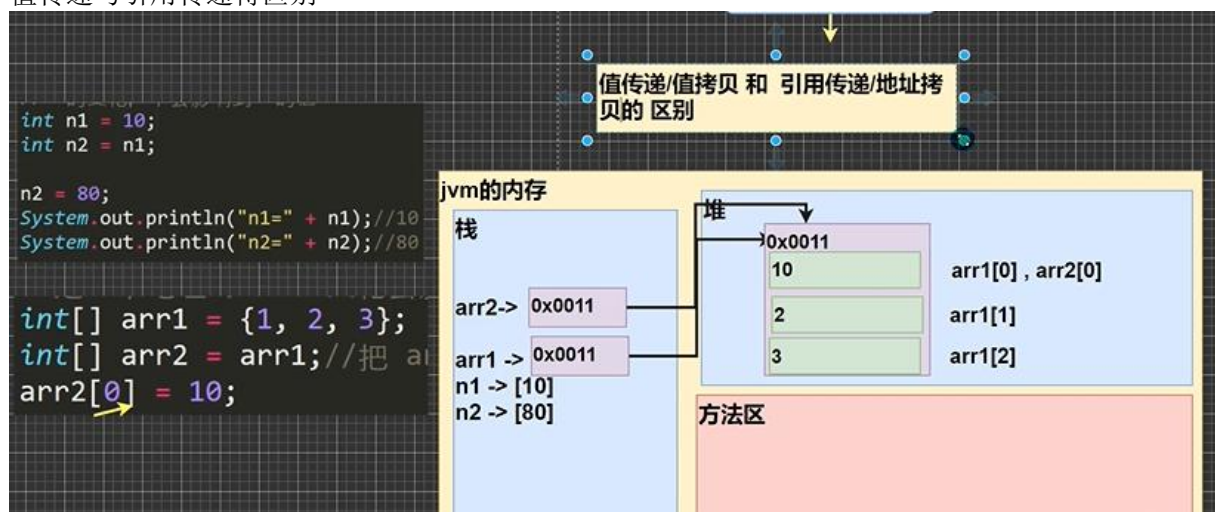
数组使用注意事项和细节

ArrayDetail.java

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

数组赋值一般情况下是引用传递，赋的值是地址

值传递与引用传递得区别



数组反转 2 逆序赋值法

空间变换讲解如图


```

// 逆序赋值
int[] arr = {11, 22, 33, 44, 55, 66};
// 使用逆序赋值方式
// 老韩思路
// 1. 先创建一个新的数组 arr2, 大小 arr.length
// 2. 逆序遍历 arr, 将 每个元素拷贝到 arr2 的元素中 (顺序拷贝)
// 3. 建议增加一个循环变量 j -> 0 -> 5
int[] arr2 = new int[arr.length];
// 逆序遍历 arr
for(int i = arr.length - 1, j = 0; i >= 0; i--, j++) {
    arr2[j] = arr[i];
}
// 4. 当for循环结束, arr2就是一个逆序的数组 {66, 55, 44, 33, 22, 11}
// 5. 让 arr 指向 arr2数据空间, 此时 arr原来的数据空间就没有变量引用
// 会被当做垃圾, 销毁
arr = arr2;
System.out.println("====arr的元素情况====");

```

排序的介绍

排序是将多个数据, 依指定的顺序进行排列的过程。

排序的分类:

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

冒泡排序对于五个数得数组来说

分析冒泡排序

数组 [24,69,80,57,13]

第1轮排序: 目标把最大数放在最后

第1次比较[24,69,80,57,13]

第2次比较[24,69,80,57,13]

第3次比较[24,69,57,80,13]

第4次比较[24,69,57,13,80]

第2轮排序: 目标把第二大数放在倒数第二位置

第1次比较[24,69,57,13,80]

第2次比较[24,57,69,13,80]

第3次比较[24,57,13,69,80]

第3轮排序: 目标把第3大数放在倒数第3位置

第1次比较[24,57,13,69,80]

第2次比较[24,57,13,69,80]

顺序查找就是从头到尾遍历

总结冒泡排序特点

1. 我们一共有5个元素
2. 一共进行了 4轮排序, 可以看成是外层循环
3. 每1轮排序可以确定一个数的位置, 比如第1轮排序确定最大数, 第2轮排序, 确定第2大的数位置, 依次类推
4. 当进行比较时, 如果前面的数大于后面的数, 就交换
5. 每轮比较在减少 4->3->2->1