

顺序控制

程序从上到下逐行执行，中间没有任何判断和跳转。

分支控制

① if-else

1) 单分支 if

```
✓ 基本语法
if(条件表达式){
    执行代码块; (可以有多条语句.)
}
```

说明：当条件表达式为true 时，就会执行 {} 的代码。如果为false ,就不执行。
特别说明，如果 {} 中只有一条语句，则可以不用 {}, 韩老师建议写上 {}

✓ 案例说明

请大家看个案例[If01.java]:

编写一个程序,可以输入人的年龄,如果该同志的年龄大于18岁,则输出 "你年龄大于18,要对自己的行为负责,送入监狱"

2) 双分支 if-else

```
✓ 基本语法
if(条件表达式) {
    执行代码块1;
}
else {
    执行代码块2;
}
```

说明：当条件表达式成立，即执行代码块1，否则执行代码块2.如果执行代码块 只有一条语句，则 {} 可以省略，否则，不能省略

✓ 案例演示

请大家看个案例[If02.java]:

编写一个程序,可以输入人的年龄,如果该同志的年龄大于18岁,则输出 "你年龄大于18,要对自己的行为负责, 送入监狱"。否则 ,输出"你的年龄不大这次放过你了."

3) 多分支 if-else if -....-else

✓ 基本语法

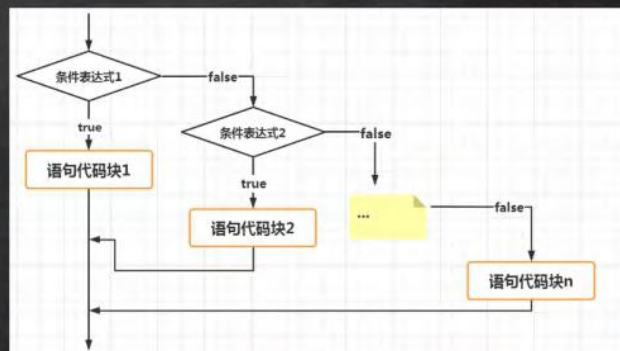
```
if (条件表达式1) {  
    执行代码块1;  
}  
else if (条件表达式2) {  
    执行代码块2;  
}  
.....  
else {  
    执行代码块n;  
}
```

特别说明: (1) 多分支 可以没有 else , 如果所有的条件表达式都不成立, 则一个执行入口都没有 (2) 如果有 else , 如果所有的条件表达式都不成立, 则默认执行 else 代码块

多分支流程图

说明:

1. 当条件表达式1成立时, 即执行代码块1,
2. 如果表达式1不成立, 才去判断表达式2是否成立,
3. 如果表达式2成立, 就执行代码块2
4. 以此类推, 如果所有的表达式都不成立
5. 则执行 else 的代码块, 注意, 只能有一个执行入口。



4) 嵌套分支

在一个分支结构中又完整的嵌套了另一个完整的分支结构, 里面的分支的结构称为内层分支外面的分支结构称为外层分支。

② switch 分支结构

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

➤ 老韩解读switch

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

注意事项:

- 1) 表达式的数据类型应该与 case 后面的常量类型保持一致或者是可以自动

转成、相互比较的类型。

2) 表达式中的返回类型必须是 `short`, `byte`, `int`, `char`, `enum`, `String`

3) `case` 后面的值必须是常量不能是变量

4) `break` 语句用来在执行完之后跳出 `switch` 语句块, 如果没有 `break`, 程序将顺序执行到 `switch` 的末尾, 除非遇到 `break`。

③ `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

循环控制

① `for` 循环控制

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

注意细节:

1) 循环条件是返回一个布尔值的表达式

2) `for(;;循环判断条件;)` 中的初始化和变量迭代可以写到其它地方, 但是两边的分号不能省略。

3) 循环初始值可以有多条初始化语句, 但要求类型一样, 并且中间用逗号隔开, 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。

② `while` 循环控制

```
循环变量初始化;  
while (循环条件) {  
  
    循环体(语句);  
    循环变量迭代;  
}  
● 老韩说明  
1) while 循环也有四要素  
2) 只是四要素放的位置和for不一样.
```

注意细节:

1) 循环条件是返回一个布尔值的表达式

2) while 循环是先判断再执行语句

③do..while 循环控制

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

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，一定会至少执行一次

3. 最后 有一个 分号 ；

使用细节:

1) 循环条件是返回一个布尔值的表达式

2) do..while 循环是先执行，再判断， 因此它至少执行一次

④多重循环控制

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。

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。

跳转控制语句

① break

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

```
{ .....  
    break;  
    .....  
}
```

② continue

1) continue 语句用于结束本次循环， 继续执行下一次循环。

2) continue 语句出现在多层嵌套的循环语句体中时，可以通过标签指明要跳过的是哪一层循环， 这个和前面的标签的使用的规则一样。

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

③ return

return 使用在方法，表示跳出所在的方法