第三章 操作符、流程控制

1 操作符

之前的课程中我们已经学过了字面值常量和变量，它们都可以用来表示一个数据，本节课我们要学习如何使用运算符对常量和变量进行运算。

在具体讲解之前，我们需要概括下运算符和表达式的概念：

操作符：对字面值常量或变量进行操作的**符号**，也称`运算符`。

表达式：用**操作符**把字面值常量或变量连接起来的式子（符合Java语法），就称之为表达式。

例如：

```
int a = 10;
int b = 20;
int c = a + b;  // + 属于是运算符（操作符）
                // a + b 叫做表达式
c = c + 30;     // c + 30 也叫做表达式
```

1.1 运算符分类

操作符可以分为下面几类：

- 算术运算符
- 赋值运算符
- 比较|关系|条件 运算符
- 逻辑运算符
- 位运算符
- 三目运算符

- 算术运算符

+	-	*	/	%

- 赋值操作符

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

- 比较操作符

>	>=	<	<=	instanceof
==	!=			

- 移位操作符

>>	<<	>>>	

- 位操作符

&	|	^	~

- 逻辑操作符

&&	||			

- 三目运算符

?	:			

1.2 算术运算符

操作符	作用	例子
+	数字之间使用+,表示俩个值相加	int a = 1+1;
-	两个数字相减	int a = 1-1;
*	两个数字相乘	int a = 1*1;
/	两个数字相除	int a = 1/1;
%	两个数字取余	int a = 5%2;

+、-、*、/ 这几个操作符很常用，较简单，我们看下述案例即可：

```
package com.briup.chap03;
```

```java
public class Test012_Arithmetic {
    public static void main(String[] args) {
        //定义两个int类型的变量
        int a = 6;
        int b = 4;

        System.out.println(a + b);  //10
        System.out.println(a - b);  //2
        System.out.println(a * b);  //24
        System.out.println(a / b);  //?
        System.out.println(a % b);  //2

        //整数相除结果是整数，要想得到小数，必须有浮点数的参与
        double c = a;
        System.out.println(c / 4);
    }
}
```

% 求余运算符：

具体案例如下，请思考输出结果分别是什么：

```java
package com.briup.chap03;

public class Test012_Mod {
    public static void main(String[] args) {
        int n = 13 % 5;
        System.out.println("n: " + n);  // ?

        n = -13 % 5;
        System.out.println("n: " + n);  // ?

        n = 13 % -5;
        System.out.println("n: " + n);  // ?

        n = -13 % -5;
        System.out.println("n: " + n);  // ?
    }
```

```
}
```

其输出结果为：3、-3、3、-3

结论：求余运算，结果符号只跟左操作数的符号有关

字符串相加：

+除了可以作为加法运算符，也可以作为字符串连接符。

字符串 + 其他任意类型数据，得到的结果都是字符串。

案例：

```
package com.briup.chap03;

public class Test012_Connect {
    public static void main(String[] args) {
        //+: 字符串连接符

        //运算：从左往右 逐步进行
        //                "helloa"+1 == "helloa1"
        System.out.println("hello"+'a'+1);

        //        char+int==int 98+"hello" ==> "98hello"
        System.out.println('a'+1+"hello");
        //                  "5+5=55"
        System.out.println("5+5="+5+5);
        //               10+"" ==> "10=5+5"
        System.out.println(5+5+"=5+5");
    }
}
```

自增自减：

++为自增，--为自减，两者使用方式类似，下面重点介绍++。

自增或自减有两种使用方式，分别如下：

- 变量名++;
- ++变量名;

如果单独使用，目的是对变量进行自增或自减，上述2种方式作用相同：

```
package com.briup.chap03;

public class Test012_Increment {
    public static void main(String[] args) {
        int a = 10;
        a++;
        System.out.println("a: " + a);  // a: 11

        a = 10;
        ++a;
        System.out.println("a: " + a);  // a: 11
    }
}
```

如果作为表达式使用，则两者有明显区别：

```
package com.briup.chap03;

public class Test012_Increment2 {
    public static void main(String[] args) {
        int a = 10;
        // 先保留a的值10，然后将a的值自增，最后将a之前的值10赋值给b
        int b = a++;
        System.out.println("a: " + a);  // a: 11
        System.out.println("b: " + b);  // b: 10

        a = 10;
        // 先对a自增，然后获取到a的值11，赋值给b
        b = ++a;
        System.out.println("a: " + a);  // a: 11
        System.out.println("b: " + b);  // b: 11
```

```
    }
}
```

`--自减`的使用方式与自增类似。

面试题1：

```
public static void main(String[] args) {
    int x = 4;
    int y = (x--) + (--x) + (x * 10);

    System.out.println("x: " + x);  // x: ?
    System.out.println("y: " + y);  // y: ?
}
```

面试题2：

```
public static void main(String[] args) {
    byte b = 10;
    b++;
    b = b + 1;
    //问哪句会报错,为什么？
}
```

1.3 赋值运算符

操作符	作用	例子
=	最基础的赋值操作符，=号右边的值，赋给=号左边变量	int a = 1; int x = 0;
*=	一个变量和另一个数据相乘，并把结果再赋值给这个变量	int a = 1; a*=2; //a = a*2;
/=	一个变量和另一个数据相除，并把结果再赋值给这个变量	int a = 2; a/=2; //a = a/2;
%=	一个变量和另一个数据相余，并把结果再赋值给这个变量	int a = 5; a%=2; //a = a%2;
+=	一个变量和另一个数据相加，并把结果再赋值给这个变量	int a = 5; a+=2; //a = a+2;
-=	一个变量和另一个数据相减，并把结果再赋值给这个变量	int a = 5; a-=2; //a = a-2;

除此之外，还有`<<=` `>>=` `>>>=` `&=` `^=` `|=` ，和上面的含义是一样的，了解过<< >> >>> & ^ |这几个二进制操作后，那么加上=号的意思也就知道了。

```
package com.briup.chap03;

public class Test013_Basic {
    public static void main(String[] args) {
        int a = 10;
        a += 5;     //a = a + 5;
        System.out.println("a: " + a);  //15

        a -= 10;    //a = a - 10;
        System.out.println("a: " + a);  //5

        //面试题
        short s = 10;
```

```
        //s = s + 5;    //error
        s += 5;     // s = (short)(s+5);
        System.out.println("s: " + s);
    }
}
```

注意：**+=、-=、*=、/= 等扩展的赋值运算符，隐含了强制类型转换！**

1.4 比较运算符

操作符	作用	例子
>	比较是否大于	1 > 0
>=	比较是否大于等于	1 >= 0
<	比较是否小于	1 < 2
<=	比较是否小于等于	1 <= 2
instanceof	判断对象是否属于指定类型	stu instanceof Student
==	判断左右2个操作数是否相等，结果为boolean值	boolean f = (a == b);
!=	判断左右2个操作数是否不相等，结果为boolean值	boolean f = (a != b);

程序获取从键盘录入的整数值（补充内容）：

`Scanner`类是Java提供好的API，可以接收用户从键盘中的输入。具体使用步骤如下：

1. 导包

   ```
   import java.util.Scanner;
   ```

2. 创建Scanner对象

   `Scanner sc = new Scanner(System.in);`

3. 从键盘获取值

   `int a = sc.nextInt();`

案例展示：

从键盘录入2个整形数，比较其大小，并输出结果。

```
package com.briup.chap03;

//1.导包
import java.util.Scanner;

public class Test014_Compare {
    public static void main(String[] args) {
        //2.创建Scanner对象
        Scanner sc = new Scanner(System.in);
        System.out.println("请录入两个整形数: ");

        //3.从键盘获取值
        int a = sc.nextInt();
        int b = sc.nextInt();

        System.out.println("a: " + a);
        System.out.println("b: " + b);

        //比较运算符 表达式：结果boolean
        boolean flag = a > b;
        System.out.println("flag: " + flag);
        if(flag) {
            System.out.println("a > b");
        }

        if(a < b) {
```

```
            System.out.println("a < b");
        }

        if(a == b) {
            System.out.println("a == b");
        }
    }
}
```

操作符 `instanceof` 暂时先不用管，面向对象部分会学习。

1.5 逻辑运算符

操作符	作用	例子
&&	与运算，带逻辑短路	a && b，a、b可以是表达式，两者都为true结果为true，否则为false
&	与运算	a & b，a、b都为true结果为true，否则为false
||	或运算，带逻辑短路	a || b，a、b有一个为true结果为true，都为false结果为false
|	或运算	a | b，a、b有一个为true结果为true，都为false结果为false
!	非，取反的意思	!a，如果a为true则结果为false，a为false结果为true
^	异或，相同为假,不同为真	a ^ b，如果a和b中一个为false，另一个为true则结果为true，否则结果为false

案例1：

&& || 基本功能测试

```
package com.briup.chap03;

//1.导包
import java.util.Scanner;

public class Test015_Basic {
    public static void main(String[] args) {
        //2.实例化对象
        Scanner sc = new Scanner(System.in);
        System.out.println("input two num: ");

        //3.获取键盘录入的值
        int v = sc.nextInt();
        int n = sc.nextInt();

        // 判断是否同时能被3 5 整除
        // 注意：使用 & 效果也一样
        if((v % 3 == 0) && (v % 5 == 0)) {
            System.out.println("yes");
        }else {
            System.out.println("no");
        }

        //判断是否 是3的整数倍 或 是5的整数倍
        // 注意：使用 | 效果也一样
        if((n % 3 == 0) || (n % 5 == 0)) {
            System.out.println("yes");
        }else {
            System.out.println("no");
        }
    }
}
```

案例2：

&& || 逻辑短路功能测试

```java
package com.briup.chap03;

public class Test015_Logic {
    public static void main(String[] args) {
        int x = 1;
        int y = 5;

        // 注意：用 && 与 & 测试的结果不同，因为 && 具有逻辑短路功能
        // 逻辑短路：如果计算完第一个表达式，得到的结果能够决定整个表达式结果的话，则不再运算第二个表达式
        boolean f = (x > 4) && (y++ > 5);   //不会运算 y++ > 5
        //boolean f = (x > 4) & (y++ > 5);
        System.out.println("f: " + f); //false
        System.out.println("y: " + y); //5

        // ! 逻辑非
        if(!f) {
            System.out.println("true");
        }else {
            System.out.println("false");
        }
    }
}
```

案例3：

！逻辑非功能测试

```java
package com.briup.chap03;

public class Test015_LogicNot {
    public static void main(String[] args) {
        int x = 1;
        int y = 5;

        System.out.println(!(x > y));   //!false
        System.out.println(!!(x > y));  //!!false
        System.out.println(!!!(x > y)); //!!!false
    }
}
```

1.6 移位运算符

如果要进行移位操作，`则需要先获取操作数的二进制形式（补码）`，然后按位进行操作。

操作符	作用	例子
>>	算术右移位运算，也叫做【带】符号的右移运算	8 >> 1
<<	左移位运算	8 << 1
>>>	逻辑右移位运算，也叫做【不带】符号的右移运算	8 >>> 1

右移：

>> 低位抛弃，高位补符号位的值

>>> 低位抛弃，高位补0

```java
package com.briup.chap03;

public class Test016_RightShift {
    public static void main(String[] args) {
        // 0 0(23) 0000 1010
```

```
        int a = 10;
        System.out.println("a: " + a);
        // 0 0(23) 0000 1010
        // 00 0(23) 000 0101 [0] ==> 5
        int b = a >> 1;
        System.out.println("a >> 1: " + b);

        // 000 0(23) 0000 10 ==> 2
        b = a >> 2;
        System.out.println("a >> 2: " + b);

        // 负数移位操作
        a = -10;
        // 获取-10的二进制补码：
        // 原 1 0(23) 0000 1010
        // 反 1 1(23) 1111 0101
        // 补 1 1(23) 1111 0110
        // 运算：111 1(23) 1111 01 [10]
        b = a >> 2; // 高位补1 负
        // 结果推导：补 1 1(23) 1111 1101
        // - 1
        // 反 1 1(23) 1111 1100
        // 保留符号位，其他位取反
        // 原 1 0(23) 0000 0011
        // 结果：-3
        System.out.println("-10 >> 2: " + b);

        b = a >>> 2;
        // -10补：  1 1(23) 1111 0110
        // >>> 2位，高位补0
        // 00 1 1(23) 1111 01
        // 结果：很大的一个正整数 1073741821
        System.out.println("-10 >>> 2: " + b);
    }
}
```

左移：

<< 高位抛弃，低位补0

```
package com.briup.chap03;

public class Test016_LeftShift {
    //左移: 高位抛弃 低位补0
    public static void main(String[] args) {
        int a = 10;
        System.out.println("a: " + a);      // 10
        int b = a << 1;
        System.out.println("a << 1: " + b); // 20
        b = a << 2;
        System.out.println("a << 2: " + b); // 40
        b = a << 3;
        System.out.println("a << 3: " + b); // 80

        //结果：每左移1位，等同 == (值 * 2)

        a = -10;
        b = a << 2;
        // -10补码：1 1(23) 1111 0110
        //  左移2位，结果：
        //           1(22) 1111 011000
        //      即：  1 1(23) 1101 1000
        //  -1得反码：  1 1(23) 1101 0111
        //  保留符号位，其他为取反，得原码：
        //      原：  1 0(23) 0010 1000  ==>  -40
        System.out.println("a << 4: " + b);

        // 如果左移位数太多，超出了数值表示范围，如何处理？
        //  数值 << n 等同  数值 << (n%当前数值所占比特位数)
        b = a << 33;
        System.out.println("a << 33: " + b);        //20
        System.out.println("a << (33%32): " + b);   //20
    }
}
```

小结:

- 每左移1位，等同于 `值*2`
- 数值左移n位(移动后将超出数值最高位)，等同 `数值 << (n%当前值所属类型所占比特位)`

1.7 位运算符

操作符	作用	例子
&	与运算	1&1=1，1&0=0， 0&1=0， 0&0=0
|	或运算	1|1=1，1|0=1，0|1=1，0|0=0
^	异或运算	1^1=0， 0^0=0， 1^0=1， 0^1=1，相同为0，不同为1
~	取反运算	0 -> 1 ， 1 -> 0

案例1:

位运算符基本使用

```
package com.briup.chap03;

//1.导包
import java.util.Scanner;

public class Test017_BitBasic {
    //从键盘录入两个数  进行 位运算
    public static void main(String[] args) {
        //2.实例化对象
        Scanner sc = new Scanner(System.in);
        System.out.println("input two num: ");

        // 10   0 0(23) 0000 1010
        int n1 = sc.nextInt();
        //  3   0 0(23) 0000 0011
```

```
        int n2 = sc.nextInt();

        //&: 1&1 == 1   0&? == 0
        //  0 0(23) 0000 1010
        //& 0 0(23) 0000 0011
        //  0 0(23) 0000 0010  ==> 2
        int value = n1 & n2;
        System.out.println("n1 & n2: " + value);

        //  0 0(23) 0000 1010
        //| 0 0(23) 0000 0011
        //  0 0(23) 0000 1011  ==> 11
        value = n1 | n2;
        System.out.println("n1 | n2: " + value);

        //  0 0(23) 0000 1010
        //^ 0 0(23) 0000 0011
        //  0 0(23) 0000 1001  ==> 9
        value = n1 ^ n2;
        System.out.println("n1 ^ n2: " + value);

        //  0000 1010
        byte num = 10;
        //~ 0000 1010
        //  1111 0101 ==> -?

        // 原码 <-> 反码 <-> 补码
        // 补: 1111 0101
        // 反：补码-1
        //    1111 0100
        // 原: 反码保留符号位，其他位取反
        //    1000 1011
        // 结果：  -11
        value = ~num;
        System.out.println("~num: " + value);
    }
}
```

案例2：

^特殊用法，不使用中间变量的前提下，可以交换2个变量的值。

```
package com.briup.chap03;

public class Test017_XorOperator {
    //定义方法，交换两个整形数的值 【先用，方法具体讲解会在本章最后一个内容中讲解】
    //修饰符 返回值类型 函数名(形式参数列表) {
    //  方法体语句;
    //}
    public static void swap(int a,int b) {
        System.out.println("交换前：a: " + a + ",b: " + b);
        int t = a;
        a = b;
        b = t;
        System.out.println("交换后：a: " + a + ",b: " + b);
    }

    //不借助第三个变量，完成2个数的交换
    public static void swap2(int a,int b) {
        System.out.println("交换前：a: " + a + ",b: " + b);
        // 一个数 异或 另一个数 2次，结果是自己
        a = a ^ b;
        b = a ^ b; // a^b^b ==> a
        a = a ^ b; // a^b^a ==> b;
        System.out.println("交换后：a: " + a + ",b: " + b);
    }

    //测试代码
    public static void main(String[] args) {
        int x = 10;
        int y = 20;

        swap(x,y);
```

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

        swap2(30,40);
    }
}
```

扩展练习：

使用位操作符，对变量a（10）进行某一位置0或置1操作。

```
public class Test017_Extend {
    public static void main(String[] args) {
        int a = 10;

        /*
            题目：将a的第5位 置为1
            解题思路分析：
                只要得到 0 0(23) 0001 0000，然后和a进行或运算即可得到
            如何得到 0 0(23) 0001 0000 ？
                1<<(5-1) 运算即可得到：0 0(23) 0001 0000
        */
        int b = a | (1 << (5-1));
        System.out.println("b: " + b);

        //调用方法实现
        b = setBit1(10, 5);
        System.out.println("b: " + b); //26

        b = setBit0(10, 4);
        System.out.println("b: " + b); //2
    }

    //将value的第n位置为1，结果返回
    public static int setBit1(int value, int n) {
        int r = value | (1 << (n - 1));
        return r;
    }
```

```
    //将value的第n位置为0，结果返回
    //                          10      4
    //  0 0(23) 0000 1010
    //& 1 1(23) 1111 0111

    //  0 0(23) 0000 1000
    //  0 0(23) 0000 0001
    public static int setBit0(int value, int n) {
        int r = ~(1<<(n-1)) & value;
        return r;
    }
}
```

1.8 三目运算符

三目运算符也称条件运算符。

格式：

(关系表达式) ? 表达式1 : 表达式2;

关系表达式成立，返回表达式1，否则返回表达式2。

案例展示：

```
import java.util.Scanner;

//从键盘录入2个整数，比较大小
public class Test018_Three {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("input two num: ");

        int a = sc.nextInt();
        int b = sc.nextInt();
```

```
        int max = (a > b) ? a : b;
        int min = (a < b) ? a : b;

        System.out.println("max: " + max);
        System.out.println("min: " + min);
    }
}
```

2 流程控制

在一个程序执行的过程中，各条语句的**执行顺序**对程序的结果是有直接影响的。所以，我们必须清楚每条语句的执行流程。而且，很多时候要通过控制语句的执行顺序来实现我们想要的功能。

程序中，需要执行的代码，其结构主要分为以下三种

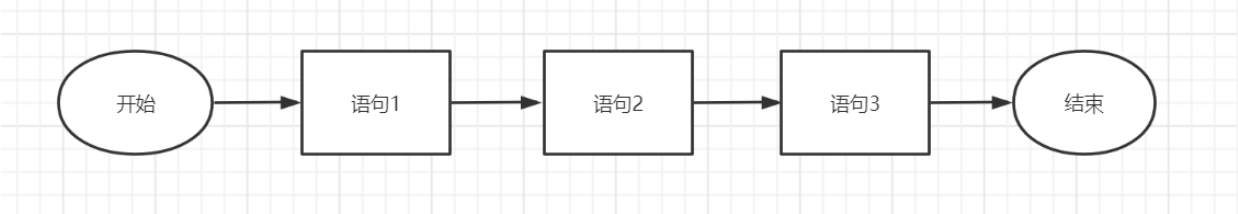
- 顺序结构
- 分支结构

  if语句、switch分支语句
- 循环结构

  for循环、while循环、do while循环

2.1 顺序结构

顺序结构就是最基本的流程控制，只要将代码从上到下，按照顺序依次编写即可，大多数代码都是这样的结构，如图：



案例：

```
package com.briup.chap03;

public class Test021_Statement {
    //顺序结构语句
    public static void main(String[] args) {
        System.out.println("开始");
        System.out.println("语句1");
        System.out.println("语句2");
        System.out.println("语句3");
        System.out.println("结束");
    }
}
```

2.2 if判断

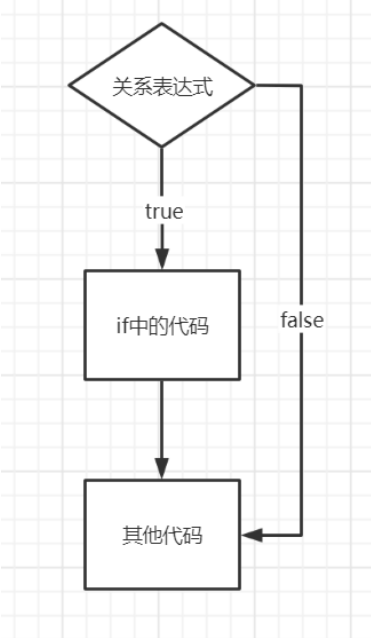
> **语法1：**

```
if（关系表达式）{
    语句体;
}

//后续代码
```

执行流程：

1. 计算 **关系表达式** 的值
2. 如果关系表达式的值为true，就执行语句体
3. 如果关系表达式的值为false，则不执行语句体
4. 继续执行if代码块后面的其他代码

执行流程图：



案例：

```
import java.util.Scanner;

//从键盘录入2个数，比较并输出较大值
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int a = sc.nextInt();
    int b = sc.nextInt();

    int max = a;
    //条件满足则执行，不满足则跳过往下执行
    if(a < b) {
        max = b;
    }

    System.out.println("max: " + max);
    System.out.println("");
}
```

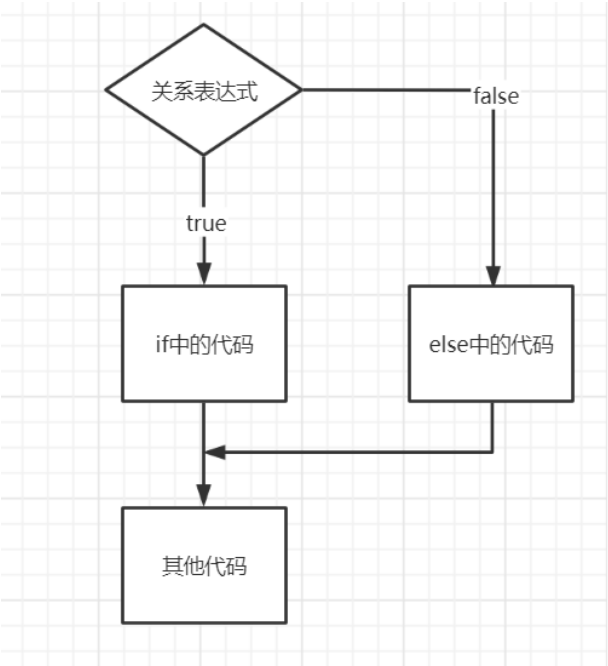
> **语法2：**

```
if (关系表达式) {
    语句体1;
} else {
    语句体2;
}
//后续代码
```

执行流程：

1. 计算 **关系表达式** 的值
2. 如果关系表达式的值为true，就执行语句体1
3. 如果关系表达式的值为false，就执行语句体2
4. 继续执行if代码块后面的其他代码

执行流程图：



案例：

```
package com.briup.chap03;

import java.util.Scanner;

public class Test022_LeapYear {
    //从键盘录入年份值，判断并输出该年份是否为闰年
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int year = sc.nextInt();

        /* 闰年判断
            (4年闰 && 百年不闰) || (400年闰)
        */
        if((year%4 == 0 && year%100 != 0)
           || (year % 400 == 0)) {
            System.out.println("是闰年");
        }else {
            System.out.println("不是闰年");
        }
    }
}
```

注意事项：

如果有俩个if语句，那么它们俩是相互独立切互不影响的俩个结构。

```
int a = 10;
if(a % 2 == 0) {
    System.out.println("变量a的值为偶数");
}

if(a % 2 == 1) {
    System.out.println("变量a的值为奇数");
}
```

第一个if条件无论是true还是false，第二个if条都会继续判断，这个逻辑和if-else是不同的。

> **语法3：**

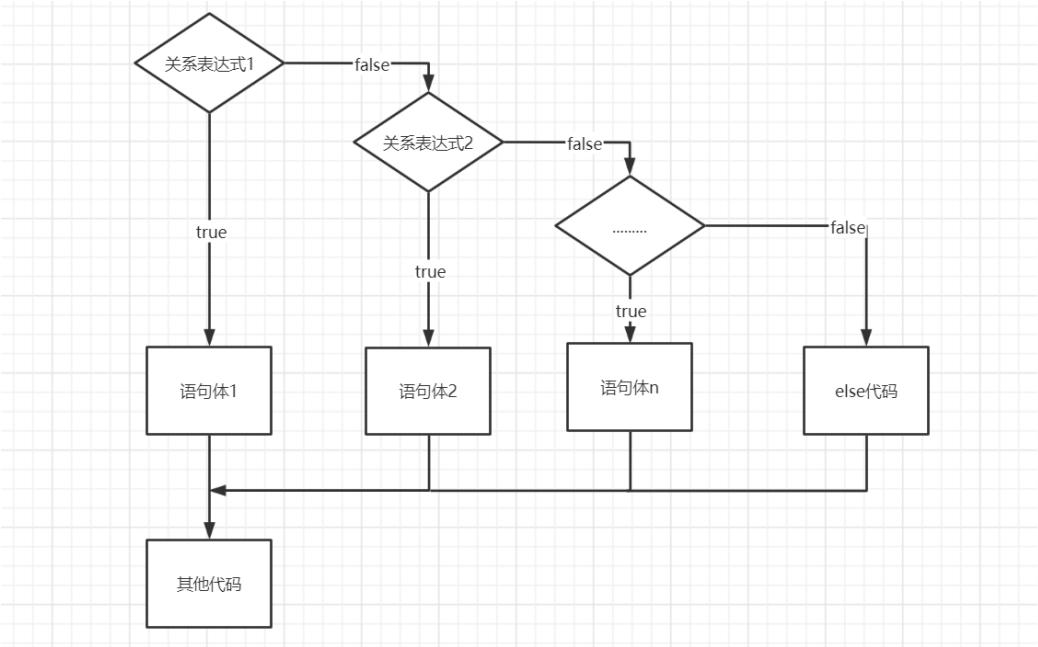
```
if (关系表达式1) {
    语句体1;
}
else if (关系表达式2) {
    语句体2;
} …
else {
    //else代码;
}

//后续代码
```

执行流程：

1. 首先计算 **关系表达式1** 的值
2. 如果表达式1的值为true，就执行语句体1；如果值为false就计算**关系表达式2**的值
3. 如果表达式2的值为true，就执行语句体2；如果值为false就计算**关系表达式3**的值

4. ....
5. 如果上面关系表达式结果都为false，就执行else代码
6. 如果中间有任何一个关系表达式为true，那么执行完对应的代码语句之后，整个if-elseif-else退出

执行流程图：



案例：

从键盘录入一个成绩值，判断其对应的等级【优、良、中、及格、不及格】

```
package com.briup.chap03;

import java.util.Scanner;

//从键盘录入一个成绩值，判断其对应的等级【优、良、中、及格、不及格】
public class Test022_ScoreGrade {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int score = sc.nextInt();
```

```
        // 110 65
        if (score > 100) {
            System.out.println("成绩有误,不能超过100");
        } else if (score >= 90) {
            System.out.println("优");
        } else if (score >= 80) {
            System.out.println("良");
        } else if (score >= 70) {
            System.out.println("中");
        } else if (score >= 60) {
            System.out.println("及格");
        } else if (score >= 0) {
            System.out.println("不及格");
        } else {
            System.out.println("成绩有误，不能小于0");
        }
        // 注意：最后的else语句也可以不写
    }
}
```

注意事项：

从上到下依次判断，有一个判断为true执行了代码，那么后续判断都不再执行，if语句结束；如果判断都为false，则执行else语句代码。

课堂练习：

从键盘录入一个hour值，如果录入的是8~12之间的值，那么就输出早上好，如果是12点~14点，则输出中午好，如果是14点~18点，则输出下午好，其他情况，输出晚上好。

```
package com.briup.chap03;

import java.util.Scanner;
```

```java
public class Test022_Hello {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("input a hour: ");
        int hour = sc.nextInt();

        String message;

        if (hour >= 8 && hour < 12) {
            message = "早上好";
        } else if (hour >= 12 && hour < 14) {
            message = "中午好";
        } else if (hour >= 14 && hour < 18) {
            message = "下午好";
        } else {
            message = "晚上好";
        }

        System.out.println(message);
    }
}
```

2.3 switch分支

switch语句也称为**分支语句**，其和if语句有点类似，都是用来判断值是否相等，但switch默认只支持byte、short、int、char这四种类型的比较，JDK8中也允许String类型的变量做对比。

语法：

```java
switch (表达式) {    //表达式可以为byte、short、int、char，JDK5加入枚举，JDK7加入String
    case 1:          //分支入口
        语句体1;
```

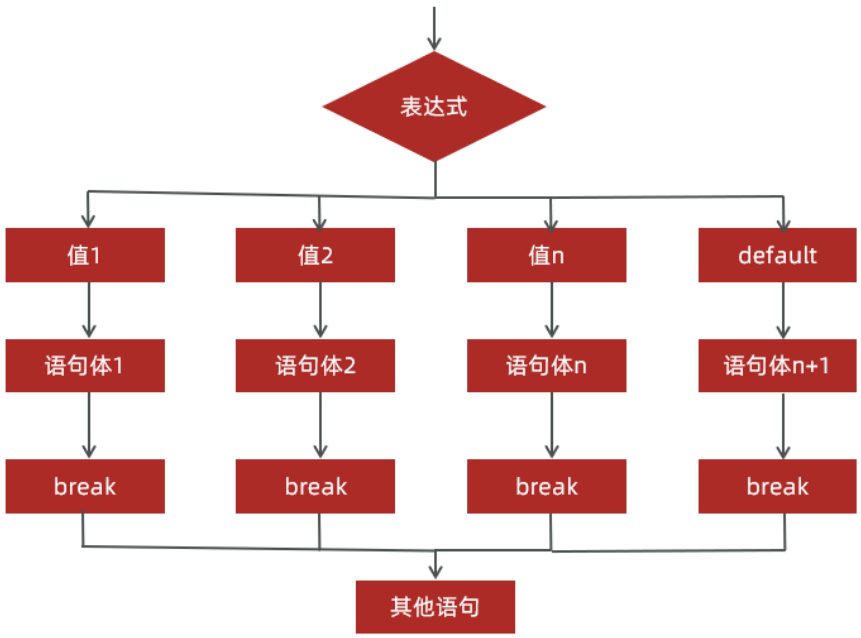
```
            break;
        case 2:          //分支入口
            语句体2;
            break;
        ...
        default:
            语句体n+1;
            break;
}

//后续语句
```

格式说明：

- 表达式：可以是byte类型,short类型,int类型,char类型

  JDK5之后可以是枚举类型，JDK7之后可以是String类型
- case：后面跟的是要和表达式进行比较的值
- break：表示中断，结束的意思。用来结束switch语句
- default：所有值都不匹配的时候，执行该处内容。和if语句的else相似

执行流程：



基础案例：

从键盘录入mode值，其值为0 1 2 3中任意一个，然后输出相应字符串

```
package com.briup.chap03;

import java.util.Scanner;

public class Test023_SwitchBasic {
    //从键盘录入mode值，其值为0 1 2 3中任意一个，然后输出相应字符串
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("input a mode(0 1 2 3): ");
        int mode = sc.nextInt();

        switch(mode){
            case 0:
                System.out.println("默认模式开启");
                //注意讲解break用法：结束switch语句
                //break;
            case 1:
                System.out.println("1模式开启");
                break;
            case 2:
                System.out.println("2模式开启");
                break;
            case 3:
                System.out.println("3模式开启");
                break;
            default:
                System.out.println("无效录入");
                //break;
        }
    }
}
```

案例2：

从键盘录入一个0到6之间的整数，然后输出这个数对应的星期几，例如0对应星期天，1对应星期一。

```
package com.briup.chap03;

import java.util.Scanner;

public class Test023_WeekDay {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("请录入整数 [0,6]: ");
        int day = sc.nextInt();
        String msg = null;

        switch(day){
            case 0:
                msg = "星期天";
                break;
            case 1:
                msg = "星期一";
                break;
            case 2:
                msg = "星期二";
                break;
            case 3:
                msg = "星期三";
                break;
            case 4:
                msg = "星期四";
                break;
            case 5:
                msg = "星期五";
                break;
            case 6:
                msg = "星期六";
                break;
```

```
            default:
                msg = "参数有误，参数day的值可以在[0,6]之间";
        }

        System.out.println(msg);
    }
}
```

扩展案例：

从键盘录入一个年份和月份，然后输出该月份的天数

```
package com.briup.chap03;

import java.util.Scanner;

public class Test023_Extend {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("请录入年份和月份： ");
        int year = sc.nextInt();
        int month = sc.nextInt();

        switch(month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                System.out.println("31天");
                break;
            case 4:
            case 6:
            case 9:
            case 11:
```

```
                System.out.println("30天");
                break;
            case 2:
                if((year % 400 == 0) || (year%4 ==0 && year%100
!= 0)) {
                    System.out.println("29天");
                }else {
                    System.out.println("28天");
                }
                break;
            default:
                System.out.println("录入月份有误!");
                break;
        }
    }
}
```

2.4 for循环

循环语句可以在满足循环条件的情况下，反复执行某一段代码，这段被重复执行的代码被称为循环体语句。

当反复执行这个循环体时，需要在合适的时候把循环判断条件修改为false，从而结束循环，否则循环将一直执行下去，形成死循环。

语法：

```
for (初始化语句1; 条件判断语句2; 条件控制语句4) {
    循环体语句3;
}
```

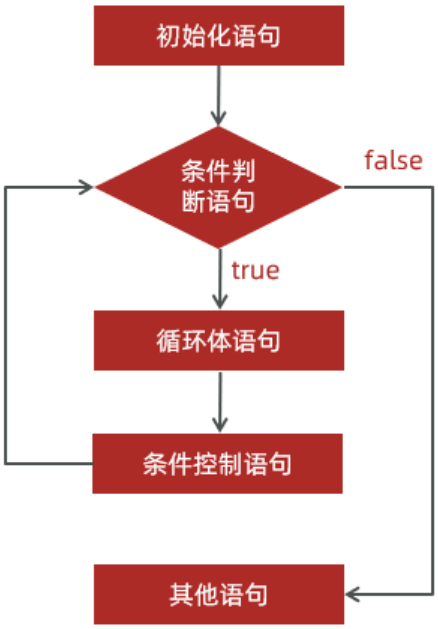
执行流程：

1. 执行初始化语句1
2. 执行条件判断语句2，看其结果是true还是false
    - 如果是false，循环结束
    - 如果是true，执行循环体语句3
3. 执行条件控制语句4
4. 回到步骤2继续执行

案例：

```
public static void main(String[] args) {
    for(int i = 0; i < 5; i++) {
        System.out.println("循环体语句，i: " + i);
    }

    System.out.println("out of for...");
}
```

执行流程：



案例1：

求1-5之间的数据和，并把求和结果在控制台输出

```
package com.briup.chap03;

public class Test024_ForSum {
    public static void main(String[] args) {
        // 求和的最终结果必须保存起来，需要定义一个变量，用于保存求和的结果，初始值为0
        int sum = 0;
        // 从1开始到5结束的数据，使用循环结构完成
        for (int i = 1; i <= 5; i++) {
            // 将反复进行的事情写入循环结构内部
            // 此处反复进行的事情是将数据 i 加到用于保存最终求和的变量 sum 中
            sum += i;
            /*
             * sum += i; sum = sum + i;
             第一次：sum = sum + i = 0 + 1 = 1;
             第二次：sum = sum + i = 1 + 2 = 3;
             第三次：sum = sum + i = 3 + 3 = 6;
             第四次：sum = sum + i = 6 + 4 = 10;
             第五次：sum = sum + i = 10 + 5 = 15;
             */
        }
        // 当循环执行完毕时，将最终数据打印出来
        System.out.println("1-5之间的数据和是：" + sum);
    }
}
```

案例2：

求1-100之间的偶数和

```
package com.briup.chap03;
```

```java
public class Test024_OSum {
    public static void main(String[] args) {
        // 求和的最终结果必须保存起来，需要定义一个变量，用于保存求和的结果，初始值为0
        int sum = 0;
        // 对1-100的数据求和与1-5的数据求和几乎完全一样，仅仅是结束条件不同
        for (int i = 1; i <= 100; i++) {
            // 对1-100的偶数求和，需要对求和操作添加限制条件，判断是否是偶数
            if (i % 2 == 0) {
                sum += i;
            }
        }
        // 当循环执行完毕时，将最终数据打印出来
        System.out.println("1-100之间的偶数和是：" + sum);
    }
}
```

思考，是否还有其他方式，可以实现1-100之间的偶数和

扩展案例：

输出所有的水仙花数。

提示：水仙花数是一个3位数，153就是：`153 == 1*1*1 + 5*5*5 + 3*3*3`

```java
package com.briup.chap03;

public class Test024_OutFlower {
    public static void main(String[] args) {
        int g, s, b;

        // 逐个值 判断是否为水仙花数
```

```
        for (int i = 100; i < 1000; i++) {
            // 1.拆分个、十、百位数
            g = i % 10;
            s = i / 10 % 10;
            b = i / 100;

            // 2.判断立方和 是否 等于自己
            if (i == g * g * g + s * s * s + b * b * b) {
                System.out.println("水仙花数: " + i);
            }
        }
    }
}
```

for循环其他形式

```
public static void main(String[] args) {
    //for循环常规写法
    for(int i = 0;i < 5; i++){
        System.out.println("hello world");
    }

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

    //初始化变量和改变变量的值，是可以写到其他地方的
    //这个最后的效果和上面的for循环是一样的
    int i = 0;
    for(;i < 5;){
        System.out.println("hello world");

        i++;
    }
}
```

死循环

```
//这是一个死循环代码,for的小括号中，只有俩个分号
for(;;) {
    System.out.println("hello world");
}

//for循环的大括号中，如果只有一句代码，那么可以把大括号省去不写
for(;;)
    System.out.println("hello world");
```

思考，一般情况下我们不会编写死循环代码，但是死循环是否有它专门的应用场景呢?

2.5 while循环

一般情况下，循环次数是确定的，我们会选择for循环，如果循环的次数无法确定，我们选择while循环实现功能。

语法:

```
初始化语句1;
while (条件判断语句2) {
    循环体语句3;

    条件控制语句4;
}
```

执行流程:

1. 执行初始化语句1
2. 执行条件判断语句2，看其结果是true还是false
    - 如果是false，循环结束

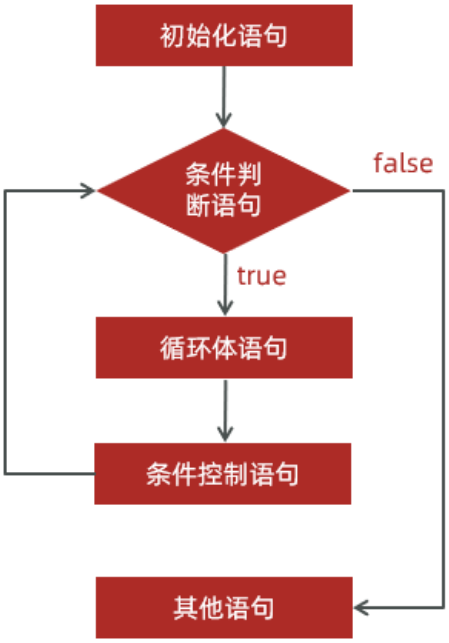
- 如果是true，执行循环体语句3

3. 执行条件控制语句4
4. 回到步骤2继续执行

范例：

```
public static void main(String[] args) {
    int i = 5;
    while(i > 0) {
        System.out.println("循环体语句，i: " + i);

        i--;
    }

    System.out.println("out of for...");
}
```

执行流程：



案例1：

求1-100之间的奇数和，并输出求和结果

```
package com.briup.chap03;

public class Test025_WhileJSum {
    public static void main(String[] args) {
        int sum = 0;

        int i = 1;
        while (i <= 100) {
            sum += i;

            i += 2; // 1,3,5,7...99,101
        }

        // 输出结果
        System.out.println("1-100之间的奇数和是：" + sum);
    }
}
```

案例2：

世界最高山峰珠穆朗玛峰(8844.43米==8844430毫米)，假如有一张足够大的纸，它的厚度是0.1毫米，那么折叠多少次，可以折成珠穆朗玛峰的高度。

```
package com.briup.chap03;

public class Test025_Altitude {
    public static void main(String[] args) {
        // 定义一个计数器，初始值为0
        int count = 0;
        // 定义纸张厚度
        double paper = 0.1;

        // 定义珠穆朗玛峰的高度
```

```
        int zf = 8844430;
        // 因为要反复折叠，所以要使用循环，但是不知道折叠多少次，这种情况下更适合使用while循环
        // 折叠的过程中当纸张厚度大于珠峰就停止了，因此继续执行的要求是纸张厚度小于珠峰高度
        while (paper <= zf) {
            // 循环的执行过程中每次纸张折叠，纸张的厚度要加倍
            paper *= 2;
            // 在循环中执行累加，对应折叠了多少次
            count++;
        }

        // 打印计数器的值
        System.out.println("需要折叠：" + count + " 次");
    }
}
```

扩展案例：

从键盘录入2个正整数，输出其最大公约数和最小公倍数。

```
package com.briup.chap03;

import java.util.Scanner;

public class Test025_Divisor {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        //1.键盘录入2个数
        System.out.println("input 2 num: ");
        int a = sc.nextInt();
        int b = sc.nextInt();

        //预处理
        if(a == b){
            System.out.println("最大最小：" + a);
```

```
            return;
        }

        //2.区分两个数大小
        int max = (a > b) ? a : b;
        int min = (a < b) ? a : b;
        if(a % b == 0){
            System.out.println("最大: " + min);
            System.out.println("最小: " + a*b/min);
            return;
        }

        //3.辗转相除法
        /*
                12   9   ==>   3
                {
                    a.用max % min 得到 yu
                        18 % 12 == 6
                    b.如果yu值不为0
                    则
                        max = min;
                        min = yu;
                        再重复上述操作
                    c.如果yu值为0
                        则 最大公约数为min
                        return 结束
                }
            */
        int yu = max % min;
        while(yu != 0) {
            max = min;
            min = yu;

            yu = max % min;
        }

        System.out.println("最大: " + min);
        System.out.println("最小: " + a * b / min);
```

```
    }
}
```

while死循环

```
while(true){
    //循环体代码
}
```

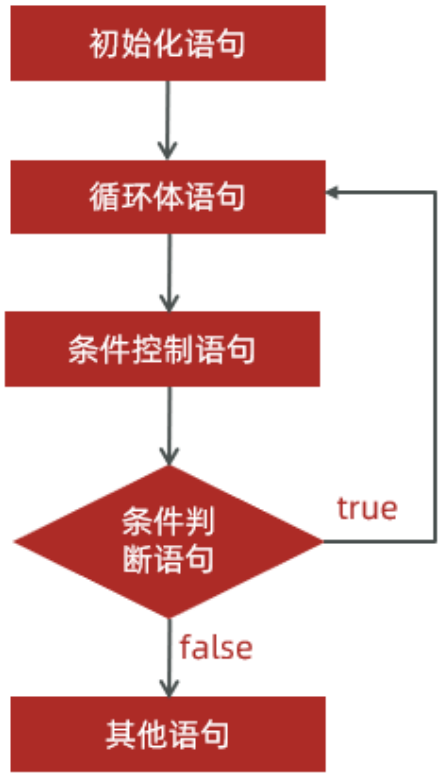
2.6 do-while

do-while也可以实现循环，但应用不多，具体如下。

格式：

```
初始化语句1;
 do {
    循环体语句3;

    条件控制语句4;
}while(条件判断语句2);
```

执行流程：



案例1：

```
public static void main(String[] args) {
    int i = 1;
    do {
        System.out.println("Hello World");

        i++;
    } while (i<=5);
}
```

案例2：

循环不断的生成[10,20]随机数，直到生成随机数为15的时候，结束循环，输出循环的次数。

```
package com.briup.chap03;

public class Test026_Random {
```

```
    public static void main(String[] args) {
        /*
         * 固定公式：生成[min,max]随机数
            (int)(Math.random() * (max-min+1)) + min;
         *
         * java.lang.Math, 其中random()方法返回值：[0,1)
         */

        int count = 0;
        int v;
        do {
            v = (int) (Math.random() * (20 - 10 + 1)) + 10;
            count++;
        } while (v != 15);

        System.out.println("循环次数： " + count);
    }
}
```

do-while死循环

```
do{
    //循环体语句
}while(true);
```

2.7 区别

三种循环语句的区别：

- for和while循环**先判断条件是否成立**，然后决定是否执行循环体（先判断后执行）
- do...while**先执行一次循环体**，然后判断条件是否成立，是否继续执行循环体（先执行后判断）

for和while的区别：

- 条件控制语句所控制的自增变量，因为**默认情况下**归属for循环的语法结构中，在for循环结束后，就不能再次被访问到了

```
for(int i=0;i<10i++){
    //...
}

//这里无法使用变量i
```

- 条件控制语句所控制的自增变量，对于while循环来说不归属其语法结构中，在while循环结束后，该变量还可以继续使用

```
int i = 0;
while(i<10){
    //...
    i++;
}

//这里可以继续使用变量i
```

死循环格式：

for(;;){}

while(true) {}

do{}while(true);

三种循环使用场景：

> - 明确循环次数，推荐使用for

> - 不明确循环次数，推荐使用while
> - do..while 很少使用

2.8 break

循环语句中遇到break关键字，循环直接结束。

案例：

for循环从1到10进行输出，当i的值为4或8时，跳出当前循环（循环整体结束）

```
package com.briup.chap03;

public class Test028_Break {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i == 4 || i == 8) {
                break;
            }

            System.out.println("i = " + i);
        }

        System.out.println("out of for ...");
    }
}
```

2.9 continue

循环语句中遇到continue关键字，**本次循环结束，进入到下一次循环。**

案例：

for循环从1到10进行输出，当i的值为4或8时，跳出本次循环进入下一次循环

```
package com.briup.chap03;

public class Test029_Continue {
    public static void main(String[] args) {
        for(int i = 1;i <= 10; i++){
            if(i == 4 || i == 8){
                continue;
            }

            System.out.println("i = "+i);
        }

        System.out.println("out of for ...");
    }
}
```

break continue综合案例：

小芳的妈妈每天给她2.5元钱，她都会存起来，但是，每当这一天是存钱的第5天或者5的倍数的话，她都会花去6元钱，请问，经过多少天，小芳才可以存到100元钱。

```
package com.briup.chap03;

public class Test029_SaveMoney {
    public static void main(String[] args) {
        // 小芳的妈妈每天给她2.5元钱
```

```
        double dayMoney = 2.5;

        // 她都会存起来，涉及到了求和思想，定义求和变量，初始化值为0
        double sumMoney = 0;

        // 存到100元钱
        int result = 100;

        // 定义一个统计变量，用来纪录存钱的天数，初始化值为1
        int dayCount = 1;

        // 因为不知道要多少天才能够存到100元，所以，这里我们采用死循环来实现，当存到100元的时候，通过break跳转语句让循环结束
        while (true) {
            // 存钱
            sumMoney += dayMoney;

            // 判断存的钱是否大于等于100了，如果是，就退出循环
            if (sumMoney >= result) {
                break;
            }

            // 每当这一天是存钱的第5天或者5的倍数的话，她都会花去6元钱
            if (dayCount % 5 == 0) {
                sumMoney -= 6;
                System.out.println("第" + dayCount + "天花了6元");
            }

            dayCount++;
        }

        // 输出存钱总天数
        System.out.println("共花了 " + dayCount + " 天存了100元");
    }
}
```

3.循环嵌套

在一个循环内部可以嵌套另一个或多个循环。

案例1：

输出以下内容，要求每次只能输出一个 `'*'`

```
*
**
***
****
*****
```

```
package com.briup.chap03;

public class Test03_Nest1 {
    public static void main(String[] args) {
        for(int i = 1; i <= 5; i++){
            for(int j = 0; j < i; j++){
                System.out.print("*");
            }

            System.out.println();
        }
    }
}
```

注意：

- println方法会自动换行
- print 方法不会自动换行

案例2：

输出以下内容，要求每次只能输出一个 `'*'`

```
    *
   ***
  *****
 *******
*********
```

```
package com.briup.chap03;

public class Test03_Nest2 {
    public static void main(String[] args) {
        //line表示要输出的行数
        int line = 5;

        //外层循环控制打印的行数
        for(int i = 1; i <= line; i++){
            //这个循环控制每行打印的空格
            for(int j = 0; j < line - i; j++){
                System.out.print(" ");
            }

            //这个循环控制每行打印的*
            for(int k = 0; k < (2 * i - 1); k++){
                System.out.print("*");
            }
            //当前行中的空格和*都打印完了，最后输出一个换行
            System.out.println();
        }
    }
}
```

案例3：

输出以下内容，要求每次只能输出一个 `'*'`

```
**********
 *********
  *******
   *****
    ***
     *
```

```
package com.briup.chap03;

public class Test03_Nest3 {
    public static void main(String[] args) {
        //line表示要输出的行数
        int line = 6;

        //外层循环控制打印的行数
        for(int i = 1; i <= line; i++){
            //这个循环控制每行打印的空格
            for(int j = 0; j < i - 1; j++){
                System.out.print(" ");
            }

            //这个循环控制每行打印的*
            for(int k = 0; k < (2*line-2*i+1); k++){
                System.out.print("*");
            }

            //当前行中的空格和*都打印完了，最后输出一个换行
            System.out.println();
        }
    }
}
```

4.label

代码中出现多层循环嵌套，label标签配合break关键字，可以使程序从内部循环中跳出。

案例描述：

> 思考：下面案例中break关键字能跳出外层循环吗？

```
public static void main(String[] args) {
    for(int i = 0; i < 3; i++){//外层循环
        for(int j = 0; j < 5; j++){//内层循环
            if(j == 2){
                break;
            }
        }
    }
}
```

> 注意，默认情况下，在嵌套循环中，break和continue只能对当前循环起作用。

如果想让break或continue针对某一个指定的循环起作用，那么可以使用`label标签`给循环起名字，然后`使用break或continue加上label标签名`即可。

例如，

```
package com.briup.chap03;

public class Test04_Label {
```

```
    public static void main(String[] args) {
        label1: for(int i = 0; i < 3; i++) {//外层循环
            label2: for(int j = 0; j < 5; j++) {//内层循环
                if(j == 2) {
                    //此时可直接跳出外层循环，然后往下执行
                    break label1;
                }
                System.out.println("i = " + i + ", j = " + j);
            }
            System.out.println("----------------------");
        }

        System.out.println("out of 外层循环!");
    }
}
```

5.Random

Random类，即`java.util.Random`，是Java提供好的API，它提供了产生随机数的功能，在这里可以先简单的使用下，后续的学习中会了解到更多详细细节。

基础案例：

产生10个随机数，范围[10-20]。

```
package com.briup.chap03;

// 第一步，导入包
import java.util.Random;

/*
    Random ： 作用是能够产生随机数
```

```
    使用步骤如下：
        1. 导包    : import java.util.Random;
            导包的动作必须出现在类定义的上面
        2. 创建对象 : Random r = new Random();
            上面这个格式里面，r 是变量名，可以变，其他的都不允许变
        3. 获取随机数 : int num = r.nextInt(n); 其中num的取值为[0,n)
            注意：获取[min,max]范围随机数的公式如下
            int number = r.nextInt(max-min+1) + min;
*/
public class Test05_Random {
    public static void main(String[] args){
        //第二步，创建对象
        Random r = new Random();

        for(int i = 1; i <= 10; i++){
            //第三步，调用nextInt(max-min+1) + min; 产生随机数
            int num = r.nextInt(20-10+1) + 10;      // 10-20
            System.out.println(num);
        }
    }
}
```

综合案例：

完成猜数字游戏。程序自动生成一个1-100之间的数字，使用程序实现猜出这个数字是多少。

当猜错的时候根据不同情况给出相应的提示：

1. 如果猜的数字比真实数字大，提示你猜的数据大了
2. 如果猜的数字比真实数字小，提示你猜的数据小了
3. 如果猜的数字与真实数字相等，提示恭喜你猜中了

```
package com.briup.chap03;

import java.util.Scanner;
```

```java
import java.util.Random;

public class Test05_GuessNumber {
    /*
        1. 准备Random和Scanner对象, 分别用于产生随机数和键盘录入
        2. 使用Random产生一个1-100之间的数, 作为要猜的数
        3. 键盘录入用户猜的的数据
        4. 使用录入的数据(用户猜的数据)和随机数(要猜的数据)进行比较, 并给出提示

        5. 以上内容需要多次进行, 但无法预估用户输入几次可以猜测正确, 使用while(true)死循环包裹
        6. 猜对之后, break结束.
    */
    public static void main(String[] args){
        // 1. 准备Random和Scanner对象, 分别用于产生随机数和键盘录入
        Random r = new Random();
        Scanner sc = new Scanner(System.in);

        // 2. 使用Random产生一个1-100之间的数, 作为要猜的数
        int randomNum = r.nextInt(100) + 1;

        // 5. 以上内容需要多次进行, 但无法预估用户输入几次可以猜测正确, 使用while(true)死循环包裹
        while(true){
            // 3. 键盘录入用户猜的的数据
            System.out.println("请输入您猜的数据:");
            int num = sc.nextInt();
            // 4. 使用录入的数据(用户猜的数据)和随机数(要猜的数据)进行比较, 并给出提示
            if(num > randomNum){
                System.out.println("猜大了");
            }else if(num < randomNum){
                System.out.println("猜小了");
            }else{
                // 6. 猜对之后, break结束.
                System.out.println("恭喜,猜中了");
                break;
```

```
            }
        }

        System.out.println("感谢您的使用，游戏结束！");
    }
}
```

6 方法

方法(method)：就是完成特定功能的代码块！

通过方法的定义和调用，可大大提高代码的复用性与可读性！

关于方法，我们其实并不陌生，从开始学Java到现在每天都会使用main方法，具体如下：

```
public class 测试类名 {
    //main方法是一个特殊的方法，其是程序的入口
    public static void main(String[] args) {
        System.out.println("hello method");
    }
}
```

6.1 定义

在类中定义方法的固定格式如下：

```
修饰符 返回值类型 方法名(形式参数列表) {
    方法体语句;
}
```

例如：

```java
public static int getSum(int a, int b) {
    int sum = a + b;
    return sum;
}
```

方法定义细节：

- 修饰符：当下固定为`public static`，后续再补充其他形式

- 返回值类型：根据该方法的具体功能而定
  - 如果方法的作用仅做输出，不需要返回数据，则为 void
  - 如果方法需要计算一个结果并返回，则为结果类型，比如方法要求2个int数的和，则为 int

- 方法名
  - 见名知意，方法名的描述性要好
  - 驼峰命名法，如果只有一个单词全小写

    例：`display()` `sum()`
  - 如果多个单词构成，则第一个单词全小写，后续单词首字母大写

    例：`sayHello()` `getMax()` `findStudentById()`

- 形式参数列表
  - 根据该方法的具体功能而定
  - 具体案例如下

    计算2个int数的和，则`int getSum(int a, int b);`

    计算3个int数的平均数，则`double getAvg(int x, int y, int z);`

遍历一个数组，则 `void outArray(int[] array);`

- 方法体语句：方法的具体实现，要根据方法功能书写代码

案例：

```
package com.briup.chap03;

public class Test06_Function {
    // 获取两个int数较大值
    public static int getMax(int a, int b) {
        if(a > b)
            return a;

        return b;
    }

    // 求三个int数的平均值
    public static double getAvg(int x, int y, int z) {
        double sum = x + y + z;
        double avg = sum / 3;

        return avg;
    }
}
```

注意事项：

- 带参方法定义时，参数中的**数据类型**与**变量名**都不能缺少，缺少任意一个程序将报错
- 带参方法定义时，多个参数之间使用**逗号**(,)分隔

6.2 调用

方法调用格式：

`方法名(实际参数列表);`

```
//在上述Test06_Function类中添加测试方法
public static void main(String[] args) {
    // 10,20是实际参数
    int max = getMax(10, 20);
    System.out.println("max: " + max);

    int x = 10;
    int y = 20;
    // a,b是实际参数
    max = getMax(x, y);
    System.out.println("max: " + max);

    // ...
}
```

注意事项：

- 方法必须先定义，再调用
- 实际参数列表可以是常量，也可以是变量，也可以是表达式
- 实际参数类型要匹配形式参数类型（要么完全相同，要么能自动隐式类型转换）
- main方法是入口方法，一个程序中唯一，其可以调用其他普通方法
- 其他方法不能调用main方法，普通方法可以相互调用

案例1：

设计一个方法用于判断一个整数是否为闰年。

```java
package com.briup.chap03;

import java.util.Scanner;

public class Test06_LeapYear {
    //判断是否为闰年
    //函数传参过程：main方法中函数调用 isLeapYear(y);
    //                        int year = y; 将y的值传递给year，让其参与运算
    public static boolean isLeapYear(int year) {
        if((year % 4 == 0 && year % 100 != 0)
                || year % 400 == 0)
            return true;

        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入一个年份：");
        int y = sc.nextInt();

        if(isLeapYear(y))
            System.out.println("闰年");
        else
            System.out.println("非闰年");
    }
}
```

案例2：

设计一个方法用于判断一个整数是否是水仙花数，在控制台输出所有的水仙花数。

```java
package com.briup.chap03;
```

```java
public class Test06_Flower {
    //判断一个整数是否是水仙花数
    public static boolean isFlower(int number) {
        int g = number % 10;
        int s = number / 10 % 10;
        int b = number / 100 % 10;

        if ((g*g*g + s*s*s + b*b*b) == number) {
            return true;
        } else {
            return false;
        }
    }

    public static void main(String[] args) {
        // 遍历所有的3位数，逐个判断是否是水仙花数
        for(int i = 100; i < 1000; i++) {
            boolean f = isFlower(i);
            if(f)
                System.out.println(i + " 是水仙花数!");
        }
    }
}
```

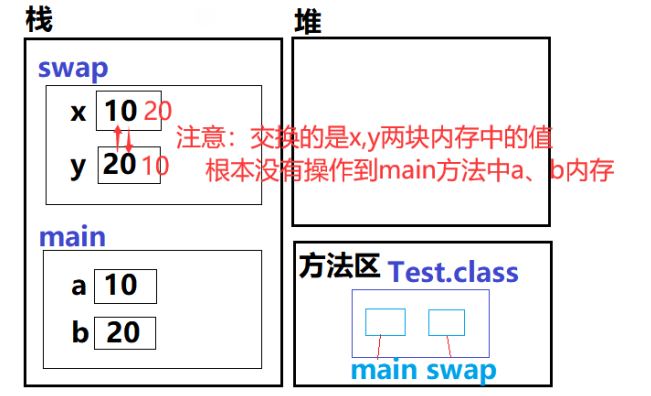
传值调用：

定义一个方法，用来交换两个整形数。

```java
package com.briup.chap03;

public class Test06_Swap {
    //    函数调用：  swap(a,b);
    //                int x = a; int b = y;
    public static void swap(int x, int y) {
        System.out.println("交换前，x: " + x + " y: " + y);
        x = x ^ y;
        y = x ^ y;
```

```
        x = x ^ y;
        System.out.println("交换后，x: " + x + " y: " + y);
    }

    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        //调用方法
        swap(a,b);

        System.out.println("交换后，a: " + a + " b: " + b);
    }
}

//输出结果
//交换前，x: 10 y: 20
//交换后，x: 20 y: 10
//交换后，a: 10 b: 20
```

思考：为什么a、b两个元素的值没有交换?



传值调用

只是把实际参数的值传递给了形式参数

函数调用执行过程中，不会操作到实参对应的内存空间

6.3 重载

同一个类中定义的多个方法，如果同时满足下列条件，则构成方法重载：

- 多个方法在同一个类中
- 多个方法具有相同的方法名
- 多个方法的参数列表不相同（类型不同或者数量不同）
- 重载跟函数的返回值类型无关

案例：

```
package com.briup.chap03;

//下面三个求和的方法就构成了重载
public class Test06_OverLoad {
    //重载1：求两个int类型数据和的方法
    public static int sum(int a,int b) {
        return a + b;
    }

    //重载2：求两个double类型数据和的方法
    public static double sum(double a,double b) {
        return a + b;
    }

    //重载3：求三个int类型数据和的方法
    public static int sum(int a,int b,int c) {
        return a + b + c;
    }

    // 测试
    public static void main(String[] args) {
        //调用方法时，Java虚拟机会通过实际参数的不同来区分同名的方法，从
而实现精准调用
        int result = sum(10, 20);
```

```
        System.out.println(result);

        double result2 = sum(10.0, 20.0);
        System.out.println(result2);

        int result3 = sum(10, 20, 30);
        System.out.println(result3);
    }
}
```

方法重载可以提高代码的可读性、简化方法调用、减少命名冲突、提高代码的复用性和提供灵活性。这些优势使得方法重载成为Java编程中常用的技术手段之一。