

1. 第一个java程序

```
public class helloworld{  
    public static void main(String[] args){  
        System.out.println("*");  
        System.out.println("hello, world!");  
    }  
}
```

1. 第一个java程序

```
→ mycode_java javac helloworld.java
→ mycode_java java helloworld
*
hello, world!
→ mycode_java java helloworld.class
Error: Could not find or load main class helloworld.class
Caused by: java.lang.ClassNotFoundException: helloworld.class
→ mycode_java ls -a
.          example          helloworld.java
..         helloworld.class
→ mycode_java ll -a
total 16
drwxr-xr-x  5 kunliu  staff   160B Sep  4 23:01 .
drwxr-xr-x 51 kunliu  staff  1.6K Jul 10 14:01 ..
drwxr-xr-x  4 kunliu  staff  128B Sep  4 16:21 example
-rw-r--r--  1 kunliu  staff  446B Sep  4 23:01 helloworld.cl
ass
-rw-r--r--@ 1 kunliu  staff  154B Apr 15 2020 helloworld.ja
va
→ mycode_java █
```



2. JDK

▼ Java	May
> Extensions	May
▼ JavaVirtualMachines	Yeste
▼ jdk-12.0.2.jdk	July
▼ Contents	July
> Home	July
Info.plist	July
> MacOS	July
▼ jdk-21.jdk	Yeste
▼ Contents	June
> _CodeSignature	June
▼ Home	Yeste
> bin	June
> conf	June
> include	June
> jmods	June
> legal	June
> lib	June
LICENSE	Yeste
> man	June
README	June
release	June
Info.plist	June
> MacOS	June

第二章 数据类型、运算符 和表达式

本章知识点

- ▶ 标识符和关键字
 - ▶ 基本数据类型、变量、常量
 - ▶ 运算符
 - ▶ 表达式的类型转换
 - ▶ 流程控制
 - ▶ 方法
- 

2.1 标识符和关键字

myName , My_name , Points ,
\$points , _sys_ta , OK , _23b , _3_

#name , 25name , class , &time , if

合法的标识符

非法的标识符

2.1 标识符和关键字

- ▶ **关键字**: Java预定义的单词。
 - 数据类型: byte、short、int、long、char、float、double、boolean
 - 包引入和包声明: import、package
 - 类和接口的声明: class、extends、implement、interface
 - 流程控制: if、else、switch、case、break、default、while、for、do、continue、return
 - 异常处理: try、catch、finally、throw、throws
 - 修饰符: abstract、final、private、protected、public、static、synchronized
 - 其他: new、instanceof、this、super、void、enum

2.2 基本数据类型与变量、常量



2.2 基本数据类型与变量、常量

数据类型	关键字	在内存中占用的位数	取值范围	成员默认值
字节型	byte	8	-128~127	(byte)0
短整型	short	16	-32768~32767	(short)0
整型	int	32	$-2^{31} \sim 2^{31} - 1$	0
长整型	long	64	$-2^{63} \sim 2^{63} - 1$	0L
字符型	char	16	0~65535	'\u0000'
单精度浮点型	float	32	1位符号,8位指数,23位尾数	0.0F
双精度浮点型	double	64	1位符号,11位指数,52位尾数	0.0D
布尔型	boolean	1	true, false	false

2.2.1 Java中的整数类型

- ▶ 整型常量按照所占用的内存大小分类
 - 整型(int)常量：占用32位。
如123, -34
 - 长整型(long)常量：占用64位，长整型常量的尾部有一个大写的L或小写的l。
如-386L, 0177771
 - 说明：java中的整型常量默认为int，表示long型整数后面加后缀。

2.2.2 Java中的字符类型

- ▶ **字符常量**：用一对单引号括起的单个字符。
 - 可见字符：'a'，'Z'，'8'，'#'
 - 转义字符
 - '\n'
 - '\t'
 - '\ddd'：8进制表示一个字符
 - '\uxxxx'：16进制无符号整数，表示Unicode码。

如：'\101' 用8进制表示一个字符'A'
'\u0041' 用Unicode码表示一个'A'

2.2.2 Java中的字符类型

转义字符	含义	对应Unicode码
'\b'	退格	'\u0008'
'\t'	水平制表符tab	'\u0009'
'\n'	换行	'\u000a'
'\f'	表格符	'\u000c'
'\r'	回车	'\u000d'
'\"'	双引号	'\u0022'
'\''	单引号	'\u0027'
'\\'	反斜线	'\u005c'
'\ddd'	三位8进制数表示的字符	
'\uxxxx'	四位16进制数表示的字符	

2.2.2 Java中的字符类型

- ▶ 字符串常量是用双引号括起的一串字符（可以0个）。

例子： " Hello" ，

" My \nJava" ，

" How old are you? 1234" ，

" "

" "

" My" + " name"



字符串常量是String类的对象

2.2.3 浮点类型

- ▶ 浮点型常量：表示可以含有小数部分的数值常量。
- ▶ 根据占用内存长度的不同分类
 - 单精度浮点常量：占用32位内存，用F、f表示。如：
19.4F, 3.0513E3, 8701.52f
 - 双精度浮点常量：占用64位内存，用带D或d或不加后缀的数值表示，
如：**2.433E-5D, 700041.273d, 3.1415。**
 - 说明：在java中的实型常量默认为double，所以写单精度的实数时要在数字后面写f，如3.14f。

2.2.4 布尔类型

- ▶ 布尔常量: true(真)和false(假)。
- ▶ 在流控制中经常用到布尔常量。

```
if (条件) 动作1  
else      动作2
```

- ▶ 注意: Java是一种严格的类型语言, 它不允许数值类型和布尔类型之间进行转换。

```
int a=3;  
if ( 0<a<1 ) .....
```

2.2.5 符号常量

- ▶ 在Java中必须用final关键字声明符号常量
- ▶ final关键字表示这个变量只能被赋值一次，一旦赋值后就不能够再更改。
- ▶ 声明格式
 - final 数据类型 常量名 = 缺省值;

```
final int STUDENT_NUM = 10;
```
- ▶ 习惯上，符号常量名采用全部大写，词与词之间用下划线分隔。

变量

- ▶ 变量: 在程序的运行过程中数值可变的数据, 用来记录运算中间值
- ▶ 变量的声明

byte, short, int, long,
float, double, char, boolean
复合类型

数据类型 变量名1, 变量名2, ... 变量n;

例如:

```
int num, total;
```

```
double d;
```

给变量分配
空间

4字节

num:

total:

8字节

d:

变量

- ▶ 变量的动态初始化: 在变量声明时使用表达式初始化变量。

```
class DynInit {  
    public static void main(String[] args){  
        double a = 3.0, b = 4.0;  
        double c = Math.sqrt(a * a + b * b);  
        System.out.println("Hypotenuse is: " + c);  
    }  
}
```

c被动态初始化

2.3 运算符

```
int S=-a*x*x+b*x+c;  
boolean l=a>b;
```

- ▶ **运算符**: 指明对操作数的运算方式。
- ▶ **按操作数的个数分**: 单目运算符 (如 $-a$), 双目运算符 (如 $a+b$), 三目运算符 (如 $e1?e2:e3$)。
- ▶ **按功能分类**
 - 算术运算符: $+$, $-$, $*$, $/$, $\%$, $++$, $--$
 - 关系运算符: $>$, $<$, $>=$, $<=$, $==$, $!=$
 - 逻辑运算符: $!$, $\&\&$, $\|\|$, $\&$, $\|$
 - 赋值运算符: $=$, $+=$, $-=$, $*=$, $/=$ 等
 - 位运算符:
 - 条件运算符: $?:$
 - 其它: \cdot , $[\]$, `instanceof`, $()$ 等

表达式: 由运算符、操作数 (常量、变量、方法调用) 和圆括号组成的式子。

2.3.1 算术运算符

- ▶ 算术运算符:对整型或实型数据的运算。
- ▶ 算术运算符分类
 - 双目运算符
 - 单目运算符

2.3.1 算术运算符与算术表达式

▶ 双目算术运算符

运算符	运算	例	功能
+	加	$a + b$	求a与b相加的和
-	减	$a - b$	求a与b相减的差
*	乘	$a * b$	求a与b相乘的积
/	除	a / b	求a除以b的商
%	取余	$a \% b$	求a除以b所得的余数

▶ 单目运算符：操作数只有一个。

运算符	运算	例	功能等价
++	自增	$a++$ 或 $++a$	$a = a + 1$
--	自减	$a--$ 或 $--a$	$a = a - 1$
-	求负数	$-a$	$a = -a$

2.3.1 算术运算符与算术表达式

前缀和后缀运算符举例

例如：

```
int x = 5 ;
```

```
int y = (--x) * 3;
```

x为4 y为12

x为4 y为15

int y = (x--) * 3; ? ?

2.3.1 算术运算符与算术表达式

【例2-1】 写出下面程序运行的结果。

```
public static void main(String[] args) {  
    int a=10, b=20;  
    System.out.println("a+b="+a+b);  
    System.out.println("a+b="+(a+b));  
}
```

2.3.2 关系运算符和逻辑运算符

运算符	运算	例
$=$	等于	$a==b$
$!=$	不等于	$a!=b$
$>$	大于	$a>b$
$<$	小于	$a<b$
$>=$	大于等于	$a>=b$
$<=$	小于等于	$a<=b$

2.3.2 关系运算符和逻辑

true || false

(3>1) && (5>-4)

! false

- ▶ 逻辑运算是对布尔型数据进行的运算，运算的结果仍然是布尔型。
- ▶ 常用的逻辑运算符

运算符	运算	例	运算规则
!	逻辑取反	!x	x真时为假,x假时为真
	逻辑或	x y	x,y都假时结果才为假(短路)
&&	逻辑与	x && y	x,y都真时结果才为真(短路)
^	布尔逻辑异或	x ^ y	x,y同真同假时结果为假
&	布尔逻辑与	x & y	x,y都真时结果才为真
	布尔逻辑或	x y	x,y都假时结果才为假

2.3.2 关系运算符和逻辑运算符

短路

逻辑运算符与布尔逻辑运算符的区别

例如： `int x = 3, y = 5;`

```
boolean b = x > y && x++ == y--;
```

//x为3, y为5, b为false

```
boolean b = x > y & x++ == y--;
```

//x为4, y为4, b为false

2.3.3 位运算符

- ▶ **位运算**是对操作数以二进制比特位为单位进行的操作和运算，位运算的运算对象只能是整型和字符型，结果为整型。

2.3.4 赋值运算符

- Java中赋值运算符：`=`、`+=`、`-=`、`*=`等。
- 赋值表达式:带有赋值运算符的表达式。
- 赋值表达式的含义:等号右边表达式的值赋给等号左边的变量。
- 赋值表达式的类型：等号左边变量的类型。
- 赋值运算的结合性：自右向左。

例如，`i=5` //赋值表达式的值是5

```
i= 1; //表达式值为1
i=j=k=1; //表达式值为1，i,j,k的值为1
i=2+(j=4); //表达式值为6，j的值为4，i的值为6
i=(j=10)*(k=2); //表达式值为20，j的值为10，k的值为2，i的值为20
```

2.3.4 赋值运算符

- 常用的复合赋值运算符

运算符	例子	等价于
$+=$	$x += a$	$x = x + a$
$-=$	$x -= a$	$x = x - a$
$*=$	$x *= a$	$x = x * a$
$/=$	$x /= a$	$x = x / a$
$\%=$	$x \% = a$	$x = x \% a$

- 例： $a+=3$ 等价于 $a=a+3$
 $x*=y+8$ 等价于 $x=x*(y+8)$

其它运算符

▶ 条件运算符与条件表达式

$e1? e2: e3$

- $e1$ 为 boolean 类型
- $e2$ 与 $e3$ 的类型相同
- 执行顺序
 - 若 $e1$ 的值为true， $e2$ 的值为最终结果
 - 若 $e1$ 的值为false， $e3$ 的值为最终结果

例如: $y = x \geq 0 ? x : -x$

$\text{max} = x > y ? x : y$

2.3.5 运算符的优先级与结合性

- ▶ **表达式的运算次序**: 取决于表达式中各种运算符的优先级。优先级高的运算符先运算, 优先级低的运算符后运算, 同一行中的运算符的优先级相同。
- ▶ **运算符的结合性**: 决定了并列的相同运算符的先后执行顺序。

2.3.5 运算符的优先级与结合性

优先级	描述	运算符	结合性
1	最高优先级	. [] ()	左→右
2	单目运算	+(正号) -(负号) ++ -- ~ ! 强制类型转换符	右→左
3	算术乘除运算	* / %	左→右
4	算术加减运算	+ -	左→右
5	移位运算	>> << >>>	左→右
6	关系运算	< <= > >=	左→右
7	相等关系运算	== !=	左→右
8	按位与,布尔逻辑与	&	左→右
9	按位异或	^	左→右
10	按位或,布尔逻辑或		左→右
11	逻辑与	&&	左→右
12	逻辑或		左→右
13	三目条件运算	? :	右→左
14	赋值运算	= += -= *= /= %= <<= >>=	右→左

基本类型数据占有的内存宽度

数据类型	关键字	占用位数	取值范围
布尔型	boolean	8	true, false
字符型	char	16	'\u 0000' ~ '\u FFFF'
字节型	byte	8	-128~127
短整型	short	16	-32768~32767
整型	int	32	-2147483648 ~ 2147483647
长整型	long	64	$-2^{63} \sim 2^{63}-1$
浮点型	float	32	1.40129846432481707e-45~ 3.40282346638528860e+38
双精度型	double	64	4.94065645841246544e-324~ 1.79769313486231570e+308d

2.4 表达式的类型转换

- 当表达式中出现了多种类型数据的混合运算时, 需要进行类型转换。

数据类型	关键字	占用位数	取值范围
布尔型	boolean	8	true, false
字符型	char	16	'\u 0000' ~ '\u FFFF'
字节型	byte	8	-128~127
短整型	short	16	-32768~32767
整型	int	32	-2147483648 ~ 2147483647
长整型	long	64	$-2^{63} \sim 2^{63}-1$
浮点型	float	32	1.40129846432481707e-45~ 3.40282346638528860e+38
双精度型	double	64	4.94065645841246544e-324~ 1.79769313486231570e+308d

2.4.1 数据类型自动转换的规则

【例2-6】分析下面的赋值出错的原因。

```
public static void main(String[] args) {  
    int a=1.2345;  
  
    byte b = 1;  
    b=b+1;  
  
    float c = 1.5;  
}
```

2.4.2 强制类型转换

- ▶ 从较长的数据类型转换成较短的数据类型时，必须做强制类型转换。即将表达式的类型强制性地转换成某一数据类型。
- ▶ 强制类型转换的格式
 （数据类型）表达式

2.5 流程控制

- ▶ 算法的基本控制结构
 - 顺序结构
 - 选择结构
 - 循环结构
- 与算法的基本控制结构相关的Java语句
 - 分支语句：if-else, switch
 - 循环语句：while, do-while, for
 - 与程序转移有关的其它语句：break, continue, return

2.5.1 if语句

(1) if (表达式) 语句

例: `if (x <= 0) x = -x ;`

(2) if (表达式) 语句1 else 语句2

例: `if (x>y) z=x;
else z=y;`

(3) if (表达式1) 语句1
else if (表达式2) 语句2
else if (表达式3) 语句3
...
else 语句 n

`-10<x<0`

例:

`if (x>0) y=1;`

`else if (x==0) y=0;`

`else if (x<0 && x>-10)`

`y= -1;`

2.5.1 if语句

```
if(year%4==0 && year%100!=0 || year%400==0){  
    System.out.println(year+"是闰年");  
}else{  
    System.out.println(year+"不是闰年");  
}
```

例2-7

2.5.2 switch语句

▶ 一般形式

switch (表达式)

```
{ case 常量表达式 1: 语句1  
  case 常量表达式 2: 语句2  
    |  
  case 常量表达式 n: 语句n  
  default: 语句n+1  
}
```

可以是整型、字符型

每个常量表达式的值不能相同，次序不影响执行结果。

可以是多个语句，但不必用{ }。

● 执行顺序

以case中的**常量表达式值**为入口标号，由此开始顺序执行。因此，每个case分支最后应该加**break**语句。

2.5.2 switch语句

```
char myGrade= 'A';
```

```
switch (myGrade){  
    case 'A' :    myScore = 5 ;  
    case 'B' :    myScore = 4 ;  
    case 'C' :    myScore = 3 ;  
    default :    myScore = 0 ;  
}
```

myGrade的值为'A'，执行完switch语句后，myScore的值被赋值为0

```
switch (myGrade){  
    case 'A' :    myScore = 5 ;    break;  
    case 'B' :    myScore = 4 ;    break;  
    case 'C' :    myScore = 3 ;    break;  
    default :    myScore = 0 ;  
}
```

myGrade的值为'A'，执行完switch语句后，myScore的值被赋值为5

2.5.2 switch语句

多个不同的case值可以执行一组相同的操作。

```
switch (myGrade)
{
    case 'A' :
    case 'B' :
    case 'C' : myScore = 1 ; //及格
                break ;
    default : myScore = 0 ; //不及格
}
```

例：2-8

分支语句练习：

1.星座查询程序。

2.个人所得税计算。

3.属相查询程序。

。

2.5.3 while循环语句

- ▶ 循环结构是在**一定条件下**，反复执行某段程序的流程结构，被反复执行的程序被称为**循环体**。
- ▶ Java的循环语句
 - while语句
 - do-while语句
 - for语句

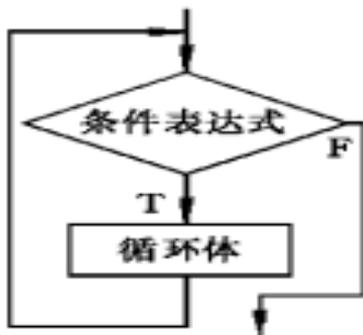
2.5.3 while循环语句

- while 语句形式

while (条件表达式) 语句

循环体可以是复合语句，其中必须含有改变条件表达式值的语句。

- 执行顺序



例2-9

2.5.4 for循环语句

▶ 语法形式

for (表达式1; 表达式2; 表达式3) 语句

循环前先求解,

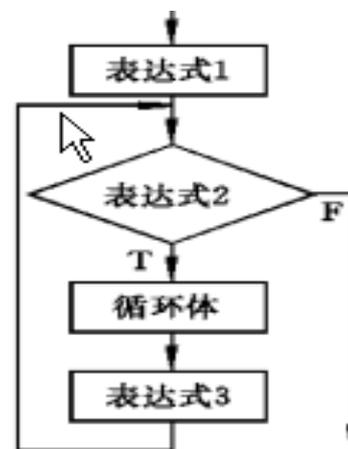
完成初始化循环变量和其他变量

为true时执行循环体

每次执行完循环体后求解.

用于改变循环控制变量的值

例: `for(i=1; i<=100; i++) sum+=i;`



2.5.4 for循环语句

关于for语句的几点说明

(1) for语句的三个表达式可以为空 (但分号不能省略)

```
for (; ;) 语句;           //相当于 while (true) 语句;  
for (; i<=100;) 语句;    //相当于 while (i<=100) 语句;
```

(2) 在表达式1和表达式3的位置上可包含多个语句

```
for(sum=0, int i=1; i<=100; i++) sum+=i;
```

2.5.4 for循环语句

(3) 多种表达方式

```
sum=0;  
i=1    //在for语句之前给循环控制变量赋初值  
for (; i<100; i++) sum=sum+i;    //省略表达式1
```

```
i=1    //在for语句之前给循环控制变量赋初值  
for (sum=0; i<100; i++) //表达式1与循环控制变量无关  
    sum=sum+i;
```

```
for (sum=0, i=1; i<100; ){    //省略表达式3  
    sum=sum+i; i++; }        //在循环体中改变循环控制条件
```

```
for( i=0, j=10; i<j; i++, j--) {.....}  
    // 表达式1和表达式3可以是逗号表达式
```

2.5.4 for循环语句

▶ 注意事项



```
sum=0;  
for(int i=1; i<=100; i++) //在for语句中声明循环控制变量并赋初值  
    sum+=i;  
System.out.println(i);    //!Error
```

例：2-10

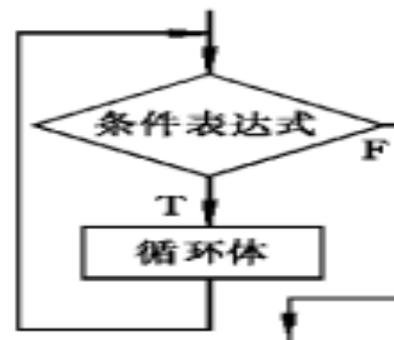
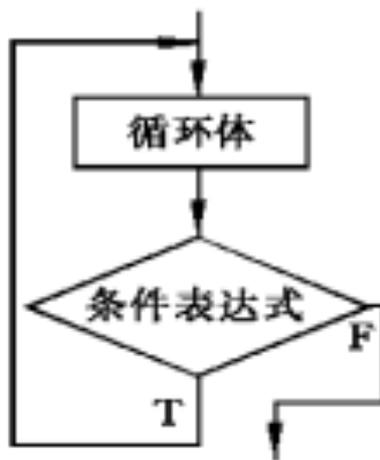
2.5.5 do-while循环语句

■ 一般形式

do **语句**
while (表达式)

← 循环体可以是复合语句，其中必须含有改变条件表达式值的语句。

■ 与while语句的比较

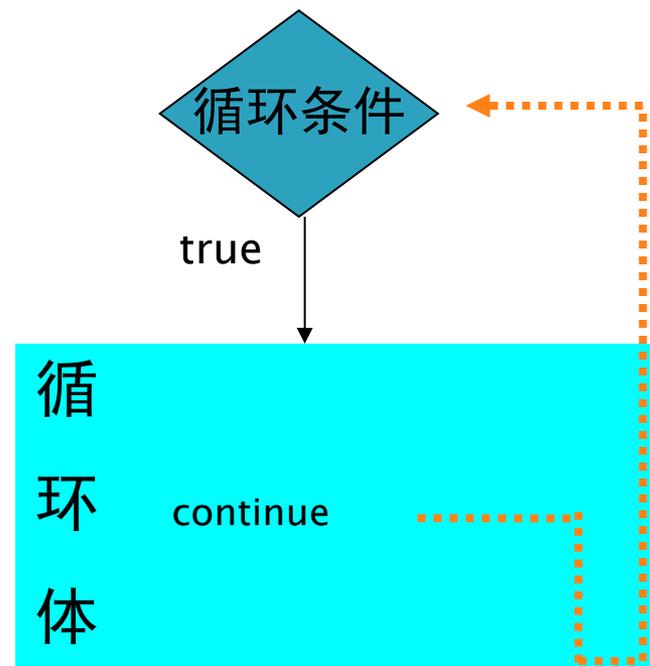
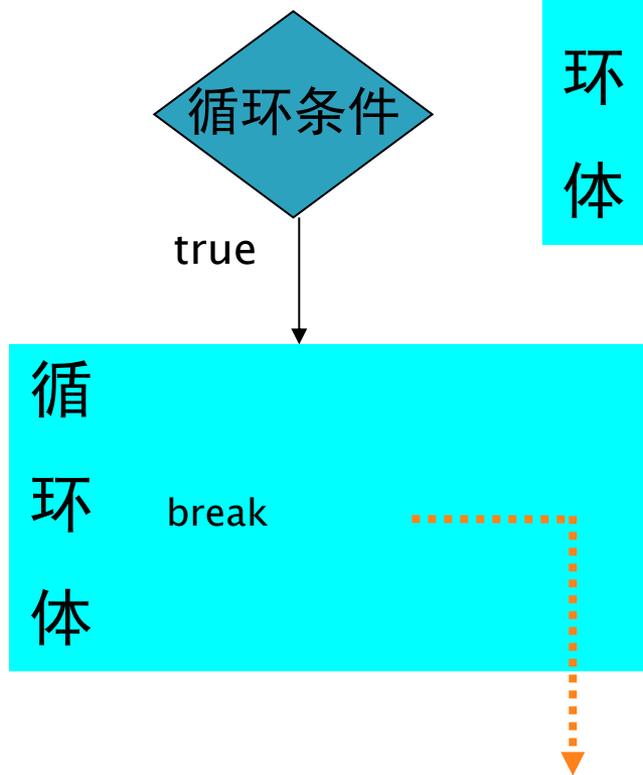


例2-11

2.5.6 break语句

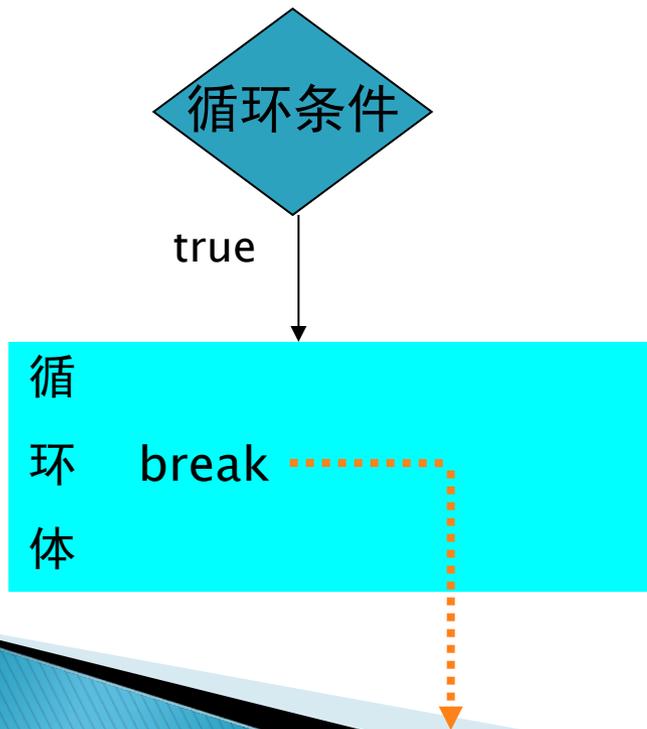
▶ 改变程序控制流语句

- break
- continue
- return



2.5.6 break语句

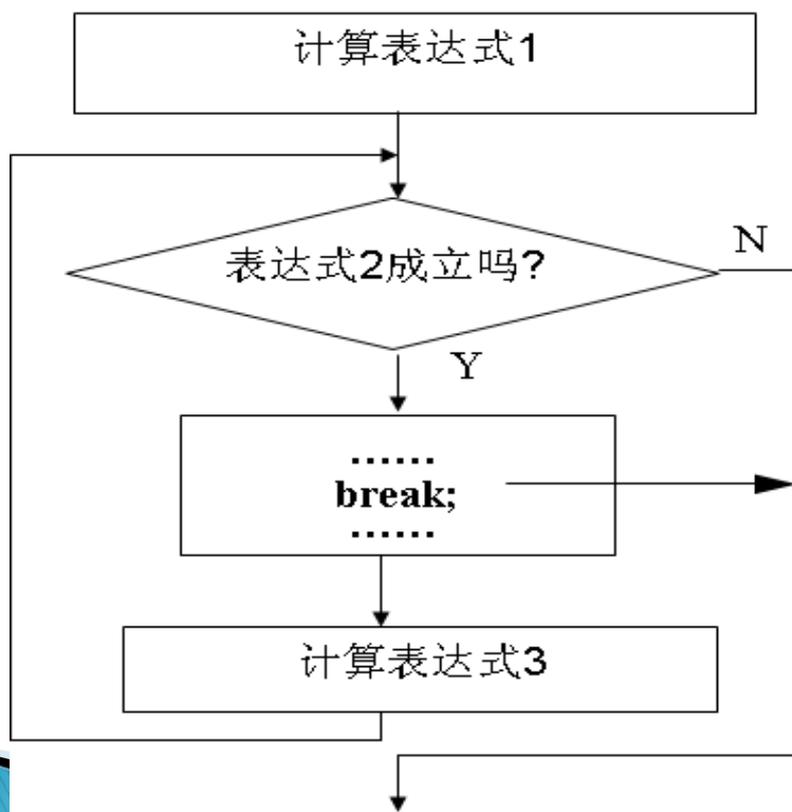
- ▶ break语句仅出现在switch语句或循环体中。
 - 作用：使程序的流程从一个语句块内部跳转出来，即从switch语句的分支中跳出，或从循环体内部跳出。



2.5.6 break语句

▶ for循环结构中的break语句

例：2-22



2.5.6 break语句

- Java语言中break语句的特殊格式
 - break [标号];

```
stop:
for (int i=1; i<=10; i++) {
    for (int j=1; j<=5; j++) {
        if (i==5) break stop;
        System.out.print( "*" );
    }
    System.out.println();
}
```

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

作用：快速地从多重循环内部退出

2.5.7 循环的嵌套

- ▶ 循环的嵌套：一个循环体内又包含另一个**完整**的循环结构。
- ▶ 三种循环语句（while循环, do-while循环和for循环）它们可以相互嵌套使用。

循环语句练习：

1. 用三种循环语句输出1-100的和。
2. 打印9*9乘法表。
- 3 学生成绩等级转换程序。
4. 输出n以内的所有“亲密数”。

2.6 方法

重点、难点

重点：

1. 方法声明
2. 方法调用
3. 参数传递

难度：理解参数传递。

2.6 方法

- ▶ 函数=方法=模块化设计
- ▶ Java中所有的方法都必须封装在类中，不能单独出现、使用。

2.6.1 方法的定义

- ▶ Java中方法定义的基本格式为：

```
[修饰符] 返回值类型 方法名([形式参数列表]){  
    [方法体]  
}
```

- ▶ 修饰符：定义方法在类中的存在属性（如公有/私有、是否可以被重载等）
- ▶ 返回值类型：任何合法的数据类型（Java基本数据类型或自定义数据类型），如果方法没有返回值则定义为“void”
- ▶ 形式参数列表：定义方法需要接收的数据及相应数据类型，参数列表可缺省
- ▶ 方法体：由完成其逻辑功能的Java语句组成，可为空。

补充说明：

- ▶ **参数表**指定在调用该方法时，应该传递的参数的个数和数据类型。
- ▶ **对于有返回值的方法**，其方法体中至少有一条return语句。
- ▶ **方法声明不能嵌套**，即不能在方法中再声明其它的方法。

2.6.1 方法的定义

【例2-16】判断某数是否是素数的方法

```
public static boolean isPrime(int x){  
    for(int div=2; div<=Math.sqrt(x); div++){  
        if(x%div==0){  
            return false;  
        }  
    }  
    return true;  
}
```

补充内容1:方法调用

1) 方法表达式

对于有返回值的方法作为表达式或表达式的一部分来调用，其在表达式中出现的形式为：

方法名（[实际参数表]）

例 调用前面定义的方法

```
public class SquareC
```

```
{ static int square(int x)
```

```
{ int s;
```

```
    s=x*x;
```

```
    return (s);
```

```
}
```

```
public static void main(String[] args)
```

```
{
```

```
int n = 5; int result = square(n);
```

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

```
}
```

```
}
```

补充内容1:方法调用

2) 方法语句

方法名([实际参数表])

即以独立语句的方式调用方法。

例 以方法语句方式调用方法

```
class AreaC
```

```
{ static void area(int a , int b )
```

```
{int s;
```

```
    s = a * b;
```

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

```
}
```

```
public static void main(String[] args)
```

```
{
```

```
    int x = 5;  int y=3;  area(x, y);
```

```
}
```

```
}
```

例 无参方法

```
class SumC
```

```
{
```

```
    static void sum( )
```

```
    {
```

```
        int i, j, s;
```

```
        i=3; j=6; s=i+j;
```

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

```
    }
```

```
    public static void main(String[] args)
```

```
    {    sum( );    }
```

```
}
```

补充内容2： 参数传递

在调用一个带有形式参数的方法时，必须为方法提供实际参数，完成实际参数与形式参数的结合，称为参数传递，然后用实际参数执行所调用的方法体。

在Java中，参数传递是以传值的方式进行，即将实际参数的值传递给形式参数。

例一—交换两个变量的值

```
public class Swaping
{
    static void swap(int x , int y )
    {   int temp ;
        System.out.println("Before Swapping");
        System.out.println("x= "+x+" y= "+y);
        temp = x; x = y; y = temp;
        System.out.println("After Swapping");
        System.out.println("x= "+x+" y= "+y);
    }
}
```

```
public static void main(String[] args)
{
    int u=23 , v=100;
    System.out.println("Before Calling");
    System.out.println("u= "+u+" v= "+v);
    swap(u, v);
    System.out.println("After Calling");
    System.out.println("u= "+u+" v= "+v);
}
}
```

输出结果如下所示：

Before Calling

$u = 23$ $v = 100$

Before Swapping

$x = 23$ $y = 100$

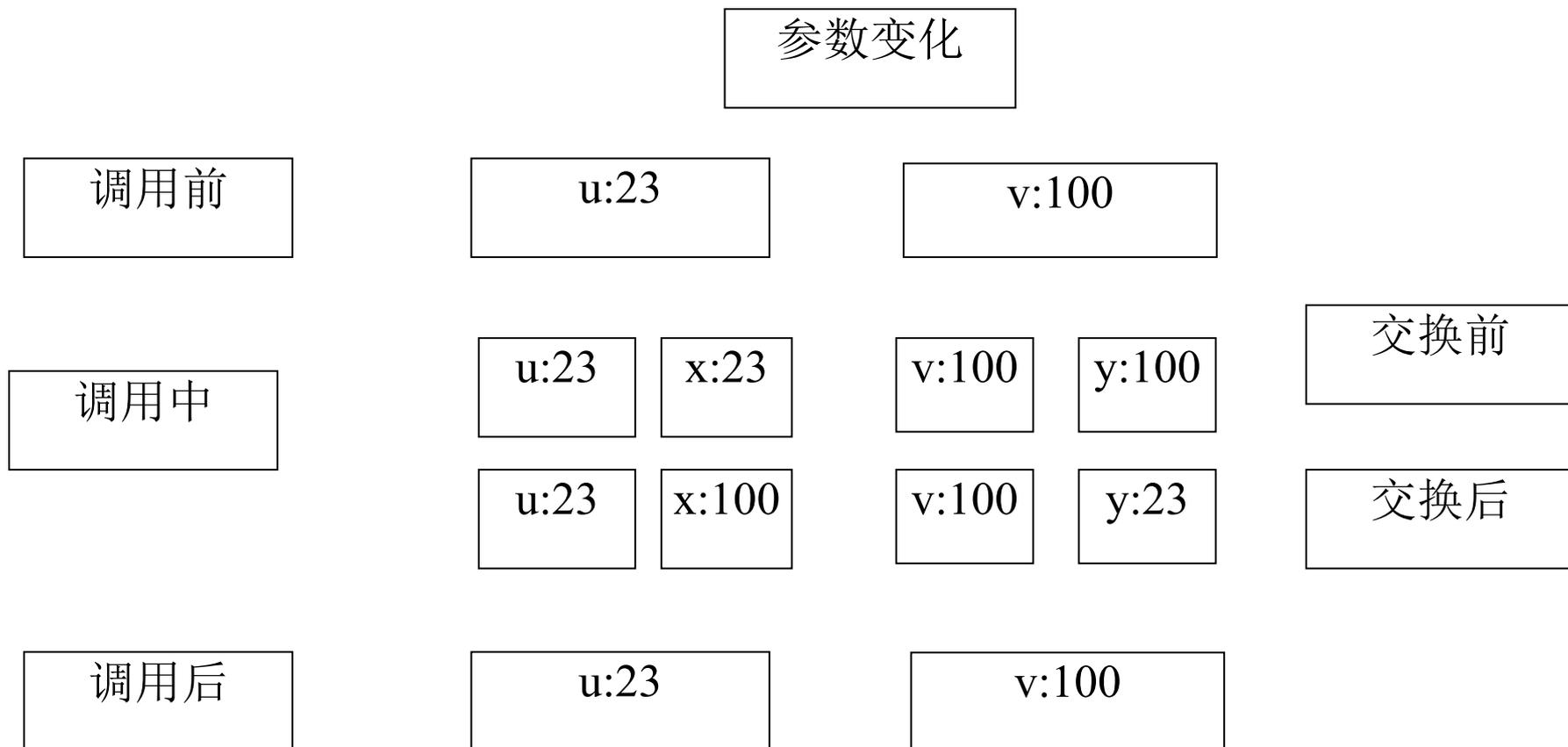
After Swapping

$x = 100$ $y = 23$

After Calling

$u = 23$ $v = 100$

实际参数和形式参数值的变化过程如图所示



2.6.2 方法的重载

- ▶ 方法的重载：在一个类中定义多个同名的方法，但方法有不同类型的参数或参数个数。
 - 方法重载能减少程序员为方法命名的苦恼，使相同功能的方法使用统一的名称来调用。
 - 匹配的过程由编译器完成（重载解析），如果编译器找不到参数相匹配的方法，或者找出多个参数匹配的方法，就会产生编译时错误。



重载只与参数有关，与返回值无关。

- ①参数的类型不同
- ②参数的个数不同

2.6.2 方法的重载

【例2-16】设计打印金字塔的方法printPyramid(), 可以打印数字金字塔, 也可以打印字母金字塔。

```
public void printPyramid(int n){
    //打印n行数字组成的金字塔
    .....
}
public void printPyramid(char ch){
    //打印'a'~ch字母组成的金字塔
    .....
}
```

```
      1
     1 2 1
    1 2 3 2 1
   1 2 3 4 3 2 1
  1 2 3 4 5 4 3 2 1
 1 2 3 4 5 6 5 4 3 2 1
1 2 3 4 5 6 7 6 5 4 3 2 1
```

```
      a
     a b a
    a b c b a
   a b c d c b a
  a b c d e d c b a
 a b c d e f e d c b a
a b c d e f g f e d c b a
```

上机：

2.7 综合实践

2.9 实验指导

【题目】将几个算术运算功能组织为菜单的形式，供用户选择。

- 菜单：while循环+输入控制
- 算术练习器：do-while循环结构+switch+随机数

本章思维导图

