第二章 Java语法基础

1. 词法规则
2. 数据类型
3. 常量与变量
4. 运算符和表达式
5. 语句
6. 数组和字符串

词法规则

- Java程序的运行体系

1. Source code (.java file)
2. javac: Lexical Analysis & Parsing + Type-checking → Byte code (.class file)

   Java编译器对源代码进行词法分析和类型校验，生成字节码文件
3. JVM: Verification (essentially repeating static checks) + (Interpretation OR Compilation + Loading + Executing)

   Java解释器执行字节码文件中的类，Java解释器在加载和执行类时验证类的完整性、正确操作和安全性，并与所在的操作系统、窗口环境和网络设备进行交互以产生所期望的程序行为

关键字

1.	abstract	double	int	strictfp **
2.	boolean	else	interface	super
3.	break	extends	long	switch
4.	byte	final	native	synchronized
5.	case	finally	new	this
6.	catch	float	package	throw
7.	char	for	private	throws
8.	class	goto *	protected	transient
9.	const *	if	public	try
10.	continue	implements	return	void
11.	default	import	short	volatile
12.	do	instanceof	static	while

注: *→当前未被使用　　**→使用于Java2

标识符

- 标识→常量、变量、数据类型、类和方法

```
public class HelloWorld1 {
  public static void main(String[] args) {
        String message = “Hello World!”;
        myPrint(message);
  }
  private static void myPrint(String s) {
        System.out.println(s);
  }
}
```

标识符

- 组成规则
  1. 字母(A~Z、a~z)、特殊符号($、_)和数字(0~9)
  2. 第1个符号不能为数字
  3. 不能为关键词、true、false、null
  4. 区分大小写

标识符

- 例：point4、5w、A%、thisPicture、$currentValue、OK、_23b、Y_123、#length、a+b、if
- 5w、A%、#length、a+b、if

标识符

- 一般约定
  1. 表示常量的标识符全部大写，如RED
  2. 表示类名的标识符用大写字母开始，如MyCar
  3. 表示公有方法和实例变量的标识符用小写字母开始，后面的描述性词以大写开始，如getCurrentValue
  4. 表示私有或局部变量的标识符全部用小写字母，如next_value

分隔符

- 空白符
  - 空格、换行符、制表符
- 分号
  - 表示语句结束，或用于for循环语句中
- 逗号
  - 变量之间的分隔
- 冒号
  - ?: /switch循环中的case语句
- 花括号
  - 类体、方法体、复合语句(for/while/switch/if)

第二章 Java语法基础

1. 词法规则
2. 数据类型
3. 常量与变量
4. 运算符和表达式
5. 语句
6. 数组和字符串

数据类型

- 基本数据类型
  - 数字 (number)
    - 整型 (integers)
      1. 字节整数 (byte, 8 bits): -128 ~127, 0
      2. 短整数 (short, 16 bits): -32768 ~ 32767, 0
      3. 整数 (int, 32 bits): -2147483648 ~ 2147483647, 0
      4. 长整数 (long, 64 bits): … …, 0L
    - 实型 (real numbers): 浮点型 (有效位数不同)
      1. 单精度(float, 32 bits): … …, 0.0F
      2. 双精度(double, 64 bits): … …, 0.0D
  - 字符 (char, 16-bit Unicode字符): \u0000 ~ \uffff
  - 布尔 (boolean): true, false
- 复合数据类型
  - 数组(Array), 类(class), 接口(interface)

数据类型

- 示例

1. int i = 178;
2. long l = 8864L; (8864l)
3. double d1 = 37.266;
4. double d2 = 37.266D; (37.266d)
5. double d3 = 26.77e3;
6. float f = 87.363F; (87.363f)
7. char c = 'd';
8. boolean b1 = true;
9. boolean b2 = false;

类型转换 (Casting)

- 将一种类型的数据转换为另一种类型的数据
  - 操作数转换为同种类型，然后运算
  - 整数型、实数型和字符型
  - 表达形式：(类型) 操作数
- 应用场合
  1. 二元运算符的二个操作数类型不同
  2. 表达式值的类型与变量的类型不同
- 两种方法
  1. 隐型类型转换：自动类型转换(系统完成)
  2. 显型类型转换：强制类型转换

类型转换 (casting)

- 隐型类型转换：自动类型转换(系统完成)
  - 宽化转换(widening conversion)

```
byte j=60; short k=4; int l=31; long m=4l;
long result=0l;
result +=j-8;
result *=k+2;
result /=m+1;
result -=l;
result %=m;
```

类型转换 (casting)

- 隐型类型转换：自动类型转换(系统完成)

类型转换表

源类型	转换后不会丢失数据的目的类型
byte	short, char, int, long, float, double
short	char, int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

类型转换 (casting)

- 显型类型转换：强制类型转换
  - 窄化转换 (narrowing conversion)

```
double a = 1.5;
float b = a;
System.out.println("b=" + b);
```

编译："possible loss of precision"

数据精度丢失→数据丢失

```
double a = 1.5;
float b = (float)a;
System.out.println("b=" + b);
```

类型转换 (casting)

- 显型类型转换：强制类型转换

```
class Test {
    public static void main(String args[]) {
        int a = 257;
        byte b = (byte)a;
        System.out.println("a=" + a);
        System.out.println("b=" + b);
    }
}
```

257　00000000 00000000 00000001 00000001

1　00000001

类型转换 (casting)

- 显型类型转换：强制类型转换

```
class Test {
    public static  void main(String args[]) {
        char c1 = 'A',  c2;      // A的ASCII值为65
        int i;
        i = (int) c1 + 1;
        c2 = (char) i;
        System.out.println(c1 + c2);
        System.out.println(c1 + " ," +c2);
    }
}
```

第二章 Java语法基础

1. 词法规则
2. 数据类型
3. 常量与变量
4. 运算符和表达式
5. 语句
6. 数组和字符串

常量

- 程序执行过程中，值保持不变的量
- 整型常量
- 实型常量
- 布尔型常量
- 字符型常量
- 字符串常量

常量

- 整型常量
  - 常用十进制、八进制、十六进制表示
  - 有正负号

	起始	最大整数(正)	最大长整数(正)	举例
十进制	0,1~9	2147483647	9223372036854775807L	23, +567, -12,0,1234
八进制	0	017777777777	0777777777777777777777L	034,0175, -0777L
十六进制	0x	0x7FFFFFFF	0x7FFFFFFFFFFFFFFFL	0xFF, 0x45L

常量

- 实型常量
  - 双精度实数(double, 8个字节, 数字后加字母D或d)
  - 浮点实数(float, 4个字节,数字后加字母F或f)
  - 若无明确字母标识，则系统默认为双精度实数
  - 两种表示方法
    - 十进制：数字和小数点组成，必须有小数点，例 0.12, .12, 12., 12.0
    - 科学计数法：123e3, 123E3, 0.4e8D, -5e9

常量

- 布尔型常量
  - true
  - false

常量

- 字符型常量
  - 用单引号括起来的单个字符
    - 例：'a'，'A'，'@'，''，'&'
    - 例：'''，'\'，"a"
  - JAVA中的字符为Unicode字符
    - 双字节，范围 '\u0000'~'\uFFFF'
    - 转义字符序列
      - \b 退格　　\t 制表符
      - \n 换行 (Newline)
      - \r 回车 (Carriage return)
      - \' 单引号　\" 双引号　\\ 反斜杠

常量

- 字符串常量
  - 用双引号括起来的若干个字符
  - 例，“I am a student”，“java语言”，“A”
  - 转义字符序列表示
    - “ \" ”
    - “ \' ”

常量

转义字符序列

```
class Test {
    public static void main(String args[]) {
        System.out.println(“java\n语\b言");
        System.out.println("java\r语言");
        System.out.println("java\t语言");
        System.out.println("\\java语言\\");
        System.out.println("\'java语言\'");
        System.out.println(“\”java语言\“”);
    }
}
```

```
C:\>java Test
java
言
语言
java    语言
\java语言\
‘java语言’
 “java语言”

C:\>
```

变量

- 程序执行过程中，值可以改变的量
- 整型变量、实型变量、字符型变量、字符串变量、布尔变量等
- 变量定义
  - 类型 变量名 [=初值][,变量名[=初值]...]
  - 类型：基本数据类型或引用类型

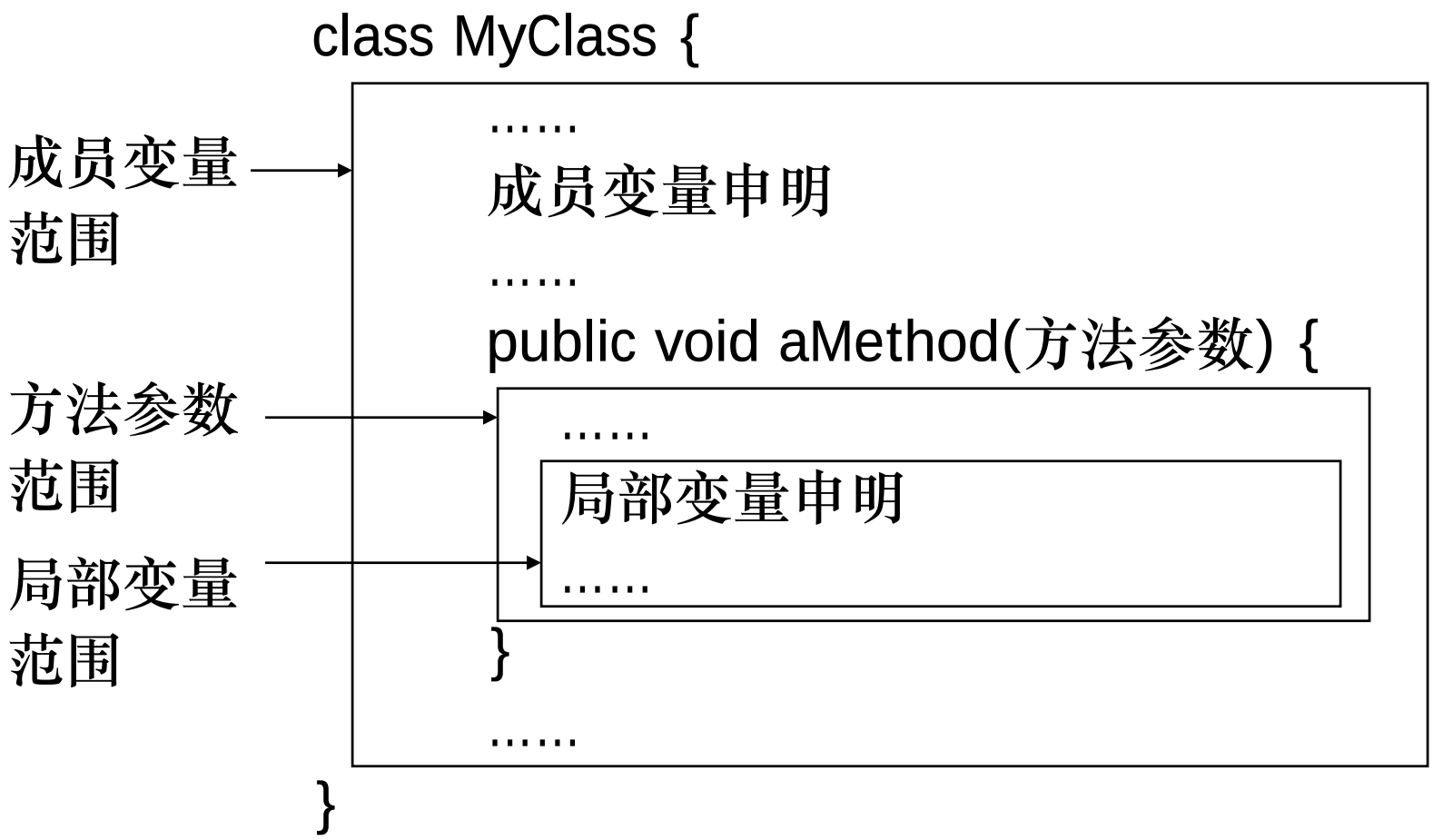
```
int       x, y, z;
float     a, b;
char      c1, c2, c3;
double    d1;
boolean   mycom;
```

变量

- 变量赋初值/初始化
- 在变量声明时赋值
  1. 类型 变量名 [=初值][, 变量名[=初值] ...]
  2. int x=1, y=2, z=3;
  3. float e = 2.718281828f;
- 采用赋值语句
  1. float pi, y;
  2. pi = 3.1415926f;
  3. y = 2.71828f;

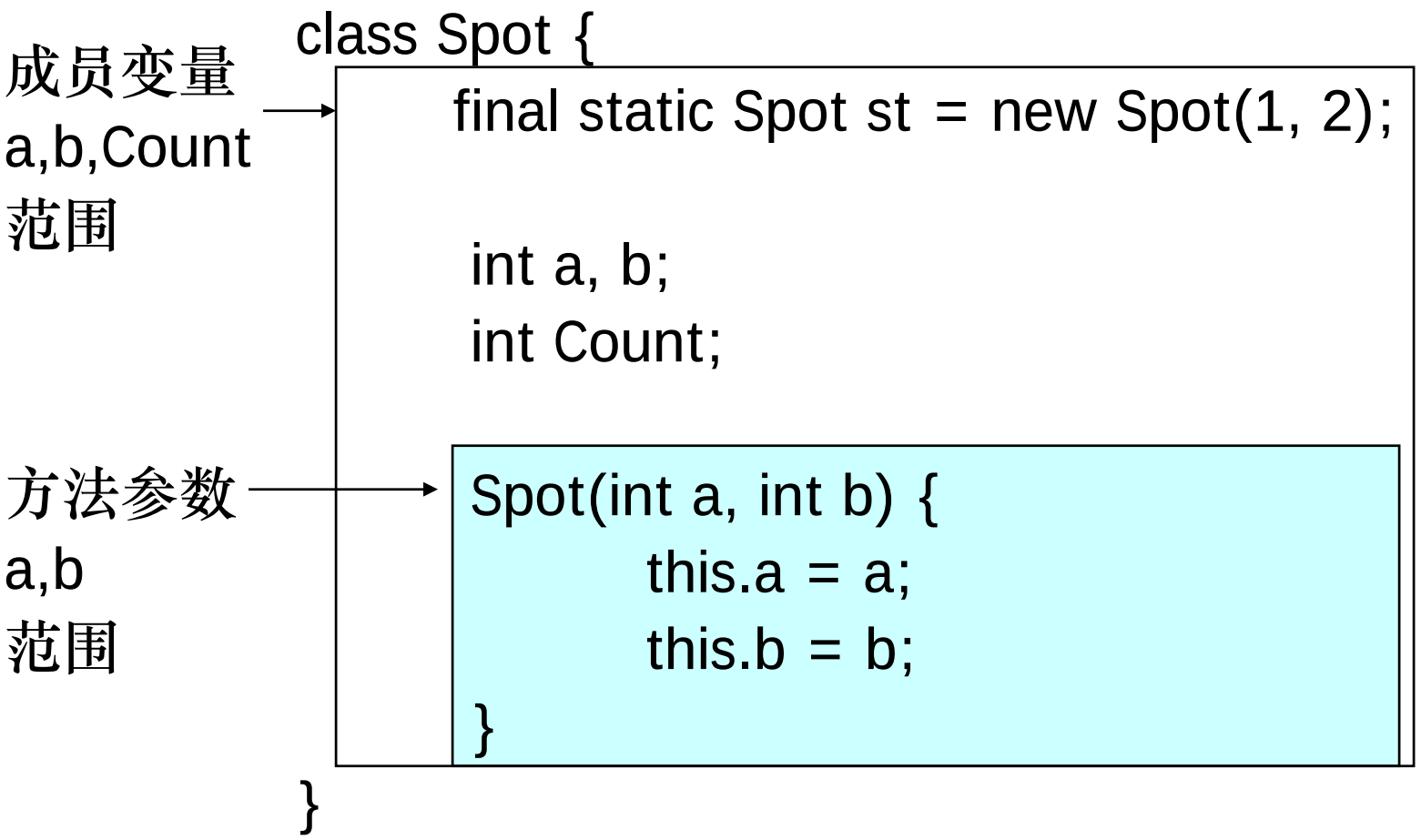
变量

变量的作用域


class MyClass {
.....
成员变量申明
.....
public void aMethod(方法参数) {
.....
局部变量申明
.....
}
.....
}
成员变量
范围
方法参数
范围
局部变量
范围

变量

■ 变量的作用域—变量的使用范围



变量

- 作用域(Scope)

```
if ( ... ) {
    int i = 17;
    ...
}
System.out.println("The value of i = " + i);
```

变量

- final变量
  - 带有关键字final的变量

    final int aFinalVar = 0;
  - final变量初始化后不能再改变

    final int blankfinal;

    . . .

    blankfinal = 0;

    ...

    blankfinal = 3;

Unicode

- **What is Unicode?**
- **Unicode คืออะไร?**(in Thai)
- **یونی‌کُد چیست؟**(in Persian)
- **什么是Unicode(统一码)?**
- **什麼是Unicode(統一碼/標準萬國碼)?**
- (in Arabic)؟ **ما هي الشفرة الموحدة "يونِكود"**
- **Was ist Unicode?** (in German)
- **Что такое Unicode?**(in Russian)
- ユニコードとは何か? (in Japanese)

汉字字符集

- GB2312-80
  - 中华人民共和国国家汉字信息交换用编码，全称《信息交换用汉字编码字符集——基本集》，国家标准总局发布，1981年5月1日实施
  - 收录简化汉字及符号、字母、日文假名等共7445个图形字符，其中汉字占6763个
  - “对任意一个图形字符都采用两个字节表示，每个字节采用七位编码表示”，第一个字节为“高字节”，第二个字节为“低字节”
- GBK
  - 全国信息技术化技术委员会于1995年12月1日《汉字内码扩展规范》，GBK共收入21886个汉字和图形符号

汉字字符集

- BIG5
  - 1983年10月，台湾国家科学委员会、教育部国语推行委员会、中央标准局、行政院共同制定了《通用汉字标准交换码》，后经修订于1992年5月公布，更名为《中文标准交换码》，BIG5是台湾资讯工业策进会根据以上标准制定的编码方案
  - BIG5码是双字节编码方案，第一个字节的值在0xA0-0xFE之间，第二个字节在0x40-0x7E和0xA1-0xFE之间
  - BIG5收录13461个汉字和符号

为什么需要Unicode？

- 不同字符集编码的内码定义不一样
  - 汉字　GBK　BIG5
  - 一　D2BB　A440
  - 丁　B6A1　A442
  - 七　C6DF　A443

Unicode

- Unicode是16比特的字符编码，其支持当前世界上绝大多数的语言
- Unicode给每个字符提供了一个唯一的编码表示，不论是什么平台、程序或语言
- Windows系统(Windows 2000版本)、JAVA、XML、LDAP等已支持Unicode
- Unicode实现了ISO/IEC 10646标准
- 安装附加的语言
  - 开始 > 设置 > 控制面板 > 区域选项
- 字体(font)
  - Full fonts: If you have Microsoft Office 2000, you can get the Arial Unicode MS font, which is the most complete.

Unicode

- JAVA中的字符、字符串、标识符(变量名、方法名和类名称)
- 16比特的Unicode字符
- Unicode字符集(Unicode character set)
  - 用途：国际化
  - \u0000 ~ \uffff，\u是Unicode转义符
  - 当前定义了34,000个Unicode字符
  - \u0020~\u007e等效于ASC-II字符和ISO8859-1(Latin-1)字符 0x20~0x7e

```
char c = '\u5E74';                    //char c = '年' ;
String s1 = "Java\u8BED\u8A00";       //String s1 = "Java语言" ;
String s2 = "\u0030\u0031";           //String s2 ="01";
System.out.println(c );
System.out.println(s1);
System.out.println(s2);
```

第二章 Java语法基础

1. 词法规则
2. 数据类型
3. 常量与变量
4. 运算符和表达式
5. 语句
6. 数组和字符串

运算符 (operator)

- 一元/单目运算符
  - operator op
  - op operator
- 二元/双目运算符
  - op1 operator op2
- 三元/三目运算符
  - op1 ? op2 : op3

注: op表示操作数

运算符 (operator)

1. 算术运算符 (Arithmetic Operators)
2. 关系运算符 (Relational Operators)
3. 逻辑运算符 (Logical Operators)
4. 位运算符 (Bitwise Operators)
5. 移位运算符 (Shift Operators)
6. 条件运算符 (Conditional Operator)
7. 运算符的优先级

运算符 (operator)

- 算术运算符(Arithmetic Operators)
  1. 加法运算符 + "op1 + op2"
  2. 减法运算符 - "op1 - op2"
  3. 乘法运算符 * "op1 * op2"
  4. 除法运算符 / "op1 / op2"
  5. 求模运算符% "op1 % op2" 计算余数

```
int i = 37
int j = 42
double x = 27.475
double y = 7.22
```

```
Adding...
    i + j = 79
    x + y = 34.695
```

```
Computing the remainder...
    i % j = 37
    x % y = 5.815
```

```
Subtracting...
    i - j = -5
    x - y = 20.255
```

```
Multiplying...
    i * j = 1554
    x * y = 198.37
```

```
Dividing...
    i / j = 0
    x / y = 3.8054
```

运算符 (operator)

- 关系运算符(Relational Operators)
  - 比较运算，计算结果“true”或“false”
    1. 大于　　>　“op1 > op2”
    2. 大于等于 >=　“op1 >= op2”
    3. 小于　　<　“op1 < op2”
    4. 小于等于 <=　“op1 <= op2”
    5. 等于　　==　“op1 == op2”
    6. 不等于　!=　“op1 != op2”
  - 优先级
    - (>、>=、<、<=) > (==、!=)
    - 关系运算符低于算术运算符

运算符 (operator)

- 关系运算符(Relational Operators)

```
class Test {
    public static void main(String args[]) {
        int w=25, x=3;
        boolean y = w < x;
        boolean z = w >= w * 2 - x * 9;
        boolean cc = 'b' > 'a';
        System.out.println("w < x = " + y);
        System.out.println("z = " + z);
        System.out.println("cc = " + cc);
    }
}
```

运算符 (operator)

- 逻辑运算符(Logical Operators)
  - 操作数的逻辑关系，计算结果 “true”或 “false”
    - 逻辑与 && “op1 && op2”
      1. 操作数都为真 “true”，结果为真 “true”
      2. 否则结果为假 “false”
    - 逻辑或 || “op1 || op2”
      1. 有一个操作数为真 “true”，结果为真 “true”
      2. 否则结果为假 “false”
    - 逻辑非 ! “! op”
      1. 取反，操作数为真 “true”，结果为真 “false”，反之……

    例: 0 <= index && index < NUM_ENTRIES
  - 优先级
    - (!) > (&&) > (||)
    - (!)>算术运算符>关系运算符>(&&) > (||)

运算符 (operator)

- 位运算符(Bitwise Operators)
  1. 按位取反　~　"~op2"
  2. 按位与　&　"op1 & op2"
  3. 按位或　|　"op1 | op2"
  4. 按位异或　^　"op1 ^ op2"

通常：操作数为整数

运算符 (operator)

- 补码
  - 采用补码表示二进制数
  - 最高位是符号位
    - 正数的符号位是0，

      例，十进制＋42的补码为00101010
    - 负数的符号位是1

      该负数绝对值的补码按位取反，然后加1，为该负数的补码

      例，十进制-42的补码—负数的绝对值42

      绝对值的补码00101010

      按位取反11010101

      加1得11010110

运算符 (operator)

- 按位取反 ~ “~op2”

  对操作数的每一个二进制位进行“取反”操作

  - int a = 42;
  - int aa = ~a;
  - System.out.println(“aa=” + aa);

  42 00101010

  ~

  -43 11010101

运算符 (operator)

- 按位与 & "op1 & op2"

将操作数的对应位逐位进行位逻辑与运算

1 1→1，其余为0

用途

1. 取某一整数中指定的几个二进制位

42 & 15　　00101010

& 00001111

00001010

取后四位

运算符 (operator)

- 按位与 &　"op1 & op2"

用途

2. 将某一整数的最后一位置0

```
   00101011
&  11111110
   00101010
```

43 & ~1

运算符 (operator)

- 按位或 | “op1 | op2”

将操作数的对应位逐位进行位逻辑或运算

有1→1，其余为0

42 | 15

```
  00101010
| 00001111
  00101111
```

用途：将一个整数的某一位或几位置1

运算符 (operator)

- 按位异或 ^ "op1 ^ op2"

  将操作数的对应位逐位进行位异或运算

  对应位不同→1，其余为0

  42 ^ 15

  ```
    00101010
  ^ 00001111
    00100101
  ```

  用途：将一个整数的某一位或几位取反

运算符 (operator)

- 移位运算符(Shift Operators)
  1. 左移 << "op1 << op2"
  2. 右移 >> "op1 >> op2"
  3. 无符号右移 >>> "op1 >>> op2"

运算符 (operator)

- 左移 << "op1 << op2"
  1. 将操作数op1的二进制位向左移op2(正整数)位
  2. 低位补零
  - int a = 42;
  - int aa = a << 2;
  - System.out.println("aa=" + aa);

42 00101010

<<2

168 10101000 相当于 $42*2^2 = 168$

运算速度比乘法快

注意：溢出

运算符 (operator)

- 左移 << "op1 << op2"

溢出

```
byte j = 42;
byte j1 = (byte) (j << 1);          //84
byte j2 = (byte) (j << 2);          //168
byte j3 = (byte) (j << 3);          //336
System.out.println("j1=" + j1);
System.out.println("j2=" + j2);
System.out.println("j3=" + j3);
00101010  //42
01010100  //84
10101000  //-88
01010000  //80
```

运算符 (operator)

- 右移 >> "op1 >> op2"
  1. 将操作数op1的二进制位向右移op2(正整数)位
  2. 高位补零(原为正数)、高位补1(原为负数)

```
int  a = 42;
int aa = a >> 2;
System.out.println("aa=" + aa);
```

42　00101010

>>2

10　00001010 相当于 $42/2^2 = 10.5$

运算速度比除法快

运算符 (operator)

- 无符号右移 >>> "op1 >>> op2"
  1. 将操作数op1的二进制位向右移op2(正整数)位
  2. 高位补零，零扩展(zero-extension)

```
int  a = 42;
int aa = a >>> 2;
System.out.println("aa=" + aa);
```

42　00101010

>>2

10　00001010 相当于 $42/2^2 = 10.5$

运算速度比除法快

实现数的拼接

运算符 (operator)

- 条件运算符
  - op1 ? op2 : op3
  - 若op1为真，则运算结果为op2，否则为op3

例

```
z = a > 0 ? a : -a;
z = a > b ? a : b;
```

自增、自减运算符

- 变量赋值，一元运算符
- 自增运算符(++)、自减运算符(--)
  - int i=5; i++; ++i; i--; --i;
  - “赋值”和“运算”的先后顺序

```
float x =7, y=15, v1, v2;
v1 = x++;
v2 = ++y;
```

```
x=8    y=16
v1=7  v2=16
```

```
int i = 10;
int n = i++%5;
```

i = 11, n = 0

```
int i = 10;
int n = ++i%5;
```

i = 11, n = 1

运算符的优先级

后缀运算符 postfix operators	[] . (params) expr++ expr--
一元运算符 unary operators	++expr --expr +expr -expr ~ !
构造或类型转换creation or cast	new (type)expr
乘法 multiplicative	* / %
加法 additive	+ -
移位 shift	<< >> >>>
关系 relational	< > <= >= instanceof
相等 equality	== !=
按位与 bitwise AND	&
按位异或 bitwise exclusive OR	^
按位或 bitwise inclusive OR	|
逻辑与 logical AND	&&
逻辑或 logical OR	||
条件 conditional	? :
赋值 assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

表达式 (expression)

- 用运算符和括号将操作数连接起来求值的式子
  - 操作数(常量、变量和函数)
  - 算术表达式
  - 关系表达式
  - 逻辑表达式
  - 赋值表达式
  - 复合赋值运算

表达式 (expression)

- 算术表达式
  - 用算术运算符和括号将操作数连接起来，求整数或实数
    - 运算符的优先级和结合性
  - 例
    - int x=20, y=3, z=5;
    - x+y*z　(x+y)*z
    - x*-y
  - 说明
    - 表达式力求简单明了
    - 表达式中的变量必须赋值

表达式 (expression)

- 算术表达式

```
public class Test2{
 public static void main(String[] args) {
      byte b=12;
char c='b';
short s=100;
int i=2400;
int result1=b*c;
int result2=++i-b--+s++;
System.out.println("rsult1="+result1);
System.out.println("rsult2="+result2);
System.out.println("c="+(byte)c);
   }
}
```

表达式 (expression)

- 关系表达式
  - 将两个表达式连接起来的式子
    - 算术表达式、赋值表达式、字符表达式
      - a>b; a+b>b-c; (a=3)>(b=5); 'b'>'a';
    - 返回结果为一个布尔类型的值
  - 例
    - 若 int a=3, b=2, c=1; boolean d, f;
    - d=a>b;
    - f=(a+b)>(b+5);
    - d=a>b>c; ???

表达式 (expression)

- 逻辑表达式
  - 用逻辑运算符将关系表达式和布尔值连接起来的式子
  - 例
    - int x=23, y=98;
    - boolean a = true, b=false, c, d;
    - c=(x>y)&a;
    - d=!a&&(x<=y);

表达式 (expression)

- 逻辑表达式
  - 例：闰年(leap year)

    A year in the Gregorian calendar having 366 days, with the extra day, February 29, intercalated to compensate for the quarter-day difference between an ordinary year and the astronomical year.

    条件符合下例两者之一

    1. 能被4整除，但不能被100整除
    2. 能被4整除，又能被400整除

    (year%4==0 && year%100!=0) || year%400==0

    (year%4!=0) || (year%100==0 && year%400!=0)

表达式 (expression)

- 赋值表达式
  - 用赋值运算符将一个变量和一个表达式连接起来的式子
    - <变量><赋值运算符><表达式>
    - 优先级：赋值运算符 < 算术、关系和逻辑运算符
    - a=5+6;
    - b=c=d=a+5;
    - a=5+c=5;
    - a=(b=4)+(c=6);

表达式 (expression)

- 复合赋值运算
  - 复合赋值运算符
    - +=、 -=、 *=、 /=、%=
    - <<=、>>=、&=、^=、|=
    - <变量><复合赋值运算符><表达式>
    - 例
      - a += b+5; 等价于 a=a+(b+5);
      - a *= b; 等价于 a=a*b;
      - a *= b-c; 等价于 a=a*(b-c);
    - <变量>=<变量><运算符>(<表达式>)

第二章 Java语法基础

1. 词法规则
2. 数据类型
3. 常量与变量
4. 运算符和表达式
5. 语句
6. 数组和字符串

语句 (statement)

- 表达式 + 分号 ";"→表达式语句
  - x = 25;
  - y += a*b+c;
  - a+b;
- 只有分号→空语句
  - i = 5; ; ;
  - 符合语法规则(程序设计的初始阶段)

条件选择语句

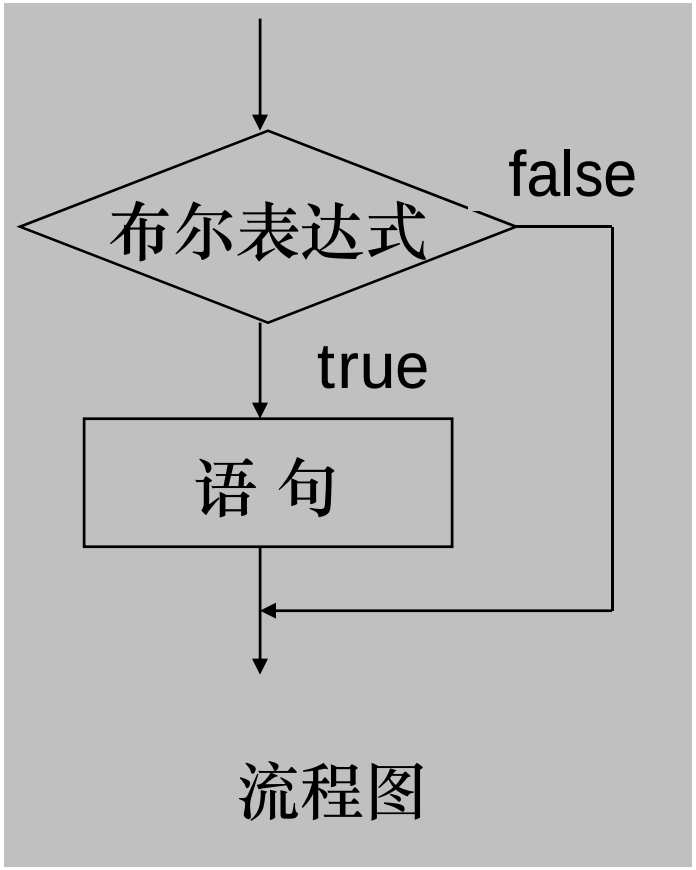
if 语句

1. if语句是一个条件表达式，若条件表达式为真，则执行下面的代码块，否则跳过该代码块
2. 单行代码

```
if (布尔表达式)
        语句;
```

2. 多行代码

```
if (布尔表达式){
        ... ...;
        语句;
}
```



条件选择语句

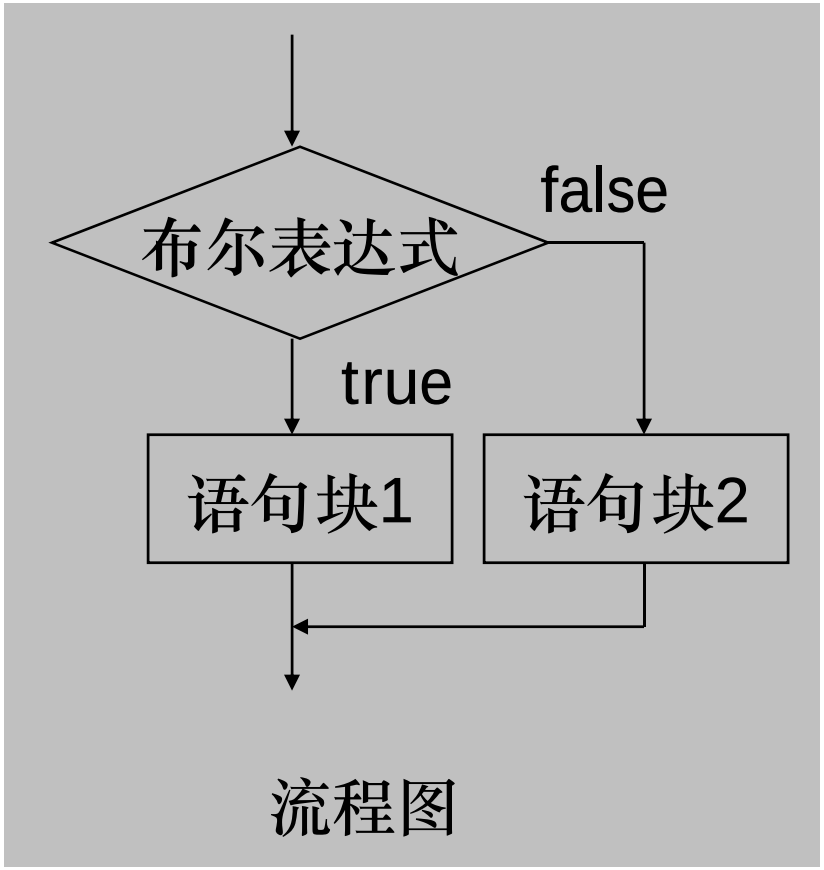
- 示例

```
import java.io.IOException;
public class test3{
    public static void main(String args[]) throws
    IOException
    {

            System.out.println("你喜欢javaa");
            char like = (char)System.in.read();
            if (like =='Y'|| like=='y')
              System.out.println("good");
    }
}
```

条件选择语句

- if-else 语句
  1. 根据判定条件的真假执行不同的操作
  2. 语法

```
if (布尔表达式){
        语句块1;
} else {
        语句块2;
}
```



流程图

条件选择语句

- 示例

```
import java.io.IOException;
class Test {
    public static void main(String args[]) throws IOException {
        System.out.println("请输入你的成绩: ");
        char a = (char)System.in.read();
        char b = (char)System.in.read();
        int score = (a-'0')*10 + b-'0';
        if (score >= 60)
            System.out.println("你及格了！" );
        else
            System.out.println("你没及格了！" );
    }
}
```

```
C:\>java Test
请输入你的成绩:
65
你及格了!

C:\>
```

条件选择语句

默认Java虚拟机:

```
if (a>c) {
    if (c>b)
        System.out.print( c );
    else
        System.out.print(a);
}
```

修改配对关系:

```
if (a>c) {
    if (c>b)
        System.out.print( c );
} else {
        System.out.print(a);
}
```

3. 例 int a=1, b=2, c=3;

```
if (a>c) if (c>b) System.out.print( c );
else System.out.print(a);
```

4. 一定要明确地写上配对符

条件选择语句

- 条件运算符
  1. 三元运算符(ternary operator): "? : "
  2. 表达式1？表达式2：表达式3
  3. 表达式1的结果为布尔型，表达式2和表达式3的类型相同
  - 表达式1→true→表达式2
  - 表达式1→false→表达式3

  Shortcut if-else statement

```
if (表达式1)
        表达式2
else
        表达式3
```

条件选择语句

```
C:\>java Test
请输入三个0～9之间的数
370
max= 7

C:\>
```

- 示例

```
import java.io.IOException;
class Test {
    public static void main(String args[]) throws IOException {
        System.out.println("请输入三个0～9之间的数");
        byte x = (byte)System.in.read();
        byte y = (byte)System.in.read();
        byte z = (byte)System.in.read();
        x -= 48;
        y -= 48;
        z -= 48;
        byte n = x>y? x : y;
        byte m = n > z? n : z;
        System.out.println("max= "+m);
    }
}
```

```
char 0    (48)
char 1    (49)
…    …    ……
x=x-48;
y=y-48;
x=z-48;
```

条件选择语句

- 示例

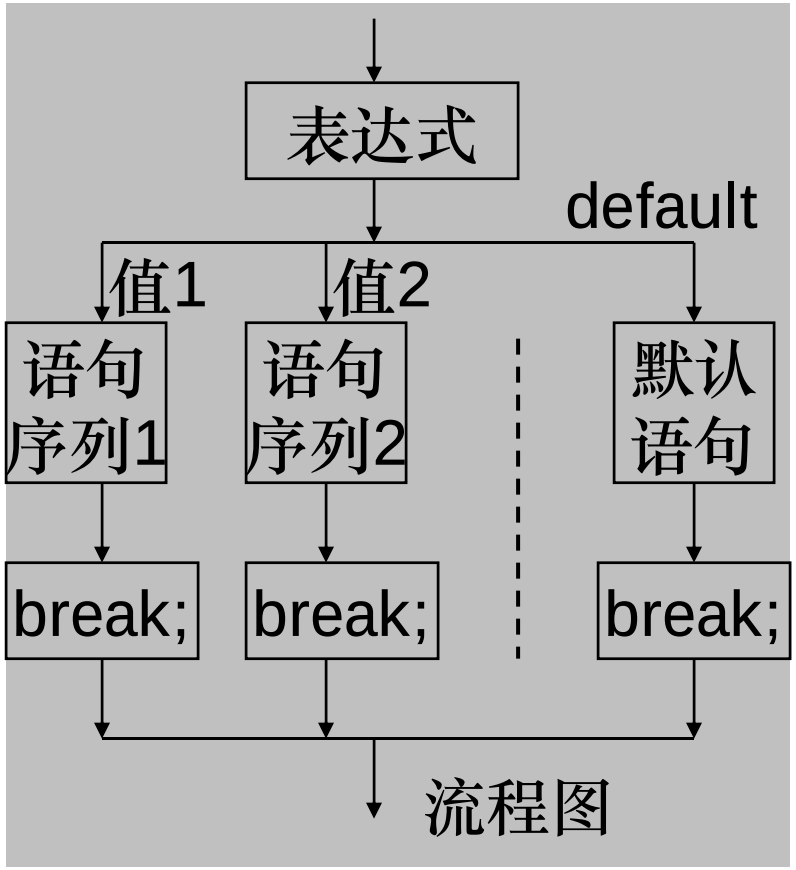
```
import java.io.IOException;
class Test {
    public static void main(String args[]) throws IOException {
        System.out.println("中国足球能否进入世界杯?");
        System.out.println("是(y) 否(n) 不一定(p)");
        char c = (char) System.in.read();
        if (c == 'y')
            System.out.println("Cool");
        else if (c == 'n')
            System.out.println("Bad");
        else if (c == 'p')
            System.out.println("Sorry");
        else
            System.out.println("Input Error");
    }
}
```

逐条if语句进行判断
条件匹配，进入语句体
否则对if语句继续匹配

switch/开关语句

- 根据表达式的结果执行多个操作中的一个
- 语法

```
switch (表达式){
    case 值1: 语句序列;
              [break];
    case 值2: 语句序列;
              [break];
     …… ……    …… ……
    [default: 默认语句;]
}
```



与任一case值不匹配，则进入default语句

switch/开关语句

- 语法

```
switch (表达式){
    case 值1: 语句序列;
              [break];
    case 值2: 语句序列;
              [break];
     …… ……      …… ……
    [default: 默认语句;]
}
```

- 几点注意

1. switch语句表达式的结果必须是byte, char, short, int 类型
2. 表达式的结果依次与每个case子句比较
3. break语句用于跳出switch语句
4. default子句是可选的

switch/开关语句

- 示例 1

```
import java.io.IOException;
class Test {
    public static void main(String args[]) throws IOException {
        System.out.println("中国足球能否进入世界杯?");
        System.out.println("是(y) 否(n) 不一定(p)");
        char c = (char) System.in.read();
        switch (c ) {
            case 'y': System.out.println("Cool"); break;
            case 'n': System.out.println("Bad"); break;
            case 'p': System.out.println("Sorry"); break;
            default: System.out.println("Input Error"); break;
    }
}
```

switch/开关语句

示例 2

```
public class Test {
    public static void main(String[] args) {
        int month = 2, year = 2000;
        int numDays = 0;
        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31; break;
            case 4:
            case 6:
            case 9:
            case 11:
                numDays = 30; break;
            case 2:
                if  (((year % 4 == 0)
                    && !(year % 100 == 0))
                    || (year % 400 == 0) )
                    numDays = 29;
                else
                    numDays = 28;
                break;
        }
        System.out.println("Number of
                Days = " + numDays);
    }
}
```

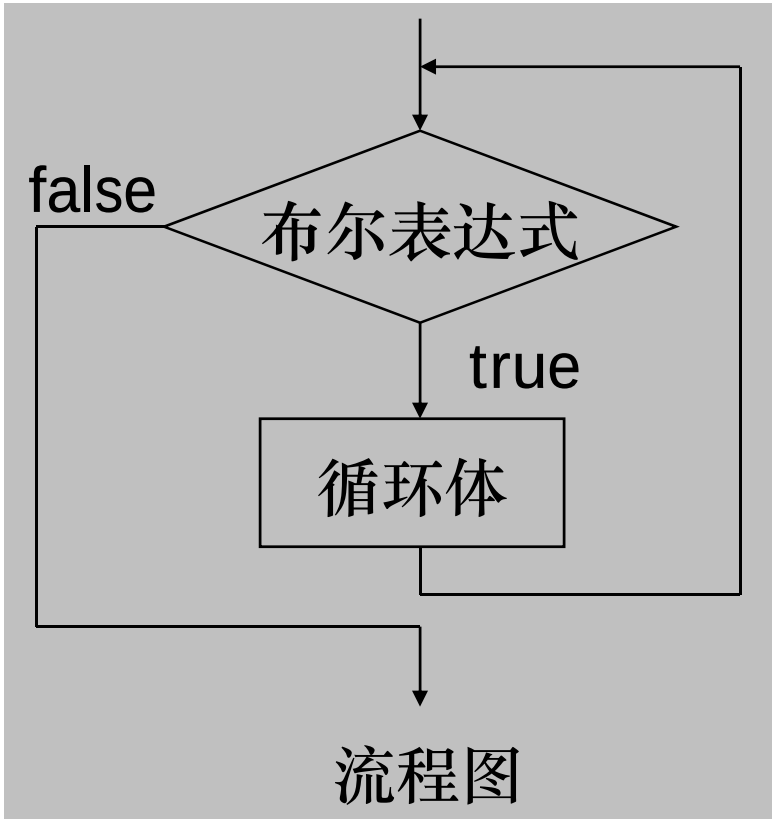
循环控制语句

- 反复执行同一代码块直到满足结束条件
- 组成
  1. 循环的初始状态
  2. 循环体
  3. 迭代因子(计数器的递增或递减)
  4. 控制表达式
- 3种循环语句
  1. while循环
  2. do-while循环
  3. for循环

循环控制语句

- while循环
- 语法

```
while (布尔表达式){
    循环体;
}
```



流程图

循环控制语句

- 示例

```
class Test {
    public static void main(String args[]) {
        int i, sum;
        sum = 0;
         i = 0;
        while (i <= 100) {
            sum += i;
            i++;
        }
        System.out.println("Sum= " + sum);
    }
}
```

控制表达式

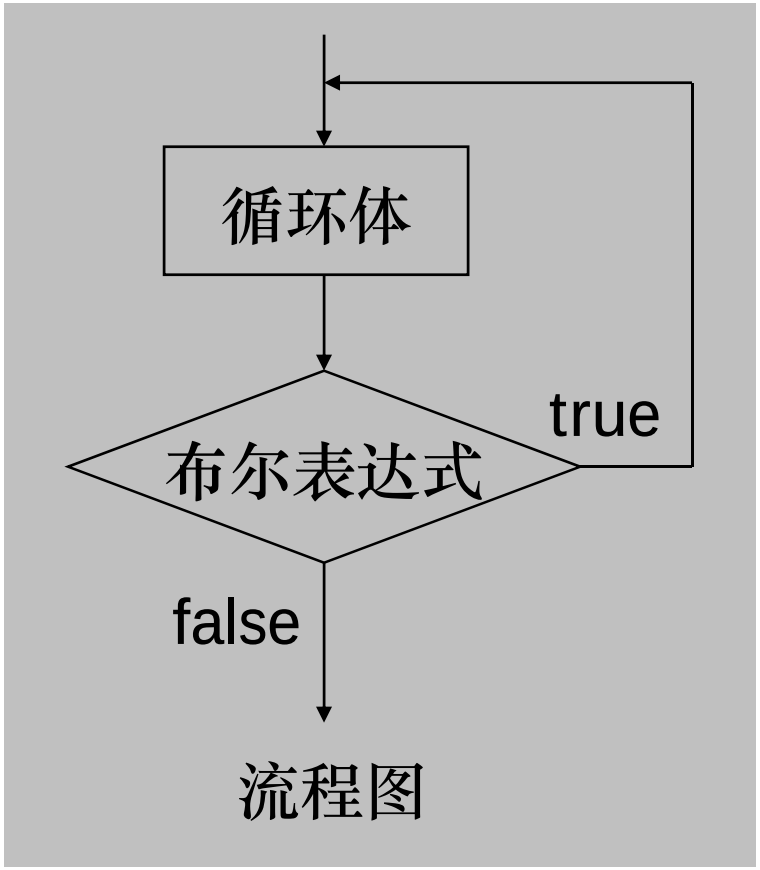
循环体

循环控制语句

- do-while循环
- 语法

```
do {
    循环体;
} while(布尔表达式);
```

  - 先执行循环体
  - 后判断布尔表达式
  - 循环体至少执行一次



循环控制语句

- 示例

```
import java.io.IOException;
class Test {
        public static void main(String args[]) throws IOException {
            char c;
            StringBuffer buffer = new StringBuffer();
            System.out.println("输入一句子以.表示结束");
            do {
                c = (char) System.in.read();
                buffer.append(c);
            } while (c != '.');
            System.out.println("Output =  " + buffer.toString());
        }
}
```

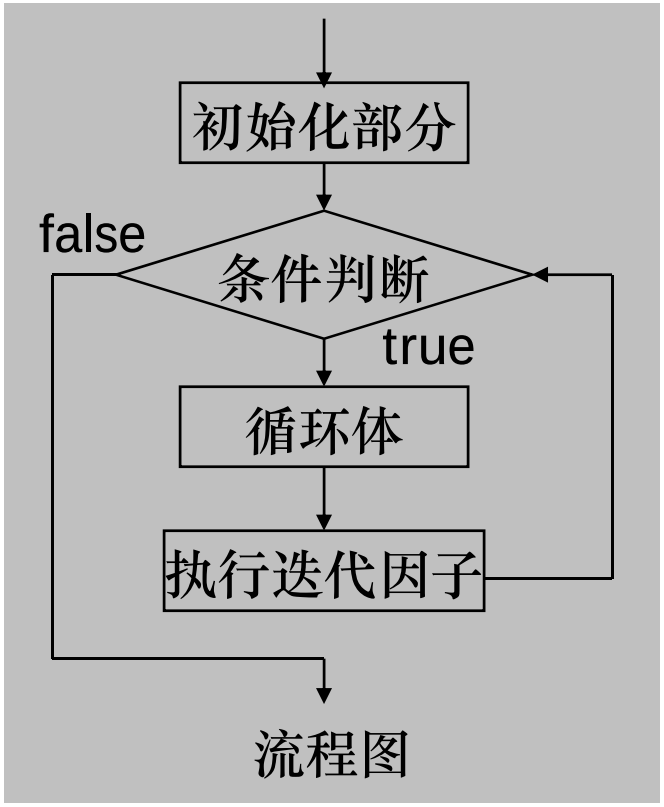
```
C:\>java Test
输入一句子以.表示结束
fdsfs.
Output = fdsfs.

C:\>java Test
输入一句子以.表示结束
fdsf中国.
Output = fdsf???ú.

C:\>
```

循环控制语句

- for循环：最有效、最灵活的循环结构
- 语法

```
for (初始化部分;条件判断部分;迭代因子){
    循环体;
}
```

  - 初始化部分
    - 设置循环变量的初值
  - 条件判断部分
    - 任意布尔表达式
  - 迭代因子
    - 控制循环变量的增减



循环控制语句

- for语句求0~7之间任意数的阶乘

```
import java.io.IOException;
class Test {
    public static void main(String args[]) throws
    IOException {
        int i, n, sum=1;
        System.out.println("Please inpu
        n = System.in.read();
        n -= 48;
        for (i = 1; i <= n; i++)
            sum *= i;
        System.out.println(n + "!= " + sum);
    }
}
```

```
D:\>java Test
Please input(0~7):
5
5!=120
```

1.能否算0~9
2.能否算12, 134?

循环控制语句

- for循环的几点注意
  - 初始化部分和迭代因子可以包含多个语句，以“,”分开

```
for (int i=0, j=10; i<j; i++, j--) {
……
}
```

  - 初始化部分、条件判断部分和迭代因子可以为空语句，但以“;”分开，表示无限循环

```
for ( ; ; ) {   // infinite loop
   …
}
```

循环控制语句

- 如果一个人出生于1970年，那么他这一辈子能有几个闰年(以70岁为寿命长度)

```
int length = 70;
int firstYear = 1970;
int year;
for (int i =0; i < length; i++) {
    year = firstYear + i;
    if((year%4==0&&year%100!=0)||
       year%400==0)
           System.out.println(year);
}
```

1972
1976
1980
1984
1988
1992
1996
2000
2004
2008
2012
2016
2020
2024
2028
2032
2036

循环控制语句

- 循环的嵌套
  - 一个循环体内包含另一个完整的循环结构
  - 嵌套的层次多，多重嵌套
  - while循环、do-while循环、for循环相互嵌套

循环控制语句

- 求 1!+2!+3!+... ... +9!

输入0的结果!

```
C:\>java Test
Please input(0~9):
4
各阶乘之和为: 33
```

```
import java.io.IOException;
class Test {
    public static void main(String args[]) throws IOException {
        int n, sum, total=0;
        System.out.println("Please input(0~9):");
        n = System.in.read();
        n -= 48;
        for (int j = 1; j <= n; j++) {
            sum = 1;
            for (int i = 1; i <=j; i++)
                sum *= i;
            total +=sum;
        }
        System.out.println("各阶乘之和为: " + total);
    }
}
```

求阶乘

阶乘之和

跳转/转向语句

- 将程序的执行跳转到其他部分的语句
- break: 跳出(中止)循环
- continue: 结束本次循环
- return: 方法返回
- throw: 抛出异常(Exception)

若只有一层循环，带标号和不带标号作用相同
若存在循环嵌套，两者作用不同

- break语句
  - break语句用以中断当前执行的循环语句(for、do-while、while)或switch语句
  - 两种形式
    - 不带标号的break语句
      - 表达形式：　break;
      - 从本层循环中跳出
    - 带标号的break语句
      - 表达形式：
      - 从整个程序块中跳出

标号:
程序块
(break 标号;)
……

跳转语句

- 不带标号的break语句

```
class Test {
    public static void main(String args[])  {
        for (int j = 1; j <6; j++) {
            if (j ==3)
                break;
            System.out.print("j=" + j);
        }
        System.out.println(" stop");
    }
}
```

j=1 j=2 stop

若只有一层循环，带标号和不带标号作用相同
若存在循环嵌套，两者作用不同

- 带标号的break语句

```
class Test {
    public static void main(String args[]) {
        int j, k;
        Rep:
        for (j = 8; j >1; j--) {
            for (k=1; k<=9; k++) {
                if (k ==5)
                    break;
                if (j==6)
                    break Rep;
                System.out.print(j*k+“ ”);
            }
            System.out.println(“<>”);
        }
    }
}
```

```
8 16 24 32 <>
7 14 21 28 <>
```

若只有一层循环，带标号和不带标号作用相同
若存在循环嵌套，两者作用不同

- continue语句
  - continue语句用以结束循环语句(for、do-while、while)的本次循环
  - 两种形式
    - 不带标号的continue语句
      - 表达形式：continue;
      - 结束本次循环，即跳过continue语句本层循环体的条件测试部分
    - 带标号的continue语句
      - 表达形式：
      - 跳至标号所指语句块的条件测试部分继续执行
    - 注意与break语句的比较

标号:
程序块
(continue 标号;)
……

跳转语句

- 不带标号的continue语句

```
class Test {
    public static void main(String args[]) {
        for (int k = 6; k >=0; k-=2) {
            if (k == 4)
                continue;
            System.out.print("k=" + k + "\t");
        }
    }
}
```

```
k=6 k=2 k=0
```

若只有一层循环，带标号和不带标号作用相同
若存在循环嵌套，两者作用不同

- 带标号的continue语句

```
class Test {
    public static void main(String args[]) {
        iLoop:
        for (int i=1; i <=5; i++) {
            for (int j=1; j<=5; j++) {
                int p = i*j;
                if (j==3)
                    continue;
                if (i==2)
                    continue iLoop;
                if (p >=10) System.out.print(p +" ");
                else System.out.print(p + "  ");
            }
            System.out.println();
        }
    }
}
```

```
1  2  4  5
3  6  12 15
4  8  16 20
5  10 20 50
```

其他语句

- import/包含语句
  - 引入程序所需要的类
  - import java.io.*;
  - import java.applet.Applet;
- package/打包语句
  - 指明所定义的类属于哪个包
  - 通常作为源程序的第一条语句
  - package test;

第二章 Java语法基础

1. 词法规则
2. 数据类型
3. 常量与变量
4. 运算符和表达式
5. 语句
6. 数组和字符串

概念

- 数组是一组同类型的变量或对象的集合
  - 数组的类型可以是基本类型，或类和接口
  - 数组中每个元素的类型相同
  - 引用数组元素通过数组名[下标]
  - 数组下标(数组的索引)从0开始
- 数组是一种特殊的对象(Object)
  - 定义类型(声明)
  - 创建数组(分配内存空间)：new
  - 释放(Java虚拟机完成)
- 一维数组、多维数组

一维数组

- 一维数组的元素只有一个下标变量
  - 例：A[1], c[3]
- 一维数组的声明
  - 方法1： 类型 数组名[];
    - String args[]; int a[]; double amount[]; char c[];
  - 方法2： 类型[] 数组名；
    - String[] args; int[] a; double[] amount; char[] c;
  - 注意
    - 类型是数组中元素的数据类型(基本和构造类型)
    - 数组名是一个标识符
    - 数组声明后不能被访问，因未为数组元素分配内存空间

```
double[] d;
System.out.println(d[0]);

variable d might not have been initialized
System.out.println(d[0]);
                   ^
1 error
```

一维数组

- 数组的创建
  - 用new来创建数组
  - 为数组元素分配内存空间，并对数组元素进行初始化
  - 格式：数组名 = new 类型[数组长度]
  - 例：a = new int[3];
  - 声明和创建的联用：int[] a = new int[3];
  - 默认赋初值
    - 整型→初值为0　　int[] i = new int[3];
    - 实型→初值为0.0　　float[] f = new float[3];
    - 布尔型→初值为false　boolean[] b = new boolean[3];
    - 字符型→初值为\u0000(不可见) char[] c = new char[3];

一维数组

```
class Test {
        public static void main(String args[]) {
            int[] i = new int[3];
            float[] f = new float[3];
            boolean[] b = new boolean[3];
            char[] c = new char[3];
            for (int j = 0; j < i.length; j++)
                        System.out.println(i[j]);
            for (int j = 0; j < f.length; j++)
                        System.out.println(f[j]);
            for (int j = 0; j < b.length; j++)
                        System.out.println(b[j]);
            for (int j = 0; j < c.length; j++)
                        System.out.println(c[j]);
        }
}
```

```
C:\>java Test
0
0
0
0.0
0.0
0.0
false
false
false


C:\
```

一维数组

- 一维数组的初始化
  - 为数组元素指定初始值
  - 方式一：声明和创建数组后对数组初始化

```
class Test {
    public static void main(String args[]) {
        int a[] = new int[5];
        System.out.println("\t输出一维数组a: ");
        for (int i = 0; i < 5; i++) {
            a[i] = i +1;
            System.out.println("\ta["+i+"]="+a[i]);
        }
    }
}
```

a.length

一维数组

- 一维数组的初始化
  - 方式二：在声明数组的同时对数组初始化
    - 格式：类型 数组名[] = {元素1[, 元素2 ……]};
    - int a[] = {1, 2, 3, 4, 5};

```
class Test {
    public static void main(String args[]) {
        int a[] = {1,2,3,4,5};
        System.out.println("\t输出一维数组a: ");
        for (int i = 0; i < 5; i++)
            System.out.println("\ta["+i+"]="+a[i]);
    }
}
```

一维数组

- 数组的赋值
  - 数组的整体赋值
  - 用java.lang.System类的方法进行数组复制

一维数组

```
int a[] = {2, 4, 6, 8, 0};
int b[];  int c[] = {1, 3, 5, 7};
```

- 数组整体赋值

```
int a[] = {2, 4, 6, 8};
int b[];  int c[] = {1, 3, 5, 7, 9};
```

```
class Test {
    public static void main(String args[]) {
        int a[] = {2, 4, 6, 8};
        int b[];    int[] c = {1, 3, 5, 7};
        b = a;      c = a;
        for (int j = 0; j < a.length; j++)
                System.out.print(a[j] + " ");
        System.out.println();
        for (int j = 0; j < b.length; j++)
                System.out.print(b[j] + " ") ;
        System.out.println();
        for (int j = 0; j < c.length; j++)
                System.out.print(c[j]);
    }
}
```

```
C:\>java Test
2 4 6 8
2 4 6 8
2 4 6 8
C:\>
```

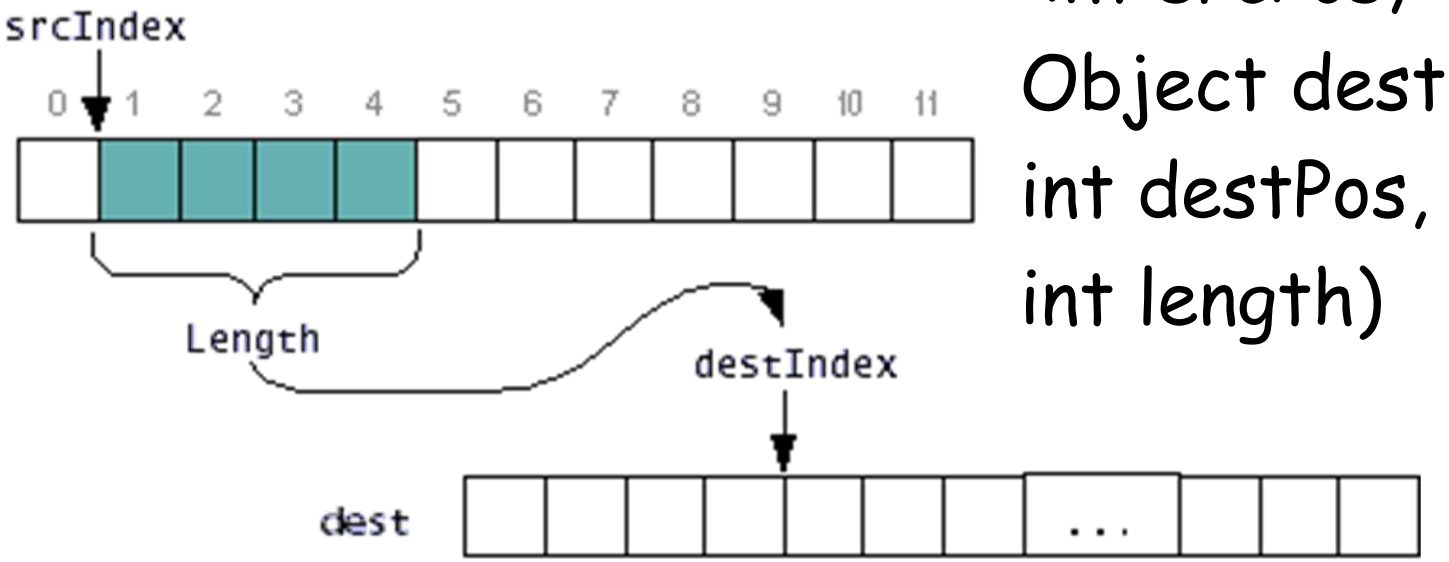
一维数组

- 数组的复用(reuse)

```
public class Test {
    public static void main(String[] args) {
        int[] array = { 32, 87, 3, 589, 12, 1076, 2000 };
        for (int i = 0; i < array.length; i++)
            System.out.print(array[i] + " ");
        array = new int[4];
        for (int i = 0; i < array.length; i++)
            array[i] = i + 1;
        System.out.println();
        for (int i = 0; i < array.length; i++)
            System.out.print(array[i] + " ");
    }
}
```

一维数组

- 一维数组的数组复制
  - java.lang.System类的方法
  - public static void arraycopy(Object src,
    int srcPos,
    Object dest,
    int destPos,
    int length)



一维数组

一维数组的数组复制

```
class Test {
   public static void main(String args[]) {
      int a[] = {2, 4, 6, 8};
       int b[];
       int[] c = {1, 3, 5, 7, 9};
       b = a;
       System.arraycopy(a, 1, c, 0, 3);
       System.out.print("数组a: ");
       for (int i = 0; i < a.length; i++)
           System.out.print(a[i] + " ");
       System.out.println();
       System.out.print("数组b: ");
       for (int i = 0; i < b.length; i++)
           System.out.print(b[i] + " ");
       System.out.println();
       System.out.print("数组c: ");
       for (int i = 0; i < c.length; i++)
           System.out.print(c[i] + " ");
       System.out.println();
   }
}
```

```
数组a: 2 4 6 8
数组b: 2 4 6 8
数组c: 4 6 8 7 9
```

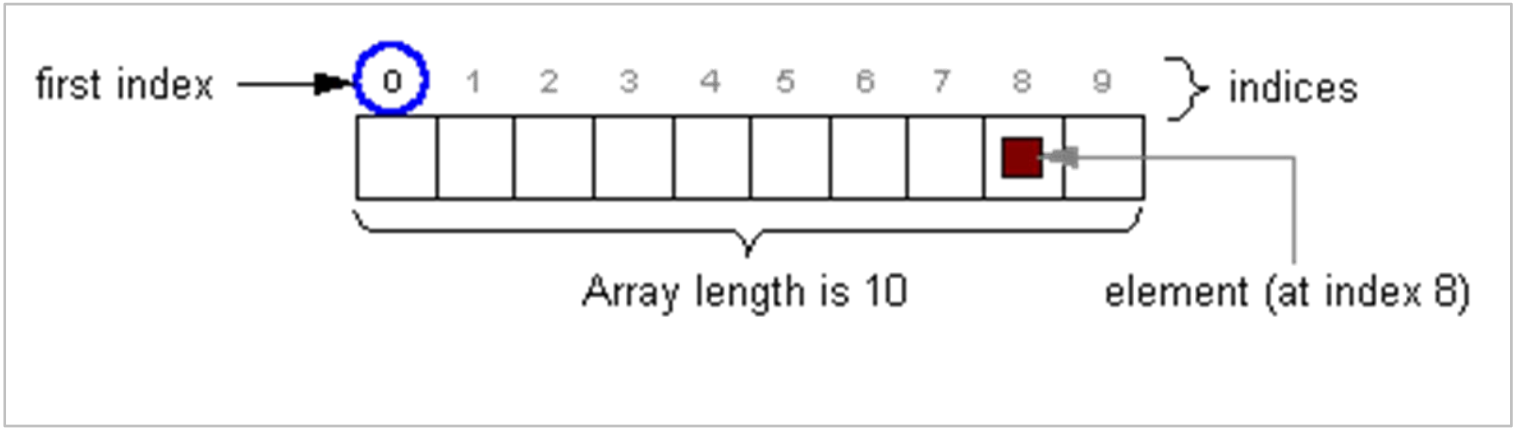
一维数组

- 数组的排序

```
public class Test {
    public static void main(String[] args) {
        int[] array = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
        for (int i = array.length; --i >= 0; ) {
            for (int j = 0; j < i; j++) {
                if (array[j] > array[j+1]) {
                    int temp = array[j];  array[j] = array[j+1];
                    array[j+1] = temp;
                }
            }
        }
        for (int i = 0; i < array.length; i++)
            System.out.print(array[i] + " ");
    }
}
```

一维数组

- 小结
  - 类型相同、数量确定的存储结构
  - 用下标访问数组中任一个元素，数组名[下标]



  - 声明、创建(new)、初始化/赋值

```
int d[];
System.out.println(d[0]);
System.out.println(d.length);

variable d might not have been initialized
System.out.println(d[0]);
                   ^
System.out.println(d.length);
                   ^
```

多维数组

- 数组的数组
  - Arrays of Arrays
  - 例：表格(行和列)

姓 名	期中考试	期末考试	总 分
学生A	68	70	69
学生B	80	85	84
学生C	75	90	86

- 以二维数组为例

多维数组

- 二维数组的声明
  - 类型 数组名[][]， 例 int a[][];
  - 数组声明后不能被访问，因未为数组元素分配内存空间
- 二维数组的创建
  - 方法一：直接分配空间(new)

    例 int a[][] = new int[2][3];

    a[0][0] a[0][1] a[0][2]

    a[1][0] a[1][1] a[1][2]

    两个一维数组，每个数组包含3个元素

多维数组

- 二维数组的创建
  - 方法二：从最高维开始，为每一维分配空间

例 int c[][] = new int[2][];

```
c[0] = new int[4];
c[1] = new int[3];
c[0][0]  c[0][1] c[0][2] c[0][3]
c[1][0]  c[1][1] c[1][2]
```

  - 注：为数组分配空间需指定维数大小，至少最高维(最左边)大小
  - 错误：int b[][] = new int[][];

多维数组

- 二维数组的初始化
  - 对每个元素单独进行赋值
  - 声明数组的同时初始化
- 对数组元素的引用
  - 数组名[下标1][下标2]
  - 下标为非负的整型常数0~

多维数组

- 二维数组的初始化
  - 每个元素单独进行赋值

```
class Test {
    public static void main (String args[]) {
        int a[][] = new int[3][3];
        a[0][0]=1;a[1][1]=1;a[2][2]=1;
        System.out.println("数组a: ");
        for (int i=0; i< a.length; i++){
            for (int j=0; j<a[i].length; j++)
                System.out.print(a[i][j]+" ");
            System.out.println();
        }
    }
}
```

```
1  0  0
0  1  0
0  0  1
```

最高维数组长度

多维数组

- 二维数组的初始化
  - 声明数组的同时初始化

例 int a[][] = {{1,2,3}, {3,4,5}};

a[0][0]=1 a[0][1]=2 a[0][2]=3

a[1][0]=3 a[1][1]=4 a[1][2]=5

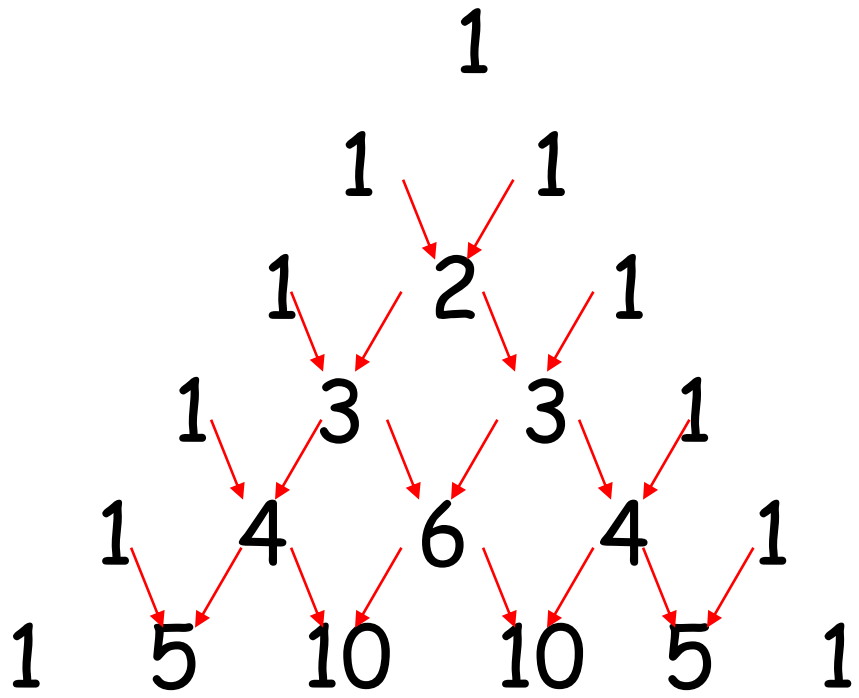
例 String[][] cartoons =

```
{
{ "Flint","Fred","Wim","Pebbles","Dino"},
{ "Rub","Barn","Bet", "Bam"},
{ "Jet","Geo","Jane","Elroy","Judy","Rosie", "Astro"},
{ "Sco","Sco",  "Shag", "Velma", "Fred", "Dap" }
};
```

int i = cartoons.length

多维数组

- 杨辉三角形



  - 三角形腰上的数为1
  - 其他位置的数为其上一行相邻两个数之和

多维数组

- 杨辉三角形
  - 用二维数组描述杨辉三角形

a[1][1]
a[2][1] a[2][2]
a[3][1] a[3][2] a[3][3]
a[4][1] a[4][2] a[4][3] a[4][4]
a[5][1] a[5][2] a[5][3] a[5][4] a[5][5]

  - 第1列元素为1
  - 对角线上的元素为1
  - 其他元素a[i][j]=a[i-1][j-1]+a[i-1][j]

纵轴为i, 横轴为j

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
```

多维数组

- 杨辉三角形

```
class Test {
    public static void main(String args[]){
        int n=6, indent, i, j;
        int a[][] = new int[n][n];
        a[1][1]=1;
        for (i=2; i<n; i++) {
                a[i][1]=1;a[i][i]=1;
                for(j=1; j<i; j++)
                        a[i][j]=a[i-1][j-1]+a[i-1][j];
        }
```

纵轴为i, 横轴为j

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
0	1	2	1	0	0
0	1	3	3	1	0
0	1	4	6	4	1

多维数组

- 杨辉三角形

```
indent=25;
 for (i=1; i<n; i++) {
        for(int k=1; k<=indent; k++)
                System.out.print(“ ”);
        for(j=1; j<=i; j++)
                System.out.print(a[i][j]+“  ”);
        System.out.println();
        indent=indent-2;
                }
            }
        }
```

```
        1
      1  1
    1  2  1
  1   3   3  1
1   4    6   4  1
```

纵轴为i, 横轴为j

0	0	0	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
0	1	2	1	0	0
0	1	3	3	1	0
0	1	4	6	4	1

多维数组

- 例1

```
class Test {
    public static void main(String args[]) {
            int a[][];
            System.out.println(a.length);
    }
}
```

```
C:\>javac Test.java
Test.java:4: variable a might not have been initialized
                        System.out.println(a.length);
                                           ^
1 error
```

多维数组

- 例2

```
class Test {
    public static void main(String args[]) {
            int a[][] = new int[][];
            System.out.println(a.length);
    }
}
```

```
C:\>javac Test.java
Test.java:3: '{' expected
        int a[][] = new int[][];
                                ^
1 error
```

多维数组

- 例3

```
class Test {
    public static void main(String args[]) {
            int a[][] = new int[9][];
            System.out.println(a.length);
    }
}
```

```
C:\>javac Test.java

C:\>java Test
9
```

数组的界限

- 起点和终点
  - 数组的长度：数组名.length
  - 起点： 数组名[0]
  - 终点： 数组名[length-1]

int i = {4, 56, 78, 9, 34};

i.length → 5

i[0] → 4

i[length-1]=i[4]→34

i[a] 若a>4 则???

命令行参数

- JAVA应用程序的主方法(程序的入口)
  - public static void main (String args[]) {...}
  - public static void main (String[] args) {...}
- 命令行参数
  - 在启动JAVA应用程序时一次性地传递多个参数
  - C:\java 类名 参数 参数 .....
  - 空格将参数分开
  - 若参数包含空格，用双引号引起来

命令行参数

- 示例 1

```
class Test {

    public static void main(String[] args) {
        int len = args.length;
        System.out.println(len);
        for (int i = 0; i < len; i++)
            System.out.println(args[i]);
    }

}
```

```
C:\>java Test s1 s2
2
s1
s2

C:\>
```

```
C:\>java Test
0

C:\>
```

```
C:\>java Test “s1 s2”
1
s1 s2

C:\>
```

命令行参数

- 示例 2

```
C:\>java Test s1 “s2” “s3”
args[0]=s1
args[1]=s2
args[2]=s3

C:\>
```

```
class Test {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(“args[“+i+”]=“+args[i]);
    }
}
```

命令行参数

- 命令行参数的转换
  - 传递字符串数组
  - 向JAVA应用程序传递数值
    - byte、short、int、long、double、float

```
C:\>java Test 1 2
3

C:\>
```

String args[i] → byte
short
int
long
double
float

命令行参数

- 命令行参数的转换
  - java.lang.Byte类
    - public static byte **parseByte**(String s) throws NumberFormatException
  - java.lang.Integer类
    - public static int **parseInt**(String s) throws NumberFormatException
  - ……

命令行参数

```
C:\>java Test 1 2
3

C:\>
```

- 示例

```
import java.lang.NumberFormatException;
class Test {
    public static void main(String[] args) throws
        NumberFormatException {
        int sum = 0;
        for (int i = 0; i < args.length; i++)
                sum = sum + Integer.parseInt(args[i]);
        System.out.println(sum);
    }
}
```

第二章 结束！