第二章　Java 编程基础

2.1 Java的关键字、标识符

2.1.1 标识符

Java中的包名、类名、接口名、方法名、对象名、常量名、变量名等统称为标识符。 Java语言规定：

1. 标识符必须是字母、下划线(_)、美元符号($)**开头**，即**数字不能作为标示符的开头！** 后续字符除了这三类之外，还可以是数字及Unicode字符集中序号大于0xC0的所有符号(包括中文字符、日文字符、韩文字符、阿拉伯字符等)。除此之外的所有符号，比如#，+，-，*，&都不能用来作为标示符

2. Java标识符 (**严格区分大小写**）

3. 关键字不能单独作为标识符，可作为标识符的一部分，比如class。

请判断下列标识符哪些是合法的，哪些是不合法的

HelloJavaWorld	_xy3c	$histk	Ad_gs
Hello&Java	Xy-3c	3histk	Ad gs

2.1 Java的关键字、标识符

2.1.1 标识符

在Java中，有一些约定俗成的命名规则，熟知并使用这些规则有助于你读懂别人的程序、让自己的程序更规范、大方：

1.包名通常为小写，

2.类名、接口名的首字母都为大写；

3.方法名的第一个字母通常是小写；

4.当类名、接口名、方法名由多个单词构成时，后面各单词的首字母通常大写;

5.用户声明的变量名、一个类的对象名通常为小写。

2.1 Java的关键字、标识符

2.1.2 标识符

此外，关键字不能单独作为标识符，可作为标识符的一部分！

Java 关键字……

2.1 Java的关键字、标识符

2.1.2 关键字

关键字又称保留字，是语言中具有特定含义的单词，用户在编写程序时只能按系统规定的方式来使用它们。Java中的关键字有50多个，按用途可划分为几个组别：

1.标识数据类型、对象：boolean、byte、char、double、false、float、int、long、new、null、short、true、void、instanceof；

2.语句控制：break、case、catch、continue、default、do、else、for、if、return、switch、try、while、finally、throw；

3.修饰功能：this、super、abstract、final、native、private、protected、public、static、synchronized、transient、volatile；

4.类、接口、方法、包和异常等的声明、定义要求：class、extends、implements、interface、package、import、throws；

2.2 Java的数据类型

Java是一种强类型语言，也就是说，Java程序中的数据要先声明其数据类型，再使用。Java的数据类型可分为基本数据类型和引用类型两大类，具体如下：

- 数据类型
  - 基本数据类型
    - 数值型
      - 整数类型(byte, short, int, long)
      - 浮点类型(float, double)
    - 字符型(char)
    - 布尔型(boolean)
  - 引用数据类型
    - 类(class)
    - 接口(interface)
    - 数组

2.2 Java的数据类型

2.2.1 基本数据类型

- 基本数据类型：预先定义的，长度固定，不能再分的类型

2.2 Java的数据类型

2.2.1 基本数据类型

位数固定！想想为什么？

类型名称	关键字	占用字节及位数	数值范围
字节型	byte	1字节(即8位)	-2^7～2^7-1 (即:-128～127)
短整型	short	2字节(即16位)	-2^{15}～$2^{15}-1$ (即:-32768～32767)
整型	int	4字节(即32位)	-2^{31}～$2^{31}-1$ (即:-21亿～21亿)
长整型	long	8字节(即64位)	-2^{63}～$2^{63}-1$ (即:-922亿亿～922亿亿)
单精度浮点数	float	4字节(即32位)	绝对值：3.4e-038 ～3.4e+038
双精度浮点数	double	8字节(即64位)	绝对值：1.7e-308 ～1.7e+308
字符型	char	2字节(即16位)	0～$2^{16}-1$ (即0 ～65535)
布尔型	boolean	1字节(即8位)	true 和 false

2.2 Java的数据类型

2.2.1 基本数据类型

整数类型

包括byte、short、int、long，表示无小数部分的数字，包括：正整数、零、负整数

字符型

即char，用来表示通常意义上字符、文本

浮点类型

包括float、double，表示有小数部分的数字

布尔型

即boolean，表示逻辑判断的“真”、“假”

2.2 Java的数据类型

2.2.1 基本数据类型

- 整型：
  - 进制：
    - 八进制：0开头
    - 十进制
    - 十六进制：0x或0X开头
    - 二进制：0b100
  - 子类型：
    - 注意：Java中没有无符号类型
    - (DecimalForm.java)

2.2 Java的数据类型

2.2.1 基本数据类型

- 浮点型数据
  - 表示形式：
    - 标准浮点形式：例如，3.17582
    - 科学计数形式：例如，3.5E+8
  - 子类型：
    - float 单精度浮点数，占4字节
    - double 双精度浮点数，占8字节
  - 子类型常量的区分：
    - 数字后＋D/F来分别表示double、float

2.2 Java的数据类型

2.2.1 基本数据类型

- 布尔型（boolean）
  - 取值：false（假）和true（真）
  - Java中整型和boolean型不能相互转换
    - 因此，下列式子是错误的！
    - if (x=0)　　//编译错误

2.2 Java的数据类型

2.2.1 基本数据类型

- 字符型char：
  - 单引号括起
  - Java中char用于表示Unicode编码表中的字符，占2个字节（16位），最多可表示65536个字符，目前使用的约有35000个字符——使得Java语言的多语种处理能力大大加强，为其在不同语言的平台间实现平滑移植奠定了基础

2.2 Java的数据类型

2.2.1 基本数据类型

- 字符型char：
  - C/C++中默认使用ASCII编码，只占1字节，可允许128个字符，扩展后允许256个字符。
  - Unicode编码表示范围：\u0000～\uFFFF，用于表示0～65535之间的编码
  - 常用的转义字符

2.2 Java的数据类型

2.2.1 基本数据类型

字符型还有一种常用方式就是转义字符，用来改变一些字符的原有含义，实现特定功能。格式为：’\特殊字符’，常用的转义字符如表所示(与C++类似)：

转义字符	功能	转义字符	功能
\’	输出单引号’	\”	输出双引号”
\\	输出反斜杠\	\b	退格(backspace)
\n	换行	\r	回车，光标移至当前行开始
\t	光标移至下一个制表位(tab)		

2.2 Java的数据类型

Java是一种强类型语言，也就是说，Java程序中的数据要先声明其数据类型，再使用。Java的数据类型可分为基本数据类型和引用类型两大类，具体如下：

- 数据类型
  - 基本数据类型
    - 数值型
      - 整数类型(byte, short, int, long)
      - 浮点类型(float, double)
    - 字符型(char)
    - 布尔型(boolean)
  - 引用数据类型
    - 类(class)
    - 接口(interface)
    - 数组

2.2.3 引用数据类型

万物皆对象”是Java的一个重要观点，前面介绍的8种基本数据类型是不能用作“对象”来处理的，但可将它们转换为对应的对象类型，即：Byte、Short、Integer、Long、Float、Double、Character、Boolean，它们被称为基本类型的**包装类**(你是否发现它们的命名规律?)，这些类中的大多数都定义了MIN_VALUE和MAX_VALUE来表示对应的基本类型的数值范围。此外，还定义了许多有用的方法，有兴趣的可以查阅Java API文档。

2.2.3 常量

顾名思义，常量是指在程序运行过程中，其值保持不变的量。常量除了前面说过的数值常量、字符常量、布尔常量之外，有时还可以用符号来表示(这称为符号常量)。符号常量要使用关键字final来定义，

格式为：final 数据类型 常量名=值

例如：final double PI=3.1415926;

按照Java编程规范要求，符号常量名通常为大写，且多个单词之间用下划线连接。如果是类常量，还要在数据类型前加上static关键字。查阅Java API文档，你会发现许多类的符号常量都是用这种方式来命名的。常量的调用格式是类名.常量名 。

2.2.3 常量

例 2.2 显示类的静态常量(通过 类名.常量名 方式来访问)。代码如下：

```
public class MinMaxValueTest {
        public static void main(String args[]) {
                System.out.println("int型的最小值：" + Integer.MIN_VALUE);
                System.out.println("int型的最大值：" + Integer.MAX_VALUE);
                System.out.println("int型数据所占位数：" + Integer.SIZE);
        }
}
```

运行结果

```
int型的最小值：-2147483648

int型的最大值：2147483647

int型数据所占位数：32
```

2.2.4 变量

与常量不同，变量是指程序执行过程中，其数值可以改变的量。变量包括变量名和变量值两部分，变量名起标识作用，变量值是计算机内存单元存放的具体内容。我们常用“铁打的营盘流水的兵”来形容部队的建制特点，这里的变量名相当于“铁打的营盘”，是不变部分；变量值类似“流动的兵”，是可变部分。变量是程序的重要组成部分，应熟练掌握才行。

2.2.4 变量

1 变量的声明

Java中的变量遵循“先声明，再使用”的原则，通过声明来指定变量的数据类型和名称，变量的值可以在后续语句中赋予或改变。声明格式：

数据类型 变量名；或 数据类型 变量名1, …, 变量名k ;

例如：double salary;

boolean done;

String 姓名;

int studentNumber, peopleNumber;

从程序的可读性角度来看，不建议将多个变量的声明写在同一行上。

2.2.4 变量

2 变量的赋值

赋值前，首先要检查赋值号两端的数据类型是否一致。当类型不一致时，如果符合类型自动转换条件，则赋值自动完成；否则，必须进行强制类型转换，不然会造成编译错误。

```
double   d1=2.5d;
int a1=10;

double d2=20;        //double d2=a1;
int  a2=(int)10.0;   //int a2=(int)d1;
```

2.2 Java数据类型的转换

2.2.2 数据类型的转换

如果是两种相容的数据类型(如同为数值型)，则它们之间可以进行转换。转换的方式有两种：

1.自动类型转换(系统自动完成) ：从取值范围小的类型向取值范围大的类型转换(如：byte,short,char—> int —> long—> float —> double)，这种转换是自动进行，如：float f=10;

2.2 Java数据类型的转换

2.2.2 数据类型的转换

如果是两种相容的数据类型(如同为数值型)，则它们之间可以进行转换。转换的方式有两种：

1.自动类型转换(系统自动完成);

2.显式类型转换（强制转换）：从取值范围大的类型向取值范围小的类型转换，需要进行强制转换，

格式：目标数据类型 变量=(目标数据类型) 值;

```
double y=5.6;
int x=(int)(y+7.8d-20);
```

参见SayHello.java

2.2 Java数据类型的转换

2.2.1 数据类型的转换

例2.1 char类型与int类型相互转换。代码如下：

```
public class UnicodeTest {
        public static void main(String[] args) {
                char c = '大';
                System.out.println("\'" + c + "\'的Unicode编码： " + (int) c);
                int num = 23398;
                System.out.println("Unicode编码为" + num + "的字符是：\'" +
(char) num + "\'");
        }
}
```

运行结果

```
'大'的Unicode编码： 22823

Unicode编码为23398的字符是：'学'
```

2.2.4 变量

3 变量的分类

依据的标准不同，变量分类的结果也不一样。这里主要按变量的作用范围来划分，**全局变量**是指在类中声明的类或对象的成员，称为成员变量，其作用范围是整个类；**局部变量**是指在一个方法或一个方法的程序块中声明的变量，亦称为本地变量，它的作用域就是该方法或对应的程序块内。　　参见VarialType.java

2.2.4 变量

4 变量的初始化

格式：数据类型 变量名 = 值; 或 数据类型 变量名1 = 值1, …, 变量名k = 值k ;

实践中，大家对于“变量的初始化”问题可能会有一些困惑，现总结为以下两点：

参见Initialization.java

(1)全局变量(即成员变量)如果不初始化，如表所示：

变量类型	默认值	变量类型	默认值	变量类型	默认值
byte	0	short	0	int	0
long	0L	float	0.0f	double	0.0
char	'\u000'	boolean	false	引用类型	null

(2)局部变量(即本地变量)必须初始化，否则将出错。

2.3 Java的运算符、表达式及语句

2.3.1 运算符

1 算术运算符

+ - * / %

所实现的功能与数学中的运算差不多.

(1) **+：Java中对该运算符进行了重载**

例如：

3+2

“Hello ”+”world!”

“日期:”+2004+”年” +9+”月” +10+”日”

‘A’+32

2.3.1 运算符

1 算术运算符

(2)“/”进行的是除法运算，运算结果与操作数的类型有关：当操作数为整数时，执行的是除法取整运算，结果仍为整数，例如：5/2的值为2；当操作数为浮点数时，则是通常意义上的除法，例如：5.0/2.0的结果为2.5；

(3)“%”完成的是取模运算，即求余数，

例如：5%2的结果为1，这可用来判断整数的奇偶性。

%：扩展到了实数

例如：325.24%10的结果为：5.24

2.3.1 运算符

2 自增(自减)运算符

均为单目运算符，功能是让操作数的值增 1 (或减 1)，在循环语句中常用来修改循环变量的值，以控制循环次数。按照运算符的位置不同，又可细分为前缀、后缀两种形式，它们的功能不尽相同，现用两个赋值表达式来说明它们的差异，设x、y是两个数值变量，那么：

(1)y=++x(或y=--x)：表示先让x的值增 1 (或减 1)，再获取x的值。

(2)y=x++(或y=x--)：表示先获取x的值，再让x的值增 1 (或减 1)。

从上不难看出，无论是前缀形式还是后缀形式，x的最终结果都是一样，但是y值则不同。

```
int m=7;
int n=7;
int a=2*++m;        //a为16，m为8
int b=2*n++;        //b为14，n为8
```

2.3.1 运算符

3 关系运算符

它们的含义与数学中的关系运算符相同，但是要注意书写方法的差异，**不能将==写成=，运算结果为boolean型，只能是true 或 false**，主要用来进行条件判断或循环控制。仔细分析，可以发现有三组关系式：<和>=、>和<=、==和!=，每对中的两个运算符都是互为相反结果的运算，当其中的一个值为true时，另一个运算结果必定为false。清楚了这些关系，在构造条件表达式时，就能针对同一问题，使用两种不同的表达式，达到“异曲同工”的效果。

2.3.1 运算符

4 逻辑运算符

这三个运算符的操作数都是boolean型，运算结果也为boolean型。

(1)!，单目运算符，运算规则是：!true即为false，!false则是true；

(2)&&，运算规则是：只有同时为true时，结果才为true；

(3)||，运算规则是：只有同时为false时，结果才为false。

2.3.1 运算符

5 位运算符

计算机中的数据是以二进制方式存储的，利用位运算符可以操作数据的“位”。其中：

(1)~的运算规则是：~1即为0，~0则是1；

(2)&的运算规则是：只有同时为1时，结果才为1；

(3)|的运算规则是：只有同时为0时，结果才为0；

(4)^的运算规则是：只有一个位为1，另一个位为0时，结果才为1。

由“异或”运算规则还可推出下列式子：a^a=0，a^0=a, c=a^b, a=c^b。如果双方约定数据与同一个数b进行^运算，则可以实现加密、解密功能。

2.3.1 运算符

6 移位运算符

(1)左移：a<<b 表示将二进制形式的a逐位左移b位，最低位空出的b位补0；

(2)带符号右移：a>>b 表示将二进制形式的a逐位右移b位，最高位空出的b位补原来的符号位(即正数补0，负数补1)；

(3)无符号右移：a>>>b 表示将二进制形式的a逐位右移b位，最高位空出的b一律补0。

说明：

①移位运算适用byte、short、char、int、long类型数据，对低于int型的操作数将先自动转换为int型再移位；

②对于int(或long)型整数移位a>>b，系统先将b对32(或64)取模，得到的结果才是真正移位的位数。

在java中请注意：

&&　||

与

&　|

的条件表示式的区别

```
int b=12;
if (false && b++>20) {

}
System.out.print(b);
```

2.3.1 运算符

7 赋值运算符

在程序中大量使用赋值运算符，其功能是：先计算右边表达式的值，再赋给左边的变量，形式：

<变量>=<表达式>

+=； -=； *=； /=； %=.....

x+=3;⇔ x=x+3

结构方向：从右至左

x-=x*=2+3;

=>x=x-(x*(2+3))

2.3.1 运算符

8 条件运算符

格式：逻辑表达式？值1:值2

执行过程：若逻辑表达式为true，就取值1，否则取值2。

例如：设x、y是double型数据，则：y= (x>=0)? x : (-x)；//得到x的绝对值

2.3.3 语句

1 程序的注释

给程序添加注释的目的，就是对程序某些部分的功能和作用进行解释，以增加程序的可读性。注释在程序编译时被删除，所以它不是程序的必要部分，更不属于语句范围。但是，注释是为语句服务的，两者联系密切，因此，放在这里介绍。

Java程序的注释有三种格式：

(1)单行注释：以//开始，到行尾结束；

(2)多行注释：以/*开始，到*/结束，可以跨越多行文本内容。

(3)文档注释：以/**开始，中间行以*开头，到*/结束。使用这种方法生成的注释，可被Javadoc类工具生成程序的正式文档。

2.3.3 语句

2 复合语句

又称块语句，是包含在一对大括号(即由{、}包含)中的语句序列，整体可以看作是一条语句，所以，{之前和}之后都不要出现分号(;)。

说明：

(1)在复合语句中可以定义常量、变量，但该常量、变量数据的作用域仅限该复合语句；

(2)在复合语句中还可以包含其它的复合语句，即复合语句允许多层嵌套。

2.5.1 程序的基本结构

Java程序大致有如下的形式：

```
package 包名;     //包语句最多1条，位置在最前面
import  包名.类名;//导入语句可以没有，也可以1条或多条
……
[public]class类名{//主类应该用public关键字修饰，且文件主名与主类名称相同
        // public static void main(String args[]) {
……
        }
        //程序其它代码
}
```

从功能上看，Java程序通常应包含输入、处理、输出等几部分。

2.5.2 常用的输入输出格式

1 常用的输入格式

(1)命令行方式：用main()方法的参数来表示，args[0]代表第1个参数，args[1]代表第2个参数，以此类推。如果数据的目标类型是数值型，则需要调用包装类的静态方法parseXxx(…)把字符串转换成数值型。由于这种方法是在命令行下提供数据，一定程度上限制它的使用。

(2)传统的“I/O流”方式：采用“字节流—>字符流—>缓冲流”逐层包装方法，将代表键盘的System.in最终包装成字符缓冲输入流，这样，就可以调用它的readLine()方法来获取键盘输入内容。

2.5.2 常用的输入输出格式

1 常用的输入格式

(3)使用Scanner类：这是JDK 1.5后新增的内容，该类位于java.util包中，只需将System.in包装成Scanner实例即可，调用相应的方法来输入目标类型的数据，不需要再进行类型转换。

(4)图形界面的输入方式：通过调用javax.swing包中JOptionPane类的静态方法showInputDialog()来实现，输入的是字符串，也可能需要进行类型转换。

2.5.2 常用的输入输出格式

2 常用的输出格式

(1)传统的“I/O流”方式： System.out.print(输出内容)

最常用，可以用“+”运算符将各种数据类型数据与字符串连接起来。

(2)图形界面的输出方式：通过调用javax.swing包中JOptionPane类的静态方法showMessageDialog()来实现，当输出内容要分成多行时，可在字符串中插入’\n’。

输入/输出简介

- 输出：
  - java.lang.System
  - System.out.println将结果输出到标准输出设备
  - 输入：Scanner sn=new Scanner(System.in)
    - (java.util.*)
- 输入：
  - 为键盘输入提供一个对话框，可调用：
    - javax.swing.JOptionPane.showInputDialog(promptString)
    - 上述函数返回用户输入的数据——String类型

输入/输出简介

```
import javax.swing.*;
public class InOutTest{
    public static void main(String[] args)    {
      String name=JOptionPane.showInputDialog("姓名:");
      String input=JOptionPane.showInputDialog("年龄:");
      int age=Integer.parseInt(input);
      System.out.println("Hello,"+name
              +". Next year, you'll be "+(age+1));
      System.exit(0);
    }
}
```

带格式控制的输出

System.out.printf(format, args)；

参看

DecimalForm.java

printf的格式控制的完整格式：

% - 0 m.n l或h 格式字符

下面对组成格式说明的各项加以说明：

①%：表示格式说明的起始符号，不可缺少。

②-：有-表示左对齐输出，如省略表示右对齐输出。

③0：有0表示指定空位填0,如省略表示指定空位不填。

④m.n：m指域宽，即对应的输出项在输出设备上所占的字符数。N指精度。用于说明输出的实型数的小数位数。为指定n时，隐含的精度为n=6位。

⑤l或h:l对整型指long型，对实型指double型。h用于将整型的格式字符修正为short型。

2.4 Java的流程控制

2.4.1 顺序结构

通常，程序中的语句是按照书写顺序从上到下、逐条执行的，这种程序执行方式称为顺序执行，对应的程序结构称为顺序结构。

顺序结构是程序设计的基础，经常使用。该结构比较简单，毋须作更多的介绍。

2.4.2 选择结构

又称分支结构，是指在程序执行过程中，将根据条件是否满足来选择某一语句的执行，也即是说某些语句可能因为条件不满足而跳过。由于所执行的语句经过筛选，而非全部，所以，这种程序结构就称为选择结构。

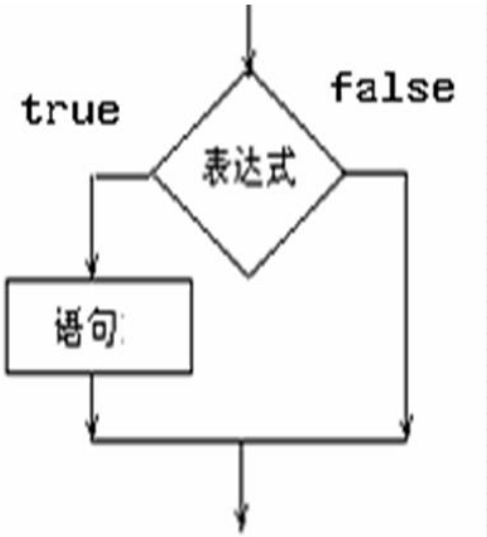
需要指出的是，Java中的“条件”只能是结果为boolean型的表达式，其值为true或false，而其它类型(包括：byte、int、short、long、char等)均无资格担当这一角色。在这一点上，Java与C++/C是不一样的，请加以注意。

2.4 Java的流程控制

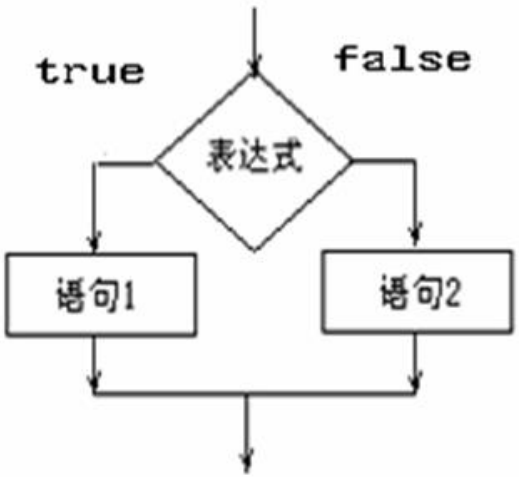
2.4.2 选择结构

1 if语句：又称条件语句

单分支语句



双分支语句



多分支语句

```
if  (条件 1){
      if  (条件 2)
            语句 1
    else
            语句 2
 }else{
    语句 3
}
```

```
if  (条件 1){
      语句 1
 }else{
      if  (条件 2)
          语句 2
      else
          语句 3
}
```

2.4.2 选择结构

2 switch语句：亦称开关语句

引进switch的目的，就是要在实现多分支时，让程序的结构更加清晰、易懂。事实上，Java中的switch语句用法与C++中的类似。

```
switch (表达式){
    case 常量 1:
        语句 1;
        [break;]
    case 常量 2:
        语句 2;
        [break;]
        ... ...
    default:
        语句 n+1;
        [break;]
}
```

2.4.3 循环结构

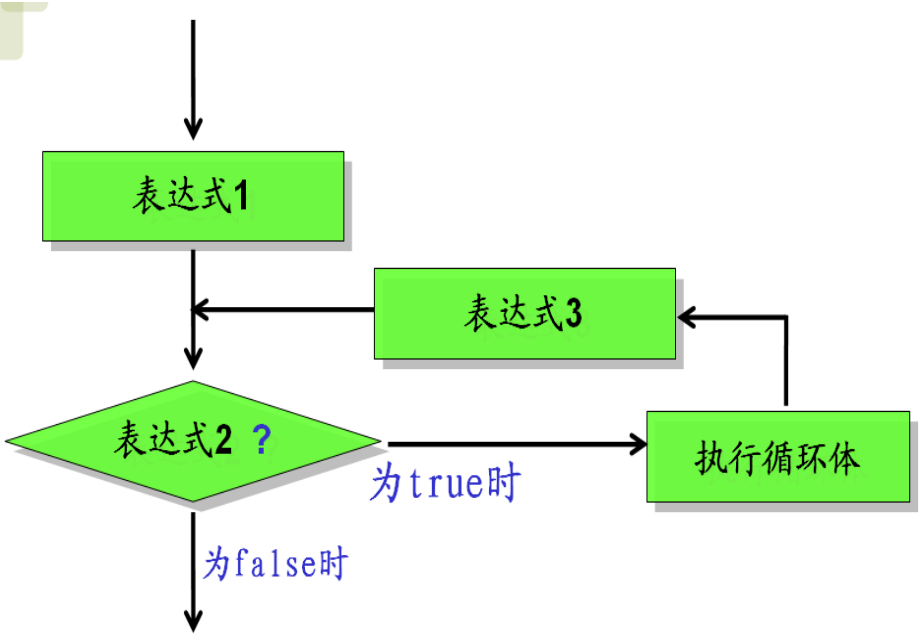
1 for语句

格式：

```
for (表达式1 ; 表达式2 ;  表达式3 ){

        循环体;

}
```



For在集合类的特殊用法：

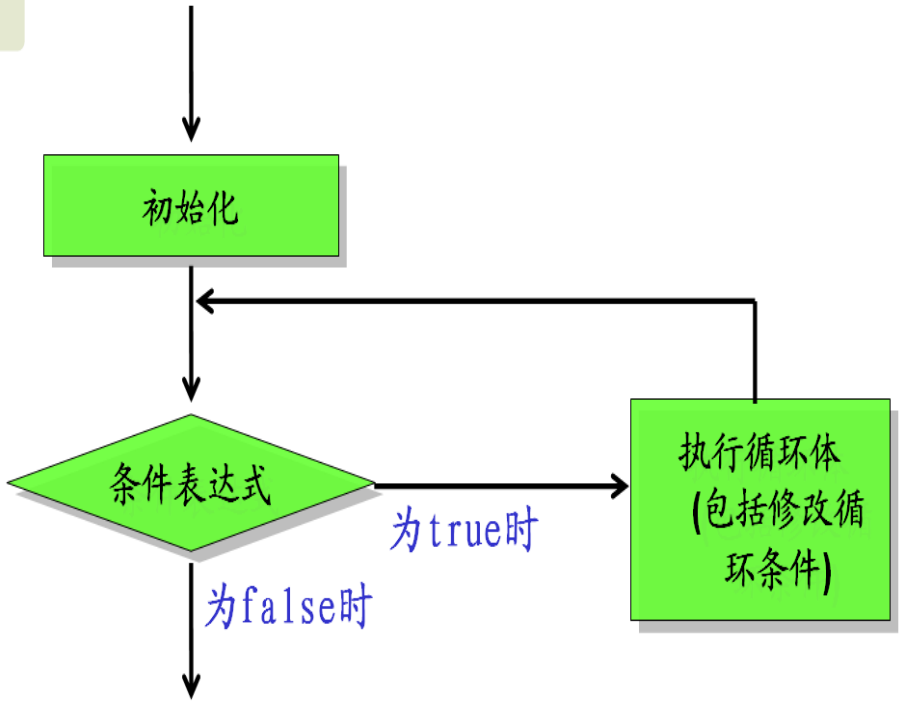
```
int[] arrs={10,5,6,7};
for(int s:arrs){
    System.out.println(s);
}
```

2.4.3 循环结构

2 while语句

格式：

```
        …          //初始化语句
  while (条件表达式) {
        语句块      //循环体
        …          //修改循环变量语句
}
```

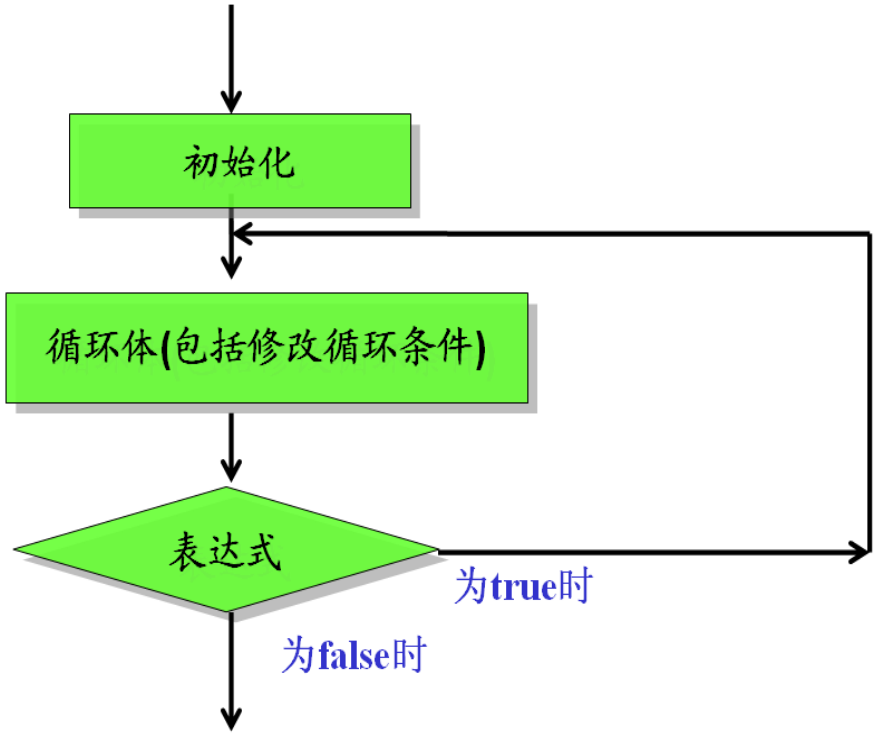


2.4.3 循环结构

3 do…while语句

格式：

```
        …          //初始化语句
do {
  语句块           //循环体
        …          //修改循环变量语句
} while  (条件表达式); //进行条件判断
```



2.4.3 循环结构

4 while与do…while的比较

主要有两点不同：

(1)有无分号：while语句的(条件表达式)后一定不要加分号；do…while语句的(条件表达式)后应加分号；

(2)循环体执行次数：while语句先判断条件表达式的值是否为true，再决定是否执行循环体，这样，循环体有可能一次也不被执行；而do-while语句是先执行一次循环体，再根据条件表达式值的真假，以确定下一次循环是否进行，因此，循环体至少被执行一次。

2.4.3 循环结构

5 多重循环

如果在一个循环体内允许包含另一个循环，这称为嵌套循环。其中：外层的循环称为外循环，内层的循环称为内循环，嵌套的层数可以根据需要达到一二十层之多。但是应注意：外循环和内循环不允许交叉嵌套。

2.4.3 循环结构

6 跳转语句

break语句

使程序的流程从一个语句块内部转移出去。该语句可用在循环结构和switch语句中，允许从循环体内部跳出或从switch的case子句跳出。

continue语句

终止本次循环，根据条件来判断下一次循环是否执行，只能用在循环结构中。

return语句

从某一方法中退出，返回到调用该方法的语句处，并执行下一条语句。

一个综合示例：

十进制转十六进制

```
static String decimalToHex(int num){
String str="";
int remain=0;

while (num!=0){
_____________________ //思考，这里填什么？
if (remain>=10){
            //思考，这里填什么？

}
else
            str=remain+str;
            //important

}
return str;
}
```

参见DecimalOctalHex.java

2.6 数组

2.6.1 数组的概念

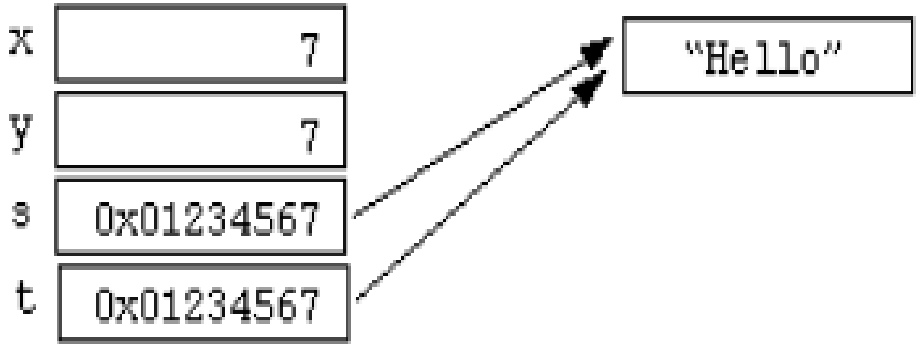
在Java中，数组是一种**引用类型**(即对象类型)，是由类型相同的若干数据组成的有序集合，其中的每一个数据称为元素

每个数组都是一个对象。声明不创建对象本身，而是创建引用

引用

- 引用类型的赋值：
  - 例如：有如下代码片段：

```
int x=7;int y=x;
String s=new String("Hello");
String t=s;
```

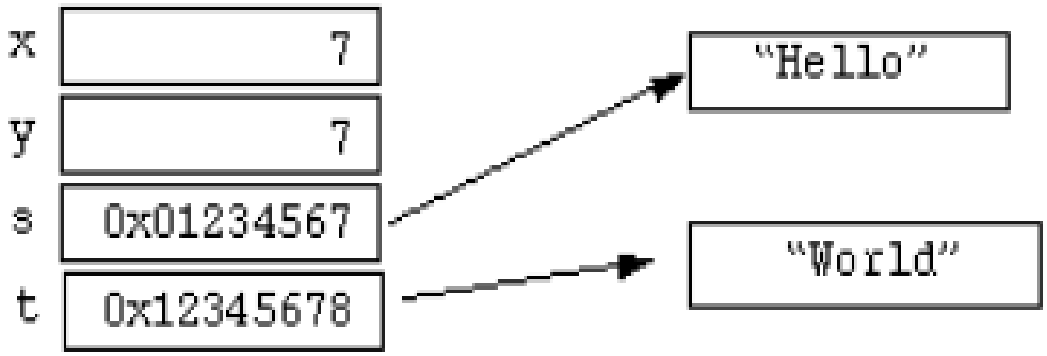


引用

- 引用类型的赋值：
  - 上述代码执行后，又执行：

    t=“World”;

    结果内存图为：



引用

- 引用实质是指针，但是“安全的指针”
  - 不能直接利用指针运算对其值进行修改；
  - 分配利用new动作完成；
  - 回收由垃圾回收机制处理；

2.6 数组

2.6.1 数组的特点

在一个数组中：

1．每一个元素的**数据类型都是相同的**，数组元素可以是基本类型，也可以是对象类型，甚至还可以是数组类型(如：多维数组)；

数组的声明：

数组

- 数组的声明：
  - 可以声明任何类型的数组(基本类型或类类型)

    char s[];⇔ char[] s1,s1;
    - char[]——字符数组类型
    - s1,s2——变量名

    MyDate p[];⇔ MyDate[] p;

  - **数组变量声明时，不能指定数组的长度**

2.6 数组

2.6.1 数组的特点

在一个数组中：

1．每一个元素的**数据类型都是相同的**，数组元素可以是基本类型，也可以是对象类型，甚至还可以是数组类型(如：多维数组)；

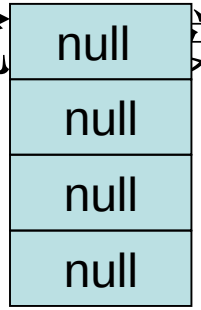
2．数组要**经过声明、分配内存及赋值**后，才能使用。

数组

- 数组元素使用new或数组初始化动态分配实际存储空间
- 创建数组（new）

```
char[] str1;
str1=new char[2];
MyDate[] dt;        //dt==null
dt=new MyDate[4];
```

  - 创建数组时，每个元素都将被初始化安全
    - 基本类型使用默认值；类对象使用null



数组

- 数组初始化：
  - 初始化的两种等价形式：

```
String names[]={ “Georgianna”,        “Jen”,“Simon”};
```

⇔下列代码片段

```
String names[];
names=new String[3];
names[0]=“Georgianna”;
names[1]=“Jen”;
names[2]=“Simon”;
```

数组

2.6.1 数组的特点

在一个数组中：

1．每一个元素的**数据类型都是相同的**，数组元素可以是基本类型，也可以是对象类型，甚至还可以是数组类型(如：多维数组)；

2．数组要**经过声明、分配内存及赋值**后，才能使用。

3．所有元素共用一个数组名，数组中的每一个元素都是有顺序的，利用数组名和数组下标可以唯一地确定数组中每一个元素的位置；

数组

- 数组元素：

  访问格式：数组名[index]

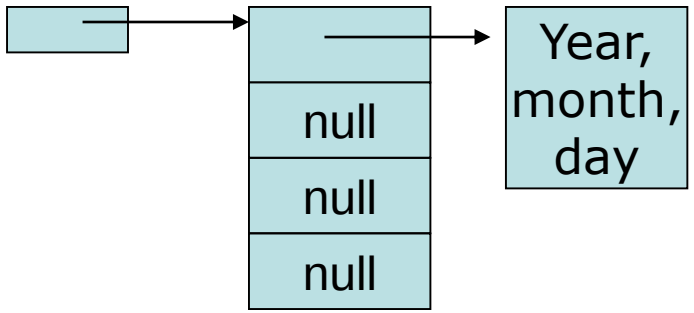
  ➢元素是基本类型

```
int[] st=new int[3];
st[0]=2;st[1]=5;st[2]=10;
int max=(st[0]>st[1]?st[0]:st[1]));
max= (max>st[2]?max:st[2]));
System.out.println(“the max is ”+max);
```

数组

- 数组元素：
  - ➢元素是引用类型

```
MyDate[] dt;        //dt==null
dt=new MyDate[4];
//生成的数组中，四个单元都是null
dt[0]=new MyDate();
```



数组的使用

- 数组的界限
  - 所有数组下标都从0开始
  - 数组对象具有length属性，用于检验访问边界
    - length属性的声明形式为：

      public final length;//可以读取，不能修改
  - 例如：int[] list=new int[10];

    for (int i=0;i<list.length;i++)

    System.out.println(list[i]);

数组

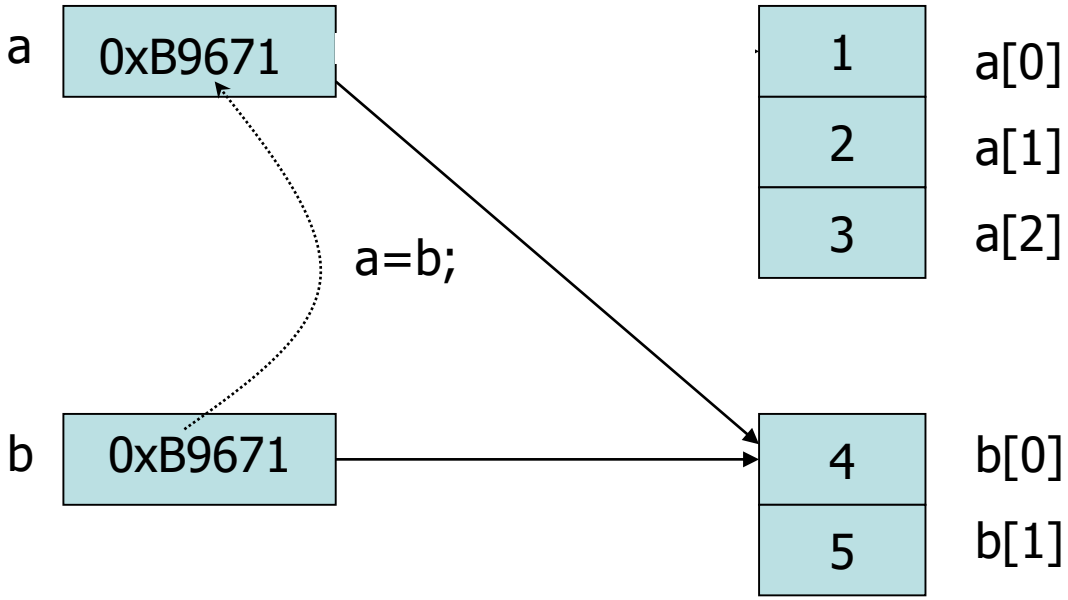
- 数组的使用
  - 数组一旦创建，不能调整其大小
  - 可以使用相同的引用变量来引用一个**全新**的数组
    - 例如：int[] elements=new int[6];
      elements=new int[10];

数组

数组整体赋值的问题

int [] a={1,2,3},b={4,5};

a=b;



数组

- 拷贝数组：
  - Java中在System类中提供了特殊的方法拷贝数组：arraycopy()——如果数组是对象型，则拷贝的是引用，而不是对象，即对象本身不变

```
System.arraycopy(from,fromindex,to,toindex,count)

int myArray[]={1,2,3,4,5,6};
int hold[]={10,9,8,7,6,5,4,3,2,1};
System.arraycopy(myArray,0,hold,0,
        myArray.length);
```

//考虑：结果hold数组中的值是？

数组

- 拷贝数组：
  - 位于Arrays类，要导入java.util包

public static double[] copyOf(double[] original, int newlength)

public static double[] copyOfRange(double[]original, int from,int to)

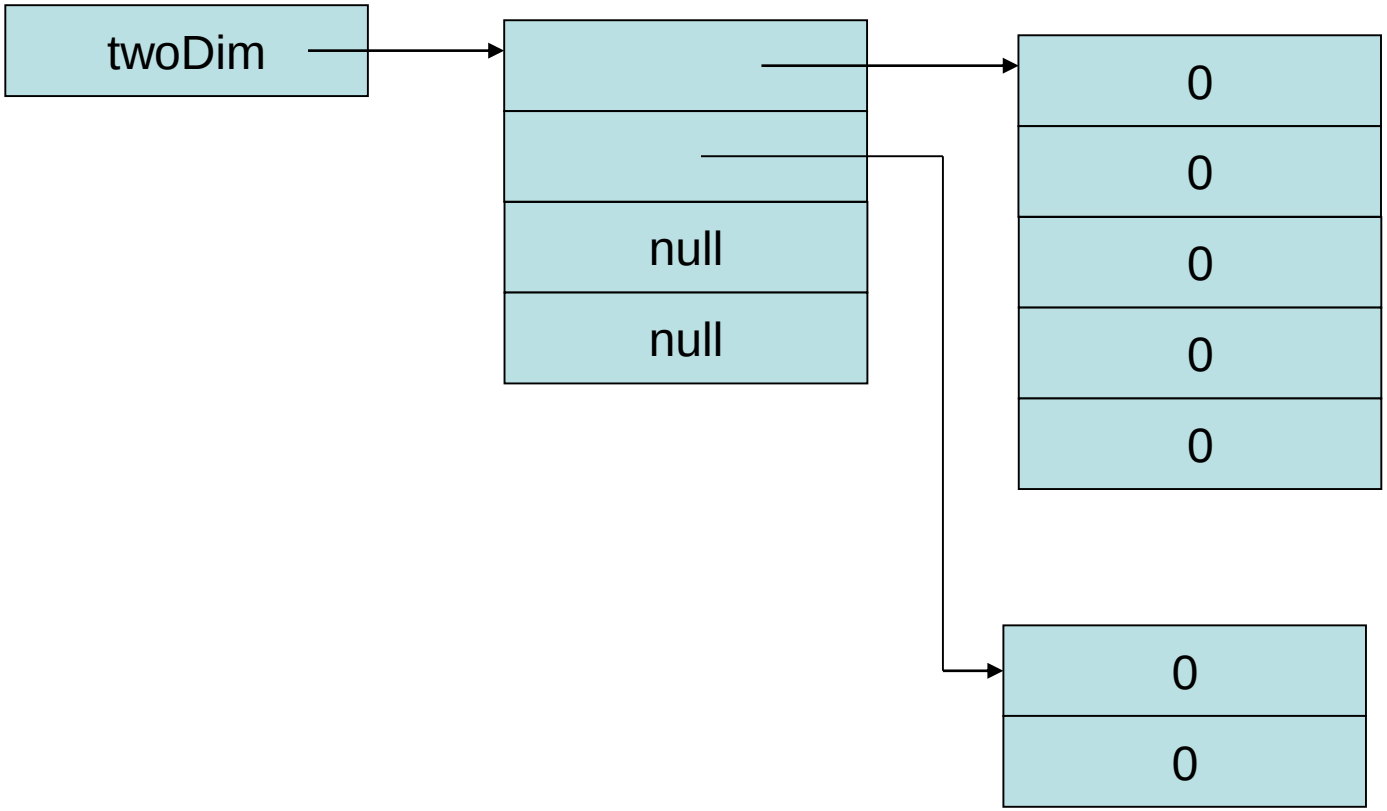
```
int myArray[]={1,2,3,4,5,6};
int hold[]=copyof(myArray,3);//3->7
//考虑：结果hold数组中的值是？
```

数组

- 多维数组：
  - 实质上，Java中不存在多维数组——因为数组可以声明成具有任何类型。
  - 所谓多维数组，就是数组的数组
  - 例如：

```
Int [][]  twoDim=new int[4][];
          //int (twoDim[])[]=new int[4][];
twoDim[0]=new int[5];
twoDim[1]=new int[2];
```

    - 多维数组时，[]不能放在左侧，即new int [][4]是非法的


twoDim
null
null
0
0
0
0
0
0
0

数组

- 多维数组：
  - 对于规则矩形数组，可简化进行初始化：
  - 例如：int twoDim[][]=new int[4][5];

数组

- 多维数组：
  - 由于多维数组中对每个数组元素分别初始化，所以可以形成非矩形数组的数组
  - 例如：

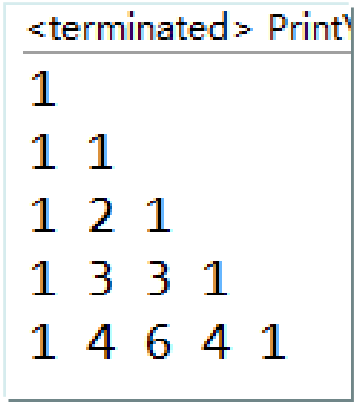
```
int twoDim[][]=new int[4][];
twoDim[0]=new int[2];
twoDim[1]=new int[4];
twoDim[2]=new int[4];
twoDim[3]=new int[8];
```

小练习

- 一维数组，随机赋值，求最大、最小，平均分
  - Math.Random

小练习

- 二维数组，接受人工赋值，打印二维数组和对角线之和
  - Scanner
- **打印杨辉三角形**



数组

```
int[][] arrs=new int[5][];
//声明第二维，并赋初值
for(int i=0;i<5;i++){
        arrs[i]=new int[i+1];
        arrs[i][0]=1;
        arrs[i][i]=1;
}

//处理数组元素
for(int i=2;i<5;i++){
     for(int j=1;j<i;j++)
              arrs[i][j]=arrs[i-1][j-1]+arrs[i-1][j];
}
```

```
//打印
for(int i=0;i<5;i++){
      int len=arrs[i].length;
      for(int j=0;j<len;j++)
            System.out.print(arrs[i][j]+" ");
      System.out.println();
}
```

```
<terminated> Print
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

数组

- 数组的使用
  - 数组作为方法参数或返回值
  - 数组的常用动作：
    - 排序 sort
    - 二分查找 binarySearch （先排序再二分查找）
    - 都在Arrays类中

```
int[] a={10,2,7,30};
Arrays.sort(a);
System.out.print(Arrays.toString(a));
int res=Arrays.binarySearch(a, 7);
System.out.print(res);
```

数组

- 命令行参数：
  - Java程序中main的形式固定为：

    public static void main(String[] args)
    - args：字符串数组，接收命令行参数
    - 例如：public class Message{

```
        public static void main(String[] args){
        if (args[0].equals("-h"))
                System.out.println("Hello");
        else
                System.out.println("NoMess");
        }
    }
```

枚举类型

语法格式：

enum 枚举名

{常量列表

}

常量列表是用逗号分隔的字符序列

```
enum Season{
spring,summer,autumn,winter
}
```

声明枚举类型变量

Season x；

枚举变量只能取枚举类型中的常量

x=Season.spring;

System.out.println(x);

枚举变量能转成数组

Season[] a=Season.values();

例：EnumClass.java

本章小结

Java程序是由一系列语句组合而成的，而语句又是由关键字、常量、变量、运算符、表达式等基本元素构成。本章讲述的内容就是这些构成程序的语句及其组成要素，学习编程首先要掌握的是语句、程序的基础知识。

Java中的数据类型可分为基本数据类型和引用类型两大类，其中：基本类型包括byte、short、int、long、char、float、double、boolean等8种，引用类型包括类、接口、数组等。对于初学者，基本数据类型的主要内容一定要掌握。

本章小结

关键字是一些有特殊含义的单词,它们只能按系统规定方式来使用,不能单独用作标识符；变量在程序中最常用，应掌握其声明、赋值、初始化方法，并能区分全局变量与局部变量的不同之处；Java中的符号常量是用final关键字来定义的，常量名通常为大写，且多个单词之间用下划线连接。

Java的运算符与C++类似，非常丰富，包括：算术、自增自减、关系、逻辑、位运算、赋值、条件等运算符。运算符将操作数连接起来形成表达式，表达式是构成语句的重要基础。为了增加程序的可读性，经常需要在程序中加入注释，Java中的注释语句常用的有三种类型：单行注释、多行注释和文档注释，应熟练掌握注释语句的使用，这是规范化编程的基本要求。

本章小结

Java的程序结构有：顺序、选择和循环三种基本类型，其中：顺序结构最简单，不需要什么控制语句，选择结构主要有：if...else…和switch两种主要形式，循环结构有：for、while和do…while三种形式。break、continue在switch和循环语句中都有特定用途，与C++不同的是，Java的这两条语句还可以带上标签，功能进一步增强。一个完整的Java程序有相应的格式要求，熟悉这些内容对于编程大有裨益。在输入输出方面，Java要比C++复杂一些，本章列举出4种常用的输入格式和2种输出格式，目的是让大家根据不同要求选用，相关知识将在后续章节学习，暂时只要能模仿、会使用就行。

本章小结

在Java中，数组是一种引用类型(即对象类型)，它是由类型相同的若干数据组成的有序集合。数组需经过声明、分配内存及赋值后，才能使用。数组的属性length用来指明它的长度，与循环结合起来可以访问其全部或部分元素。根据维数的不同，数组还可分为一维数组、多维数组。事实上，Java是把多维数组当作“数组的数组”来处理的，即把一个多维数组也看作是一个一维数组，而这个一维数组的元素又是一个降了1维的数组。

本章的内容比较基础，在C++等高级语言中包含类似的知识点，在学习时请注意区分，以免张冠李戴。

本章重点：Java的基本数据类型，变量、常量的使用，运算符与表达式，流程控制语句，程序的基本结构，数组；难点：基本数据类型，运算符，常用的输入输出格式，数组。