第 3 章　Java 控制结构、数组和字符串

Java 程序的基本控制结构包含顺序、分支和循环 3 种。顺序结构的执行是按照先后次序顺次执行，分支结构的执行是根据判定结果选择对应的语句来执行，循环结构的执行是根据循环条件来决定是否重复执行循环体。本章除了介绍上述内容之外，还将介绍数组和字符串应用的语法格式。

3.1　顺 序 结 构

3.1.1　语句和语句块

Java 程序是由一条条语句组成的，语句就成为其最小的执行单位。

2.3.5 节讲到了赋值运算，它实际上就对应着语句，例如 int a＝9；就是一条赋值语句。例 2-3 中出现了两种输出语句：

```
System.out.println(a=a * b);
System.out.println(a);
```

在这两行语句中，前一行有赋值号，需要计算表达式 a * b 并赋值给变量 a；后一行没有赋值运算符，就把变量 a 的值返回给输出语句。Java 语言也可以将方法的调用写在输出语句中，例如：

```
System.out.println(Math.sqrt(16));
```

Java 语言的语句块是指用“{}”括起来的语句组，这在语法上将作为一条语句使用。例如，下面就是一个语句块：

```
{  int y=1;
   y=y+20;
}
```

2.2.3 节提到了变量的作用域。有了语句块的定义以后，变量的作用域就可以解释为变量定义所在的语句块。

语句块可以嵌套。在嵌套时，变量可以定义在语句块的任何位置。外层语句块中定义的变量既可以在外层使用，也可以在内层使用。也就是说，外层块定义的变量，其作用域涵盖了内层块，反之则不然。下例出现了此类作用域问题：

```
{  int a=3, b=5;
       {  int c=7;
          c=c-a;
       }
   b=b+c;
}
```

上面的语句块中，外层定义了变量 a 和 b，其作用域是在整个内、外层块的范围中，即内层块也可以使用 a 和 b，因此语句

```
c=c-a;
```

正确。但是，内层块定义的变量 c 作用域仅限于内层，外层块不可以使用，因此语句

```
b=b+c;
```

错误。

3.1.2 顺序结构

和其他结构化程序设计语言类似，Java 语言也有顺序结构、分支结构和循环结构 3 种基本控制结构。顺序结构的程序在执行时是按照前后顺序依次执行的，其流程如图 3-1 所示。

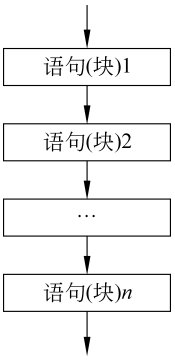

图 3-1 顺序结构的流程图

顺序结构在程序设计中用得非常频繁。这种结构的语句比较简单，但在具体程序中是必不可少的。例 1-1～例 1-3 以及例 2-1～例 2-7，都是属于顺序结构的程序。

【例 3-1】 顺序结构应用举例。编程，实现交换两个变量的值。

程序代码如下：

```
1    public class Example3_1
2    {
3        public static void main(String args[])
4        {
5            int m=1, n=6, t;
6            t=m;
7            m=n;
8            n=t;
9            System.out.println("m="+m);
10           System.out.println("n="+n);
11       }
12   }
```

程序运行结果如下：

```
m=6
n=1
```

程序分析如下：

交换两个变量的值要借助第三个变量才能实现。

3.2 分 支 结 构

在例 2-7 中，用户提供的一元二次方程系数 a、b 和 c 满足判别式大于 0，程序运行后可以得到两个不相等的实数根。但是，若考虑判别式等于 0 和小于 0 的情况，就对应着两根相等和

根不存在的结果。此时,需要分不同情形选择不同的语句块来得出结论。仅使用顺序结构无法实现这样的思路,应用分支结构就成为了编程的关键。许多现实生活中的实例都与分支结构的思想相对应。例如,打客服电话时话务员会提醒用户选择服务种类;再如使用自助存取现金时,ATM 机会根据提示选择币种,等等。在 Java 语言中,分支结构主要包括 if 语句和 switch 语句。

3.2.1 if 语句

if 语句有双分支结构和单分支结构,其语法格式如下:

```
if(条件表达式)
    语句块 1
[ else
    语句块 2
]
```

在执行时,先判断条件表达式的值,若为 true 则执行语句块 1,之后执行该 if 分支结构的后续语句;否则执行 else 后面的语句块 2。流程如图 3-2(a)所示。如果省略 else 子句,条件表达式值为 true 执行语句块 1,为 false 时什么也不做,其流程如图 3-2(b)所示。

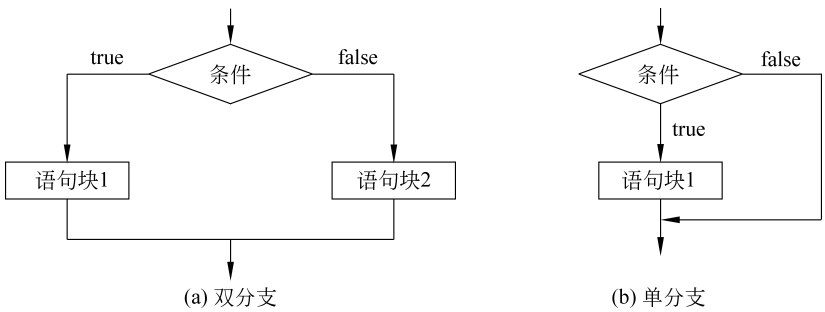

图 3-2 if 语句流程图

【例 3-2】 比较两个数,求出其中的较大者并输出。若使用双分支的 if 语句,则程序代码如下:

```
1   import java.util.Scanner;
2   public class Example3_2
3   {
4       public static void main(String args[])
5       {
6           Scanner scan=new Scanner(System.in);
7           System.out.println("请输入两个数:");
8           double x=scan.nextDouble();
9           double y=scan.nextDouble();
10          if(x<y)
11          {
12              System.out.println("两个数中的较大者: "+y);
13          }
14          else
```

```
15          {
16              System.out.println("两个数中的较大者: "+x);
17          }
18      }
19  }
```

程序运行后,在提示“请输入两个数:”后面输入 3.14 和 2.7,则输出结果如下:

```
两个数中的较大者: 3.14
```

程序采用了双分支的 if…else… 语句,这是二选一型结构。如果采用单分支 if 语句,则需要将上述程序代码进行修改,要将第 10～17 行替换为下列语句:

```
double max=x;
if(x<y)
{
    max= y;
}
System.out.println("两个数中的较大者: "+max);
```

程序的运行结果不变。这里 if 语句仅有一个分支处理,故预设变量 max 并先将 x 的值赋给它。经过比较,确定调整或不调整 max 的值,再输出 max 即可。

【例 3-3】 已知下面的分段函数:

$$y=\begin{cases}5x, & x\leqslant 0\\ 1+2x, & x>0\end{cases}$$

设计程序,输入 x 得到 y 的值。

程序代码如下:

```
1   import java.util.Scanner;
2   public class Example3_3
3   {
4       public static void main(String args[])
5       {
6           Scanner scan=new Scanner(System.in);
7           System.out.println("请输入 x:");
8           double x=scan.nextDouble(),y;
9           if(x<=0)
10          {
11              y=5*x;
12          }
13          else
14          {
15              y=1+2*x;
16          }
17          System.out.println("y="+y);
18      }
19  }
```

程序运行后，用户若输入 0 则输出 $y=0.0$；若输入 8 则输出 $y=17.0$。

本例中的分段函数也是二选一型的，应当用双分支 if 语句完成。但是，如果分段函数存在超过两种情形来选择时，使用前面的语句结构就难以完成了，此时可以用 if 语句的嵌套形式实现。

3.2.2 if 语句的嵌套

if 语句的嵌套是指单分支或双分支 if 语句中，语句块又包含着一重或多重 if 语句。其语法格式如下：

```
if<条件表达式>
  { if <条件表达式>
          语句块
      [ else
          语句块
      ]}
[ else
  { if <条件表达式>
      语句块
    [ else
      语句块
       ]}
]
```

嵌套时，语句中的每一个 else 必须和一个 if 相对应，以避免产生混乱。

【例 3-4】 再计算下面分段函数的值：

$$y=\begin{cases}2x+1, & x<2\\ x-3, & 2\leqslant x<8\\ 3x-1, & x\geqslant 8\end{cases}$$

程序代码如下：

```
1    import java.util.Scanner;
2    public class Example3_4
3    {
4        public static void main(String args[])
5        {
6            Scanner scan=new Scanner(System.in);
7            System.out.println("请输入 x:");
8            double x=scan.nextDouble(),y;
9            if(x<2)
10           {
11               y=2*x+1;
12           }
13           else
14           {    if(x<8)
15                   y=x-3;
```

```
16              else
17                  y=3*x-1;
18          }
19          System.out.println("y="+y);
20      }
21  }
```

用户在程序运行后，分别输入 0，7 和 9，对应输出结果为 1.0，4.0 和 26.0。该程序使用了 if 语句嵌套形式：在 else 子句部分，又嵌入了 if 语句，这样建立了三选一型结构。请读者思考一下：若换作在 if 和 else 之间嵌入另外一个 if 语句，应当如何完成？

3.2.3 多分支 if 语句

例 3-4 中的 if 语句嵌套形式也可以改写为多分支 if 语句，这种格式看起来比较清晰。其语法格式如下：

```
if(条件表达式 1)
    语句块 1
else if(条件表达式 2)
    语句块 2
…
else if(条件表达式 n)
    语句块 n
[else
    语句块 n+1
]
```

多分支 If 语句的执行过程为，先计算条件表达式 1，若条件表达式 1 的值为 true 则执行其后的语句块 1，然后执行多分支 if 结构后面的语句；若条件表达式 1 的值为 false，则判断条件表达式 2 的值是否为 true，为 true 则执行其后的语句块 2，否则依次继续向下判断。如果所列出的前 n 个条件表达式的值都为 false，再看有没有 else 子句，有则执行它后面的语句块 $n+1$，若没有则执行该多分支 if 结构后面的语句。整个流程如图 3-3 所示。

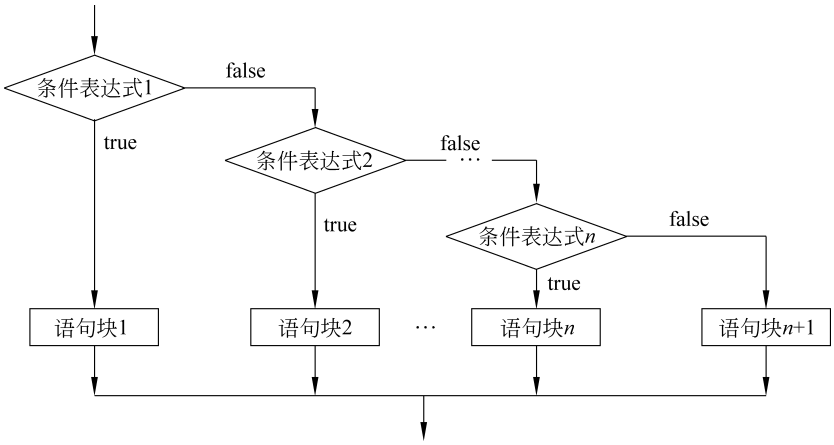

图 3-3　多分支的 if 语句流程图

将例 3-4 改写为多分支 if 语句，核心部分如下：

```
if(x<2)
    y=2*x+1;
else if(x<8)
    y=x-3;
else
    y=3*x-1;
```

相比于 if 语句嵌套，这样的程序看起来层次更加分明。例 2-7 中，将判别式等于 0 和小于 0 的情况都考虑进来，则完整的程序代码如下：

```
1    import java.util.Scanner;
2    public class Example2_7
3    {
4        public static void main(String args[])
5        {
6            Scanner scan=new Scanner(System.in);
7            System.out.println("请输入方程的系数 a,b,c");
8            double a=scan.nextDouble();
9            double b=scan.nextDouble();
10           double c=scan.nextDouble();
11           double p,q,delta=b*b-4*a*c;
12           p=-b/(2*a);
13           if(delta<0)
14               System.out.println("该一元二次方程无实数根");
15           else if(delta==0)
16           {   double x=p;
17               System.out.println("该一元二次方程两根相等,x="+x);
18           }
19           else
20           {   q=Math.sqrt(delta)/(2*a);
21               double x1=p+q;
22               double x2=p-q;
23               System.out.println("x1="+x1);
24               System.out.println("x2="+x2);
25           }
26       }
27   }
```

程序运行后，用户若输入 1，2 和－15 则输出结果为“x1＝3.0”和“x2＝－5.0”；用户若输入 1，4 和 4 则输出结果为“该一元二次方程两根相等，x＝－2.0”；用户若输入 5，1 和 2 则输出结果为“该一元二次方程无实数根”。

3.2.4 switch 语句

switch 语句是 Java 语言中另一种表示多分支结构的语句，该结构使得此类程序更为简练，其语法格式如下：

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

switch 语句的执行过程为，先对表达式求值，然后将它逐一与 case 子句中的常量值相匹配：如果匹配成功，则执行该 case 子句中的语句块，直到遇见 break 才算结束，去执行 switch 结构的后续语句。假如某个 case 子句中没有 break，若表达式的值与该 case 后的常量值相等，在执行完毕该 case 子句中的语句块后，要紧接着去执行后继的 case 子句中的语句序列，直到遇见 break 才算结束。如果所有 case 子句的常量均未能匹配成功，则往下看有没有 default 子句，若有就执行语句块 $n+1$，若无就算结束，去执行 switch 结构的后续语句。整个流程如图 3-4 所示。

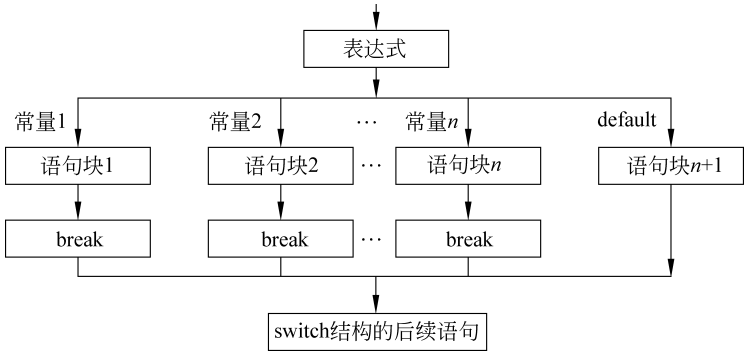

图 3-4　switch 语句的流程图

【例 3-5】 将数值 0～6 表示的星期几转换为由英文表示的形式。

程序代码如下：

```
1    import java.util.Scanner;
2    public class Example3_5
3    {
4        public static void main(String args[])
5        {
6            Scanner scan=new Scanner(System.in);
7            System.out.println("请输入一个数字(0～6): ");
```

```
8           int when=scan.nextInt();
9           switch(when)
10          {
11              case 0: System.out.println("Sunday");    break;
12              case 1: System.out.println("Monday");    break;
13              case 2: System.out.println("Tuesday");   break;
14              case 3: System.out.println("Wednesday"); break;
15              case 4: System.out.println("Thursday");  break;
16              case 5: System.out.println("Friday");    break;
17              case 6: System.out.println("Saturday");  break;
18              default: System.out.println("Error Input");
19          }
20      }
21  }
```

程序中的 switch 语句有 8 个分支，对应着用户输入不同数据的处理结果。

3.3 循环结构

循环是指在程序中有规律地反复执行某一语句块的现象。被重复执行的语句块称为循环体，循环体的执行与否以及次数多少要视循环类型与条件而定。当然，无论何种类型的循环结构，其共同的特点是必须确保循环体的重复执行能被终止。

Java 的循环语句有 for 语句，while 语句和 do 语句。

3.3.1 for 语句

for 语句在循环结构中最常用，属于计数型循环，其语法格式如下：

```
for ([表达式 1];[表达式 2];[表达式 3])
    循环体语句块
```

其中，[表达式 1]是对循环控制变量初始化操作；[表达式 2]是关系表达式或逻辑表达式，是循环控制条件；[表达式 3]用于改变循环控制变量的值。这 3 个表达式都可以省略，当[表达式 2]省略时就默认该式的值为 true。

for 语句运行后，首先执行 1 式，进行循环控制变量的初始化。接下来判定 2 式的真假。如果为真，就去执行循环体语句块，并且要由 3 式来修改循环控制变量，这就完成了一次循环。然后再次判定 2 式的真假，如果为真就重复上述过程。这样的操作一直进行下去，直到 2 式的值为假时，不再执行下去，整个循环宣告结束，转向 for 结构的后续语句。当然，若首次判定 2 式的值就为假，则循环一次也不会执行。上述过程如图 3-5

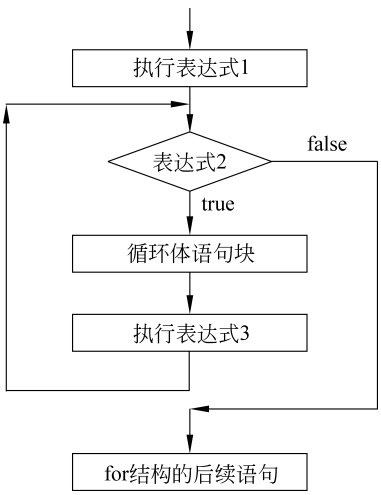

图 3-5　for 语句流程图

所示。

【例 3-6】 求 1＋2＋3＋…＋10 的值。

程序代码如下：

```
1    public class Example3_6
2    {
3        public static void main(String args[])
4        {
5            int sum=0;
6            for(int i=1;i<=10;i++)
7                sum=sum+i;
8            System.out.println("1+2+3+…+10="+sum);
9        }
10   }
```

程序运行结果如下：

```
1+2+3+…+10=55
```

本例题编程实现起来较为简单，理解了累加的思想就能顺利完成，第 7 行代码是关键。

【例 3-7】 求 Fibonacci 数列前 9 项的值。已知 Fibonacci 数列为 1,1,2,3,5,……该数列有如下特点：第 1 项、第 2 项均为 1，从第 3 项开始，每一项都是前两项之和，即

$$F_n=\begin{cases}1, & n=1\\ 1, & n=2\\ F_{n-1}+F_{n-2}, & n\geqslant 3\end{cases}$$

程序代码如下：

```
1    public class Example3_7
2    {
3        public static void main(String args[])
4        {
5            int x1=1, x2=1, x3;
6            System.out.println("数列第 1 项: "+x1);
7            System.out.println("数列第 2 项: "+x2);
8            for(int i=3;i<=9;i++)
9            {
10               x3=x1+x2;
11               System.out.println("数列第"+i+"项: "+x3);
12               x1=x2;
13               x2=x3;
14           }
15       }
16   }
```

程序运行结果如下：

```
数列第 1 项: 1
```

```
数列第 2 项：1
数列第 3 项：2
数列第 4 项：3
数列第 5 项：5
数列第 6 项：8
数列第 7 项：13
数列第 8 项：21
数列第 9 项：34
```

程序分析如下：

定义变量 x1 代表第 1 项，x2 代表第 2 项。x3 代表第 3 项的值，它由 x1＋x2 求得(代码第 10 行)。当首轮计算完成后，x2 成为第 1 项(代码第 12 行)，x3 成为第 2 项(代码第 13 行)，然后进行下一轮的运算。继续后面各轮循环，直到最后计算出第九项的值，这样就得到了全部结果。

【例 3-8】 如果一个三位数等于其各位数字的立方和，则称这个数为水仙花数，例如 $153=1^3+5^3+3^3$，$371=3^3+7^3+1^3$。试编程找出所有的水仙花数。

程序代码如下：

```
1    public class Example3_8
2    {
3        public static void main(String args[])
4        {
5            int x, y, z, k;
6            System.out.println("全部水仙花数: ");
7            for(int i=100; i<=999; i++)
8            {
9                x=i/100;
10               y=i/10-x*10;
11               z=i-x*100-y*10;
12               k=x*x*x+y*y*y+z*z*z;
13               if(i==k)
14               {
15                   System.out.print(i+"  ");
16               }
17           }
18       }
19   }
```

程序运行结果如下：

```
全部水仙花数:
153  370  371  407
```

程序分析如下：

100，101，102，…，999，每个数均要被检测。设每轮要测试的数为 i。针对判断条件，求出待测试数据 i 的百位(代码第 9 行)、十位(代码第 10 行)和个位(代码第 11 行)的数字，看

看是否符合水仙花数的条件(代码第 12、13 行)。再一轮要测试加 1 之后的 i,如此循环 900 轮,完成对全部三位数的检测。

3.3.2 while 语句

循环结构中 for 语句一般用于循环次数可预知的情形。然而,编程时往往存在循环次数无法确定、要通过条件终止循环的状况,此时可以采用 while 语句来实现。while 语句属于条件型循环,它根据某一条件进行判断,决定是否执行循环。其语法格式如下:

```
while(布尔型表达式)
    循环体语句块
```

while 语句执行时,如果布尔型表达式的值为 true,则执行循环体语句块;否则退出循环。其执行流程如图 3-6 所示。

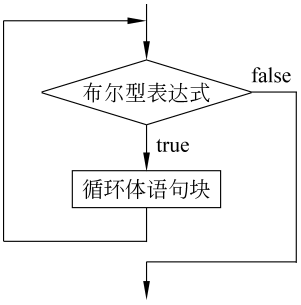

图 3-6 while 语句流程图

实际编程中,循环体语句块在执行时应能使条件发生改变,以确保布尔型表达式的值最终可以出现 false,否则会出现死循环。

【例 3-9】 我国现有人口 13 亿,求按照年增长率 1.2%计算,需要多少年后我国人口超过 20 亿。

程序代码如下:

```
1    public class Example3_9
2    {
3        public static void main(String args[])
4        {
5            double p =13, r =0.012;
6            int i =0;
7            while(p<=20)
8              {
9                p =p * (1 +r);
10               i =i +1;
11           }
12           System.out.println(i+"年后,我国人口达到"+p+"亿");
13       }
14   }
```

程序运行结果如下:

```
37 年后,我国人口达到 20.212606208344745 亿
```

程序分析如下:

定义变量 p 表示人口数并赋初值为 13(亿),变量 r 表示增长率并赋初值为 0.012,变量 i 表示经过多少年。while 语句的循环体主要来计算人口每过一年以后的数量(代码第 9 行),并且要修改循环控制变量 i 的值(代码第 10 行)。

【例 3-10】 求满足 $1+2+3+\cdots+n \geqslant 1000$ 时,n 的最小值。

程序代码如下:

```
1    public class Example3_10
2    {
3        public static void main(String args[])
4        {
5            int n =0, sum =0;
6           while(sum <1000)
7            {
8                n =n +1;
9                sum =sum +n;
10           }
11           System.out.println("sum="+sum+" "+"n="+n);
12       }
13   }
```

程序运行结果如下：

```
sum=1035  n=45
```

程序分析如下：

题目要用累加的思路来解决，设计条件型循环，第 6 行代码对应着循环控制条件。

3.3.3 do 语句

do 语句与 while 语句较为相似，具有如下的语法格式：

```
do
    循环体语句块;
while(布尔型表达式)
```

do 语句的特点是首先要执行一次循环体语句块，再去判断表达式是否为真。若为真则继续执行循环体，否则退出 do 循环结构。因此，do 语句的循环体至少被执行一次。

【例 3-11】 已知 π 的近似值可用如下公式表示：

$$\frac{\pi}{4} \approx 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + (-1)^{n-1} \frac{1}{2n-1}$$

利用该公式求 π，要求最后一项的绝对值小于 0.000 000 01。

程序代码如下：

```
1    public class Example3_11
2    {
3        public static void main(String args[])
4        {
5            double r=0, sum=0;
6            int i=1,j=-1;
7            do
8            {    j=j*(-1);
9                r=(double)1/(2*i-1);
10               r=r*j;
```

```
11              sum=sum+r;
12              i=i+1;
13          }
14          while((Math.abs(r)) >=0.00000001);
15          sum=sum * 4;
16          System.out.println("π 的近似值为: "+sum);
17      }
18  }
```

程序运行结果如下：

```
π 的近似值为: 3.1415926735902504
```

程序分析如下：

编程所采用的思路仍然是累加，代码第 11 行是关键。代码第 8～10 行都是为正确求得累加和所做的处理。

3.3.4 循环嵌套

3.3.1～3.3.3 节介绍的都是单层循环。实际应用中常涉及在循环体语句块内又包含其他循环的结构。一个循环结构中又包含一个或多个循环结构被称为循环嵌套，或称多重循环。

设计多重循环时，往往根据需要确定嵌套的层数。有几层嵌套，就称为几重循环，如二重循环、三重循环等。循环嵌套的执行过程是，外层循环每执行一次，内层循环要从头到尾执行一遍。

设计循环嵌套时要注意，内层循环变量与外层循环变量不能同名。

为了增强可读性，循环嵌套的程序应当采用缩进方式书写代码，也就是要使程序具有锯齿形样式。

【例 3-12】 将例 3-8 用循环嵌套的思路重新设计。

```
1   public class Example3_12
2   {
3       public static void main(String args[])
4       {
5           int i=1, j, h, k;
6           System.out.println("全部水仙花数:  ");
7           while(i<=9)
8           {
9               j=0;
10              while(j<=9)
11              {
12                  h=0;
13                  while(h<=9)
14                  {
15                      k=i * i * i+j * j * j+h * h * h;
16                      if(k==100 * i+10 * j+h)
```

```
                    {
                        System.out.print(k+"  ");
                    }
                    h++;
                }
                j++;
            }
            i++;
        }
    }
}
```

程序运行结果不变。

程序分析如下：

定义变量 i、j、k 分别代表百位、十位、个位数字，使用三重循环产生 100～999 的整数，最内层循环(第 15～20 行代码)用于判定当前的三位数是否符合水仙花数的规则。

【例 3-13】 用循环嵌套语句生成如图 3-7 所示的图形。

```
#
###
#####
#######
#########
###########
#############
###############
#################
###################
```

图 3-7　循环嵌套程序运行结果

程序代码如下：

```
public class Example3_13
{
    public static void main(String args[])
    {
        int i=1, j=1;
        do
        {
            j=1;
            do
            {
                System.out.print("#");
                j++;
            }
            while(j<=2*i-1);
```

```
            System.out.print("\n");
            i++;
        }
        while(i<=10);
    }
}
```

程序分析如下：

设计循环嵌套结构，外层循环变量 i 的值从 1 变化到 10，一共 10 轮；内层循环变量 j 的值从 1 变化到“2 * i－1”，控制输出“2 * i－1”个“＃”。

【例 3-14】 打印如下的九九乘法表。

```
1*1=1  1*2=2  1*3=3  1*4=4  1*5=5  1*6=6  1*7=7  1*8=8  1*9=9
2*1=2  2*2=4  2*3=6  2*4=8  2*5=10 2*6=12 2*7=14 2*8=16 2*9=18
3*1=3  3*2=6  3*3=9  3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27
4*1=4  4*2=8  4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36
5*1=5  5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45
6*1=6  6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54
7*1=7  7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63
8*1=8  8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72
9*1=9  9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

程序代码如下：

```
public class Example3_14
{
    public static void main(String args[])
    {
        System.out.println("九九乘法表");
        for(int i=1;i<=9;i++)
        {
            for(int j=1;j<=9;j++)
            {
                System.out.print(i +"*" +j +"=" +i*j);
                if(i*j<=9)
                {
                    System.out.print("  ");
                }
                else
                {
                    System.out.print(" ");
                }
            }
            System.out.print("\n");
        }
    }
}
```

程序运行将输出题目要求的结果。程序设计了二重循环,第 6 行代码对应外层循环,用于行循环变量的控制(i:1～9);第 8 行代码对应内层循环,用于列循环变量的控制(j:1～9)。外层每循环 1 轮,内层要循环 9 次。第 10 行代码是关键,用于产生乘法公式。第 11～18 行代码属于输出格式调整语句,用于使同一列上下对齐,其中第 13 行输出两个空格,第 17 行输出一个空格。也可以将第 11～18 行代码删去,同时将第 10 行代码改为

```
System.out.print(i+"*"+j+"="+i*j+"\t");
```

也可以达到同样的效果,而且整个程序显得更为简练。

3.4 转移语句

转移语句有两个:break 语句和 continue 语句。

3.4.1 break 语句

在 3.2.4 节中已经出现了关键字 break,其作用是跳出 switch 部分,转向多分支结构后面的语句。在循环结构中也可以添加 break 语句,这时候它的作用是跳出循环体。应当注意,break 只能跳出其所在的最内层循环体。如果需要跳出多重循环,则要使用带标号的 break 语句。其语法格式如下:

```
break [标号];
```

【例 3-15】 将例 3-10 用含有 break 语句的循环结构实现。

程序代码如下:

```
1   public class Example3_15
2   {
3       public static void main(String args[])
4       {
5           int n=0, sum=0;
6           while(true)
7           {
8               n++;
9               sum+=n;
10              if(sum>1000)
11                  break;
12          }
13          System.out.println("n="+n+" "+"sum="+sum);
14      }
15  }
```

程序运行结果如下:

```
n=45  sum=1035
```

可以看到,使用含 break 语句的循环结构解决一些问题比较方便。

在多重循环中，若要连跳几层，甚至直接跳出最外层，则要使用带标号的 break 语句，例如：

```
label1:
    while(…)
    {
        while(…)
        {
            for(…)
            {
                if(…)
                    break label1;
            }
        }
    }
```

3.4.2 continue 语句

continue 语句和 break 语句相比，它不是跳出循环体，而是中断执行当前循环体的剩余部分，并准备进入下一轮循环。其语法格式如下：

```
continue [标号];
```

【例 3-16】 用含有 continue 语句的循环结构求 1～100 的所有奇数和。

程序代码如下：

```
1    public class Example3_16
2    {
3        public static void main(String args[])
4        {
5            int total=0, i;
6            for(i=1; i<=100; i++)
7            {
8                if(i %2 ==0)
9                    continue;
10               total+=i;
11           }
12           System.out.println("total="+total);
13       }
14   }
```

程序运行结果如下：

```
total=2500
```

程序分析如下：

当 i 为偶数时，continue 语句将使得程序跳转。此时循环体语句块的剩余部分（第 10 行代码）不再执行，转向执行将循环控制变量 i 的值加 1，再判断是否进行下一轮循环。

3.5 数　　组

Java 中有序数据的集合可以用数组来表示，一般一个数组中的元素属于同一种数据类型。数组在内存中对应着一段连续存储区域，数组名是这块区域的起始地址。利用数组名和下标可以方便地访问每一个数组元素。数组下标从 0 开始，数组的长度即为数组元素的个数。数组根据其下标个数的不同分为一维数组、二维数组和多维数组，这里重点介绍前两种。

3.5.1 数组的声明

从上一个程序中可以看到，如果首次使用变量 i，要通过语句 int i；和 i＝1；来完成声明和初始化操作，也可以通过语句 int i＝1 来完成。类似地，首次使用一个数组也需要先声明和初始化。

1. 声明一维数组

声明一维数组的语法格式有两种：

```
类型　数组名[];
类型[]　数组名;
```

这两种格式的作用相同。其中，类型对应于 Java 语言中的任意数据类型；数组名应当是一个合法的标识符，[]表明这是个数组类型的变量。例如：

```
double a[];
long array1[], array2[];
```

前者声明了一个 double 型的数组 a，后者声明了两个 long 型的数组 array1 和 array2。它们也可以写成下面的格式：

```
double[] a;
long[] array1, array2;
```

2. 声明二维数组

声明二维数组的语法格式有 3 种：

```
类型 数组名 [][];
类型 [][] 数组名;
类型 [] 数组名[];
```

例如，下面声明了 3 个 short 型二维数组，采用的是 3 种不同的格式：

```
short Arr1[][];
short[][] Arr2;
short[] Arr3[];
```

3.5.2 数组的初始化

数组被声明后，仅代表想创建一个数组，系统并没有为其分配内存储空间。只有将数组

初始化以后，才标志着以数组名为内存首地址的一段连续存储区域被分配给数组。数组的初始化有两种：静态初始化和动态初始化。

1. 静态初始化

在声明数组后，直接对其赋值，根据值的类型及个数，系统为它分配内存储区域，并保存下来首地址，这就是数组的静态初始化。例如，一维数组的静态初始化如下：

```
float price[];
price={13.7f, 28.8f, 30.2f};
```

上述语句创建了数组 price，含有 3 个 float 型的下标变量，并为其分配内存储区域，它们是 price[0]、price[1]和 price[2]，对应的值为 13.7f、28.8f 和 30.2f。

另外，采用将数组声明和初始化合并的写法，会显得较为简练。例如，将上面两行代码合并为：

```
float price[]={13.7f,28.8f,30.2f};
```

【例 3-17】 对二维数组进行静态初始化，再输出全部数组元素。

```
1    public class Example3_17
2    {
3        public static void main(String args[])
4        {
5            int array[][]={{6,2,7}, {5,2,0}, {3,9,1}};
6            for(int i=0; i<=2; i++)
7            {
8                for(int j=0; j<=2; j++)
9                {
10                   System.out.print("array"+"["+i+"]"+"["+j+"]"+"=");
11                   System.out.print(array[i][j]+" ");
12               }
13                 System.out.print("\n");
14           }
15       }
16   }
```

程序运行结果如下：

```
array[0][0]=6  array[0][1]=2  array[0][2]=7
array[1][0]=5  array[1][1]=2  array[1][2]=0
array[2][0]=3  array[2][1]=9  array[2][2]=1
```

程序分析如下：

第 5 行代码完成了对二维数组 array 的声明和静态初始化操作。由此可知，数组 array 第一维和第二维的长度均为 3，故一共包含 9 个 int 型的下标变量，它们是 array[0][0]、array[0][1]、array[0][2]、array[1][0]、array[1][1]、array[1][2]、array[2][0]、array[2][1]和 array[2][2]。第 6～14 行代码用于输出这 9 个数组元素，采用二重循环结构，设计了 3 行 3 列的显示样式。第 13 行代码用于换行。