第3章　控制语句

执行程序时，总是从程序的开始到结尾一条接一条顺序执行。如果希望改变程序的执行顺序，需要使用控制语句。Java 程序有三种结构：顺序结构、选择结构和循环结构。

(1) 顺序结构。程序从上到下一行一行的执行，中间没有判断和跳转，直到程序结束。默认的执行方式总是采用顺序结构进行。

(2) 选择结构。在满足条件时才执行，主要有 if 语句和 switch 分支语句。

(3) 循环结构。在某种条件下使程序循环执行，主要有 for 循环、while 循环和 do…while 循环。

本章主要介绍选择和循环结构。

3.1　if 语句

3.1.1　if 语句的三种形式

if 语句是使用普遍的条件语句，它有多种形式。

1. 第一种形式

```
if(条件语句)
{
    执行语句块
}
```

条件语句可以是任何一种逻辑表达式，如果条件语句的返回结果为 true，则先执行后面大括号({})内的语句，然后再执行后面的其他程序代码；如果为 false，则跳过大括号内的内容，直接执行后面的程序代码。大括号的作用就是将多条语句组合成一个复合语句，作为一个整体来处理。如果大括号内只有一条语句，也可以省略大括号。

例如，要判断存款金额是否大于 1 元，如果小于 1 元就让用户重新输入。

输入小数的语句为：

```
System.out.println("请输入要存入的金额…");
double  deposit= sc.nextDouble();                    //输入要存入的金额
```

选择语句如下：

```
if(deposit<1.0){
    System.out.println("存入金额小于 1 元，请重新输入要存入的金额…");
}
```

在上面的例子中用到了“小于”运算符(<)。回顾第 2 章介绍的比较运算符，如表 3.1 所示。

表 3.1　比较运算符

运算符	表 达 式	名称及功能	运算符	表 达 式	名称及功能
＞	A＞B	大于	＜＝	A＜＝B	小于等于
＞＝	A＞＝B	大于等于	＝＝	A＝＝B	等于
＜	A＜ B	小于	!＝	A!＝B	不等于

这些比较运算符只适用于基本数值的比较，不适用于字符串。比较相等关系使用双等号(＝＝)，注意不是单等号(＝)。例如判断密码是否为 123。

```
if(password==123)
{
    System.out.println("密码正确。");
}
```

2. 第二种形式

```
if(条件语句)
{
    执行语句块 1
}
else
{
    执行语句块 2
}
```

这种格式在 if 语句后面添加了 else 从句。表示在 if 判断条件为 false 时，执行 else 后面的从句。例如，判断成绩是否大于 60，大于显示及格，否则显示不及格。

```
int mark=sc.nextInt();
    if (mark>=60)
      System.out.println(" 恭喜你，通过了考试。");
    else
      System.out.println("抱歉，没有通过考试。");
```

if 判断需要多个条件组合时，使用逻辑运算符。例如，成绩在 90～100 分之间显示为优秀。

```
if(mark>=90 && mark<=100)
    System.out.println("恭喜你，成绩为优秀。");
```

常用的逻辑运算符有三种，如表 3.2 所示。

表 3.2　常用的逻辑运算符

逻辑运算符	对应符号	逻辑运算符	对应符号
与	&&	非	!
或	‖		

“与”和“或”运算符是将两个判断连接起来，得到一个最终结果。“与”运算符需要两个判断结果都是真时才会得到为真的结果。“或”运算符只需要至少有一个结果是真即为真。“非”运算符将真或假反转。

3. 第三种形式

if 的嵌套，嵌套允许处理多重选择，使用方法如下：

```
if(条件语句 1 )
{
    执行语句块 1
}
else  if(条件语句 2)
      {
        执行语句块 2
      }
      else if (条件语句 3)
      {
        执行语句块 3
      }
...
```

3.1.2 分段显示实例(if)

【例 3.1】 使用 if 嵌套语句实现分段显示上课时间(分 A、B、C 时段)。

```
//TimeTable.java
import java.util.*;
public class TimeTable {
  public static void main(String args[]){
    char group;                                    //上课时段分组
    Scanner sc=new Scanner(System.in);
    System.out.print("Enter your group(A,B,C)");
    group=sc.next().charAt(0);
      if(group=='A')
          System.out.println("8:00 a.m");
      else
       {  if(group=='B')
              System.out.println("10:00 a.m");
            else
            {
                 if(group=='C')
                  System.out.println("1:00 p.m");
                  else
                  System.out.println("没有这个时段");
            }
      }
```

```
  }
}
```

实例中读取键盘输入单个字符,使用语句:

```
group=sc.next().charAt(0);
```

sc.next()方法得到键盘输入的一个字符串 String。String.charAt(0)是获取字符串的第一个字符。

if 嵌套,代码中会出现许多括号,难以阅读。嵌套的 if 语句虽然实现了多重条件选择的任务,但当选择条件增大时,程序会显得混乱。多条件选择还有另一种形式——switch 语句,下面讨论 switch 选择。

3.2 switch 语句

3.2.1 分段显示实例(switch)

【例 3.2】 使用 switch 语句解决例 3.1 中程序的问题。观察这个程序,然后加以讨论。

```
//TimeTable2.java
import java.util.*;
public class TimeTable2 {
  public static void main(String args[]){
    char group;                                    //上课时段分组
    Scanner sc=new Scanner(System.in);
    System.out.print("Enter your group(A,B,C)");
    group=sc.next().charAt(0);
    switch  (group)                                //开始使用 switch 语句
    {
        case 'A':  System.out.println("8:00 a.m");
                   break;
        case 'B':  System.out.println("10:00 a.m");
                   break;
        case 'C':  System.out.println("1:00 p.m");
                   break;
     default:    System.out.println("没有这个时段");
    }                                              //switch 语句结束
}
}
```

使用 switch 语句程序简洁了许多。switch 语句与一组 if 语句的运行方式完全相同,但更为紧凑易读。switch 语句用于以下情形:

(1) 每个条件只检查一个变量(例 3.2 中是 group);

(2) 检查涉及变量的具体值(如 A、B)而不是范围(如>=60)。

3.2.2 switch 语句详解

switch 语句格式如下：

```
switch(表达式){
 case  取值 1:   语句块 1;  break;
 case  取值 2:   语句块 2;  break;
 case  取值 3:   语句块 3;  break;
 …
 default:       语句块 n;  break;
}
```

表达式是待测试的变量名称。switch 语句判断条件可以接受 int、byte、char 和 short 型，JDK7.0 以后的版本都支持 string 字符串类型。

switch 执行时，case 语句依次执行，一旦找到匹配者，就从该位置顺序执行，直至遇到 break 语句强制退出。因此，如果没有 break 语句，程序会执行从匹配位置开始到 default 的所有语句。

default 语句是可选项，可以没有。

case 中的值必须是常量，并且所有 case 子句的值应该是不同的。

break 语句用来在执行完一个 case 分支后，使程序跳出 switch 语句，即终止 switch 语句的执行。

考虑这样一个问题：用同一段语句来处理多个 case 条件，比如例 3.2 中 A、B 是同一时段，程序该如何编写？示例代码如下：

```
switch  ( group )                                  //开始使用 switch 语句
{
    case 'A':
    case 'B':     System.out.println("8:00 a.m");
                 break;
    case 'C':  System.out.println("1:00 p.m");
                 break;
    default:    System.out.println("没有这个时段");
}                                                  //switch 语句结束
```

这个例子中，当选择 A、B 时显示的是同一时间。当 group 值为 A 时，检测到 case 'A'满足条件，程序从 case 'A'后的语句开始执行，直至遇到 break 语句停止。因此当 group 的值是 A 或者 B，执行效果一样。

3.3 for 循 环

3.3.1 for 循环语法

循环次数已知，一般使用 for 循环实现，语法如下：

```
for ( 初始化表达式; 循环条件表达式;  循环后的操作表达式)
```

```
{
    执行语句;
}
```

for 语句第一行是循环声明,第二行是循环体。大括号是可选部分,如果执行语句只有一条,大括号可以省略。for 语句执行时,首先执行初始化操作,然后判断终止条件是否满足,如果满足,则执行循环体中的语句,最后执行迭代部分。完成一次循环后,重新判断终止条件。

在初始化部分和迭代部分可以使用逗号语句来进行多个操作。逗号语句是用逗号分隔的语句序列。

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

3.3.2 求和运算实例

【例 3.3】 计算 10 以内整数的和。

```
//ForSample.java
class ForSample {
  public static void main(String args[]){
      int sum=0;
      for(int i=0; i<10; i++)          //控制循环 10 次,每次执行 sum+i
      {
           sum=sum+i;
      }
      System.out.println("sum="+sum);
    }
}
```

for 循环体内可以包含任意数量的指令,包括 if 语句、switch 语句甚至可以是另一个循环。比如要打印出 10×4 的“*”组成的方阵。

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

使用一条 for 语句时,可以采用如下形式:

```
for(i=0; i<4; i++)
{
     System.out.println("**********");
}
```

实例用 i 控制了打印“**********”的行数。如果每一行的“*”也用循环完成,需要嵌套另一个 for 循环完成。

```
for(i=0; i<4; i++)                    //外部循环,实现打印出多行
{
    for (int j=0; j<9; j++){
        System.out.print("* ");       //内部循环实现打印出一行“*”
    }
    System.out.println();             //每行结束后换行
}
```

【例 3.4】 显示九九乘法表。

编程思路：九九乘法表结构与上题类似,也是行列矩阵。使用嵌套的 for 循环可以实现行列的控制,难点在于如何显示每行信息。观察九九乘法表可得,每行显示的列数与该行的行数相等。因此,每行显示内容为从 1×j 到第 i 行的 i×j。具体程序如下：

```
//ForLoopStatement.java
public class ForLoopStatement {
  public static void main(String[] args){
    int i, j;
    for (i=1; i<10; i++){             //控制行
      for (j=1; j<=i; j++)            //控制列
        System.out.print(i+"×"+j+"="+(i*j)+"  ");
        System.out.println();
    }
  }
}
```

运行结果为：

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

3.4 while 循环

3.4.1 while 循环语句

for 循环是一般用于循环次数已知的结构。有时循环次数未知,就可以考虑 while 循环或者 do…while 循环。

while 循环用于不知道代码需要重复多少次,但有明确的终止条件的情况。语法如下：

```
while ( 条件表达式 )
```

```
{
    执行语句
}
```

当条件表达式为真时，执行{}中的执行语句，执行完后再返回判断条件，直到条件是假，循环终止。

3.4.2 while 循环实现输入控制

考虑成绩的输入问题。输入成绩时，成绩必须大于 0，小于 100。程序在接收用户输入时，应该判断成绩是否有效。如果无效就让用户重新输入，直到输入了有效成绩。

伪代码表达如下：

```
提示用户输入成绩
用户输入
循环判断成绩是否小于 0 或者大于 100
begin
    显示错误信息
    用户再次输入
end
```

在用户输入无效成绩时显示错误信息。用户可能会多次输入无效成绩，就需要循环处理。由于循环次数未知，因此考虑使用 while 循环实现。

```
System.out.println("请输入成绩:");
int mark=sc.nextInt();
while (mark<0 || mark>100)
{
    System.out.println("无效的成绩,请再次输入成绩:");
    mark=sc.nextInt();
}
```

3.5 do…while 循环

3.5.1 do…while 语句

do…while 语句功能和 while 语句类似，不过 do…while 语句是执行完第一次后才检测条件表达式。这意味着包含在大括号中的程序段至少要被执行一次。do…while 的语法结构如下：

```
do {
    执行语句
} while (条件表达式);
```

注意：do…while 的 while（条件表达式）后有“;”号，表示语句结束。而 while 循环的表达式后没有分号。

3.5.2 do…while 实现退出操作

do…while 适用于循环次数未知，并且循环体至少执行一次的结构。

程序一般是任务完成就会自行终止。如果希望多次执行同样的代码，可以将代码放入循环体内，直到用户选择退出时终止。由于循环次数未知，不宜使用 for 循环；while 循环要在程序开始时就判断是否重复，也不合理；最佳的方法是使用 do…while 循环。

```
char response='n';
 do {
    //执行的任务
    System.out.println("是否重复执行程序?(y/n) ");
    response=sc.next().charAt(0);
} while (response=='y');                    //如果选择 y,则返回循环体内部执行
```

3.6 break 与 continue

循环语句只有循环条件表达式为假时才结束循环。如果希望提前中断循环，使用 break 语句就终止循环。也可以使用 continue 语句跳过本次循环，然后开始执行下一次循环。

3.6.1 break 语句

在 switch 语句中已经使用过 break，它的作用是使程序从它所在 switch 语句中跳出。break 语句也可以用在循环语句中。break 语句在循环中有两种使用形式：

(1) 不带标号的 break 语句。

```
break;
```

(2) 带标号的 break 语句。

```
break  标号;
```

不带标号 break 是中断当前循环体的执行。对于多重循环，执行不带标号的 break 只能使程序从所在的那种循环中跳出。

带标号的 break，标号是一个标识符，用来标示某一程序块。Java 语言中，标号的定义只能出现在循环语句之前，在标号和循环语句之间不能插入任何其他语句。形式如下：

```
标号:循环语句
```

带标号的 break 语句可以中断多重循环，使程序流程跳转到标号标示的循环体之外。例如：

```
st: while(true)                             //标号 st 所在循环
{
    while (true)
      {
```

```
            break st;                    //终止两种 while 循环
        }
    }
```

执行 break st;程序会跳出最外层的 while 循环。如果不使用 st 标号,程序只能跳出里层的 while 循环。

3.6.2 continue 语句

continue 语句只能出现在循环语句(while、do、for)中。它也有两种形式:带标号的和无标号的。带标号的使用方法同 break。无标号的 continue 语句是跳过当前循环的剩余语句块,接着执行下一次循环。请看下面打印 100 以内 7 的倍数的例子。

【例 3.5】 打印 100 以内 7 的倍数。

算法设计:100 以内的循环,循环次数已知,可以使用 for 循环实现。当计数器为 7 的倍数,打印计数器;如果不是,则什么也不做,继续计数。

源程序:

```
//Find7Times.java
class Find7Times {
    public static void main(String args[]){
        for (int i=1; i<=100; i++){
            if(i%7==0)
                System.out.print(i+"   ");
            else
                continue;              //结束本次循环,继续执行下一次循环
        }
    }
}
```

运行结果为:

```
7  14  21  28  35  42  49  56  63  70  77  84  91  98
```

3.7 综合实例:十进制与二进制转换

实例将十进制数字转换为二进制数字,转换方法是基于十进制数字的分解。例如,19 转换为二进制为 10011,因为 19 可以表示成 19=16+2+1,即 $19=1\times2^4+1\times2^1+1\times2^0$。

进制转换可以用展开式中 2 的幂方表示。问题焦点是如何用 2 的幂方表示十进制数。

3.7.1 问题分析

从前面分析可知,十进制转换为二进制的关键是将十进制数字表示为不同 2 的幂方相加。因此,程序可以从最高幂比较。例如,19 在 $2^5\sim2^4$ 之间,最高幂是 2^4。程序首先需找到最接近 19,且不大于 19 的 2 的最高幂。然后依次将 2^3、2^2、2^1、2^0 与最高幂累加,如果大于 19 就补 0,如果小于 19 就补 1。

设计思路如图 3.1 所示。

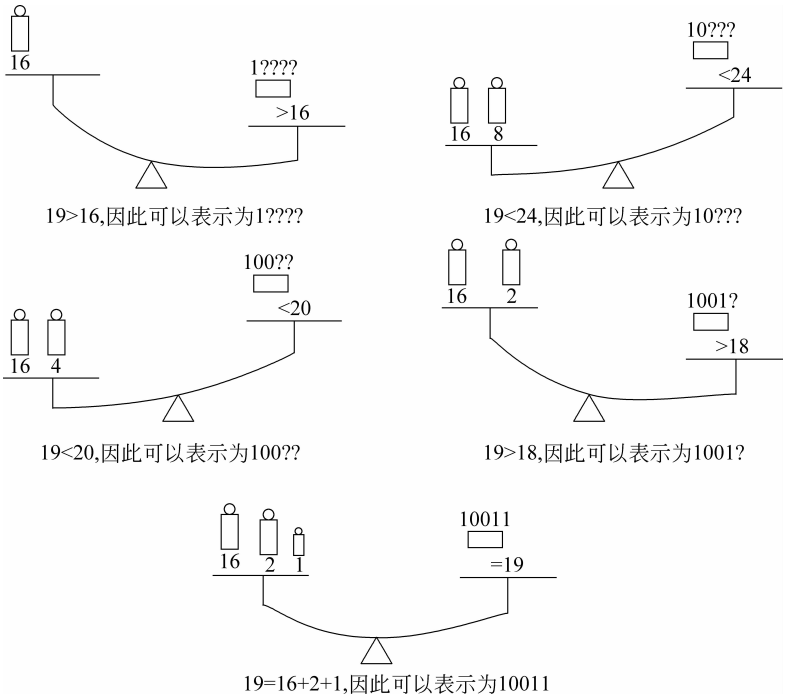

图 3.1　设计思路

3.7.2　算法设计

第一步需找到最高幂。设 n 为需要转换的数字，待求的最高幂为 v。循环除 2 可得到最高幂，方法如下：

```
int v=1;                                    //当前 2 的幂方
n=sc.nextInt();                             //n 为需要转换的数字
while (v<=n/2)
v=2*v;
```

求得最高幂 v 后，从最高幂 v 开始依次递减，与 $n-v$ 的余数相比，如果大于余数，则输出 1；小于余数则输出 0。

3.7.3　主程序

【例 3.6】 进制转换。

```
//Binary.java
import java.util.*;
public class Binary {
    public static void main(String args[]){
      Scanner sc=new Scanner(System.in);
      int  n=0;                             //待转换的十进制数字
```

```
        int  v=1;                          //当前 2 的幂方
        n=sc.nextInt();
        while(v<=n/2)
           v=2*v;
        int x=n;                           //当前余数
        while(v>0)
        {
            if(n<v)
                System.out.print(0);
            else
            {
                System.out.print(1);
                n-=v;
            }
            v=v/2;
        }
          System.out.println();
          }
    }
```

运行结果为：

```
20
10100
```

习　题　3

1. 判断以下语句是否有误？如果有错，请指出。

(1) if (a>b) then c=0;

(2) if a>b { c=0;}

(3) if (a=b) c=0;

(4) if (a>b) c=0 else b=0;

2. 给出下列语句执行后 j 的值是多少？假设 i 和 j 都是整数。

(1) for (i=0, j=0; i<10; i++) j+=i;

(2) for (i=0, j=1; i<10; i++) j+=j;

(3) for (j=0; j<10; j++) j+=j;

(4) for (i=0, j=0; i<10; i++) j+=j++;

3. 给出 m 和 n 的运行结果。

```
int n=12345;
int m=0;
while (n!=0)
{
    m= (10*m)+(n%10);
```

```
    n=n/10;
}
```

4. 下列语句执行后 k 的值是多少?

```
int i=4,j=5,k=9,m=5;
      if(i>j||m<k)
      k--;
    else k++;
```

5. 下面程序的输出结果是什么?

```
public class Mystery
{
  public static void main (String[]args ){
    int y,x=1,total=0;
    while (x<=10)
  {
    y=x*x;
    System.out.println(y);
    total+=y;
    System.out.println("Total is"+total);
            }
        }
    }
```

编 程 练 习

1. 设计并实现一个要求用户输入两个数并猜测两数之和的程序。如果用户猜对结果,就显示祝贺消息,否则显示慰问信息以及正确答案。

2. 甲乙两人打赌,看谁赚的钱多。甲承诺每天给乙 1000 元钱。乙承诺第一天给甲 1 分钱,第二天给甲 2 分钱,第三天给甲 4 分钱,第四天给甲 8 分钱……请问 30 天后,甲乙谁赢?

3. 设计一个自动售货机,提供如下选择:

[1] 口香糖

[2] 巧克力

[3] 爆米花

[4] 果汁

[5] 显示购买总数

[6] 退出

允许用户连续的从这些选项中进行选择。当选中[1]~[4]选项时,显示适当的信息确认选项。例如当用户选择[3]时,可以显示如下信息:

您购买了爆米花

当用户选择[5]时,显示已经售出的每种商品的数量。例如:

```
您购买了 2 个口香糖
您购买了 3 个巧克力
您购买了 3 杯果汁
```

当用户选择[6]时,程序终止。如果输入[1]～[6]以外的选项,显示出错信息。例如:

```
错误,请输入[1]~[6]内的数字!
```

4. 在直角三角形中,一边长度的平方等于另外两边长度的平方和。编写程序,提示用户输入三角形三个边的长度,然后指出此三角形是否为直角三角形。

5. 编程求 1!+2!+3!+…+10!的值。

6. 编写程序,提示用户输入笛卡儿平面中某一点 x－y 坐标。程序应输出一条消息指出此点是原点,位于 x 或 y 轴上,还是在特定象限上。

例如:

```
(0,0)is origin
(4,0)is on the x-axis
(0,-8)is on the y-axis
(-2,5)is in the second quadrant
```

第4章　方　　法

方法表示类的行为，只能作为类的一部分存在。前面学习过 main()方法，本章介绍方法的定义和调用以及方法重载等问题。

4.1　定义方法

方法包括方法的声明以及方法体，语法如下：

```
返回值类型　方法名(参数列表)
{
    方法体
}
```

其中返回值类型是指调用方法后返回数据的类型，参数列表给出了方法参数的类型和名称，方法体是方法功能的实现。下面给出一个方法的代码片段。该方法的功能是在ATM取款时显示提示信息。

```
static  void  displayMessage(){                    //方法声明
    System.out.println("请注意保护个人密码。");      //方法体
    System.out.println("请输入取款密码……");
}
```

方法名为 displayMessage，该方法不需要输入参数，方法即使没有参数也要保留小括号()。该方法没有返回值，没有返回值时需要声明返回类型为 void。static 关键字表示该方法是静态方法，main 方法也是静态方法，静态方法会在后续章节详细介绍，这里只要知道静态方法可以直接调用。方法体内完成了输出提示信息的功能。

4.2　调用方法

Java 中除了 main 方法是系统自动调用外，其他方法必须明确被调用。通过方法名和参数列表调用方法，形式如下：

```
方法名(实际参数表);
```

实际参数又称为实参，用来初始化调用方法的参数列表。实际参数必须与方法定义中的参数一致，包括参数的个数和类型。如果不一致，则会编译出错。

【例 4.1】 方法声明和调用实例。

```
//MyMethod.java
import java.util.*;
  class MyMethod {
```

```
    static void displayMessage(){                    //方法定义
        System.out.println("请注意保护个人密码。");  //方法体
        System.out.println("请输入取款密码……");
    }
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        String myPSW;
        displayMessage();                            //调用 displayMessage()方法
        myPSW=sc.next();
    }
}
```

displayMessage()方法定义独立于 main 方法,它们是并列关系,都属于同一个类。方法定义都要在某个类的内部,不可以单独编写。方法定义在类中的顺序不会影响编译器的执行过程,因此 displayMessage()方法在 main 方法之前还是之后是无关紧要的。程序执行总是从 main 方法开始。在 main 方法中,执行到方法调用语句 displayMessage()时才会真正运行 displayMessage 方法。

需要强调的是,当一个方法调用另一个方法时,调用方法会在调用点暂停执行,而跳转到被执行方法中执行相应语句;当被调用方法结束时程序又返回调用方法中再次继续运行。调用过程如图 4.1 所示。例 4.1 是 main 方法调用其他方法,其实方法之间可以相互调用。

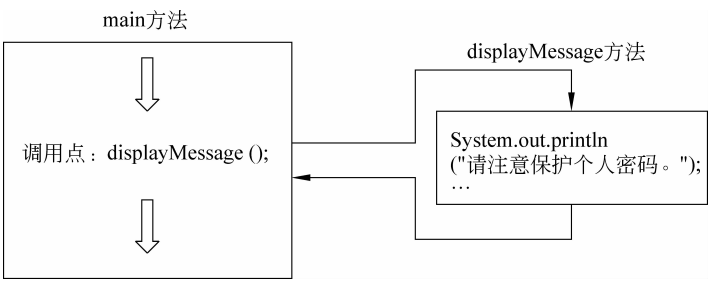

图 4.1　方法调用示意图

4.3　方 法 实 例

前面介绍的方法没有参数和返回值,较为简单。下面将介绍有参数和返回值的方法。

继续以第 2 章判断闰年为例,希望把闰年判断编写为独立的方法。计算闰年时,需要给定一个确定的年份,根据此年份才可计算是不是闰年。计算完毕,需要告诉调用者判断结果。因此,方法有一个输入参数——年份;还有一个返回值——是否为闰年的判断结果。方法体完成判定的任务。

【例 4.2】　闰年判断方法。

```
static  boolean  isLeapYear (int  yearIn)
  {
  boolean  resultIn=false;
```

```
    if((yearIn%4==0)&& (yearIn%100!=0) ‖(yearIn%400==0))
      resultIn=true;
  return resultIn;
  }
```

方法的首部,修饰为 static 静态的,表示方法可以直接调用。boolean 是返回值类型,表示方法调用时将返回 boolean 型的判断结果。isLeapYear 是方法名,调用要通过方法名实现。方法有一个整数类型参数 yearIn,表示在调用时需提供一个整数类型的参数,即待判断年。方法体将对 yearIn 计算,判断是不是闰年,并将 boolean 型的判断结果返回。

本例中,方法内部的变量命名均带有 In,表示在方法内部有效。最后要把判定结果返回,使用 return 语句:

```
return resultIn;
```

return 语句有两个功能。第一个功能是方法的结束语句,一旦程序运行到 return 语句,方法结束同时程序控制权跳回到调用方法。第二个功能是它可以携带参数,将参数返回给调用者。本例中它返回了判断结果。注意,返回值的类型必须与方法头部声明的类型一致。

方法什么时候结束运行呢? 通常发生在下列情况:

(1) 运行完方法体的最后一条语句。

(2) 执行到 return 语句。

编程人员根据方法的复杂性,可能在程序的多处安排有 return 语句,方法运行时只要执行到某个 return 语句,方法就立即结束,不管后面是否还有其他语句。

return 语句有两种形式:

```
(1) return;
(2) return 表达式;或 return  (表达式);
```

(1)适合于方法返回值类型是 void 的方法;(2)适合于方法返回值类型非 void 的方法。换句话说,void 返回类型的方法不需要也不能返回数据,而返回值类型为其他类型的方法必须要使用语句 return 表达式。

例如 main 方法,返回值类型是 void,因此无须返回数据(main 方法结束就是程序结束,相当于返回到操作系统对方法的调用处)。接下来讨论调用带参数的方法。

【例 4.3】 带参数和返回值的方法。

```
//LeapYear.java
import java.util.*;
public  class LeapYear {
    static boolean isLeapYear(int yearIn){
            boolean  resultIn=false;
    if((yearIn%4==0) && (yearIn%100!=0) ‖
                                (yearIn%400==0))
            resultIn=true;
            return resultIn;
```

```
        }
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        int year;
        boolean result;
        year=sc.nextInt();
        //调用 isLeapYear 方法,传入实际参数 year
        result=isLeapYear(year);
        System.out.println(year+" is  Leap Year?"+result);
    }
}
```

正如前面介绍过,Java 程序运行时虚拟机会先找到 main 方法,从 main 方法始执行。在 main 方法中,当执行到调用方法语句:

```
result=isLeapYear(year);
```

程序会跳转到 isLeapYear 方法内部去执行。执行前,先把实际参数(year)赋值给形式参数(yearIn):yearIn = year。在 isLeapYear 方法内部执行到 return resultIn 语句,会把 resultIn 的值返回给 main 方法。接着回到 main 方法中,把执行结果赋给布尔变量 result。这就是方法的参数传值的整个过程,如图 4.2 所示。

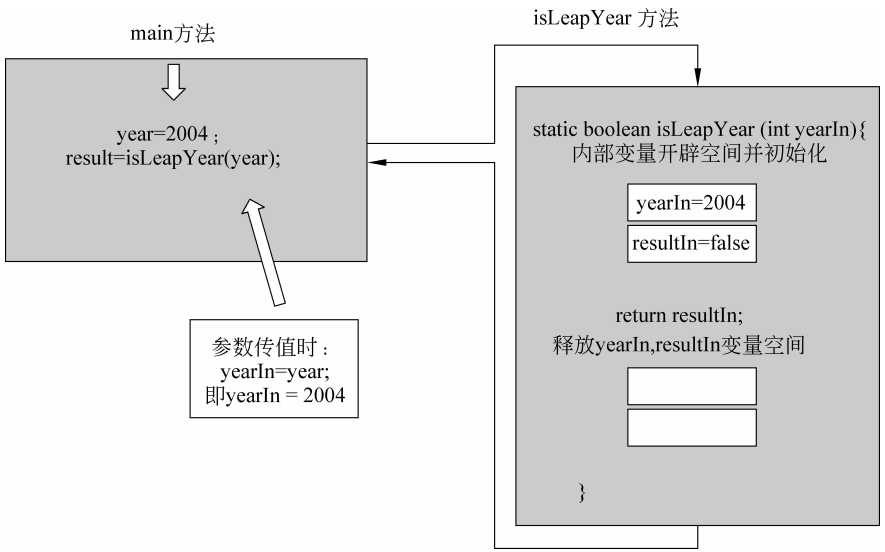

图 4.2　方法参数传递示意图

要注意变量的作用域问题。在方法内部定义的变量,仅在方法内部有效。例如 isLeapYear 方法中,yearIn、resultIn 变量是方法内部变量。方法调用时会创建存储两个变量的内存空间,方法执行完毕会释放空间。因此,main 方法无法访问到 yearIn、resultIn 变量。

4.4 方法应用

在定义方法的过程中,一般要考虑4个问题:

(1) 方法名。

(2) 方法的输入(即形参)。

(3) 方法的输出(即返回值)。

(4) 方法体(完成相应功能的语句)。

方法名的定义要符合标识符规则,尽量做到见名知义。如果方法名由多个单词组成,一般第一个单词全部小写,后续单词的首字母要大写,如myMethod、isEven。

4.4.1 单个参数的方法

【例4.4】 计算圆的面积方法。

编程思路:计算圆的面积,方法命名为circleArea。要完成计算,必须知道圆的半径,因此方法需要一个输入参数——半径。计算完毕,需要把计算结果返回给调用方法main,因此也需要一个返回值。计算面积的算法是πr^2,方法体内按照公式计算得到结果。用方法实现计算圆的面积:

```
//MethodExp3.java
public class MethodExp3 {
    static double circleArea(double radiusIn){
        final double PI=3.14;                          //定义常量 PI
        //根据传入参数计算圆的面积,并将结果返回
        return (PI * radiusIn * radiusIn );
    }

    public static void main(String[] args){
        double result=0.0;
        //调用 circleArea 方法,计算半径数值为 2.4 的圆面积
        result=circleArea(2.4);
        System.out.println(result);
        //调用 circleArea 方法,计算半径数值为 4.0 的圆面积
        System.out.println(circleArea(4.0));
    }
}
```

程序中通过方法名和参数实现调用,调用语句为:

```
result=circleArea(2.4);
System.out.println(circleArea(4.0));
```

调用时,传入的参数即实参赋值给形参(radiusIn),即

```
radiusIn=2.4;                                          //第一次调用
radiusIn=4.0;                                          //第二次调用
```

两次调用后，处理方法有所不同。第一次调用，将结果返回给 main 方法内定义的变量 result；第二次调用，直接将结果输出显示。

4.4.2 多个参数的方法

方法有时需要多个参数，下面通过具体问题来了解多参数方法的处理。

【例 4.5】 实现由任意符号组成的 i * j 的矩形。

编程思路：方法的功能是绘制矩形，方法命名为 drawRectangle。绘制时，要知道绘制符号，以及 i * j 的值。因此，输入参数应该有三个，分别为绘制符号、矩形宽和矩形高。方法体实现矩形的输出显示，不需要返回结果，因此返回值类型是 void。方法体内绘制算法，回顾第 3 章介绍过的 for 循环的实例，它实现了由 10×4 个 * 组成的矩形：

```
for(i=0; i<4; i++)                                    //外部循环，实现打印出多行
  {
    for( int j=0; j<9; j++){
    System.out.print("* ");                           //内部循环，实现打印出一行 *
}
System.out.println();                                 //每行结束后换行
}
```

在此算法中，只要把固定数值和符号转换为传入的参数就可以实现。

```
//MethodExp4.java
public class MethodExp4 {
    //三个输入参数的方法
static void drawRectangle(char symbolIn, int xIn, int yIn)
{    //外部循环，实现打印出多行
     for( int i=0; i<yIn; i++){
     //内部循环，实现打印出一行 symbolIn
     for(int j=0; j<xIn; j++){
        System.out.print(symbolIn);
     }
     System.out.println();                            //每行结束后换行
    }
}
  public static void main(String[] args){
    drawRectangle('@',8,4);                           //调用 drawRectangle 方法
    System.out.println();
    drawRectangle('#', 15, 5);                        //调用 drawRectangle 方法
    }
}
```

运行结果为：

```
@@@@@@@@
@@@@@@@@
@@@@@@@@
```