第 3 章 控制结构

CHAPTER

计算机程序是由若干条语句组成的语句序列，但是程序的执行并不一定按照语句序列的书写顺序。程序中语句的执行顺序称为“程序结构”。如果程序中的语句是按照书写顺序执行的，称其为“顺序结构”，它是最基本的程序结构；如果某些语句是按照当时的某个条件来决定是否执行，称其为“分支结构”；如果某些语句要反复执行多次，称其为“循环结构”。

3.1 分支结构

分支结构表示在某一条件成立的情况下，进行分流行为的程序结构。分支结构在 Java 语言中有两种形式：条件分支结构与开关分支结构。其中条件分支又有单分支、双分支的基本形式。将分支基本结构嵌套就成为多分支结构，而开关分支结构则是另一种多分支结构。

3.1.1 if 语句

Java 的 if 语句与其他编程语言中所用的 if 语句类似，分为单分支、双分支的基本形式。

单分支结构的 if 语句为

```
if(条件表达式) 语句组
```

其中，if 括号中的条件表达式的值为 true 时，其后的语句组将被执行；当条件表达式值为 false 时，其后的语句组将不被执行。if 语句的逻辑如图 3-1 所示。

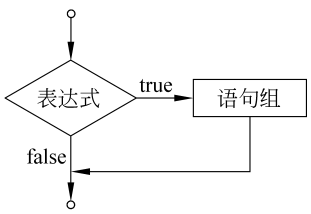

图 3-1 if 语句逻辑

双分支结构也称为 if-else 结构，其语句格式为

```
if(条件表达式) 语句组 1
else 语句组 2
```

其中，if 括号中的表达式值可由任何表达式求得，当表达式值为 true 时，表示 if 括号中的条件为逻辑真，其后的语句组 1 将被执行；当表达式值为 false 时，表示 if 括号中的条件为逻辑假，则语句组 2 将被执行。因此，语句组 1 与语句组 2 在双分支结构中必定会有一个将被执行。if-else 语句的逻辑如图 3-2 所示。

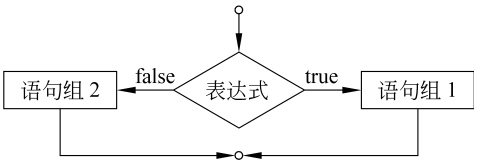

图 3-2　if-else 语句逻辑

if 型的多分支结构的语句实际上是由 if 结构嵌套而成的，其过程及分析与单、双分支情况相同。在理论上，其嵌套的深度（层次）没有限制，但嵌套太深，运行速度变慢，可读性差，查错、调试困难，因此，除了非用此法的情况外，一般可用开关分支结构来代替多路分支结构。

多分支结构的 if 语句为

```
if(条件表达式 1)
  if(表达式 1_1) 语句组 1_1
    else 语句组 1_2
else
  if(条件表达式 2) 语句组 2_1
    else 语句组 2_2
```

在多分支结构中，每个 if 要与对应的 else 配对，当有多个 if 结构嵌套时，else 将与最近的 if 配对。若 if 需要与另一个 else 配对，就可将该 if 与 else 之间的所有语句用一对花括号“{}”括起，这样就改变了 if 与最近的 else 配对的原则。

Java 没有 end if 关键字，所以必须根据约定使用正确的缩排和空白，以使代码易于读取，逻辑易于解释。

例 3-1　if 语句的应用。

程序清单：ch03\IfDemo.java

```
public class IfDemo{
    public static void main(String[] args){
        int c1=2;   int c2=3;
        if (c1==1){
            System.out.println("c1=1");
        }
        else if(c2<=1){
            System.out.println("c2<=1");
        }
        else{
            System.out.println("c1 !=1, c2>1");
        }
    }
}
```

程序运行结果：

```
c1!=1, c2>1
```

例 3-2　将一个字符串中的小写字母变成大写字母，大写字母变成小写字母。

程序清单：ch03\ChangeLetter.java

```
import java.util.*;
public class ChangeLetter{
public static void main(String args[]){
     String s=new String("abcABC123");
     System.out.println(s);
     char a[]=s.toCharArray();
     for(int i=0;i<a.length;i++){
      if(Character.isLowerCase(a[i])){
         a[i]=Character.toUpperCase(a[i]);
      }
      else
        if (Character.isUpperCase(a[i])){
            a[i]=Character.toLowerCase(a[i]);
        }
     }
      s=new String(a);
      System.out.println(s);
}
}
```

程序运行结果：

输入：

```
abcABC123
```

输出：

```
ABCabc123
```

3.1.2　switch 语句

当程序中多分支结构的各分支之间没有相互依赖关系时，可以采用 switch 开关分支结构，其语句为

```
switch(整型/字符型表达式)
{
  case 整型/字符常量 1: 语句 1; [break;]
  case 整型/字符常量 2: 语句 2; [break;]
   ⋮
  case 整型/字符常量 n: 语句 n; [break;]
```

```
  [ default: 语句 n+1; ]
}
```

每一个 case 构成一个分支。根据 switch 中的整型/字符表达式值，依次判定 case 中的整型/字符常量表达式是否与其相等。

(1) 当有一个 case 的整型/字符常量与其相等，则其后的语句组被执行，若有 break 语句，则跳出该 switch 语句；若无 break 语句，则继续执行下一个 case 项，直到遇到 break 语句或右花括号“}”。当有一个以上 case 的整型/字符常量与其相等，则仅执行第一个 case 项的语句组。

(2) 当无任何 case 的整型/字符常量与其相等，且有 default 项，则跳过所有的 case 项，执行 default 项的语句组；若无 default 项，则不执行 switch 中的任何语句。

(3) default 项可以出现在 switch 语句中的任何位置。但无论 default 项出现在何处，程序首先会检查所有的 case 分支，如没有相符合项，则再寻找 default 项。

(4) switch 语句必须用一对花括号“{}”括起所有项内容。

JDK 7 以后的版本可以在 switch 中使用字符串，示例如下：

```
String s="test";
switch (s) {
case "test" :
    System.out.println("test");
    break ;
case "test1" :
    System.out.println("test1");
    break ;
default :
  System.out.println("break");
    break ;
  }
```

例 3-3 switch 语句的应用。

程序清单：ch03\SwitchDemo.java

```
public class SwitchDemo{
    public static void main(String[] args){
        int c=2;
        switch(c)
        {
            case 1: System.out.println("case 1 c="+c);
                    break;
            case 3: System.out.println("case 3 c="+c);
                    break;
            case 5: System.out.println("case 5 c="+c);
                    break;
            default: System.out.println("default c="+c);
        }
    }
}
```

该程序运行后输出：default c=2，表示不属于程序中列举的任一事件。

3.2　循 环 语 句

循环是指在给定条件为真时，反复执行某个程序段。它包括当型与直到型循环。当型循环有 for 与 while 两种形式，在进入循环体前必须先判定条件值是否为真（前判断循环），因而，其最少循环次数为 0。而直到型的形式为 do-while，第一次进入循环体不必理会循环条件是否为真（后判断循环），因而，其最少循环次数为 1。

3.2.1　for 循环语句

for 循环语句的基本形式为

```
for(表达式 1;表达式 2;表达式 3)
    循环体语句组
```

执行过程：先计算表达式 1，得到循环变量的初值，代入表达式 2 中，求出表达式 2 的逻辑值。当其逻辑值为真时，执行循环体语句组，否则，退出该循环。当执行完循环体语句组后，再计算表达式 3，再代入表达式 2……。因此，for 循环也可写成：

```
for(初值;条件;增量)
    循环体语句组
```

表达式 1 给循环变量赋初值，不管循环多少次，它只执行一次；表达式 2 是循环条件，当循环条件为真时，方可执行循环体的语句，否则将退出循环体；表达式 3 是每循环一次都必须改变的循环变量表达式，是使循环条件由真变为假的必要过程，也是防止死循环的重要环节。

三个表达式之间要用分号“;”分隔，for 中的两个分号不能省略，而三个表达式可以省略。

省略表达式 1：for(;表达式 2;表达式 3)，表达式 1 须在 for 之前执行。

省略表达式 1 和表达式 3：for(;表达式 2;)，表达式 1 须在 for 之前执行，表达式 3 须在循环体内执行。

省略表达式 1、表达式 2 和表达式 3：for(;;)，表达式 1 须在 for 之前执行，表达式 3 须在循环体内执行。当省略表达式 3 时，将失去控制进入循环体的判断条件，因而在循环体内还必须用 break 语句强制退出本次循环，否则将产生死循环。for 循环语句的框图如图 3-3 所示。

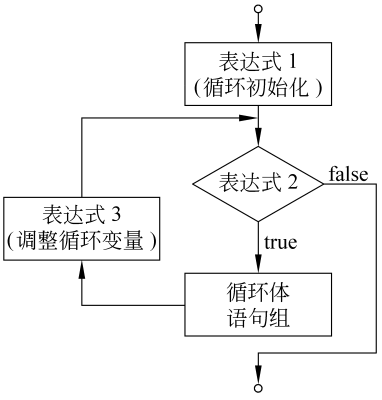

图 3-3　for 循环语句框图

循环体可以是单语句、复合语句，也可以是空语句（用“;”表示）。

例 3-4　用 for 循环语句对数组排序输出的例子。

程序清单：ch03\ForDemo.java

```
import java.util.*;
public class ForDemo{
  public static void main(String[ ] args)
    {
1       int[] a={22,2,222,-1,-110,0};
2       Arrays.sort(a);
3       for(int i=0;i<=5;i++) System.out.print(a[i]+",");
    }
}
```

程序运行结果：

```
-110, -1, 0, 2, 22, 222,
```

由于程序中要用到 Arrays 类，java.util 包中包含了这个类，所以第一行通过 import 语句引入 java.util 包，* 号表示引入包中所有的类。也可以将该语句改成：

```
import java.util.Arrays;
```

这样就只引入 java.util 包中 Arrays 一个类，现在程序中只需要这个类，所以可以改用这个语句。

语句 1 定义了一个 int[]型数组 a。

语句 2 调用 Arrays 类的 sort()方法，将数组元素按数字升序重新排列。sort()方法的原型为

```
public static void sort(int[] a)
```

由于这是一个 static 修饰的类方法，所以可通过类名 Arrays 调用。

语句 3 是 for 循环语句的应用。该语句将数组元素按照排列好的顺序，从第 1 个到第 6 个逐一打印出来。从中可以看到数组元素的顺序已经按数字升序重新排列。

有时可能想在一个称为嵌套循环的循环中编写一个循环，这在处理排列成多行和多列的数据时特别有用。可以使用外部循环在列之间移动，使用内部循环访问每一行。

下面的代码段演示了一个 for 循环位于另外一个 for 循环中。外部循环使用名为 m 的循环计数器，内部循环使用名为 n 的循环计数器。

```
for (int m=1; m<=2; m++){       //外部循环
   for (int n=1; n <=3; n++) {  //内部循环
     System.out.println("嵌套循环: m=" +m +", n=" +n);
   }
}
```

在此示例中，外部循环执行两次，它每次执行时，内部循环都会执行 3 次。也就是说，当变量 m 包含 1 时，变量 n 就会从 1 递增到 2，然后再递增到 3。对于内部循环的每一次迭代来说，都会显示这 2 个变量的内容。总共会调用 6 次 println()方法。

例 3-5 编写程序解决实际问题。已知公鸡 5 元 1 只，母鸡 3 元一只，小鸡 1 元 3 只，要求用 100 元刚好买 100 只鸡，问有多少种采购方案。

程序清单：ch03\BuyChicken.java

```
public class BuyChicken{
  public static void main(String[] args) {
         int I,J,K;
         System.out.println(" 公鸡数 I   母鸡数 J   小鸡数 K ");
         for (I=0;I<=20;I++) {       //I 为公鸡数
              for (J=0;J<=33;J++) {  //J 为母鸡数
                   K=100-I-J;        //K 为小鸡数
                   if (5*I+3*J+K/3.0==100)
                      System.out.println("      "+I+"      "+J+"      "+K);
                  }
              }
        }
}
```

程序运行结果：

```
公鸡数 I   母鸡数 J   小鸡数 K
   0         25        75
   4         18        78
   8         11        81
  12          4        84
```

例 3-6　编写 for 循环嵌套程序，要求输出如下图形：

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

程序清单：ch03\NestFor.java

```
public class NestFor {
 public static void main(String[] args) {
   int i, j;
   for (i=0; i<6; i++) {
     for(j=0; j<=i; j++) {
        System.out.print("* ");
      }
     System.out.println();
    }
  }
}
```

例 3-7　应用程序中 main()方法中的参数 args 能接收从键盘输入的字符串。求若

干个数的平均数，若干个数以命令行参数的形式从键盘输入。

程序清单：ch03\Average.java

```
public class Average{
    public static void main(String [] args){
        double n,sum=0.0;
        for (int i=0;i<args.length;i++){
              sum=sum+Double.valueOf(args[i]).doubleValue();
            }
        n=sum/args.length;
        System.out.println("平均数:"+n);
    }
}
```

编译通过后，输入如下执行命令：

```
java Average 23 45 67 (回车)
```

程序中的 args[0]、arg[1]、arg[2]分别得到字符串"23"、"45" 和"67"。在源程序中再将这些字符串转化为数值进行运算，得到的结果为

```
平均数：45
```

例 3-8 计算购房贷款利息。计算购房贷款月供的公式为 $R=PI(1+I)^N/((1+I)^N-1)$。R 为每月需付给银行的钱，简称月供。其中 P 为总贷款额，N 为按月计算的贷款期，I 为按月计算的贷款利息，简称月息。假设总贷款额为 30 万元，贷款期为 30 年，年息 4.59%，即月息 $I=0.0459/12$。用 for 循环程序计算月供。

程序清单：ch03\TestLoan.java

```
public class TestLoan{
  public static void main(String args[ ]){
      double dbp=300000;
      double dbi=0.0459/12;
      int nt=30*12;
      double dbpow=1;
      for(int i=0;i<nt;i++) dbpow=dbpow*(1+dbi);
      double dbr=dbp*dbi*dbpow/(dbpow-1);
      System.out.println("你每月需还贷款 R="+dbr+"元");
  }
}
```

程序运行结果：

```
你每月需还贷款 R=1536.1407316943316 元
```

3.2.2 for-each 循环语句

for-each 循环是 JDK 1.5 新增加的功能，提供一种更简洁的语句遍历集合(或数组)

的方法。它的一般形式为

```
for(循环变量类型　循环变量名：被遍历的对象名){循环体}
```

其中，“循环变量类型”指定了循环变量的取值类型，循环变量用来接收“集合”中的元素。每一次循环，会按顺序从“集合”中取出一个元素存储在循环变量中，如此重复，直到集合中的所有元素都已取出为止。由于循环变量从集合中接收值，所以“循环变量类型”必须与集合中存储的元素类型相同。

例 3-9　for-each 循环举例。

程序清单：ch03\ForEachTest.java

```
public class ForEachTest {
  public static void main(String[] args) {
     int arr[]={1,2,3,4,5,6,7,8};
     for (int a:arr)
      System.out.println(a);
  }
}
```

运行结果为逐行输出数字 1～8。

3.2.3　while 与 do-while 语句

1. while 语句

while 语句的基本形式为

```
while(表达式)
  {
      循环体语句组
  }
```

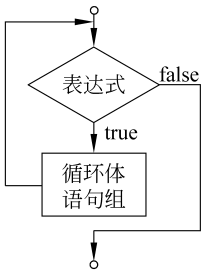

图 3-4　while 循环语句框图

while 循环语句的框图如图 3-4 所示。

while 循环语句功能和 for 循环语句相似，括号中的表达式是一个判断条件，类似于 for 语句中的判断条件，如果条件得到满足，循环才能继续下去。花括号中的语句构成循环体。

当 while 括号中的表达式值为 true 时，循环体语句被执行，否则，退出循环。

while 中的循环变量初值要在该 while 语句之前给出，而循环变量的改变必须在循环体或循环条件语句中完成，与 for(;表达式 2;)的功能相同。

while 中的循环条件可为永真值，如 while(true)。此时，要用 break 语句强制退出该循环，否则会产生死循环。如：

```
int i=5;
```

```
while(true) {
    if(i==0) break;
    i=i-1;
    System.out.println("sum="+i);
}
```

例 3-10 用 while 语句计算 1+1/2!+1/3!+1/4! +…的前 20 项的和。

程序清单：ch03\WhileSum.java

```
public class WhileSum
{ public static void main(String[] args)
    { double sum=0,a=1;int i=1;
      while(i<=20)
        { a=a * (1.0/i);
          sum=sum+a;
          i=i+1;
        }
      System.out.println("sum="+sum);
    }
}
```

程序运行结果：

```
sum=1.7182818284590455
```

2. do-while 语句

do-while 循环语句的基本形式为

```
do
   {
      循环体语句组
   }
while(表达式);
```

do-while 循环语句的框图如图 3-5 所示。

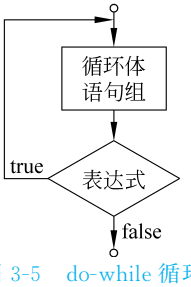

图 3-5 do-while 循环语句框图

do-while 和前面形式的区别是，判断语句“while(表达式);”处在循环体的后面，该语句必须以分号结尾。即使表达式不成立，至少也要执行一次循环体中的语句。

该循环结构是先执行循环体一次，再判断 while 括号中表达式的值，若为 true 值，就再次进入循环体，否则，退出循环。

循环变量初值要在该 do-while 语句之前给出，而循环变量的改变必须在循环体中完成，其执行完一次循环体后与 for(;表达式 2;)、while(表达式)循环结构功能相同。

while(表达式)后必须要有“;”，当循环体语句是复合语句时，须用一对花括号“{}”括起。

下面的代码段是用 do-while 循环语句计算 1～100 之和的例子。

```
int sum=0, j=1;
do {
    sun=sum +j;
    j++;
  }
while(j <=100);
```

3.3　break 与 continue 语句

break 用于跳出整个循环语句，已经在前面的程序中用过。continue 用于跳过本次循环中尚未执行的语句，但是仍然继续执行下一次循环中的语句。

在循环结构中一旦遇到 break 语句，不管循环条件如何，程序立即退出所在的循环体。

例 3-11　break 跳出整个循环举例。

程序清单：ch03\BreakLoop.java

```
public class BreakLoop{
    public static void main(String[] args){
    int i, j;
      for (i=0; i<6; i++) {
      for(j=0; j<100; j++) {
         if (j==5) break;
            System.out.print("*");
      }
        System.out.println();
      }
    }
}
```

程序输出如下：

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

continue 语句仅用于循环结构，在循环体中执行 continue 语句，就像执行循环结构的右花括号“}”一样，返回到对循环条件的判断，以确定是否进入下次循环体。不管 continue 与“}”之间还存在多少语句，一概不执行，因此，它又称短路语句。

break、continue 语句一般与 if 语句配合使用。

在一个循环中，如循环 50 次的循环语句中，如果在某次循环体的执行中执行了 break 语句，那么整个循环语句就结束。如果在某次循环体的执行中执行了 continue 语句，那么本次循环就结束，即不再执行本次循环中循环体中 continue 语句后面的语句，而转入进行下一次循环。

例 3-12 利用 break 和 continue 语句计算 10 以内的奇数之和并求 10 以内的所有素数。

程序清单：ch03\Con_Break2.java

```
class Con_Break2{
   public static void main(String[] args) {
      int sum=0,i,j;
      for( i=1;i<=10;i++) {     //计算 1+3+5+7+9
          if(i% 2==0)
             continue;
          sum=sum+i;
       }
      System.out.println("sum="+sum);
      for(j=2;j<=10;j++) {   //求 10 以内的素数
          for(i=2;i<=j/2;i++) {
                if(j% i==0)
                break;
          }
          if(i>j/2) {
                System.out.println(""+j+"是素数");
          }
       }
   }
}
```

程序运行结果如下：

```
sum=25
2 是素数
3 是素数
5 是素数
7 是素数
```

例 3-13 从键盘读取若干个整数，以“－1”结束，计算平均值。

这里采用 java.util.Scanner 类的 nextInt()方法读取键盘输入的整数。Scanner 对象使用如下方法读取指定类型的值：int nextInt()；float nextFloat()；double nextDouble()；long nextLong()；byte NextByte()；short nextShort()；boolean nextBoolean()。String nextLine() 读取一行的值并作为字符串返回。

程序清单：ch03\ReadKey.java

```
import java.util.Scanner;
public class ReadKey {
  public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        int score=0;
        int sum=0;
        int count=-1;
        System.out.println("输入整数(-1 结束):");
        while(score!=-1){
            count++;
            sum+=score;
            score=scanner.nextInt();
         }
         System.out.println("平均:"+(double)sum/count);
  }
}
```

程序运行结果：

```
输入整数(-1 结束):
45  67  98  34  56  -1
平均: 60.0
```

习　题　3

1. 结构化程序设计有哪三种流程？它们分别对应 Java 中的哪些语句？
2. 在一个循环中使用 break、continue 有什么不同？
3. 下面的代码将输出_____。

```
int i=1;
switch (i) {
 case 0: System.out.println("zero");
         break;
 case 1: System.out.println("one");
 case 2: System.out.println("two");
 default:System.out.println("default");
}
```

4. 下面的代码将输出_____。

```
class EqualsTest {
  public static void main(String[] args) {
       char a='\u0005';
       String s=a==0x0005L?"Equal":"Not Equal";
         System.out.println(s);
```

```
    }
}
```

5. 编写程序，求两个整数的最大公约数。

6. 编写程序，打印出如下九九乘法表。

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

7. 下面的代码将输出_____。

```
int i=1;
switch (i) {
 case 0: System.out.println("zero");
          break;
 case 1: System.out.println("one");
 case 2: System.out.println("two");
 default:System.out.println("default");
}
```

8. 下面的代码将输出_____。

```
class EqualsTest {
  public static void main(String[] args) {
        char a='\u0005';
        String s=a==0x0005L "Equal":"Not Equal";
         System.out.println(s);
  }
}
```

9. 编写程序，对 A[]={30,1,-9,70,25}数组由小到大排序。

10. 运行下面的代码将输出什么内容？

```
int i=1;
switch(i){
case 0:
    System.out.println("zero");
    break;
case 1:
```

```
        System.out.println("one");
    case 2:
        System.out.println("two");
    default:
        System.out.println("default");
```

11. 编写程序，求 2～1000 内的所有素数，并按每行 5 列的格式输出。
12. 编写程序，生成 100 个 1～6 的随机整数，统计 1～6 每个数字出现的概率。
13. 编写程序，求 1!＋2!＋3!＋…＋15!。
14. 编写程序，分别用 do-while 和 for 循环计算 1＋1/2!＋1/3!＋1/4!＋…前 15 项的和。
15. 编写一个程序，用选择法对数组 a[]＝{20，10，55，40，30，70，60，80，90，100}从大到小排序。
16. 编写程序，产生 30 个素数，按从小到大的顺序放入数组 prime[]中。
17. 一个数如果恰好等于它的因子之和，这个数就称为“完数”。编写程序求 1000 之内的所有完数。
18. 从键盘读取若干个数，以“－1”结束，按从小到大的顺序排序。

第4章 类与对象的基本概念

CHAPTER

4.1 类与对象的概念

类是实现 Java 面向对象程序设计的基础，是对基本数据类型的扩充。类封装了对象的行为和属性，它是具有相同特征的同类对象的抽象模型(template)，利用这个抽象模型可以构造具体的实例对象(instance)。

对象是 Java 程序中最核心、最基础的部分。对象在现实生活中是很普通的概念。所有的物体都可以被视为对象，大到宇宙，小到原子，都可以将其看作对象。我们时常与对象打交道，如钢笔、自行车、公交车等。而我们经常见到的卡车、公交车、小轿车等都会涉及以下几个重要的变量：可乘载的人数、运行速度、发动机的排量、耗油量、自重、轮子数目等。另外，还有加速、减速、刹车、转弯、播放音乐等几个重要的功能，这些功能称为它们具有的方法。一个对象具有本身的属性即特征，这些特征决定对象的状态，对象还可以通过自己的行为不断改变自己的状态。

类与对象的关系犹如图纸与零件的关系，先有图纸后有零件，图纸描述了零件的共同特征，零件是按图纸制造出来的。在程序中只能有类的一个定义，但该类可以有多个实例对象。在 Java 编程语言中使用 new 运算符实例化对象。

要学习 Java 编程就必须首先学会怎样去写类，即怎样用 Java 的语法去描述对象共有的属性和功能。属性通过变量来刻画，功能通过方法来体现。类把属性和对属性的操作封装成一个整体。Java 程序设计就是从类的设计开始的。

基于对象的编程更加符合人的思维模式，编写的程序更加健壮和强大。更重要的是，面向对象编程鼓励创造性的程序设计。

4.1.1 类的声明

类由关键词 class 定义。一个类的定义包括两个部分：类声明和类体。类体的内容由两部分构成，一部分是变量的定义，用来刻画属性；另一部

分是方法的定义，用来描述功能。

类的结构如图 4-1 所示。

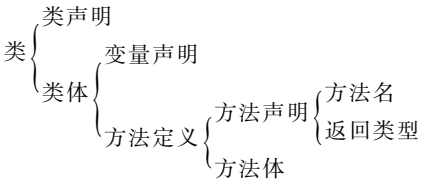

图 4-1　类的结构

类的定义的基本格式为

```
[public][abstract|final] class className [extends superclassName] [implements
interfaceNameList]
  {
    [public | protected | private ] [static] [final] [transient] [volatile] type
    variableName; //成员变量
     [public | protected | private ] [static] [final | abstract] [native]
     [synchronized]
     returnType methodName([paramList]) [throws exceptionList] {statements}
                  //成员方法
}
```

其中，修饰符 public、abstract、final 说明了类的属性，className 为类名，superclassName 为类的父类的名字，interfaceNameList 为类所实现的接口列表。

1. 类的成员变量

在类中声明的变量就是成员变量，作用域是整个类。类的成员变量分为类成员变量和实例成员变量。类的成员变量的声明方式如下：

```
[public | protected | private ] [static] [final] [transient] [volatile] type
variableName;
```

其中：

public：公有变量。

protected：保护变量。

private：私有变量。

static：静态变量(类成员变量)，相对于实例成员变量。

final：常量。

transient：暂时性变量，用于对象存档。

volatile：贡献变量，用于并发线程的共享。

2. 类的成员方法

成员方法描述对象所具有的功能或操作，反映对象的行为，是具有某种相对独立功能的程序模块。一个类或对象可以有多个成员方法，对象通过执行它的成员方法对传来的

消息作出响应，完成特定的功能。

成员方法一旦定义，便可在不同的场合中多次调用，故可增强程序结构的清晰度，提高编程效率。

成员方法的结构包括两部分内容：方法声明和方法体。方法声明包括方法名、返回类型和外部参数。其中参数的类型可以是简单数据类型，也可以是引用数据类型。

成员方法的定义方式如下：

```
[public | protected | private ] [static] [final | abstract ] [native]
[synchronized]
  returnType methodName ([paramList]) [throws exceptionList]
  {
      statements
  }
```

方法声明中的限定词的含义如下：

public：公共访问控制符。

protected：保护访问控制符。

private：私有访问控制符。

static：类方法，可通过类名直接调用。

final：方法不能被重写。

abstract：抽象成员方法，没有方法体。

native：本地成员方法修饰符，集成其他语言的代码。

synchronized：控制多个并发线程的访问。

下面是一个类名为“梯形”的类，类体内容的变量定义部分定义了 4 个 float 类型的变量“上底”“下底”“高”和 laderArea。方法定义部分定义了两个方法“计算面积()”和“修改高()”。

```
class 梯形{
    float 上底,下底,高,laderArea;
    float 计算面积() {
          laderArea=(上底+下底) * 高/2.0f;
          return laderArea;
    }
    void 修改高(float h) {
           高=h;
    }
}
```

类体变量定义部分所定义的变量被称为类的成员变量。在方法体中定义的变量和方法的参数被称为局部变量。

成员变量和局部变量的类型可以是 Java 中的任何一种数据类型，包括基本类型整型、浮点型、字符型和引用类型。

成员变量在整个类内都有效，局部变量只在定义它的方法内有效。例如：

```
class LocalVariable {
  int distance;
  int find() {
          int a=12;
          distance=a;        //合法，distance 在整个类内有效
          return distance;
  }
     void gety() {
          int y;
          y=a;               //非法，a 是局部变量，这里无法访问
     }
}
```

例 4-1 创建一个类，该类含有类的成员变量和成员方法，并对所创建的类进行测试。

程序清单：ch04\Text1.java

```
class Text1 {
      static int a;             //当被定义为 static 类型时，为类变量，可被对象或类调用
      int b;                    //实例对象变量，只能被对象调用
    public void display(int a,int b) {     //成员方法
        System.out.println("static int a="+a);
        System.out.println("  int b="+b);
    }
    public static void display(int b) {   //类方法，可通过类名直接调用
        System.out.println("static display: int b="+b);
    }
    public static void main(String[] args) {
          Text1 tt=new Text1();    //创建实例对象 tt
          t t.display(5,6);     //不可以用 Text1.display(5,6);因为对象变量或对象方法
                                //只能被对象 tt 调用
          Text1.display(0);    //当被定义为 static 类型时，为类方法，可被对象或类调用
          tt.display(23);
          tt.a=9;
          Text1.a=24;
          tt.b=3;
          tt.display(a,15);
    }
}
```

运行结果：

```
static int a=5
       int b=6
```