# 第3章　控制语句

Java 的流程控制有 3 种，即顺序、选择和循环，其中选择又称分支。熟悉 C/C++ 的读者可以发现，本章所介绍的控制语句的语法格式与 C/C++ 的类似。熟悉 C/C++ 的读者可以粗略地浏览 3.1 节和 3.2 节，直接学习 3.3 节带标号的 break 和 continue 语句。

本章的学习目标：

- ◆ 学会使用 Java 的分支语句
- ◆ 学会使用循环控制语句
- ◆ 学会使用 break 和 continue 语句
- ◆ 学会使用带标号的 break 和 continue 语句

## 3.1　分支语句

Java 中的分支语句有两个，一个是 if-else 语句；另一个是 switch-case 语句。

### 3.1.1　if 语句

if 语句的语法格式如下：

```
if(Expression)
    Statement
[else
    Statement]
```

**【注意】** if 语句的条件表达式 Expression 与 C/C++ 不同，Java 要求该条件表达式必须是布尔类型，否则无法通过编译。在 C/C++ 中条件表达式可以是一个数值，用非 0 代表真，0 代表假，而 Java 不允许这样。

例如，下面的写法在 C/C++ 中是合法的：

```
int  i=10;
    ⋮
if(i)
    ⋮
```

但在 Java 中却是错误的。必须将条件表达式转换为一个 boolean 值。例如，将 if(i)语句修改为 if(i !=0)。

if 语句中含有 else 时，要注意 else 与 if 的匹配关系，例如这样写代码：

```
if(x<100)
    if(y==60)
    {
```

```
        System.out.println(x+y);
        y--;
    }
else
    y++;
```

从程序缩排格式上,可以推断出这段程序含义,但由于 Java 规定:else 与最靠近自己的、上面的一个 if 语句匹配,因此 else 与第二个 if 匹配,若要与第一个 if 匹配,需要加括号改变匹配关系:

```
if(x<100)
{
    if(y==60)
    {
        System.out.println(x+y);
        y--;
    }
}else                                    //此时 else 匹配第一个 if
    y++;
```

因此,写程序时不但要注意缩排格式,而且还要注意 if-else 之间的匹配关系。

**【例 3-1】** 采用 Java Applet 小程序从文本框中获取数据,然后显示比较结果。

```
package chapter3;                        //第 3 章的例子程序,都放在 chapter3
包中
import java.awt.*;
import java.applet.*;

public class compareNumbers extends Applet
{
    Label  lab1, lab2;
    TextField  input1, input2;
    int  num1, num2;

    public void init()
    {
        lab1=new Label("输入第 1 个整数");   //产生第 1 个标签对象
        input1=new TextField(10);          //产生第 1 个文本框对象
        lab2=new Label("输入第 2 个整数");
        input2=new TextField(10);

        add(lab1);                         //将标签 lab1 对象放到网页上
        add(input1);                       //将文本框 input1 对象放到网页上
        add(lab2);
        add(input2);
    }

    public boolean action(Event e , Object o)
```

```
    {
        if(e.target==input1||e.target==input2)
        {
            num1=Integer.parseInt(input1.getText());        //获取文本框中的
数值
            num2=Integer.parseInt(input2.getText());
            if(num1<num2)
                showStatus(num1+"<"+num2);
            else  if(num1>num2)
                showStatus(num1+">"+num2);
            else  showStatus(num1+"=="+num2);
        }
        return  true;
    }
}
```


图 3-1　采用小程序输出比较结果

**【程序运行结果】** 假设输入 123 和 432 两个数,程序运行结果如图 3-1 所示。

**【程序解析】** 程序中的 init()和 action()均是系统规定的方法名,在此进行了覆盖(见第 4 章)。程序运行时,先执行 init()方法,若用户在文本框中按了 Enter 键,将调用 action()方法。主类中定义的 lab1 和 lab2 均是标签类型的引用,input1 和 input2 是文本框类型的引用。在 init()方法中,产生了 4 个对象,并将它们放置到屏幕上。

在 action()方法中,if(e. target==input1 ‖ e. target==input2)语句用于确定是否在 input1 或 input2 中按了 Enter 键。input1. getText()是获得文本框 input1 中的文本信息,然后采用系统提供的 Integer. parseInt()方法将这个文本转换为一个整数,并传递给变量 num1。

程序中的 if-else 语句用于比较 num1 和 num2 的大小,showStatus()方法是将其括号中的参数在状态栏输出。若 num1 的值是 123,num2 的值是 432,num1+"<"+num2 就是将 num1 的值后面跟上一个字符串"<",然后再跟上 num2 的值,共同构成了一个字符串,此处的+是一个 String 连接符,而不是一个算术运算符。

### 3.1.2　switch 语句

Java 的 switch 语句的语法结构如下:

```
switch(integral-expression)
{
    case  integral-value1:
                        statement;
                        break ;
        ...
    case  integral-valuen:
                        statement;
                        break ;
    default:
                        statement;
```

```
                    break ;
}
```

switch 语句中的 integral-expression 表达式必须是 int、byte、char 和 short 这几种类型之一。integral-value 表达式的值必须是对应类型的常量，并且常量不能重复。switch 把 integral-expression 的值和每个 integral-value 逐一比较，若找到相同者，就执行相应的 statement，若找不到相同者，就执行 default 中的 statement。每个 case 语句后面都有一个 break 语句，若缺少了该语句，便会继续执行其后的 case 语句。虽然 default 语句的 break 是多余的，不过考虑到编程风格问题，带上也无妨。下面看一个 switch 语句的举例。

**【例 3-2】** switch 语句应用举例。

```
package  chapter3;
public class justVowels                    //判断一个字母是否为元音
{
    public static void main(String args[])
    {
        char c;
        for(int i=0;i<100;i++)             //随机产生 100 个字母,进行判断
        {
            c=(char)(Math.random()*26+'a');
            System.out.print(c);
            switch(c){
                case  'a':
                case  'e':
                case  'i':
                case  'o':
                case  'u':
                      System.out.println("是元音");
                      break;
                case  'y':
                case  'w':
                      System.out.println("有时是元音");
                      break;
                default:
                      System.out.println("不是元音");
                      break;
            }
        }
    }
}
```

**【程序运行结果】** 由于是随机生成 100 个字母，故编译运行后，一个可能的结果是：

```
i 是元音
w 有时是元音
k 不是元音
```

```
w 有时是元音
v 不是元音
n 不是元音
b 不是元音
d 不是元音
k 不是元音
w 有时是元音
```

**【程序解析】** 程序中的变量 i 定义在 for 循环中,其作用域为该循环语句,当循环结束时,变量 i 就消失。Math.random()产生的是一个[0,1) 之间的随机值,将该值乘以 26 加上'a',然后取整,就是 26 个小写字母的 ASCII 码。需要说明的是,类型强制转换是临时的,并且不会进行四舍五入运算,而是截断,例如(char)(97.89),结果就是 97,即'a'的 ASCII,而不是 98。程序最后利用 switch-case 语句判断是否为元音字母。

**【例 3-3】** 采用 Java Applet 小程序实现将学生的百分制成绩转换为优秀、良好、中等、及格和不通过 5 个等级。

```
package  chapter3;
import java.awt.*;
import java.applet.*;

public class scoreConvert extends Applet
{
    Label  prompt;
    TextField  input;
    int  Excellent, Good, Middle, Pass,Failure;

    public void init()
    {
        prompt=new Label("输入成绩");
        input=new TextField(2);
        add(prompt);
        add(input);
    }

    public void paint(Graphics g)
    {
        g.drawString("各等级的人数:",25,40);
        g.drawString("优秀  :"+Excellent,25,55);
        g.drawString("良好  :"+Good,25,70);
        g.drawString("中等  :"+Middle,25,85);
        g.drawString("及格  :"+Pass,25,100);
        g.drawString("不通过:"+Failure,25,115);
    }
    public boolean action(Event e , Object o)
    {
            //从当前引发事件的对象 input 获取一个整数
```

```
        int  score=Integer.parseInt(input.getText());
        showStatus("");
        input.setText("");
        switch(score/10)
        {
            case  10:
            case  9:
                      Excellent++;
                      break;
            case  8:
                      Good++;
                      break;
            case  7:
                      Middle++;
                      break;
            case  6:
                      Pass++;
                      break;
            case  5:
            case  4:
            case  3:
            case  2:
            case  1:
            case  0:
                      Failure++;
                      break;
            default:
                     showStatus("输入有误,请重新输入!");     //显示错误信息
        }
        repaint();      //注意: 容易忘记的地方
        return  true;
    }
}
```

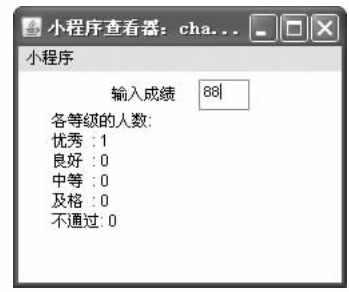


图 3-2　采用小程序对百分制成绩转换为五分制进行统计

**【程序运行结果】** 如图 3-2 所示。

**【程序解析】** 在主类 scoreConvert 中定义的 Excellent、Good、Middle、Pass 和 Failure 分别用于统计各个等级的人数，由于它们是类内的成员变量，并且是整型，因此初始值默认为 0。程序首先执行 init()方法，然后调用 paint()方法，所以程序的开始界面显示这 5 个变量的值是 0。

当用户在文本框中输入一个整型数据并且按了 Enter 键后，自动调用 action()方法。在 action()方法中，调用 input. getText()方法从文本框对象获取对应的字符串，Integer. parseInt()方法将该字符串转换为一个整数，因此 score 的值就是用户在文本框中所输入的值。showStatus("")是将状态栏显示的信息清空，input. setText("")是将文本框对象显示的内容清空。若用户在文本框中输入有误，例如输

入了 567,或者是－12 等,都会在状态栏显示错误信息。

**【注意】** repaint()方法是一个系统方法,它自动调用 paint()方法(后续章节将分析其具体调用),从而实现了对网页的刷新。若漏写了这一行,就看不到正确结果。此外,由于 action()方法是 boolean 类型,所以该方法最后要返回一个布尔值。

## 3.2 循环控制语句

Java 中的循环控制语句有 3 种,分别是 while、do-while 和 for 语句。循环体内的语句会反复执行,直到用于控制循环的布尔表达式的值变为 false 为止。

### 3.2.1 while 语句

while 语句的循环形式是:

```
while(Boolean-Expression)
    statements;
```

循环控制条件 Boolean-Expression 会在循环开始时判断一次,在循环体执行结束以后,再次判断 Boolean-Expression,以确定是否还执行循环体。

**【例 3-4】** 产生 10 个 60～100 之间的随机整数。

```
package  chapter3;
public class GenerateNumbers
{
    public static void main(String  args[])
    {
        int  i=0, Int_val=0;
        while(i<10)
        {
            Int_val=(int)(Math.random() * (100-60)+60);
            System.out. printf("%5d",Int_val);
            i++;
        }
    }
}
```

**【程序运行结果】** 由于采用了随机数,故一次可能的运行结果如下:

```
66   99   95   68   78   75   87   69   66   91
```

**【程序解析】** 程序通过循环控制变量 i,保证循环执行 10 次,每次产生一个整数,输出时的域宽为 5。

### 3.2.2 do-while 语句

do-while 语句的语法格式如下:

```
do{
    statement;
}while(Boolean-Expression);
```

**【注意】** do-while 语句和 while 语句类似，它们的区别是 while 语句是先判断后执行，若 Boolean-Expression 为假，整个循环体一次也不执行；而 do-while 语句是先执行后判断，所以循环体至少要执行一次。

**【例 3-5】** 直到产生一个大于 0.9 的随机数为止。

```
package  chapter3;
public class GenerateDoubleNumbers
{
    public static void main(String  args[])
    {
        double  d;
        do{
            d=Math.random();
            System.out. println(d);
        }while(d<0.9);
    }
}
```

**【程序运行结果】** 某次运行结果如下：

```
0.418710        0.524779        0.038600        0.936135
```

**【程序解析】** 由于要求产生一个大于 0.9 的随机数，因此程序先产生一个随机数，而后判断是否满足条件，若满足条件就停止循环，否则继续产生下一个随机数。

### 3.2.3　for 语句

for 语句的语法格式如下：

```
for(ForInitopt; Boolean-Expression; ForUpdateopt)
    Statement;
```

在第一次循环前，先执行初始化语句 ForInitopt；再进行条件测试，若 Boolean-Expression 为真，就执行循环体；然后执行 ForUpdateopt 部分，转入条件测试；若 Boolean-Expression 为假，整个循环结束。

**【注意】** for 语句中的 ForInitopt、Boolean-Expression 和 ForUpdateopt 都可以为空。

**【例 3-6】** 编写一个 applet，输出一个倒三角形图案。

```
package  chapter3;
import java.awt.*;
import java.applet.Applet;
public class printGraphics extends Applet
{
```

```
    public void paint(Graphics g)
    {
        int xpos,ypos=80;
        for(int row=6;row>=1;row--)
        {
            xpos=25;
            ypos+=10;
            for(int column=1;column<=row;column++)
            {
                g.drawString("*",xpos,ypos);
                xpos+=7;
            }
        }
    }
}
```


图 3-3 采用小程序输出一个倒三角形

**【程序运行结果】** 如图 3-3 所示。

**【程序解析】** 程序中的 g. drawString(" * ", xpos, ypos)是在指定的 xpos 和 ypos 位置输出一个 * 。ypos＋＝10 是对输出 * 的 Y 轴坐标增量,即行距设定为 10 个像素;xpos＋＝7 是对 X 轴坐标增量,即字符 * 之间的距离设定为 7 个像素。

**【思考】** 如果将程序中的“ypos＋＝10”改为“ypos－－10”,你还知道结果吗?

## 3.3 break 语句和 continue 语句

将 Java 中的 break 语句和 continue 语句分为两类讨论,一类是不带标号的语句;另一类是带标号的语句。

### 3.3.1 不带标号的 break 语句和 continue 语句

采用 break 语句可以控制循环的流程,break 语句可以跳出包含它的最内层的循环,不再执行剩余的语句,continue 语句会停止执行当前的循环,回到循环处,开始执行下一轮的循环。这些特性和 C/C++ 中的 break 语句和 continue 语句的功能一样。

**【例 3-7】** 基本的 break 语句和 continue 语句。

```
package  chapter3;
public  class breakANDcontinue
{
     public static void main(String args[])
     {
         for(int i=1;i<20;i++)
         {
              if(i%9==0)
                   break;
```

```
            if(i%3==1)
                continue;
            System.out. printf("%5d",i);
        }
    }
}
```

**【程序运行结果】** 输出如下：

```
2       3       5       6       8
```

**【程序解析】** 当 i 的值等于 9 时，break 语句就终止了当前的循环，所以输出的值是到 8 为止。在 1～8 之间，若满足条件 i%3＝＝1，将转入下一次循环，其后面的语句也不再执行，所以 i 的值等于 1、4、7 时，均未输出。

### 3.3.2　带标号的 break 语句和 continue 语句

Java 不但保留了与 C/C++ 相似的 break 语句和 continue 语句特性，而且还引入了带标号的 break 语句和 continue 语句。当在循环体中执行带标号的 break 语句时，可以立即退出任意多个嵌套循环。从程序设计语言原理的角度讲，这种带标号的语句是 goto 语句的一种变形，它使程序具有一定的灵活性。

带标号的 break 语句的语法格式如下：

```
break  Identifier;
```

带标号的 continue 语句与 break 语句类似：

```
continue  Identifier;
```

其中 Identifier 是一个标识符。这种带标号的语句类似于 C/C++ 中的 goto 语句，尽管 goto 是 Java 的保留字，但 Java 并未使用它。

**【例 3-8】** 带标号的 break 语句和 continue 语句。

```
package  chapter3;
public  class hello
{
    public static void main(String args[])
    {
        int i,j=0;
outer:
        for(i=0;i<3;i++)
            for(j=0;j<5;j++)
            {
                System.out.println(i+"  "+j);
                if(j==1)
                    break  outer;                    //注意该语句
            }
        System.out.println("最终值: "+i+"  "+j);   //注意该语句的位置
```

```
    }
}
```

**【程序运行结果】**

```
0  0
0  1
最终值: 0  1
```

**【程序解析】** 程序中 outer 标号的位置在外循环开始处,当变量 i 和 j 的值都是 0 时,输出结果的第一行,即 0 0,内循环变量 j 增 1 后,再次进入循环体,输出了 0 1,由于 j 的值为 1,执行 break outer,结束内外层的所有循环。所以循环结束后,i 和 j 的值分别为 0 和 1。

如果将上述程序中的 break outer 修改为 continue outer,输出结果为:

```
0  0
0  1
1  0
1  1
2  0
2  1
最终值: 3  1
```

当 i 的值等于 0,j 的值等于 1 时,执行 continue outer 语句,流程就转到 outer 所在的外循环,而不是像 break outer 语句那样结束整个循环。变量 i 的值加 1 以后,重新进入内循环,j 的初值再次被赋为 0,所以程序输出 1 0。当变量 i 的值变为 2,j 的值变为 1 时,流程转到外循环,变量 i 的值加 1,变成 3,由于不满足条件 i<3,所以内外循环全部结束,变量 i 和 j 的最终值就是 3 和 1。

这两个带标号语句的**共同点**是:

(1) 都必须用在循环中,用于流程控制。

(2) 执行这两个语句时,若后面还有其他语句,将不再继续执行。

这两个带标号语句除了功能不同,在用法上也有**区别**:

(1) continue 语句的标号必须位于封闭的循环语句的前面。如下用法就是错误的:

```
outer: {          //对于带标号的 continue 语句,这种用法是错误的
       i=0;       //这个赋值语句和下面的循环共同构成了一个复合语句
       for(i=0;i<3;i++)
           for(j=0;j<3;j++)
           {
               System.out.println(i+"  "+j);
               if(j==1)
           }      continue  outer;
}
```

尽管标号 outer 位于封闭的语句之前,但该语句仅仅是一个复合语句,而不是一个循环语句,故这种用法是错误的。

(2) break 语句的标号也必须位于封闭语句的前面,但不一定是循环语句。

```
outer:{                    //对于 continue 语句是错误的,但对 break 是正确的
       i=0;
       for(i=0;i<3;i++)
           for(j=0;j<3;j++)
           {
               System.out.println(i+"  "+j);
               if(j==1)
           }      break  outer;
}
```

上述写法对 continue outer 语句是错误的,对 break outer 语句是正确的,其原因也比较明显,因为 break outer 语句是结束整个封闭的语句。

## 本章小结

本章讨论了 Java 语言中 3 种基本结构以及相应的控制语句。其中带标号的 break 和 continue 语句与 C/C++ 的对应语句不同,Java 的这两个语句具有中断能力,读者要注意这一特点。

通过前 3 章的学习,已经可以编写功能比较完整的程序了,但 Java 是一个面向对象的语言,只有掌握面向对象的程序设计方法和 Java 语言的面向对象特性,才算真正掌握了 Java 语言。我们在第 4 章将讲述面向对象的程序设计核心——类。

## 思考和练习

(1) 编程:编写一个 applet,要求输入一个任意长度的整数(long 类型变量所允许的范围内),将这个数分成独立的数字,并分开显示。例如输入 32439 则显示 3 2 4 3 9。

(2) 编程:计算 0～10 之间各个整数的平方值和立方值,并以如下格式显示:

```
整数     平方          立方
0         0             0
1         1             1
2         4             8
3         9             27
……
10       100           1000
```

(3) 采用循环语句打印如下图形。

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

(4) 编程:编写一个 applet,读取一个矩形的边长,然后输出一个空心矩形。例如读入边长 5,应当输出:

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

(5) 编程:编写一个 applet,输入一个数,判断输入的这个数是否为回文数。所谓回文数就是从左向右看和从右向左看都一样。例如,121、13431 都是回文数,而 12345 不是回文数。

(6) 编写一个 applet,采用下列公式计算 $e^x$ 的值:$e^x = 1 + x^1/1! + x^2/2! + \cdots + x^n/n!$。从键盘输入 x 和 n,编程计算 $e^x$ 的值。

(7) 编程:产生 20 个 int 类型的随机数,针对每个数使用 if-then-else 判断它是大于、小于或等于下一个数(注意:最后一个数不参与比较)。

# 第 4 章　Java 的类

我们在前面 3 章已经编写了多个功能完整的程序，但这些程序实际上都是利用基本数据类型和结构化程序设计方法所编写的，并没有体现出 Java 语言的面向对象特点和优越性。前面已经讲过，Java 语言除了几种基本的数据类型（整型、字符型、浮点型和布尔型）以外，其他全部是对象类型。

本章的学习目标：

- ◆ 掌握类的定义和使用
- ◆ 掌握方法的定义和使用
- ◆ 理解实例变量和局部变量
- ◆ 掌握构造函数的定义和使用
- ◆ 掌握方法的覆盖和重载
- ◆ 掌握关键字 this 的用法
- ◆ 理解继承的概念和应用
- ◆ 理解组合与继承
- ◆ 理解抽象方法和抽象类
- ◆ 掌握对象类型转换
- ◆ 掌握访问权限限制符：public、private、protected

## 4.1　类与对象

Java 是在 C++ 的基础上发展起来的，与 C++ 相比，它是更为“纯粹”的面向对象编程（Object Oriented Programming）语言。在 Java 程序中，除基本类型的变量以外都是对象，连 Java 程序本身也不例外。

类（class）是面向对象程序设计的基础，OO 始终是围绕对象的封装性、继承性和多态性展开讨论的。学习 Java 编程，首先要掌握关于对象的基本概念。

### 4.1.1　类与对象的区别

假设你的名字叫张三，显然你是一个人，而我也是一个人，那么你和我都是人类中的实例（instance），也称为对象（object）。类描述了一组对象的公共特性，人类就具有姓名、身高、体重、性别、需要学习、需要吃饭等属性。

对象是类中的一个特例，它具有自己确定的属性。例如，张三是一个对象，是人类中的一个实例，他的属性为：姓名是张三，具有 1.8m 的身高，70kg 的体重，男，每天要看 8 个小时的书，每天还要吃 3 顿饭，这些特性就是张三这个对象的确定属性。要把对象的确定属性和类的属性区别开。

**【注意】** 类与对象不同。类是一个抽象的概念，对象是一个具体的概念。类是在一组

对象的基础上,通过抽象和概括所获得的一个概念。例如,“人类”就是一个抽象的概念,而“张三”就是一个具体的对象。

对象是由数据和方法紧密结合的一个封装体,它具有信息隐藏的能力。例如,若一个人不告诉你他的姓名和身高,你(另一个对象)就不可能知道他的这些属性。此外,对象可以通过方法(函数)与其他对象进行通信,但并不知道这些方法的具体实现细节。例如,张三和李四是人类中的两个对象,他们可以通过方法(口和耳朵等)进行交流,但并不清楚方法(也不需要清楚)是如何进行交流的。

### 4.1.2 Java 和 C 编程思想的区别

结构化程序设计和面向对象的程序设计是两种截然不同的编程思想。Pascal 语言的设计者 Niklaus Wirth 曾经写过一本《算法+数据结构=程序》的书,它是结构化编程的名著。从书名可以看出结构化编程的思想:对待一个要解决的问题,首先是如何设计算法,然后建立什么样的数据结构才能使操作更为简便。面向对象编程与此相反,即先建立数据结构,然后再考虑如何操作这些数据。

C 和 Java 分别是结构化编程语言和面向对象的编程语言,其编程思想截然不同。C 是以函数为编程单元,C 程序员把注意力集中在编写函数上。而 Java 是面向对象的编程语言:以类为编程单元,程序员的精力集中在对类的设计上。

进行 Java 编程时,应当尽可能地继承系统提供的类,不要一切从头开始。因为这些类都是经过测试的高质量的类。采用 OO 编程使得系统易于维护。对象将实例变量(instance variable)和对数据的操作(即方法)约束在一起,类似于一个独立的程序,易于扩充,易于维护,代码可复用。

### 4.1.3 如何定义类

类是对象的模板,它定义了对象的结构和操作数据的功能接口,即方法。Java 类的定义格式如下:

```
class className extends superClassName
{
    type  instance-variable1;
    …
    type  instance-variableN;
    return-type  methodName1(parameter-list)
    {
        method-body;
    }
    return-type  methodNameN(parameter-list)
    {
        method-body;
    }
}
```

这里的 className 和 superClassName 都是合法的用户标识符。关键字 extends 是继

承的意思，表示 className 是 superClassName 的子类。Java 提供了一个系统类 Object，它是整个类层次结构中的根。

**【注意】** 如果要定义 Object 的子类，可以不指明 extends Object，因为在默认情况下一个类就是 Object 的子类。

类内定义的变量称为实例变量。例如：

```
class  point                          //没有显示指明父类,point 类是 Object 类的子类
{
    int  x, y;                        //x 和 y 是实例变量
    void  init(int a, int b)          //init 是方法
    {
        x=a;   y=b;
    }
}
```

由于没有指明 point 类的父类，则其父类就是 Object 类。

熟悉 C++ 的读者知道，C++ 可以将类的定义与类的实现分开存放。但 Java 是将类的定义和类的实现放在一起的。Java 这样做保证了一个类必须完整地定义在一个源文件中，从而使系统维护更为便利，不需要到一个文件中找方法 A，再到第二个文件中找方法 B，从而提高了程序的执行效率。

### 4.1.4　对象和引用

对象是客观存在的一个变量，对象的引用就是对象的名字，例如：

```
point  p1, p2;
```

定义了对象变量 p1 和 p2。这两个变量都可以指代 point 类的对象，但现在它们都不能代表对象，认识到这一点很重要。此时，还不能使用方法。

```
p1.init(10,20);       //现在还不能进行方法调用
```

p1 和 p2 现在称为 point 类的对象引用，它们的值都是 null，代表没有值（注意：null 不是 0）。例如，假设一个小孩还未出世，就已经起好了名字“张三”。此时还不能说张三是一个人（对象），因为它还不存在，此时的“张三”仅仅是一个引用（可以代表一个人）。

创建对象要采用 new 运算符：

```
p1=new  point();      //创建了 point 类的一个实例
```

现在就可以对 p1 调用 init()方法。

一个对象可以有多个别名，就像一个人可以有多个名字一样。例如：

```
p2=p1;
```

就是说 p1 所代表的那个对象，现在具有两个名字，一个名字是 p1；另一个名字是 p2。通过引用 p2 修改对象，同样影响 p1 所指的对象。假设你的大名叫张三，小名叫李四，也就是说，你这个对象具有两个名字。

可以将一个变量设置为 null,表明该变量不代表任何对象。

```
p1=null;
```

p1 前面表示的那个对象已经变成了垃圾,由系统提供的垃圾回收器在方便的时候回收,从而释放该对象占用的内存空间。关于垃圾回收见 4.10 节。

**【注意】** 每个对象都有自己的实例变量,改变一个对象的实例变量并不影响另外一个对象的实例变量,请参考例 4-1。

在一定的范围内,调用实例变量的语法格式如下:

```
objectReference . variableName
```

例如,创建两个 point 类对象,输出其实例变量:

**【例 4-1】** 理解对象的实例变量。

```
class  point
{
    int  x , y;
    void  init(int a, int b)        //init 是方法
    {
        x=a;   y=b;
    }
}
public  class twoPoint
{
    public static void main(String  []args)
    {
        point  p1=new point();
        point  p2=new point();      //生成了 p1 和 p2 两个对象
        p1.x=10;     p1.y=20;       //对象 p1 的 x 和 y
        p2.x=30;     p2.y=40;       //对象 p2 的 x 和 y
        System.out.println("x="+p1.x+"  y="+p1.y);
        System.out.println("x="+p2.x+"  y="+p2.y);
    }
}
```

**【程序运行结果】**

```
x=10  y=20
x=30  y=40
```

**【程序解析】** p1 和 p2 两个对象分别具有自己的实例变量 x 和 y,它们相分离,故输出结果不同。

## 4.2 方法

Java 的方法相当于 C 的函数,是一个功能模块。方法与函数完全不同:函数是实现特定功能的模块,是 C/C++ 的产物,而方法必须属于一个类,它不能单独存在,严格地说,Java

中没有函数，只有方法。有些书中将 Java 的方法称为函数，那纯粹是为了保留 C/C++ 的习惯称谓。

方法和实例变量具有相同的层次，都必须定义在类内，方法是类的功能接口，对象之间进行信息交流就是通过方法实现的。

在定义方法时，有一个原则：每一个方法应当只执行单一的、定义良好的任务，并且方法名应当有效地表达该任务，也就是说，任何一个方法名应当能表达一个主题，如果一个名字不能有效地表示该方法，就说明该方法要完成的功能模块太多，需要将其分解为多个子模块，这样做可以提高软件的重用性。

方法的定义与 C/C++ 中函数的定义类似，语法格式如下：

```
return-type  methodName(parameter-list)
{
    method-body;
}
```

return-type 是方法的返回值类型，如果没有返回值，就要定义为 void 类型。methodName 是方法名，它是一个合法的用户自定义标识符，parameter-list 是形参表，如果方法不带参数，可以略去参数，但圆括号要保留。

**【注意】** 将方法的返回值类型、方法名和参数表共同称为方法的特征(signature)，有些文献称为方法的签名。

**【注意】** 在 C/C++ 中，不带参数的函数可以定义成 methodName (**void**)，而在 Java 中这样写是错误的。

对象通过点运算符调用方法，调用方法的语法格式如下：

```
objectReference . methodName(parameter lists);
```

现在通过 init()方法对 point 类中的实例变量 x 和 y 进行初始化：

```
point  p1=new point(),p2=new point();
p1.init(10,20);
```

对象通过调用 init()方法完成对实例变量的初始化。

**【注意】** 执行 p.init(10, 20)，虽然调用了 init()方法，但实际上仅仅对 p1 对象的实例变量 x 和 y 进行赋值，对 p2 对象的 x 和 y 没有影响。进一步讲，哪个对象调用 init()方法，那么该方法就影响哪个对象。上例是 p1 对象调用了 init()方法，所以仅仅影响到 p1 对象，对 p2 对象无影响。

## 4.3　实例变量和局部变量

Java 中的变量分为两种，一种是在类内定义的实例变量，也称为成员变量；另一种是方法中定义的局部变量。例如，前面 point 类内的变量 x 和 y 就属于实例变量。

局部变量在 Java 中表现为两个方面，一种情况是指在方法体内定义的变量；另一种情况是指方法中的形参。

在同一个作用域内,Java 不允许定义两个同名的局部变量。例如,这样写就是错误的:

```
class point
{
    int x=1,y=1;                        //实例变量 x 和 y
    void  draw()
    {
        int  y=0;                       //局部变量
        for(int y=10;y<100;y++)         //错误: 循环体内的 y 和局部变量同名
            ;
    }
}
```

在循环体内定义的变量 y 和在方法内定义的局部变量 y 同名,它们位于同一个作用域内,这是错误的。

局部变量可以和实例变量同名,也就是说,局部变量可以屏蔽实例变量。例如,将 point 类修改如下:

```
class  point
{
    int  x , y;                         //实例变量
    void  init(int x, int y)            //形参和实例变量同名
    {
        x=x;   y=y;                     //此处错误,注意赋值形式
    }
}
```

此时 init()方法体内的赋值语句,就不能将参数 x 的值赋给实例变量 x。在这种局部变量和实例变量同名的情况下,局部变量就屏蔽了实例变量。解决这个问题有两种方法,一种方法是按照前面定义 init 的格式,不要将参数与实例变量同名;另一种方法可参考 4.6 节的 this 用法。实例变量与局部变量容易混淆。

**【例 4-2】** 对象的实例变量和方法中的局部变量同名问题。

```
package chapter4;
class loc
{
    int x=1;                            //实例变量
    void  printLocVar()
    {
        int x=25;                       //局部变量
        System.out.println("x in printLocVar is: "+x);
        ++x;
        System.out.println("x in printLocVar is: "+x);
    }

    void printInstanceVar()
```

```
    {
        System.out.println("instance variable x is : "+x);
        x*=10;                                  //实例变量 x
        System.out.println(" instance variable x is : "+x);
    }
}

public class testInstanceVar                    //程序的主类
{
    public static void main(String args[])
    {
        loc  obj=new loc();
        int x=5;                                //局部变量 x
        System.out.println(" x in main is : "+x);
        obj.printLocVar();
        obj.printInstanceVar();
        System.out.println(" x in main is : "+x);
    }
}
```

**【程序运行结果】**

```
x in main is :5
x in printLocVar is :25
x in printLocVar is :26
instance variable x is :1
instance variable x is :10
x in main is :5
```

**【程序解析】** loc 类内值为 1 的 x 是一实例变量，printLocVar()方法内值为 25 的 x 是该方法的局部变量，由于该变量与实例变量同名，因此导致在该方法内无法访问实例变量。由于 printInstanceVar()方法内没有定义 x，因此其使用的 x 是实例变量。main()方法内定义的变量 x，也是一个局部变量。

**【注意】** 实例变量属于对象，它描述了对象的属性，随着对象的存在而存在，而局部变量是随着方法的调用而存在的，一旦对方法调用结束，局部变量也就消亡了。

## 4.4　构造函数

如果每创建一个对象，都要对实例变量进行初始化，就显得很不方便。Java 提供了一个特殊的方法，叫构造函数(在有些书中称之为构造方法)，其功能是在创建对象时初始化对象的实例变量。构造函数与类具有相同的名字，除了构造函数，在类中不能有其他方法与类同名。一旦定义了构造函数，在对象定义后和 new 操作完成之前，系统将自动调用构造函数。构造函数具有如下两个特点：

(1) **构造函数没有返回值，也不能在其前面加上 void 修饰符**。例如，修改 point 类，利