第 3 章　表达式和流程控制语句

3.1　Java 中常用的运算符有哪些？它们的含义分别是什么？

解：Java 运算符按功能可分为：算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符、条件运算符；除此之外，还有几个特殊用途运算符，如数组下标运算符等。

常用的运算符及其意义如表 3-1 所示。

表 3-1　运算符及其意义

类　　型	运　算　符	意　　义
算术运算符	＋	加法或字符串的连接
	－	减法
	*	乘法
	/	除法，如果参加运算的两个操作数都是整数，则为整除运算，否则为浮点数除法
	%	整数求余运算
	＋＋	加 1 运算
	－－	减 1 运算
	－	求相反数
关系运算符	＝＝	等于
	!＝	不等于
	＞	大于
	＜	小于
	＞＝	大于等于
	＜＝	小于等于
逻辑运算与位运算	&.&.	逻辑与
	||	逻辑或
	!	取反
	&.	按位与
	|	按位或
	^	按位异或
	～	按位取反

续表

类　型	运　算　符	意　义
逻辑运算与位运算	>>	右移
	>>>	右移,并用 0 填充高位
	<<	左移
赋值运算符	+=	加法赋值
	-=	减法赋值
	*=	乘法赋值
	/=	除法赋值
	%=	取余赋值
	&=	按位与赋值
	|=	按位或赋值
	^=	按位异或赋值
	<<=	左移赋值
	>>=	右移赋值
	>>>=	右移赋值
其他运算符	[]	数组下标
	.()	方法调用
	?:	三目条件运算
	instanceof	对象运算

3.2　Java 中操作符优先级是如何定义的?

解:Java 中,各操作符的运算优先顺序如表 3-2 所示。

表 3-2　操作符的运算优先顺序

优　先　级	运　算　符	运　　算	结　合　律
1	[]	数组下标	自左至右
	.	对象成员引用	
	(参数)	参数计算和方法调用	
	++	后缀加	
	--	后缀减	
2	++	前缀加	自右至左
	--	前缀减	
	+	一元加	

续表

优 先 级	运 算 符	运 算	结 合 律
2	－	一元减	自右至左
	～	位运算非	
	!	逻辑非	
3	new	对象实例	自右至左
	(类型)	转换	
4	*	乘法	自左至右
	/	除法	
	%	取余	
5	＋	加法	自左至右
	＋	字符串连接	
	－	减法	
6	＜＜	左移	自左至右
	＞＞	用符号位填充的右移	
	＞＞＞	用 0 填充的右移	
7	＜	小于	自左至右
	＜＝	小于等于	
	＞	大于	
	＞＝	大于等于	
	instanceof	类型比较	
8	＝＝	相等	自左至右
	!＝	不等于	
9	&	位运算与	自左至右
	&&	布尔与	
10	^	位运算异或	自左至右
	^	布尔异或	
11	|	位或	自左至右
	||	布尔或	
12	&&	逻辑与	自左至右
13	||	逻辑或	自左至右
14	?：	条件运算符	自右至左

续表

优 先 级	运 算 符	运 算	结 合 律
15	=	赋值	自右至左
	+=	加法赋值	
	+=	字符串连接赋值	
	-=	减法赋值	
	*=	乘法赋值	
	/=	除法赋值	
	%=	取余赋值	
	<<=	左移赋值	
	>>=	右移(符号位)赋值	
	>>>=	右移(0)赋值	
	&=	位与赋值	
	&=	布尔与赋值	
	^=	位异或赋值	
	^=	布尔异或赋值	
	|=	位或赋值	
	|=	布尔或赋值	

3.3 >>>与>>有什么区别？试分析下列程序段的执行结果：

```
int b1=1;
int b2=1;

b1 <<=31;
b2 <<=31;

b1 >>=31;
b1 >>=1;

b2 >>>=31;
b2 >>>=1;
```

解：>>>与>>都是右移运算符，它们的不同之处在于使用不同位填充左侧的空白。>>运算使用符号位填充左侧的空位，而>>>使用零填充空位。也就是说，>>运算保持操作数的符号不变，而>>>运算则可能改变原数的符号。

分析上面的程序段。初始时，b1 和 b2 都是 int 型的变量，占 32 位，初值均为 1，两个变量分别向左移动 31 位，则两个值变为：

```
10000000 00000000 00000000 00000000
```

在计算机内部，这个值是：－2147483648。

下一步，b1 再向右移动 31 位，这里使用的是＞＞运算符。b1 是一个负数，其最高位为 1，右移时使用 1 填充左侧的空位。右移 31 位后 b1 的值为：

```
11111111 11111111 11111111 11111111
```

在计算机内部，这个值是：－1。b1 继续向右移动 1 位，此时值不变，仍为－1。实际上，此后使用＞＞运算符将 b1 向右移动任何位，它的值都不会再变了。

使用＞＞＞运算符则有所不同。＞＞＞使用零填充左侧的空位，所以将 b2 向右移动 31 位后，它的值为：

```
00000000 00000000 00000000 00000001
```

即 b2 的值为 1。再向右移动 1 位，则它的值为：

```
00000000 00000000 00000000 00000000
```

即 b2 的值为 0。

测试这些语句的程序如下所示：

```
import java.util.*;

public class Test
{   public static void main(String[] args)
    {   int b1=1;                                   //b1 赋初值
        int b2=1;                                   //b2 赋初值
        System.out.println("b1="+b1);
        System.out.println("b2="+b2);

        b1 <<=31;                                   //b1 左移 31 位
        b2 <<=31;                                   //b2 左移 31 位
        System.out.println("b1="+b1);
        System.out.println("b2="+b2);

        b1 >>=31;                                   //b1 右移 31 位
        System.out.println("b1="+b1);
        b1 >>=1;                                    //b1 再右移 1 位
        System.out.println("b1="+b1);

        b2 >>>=31;                                  //b2 右移 31 位
        System.out.println("b2="+b2);
        b2 >>>=1;                                   //b2 再右移 1 位
        System.out.println("b2="+b2);
    }
}
```

相应的执行结果如图 3-1 所示。

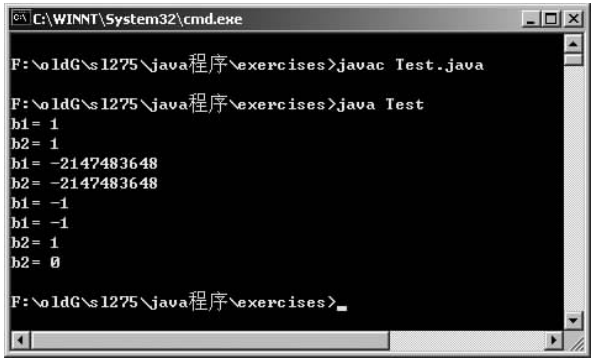

图 3-1　右移操作的执行结果

【拓展思考】

给出下面的说明，下列每个赋值语句会得到什么结果？

```
int iResult, num1=25, num2=40, num3=17, num4=5;
double fResult, val1=17.0, val2=12.78;
a. iResult=num1/num4;
b. fResult=num1/num4;
c. iResult=num3/num4;
d. fResult=num3/num4;
e. fResult=val1/num4;
f. fResult=val1/val2;
g. iResult=num1/num2;
h. fResult=(double) num1/num2;
i. fResult=num1/(double) num2;
j. fResult=(double) (num1/num2);
k. iResult=(int) (val1/num4);
l. fResult=(int) (val1/num4);
m. fResult=(int) ((double) num1/num2);
n. iResult=num3%num4;
o. iResult=num2%num3;
p. iResult=num3%num2;
q. iResult=num2%num4;
```

3.4　设 n 为自然数，

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

称为 n 的阶乘，并且规定 0!＝1。试编制程序计算 2!，4!，6!，8!和 10!，并将结果输出到屏幕上。

解：阶乘函数在数学上的定义为：

$$n! = \begin{cases} 1 & (n=0) \\ n(n-1)! & (n>0) \end{cases}$$

这是一个递归定义，因为阶乘本身又出现在阶乘的定义中。对于所有的递归定义，一定要有一个递归结束的出口，这既是定义的最基本情况，也是程序执行递归结束的地方。本定义中的第一行即是递归出口。

当一个函数使用递归定义的时候，往往直接使用递归方法实现它。

阶乘的递归实现如下所示：

```
import java.util.*;

public class Factorial
{   public static void main(String[] args)
    {   Factorial ff=new Factorial();
        for (int i=0; i<5; i++)                    //共计算 5 个阶乘结果
        {   ff.setInitVal(2*(i+1));                //计算哪个值的阶乘
            ff.result=Factorial(ff.initVal);       //计算
            ff.print();                            //输出结果
        }
    }
    public static int Factorial(int n)
    {   if (n==0) return 1;                        //递归出口
        return  n*Factorial( n-1 );                //递归计算
    }
    public void print()
    {   System.out.println(initVal+"!="+result);
    }
    public void setInitVal(int n)
    {   initVal=n;
    }
    private int result, initVal;
}
```

程序的执行结果如图 3-2 所示。

图 3-2　使用递归算法计算阶乘的执行结果

另一方面，根据阶乘的定义，$n!=n(n-1)(n-2)\cdots3\times2\times1$，完全可以用循环来计算阶乘，而不必使用递归。因为递归毕竟多次调用同一个方法，函数调用所花的时间较长，

递归调用的效率较低。

阶乘的非递归实现如下所示：

```
import java.util.*;

class Factorial2
{   private int result, initVal;
    public static void main(String[] args)
    {   Factorial2 ff=new Factorial2();
        for (int i=0; i<5; i++)
        {   ff.setInitVal(2*(i+1));              //计算初值
            ff.result=1;                         //连乘运算的初值为1
            for (int j=2; j<=ff.initVal; j++)    //循环计算连乘结果
                ff.result*=j;
            ff.print();
        }
    }
    public void print()
    {   System.out.println(initVal+"!="+result);
    }
    public void setInitVal(int n)
    {   initVal=n;
    }
}
```

3.5 使用 java.lang.Math 类，生成 100 个 0～99 之间的随机整数，找出它们之中的最大者及最小者，并统计大于 50 的整数个数。

提示：

Math 类支持 random 方法：

```
public static synchronized double random()
```

该方法返回一个 0.0～1.0 之间的小数，如果要得到其他范围的数，需要进行相应的转换。例如想得到一个 0～99 之间的整数，可以使用下列语句：

```
int num=(int) (100*Math.random());
```

解：提示中已经说明了，可以使用 Math.random()方法得到随机数，但这个随机数是一个 0.0～1.0 之间的浮点数，首先需要将数的范围变化到 0～99 之间，然后再将得到的数转换为整数。

程序中，使用了两个变量 MAXof100、MINof100 分别记录这 100 个整数中的最大值和最小值。先生成前两个随机整数，较大者放入 MAXof100 中，较小者放入 MINof100 中。随后使用一个循环生成剩余的 98 个随机整数，然后分别与 MAXof100 和 MINof100 进行比较，新生成的数如果大于 MAXof100，则将 MAXof100 修改为新的数。同样如果新生成的数小于 MINof100，则让 MINof100 记下这个数。程序中使用 count 记录大于 50

的随机数的个数，初始时，它的值为 0。

程序代码实现如下：

```
import java.util.*;

public class MathRandomTest
{   public static void main(String[] args)
    {   int count=0, MAXof100, MINof100;
        int num,i;

        MAXof100=(int) (100*Math.random());     //生成的第一个随机数
        MINof100=(int) (100*Math.random());     //生成的第二个随机数
        System.out.print(MAXof100+"   ");
        System.out.print(MINof100+"   ");
        if (MAXof100>50) count++;                //记录下大于 50 的个数
        if (MINof100>50) count++;                //记录下大于 50 的个数

        if (MINof100>MAXof100)                   //比较前两个随机数
        {   num=MINof100;
            MINof100=MAXof100;                   //较小者记入 MINof100
            MAXof100=num;                        //较大者记入 MAXof100
        }

        for (i=0; i<98; i++)                     //接下来生成其余的 98 个随机数
        {   num=(int) (100*Math.random());
            //控制每输出 10 个数即换行
            System.out.print(num+((i+2)%10==9?"\n":"   "));
            if (num>MAXof100)
                MAXof100=num;                    //更大的数记入 MAXof100
            else if (num<MINof100)
                MINof100=num;                    //更小的数记入 MINof100
            if (num>50) count++;                 //记录下大于 50 的个数
        }
        System.out.println("The MAX of 100 random integers is:  "+MAXof100);
        System.out.println("The MIN of 100 random integers is:  "+MINof100);
        System.out.println("The number of random more than 50 is:  "+count);
    }
}
```

程序的执行结果如图 3-3 所示。

【拓展思考】

(1) 哪个包包含 Scanner 类？String 类又在哪个包内？Random 类呢？Math 类呢？

解：Scanner 类和 Random 类属于 java.util 包。String 和 Math 类属于 java.lang 包。

(2) 为什么不需要在程序中引入 Math 类？

解：Math 类属于 java.lang 包，这个包自动引入到任一个 Java 程序中，所以不需要

图 3-3　随机整数的执行结果

使用单独的 import 语句来说明。

(3) 给定一个 Random 对象 rand,调用 rand.nextInt()将返回什么?

解:调用 Random 对象的 nextInt()方法返回 int 值范围内的一个随机整数,包括正数和负数。

(4) 给定一个 Random 对象 rand,调用 rand.nextInt(20)将返回什么?

解:给 Random 对象的 nextInt()方法传递一个正整数参数 x,返回 0～$x-1$ 范围内的一个随机数。所以调用 nextInt(20)将得到 0～19(含)之间的一个随机数。

(5) 写一个语句,打印 1.23 弧度的正弦值。

解:下列语句打印 1.23 弧度的正弦值:

```
System.out.println (Math.sin(1.23));
```

(6) 说明一个 double 类型变量 result,初始化为 $5^{2.5}$。

解:下列说明创建了变量 double,并初始化为 $5^{2.5}$。

```
double result=Math.pow(5, 2.5);
```

3.6　下列表达式中,找出每个操作符的计算顺序,在操作符下按次序标上相应的数字。

```
a+b+c-d
a+b/c-d
a+b/c*d
(a+b)+c-d
(a+b)+(c-d)%e
(a+b)+c-d%e
(a+b)%e%c-d
```

解:在 Java 中,在对一个表达式进行计算时,如果表达式中含有多种运算符,则要按运算符的优先顺序依次从高向低进行,同级运算符则从左向右进行。括号可以改变运算次序。运算符的优先次序参见 3.2 题答案。

各个表达式中运算符的优先次序如下:

```
a + b + c - d
  1   2   3

a + b / c - d
  2   1   3

a + b / c * d
  3   1   2

( a + b ) + c - d
    1     2   3

( a + b ) + ( c - d ) % e
    1     4     2     3

( a + b ) + c - d % e
    1     2   4   3

( a + b ) % e % c - d
    1     2   3   4
```

3.7 编写程序打印下面的图案。

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

解：从图中可以看出，该图以中间行为基准，上下对称。首先看看要打印的总行数，如果每行都不同，则需要定制各行的打印内容。如果有重复或是对称内容的话，则可以简化语句的处理。观察本题中要输出的内容，一共要输出 7 行，以 initNum 变量来表示。第 1 行和第 7 行一样，第 2 行和第 6 行一样，第 3 行和第 5 行一样，实际上我们只需要定制 4 个不同的行即可。

从 1 开始计行号。各行的星号数与行号 i 的关系为：当 i<=(initNum+1)/2 ，相应行的星号数为：(initNum-2*(i-1))；当 i>(initNum+1)/2，相应行的星号数为：(2*i-initNum)，两个星号之间有两个空格。再看每行最左侧的空格数与行号的关系：当 i<=(initNum+1)/2，相应行的空格个数为：3(i-1)；当 i>(initNum+1)/2，相应行的空格数为：(21-3i)。另外定义两个函数 printaster()和 printspace()，分别用来输出空格和星号。在程序实现时，每行先输出空格，后输出星号。

程序代码实现如下：

```
import java.util.*;

public class PrintAst
{   public static void main(String[] args)
```

```
    {   PrintAst pa=new PrintAst();
        int initNum=7;                              //总共输出 7 行

        for(int i=1;i <=initNum; i++)
        {   if(i<=(initNum+1)/ 2)                   //前半部分
            {   for(int m=1; m <=3 * (i-1); m++)
                {   pa.printSpace();                //输出空格
                }
                for(int k=1; k <=initNum-2 * (i-1); k++)
                {   pa.printAstar();                //输出星号
                }
            }
            else                                    //后半部分
            {   for(int m=21-3 * i; m>0; m--)
                {   pa.printSpace();
                }
                for(int k=1; k <=2 * i-initNum; k++)
                {   pa.printAstar();
                }
            }
            System.out.print( "\n" );
        }
    }

    public void printAstar()
    {   System.out.print( "* " );
    }
    public void printSpace()
    {   System.out.print( " " );
    }
}
```

3.8 编写程序打印下面的图案。

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

解：这个图案比 3.7 题的图案简单一些，因为它的变化是一致的，不需要分前后两部分。同样地，需要先判定输出的总行数，本题中是 10 行。接下来，确定每行输出的星号数及起始位置。从图中看出，起始位置都是从第一列开始。第一行输出 10 个星号，以后每行递减一个星号，直到最后一行仅输出一个星号。

程序代码实现如下：

```
public class PrintTriag
{   public static void main(String[] args)
    {   int initLine=10;                          //总共输出 10 行
        int initNum=10;                           //一行中最多的星号数
        PrintTriag pt=new PrintTriag();
        for (int i=0; i<initLine ; i++)
            for(int j=0; j<initNum-i; j++)
            {   pt.printAstar();
            }
        System.out.print( "\n" );
    }
    public void printAstar()
    {   System.out.print( "* " );
    }
}
```

3.9 编写程序打印乘法口诀表。

解：乘法口诀表是学习数学时入门级的知识，主要内容是 10 以内的两个数相乘的结果，按行输出。使用循环语句可以完成。输出的格式为：被乘数相同的结果输出在同一行中，乘数相同的结果输出在同一列中。

程序代码实现如下：

```
import java.util.*;

public class MultipleTable
{   public static void main(String[] args)
    {   MultipleTable mt=new MultipleTable();

        int initNum=9;                            //输出共 9 行
        int res=0;                                //计算乘法结果
        for(int i=1;i <=initNum; i++)             //行的控制
        {   for(int j=1;j <=i;j++)                //列的控制
            {   res=i*j;                          //乘积
                mt.printFormula(i, j, res);
            }
            System.out.print( "\n" );
        }
    }
```

```
    public void printFormula(int i,int j ,int res)
    {   System.out.print( i+"*" +j +"="+res+"  " );
    }
}
```

程序输出结果如图 3-4 所示。

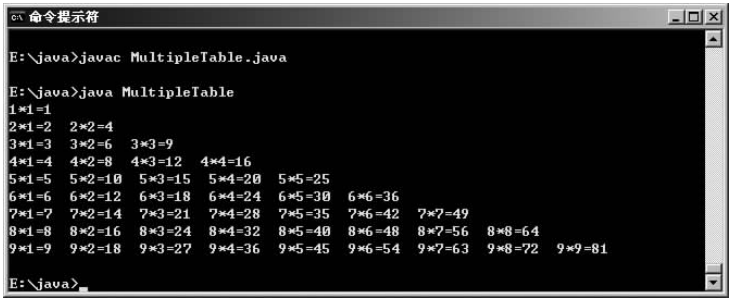

图 3-4　乘法口诀表

3.10　编写程序，要求判断从键盘输入的字符串是否为回文(回文是指自左向右读与自右向左读完全一样的字符串)。

解：如题中所说，回文即自左向右读与自右向左读完全一样的字符串。那么如何判断一个字符串是回文呢？有很多的方法，我们介绍其中比较简单的两种实现方法。

方法一：从字符串的两头相向比较，第一个字符与最后一个字符比较，第二个字符与倒数第二个字符比较，以此类推，直到字符串的中间位置为止。对中间位置字符的比较分两种情况，一是字符串中含有偶数个符号，此时刚好配对比较成功。另一种情况是字符串中含有奇数个符号，此时中间位置的符号不需要和任何符号进行比较。如果这些比较均相等，则字符串是回文。

方法二：设字符串为 w，将字符串全部反转变为 w1。例如字符串"abcdefg"反转后的结果是"gfedcba"。将 w 与 w1 进行比较，如果相等，则为回文。

从键盘输入字符串的方法，将在第 11 章详细介绍，本题中读者可以忽略此处。因为有标准输入/输出的操作，因此程序中对异常也进行了处理。异常处理是第 6 章中的内容。

方法一程序代码实现如下：

```
import java.io.*;
public class HuiWen
{   boolean isHuiWen(char str[], int n)
    {   int net=0;                                    //记录已经比较的符号数
        int i, j;                  //需要进行比较的左右标记,i 为左侧符号,j 为右侧符号
        for(i=0, j=n-1; i<n/2; i++, j--)
        {   if(str[i]==str[j]) net++;
        }
        if(net==(int)(n/2))                           //比较到了中间位置
```

```
        {   return true;
        }
        else
        {   return false;
        }
    }

    public static void main(String[] args)
    {   HuiWen hw1=new HuiWen();
        String pm="";
        Try                                          //异常处理
        {   InputStreamReader reader=new InputStreamReader(System.in);
            BufferedReader input=new BufferedReader(reader);
            System.out.print("give your test String :\n ");
            pm =input.readLine();
            System.out.println(pm);
        } catch(IOException e)                       //IOException 是个标准异常
        {   System.out.println("exception occur...");
        }

        boolean bw=hw1.isHuiWen(pm.toCharArray(), pm.length());
        if (bw==true)
        {   System.out.println("It is huiwen!");
        }
        else
        {   System.out.println("It is not huiwen");
        }
    }
}
```

程序的执行结果如图 3-5 所示。

图 3-5　HuiWen 类的执行结果

第二种方法可以让读者熟悉字符串及字符数组的使用，其程序代码实现如下：

```
import java.io.*;
public class HuiWen2
{
    String reverse(String w1)
    {
        String w2;
        char []str1=w1.toCharArray();              //字符串转成字符数组
        int len=w1.length();
        char []str2=new char[len];                 //分配一个新的字符数组
        for (int i=0; i<len; i++)
        {
            str2[i]=str1[len-1-i];                 //用来保存原字符串的反转
        }
        w2=new String(str2);                       //返回字符串型
        return w2;
    }

    public static void main(String[] args)
    {
        HuiWen2 hw1=new HuiWen2();
        String pm="";
        try
        {   InputStreamReader reader=new InputStreamReader(System.in);
            BufferedReader input=new BufferedReader(reader);
            System.out.print("give your test String :\n ");
            pm=input.readLine();
        } catch (IOException e)
        {   System.out.println("exception occur...");
        }

        String w2=hw1.reverse(pm);                 //字符串反转
        if (w2.compareTo(pm)==0)                   //字符串比较
        {   System.out.println("It is a HuiWen!");
        }
        else
        {   System.out.println("It is not a HuiWen!");
        }
    }
}
```

【拓展思考】

如何比较字符串相等?

解:使用 String 类的 equals 方法可以对字符串进行相等比较,方法返回一个布尔结果。String 类的 compareTo 方法也可用来比较字符串。它根据两字符串的大小关系,返

回正数、0 或是负数。

3.11 编写程序，判断用户输入的数是否为素数。

解：素数是只能被 1 和本身整除的整数。换句话说，除 1 和本身外，素数没有其他因子。这点可以作为判定素数的规则。对于一个整数 n，如果从 $2\sim n-1$ 之间的任何一个整数都不能整除 n，则 n 为素数。进一步的分析可知，仅需判定从 $2\sim\sqrt{n}$ 之间的任何一个整数都不能整除 n，则可判定 n 为素数。这样可以减少循环的执行次数。

据此编写判断程序如下：

```
import java.io.*;

public class PrimeNumber
{   private int pm;
    public void setPm(int pm)
    {   this.pm=pm;
    }
    public boolean isPrime()                          //判断素数
    {   boolean bl=true;
        int i=2;
        for(i=2; i <=Math.sqrt(pm);)                  //循环判定有否因子
        {   if(pm%i==0)
                {   bl=false;
                    break;                            //如果存在因子,则跳出循环
                }
            else
            {   i++;                                  //否则继续
            }
        }
        return bl;
    }

    public static void main(String args[])
    {   PrimeNumber prim=new PrimeNumber();
        int testNum=0;
        try{   InputStreamReader reader=new InputStreamReader(System.in);
               BufferedReader input=new BufferedReader(reader);
               System.out.println("give your test number : ");
               testNum=Integer.parseInt(input.readLine());
        } catch (IOException e)
        {   System.out.println("exception occur...");
        }
        prim.setPm(testNum);
        boolean bl=prim.isPrime();
        if(bl==true)
```

```
        {   System.out.print(testNum+"is a prime number\n");
        }
        else
        {   System.out.print(testNum+"is not a prime number\n");
        }
    }
}
```

程序的执行结果如图 3-6 所示。

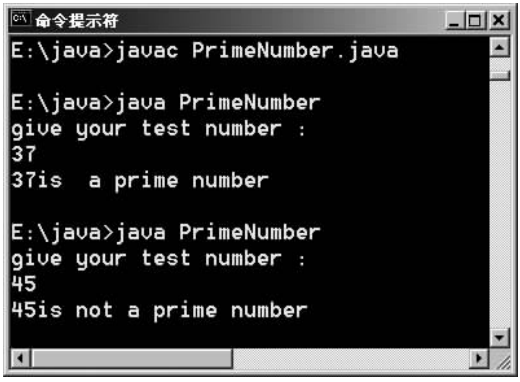

图 3-6　PrimeNumber 类的执行结果

【拓展思考】

(1) 程序中的控制流指什么?

解：对于程序的一次运行,程序的控制流决定着程序语句的执行。

(2) 条件语句与循环语句中的条件是基于什么类型的?

解：每种条件和循环语句都基于一个能得到真值或假值的布尔条件。

(3) 什么是嵌套的 if 语句? 什么是嵌套的循环语句?

解：当在 if 或是 else 子句中又出现 if 语句时,就是嵌套的 if 语句。嵌套 if 让程序做出一系列判别。同样地,嵌套的循环是在循环中又有循环。

(4) 在条件语句及循环语句中,如何使用块语句?

解：块语句将几个语句放到一起。当想基于布尔条件做多件事情时,可使用块语句来定义 if 语句或是循环语句的语句体。

(5) 在 switch 语句 case 分支的结尾处如果不使用 break 语句会发生什么?

解：如果多分支的 case 语句的最后没有 break 语句,则程序流将继续执行到下一个 case 的语句中。通常使用 break 语句,为的是跳到 switch 语句的结尾。

(6) 什么是相等运算符? 还有哪些关系运算符?

解：相等运算符是等于(==)和不等于(!=)。关系运算符还有小于(<)、小于等于(<=)、大于(>)和大于等于(>=)。

(7) 当对浮点数进行相等比较时,为什么要非常小心?

解：因为在计算机内部它们用二进制来存储,只有各个位都相同时,两个浮点值进行

精确比较时才为真。所以最好使用合理的公差值来考虑两值之间的误差。

(8) 列出 while 语句与 do 语句的相同与不同之处。

解：while 循环先计算条件。如果为真，则执行循环体。do 循环先执行循环体，再计算条件。所以 while 循环的循环体执行 0 次或多次，而 do 循环的循环体执行 1 次或多次。

(9) 何时可用 for 循环代替 while 循环?

解：当知道或能够计算循环体的迭代次数时，常使用 for 循环。while 循环处理更一般的情形。

3.12 编写程序，将从键盘输入的华氏温度转换为摄氏温度。

解：华氏温度和摄氏温度之间的转换关系为：

```
Celsius=(Fahrenheit-32)/9*5。
```

同样地，因为要处理键盘输入，所以添加了异常处理。转换程序代码实现如下：

```
import java.io.*;
public class TempConverter
{   double celsius(double y)                                    //转为摄氏温度
    {   return ((y-32)/9*5);
    }

    public static void main(String[] args)
    {   TempConverter tc=new TempConverter();
        double tmp=0;
        try
        {   //定义输入源
            InputStreamReader reader=new InputStreamReader(System.in);
            BufferedReader input=new BufferedReader(reader);
            System.out.print("give your fahrenheit temperature :\n ");
            //从键盘输入一行
            tmp=Double.valueOf(input.readLine());   //将输入转为双精度数
        }
        catch(NumberFormatException fe)
        {   System.out.println("format error...");
        }
        catch (IOException e)
        {   System.out.println("IoException occur...");
        }
        System.out.println("the celsius of temperature is"+tc.celsius(tmp));
    }
}
```

程序的执行结果如图 3-7 所示。

3.13 编写程序，读入一个三角形的三条边长，计算这个三角形的面积，并输出结果。

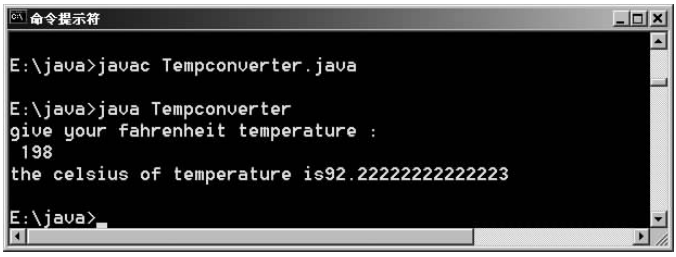

图 3-7　Temconverter 类的执行结果

提示：设三角形的三条边长分别是 a、b、c，则计算其面积的公式为：

$$s = (a+b+c)/2$$

$$面积 = \sqrt{s(s-a)(s-b)(s-c)}$$

解：根据平面几何的理论，当三个数能够满足任两数之和大于第三个数及任两数之差小于第三个数时，这三个数才能构成三角形的三条边。由三条边的逻辑关系，当上述条件满足一个时，另外一个也必然满足。因此只需要判定任两边之和大于第三边即可。根据题意，程序中定义了 Trigsquare 类，该类包括三角形的三条边这三个属性，以及判断是否能构成三角形的 isTriangle()方法和求解三角形面积的 getArea()方法。

程序代码实现如下：

```
import java.io.*;
public class Trigsquare
{   double x;                                          //x、y、z 分别为 3 条边
    double y;
    double z;
    Trigsquare (double x, double y, double z)
    {   this.x=x;
        this.y=y;
        this.z=z;
    }

    boolean isTriangle()
    {   boolean bl=false;
        if(this.x>0 && this.y>0 && this.z>0)     //必须是 3 个正数,边长不能是负值
        {   if ((this.x+this.y)>this.z &&        //任两边之和大于第三边
                (this.y+this.z)>this.x && (this.x+this.z)>this.y)
                bl=true;                         //能够构成三角形的三条边
            else
                bl=false;                        //不能构成三角形的三条边
        }
        return bl;
    }
    double getArea()
```