Java语句

本章内容

- 分支语句
  - if-else语句
  - switch语句
- 循环语句
  - for 循环
  - while 循环
  - do-while 循环
- 特殊的循环控制语句

分支语句

分支语句实现程序流程控制的功能，即根据一定的条件有选择地执行或跳过特定的语句

Java分支语句分类

- if-else 语句
- switch 语句

if-else语句语法格式

```
if(boolean类型表达式)
{
    语句或语句块;
}
```

```
if(boolean类型表达式)
{ 语句或语句块; }
else if(boolean类型表达式)
{ 语句或语句块; }
else
{ 语句或语句块; }
```

if-else语句应用举例

```
public class MyClass{
    public void judge(int age)
    {  if (age< 0)
        {       System.out.println("不可能！ ");      }
        else if (age>150)
        {       System.out.println("妖怪啊！ ");   }
        else
        {       System.out.println("此人芳龄:" + age);  }
    }
    public static void main(String args[])
    {   MyClass t = new MyClass();
        t.judge(75);    }
}
```

switch语句语法格式

```
switch(exp){
    case const1:
            statement1;
            break;
    case const2:
            statement2;
            break;
    … …
    case constN:
            statementN;
            break;
    default:
            statement_dafault;
            break;
}
```

switch语句应用举例

```
public class Test
{       public static void main(String args[])
        {   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");
             }
        }
}
```

循环语句

循环语句功能

- 在循环条件满足的情况下，反复执行特定代码

循环语句的四个组成部分

- 初始化部分（**init_statement**）
- 循环条件部分（**test_exp**）
- 循环体部分（**body_statement**）
- 迭代部分（**alter_statement**）

循环语句分类

- **for** 循环
- **while** 循环
- **do/while** 循环

for 循环语句

语法格式

```
for(init_statement; test_exp; alter_statement)
{    body_statement      }
```

应用举例

```
public class ForLoop
{    public static void main(String args[])
     {         int result = 0;
               for(int i=1; i<=100; i++)
               {      result += i;         }
               System.out.println("result=" + result);
     }
}
```

while 循环语句

语法格式

```
[init_statement]
while(test_exp)
{   body_statement;
    [alter_statement;]   }
```

应用举例

```
public class WhileLoop
{
    public static void main(String args[])
    {   int result = 0;
        int i=1;
        while(i<=100)
          {     result+= i;
                i++;        }
        System.out.println("result=" + result);
     }
}
```

do/while 循环语句

语法格式

```
[init_statement]
do
 {      body_statement;
        [alter_statement;]
} while( test_exp);
```

应用举例

```
public class WhileLoop
{       public static void main(String args[])
        {    int result = 0, int i=1;
             do
             {       result += i;
                     i++;
             }while(i<=100);
             System.out.println("result=" + result);    }  }
```

特殊流程控制语句

break 语句

- **break语句用于终止某个语句块的执行**

```
{  ……
   break;
   ……
}
```

特殊流程控制语句

break 语句用法举例

```
public class TestBreak{
    public static void main(String args[]){
        for(int i=0; i<10; i++){
            if(i==6)
                break;
            System.out.println(" i =" + i);
        }
        System.out.println("Game Over!");
    }
}
```

特殊流程控制语句

continue 语句

- **continue语句用于跳过某个循环语句块的一次执行**
- **continue语句出现在多层嵌套的循环语句体中时，可以通过标签指明要跳过的是哪一层循环**

continue语句用法举例1

```
public class ContinueTest {
    public static void main(String args[]){
        for (int i = 1; i < 101; i++) {
            if (i%10==0)
                continue;
            System.out.println(i);
        }
    }
}
```

百分制成绩转换为五分制成绩

```
import java.io.*;    //引入java.io包
class MySwitch
{
    public static void main(String args[]) throws IOException
    {    float score;
         BufferedReader  br=new  BufferedReader
                                 (new InputStreamReader(System.in));
         System.out.print("请输入成绩:");
         String str=br.readLine();  //输入字符行存入字符串
         float f=Float.parseFloat(str); //转换字符串为实型数据
```

```
    while(f>100 || f<0)
    {    System.out.println("数据错误，请重新输入");
         System.out.print("请输入成绩:");
         str=br.readLine();  //输入字符行存入字符串
         f=Float.parseFloat(str); //转换字符串为实型数据
    }
    if(Math.abs(f-100.0)<0.000001) //判断表示什么意思？
                System.out.println("A");
    else if(f>=90.0) System.out.println("A");
    else if(f>=80.0) System.out.println("B");
    else if(f>=70.0) System.out.println("C");
    else if(f>=60.0) System.out.println("D");
    else  System.out.println("E");
  }
}
```

```
    int i=(int)f/10;    //什么操作?
        switch(i)       //另一方法
        {
                case 10:
                case 9: System.out.println("A"); break;
                case 8: System.out.println("B"); break;
                case 7: System.out.println("C"); break;
                case 6: System.out.println("D"); break;
                case 5:
                case 4:
                case 3:
                case 2:
                case 1:
                case 0: System.out.println("E"); break;
        }
}}
```

常用算法

1、枚举法（穷举法）

“笨人之法”：把所有可能的情况一一测试，筛选出符合条件的各种结果进行输出。

【例一】百元买百鸡:用一百元钱买一百只鸡。已知公鸡5元/只,母鸡3元/只,小鸡1元/3只。

分析:

这是个不定方程——三元一次方程组问题（三个变量，两个方程）

$$\begin{cases} x+y+z=100 \\ 5x+3y+z/3=100 \end{cases}$$

设公鸡为x只，母鸡为y只，小鸡为z只。

```
class Hundred{
   public static void main(String[] arg){
      int x,y,z;
      for (x=0;x<=100;x++)
        for (y=0;y<=100;y++)
        {
           z=100-x-y;
           if  (5*x+3*y+z/3.0==100)
              System.out.println("cocks="+x+",
                    "+"hens="+y+","+"chickens="+z);
        }
   }
}
```

20

33

结果：cocks =0, hens =25, chickens =75
cocks =4, hens =18, chickens =78
cocks =8, hens =11, chickens =81
cocks =12, hens =4, chickens =84

2、迭代法

基本思想：不断由已知值推出新值，直到求得解为止。

迭代初值、迭代公式和迭代终止条件是迭代法的三要素

例：斐波纳契数列。前两个数都是1，第三个数是前两个数之和，以后的每个数都是其前两个数之和。

各数之间的一种递推关系，即：

$$F_n = F_{n-1} + F_{n-2}，\ F_1 = F_2 = 1$$

本题的三要素是：

迭代初值：x=1，y=1

迭代公式：z=y+x

终止条件：共计算n-2次

```
public class FibIterative
{    static int fib(int n)
     {  int first  = 1;
        int second = 1;
        int sum = first + second;
        int i = 2;
        while(++i < n)
        {   first = second;
            second = sum;
            sum = first + second;
        }
        return sum;
    }
    public static void main(String args[])
    {    System.out.println("f20=" + fib(20));      }
}
```

程序的运行结果如下:

```
java FibIterative
f20=6765
```

【累加型】类型诸如

□+□+□+□+……+□+□

求其前n项之和的编程题。

累加型算法

若设i为循环变量，s为前n项累加之和，则程序的基本结构为：

```
s=0;
for( i=1 ;i<=n  ;i++  )
    s=s+□;
```

编程求1－1/2+1/3－1/4+1/5－ … +1/99－1/100

累加型算法

程序基本结构为：

```
s=0;
for( i=1;i<=n;i++ )
    s=s+□;
```

运行结果：Sum=0.6881719

```
1) class SumDemo {
2)     static float mySum(int n) {
3)       float k = 1f;
4)       float sum = 0.0f;
5)       for(int i=1;i<=n;i++)
6)       {
7)          sum=sum+k/i;
8)          k = -k;
9)        }
10)      return sum;
11)   }
12)   public static void main(String args［］)
13)   { System.out.println(″ Sum100=″  + mySum(100)); }
14) }
```

【阶乘型】类型诸如

□×□×□×□×……×□×□

求其前n项之积的编程题。

阶乘型算法

若设i为循环变量，s为前n项相乘之积，则程序的基本结构为：

```
s=1;
for( i=1 ;i<=n  ;i++  )
    s=s*□;
```

```
class Fib {
    static int myFac(int n) {
      int fac = 1;
      for(int i=1;i<=n;i++)
      {
           fac=fac*i;
      }
      return fac;
  }
  public static void main(String args[])
  {   System.out.println("Fac10=" + myFac(5));   }
}
```

n!运行结果：
Fac10=3628800.0

编程求Σn！=1!+2!+3!…+n！（n由键盘输入）

在同一个循环中

先阶乘，后累加

$$S_i = \begin{cases} 1 & (i=1) \quad \text{初值} \\ S_{i-1} + i! & (i=2, 3, \ldots) \quad \text{递推公式} \end{cases}$$

```
1) class Example0308 {
2)     static int method(int n) {
3)       int s=0,p=1;
4)       for(int i=1; i<=n; i++) {
5)           p = p * i;      //i!
6)           s = s + p;
7)       }
8)       return s;
9)    }
10)    public static void main(String args[]) {
11)          int r = method(5);
12)          System.out.println("r=" + r);
13)    }
14) }
```

运行结果：r=153

a+aa+aaa+…+a…a=?

```
import java.io.*;
class MyLoop{
  public static void main(String args[])throws IOException
 {
      BufferedReader br=new BufferedReader(new
                              InputStreamReader(System.in));
      System.out.print("请输入一个长度n:");
      String str=br.readLine();
      //转换字符串为整型数
      int n=Integer.parseInt(str);
```

```
    System.out.print("请输入一个整数a:");
        str=br.readLine();  //输入字符存入字符串
        int a=Integer.parseInt(str);//转换字符串为整型数据
        int i=1,sn=0,tn=0;
        while(i<=n)
        {
                tn=tn+a;
                sn=sn+tn;
                a=a*10;
                ++i;
        }
        System.out.println("a+aa+aaa+...="+sn);
    }
}
```

3递归法

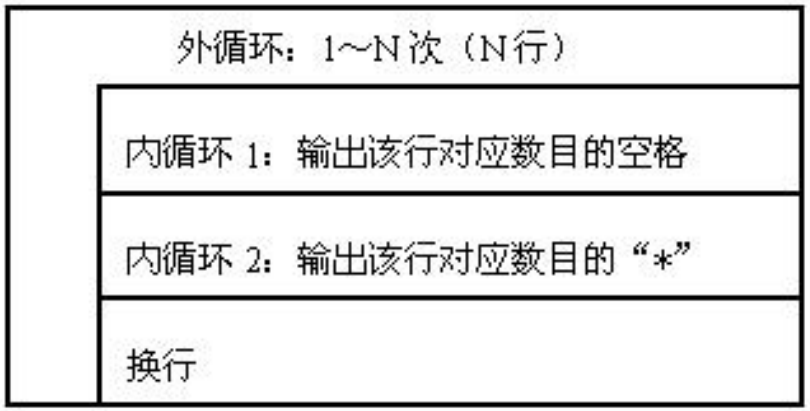
基本思想是不断把问题分解成规模较小的同类问题，直到分解形成的问题因规模足够小而能直接求得解为止。

```
1) class FibRecursion {
2)     static int fib(int n) {
3)         if(n==1 || n==2) return 1;
4)         return fib(n-1) + fib(n-2);
5)     }
6)     public static void main(String args［］) {
7)         System.out.println(″ f21=″  + fib(21));
8)     }
9) }
```

一般来说，使用递归的程序代码会更简捷，也更容易理解，但递归代码的执行效率却非常低，所以应尽量避免使用。

编程显示以下图形（共N 行，N 由键盘输入）。

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



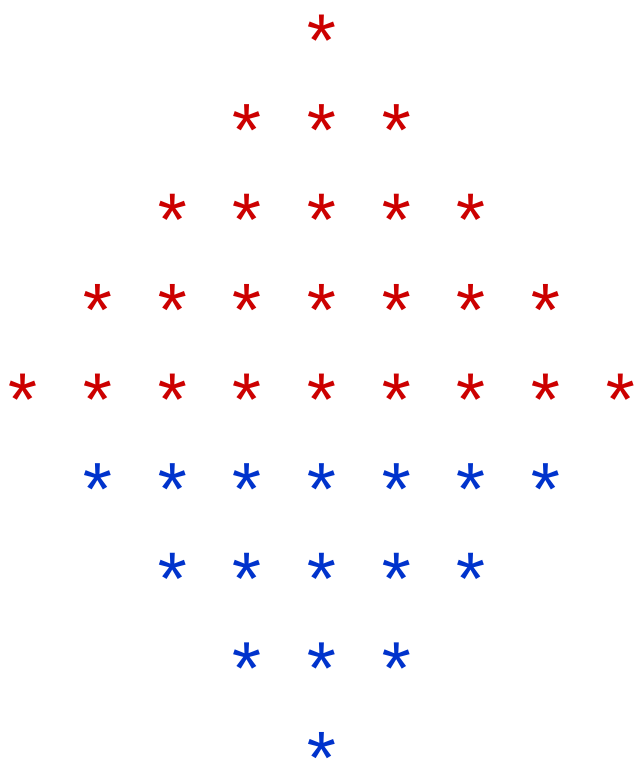
此类题目分析的要点是：通过分析，找出每行空格、* 与行号i、列号j及总行数N的关系。
其循环结构可用右图表示。

分析：（设N=5）

第1行	4个空格=5-1	1个“*”=2*行号-1
第2行	3个空格=5-2	3个“*”=2*行号-1
第3行	2个空格=5-3	5个“*”=2*行号-1
第4行	1个空格=5-4	7个“*”=2*行号-1
第5行	0个空格=5-5	9个“*”=2*行号-1

由此归纳出：第i行的空格数N-i个；
第i行的“*”数是2i-1个。

编程显示以下图形（共N 行，N 由键盘输入）。



算法：分成两部分完成：

N=9行　$N_1=(N+1)/2=5$　$N_2=N-N_1=4$

```
//第一部分,输入*个数
import java.io.*;
public class MyGraphic{
    public static void main(String args[])  throws
                                                IOException
    {   int n;
        BufferedReader br =new BufferedReader(new
              InputStreamReader(System.in));
        do{
            System.out.print("input a odd number:");
            String str=br.readLine();
            n=Integer.parseInt(str);
        }while(n%2==0); //条件值为false时结束循环
```

```
//第二部分，打印上半部分
  int topLineNum=(n+1)/2, spaceNum;
    for(int i=1;i<= topLineNum;i++)
    {
        //打印星号前的空格
        spaceNum= topLineNum-i;
        for(int j=1;j<=spaceNum;j++)
        { System.out.print(" "); }
        //打印星号
        for(int k=1;k<=2*i-1;k++)
        { System.out.print("*"); }
        System.out.println();
  }
```

```
//第二部分，打印下半部分
    for(int i=1;i<=n-topLineNum;i++)
    {
            //打印星号前的空格
            spaceNum=i;
            for(int j=1;j<=spaceNum;j++)
            {     System.out.print(" "); }
            //打印星号
            for(int k=1;k<=2*(topLineNum-i)-1;k++)
            {     System.out.print("*"); }
            System.out.println();
     }
  }
}
```

```
//第二部分另一种实现方法
  int topLineNum=(n+1)/2,spaceNum,
  maxLetterNum=n, letterNum;
  for(int i=1;i<=n;i++)
  {      //打印星号前的空格
         spaceNum=Math.abs(topLineNum-i);
         for(int j=1;j<=spaceNum;j++)
         {        System.out.print(" "); }
          //打印星号
         letterNum=maxLetterNum-2*spaceNum;
         for(int j=1;j<=letterNum;j++)
         {        System.out.print("*"); }
         System.out.println();
   }
```

方法的使用—方法的定义

方法的定义是描述实现某个特定功能所需的数据及进行的运算和操作。定义形式如下：

[modifier] returnType methodName([Parameter list])

{ // methodBody 方法体 }

返回值类型可以是基本类型、数组、类等。若方法完成的功能不返回值，**returnType**处应为**void**，而且方法体中的**return**语句不能带表达式或不用**return**语句。

修饰符**modifier**用关键字表示，修饰符是可选的，用来说明方法的某些特性。可用的修饰符有**public**、**static**、**private**等。

参数可以是基本数据,也可以是数组或类实例,参数类型可以是基本数据类型或类。方法用参数来与外界发生联系(数据传送)。

方法的使用—方法调用中的数据传送

1.值传送方式

值传送方式是将调用方法的实参的值计算出来赋予被调方法对应形参的一种数据传送方式。在这种数据传送方式下，被调方法对形参的计算、加工与对应的实参已完全脱离关系。当被调方法执行结束，形参中的值可能发生变化，但是返回后，这些形参中的值将不回带到对应的实参中。因此，值传送方式的特点是“数据的单向传送”。

使用值传送方式时，形式参数一般是基本类型的变量，实参是常量、变量，也可以是表达式。

§2.引用传送方式

使用引用传送方式时，方法的参数类型一般为复合类型（引用类型），复合类型变量中存储的是对象的引用。所以在参数传送中是传送引用，方法接收参数的引用，所以任何对形参的改变都会影响到对应的实参。因此，引用传送方式的特点是“引用的单向传送，数据的双向传送”。

1.Test类

```
public class Test
{
    int i=10;
    int j=20;
    public void method()
    {
        System.out.println("this is method!");
    }
}
```

```
public class MyTest
{   int a=10,b=20;
    void method1(int a,int b)
    { a=100;     b=200; }    //这里的a,b表示什么变量？
    void method2(Test o)
    { o.i=100;   o.j=200; }
    public static void main(String[] args)
    { MyTest obj=new MyTest();
       obj.method1(obj.a,obj.b); //按值传递
       System.out.println("a="+obj.a+" b="+obj.b); //输出？
       Test test=new Test();
       System.out.println(test.i);
       System.out.println(test.j);  //输出？
       obj.method2(test); //按引用传递
       System.out.println(test.i);
       System.out.println(test.j); //输出？
    }
}
```

下课！ 

Thank you!

用方法重载求圆、矩形、梯形面积。

```
class Area
{
    static double area(double r)
    {  return Math.PI * r * r; }
    static double area(double l,double w)
    {  return l * w; }
    static double area(double d1,double d2,double h)
    {  return (d1 + d2) * h / 2; }
```

```
public static void main(String args[]){
    double s1 = area(3.0);
    System.out.println("圆面积 = " + s1);
    double s2 = area(3.0,4.0);
    System.out.println("矩形面积 = " + s2);
    double s3 = area(3.0,4.0,5.0);
    System.out.println("梯形面积 = " + s3);
  }
}
```