


第 4 章

流程控制

( 视频讲解：31 分钟)

流程控制对于任何一门编程语言来说都是至关重要的，它提供了控制程序步骤的基本手段。如果没有流程控制语句，整个程序将按照线性的顺序来执行，不能根据用户的输入决定执行的序列。本章将向读者介绍 Java 语言中的流程控制语句。

通过阅读本章，您可以：

- » 理解 Java 语言中复合语句的使用方法
- » 掌握 if 条件语句的使用方法
- » 了解 if 语句与 switch 语句间的区别
- » 掌握 while 循环语句的使用方法
- » 掌握 do...while 循环语句的使用方法
- » 了解 while 语句与 do...while 语句的区别
- » 掌握 for 语句的使用方法
- » 了解跳转语句的使用



4.1 复合语句

 视频讲解：光盘\TM\lx\4\复合语句.exe

同 C 语言或其他语言相同，Java 语言的复合语句是以整个块区为单位的语句，所以又称块语句。复合语句由开括号“{”开始，闭括号“}”结束。

在前面的学习中已经接触到了这种复合语句。例如在定义一个类或方法时，类体就是以“{ }”作为开始与结束的标记，方法体同样也是以“{ }”作为标记。对于复合语句中的每个语句都是从上到下地被执行。复合语句以整个块为单位，可以用在任何一个单独语句可以用到的地方，并且在复合语句中还可以嵌套复合语句。

【例 4.1】 在项目中创建 Compound 类，在主方法中定义复合语句块，其中包含另一复合语句块。（实例位置：光盘\TM\sl\4\1）

```
public class Compound {
    public static void main(String args[]) {
        {
            int y = 40;
            System.out.println("输出 y 的值: "+y);
            int z = 245;
            boolean b;
            {
                b = y > z;
                System.out.println("y>z 成立吗: "+b);
            } //复合语句
        }
        String word = "hello java";
        System.out.println("输出字符串: "+word);
    }
}
```

运行结果如图 4.1 所示。

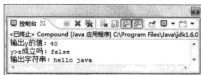


图 4.1 使用复合语句

在使用复合语句时要注意，复合语句为局部变量创建了一个作用域，该作用域为程序的一部分，在该作用域中某个变量被创建并能够被使用。如果在某个变量的作用域外使用该变量，则会发生错误，例如在本实例中如果在复合语句外使用变量 x、y、b 将会出现错误。而变量 x 可在整个方法体中使用。

4.2 条件语句

 视频讲解：光盘\TM\lx\4\条件语句.exe

条件语句可根据不同的条件执行不同的语句。条件语句包括 if 条件语句与 switch 多分支语句。本节将向读者介绍条件语句的用法。

4.2.1 if 条件语句

if 条件语句是一个重要的编程语句，它用于告诉程序在某个条件成立的情况下执行某段程序，而在另一种情况下执行另外的语句。

使用 if 条件语句，可选择是否要执行紧跟在条件之后的那个语句。关键字 if 之后是作为条件的“布尔表达式”，如果该表达式返回的结果为 true，则执行其后的语句；若为 false，则不执行 if 条件之后的语句。if 条件语句可分为简单的 if 条件语句、if...else 语句和 if...else if 多分支语句。

1. 简单的 if 条件语句

语法格式如下：

```
if(布尔表达式){
    语句序列
}
```

- ☑ 布尔表达式：必要参数，表示它最后返回的结果必须是一个布尔值。它可以是一个单纯的布尔变量或常量，或者使用关系或布尔运算符的表达式。
- ☑ 语句序列：可选参数。可以是一条或多条语句，当表达式的值为 true 时执行这些语句。如语句序列中仅有一条语句，则可以省略条件语句中的大括号。

【例 4.2】 语句序列中只有一条语句。

```
int a = 100;
if(a == 100)
    System.out.print("a 的值是 100");
```

 说明

虽然 if 和 else 语句后面的复合语句块只有一条语句，省略“{}”并无语法错误，但为了增强程序的可读性最好不要省略。

【例 4.3】 省略了 if 条件表达式中的语句序列。

```
boolean b = false;
if(b);
```

```
boolean b = false;
if(b){}
```

简单的 if 条件语句的执行过程如图 4.2 所示。

【例 4.4】 在项目中创建 Getif 类，在主方法中定义整型变量。使用条件语句判断两个变量的大小来决定输出结果。（实例位置：光盘\TM\sl\4\2）

```
public class Getif {
    public static void main(String args[]) {
        int x = 45;
        int y = 12;
        if (x > y) {
            System.out.println("变量 x 大于变量 y");
        }
        if (x < y) {
            System.out.println("变量 x 小于变量 y");
        }
    }
}
```

//创建类
//主方法
//声明 int 型变量 x，并赋给初值
//声明 int 型变量 y，并赋给初值
//判断 x 是否大于 y
//如果条件成立，输出的信息

//判断 x 是否小于 y
//如果条件成立，输出的信息

运行结果如图 4.3 所示。

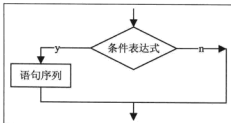


图 4.2 if 条件语句的执行过程

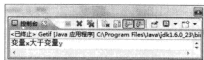


图 4.3 使用 if 语句判断大小

2. if...else 语句

if...else 语句是条件语句中最常用的一种形式，它会针对某种条件有选择地做出处理。通常表现为“如果满足某种条件，就进行某种处理，否则就进行另一种处理”。语法格式如下：

```
if(表达式) {
    若干语句
}
else {
    若干语句
}
```

if 后面()内的表达式的值必须是 boolean 型的。如果表达式的值为 true，则执行紧跟 if 语句的复合语句；如果表达式的值为 false，则执行 else 后面的复合语句。if...else 语句的执行过程如图 4.4 所示。

同简单的 if 条件语句一样，如果 if...else 语句的语句序列中只有一条语句（不包括注释），则可以省略该语句序列外面的大括号。有时为了编程的需要，else 或 if 后面的大括号里可以没有语句。

【例 4.5】 在项目中创建 Getifelse 类，在主方法中定义变量，并通过使用 if...else 语句判断变量的值来决定输出结果。（实例位置：光盘\TM\sl\4\3）

```
public class Getifelse {
    public static void main(String args[]) {
        int math = 95;
        int english = 56;
        if (math > 60) {
            System.out.println("数学及格了");
        } else {
            System.out.println("数学没有及格");
        }
        if (english > 60) {
            System.out.println("英语及格了");
        } else {
            System.out.println("英语没有及格");
        }
    }
}
```

//主方法
 //声明 int 型局部变量，并赋给初值 95
 //声明 int 型局部变量，并赋给初值 56
 //使用 if 语句判断 math 是否大于 60
 //条件成立时输出信息
 //条件不成立输出的信息
 //判断英语成绩是否大于 60
 //条件成立输出的信息
 //条件不成立输出的信息

运行结果如图 4.5 所示。

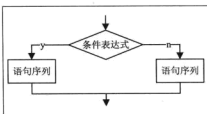


图 4.4 if...else 语句的执行过程

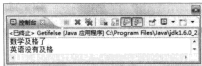


图 4.5 判断是否及格

3. if...else if 多分支语句

if...else if 多分支语句用于针对某一事件的多种情况进行处理。通常表现为“如果满足某种条件，就进行某种处理，否则，如果满足另一种则执行另一种处理”。语法格式如下：

```
if(条件表达式 1){
    语句序列 1
}
else if(条件表达式 2){
    语句序列 2
}
...
else if(条件表达式 n){
    语句序列 n
}
```

☒ 条件表达式 1~条件表达式 n：必要参数。可以由多个表达式组成，但最后返回的结果一定要

为 boolean 类型。

- ☑ 语句序列：可以是一条或多条语句，当条件表达式 1 的值为 true 时，执行语句序列 1；当条件表达式 2 的值为 true 时，执行语句序列 2，依此类推。当省略任意一组语句序列时，可以保留其外面的大括号，也可以将大括号替换为“;”。
- if...else if 多分支语句的执行过程如图 4.6 所示。

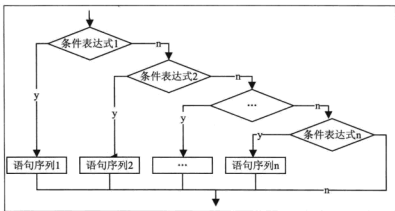


图 4.6 if...else if 多分支语句执行过程

【例 4.6】 在项目中创建 GetTerm 类，在主方法中定义变量 x，使用 if...else if 多分支语句通过判断 x 的值决定输出结果。（实例位置：光盘\TM\sl4\4）

<pre> public class GetTerm { public static void main(String args[]) { int x = 20; if (x > 30) { System.out.println("a 的值大于 30"); } else if (x > 10) { System.out.println("a 的值大于 10，但小于 30"); } else if (x > 0) { System.out.println("a 的值大于 0，但小于 10"); } else { System.out.println("a 的值小于 0"); } } } </pre>	<pre> //创建主类 //主方法 //声明 int 型局部变量 //判断变量 x 是否大于 30 //条件成立的输出信息 //判断变量 x 是否大于 10 //条件成立的输出信息 //判断变量 x 是否大于 0 //条件成立的输出信息 //当以上条件都不成立时，执行的语句块 //输出信息 </pre>
---	---

运行结果如图 4.7 所示。

在本实例中，由于变量 x 为 20，条件 x>30 为假，程序向下执行判断下面的条件；条件 x>10 为真，所以执行条件 x>10 后面的程序块中语句。输出“a 的值大于 10，但小于 30”，然后退出 if 语句。

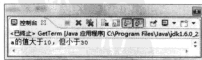


图 4.7 使用 if...else if 语句

**注意**

if 语句只执行条件为真的命令语句，其他语句都不会执行。

4.2.2 switch 多分支语句

在编程中一个常见的问题就是检测一个变量是否符合某个条件，如果不匹配，再用另一个值来检测它，依此类推。当然，这种问题使用 if 条件语句也可以完成。

【例 4.7】 使用 if 语句检测变量是否符合某个条件。

```
if(grade == "A"){
    System.out.println("真棒");
}
if(grade == "b"){
    System.out.println("做的不错");
}
```

这个程序显得比较笨重，程序员需要测试不同的值来给出输出语句。在 Java 语言中，可以用 switch 语句将动作组织起来，就能以一个较简单明了的方式来实现“多选一”的选择。语法格式如下：

```
switch(表达式)
{
    case 常量值 1;
        语句块 1
        [break;]
    ...
    case 常量值 n;
        语句块 n
        [break;]
    default;
        语句块 n+1;
        [break;]
}
```

switch 语句中表达式的值必须是整型或字符型，常量值 1~常量值 n 必须也是整型或字符型。switch 语句首先计算表达式的值，如果表达式的值和某个 case 后面的变量值相同，则执行该 case 语句后的若干个语句直到遇到 break 语句为止。此时如果该 case 语句中没有 break 语句，将继续执行后面 case 中的若干个语句，直到遇到 break 语句为止。若没有一个常量的值与表达式的值相同，则执行 default 后面的语句。default 语句为可选的，如果它不存在，而且 switch 语句中表达式的值不与任何 case 的常量值相同，switch 则不做任何处理。

**注意**

同一个 switch 语句，case 的常量值必须互不相同。



switch 语句的执行过程如图 4.8 所示。

【例 4.8】 在项目中创建 GetSwitch 类，在主方法中应用 switch 语句将周一～周三的英文单词打印出来。（实例位置：光盘\TM\sl\4\5）

```
public class GetSwitch {                                //创建类
public static void main(String args[]) {                //主方法
    System.out.println("今天是星期几：");
    int week = 2;                                       //定义 int 型变量 week
    switch (week) {                                     //指定 switch 语句的表达式为变量 week
    case 1:                                              //定义 case 语句中的常量为 1
        System.out.println("Monday");                 //输出信息
        break;
    case 2:                                              //定义 case 语句中的常量为 2
        System.out.println("Tuesday");
        break;
    case 3:                                              //定义 case 语句中的常量为 3
        System.out.println("Wednesday");
        break;
    default:                                             //default 语句
        System.out.println("Sorry,I don't Know");
    }
}
}
```

运行结果如图 4.9 所示。

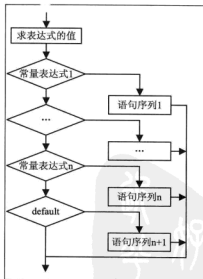


图 4.8 switch 语句的执行过程

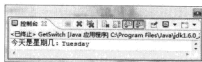


图 4.9 使用 switch 语句

注意

在 switch 语句中，case 语句后常量表达式的值可以为整数，但绝不可以是实数。例如下面的代码就是不合法的。

```
case 1.1;
```

常量表达式的值可以是字符，但一定不可以是字符串。例如下面的代码也是非法的。

```
case "ok";
```

4.2.3 范例 1：验证登录信息的合法性

大多系统登录模块都会接收用户通过键盘输入的登录信息，这些登录信息将会被登录模块验证，如果使用的是指定的用户名与密码，则允许程序登录，否则将用户拒之门外。本范例通过 if...else 语句进行多条件判断来实现登录信息的验证。运行结果如图 4.10 所示。（实例位置：光盘\TM\sl\4\6）

在项目中创建 CheckLogin 类，在该类的主方法中接收用户输入的登录用户名与登录密码，然后通过 if 条件语句分别判断用户名与密码，并输出登录验证结果。代码如下：

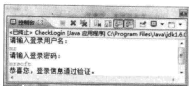


图 4.10 验证登录信息的合法性

```
import java.util.Scanner;
public class CheckLogin {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);           //创建扫描器
        System.out.println("请输入登录用户名: ");
        String username = scan.nextLine();                 //接收用户输入的登录名
        System.out.println("请输入登录密码: ");
        String password = scan.nextLine();                 //接收用户输入的登录密码
        if (username.equals("mr")) {                       //判断用户名合法性
            System.out.println("用户名非法。");
        } else if (!password.equals("mrsoft")) {          //判断密码合法性
            System.out.println("登录密码错误。");
        } else {                                           //通过两个条件判断则默认通过登录验证
            System.out.println("恭喜您，登录信息通过验证。");
        }
    }
}
```

4.2.4 范例 2：为新员工分配部门

本范例的关键技术在于 switch 多分支语句的使用，该语句只支持对常量的判断，而常量又只能是

Java 的基本数据类型，采用字符串的哈希码进行判断，也就是把 String 类的 hashCode()方法返回值作为 switch 语法的表达式，case 关键字之后跟随的是各种字符串常量的哈希码整数值。运行结果如图 4.11 所示。（实例位置：光盘\TM\sl\4\7）

在项目中创建 Example 类，在该类的主方法中创建标准输入流的扫描器，通过扫描器获取人事部门输入的姓名与应聘编程语言，然后根据每个语言对应的哈希码来判断分配部门。代码如下：

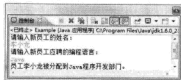


图 4.11 为新员工分配部门

```
import java.util.Scanner;
public class Example {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("请输入新员工的姓名：");
        String name = scan.nextLine();           //接收员工名称
        System.out.println("请输入新员工应聘的编程语言：");
        String language = scan.nextLine();       //接收员工应聘的编程语言

        switch (language.hashCode()) {          //根据编程语言确定员工分配的部门
            case 3254818:                        //java 的哈希码
            case 2301506:                        //Java 的哈希码
            case 2269730:                        //JAVA 的哈希码
                System.out.println("员工"+name+"被分配到 Java 程序开发部门。");
                break;
            case 3104:                           //c#的哈希码
            case 2112:                           //C#的哈希码
                System.out.println("员工"+name+"被分配到 C#项目维护组。");
                break;
            case -709190099:                      //Asp.net 的哈希码
            case 955463181:                      //Asp.net 的哈希码
            case 9745901:                        //Aap.net 的哈希码
                System.out.println("员工"+name+"被分配到 Asp.net 程序测试部门。");
                break;
            default:
                System.out.println("本公司不需要" + language + "语言的程序开发人员。");
        }
    }
}
```

在 switch 语法中每个 case 关键字可以作为一个条件分支，但是对于多个条件采取相同业务处理的情况，可以把多个 case 分支关联在一起，省略它们之间的 break 语句，而在最后一个相同的 case 分支中实现业务处理并执行 break 语句。

4.3 循环语句

 视频讲解：光盘\TM\lx\4\循环语句.exe

循环语句就是在满足一定条件的情况下反复执行某一个操作。在 Java 中提供了 3 种常用的循环语句，分别是 while 循环语句、do...while 循环语句和 for 循环语句。下面分别对这 3 种循环语句进行介绍。

4.3.1 while 循环语句

while 循环语句也称为条件判断语句，它的循环方式为利用一个条件来控制是否要继续反复执行这个语句。语法格式如下：

```
while(条件表达式)
{
    执行语句
}
```

当条件表达式的返回值为真时，则执行“{}”中的语句，当执行完“{}”中的语句后，重新判断条件表达式的返回值，直到表达式返回的结果为假时，退出循环。while 循环语句的执行过程如图 4.12 所示。

【例 4.9】在项目中创建 GetSum 类，在主方法中通过 while 循环将整数 1~10 相加。（实例位置：光盘\TM\sl\4\8）

```
public class GetSum{
    public static void main(String args[]){
        int x = 1;
        int sum = 0;
        while (x <= 10){
            sum = sum + x;
            x++;
        }
        System.out.println("1 到 10 的和是：sum = " + sum);
    }
}
```

//创建类
//主方法
//定义 int 型变量 x，并赋给初值
//定义变量用于保存相加后的结果
//while 循环语句，当变量满足条件表达式时执行循环体语句
//将变量 sum 输出

运行结果如图 4.13 所示。

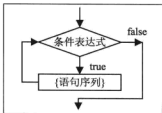


图 4.12 while 语句的执行过程

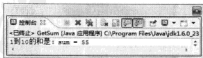


图 4.13 例 4.9 的运行结果

注意

初学者经常犯的一个错误就是在 while 表达式的括号后加 “;”。例如：

```
while(x == 5);
System.out.println("x 的值为 5");
```

这时程序会认为要执行一条空语句，而进入无限循环，Java 编译器又不会报错，可能会浪费很多时间去调试，应注意这个问题。

4.3.2 do...while 循环语句

do...while 循环语句与 while 循环语句类似，它们之间的区别是 while 循环语句为先判断条件是否成立再执行循环体，而 do...while 循环语句则先执行一次循环后，再判断条件是否成立。也就是说 do...while 循环语句中大括号中的程序段至少要被执行一次。语法格式如下：

```
do
{
    执行语句
}
while(条件表达式);
```

与 while 语句的一个明显区别是 do...while 循环语句在结尾处多了一个分号 (;)。根据 do...while 循环语句的语法特点总结出 do...while 循环语句的执行过程如图 4.14 所示。

【例 4.10】 在项目中创建 Cycle 类，在主方法中编写代码，通过本实例可看出 while 循环语句与 do...while 循环语句的区别。（实例位置：光盘\TM\4\9）

```
public class Cycle {
    public static void main(String args[]){
        int a = 100;                                //声明 int 型变量 a 并赋初值 100
        while(a == 60)                                //指定进入循环体的条件
        {
            System.out.println("ok! a==60");          //while 语句循环体
            a--;
        }
        int b = 100;                                //声明 int 型变量 b 并赋初值 100
        do
        {
            System.out.println("ok! b==100");          //do...while 语句循环体
            b--;
        } while(b == 60);                            //指定循环结束条件
    }
}
```

运行结果如图 4.15 所示。

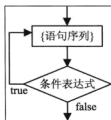


图 4.14 do...while 循环语句的执行过程

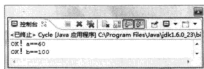


图 4.15 使用 do...while 循环语句

4.3.3 for 循环语句

for 循环语句是 Java 程序设计中最有用的循环语句之一。一个 for 循环可以用来重复执行某条语句，直到某个条件得到满足。在 Java 5 以后新增了 foreach 语法，本节将对这两种 for 循环形式进行详细的介绍。

1. for 语句

语法格式如下：

```
for(表达式 1;表达式 2;表达式 3)
{
    语句序列
}
```

- ☑ 表达式 1: 初始化表达式, 负责完成变量的初始化。
- ☑ 表达式 2: 循环条件表达式, 值为 `boolean` 型的表达式, 指定循环条件。
- ☑ 表达式 3: 循环后操作表达式, 负责修整变量, 改变循环条件。

在执行 for 循环时, 首先执行表达式 1, 完成某一变量的初始化工作; 下一步判断表达式 2 的值, 若表达式 2 的值为 true, 则进入循环体; 在执行完循环体后紧接着计算表达式 3, 这部分通常是增加或减少循环控制变量的一个表达式。这样一轮循环就结束了。第二轮循环从计算表达式 2 开始, 若表达式 2 返回 true, 则继续循环, 否则跳出整个 for 语句。for 循环语句的执行过程如图 4.16 所示。

【例 4.11】 在项目中创建 Circulate 类，在主方法中使用 for 循环语句来计算 2~100 之间所有偶数之和。

```
public class Circulate{                                //创建 Circulate 类
    public static void main(String args[]){           //主方法
        int sum = 0;                                   //声明变量，用于保存各数相加后的结果
        for(int i = 2;i<=100;i+=2){                  //指定循环条件及循环体
            sum = sum+i;
        }
    }
}
```

```

        System.out.println("2 到 100 之间的所有偶数之和为: "+sum);    //将相加后的结果输出
    }
}

```

运行结果如图 4.17 所示。

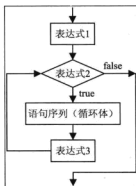


图 4.16 for 循环语句的执行过程



图 4.17 使用 for 循环语句计算偶数和

2. foreach 语句

foreach 语句是 for 语句的特殊简化版本，foreach 语句并不能完全取代 for 语句，然而任何 foreach 语句都可以改写为 for 语句版本。foreach 并不是一个关键字，习惯上将这种特殊的 for 语句格式称之为 foreach 语句。foreach 语句在遍历数组等方面为程序员提供了很大的方便（本书将在第 5 章对数组进行详细的介绍）。语法格式如下：

```

for(元素变量 x : 遍历对象 obj){
    引用了 x 的 Java 语句;
}

```

foreach 语句中的元素变量 x，不必对其进行初始化。下面通过简单的例子来介绍 foreach 语句是怎样遍历一维数组的。

【例 4.12】 在项目中创建 Repetition 类，在主方法中定义一维数组，并用 foreach 语句遍历该数组。（实例位置：光盘\TM\sl\4\10）

```

public class Repetition {                                //创建 Repetition 类
    public static void main(String args[]){              //主方法
        int arr[] = {7, 10, 1};                          //声明一维数组
        System.out.println("一维数组中的元素分别为: "); //输出信息
        for (int x : arr) {                                //foreach 语句
            System.out.println(x+"");
        }
    }
}

```

运行结果如图 4.18 所示。



图 4.18 使用 foreach 语句遍历数组

4.3.4 范例 3: 使用 while 循环遍历数组

本范例利用自增运算符结合 while 循环获取每个数组元素的值,然后把它们输出到控制台中。其中自增运算符控制索引变量的递增。运行结果如图 4.19 所示。(实例位置:光盘\TM\sl\4\11)

创建 `ErgodicArray` 类，在该类的主方法中创建一个鸟类数组，然后创建一个索引变量，这个变量用于指定数组下标，随着该索引的递增，`while` 循环会逐步获取每个数组的元素并输出到控制台中。代码如下：



图 4.19 使用 while 循环遍历数组

```
public class ErgodicArray {
    public static void main(String[] args) {
        String[] aves = new String[] { "白鹭", "丹顶鹤", "黄鹂", "鸚鵡", "乌鸦", "喜鹊",
            "布谷鸟", "灰纹鸟", "百灵鸟" };
        int index = 0;
        System.out.println("我的花园里有很多鸟，种类大约包括：");
        while (index < aves.length) {
            System.out.print(aves[index++]+" ");
        }
    }
}
```

4.3.5 范例 4：使用 for 循环输出九九乘法表

Java 基本语法中的 for 循环非常灵活并且可以嵌套使用,其中双层 for 循环是程序开发中使用最频繁的,常用于操作表格数据,对于行数与列数相同的表格操作代码比较简单,但是类似九九乘法表就不好控制了,因为它的列数要与行数对应,可以说这个表格是个三角形,本范例通过双层循环输出了这个九九乘法表。运行结果如图 4.20 所示。(实例位置:光盘\TM\sl\4\12)

创建 `MultiplicationTable` 类,在该类的主方法中创建双层 `for` 循环。第一层 `for` 循环也称为外层循环,用于控制表格的行;第二层循环也称为内层循环,用于控制表格的列。这里第二层循环的控制变量非常重要,它的条件判断是列数要等于行数的最大值,然后输出内层与外层循环控制变量的乘积,这样就实现了九九乘法表。代码如下:

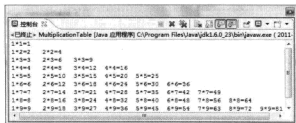


图 4.20 使用 for 循环输出乘法表

```
public class MultiplicationTable {
    public static void main(String[] args) {
        for(int i=1;i<=9;i++){
            for(int j=1;j<=i;j++){
                System.out.print(j+"*"+i+"="+i*j+" ");
            }
            System.out.println();
        }
    }
}
```

//循环控制变量从 1 遍历到 9
 //第二层循环控制变量与第一层最大索引相等
 //输出计算结果但不换行
 //在外层循环中换行

4.4 跳转语句

 视频讲解：光盘\TM\1\4\跳转语句.exe

Java 语言中提供了 3 种跳转语句，分别是 break 语句、continue 语句和 return 语句。下面对这 3 种跳转语句进行详细介绍。

4.4.1 break 语句

break 语句大家应该不会陌生，在介绍 switch 语句时已经应用过了。在 switch 语句中，break 语句用于中止下面 case 语句的比较。实际上，break 语句还可以应用在 for、while 和 do...while 循环语句中，用于强行退出循环，也就是忽略循环体中任何其他语句和循环条件的限制。

【例 4.13】 使用 for 循环语句计算 1~100 之间所有连续整数的和。(实例位置：光盘\TM\1\4\13)

```
public class MultiplicationTable {
    public static void main(String[] args) {
        int sum=0;
        String flag="从 1 到 100 之间连续整数的和是：";
        for(int i=1;i<=100;i++){
            sum+=i;
        }
    }
}
```

//定义保存数据和的 int 变量
 //给出输出信息
 //循环获取从 1~100 的数
 //将各数相加


```

        System.out.println(flag+sum);           //输出相加后的结果
    }
}

```

运行结果如图 4.21 所示。

在上面的循环中添加通过 if 语句控制的 break 语句，具体代码如下：

```

public class MultiplicationTable {
    public static void main(String[] args) {
        int sum=0;
        String flag="从 1 到 100 之间连续整数的和是：";
        for(int i=1;i<=100;i++){
            sum+=i;
            if(sum>1000){
                flag="从 1 到"+i+"之间连续整数的和是：";
                break;
            }
        }
        System.out.println(flag+sum);
    }
}

```

//循环获取从 1~100 的数
//将各数进行相加
//如果 sum 大于 1000
//break 关键字退出循环

运行结果如图 4.22 所示。



图 4.21 输出 1~100 的和

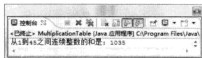


图 4.22 使用 break 语句

从上面的两段代码中可以看出，虽然 for 循环被设计为计算从 1~100 之间所有连续整数的和，但是由于当累加和大于 1000 时使用 break 语句中止了 for 循环语句，所以当循环结束时 i 的值并不等于 100，而是等于 45。需要说明的是，使用 break 语句只能退出当前循环。

4.4.2 continue 语句

continue 语句只能应用在 for、while 和 do...while 循环语句中，用于让程序直接跳过其后面的语句，进行下一次循环。

【例 4.14】 在项目中创建 ContinueDemo 类，在主方法中应用 while 循环语句和 continue 语句输出 10 以内的全部奇数。（实例位置：光盘\TM\sl4\14）

```

public class ContinueDemo{
    public static void main(String[] args) {
        int i = 0;
        System.out.println("十以内的全部奇数是：");
        while (i < 10) {
            //定义循环增量

```

```

i++;
if (i % 2 == 0) {
    continue;
}
System.out.print(i + " ");
}
}

```

//累加 i 的值
//当 i 的值能被 2 整除，表示该数不是奇数
//进行下一次循环

//输出 i 的值

运行结果如图 4.23 所示。

当使用 `continue` 语句中止本次循环后，如果循环条件的结果为 `false`，则退出循环，否则继续下一次循环。

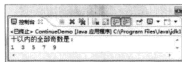


图 4.23 输出 10 以内的全部奇数

4.4.3 return 语句

`return` 语句可以从一个方法返回，并把控制权交给调用它的语句。语法格式如下：

```
return [表达式];
```

表达式：可选参数，表示要返回的值。它的数据类型必须与方法声明中的返回值类型一致，可以通过强制类型转换实现。

`return` 语句通常被放在被调用方法的最后，用于退出当前方法并返回一个值。当把单独的 `return` 语句放在一个方法的中间时，会产生 `Unreachable code` 编译错误。但是可以通过把 `return` 语句用 `if` 语句括起来的方法，将 `return` 语句放在一个方法中间，用来实现在程序未执行完方法中的全部语句时退出。

4.4.4 范例 5：终止循环体

循环用于复杂的业务处理，可以提高程序的性能和代码的可读性，但是循环中也有特殊情况，如由于某些原因需要立刻中断循环去执行下面的业务逻辑。运行结果如图 4.24 所示。（实例位置：光盘\TM\sl\4\15）



图 4.24 终止循环体

在项目中创建 `BreakCyc` 类，在该类的主方法中创建一个字符串数组，在使用 `foreach` 语句遍历

判断如果发现数组中包含字符串“老鹰”则立刻中断循环。然后创建一个整数类型二维数组，使用双层 foreach 循环遍历，当发现第一个小于 60 的数组元素时，则立刻中断整个双层循环，而不是内层循环。代码如下：

```
public class BreakCyc {
    public static void main(String[] args) {
        System.out.println("\n-----中断单层循环的例子。-----");
        String[] array = new String[] { "白鹭", "丹顶鹤", "黄鹂", "鹦鹉", "乌鸦", "喜鹊", "老鹰", "布谷鸟", "老鹰", "灰纹鸟", "老鹰", "百灵鸟" }; //创建数组
        System.out.println("在你发现第一只老鹰之前，告诉我都有什么鸟。");
        for (String string : array) { //foreach 遍历数组
            if (string.equals("老鹰")) //如果遇到老鹰
                break; //中断循环
            System.out.print("有： " + string + " "); //否则输出数组元素
        }

        System.out.println("\n\n-----中断双层循环的例子。-----");

        int[][] myScores = new int[][] { { 67, 78, 63, 22, 66 }, //创建成绩数组
                                         { 55, 68, 78, 95, 44 }, { 95, 97, 92, 93, 81 } };
        System.out.println("宝宝这次考试成绩：\n 数学\t 语文\t 英语\t 美术\t 历史");
        No1: for (int[] is : myScores) { //遍历成绩表格
            for (int i : is) {
                System.out.print(i + "\t"); //输出成绩
                if (i < 60) { //如果中途遇到不及格的，立刻中断所有输出
                    System.out.println("\n 等等，" + i + "分的是什么？这个为什么不及格？");
                    break No1;
                }
            }
            System.out.println();
        }
    }
}
```

4.4.5 范例 6：循环体的过滤器

循环体中可以通过 break 语句中断整个循环，这增加了循环的控制能力，但是对于特殊情况还是不够，例如某些条件下需要放弃部分循环处理，而不是整个循环体。Java 提供了 continue 语句来实现这一功能，continue 可以放弃本次循环体的剩余代码，不执行它们而开始下一轮的循环。本范例利用 continue 语句实现了循环体过滤器，可以过滤“老鹰”字符串，并做相应的处理，但是放弃 continue 语句之后的所有代码。运行结果如图 4.25 所示。（实例位置：光盘\TM\sl\4\16）



图 4.25 循环体的过滤器

在项目中创建 `CycFilter` 类，在该类的主方法中创建鸟类名称的字符串数组，其中包含多个“老鹰”字符串，然后通过 `foreach` 循环遍历该数组，在循环过程中如果遍历的数组元素是“老鹰”字符串，则输出发现老鹰的信息并过滤循环体之后的所有代码。代码如下：

```
public class CycFilter {
    public static void main(String[] args) {
        String[] array = new String[] { "白鹭", "丹顶鹤", "黄鹌", "鸚鵡", "乌鸦", "喜鹊", //创建数组
            "老鹰", "布谷鸟", "老鹰", "灰纹鸟", "老鹰", "百灵鸟" };
        System.out.println("在我的花园里有很多鸟类，但是最近来了几只老鹰，请帮我把它们抓走。");
        int eagleCount = 0;
        for (String string : array) { //foreach 遍历数组
            if (string.equals("老鹰")) { //如果遇到老鹰
                System.out.println("发现一只老鹰，已经抓到笼子里。");
                eagleCount++;
                continue; //中断循环
            }
            System.out.println("搜索鸟类，发现了: " + string); //否则输出数组元素
        }
        System.out.println("一共捉到了: " + eagleCount + "只老鹰。");
    }
}
```

`break` 语句和 `continue` 语句都是对循环体的控制语句，它们不仅应用于 `for` 循环，在任何循环体中都可以使用这些语句，灵活使用可以让循环实现更加复杂的运算和业务处理。

4.5 经典范例

4.5.1 经典范例 1：使用 `for` 循环输出空心的菱形

 视频讲解：光盘\TM\4\4使用 `for` 循环输出空心的菱形.exe

输出空心的菱形图案，这在等级考试与公司面试时也出现过类似题目，本范例的目的在于熟练掌握

握 for 循环的嵌套使用。运行结果如图 4.26 所示。(实例位置: 光盘\TM\sl\4\17)

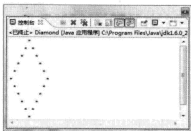


图 4.26 输出空心菱形

创建 Diamond 类, 在该类的主方法中调用 printHollowRhombus() 方法完成 10 行的空心菱形输出。其中 printHollowRhombus() 方法是范例中自定义的, 该方法使用两个双层 for 循环分别输出菱形的上半部分与下半部分。代码如下:

```
public class Diamond {
    public static void main(String[] args) {
        printHollowRhombus(10);
    }

    public static void printHollowRhombus(int size) {
        if (size % 2 == 0) {
            size++;
            //计算菱形大小
        }
        for (int i = 0; i < size / 2 + 1; i++) {
            for (int j = size / 2 + 1; j > i + 1; j--) {
                System.out.print(" ");
                //输出左上角位置的空白
            }
            for (int j = 0; j < 2 * i + 1; j++) {
                if (j == 0 || j == 2 * i) {
                    System.out.print("***");
                    //输出菱形上半部边缘
                } else {
                    System.out.print(" ");
                    //输出菱形上半部空心
                }
            }
            System.out.println("");
        }
        for (int i = size / 2 + 1; i < size; i++) {
            for (int j = 0; j < i - size / 2; j++) {
                System.out.print(" ");
                //输出菱形左下角空白
            }
            for (int j = 0; j < 2 * size - 1 - 2 * i; j++) {
                if (j == 0 || j == 2 * (size - i - 1)) {
                    System.out.print("***");
                    //输出菱形下半部边缘
                } else {
                    System.out.print(" ");
                    //输出菱形下半部空心
                }
            }
        }
    }
}
```

```

    }
    System.out.println("");
}
}
}

```

4.5.2 经典范例 2：使用 for 循环输出杨辉三角

 视频讲解：光盘\TM\lx\4\使用 for 循环输出杨辉三角.exe

杨辉三角形由数字排列，可以把它看作一个数字表，其基本特性是两侧数值均为 1，其他位置的数值是其正上方的数值与左上角数值之和。本范例通过数组来实现这个杨辉三角形。运行结果如图 4.27 所示。（实例位置：光盘\TM\sl\4\18）

创建 YanghuiTriangle 类，在该类的主方法中创建一个二维数组，并指定二维数组的第一维长度，这个数组用于存放杨辉三角形的数值表，通过双层 for 循环来实现第二维数组的长度，然后计算整个数组的每个元素的值。代码如下：

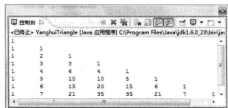


图 4.27 输出杨辉三角

```

public class YanghuiTriangle {
    public static void main(String[] args) {
        int triangle[][]=new int[8][];           //创建二维数组
        for (int i = 0; i < triangle.length; i++) {
            triangle[i]=new int[i+1];           //遍历二维数组的第一层
            //初始化第二层数组的大小
            for(int j=0;j<=triangle[i].length-1;j++){
                //遍历第二层数组
                if(i==0||j==0||j==triangle[i].length-1){
                    //将两侧的数组元素赋值为 1
                    triangle[i][j]=1;
                }else{
                    //其他数值通过公式计算
                    triangle[i][j]=triangle[i-1][j]+triangle[i-1][j-1];
                }
                System.out.print(triangle[i][j]+" ");           //输出数组元素
            }
            System.out.println();
        }
    }
}

```

4.6 本章小结

本章向读者介绍了流程控制语句（复合语句、条件语句和循环语句）。使用复合语句可以为变量定义一个有效区域。通过使用 if 与 switch 语句，可以基于布尔类型的测试，将一个程序分成不同的部分。

通过 while、do...while 循环语句和 for 循环语句，可以让程序的一部分重复地执行，直到满足某个终止循环的条件。通过本章的学习，读者应该学会在程序中灵活使用流程控制语句。

4.7 实战练习

1. 编写 Java 程序，实现判断变量 x 是奇数还是偶数。(答案位置：光盘\TM\sl\4\19)
2. 编写 Java 程序，应用 for 循环打印菱形。(答案位置：光盘\TM\sl\4\20)
3. 编写 Java 程序，使用 while 循环语句计算 $1+1/2!+1/3!+\dots+1/20!$ 之和。(答案位置：光盘\TM\sl\4\21)



