第3章

Java 语句及其控制结构

任何 Java 程序都是由类和对象组成的，对象和类是由变量与方法组成的，变量由声明变量语句与赋值语句构成，方法由一系列执行不同功能的 Java 语句构成。Java 语句有不同的类型，不同类型的语句其组成成分不同、含义不同、作用不同，有的语句在程序中的位置顺序也有一定的规则，这些也是 Java 语言的基本语法内容。学习 Java 程序设计，就是要了解不同 Java 语句的构成方式、含义与作用，了解如何组织 Java 语句构建 Java 类与对象的框架，定义变量确定类与对象的特性，定义方法实现与控制类的功能。

本章的内容主要解决以下问题。

- Java 程序有哪些主要构成成分？
- Java 类有哪些主要构成成分？
- Java 有哪些类型的语句？
- Java 的选择语句是如何构成的？其作用是什么？
- Java 的循环语句是如何构成的？其作用是什么？
- Java 的跳转语句是如何构成的？其作用是什么？

3.1 Java 语句的类型

本节主要介绍 Java 程序的构成成分、类的构成成分、Java 语句的种类以及说明性语句、表达式语句和复合语句的特点。

3.1.1 Java 程序构成

例 3.1 一个程序范例，用来说明 Java 程序的构成成分。

```
import java.awt.*;
import javax.swing.*;        ] Java 包引入语句
public class exp3_1 extends JFrame   ——类声明语句
```

```
{
  Label label1=new Label("类的结构");     ——成员变量
  public exp3_1 () {                                          构造方法
    setSize(400, 200);
      setVisible(true);
    add("North", label1);
  }
  public void paint(Graphics g) {                             paint 方法
        g.fillRect(100, 80, 80, 100);
  }
  public static void main(String[] args) {                    main 方法
        new exp3_1();
  }
}
```

类体

说明：

(1) 程序的主要构成。从例 3.1 可以看到，Java 程序一般包括两部分：Java 包引入(如果有的话)语句、类定义语句。类定义语句由类声明语句和类体组成。

(2) 类声明语句。类声明语句用 class 来声明类的名称，其他关键字为修饰符，来说明类的访问权限，以及类的属性。详细介绍看第 4 章。

(3) 类体。类体由成员变量和成员方法等组成。成员变量表示类的属性，是数据域的形式表达；方法是数据的操作的定义。在 exp3_1 类中，label1 成员变量，其他是成员方法。

成员变量在类中定义时，需要声明成员变量的名称、类型或初值。

成员方法是命令语句的集合，用来实现类的功能和行为，是程序设计的关键。

(4) 主类与普通类。类可以分为主类与普通类。

Java 程序中，必须含有一个可被外界(通常是 Java 解释器)直接调用的类，这个类称为这个 Java 程序的主类，即可以被运行的类。主类常用 public 关键字来修饰。一个 Java 程序中可以定义多个类，但只能有一个主类，主类中一定包含 main 方法，其用来控制程序进行数据初始化工作，实现不同的功能或调用其他对象，输出程序运行后的结果。整个应用程序是从 main 方法开始执行的。

不是主类的类都是普通类，可以用来定义属性、方法，设计对象的功能。

3.1.2 Java 语句的种类

Java 语句是 Java 标识符的集合，由关键字、常量、变量和表达式构成。简单的 Java 语句以分号(;)作为结束标志，单独的一个分号被看作一个空语句，空语句不做任何事情。复合结构的 Java 语句以花括号(})作为结束标志，凡是可以出现语句的地方，都可以出现复合语句。

Java 语句一般分为说明性语句和操作性语句两种类型。

1. 说明性语句

Java 的说明性语句包含包和类引入语句、声明类语句、声明变量语句、声明对象语句等。例如：

```
import java.sql.Connection;          //包引入语句
public class ReadViolation;          //声明类语句
int i;                               //声明变量语句
```

2. 操作性语句

Java 的操作性语句包含表达式语句、复合语句、选择语句和循环语句、跳转语句等。Java 规定所有的操作性语句必须放在成员方法中。

下面介绍表达式语句与复合语句的构成，其他操作性语句将分别在后面介绍。

1) 表达式语句

在表达式后边加上分号(;)就是一个表达式语句。表达式语句是最简单的语句，它们被顺序执行，完成相应的操作任务。表达式语句主要有赋值语句和方法调用语句。

例如：

```
int x, i=1, j=2;                     //声明变量语句
x=j-i;                               //赋值语句
System.out.println("x:"+k);          //方法调用语句
```

注意：声明变量时可以直接赋初值，但不属于赋值语句，为声明变量语句。

2) 复合语句

复合语句也称为块(block)语句，是包含在一对花括号({})中的任意语句序列，由一组 0 和多条语句组成。与其他语句用分号作结束符不同，复合语句右括号(})后面不需要分号。尽管复合语句含有任意多个语句，但从语法上讲，一条复合语句被看作一条简单语句。

例 3.2 复合语句示例，程序运行结果为“Condition is true.”。源代码如下所示：

```
public class exp3_2 {
    public static void main(String[] args) {
        boolean condition=true;
    if (condition) {     //执行第一个块语句
            System.out.println("Condition is true.");
        } //结束 block1
        else {           //执行第二个块语句
            System.out.println("Condition is false.");
        } //结束 block2
    }
}
```

说明：在程序中通过两条复合语句实现了对于 condition 条件的判断，如果 condition 为真，则执行第一个块语句 block1，否则执行第二个块语句 block2。在实际应用中，复合

语句更广泛地应用在结构式语句中。

3.2 选择语句

表达式语句与复合语句都是按顺序从上到下逐行执行每条命令。能否改变程序中语句执行的顺序呢？利用选择语句结构就可以根据条件控制程序流程，改变语句执行的顺序。

本节主要介绍 Java 的 4 种选择语句：单分支选择语句(if 语句)、二分支选择语句(if…else 语句)、多分支选择语句(if…else if…else 语句)和先执行后判定循环(do…while)的使用方法。

3.2.1 单分支选择语句(if 语句)

单分支选择语句是一种简单的 if 条件语句，其语法格式如下：

```
if (条件){
    s1(语句);
```

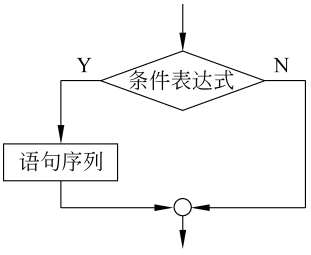

图 3.1 if 条件语句执行结果

这是最简单的单分支结构。当构成条件的逻辑或关系表达式的值为 true 时，执行 s1 语句，否则忽略 s1 语句。s1 语句可以是复合语句，如果只有一条语句，花括号可以省略不写。条件是必要的参数，可以由多个表达式组成，但是，最后结果一定是 boolean 类型，其返回的结果值一定是 true 或者 false。if 条件语句执行结果如图 3.1 所示。

3.2.2 二分支选择语句(if…else 语句)

if 语句通常与 else 语句配套使用，形成二分支结构。它的语法格式为：

```
if (条件){
    s1 语句;          //如果条件表达式的值为 true
}else{
    s2 语句;          //如果条件表达式的值为 false
}
```

当构成条件的逻辑或关系表达式的值为 true 时，执行 s1 语句，忽略 else 和 s2 语句；否则，条件表达式的值为 false，程序忽略 s1 语句，执行 else 后面的 s2 语句。s1 和 s2 都可以是复合语句。if…else 语句执行结果如图 3.2 所示。

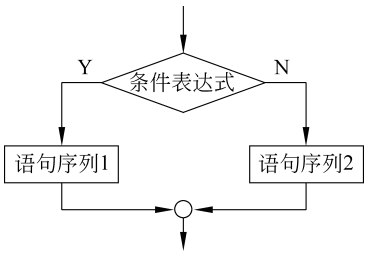

图 3.2 if…else 语句执行结果

例 3.3 判断两个数的大小并输出指定语句，程序运行结果为“这是 if 语句”，源程序代码如下所示：

```
public class exp3_3 {
    public static void main(String args[]){
        int x=10;
        if(x<20){ System.out.print("这是 if 语句");}
        else{ System.out.print("这是 else 语句");}
    }
}
```

说明：程序通过 if…else 语句实现了对传入参数的比较判断功能。参数 x 的值为 10，小于 20，条件判断 x＜20 的值为 true，因此执行输出“这是 if 语句”，忽略 else 语句，最终输出“这是 if 语句”。

3.2.3 多分支选择语句（if…else if…else 语句）

对于超过二分支选择的情况，可以使用多分支选择语句（if…else if…else 语句）。它的语法格式为：

```
if (条件 1)  s1 语句;          //如果布尔条件 1 的值为 true 执行 s1 语句
else if (条件 2)  s2 语句;     //如果布尔条件 2 的值为 true 执行 s2 语句
else if (条件 3)  s3 语句;     //如果布尔条件 3 的值为 true 执行 s3 语句
else  s4 语句;                 //如果以上布尔条件都不为 true 执行 s4 语句
```

在这里依次计算条件的值，如果某个条件的值为 true，就执行它后面的语句，其余语句被忽略；所有条件的值都为 false，就执行最后一个 else 后的 s4 语句。s1、s2、s3 和 s4 都可以是复合语句。if…else if…else 语句执行结果如图 3.3 所示。

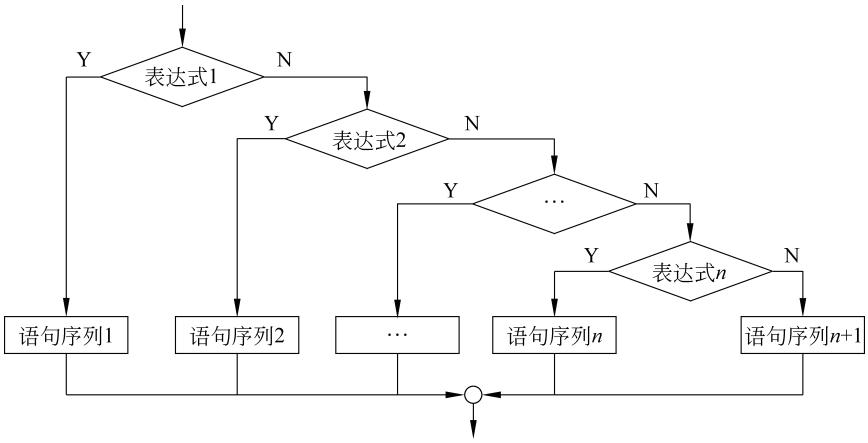

图 3.3　if…else if…else 语句执行结果

例 3.4　下面是使用 if…else if…else 语句的简单例子。输出结果为“X is 20”。源程序代码如下所示：

```
public class exp3_4{
    public static void main(String args[]){
```

```
        int x=20;
        if(x==5){System.out.print("X is 5");}
            else if(x==15){System.out.print("X is 15");}
            else if(x==25){System.out.print("X is 25");}
        else{System.out.print("X is:"+x); }
    }
}
```

说明：程序使用了多分支 if…else if…else 语句进行多个条件的判断，并在最后使用了 else 语句来对多分支选择语句进行补充。如果传入的初始化值满足其中的 else if 语句，则会跳过执行其他的 else if 以及 else 语句。要注意 if…else if…else 语句中最多只能有一个 else 语句。

3.2.4 嵌套的 if…else 语句

嵌套的 if…else 语句是指可以在另一个 if 或者 else if 语句中使用一个或者多个的 if 或者 else if 语句。在嵌套的语句中最好给语句加上花括号，以提高代码的可读性。其语法格式为：

```
if(条件 1){
    if(条件 2){
        s1 语句
    }else{
        s2 语句
    }
}else{
    if(条件 3){
        s3 语句
    }else{
          s4 语句
    }
}
```

```
2008 年是闰年
2020 年是闰年
2050 年不是闰年
```

图 3.4 嵌套使用 if…else 语句执行结果

例 3.5 下面是一个嵌套使用 if…else 语句与 if…else if…else语句构造多分支程序的例子，判断某一年是否为闰年。闰年的条件是符合下面二者之一：能被 4 整除，但不能被 100 整除；能被 4 整除，又能被 100 整除。输出结果如图 3.4 所示。

```
public class exp3_5 {
  public static void main(String args[]) {
    boolean leap;  int year=2008;
    //方法 1
    if ((year%4==0 && year%100!=0)||(year%400==0))
        System.out.println(year+"年是闰年");
    else  System.out.println(year+"年不是闰年");
```

```
    //方法 2
    year=2020;
    if (year%4!=0)  leap=false;
      else if (year%100!=0)   leap=true;
      else if (year%400!=0)   leap=false;
    else  leap=true;
    if (leap==true)  System.out.println(year+"年是闰年");
    else   System.out.println(year+"年不是闰年");
    //方法 3
    year=2050;
    if (year%4==0) {
      if (year%100==0) {
        if (year%400==0)  leap=true;
        else  leap=false;
      }
      else  leap=false;
    }
    else leap=false;
    if (leap==true)  System.out.println(year+"年是闰年");
    else  System.out.println(year+"年不是闰年");
  }
}
```

说明：方法 1 用一个逻辑表达式包含了所有的闰年条件；方法 2 使用了多分支选择语句(if…else if…else 语句)；方法 3 通过花括号({})对 if…else 进行匹配来实现闰年的判断。大家可以根据程序对比这 3 种方法，体会其中的联系和区别，在不同的场合选用适当的方法。

3.2.5 开关语句(switch 语句)

可以看出，虽然嵌套的条件语句可实现多个分支处理，但嵌套太多时容易出错和混乱，这时可以使用开关语句 switch 处理。使用它可以容易地写出判断条件，特别是有很多条件选项的时候。

开关语句 switch 的语法格式为：

```
switch(表达式) {
  case 常量 1:
    语句 1;
    break;
  case 常量 2:
    语句 2;
    break;
  …
  default:
```

```
    语句 n;
}
```

其中，switch、case、default 是关键字，default 子句可以省略。开关语句先计算条件的值，然后将条件的值与各个常量比较，如果条件值与某个常量相等，就执行该常量后面的语句；如果都不相等，就执行 default 下面的语句。如果无 default 子句，就什么都不执行，直接跳出开关语句。

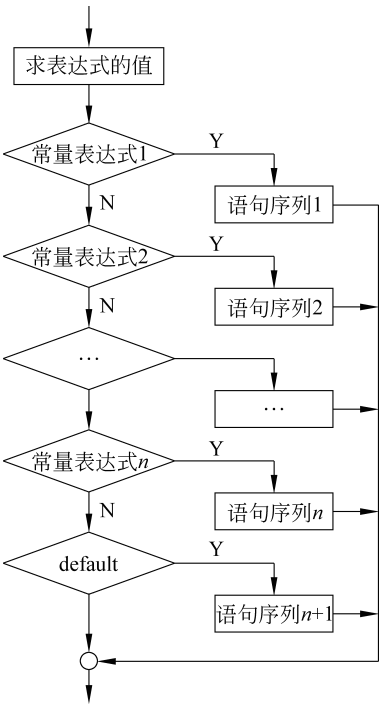

图 3.5　switch 语句执行结果

使用开关语句时，注意以下几点：

(1) case 后面的常量必须是整数或字符型，即 int、short、byte 和 char 类型，而且不能有相同的值。

(2) switch 语句可以拥有多个 case 语句。每个 case 后面跟一个要比较的值和冒号。

(3) 通常在每一个 case 中都通过 break 语句提供一个出口，使流程跳出开关语句。否则，在第一个满足条件 case 后面的所有语句都会被执行，这种情况叫作落空。

(4) switch 语句可以包含一个 default 分支，该分支必须是 switch 语句的最后一个分支。default 在没有 case 语句的值和变量值相等的时候执行。default 分支不需要 break 语句。

switch 语句执行结果如图 3.5 所示。

例 3.6　switch 语句的使用，有 break 语句。本程序当温度变量 c 小于 10℃时，显示“有点冷”；c 小于 25℃时，显示“正合适”；c 小于 35℃时，显示“有点热”；c 大于或等于 35℃时，显示“太热了”。下面的源程序输出结果为“28℃ 有点热”。

```
public class exp3_6 {
  public static void main(String args[]) {
    int c=28;
    switch(c<10? 1:c<25? 2:c<35? 3:4) {
      case 1:
        System.out.println(" "+c+"℃ 有点冷");
        break;
      case 2:
        System.out.println(" "+c+"℃ 正合适");
        break;
      case 3:
        System.out.println(" "+c+"℃ 有点热");
        break;
      default:
```

```
            System.out.println(" "+c+"℃ 太热了");
        }
    }
}
```

例 3.7 switch 语句的使用，无 break 语句。输出结果如图 3.6 所示。

```
public class exp3_7 {
    public static void main(String args[]) {
        int c=28;
        switch(c<10? 1:c<25? 2:c<35? 3:4) {
            case 1:
                System.out.println(" "+c+"℃ 有点冷");
            case 2:
                System.out.println(" "+c+"℃ 正合适");
            case 3:
                System.out.println(" "+c+"℃ 有点热");
            default:
                System.out.println(" "+c+"℃ 太热了");
        }
    }
}
```

说明：

通过这两个例子可以看出 break 语句的作用。例 3.7 由于缺少 break 语句，使得程序执行完 case 3 下面的语句，紧接着又执行了 default 下面的语句。

```
28℃有点热
28℃太热了
```

图 3.6 无 break 语句的结果

因为 case 后面的常量必须是整数或字符型，这两个程序采用了转换方法，将判断条件的取值最终转换为数值，请读者自行分析三元运算符(c＜10?1:c＜25?2:c＜35?3:4)的作用。

3.2.6 在 switch 语句中应用枚举类型

例 3.8 switch 语句与枚举操作的使用。输出结果如图 3.7 所示。

```
public class exp3_8 {
    public enum Season {SPRING, SUMMER, AUTUMN, WINTER;}

    public static void viewSeason(Season season) {
        switch(season) {
        case SPRING:
            System.out.println("[春天,枚举常量:"+season.name()+"]");
            break;
        case SUMMER:
            System.out.println("[夏天,枚举常量:"+season.name()+"]");
            break;
        case AUTUMN:
```

```
            System.out.println("[秋天,枚举常量:"+season.name()+"]");
            break;
        case WINTER:
            System.out.println("[冬天,枚举常量:"+season.name()+"]");
            break;
        }
    }

    public static void main(String[] arg) {
        viewSeason(Season.SPRING);
        viewSeason(Season.SUMMER);
    }
}
```

```
[春天, 枚举常量:SPRING]
[夏天, 枚举常量:SUMMER]
```

图 3.7　switch 语句与枚举操作的使用

说明：

(1) 从本例可以看出，在类 ViewSeason 中也可以声明枚举类型 Season。

(2) 从本例可以看出，在 switch 语句中应用枚举简洁又方便。

(3) 通过“枚举名.常量名”方式可以调用枚举中的常量，例如，本例 Season.SPRING 调用 Season 枚举类中的静态常量 SPRING。

3.3　循环语句

到目前为止，我们看到的都是线性的程序，即每行命令只有一次执行的机会(选择语句则会忽略若干行)。能否重复执行一些语句呢？利用循环语句可以解决这个问题。循环可使程序根据一定的条件重复执行某一部分程序语句，直到满足终止循环条件为止。

本节主要介绍 Java 的 3 种循环语句：确定次数循环(for)、先判定后执行循环(while)和先执行后判定循环(do…while)的使用方法。

3.3.1　确定次数循环语句(for 循环)

如果希望程序的一部分内容按固定的次数重复执行，可使用 for 循环。for 循环采用一个计数器控制循环次数，每循环一次计数器就加 1，直到完成给定的循环次数为止。

例 3.9　该程序利用 for 循环语句为一维数组中的每个元素赋值，然后按逆序输出，结果如图 3.8 所示。

```
a[4] = 4
a[3] = 3
a[2] = 2
a[1] = 1
a[0] = 0
```

图 3.8　逆序输出数组

```
public class exp3_9 {
  public static void main(String args[]) {
    int i;
    int a[]=new int[5];
      for (i=0; i<5; i++)  a[i]=i;
     for (i=a.length-1; i>=0; i--)  System.out.println("a["+i+"]="+a[i]);
```

```
  }
}
```

例 3.10 按 5 度的增量打印出一个从摄氏度到华氏度的转换表，输出结果如图 3.9 所示。

```
class exp3_10 {
  public static void main (String args[]) {
    int fahr,cels;
    System.out.println("摄氏度   华氏度");
    for (cels=0; cels<=40; cels+=5) {
      fahr=cels * 9/5+32;
      System.out.println("  "+cels+"  "+fahr);
    }
  }
}
```

说明：

从上面的例子中归纳出 for 循环的语法格式为：

```
for(初始化语句；循环条件；迭代语句){
    语句序列
}
```

(1) 初始化语句用于初始化循环体变量；循环条件用于判断是否继续执行循环体，其值是 boolean 型的表达式，即结果只能是 true 或 false；迭代语句用于改变循环条件的语句；语句序列称为循环体，当循环条件的结果为 true 时，循环体将重复执行。

(2) for 循环语句的流程首先执行初始化语句，然后判断循环条件，当循环条件为 true 时，就执行一次循环体，最后执行迭代语句，改变循环变量的值。这样就结束了一轮循环。接下来进行下一次循环(不包括初始化语句)，直到循环条件的值为 false 时才结束循环。for 循环语句执行过程如图 3.10 所示。

```
摄氏度 华氏度
  0      32
  5      41
  10      50
  15      59
  20      68
  25      77
  30      86
  35      95
  40      104
```

图 3.9 摄氏度到华氏度的转换表

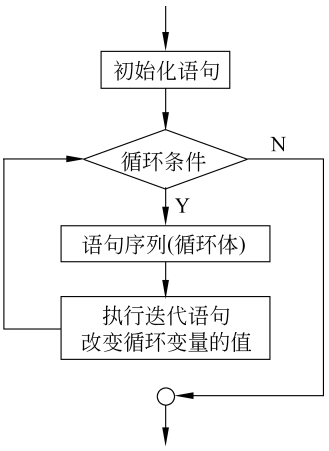

图 3.10 for 循环语句执行过程

(3) 初始化语句中的变量初始化可在 for 语句之前定义,也可在循环括号中定义。迭代语句常写成增量运算的形式,以加快运算速度。根据需要,增量可以是 1,也可以大于 1。增量计算也可以放在循环体中进行,即把迭代语句移到循环体内的适当位置,原位置为空。

(4) 使用循环语句时常常会遇到死循环的情况,就是无限制地循环下去。所以在使用 for 循环时,要注意初值、终值和增量的搭配。终值大于初值时,增量应为正值,终值小于初值时,增量应为负值。编程时必须密切关注计数器的改变,这是实现正常循环避免陷入死循环的关键。例如下面的 for 循环:

```
for(int i=0;i>=0;i++){
    System.out.println("value of i :"+i)
}
```

上述 for 循环,由于初始化“i=0”,满足循环条件,当循环每执行一次“i++”,i 就会增加 1,永远满足循环条件,因此循环永远不会终止,程序陷入死循环。

3.3.2 foreach 循环语句

foreach 语句为数组或对象集合中的每个元素重复一个嵌入语句组。foreach 语句是 for 语句的特殊简化版本,任何的 foreach 语句都可以改写为 for 语句版本,但是 foreach 语句并不能完全取代 for 语句。

例 3.11 这个程序将通过 for 语句和 foreach 语句实现数组的遍历,输出结果如图 3.11 所示。

```
使用for 循环数组
1 2 3 4
使用foreach 循环数组
1 2 3 4
```

图 3.11 for 与 foreach 语句遍历数组

```
public class exp3_11 {
    public static void main(String[] args) {
        int[] intary={1,2,3,4};  //声明数组 intary
        forDisplay(intary);
        foreachDisplay(intary);
    }
    public static void forDisplay(int[] a){
        System.out.println("使用 for 循环数组");
        for(int i=0; i<a.length; i++) {
            System.out.print(a[i]+" ");
        }
        System.out.println();
    }
    public static void foreachDisplay(int[] data){
        System.out.println("使用 foreach 循环数组");
        for(int a  : data) {
            System.out.print(a+" ");
        }
    }
}
```

说明：

从上面的例子中可以看出：

（1）for 循环在遍历数组时使用下标来定位数组中的元素，for 循环执行的次数是在执行前确定的。

（2）foreach 循环在遍历数组时不考虑循环次数。

（3）在固定循环次数或循环次数不需要计算时 for 循环效率高于 foreach，而在不确定循环次数时用 foreach 比较方便。

3.3.3 先判定后执行循环语句（while 循环）

while 循环不像 for 循环那么复杂，while 循环只要定义一个条件判断语句，便可以进行循环操作，看下面的例子。

例 3.12 这个程序将通过 while 语句按顺序输出数字 1～5，并计算 5 个数字的和。输出结果如图 3.12 所示。

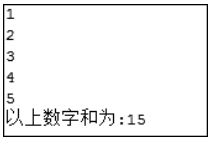

图 3.12 交互结果

```
public class exp3_12 {
    public static void main(String[] args) {
        int i=1;                //代表 1~5 的数字
        int sum=0;
        while(i<=5) {           //当变量小于或等于 5 时执行循环
            //输出变量的值,并且对变量加 1,以便于进行下次循环条件判断
            System.out.println(i);
            sum=sum+i;
            i++;
        }
        System.out.println("以上数字和为:"+sum);
    }
}
```

说明：

（1）从本例可以看出，while 循环从判断 i 的值开始，作为循环条件，花括号内的语句为循环体语句，当循环条件成立时，则重复执行循环体中的代码。

（2）循环体内部的语句会先输出此时的变量 i 的值，再将 i 的值加入变量 sum 中，并将 i 的值加 1，只要 i 小于或等于 5 就继续执行该循环。

（3）从上面的例子归纳出 while 循环的语法格式为：

```
while(条件) {
    循环体
}
```

其中，while 是关键字。每次循环之前都要计算条件的值，条件的值必须是 boolean 型的，其结果只能是 true 或 false，当条件值为 true 时，执行一次循环体中的语句，然后再计算条件，决定是否再次执行循环体中的语句；如果条件的值为 false 时，跳出循环体，执行循

环体下面的语句。while 语句执行过程如图 3.13 所示。

注意：while 循环中的条件是逻辑表达式或关系表达式，所以循环体中一定要有改变条件的语句，使条件的值变为 false，否则会陷入死循环。

例如下面的 while 循环：

```
int i=0;
while(i>=0){
    System.out.println("value of i :"+i)
}
```

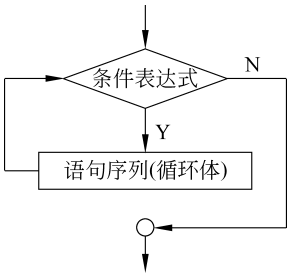

图 3.13　while 语句执行过程

上述 while 循环，由于初始化 i=0，永远满足条件 i>=0，因此循环永远不会终止，运行后程序会一直输出“value of i :0”，程序陷入死循环。

3.3.4　先执行后判定循环语句（do…while 循环）

do…while 循环与 while 循环相反，被称为后测试循环语句，其是先执行循环体中的语句，再计算 while 后面的条件，若条件值为 false 则跳出循环，否则继续下一轮循环。

什么时候使用 do…while 循环呢？有些情况下，不管条件的值是 true 还是 false，都希望把指定的语句至少执行一次，那么就应使用 do…while 循环，看下面的例子。

例 3.13　求 1＋2＋…＋100 的值，输出结果为“1＋2＋…＋100＝5050”，源程序代码如下所示：

```
class exp3_13 {
  public static void main(String args[]) {
    int n=1;   int sum=0;
    do sum+=n++;
    while (n<=100);
    System.out.println("1+2+…+100="+sum);
  }
}
```

说明：

（1）归纳 do…while 循环的语法格式为：

```
do {循环体}
while (条件);
```

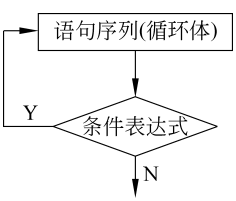

图 3.14　do…while 语句执行过程

其中，do、while 是关键字。程序首先执行 do 下面的循环体，然后计算 while 后面条件的值，如果其值为 true，则重复执行循环体，否则，结束循环。do…while 语句执行过程如图 3.14 所示。

（2）与 while 循环相同，do 循环在循环体中也一定要有改变条件值为 false 的语句，否则会陷入死循环。

例如下面的 do…while 循环：

```
int i=0;
do{
    System.out.println("value of i :"+i);
}while(i>=0);
```

上述 do…while 循环，由于初始化 i＝0，永远满足条件 i＞＝0，因此循环永远不会终止，运行后程序会一直输出“value of i :0”，程序陷入死循环。do…while 循环控制并不是很常用，但有时却非常重要，使用时特别注意不要忘记了 while 语句结尾处的分号(;)。

3.3.5 嵌套使用循环语句

嵌套使用循环语句就是在一个循环体内又包含另一个完整的循环结构，而在这个完整的循环体内还可以嵌套其他的循环结构。例如 while 循环语句嵌套、while 循环语句与 for 循环语句嵌套等，看下面的例子。

例 3.14 分别用 while 循环语句与 for 循环语句嵌套实现累计求和，程序运行结果如图 3.15 所示。

```
public class exp3_14 {
  public static void main(String args[]) {
    int n=10, sum=0;
    while (n>0) {     //while 循环
      sum=0;
      for (int i=1; i<=n; i++)  sum+=i;     //for 循环
      System.out.println("前"+n+"个数的总和为:"+sum);
      n--;            //改变计数器的值
    }
  }
}
```

说明：程序使用了 while 循环语句与 for 循环语句嵌套，对 n 和 sum 初始化并赋值，while 循环判断 n＞0 值为 true，执行 while 内嵌套的 for 循环语句，向屏幕输出前 n 个数的总和值。

```
前10个数的总和为: 55
前9个数的总和为: 45
前8个数的总和为: 36
前7个数的总和为: 28
前6个数的总和为: 21
前5个数的总和为: 15
前4个数的总和为: 10
前3个数的总和为: 6
前2个数的总和为: 3
前1个数的总和为: 1
```

图 3.15 累计求和结果

3.3.6 循环语句小结

一个循环一般应包括以下 4 部分内容。

(1) 初始化部分：用来设置循环的一些初始条件，计数器清零等。

(2) 循环体部分：这是反复被执行的一段代码，可以是单语句，也可以是复合语句。

(3) 迭代部分：这是当前循环结束，下一次循环开始时执行的语句，常用来使计数器加 1 或减 1。

(4) 终止部分：通常是描述条件的逻辑表达式或关系表达式，每一次循环要对该表达式求值，以验证是否满足循环终止条件。

3.4 跳转语句

跳转语句可以无条件改变程序的执行顺序。

本节的内容主要介绍 Java 的 3 种跳转语句：break 语句、continue 语句和 return 语句的使用方法。

3.4.1 break 语句

break 语句提供了一种方便地跳出循环的方法，用其可以立即终止循环，跳出循环体。它在 for、while 或 do…while 循环中，有强制终止循环的作用，在 switch 语句跳出该语句，继续执行其他语句。

例 3.15 break 语句的使用。输出结果如图 3.16 所示。

```
class exp3_15{
  public static void main(String args[]) {
    boolean test=true;  int i=0;
    while (test) {
      i=i+2;
      System.out.println("i="+i);
      if (i>=10)  break;
    }
    System.out.println("i 为"+i+"时循环结束");
  }
}
```

```
i=2
i=4
i=6
i=8
i=10
i为10时循环结束
```

图 3.16 break 语句的使用

说明：执行这个程序时，尽管 while 条件的值始终为真，但实际上只循环了 5 次。这是因为当 i 大于或等于 10 时遇到了 break 语句，使流程跳出了循环。

3.4.2 continue 语句

continue 语句只能用在循环结构中，它跳过循环体中尚未执行的语句，重新开始下一轮循环，从循环体第一个语句开始执行。在 for 循环中，continue 语句使程序立即跳转到更新语句。在 while 或者 do…while 循环中，程序立即跳转到布尔表达式的判断语句。下面通过一个例子来说明它的使用方法。

例 3.16 下面的程序可以输出 1～9 中除 6 以外所有偶数的平方值，输出结果如图 3.17 所示。

```
class exp3_16 {
  public static void main(String args[]) {
    for (int i=2;i<=9;i+=2) {
      if (i==6)  continue;
```

```
        System.out.println(i+"的平方="+i * i); }
    }
}
```

```
2 的平方= 4
4 的平方= 16
8 的平方= 64
```

图 3.17　使用 continue 语句

说明：程序采用 for 循环，int 数据类型 i 从 2 开始进行 for 循环，每次计数器加 2，当 i 值为 6 时执行 continue 语句，跳过“System.out.println(i＋" 的平方＝"＋i * i);”语句，重新开始下一轮循环。

3.4.3　带标号的 continue 语句

Java 也支持带标号的 continue 语句，它通常用在嵌套循环的内循环中，可以用标号标出想跳到的那条语句继续重复执行程序。它的语法格式为：

标识符：
…
continue 标识符;

例 3.17　求 100～200 的所有素数。该例通过一个嵌套的 for 循环来实现，输出结果如图 3.18 所示。

```
public class exp3_17 {
  public static void main(String args[]) {
    System.out.println(" **100～200 的所有素数**");
    int n=0;
    outer:
    for (int i=101; i<200; i+=2) {
      int k=15;
      for (int j=2; j<=k; j++)
        if (i%j==0) continue outer;
      System.out.print("  "+i);
      n++;
      if (n<10)  continue;
      System.out.println();
      n=0;
    }
    System.out.println();
  }
}
```

```
** 100～200的所有素数 **
 101  103  107  109  113  127  131  137  139  149
 151  157  163  167  173  179  181  191  193  197
 199
```

图 3.18　100～200 的所有素数

说明：程序中分别使用了 continue 语句和带标号的 continue 语句。在输出方法上分别使用了系统的 println 方法和 print 方法，二者的区别在于前者输出时换行，后者输出时不换行。

3.4.4 return 语句

return 语句用来返回方法的值，其通常位于一个方法体的最后一行，当程序执行到 return 语句时，退出该方法并返回一个值，继续执行调用这个方法语句的下一条语句。

当方法用 void 声明时，说明不要返回值(即返回类型为空)，return 语句省略。

return 语句格式为：

return 表达式;

当程序执行 return 语句时，先计算“表达式”的值，然后将表达式的值返回到调用该方法的语句。返回值的数据类型必须与方法声明数值类型一致，可使用强制类型转换使类型一致。

例 3.18 使方法具有返回值的 return 语句，输出结果如图 3.19 所示。

```
class exp3_18 {
  final static double PI=3.14159;
  public static void main(String args[]) {
    double r1=8.0, r2=5.0;
    System.out.println("半径为"+r1+"的圆面积＝"+area(r1));
    System.out.println("半径为"+r2+"的圆面积＝"+area(r2));
  }
  static double area(double r) {
    return (PI * r * r);
  }
}
```

```
半径为8.0的圆面积＝201.06176
半径为5.0的圆面积＝78.53975
```

图 3.19 带返回值的方法

说明：程序中定义了一个 double 数据类型具有返回值的 area 方法，通过 return 语句给 area 返回值(PI * r * r)。

在 main 方法中，调用了 area 方法，输出了圆面积的数值。area 方法的功能就像函数一样，通过 area 方法的参数指定一个半径的值，area 方法得到一个面积值。

3.5 知识拓展——注解

注解是 JDK 1.5 及以后版本引入的一个特性，与类、接口、枚举在同一个层次。它可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明，注释。

本节主要介绍 Java 提供的内置注解、元数据注解与自定义注解。

3.5.1 注解概述

注解(Annotation)也叫元数据,是一种代码级别的说明,其实就是代码里的特殊标记,这些标记可以在编译、类加载、运行时被读取,并执行相应的处理,通过使用注解,可以在不改变原有程序逻辑的情况下,在源文件中嵌入一些补充信息,通过反射机制编程可以实现对这些元数据(用来描述数据的数据)的访问(第5章介绍反射机制)。

java.lang.annotation.Annotation是注解的接口,所有的注解都默认实现了此接口(接口的具体内容在第5章介绍)。注解不会直接影响程序的语义,只是作为注解(标识)存在,另外,可以在编译时选择代码里的注解是否只存在于源代码级,或者在class文件、运行时中出现(SOURCE/CLASS/RUNTIME)。

使用注解时要在其前面增加@符号,将注解当成一个修饰符使用,用于修饰它支持的程序元素,在代码中的使用方式为"@注解名"。

根据注解参数的个数,可以将注解分为标记注解、单值注解、完整注解三类。根据注解的作用,可以将注解分为内置注解、元数据注解(或称元注解)与自定义注解三类。

3.5.2 内置注解与元注解

1. 内置注解

在JDK 5.0之后,Java内置了三种注解,可以在Java程序中直接使用,将其当成一个修饰符来使用,用于修饰它支持的程序元素。

(1) @Override用于限定重写父类方法,该注解只能用于方法。

(2) @Deprecated用于表示某个程序元素(类,方法等)已过时。如果在其他程序中使用了此元素,则在编译时将出现警告信息。

(3) @SuppressWarnings用于压制警告信息的出现。

2. 元注解

元注解的作用是用于修饰其他注解定义。JDK 5.0定义了4个标准的meta-annotation元注解类型。

(1) @Target。用于描述注解的使用范围,表示该注解用于什么地方,未标注则表示该注解可用于所有的地方,可能范围值在枚举类ElemenetType中,包括:

ElemenetType.CONSTRUCTOR:构造器。

ElemenetType.FIELD:域(包括enum实例)。

ElemenetType.LOCAL_VARIABLE:局部变量。

ElemenetType.METHOD:方法。

ElemenetType.PACKAGE:包。

ElemenetType.PARAMETER:参数。

ElemenetType.TYPE:类,接口(包括注解类型)或enum声明。

(2) @Retention。用于描述注解在什么级别保存其信息,即注解的生命周期。可选

的参数值在枚举类型 RetentionPolicy 中，包括：

RetentionPolicy.SOURCE：注解直接被编译器丢弃。

RetentionPolicy.CLASS：编译器把注解记录在 class 文件中。当运行 Java 程序时，JVM 不会保留注解。这是默认值。

RetentionPolicy.RUNTIME：编译器把注解记录在 class 文件中。当运行 Java 程序时，JVM 会保留注解。程序可以通过反射获取该注解。

（3）@Documented。用于描述修饰的注解将被 javadoc 工具提取成文档。此注解不常用，了解即可。

（4）@Inherited。用于描述允许子类继承父类中的注解。如果一个使用了@Inherited 修饰的 annotation 类型被用于一个类，则这个 annotation 将被用于该类的子类。此注解不常用，了解即可。

例 3.19 本例说明如何在类中使用内置注解。

```
public class exp3_19 {
    public String name;
    public int age;
    @Deprecated        //表示不建议使用 prin()方法,调用时会出现删除线
    public void prin(String name) {
          System.out.println(name);
    }

    @Override    //限定重写父类 Object 的 toString()方法,否则会报编译错误
    public String toString() {
        return "Person [name="+name+", age="+age+"]";
    }
    public static void main(String[] args) {
        exp3_19 person=new exp3_19 ();
        person.prin("");     //因 pin()方法加有@Deprecated 注解,故方法名上有横线,
                             //表明此方法已过时,不建议使用
    @SuppressWarnings("unused")
    int i;     //因为变量 i 在程序中没有被使用,因此处有注解,不会有警告
    int j;     //变量 j 在程序中没有被使用,但没有使用@SuppressWarnings("unused")
               //注解,此处会有警告信息
    }
}
```

说明：本例旨在说明如何在类中使用内置注解。

（1）使用@Override 注解，覆盖的方法一定要是其父类的方法，否则在 Eclipse 中会出现编译错误。

（2）调用由@Deprecated 注解的方法时，在 Eclipse 中会出现 `person.prin("");` 警示提示。

（3）变量 j 在程序中没有被使用，又没有使用@SuppressWarnings("unused")注解，