第3章　流程控制

程序的基本结构包括顺序、分支和循环。本章介绍如何利用 Java 语言将人的思维用计算机程序来体现，从而解决现实问题。

学习目标

- 理解程序控制的概念。
- 运用条件语句，使用 if，switch 来控制程序的不同执行路径。
- 掌握 for、while 和 do 句型结构控制程序的循环执行。
- 理解分支和循环的影响语句范围。
- 理解并能应用 break 和 continue 调整程序中的流程控制结构。
- 理解变量的生存范围。
- 理解并使用断言对程序进行调试。
- 掌握利用 Scanner 类实现基本的输入方法。
- 理解并能使用 System.out 对象中的不同输出方法。

3.1　句、块和空白

在第 1 章讨论程序结构的时候，提到构成 Java 程序的最小单位是类，而类则是由属性和方法构成。方法代表了某个具体的功能，一个方法就是为完成某个功能的若干条语句的集合，如 System.out.println()中的 println()就是 JDK 提供的一个供输出信息的方法。

3.1.1　语句

语句就和自然语言一样，告诉计算机要做什么。在 Java 编程语言中，语句是一条由分号“;”终止的代码，它是一个完整的可执行单元。例如，下面是一条赋值语句，将等号右边算术表达式的结果赋给左边的变量 total。

```
total=a+b+c+d+e+f;
```

有时候，由于编辑区宽度的限制以及为了提高程序的可阅读性，一条语句可能会分行书写，这并不影响语句的完整性，例如下面的语句和上述语句的功能完全相同。

```
total=a+b+c+
      d+e+f;
```

具体来说，在 Java 中，主要有以下的语句类型。

1. 声明语句

在第 2 章中，介绍了有关变量的声明，附加上一个表示结束的分号之后，就成了声明语句，如：

```
int a=0;
```

变量声明加了一个分号就构成了一条声明语句，程序执行到此时，将会在内存中分配 4 个字节用于存储赋给变量 a 的值。

```
Student stu;
```

这是声明了一个对象变量，其类型是 Student。

2. 表达式语句

赋值表达式、自增表达式、方法调用、对象创建都可以和分号一起构成“表达式语句”，如：

```
System.out.println("Welcome");
```

这是调用了 out 对象的方法 println()，这是一种方法调用的形式，通常用于没有返回的方法调用。

```
a=Math.abs(-3.1);
```

这是调用了 Math 类的求绝对值方法 abs()，这也是一种方法调用的形式，通常用于有返回值的方法调用，以便能够得到方法执行后的结果。

```
value=100;
```

赋值表达式加上分号，就构成了赋值语句。

```
a++;
```

自增表达式加上分号，就构成了自增语句。

3. 空语句

空语句就是仅包含一个分号的语句，没有任何实际作用，但它是一条可执行语句。

4. 控制语句

控制语句主要负责语句的执行顺序和方向，例如循环、分支、跳转等，在随后的流程控制中将详细介绍。

3.1.2 语句块

一个语句块(block)是以“{”和“}”为边界的语句集合，有时也称作复合语句。块语句

也被用来组合属于某个类的语句。

语句块可被嵌套。HelloWorldApp 类包含了一个 main 方法，这个方法就是一个语句块，它是一个独立单元。其他一些块语句或组的例子如下：

```
//a block语句
{
    x=y+1;
    y=x+1;
}
//类声明所包含的块语句
public class MyDate{
    int day;
    int month;
    int year;
}
//一个嵌套块语句的例子
while(i<large){                          //循环语句块开始,用"{"表示
    a=a+1;-
    if(a==max ) {                        //判断语句块开始,用"{"表示
        b=b+a;
        a=0;
    }  //判断语句块结束,用"}"表示
}  //循环语句块结束,用"}"表示
```

3.1.3 空白

在源代码元素之间允许空白，空白的数量不限。空白(包括空格、tabs 和新行)可以改善阅读源代码时的视觉感受，特别是利用空白明确显示出不同的嵌套结构，便于阅读和理解。如下面的代码段所示。

```
public class ComputeArea {
    public static void main(String[] args) {
        int r=10;
        ;
    }
}
```

程序中方法声明、方法中的语句等都和包容它们的上层结构向后推了几个空格，以保证这种逻辑上的嵌套关系呈现更好的视觉效果，便于理解。

最后关于语句需要强调的是：语句必须有一个存在范围。例如，循环语句必须由循环控制语句完全控制，一个方法中的语句必须放置在方法体的一对括号内，而不能出现在括号外。

3.2　顺序结构

一个程序(方法)基本的执行过程就是从前到后进行的。构成 Java 程序的是类,而类的最小功能单元是方法,而方法是有一条或多条语句构成的,这些语句间最简单的结构关系是顺序结构,也就是语句是按照它们在方法中出现的先后顺序逐一执行的。例如,程序 3-1 描述了用程序实现交换两个变量值的过程。

```
/**
 * 程序 3-1:交换两个变量的值
 */
(1)  public class Swap {
(2)      public static void main(String[] args) {
(3)          int a=10,b=20;
(4)          int t=0;
(5)          t=a;
(6)          a=b;
(7)          b=t;
(8)          System.out.println("a="+a);
(9)          System.out.println("b="+b);
(10)     }
(11) }
```

根据变量的定义,在任何一个时刻,每个变量只能保存一个值,当把一个新的值赋给变量时,变量原来表示的值就没有了,因此,两个变量交换无法做到直接相互交换,不过可以借助一个临时的中间变量来实现,具体步骤如图 3-1 所示。

第(3)行:程序用声明语句定义了两个变量 a 和 b。

第(4)行:声明了一个变量 t,用做临时保存。

第(5)行:因为无法直接把 b 的值赋给 a,因为这样 a 的值就被覆盖了,所以在将 b 的值赋给 a 之前,先将 a 的值临时保存到变量 t。

第(6)行:因为执行第 5 行语句时,将 a 的值已经保存到 t 中,所以,可以放心地将 b 的值赋给 a,执行后,a 和 b 表示同样的值。

第(7)行:因为执行第 6 行语句时,b 的值已经赋给 a,所以将 t 保存的原来 a 的值再赋给 b,执行后,b 就获得了原来 a 的值,这样就实现了两个变量相互交换各自的值。

通过图 3-1 可以看出,这个交换程序按照语句出现的先后顺序逐一执行,自然地完成了两个变量的交互。

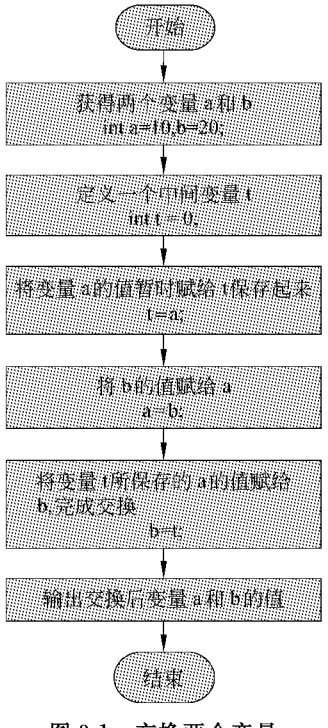

图 3-1　交换两个变量

注：图 3-1 中的图被称为“程序流程图”，其中的方框代表一条或一组语句；带箭头的直线表示程序的执行流向，流线的标准流向是从左到右和从上到下，沿标准流向的流线可不用箭头指示流向，但沿非标准流向的流线应用箭头指示流向；最后的扁圆形表示程序执行的结束。

3.3 选择结构

程序控制可以定义为对程序语句的执行顺序进行的规定。程序执行结构并不总是由顺序结构构成的，就如同早上出门时总是根据是否下雨来决定带不带雨伞一样，Java 也提供了使用分支语句在两种或更多的情况中做出选择，根据结果执行不同的程序语句。

3.3.1 if…else 语句

if…else 语句的基本句法如下：

```
if(布尔表达式){
    语句或块;}
else  {
    语句或块;
      }
```

- 在 Java 编程语言中，if()用的是一个布尔表达式，而不是数字值，这一点与 C/C++ 不同，当布尔表达式为真时，执行其后的语句或语句块，为假时，根据是否有匹配的 else 决定如何执行。
- 另外，分支中的 else 部分是选择性的，当测试条件为假时如不需做任何事，else 部分可被省略，另外，else 部分不能独立存在，必须和 if 匹配。
- if 语句和 else 影响的语句可以是一条语句或多条语句，如果是多条语句，则这些语句必须放在随后紧跟的大括号定义的复合语句内。
- if 语句有三种形式：单分支、双分支和多分支选择语句。

1. 单分支语句

单分支语句是最简单的一种情况，如图 3-2 所示，其一般形式为：

```
if(布尔表达式)
语句;
```

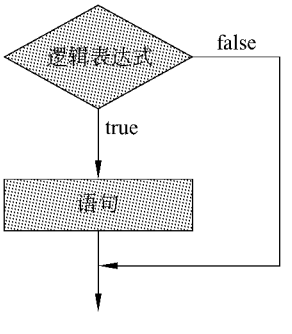

图 3-2 单分支选择语句

这种形式描述了当表达式成立时，执行随后的一条语句，如果需要执行的是多条语句，则这些语句应当放在大括号中，构成复合语句，类似下面这样。

```
if(布尔表达式){
```

```
    语句 1;
    ⋮
    语句 n;
}
```

无论哪种情况，当表达式不成立时，则不再执行分支控制的语句，而直接分支语句后续的语句，继续程序的执行。

程序 3-2 描述了一个求绝对值的例子。

```
//程序 3-2：单分支结构
public class Abs {
    public static void main(String[] args) {
①       int a=-10;
②       if(a<0)
③           a=-a;
④       System.out.println("a="+a);
    }
}
```

由于变量 a 的值是负值，程序 3-2 的执行过程是①－＞②－＞③－＞④，其中③是 if 语句的控制范围，假如 a 的值是大于 0 的，则 if 的逻辑表达式为 false，条件不满足则不会执行③这条语句，则程序 3-2 的执行过程是①－＞②－＞④。

注：为了更好地表现分支语句的影响范围，即使只影响一条语句，最好的编码风格是也要用大括号把它们括起来，这样不仅可以提供程序的可阅读性，而且会避免程序出现修改上的错误。

2. 双分支语句

双分支语句体现了非此即彼的情况，如图 3-3 所示，如果逻辑表达式为真，则执行语句块 1，否则执行语句块 2(注：语句块可以只有一条语句)。

双分支语句的一般形式如下：

```
if(逻辑表达式)
    语句 1;
else
    语句 2;
```

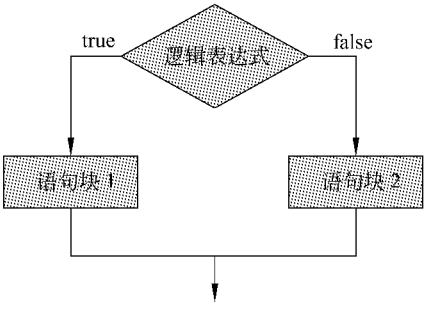

图 3-3　双分支 if 语句

这种形式的分支，如果表达式结果为真，则执行语句 1，否则执行语句 2，随后执行分支后面的其他语句。

如果分支需要影响的语句有多条，需要将多条语句用大括号括起来，构成复合语句，如下所示：

```
if(逻辑表达式){
    语句块 1;
```

```
}else{
    语句块 2;
}
```

程序 3-3 描述了求两个整数之间最大值的算法。

```
// 程序 3-3: 双分支结构
public class Max{
    public static void main(String[] args) {
①       int a=10, b=20;                    //准备两个变量,进行比较大小
②       int max=0;                         //定义一个变量,保存两个变量的最大值
③       if(a>b){
④           max=a;
⑤       }
⑥       else{
⑦           max=b;
⑧       }
⑨       System.out.println("max="+max);
    }
}
```

由于程序 3-3 中 a 小于 b,所以其执行路径是①→②→③→⑥→⑦→⑧→⑨,如果修改成 a 大于 b,则程序执行路径是①→②→③→④→⑤→⑨。

3. 多分支语句

采用 if…else if 形式以实现在多种情况下选择一种情况执行,它的执行逻辑如图 3-4 所示。一般的语法形式为:

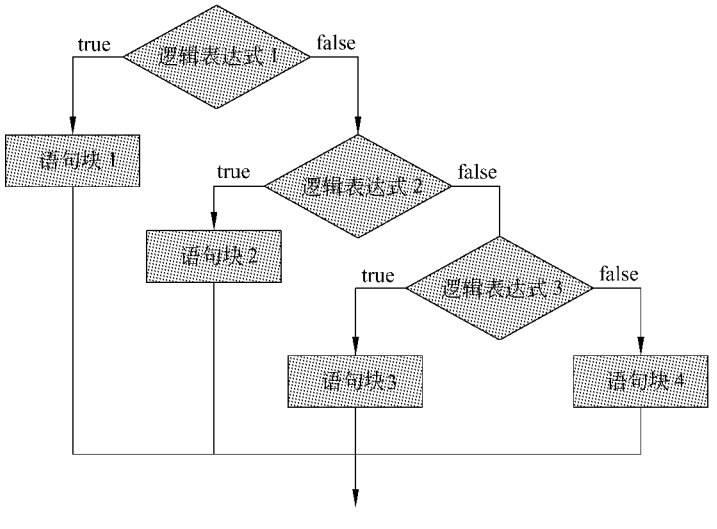

图 3-4 if…else if 多分支语句

```
if(逻辑表达式 1){
    语句块 1;
}else if(逻辑表达式 2){
    语句块 2;
} else if(逻辑表达式 3){
    语句块 3;
}
⋮
else{
    语句块 n;
}
```

程序碰到多分支语句时，将依次执行判断，如果碰到第一个表达式为真的情况，则执行对应的语句，然后跳至整个多分支语句之外继续执行后续程序；如果没有一个表达式为真，但存在一个 else 分支，则执行此分支对应的语句，否则，什么都不执行，直接跳至多分支语句后继续执行。

程序 3-4 描述一个应用多分支语句将百分制成绩转换为五级制的算法。

```
//程序 3-4：多分支结构
public class Score{
    public static void main(String[] args){
①       int score=85;
②       String level=null;
③       if(score>=80 && score<90) {
④           level="良好";
⑤       } else if (score>=70 && score<80) {
⑥           level="中";
⑦       } else if (score>=90){
⑧           level="优秀";
⑨       } else if(score>=60 && score<70){
⑩           level="及格";
⑪       }else{
⑫           level="不及格";
⑬   }
⑭   System.out.println("转换后的成绩是"+level);
    }
}
```

在 score 等于 85 分的情况下，程序的执行流程是①→②→③→④→⑭，如果是90 分，则程序执行流程是①→②→③→⑤→⑦→⑧→⑭。

使用这种多分支语句需要注意，每个 else if 分支都是在前面判断不满足的情况下再次判断的。

3.3.2 switch 语句

switch 语句是另外一种多分支情况选择语句，允许程序员在更多情况下选择执行不同的程序逻辑，switch 语句的语法是：

```
switch(expr){
    case value1:
          statements;
          break;
    case value2:
          statements;
          break;
    default:
         statements;
         break;
}
```

- 在 switch(expr) 语句中，expr 必须与 int 类型是赋值兼容的，允许的类型包括 byte，short 或 char 以及枚举类型；不允许使用结果类型为浮点或 long 等表达式，而且表达式必须放在小括号中。
- switch 的判断语句必须放在大括号中。
- case 后的 value 的类型必须和 expr 的类型应保持一致或兼容，例如都是字符型。
- switch 语句在执行时，从第一个 case 开始匹配是否相等，如相等则执行 case 语句后的语句。
- expr 的值如不能与任何 case 值相匹配时，可选缺省符(default)指出了应该执行的程序代码。default 也是一种情况，作用和其他 case 是一样的。default 并不是必须存在的，可以省略，位置也可以放在 switch 结构中的任何顺序上，但默认位置放在最后。
- break 的作用是终止一个 case 语句的执行。如果没有用 break 语句明确结束，如果还有其他 case 语句，未执行，则程序将继续执行下一个 case，且不检查 case 后的值是否和 switch 的表达式结果匹配。也就是说，当一个粗心的程序员忘了用 break 语句结束一种 case 下的语句执行时，程序就不由自主地顺序执行下一个紧接的 case，直到整个 switch 结束或碰到一个 break 语句。

程序 3-5 用 switch 语句改写了程序 3-4，可以看出，利用 switch 可以使得程序在描述多分支条件下的决策时显得逻辑更清晰。

```
//程序 3-5: switch 多分支结构
public class Switch {
    public static void main(String[] args){
①       int score=80;
②       String level=null;
```

```
③      char c=score>=80 && score<90? 'B'
                :score>=70 && score<80? 'C': score>=90? 'A'
                        :score>=60 && score<70? 'D':'E';
④      switch(c){
⑤          case'B':
⑥              level="良好";
⑦              break;
⑧          case'C':
⑨              level="中等";
⑩              break;
⑪          case'A':
⑫              level="优秀";
⑬              break;
⑭          case'D':
⑮              level="合格";
⑯              break;
⑰          case'E':
⑱              level="不合格";
⑲              break;
⑳      }
㉑      System.out.println("转换后的成绩是"+level);
    }
}
```

由于 switch 表达式类型的限制，程序 3-5 只能利用条件表达式将成绩所在的区间转换为字符才能利用 switch 语句，如第 3 行语句。

当成绩 score 为 80 分时，其执行路径为①→②→③→④→⑤→⑥→⑦→⑳→㉑，当执行到第⑦行，程序碰到了 break 语句，则退出 switch 语句，执行其后的第㉑行语句。当成绩 score 为 90 分时，其执行路径为①→②→③→④→⑤→⑧→⑪→⑫→⑬→⑳→㉑。

可以安排不同的 case 执行相同的语句，当然这需要让这几个 case 顺序排在一起，而且程序员要故意忘记写每个 case 所需的 break 语句，如下面的例子所示：

```
①  char answer='N';
②  switch(answer){
③      case 'Y':
④      case 'y'
⑤          System.out.println("Yes is selected");
⑥          break;
⑦      case 'N':
⑧      case 'n'
⑨          System.out.println("No is selected");
⑩          break;
⑪  }
```