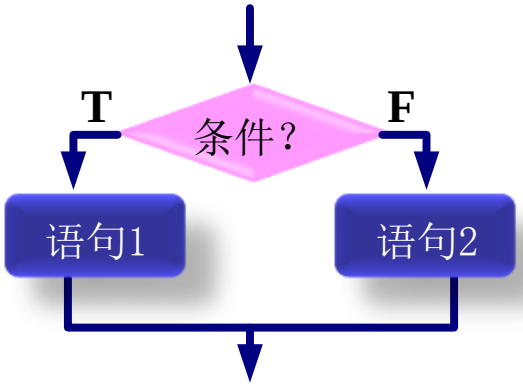
Java软件设计基础

第3章 流程控制

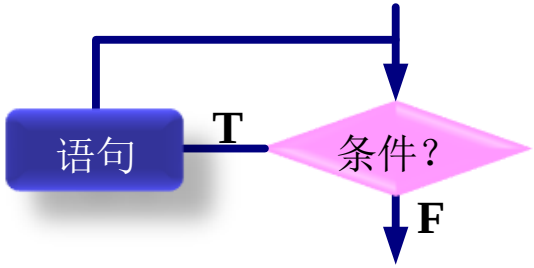
3.1 语句控制结构

- 控制语句（control statement）
  - 通常源文件中的语句按照它们出现的顺序从头到尾执行，但是控制语句通过使用决策、循环和分支来改变执行流程，使程序按照条件执行特定的代码块。控制语句分为以下几类：

- 顺序结构：在程序执行时，根据程序中语句的书写顺序依次执行的命令序列。



- 分支结构：又称为选择结构，是在程序执行时，根据不同的条件，选择执行不同的程序语句，用来解决有选择、有转移等诸多问题，完成应用程序中的智能判断功能。



- 循环结构：使某些语句或程序段按条件重复执行若干次，直至该特定条件不满足为止。循环有以下特点：

  a.只有一个入口和出口；

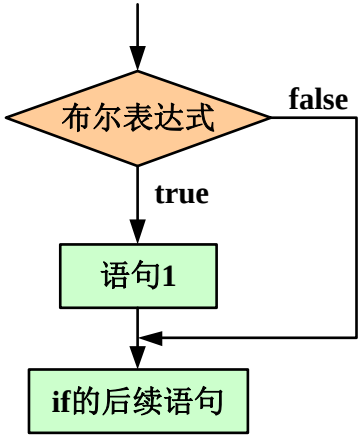
  b.结构内的每部分都有机会被执行；

  c.结构内没有“死循环”，即无终止循环或无限循环。

3.2 分支结构

- if条件语句
  - if语句
    - 是所有控制语句中最基础的语句，只有特定检测结果为true时，它才通知程序执行特定的代码段。
  - if-else语句
    - 当if子句的计算结果为false时，该语句提供第二个执行路径。
  - 说明
    - 条件表达式是任意一个返回布尔型数据的表达式；
    - 简单语句可以省略前面的花括号。

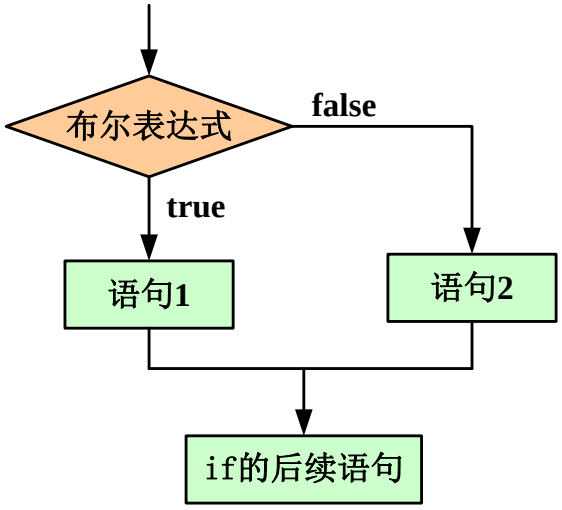
决定什么时候省略括号是个人的习惯，但会导致代码更加脆弱。如果以后在if或else语句后面加入第二个语句，常见的错误是忘记加上必须的括号。编译器不能识别这样的错误，将会得到错误的结果。



if-then

```
if (条件表达式)
{
        语句块;
}
```

首先计算布尔表达式，若为真，则执行语句1，否则就直接转入if语句的后续语句



if-then-else

```
if (条件表达式)
{
        语句块1;
}
else
{
        语句块2;
}
```

首先计算布尔表达式，若为真，则执行语句1，否则执行语句2，然后转入if语句的后续语句

```
import  java.util.*;
public class IfSample{
        public static void main(String args[]){
                int number=new Random().nextInt();
                if(number%2==0)
                        System.out.println(number+"is even.");
                if(number%2==1)
                        System.out.println(number+"is odd.");
        }
}
```

```
import  java.util.*;
public class IfSample{
        public static void main(String args[]){
                int number=new Random().nextInt();
                if(number%2==0)
                        System.out.println(number+"is even.");
                else
                        System.out.println(number+"is odd.");
        }
}
```

```
if(number%2==0);
        System.out.println(number+”is even.”);
```

```
if(number%2==0){ };
        System.out.println(number+”is even.”);
```

- 上述语句不能产生正确的结果。该错误很难发现，因为这并非编译错误或者是运行时错误。这属于逻辑错误。

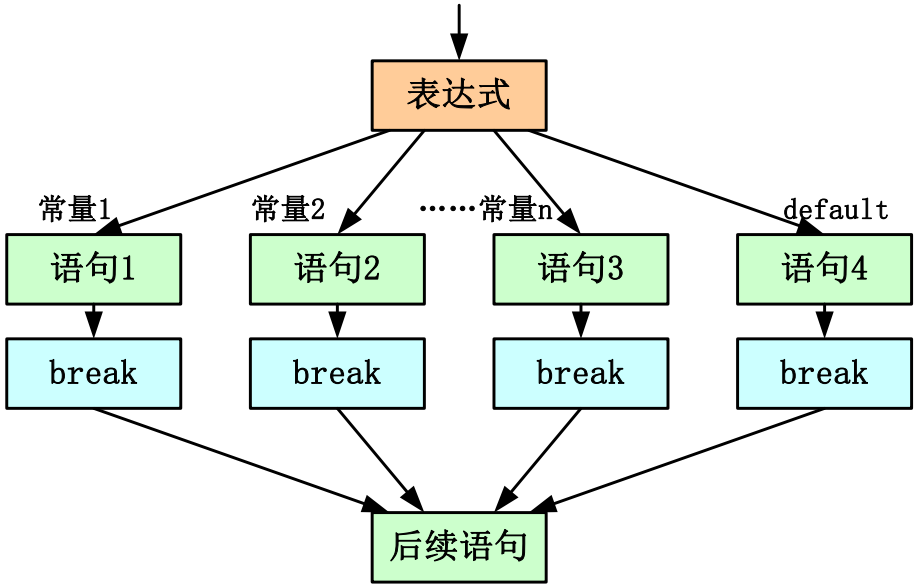
- 嵌套条件语句
  - Java中没有提供象elseif这样的关键字来进行多条件分支。三种选择或以上的分支就需要利用if语句的嵌套来实现。程序从上往下依次判断布尔表达式的条件，一旦某个条件满足，就执行相关语句，然后就不再判断其余的条件，直接转到if语句的后续语句去执行。else总是与离它最近的if语句配对。

嵌套条件语句

```
if (条件表达式1)
        {语句块1;}
else{
        语句块2;
        if(条件表达式2){语句块3;}
        else{
                语句块4;
                if(条件表达式3)
                        {语句块5;}
        }
}
```

```
//例程：一元二次方程求根
import java.util.*;
public class MyIf {
        public static void main(String args[]) {
                Scanner reader=new Scanner(System.in);
                double a,b,c,x1,x2,check;
                System.out.println("输入一元二次方程系数");
                a=reader.nextDouble();
                b=reader.nextDouble();
                c=reader.nextDouble();
                check=b*b-4*a*c;
                if(check==0){
                        x1=(-b)/(2*a);
                        x2=x1;
                        System.out.println("x1="+x1);
                        System.out.println("x2="+x2);}
                else if(check>0){
                        x1=(-b+(float)Math.sqrt(check))/(2*a);
                        x2=(-b-(float)Math.sqrt(check))/(2*a);
                        System.out.println("x1="+x1);
                        System.out.println("x2="+x2);}
                else{System.out.println("该方程无实根");}
    }
}
```

- switch多分支语句
  - 允许使用任意数量的可能执行路径，当分支过多时，比嵌套条件语句更具有直观性、便捷性。格式如下：



switch语句

```
switch(表达式)
{
        case   常量1:
               [语句块1;
               break;]
        case   常量2:
               [语句块2;
               break;]
        ……
        case   常量n:
               [语句块n;
               break;]
        [default:语句块n+1;break;]
}
```

- 显然，通过if-else语句的嵌套也能实现switch语句的功能，但通常使用switch语句更简练，而且可读性强，程序的执行效率也高；
- 表达式的返回类型必须是byte，short，char和int四种整数类型，或者是枚举类型(enumerated type)，从1.7版本开始java支持switch对String类型的分支判断；
- case子句中的值必须是常量，且值应该不相同；
- case的分支中有多个执行语句时不必用“{ }”括起；
- default子句是可选的，它处理没有被任何case语句显式处理的值。若表达式中的值与所有case都不相配，且没有default子句，则跳出switch语句；
- break语句的作用是执行完一个case分支后使程序跳出switch语句块，它是可选的，如果没有这条语句，流程控制将一个接一个的执行后续的case操作，而不再判断标签条件是否匹配。这样分支语句将变得效率极低且容易误操作。但是有时候程序员也会将其省略以达到某些特殊需要。

switch例程

```
public class SwitchDemo {
        public static void main(String[] args) {
        int month = 8;
        switch (month) {
                case 1:  System.out.println("January"); break;
                case 2:  System.out.println("February"); break;
                case 3:  System.out.println("March"); break;
                case 4:  System.out.println("April"); break;
                case 5:  System.out.println("May"); break;
                case 6:  System.out.println("June"); break;
                case 7:  System.out.println("July"); break;
                case 8:  System.out.println("August"); break;
                case 9:  System.out.println("September"); break;
                case 10: System.out.println("October"); break;
                case 11: System.out.println("November"); break;
                case 12: System.out.println("December"); break;
                default: System.out.println("Invalid month.");break;
                }
        }
}
```

```
August

Process completed.
```

switch例程

```
public class SwitchDemo {
        public static void main(String[] args) {
        int month = 8;
        switch (month) {
                case 1:  System.out.println("January");
                case 2:  System.out.println("February");
                case 3:  System.out.println("March");
                case 4:  System.out.println("April");
                case 5:  System.out.println("May");
                case 6:  System.out.println("June");
                case 7:  System.out.println("July");
                case 8:  System.out.println("August");
                case 9:  System.out.println("September");
                case 10: System.out.println("October");
                case 11: System.out.println("November");
                case 12: System.out.println("December");
                default: System.out.println("Invalid month.");
                }
        }
}
```

```
August
September
October
November
December
Invalid month.
```

- 在一些特殊情况下，多个不同的case值要执行一组相同的操作，可以写成如下形式：

case语句

```
……
case   常量n:
case   常量n+1:
       语句块;
       [break;]
……
```

switch例程2

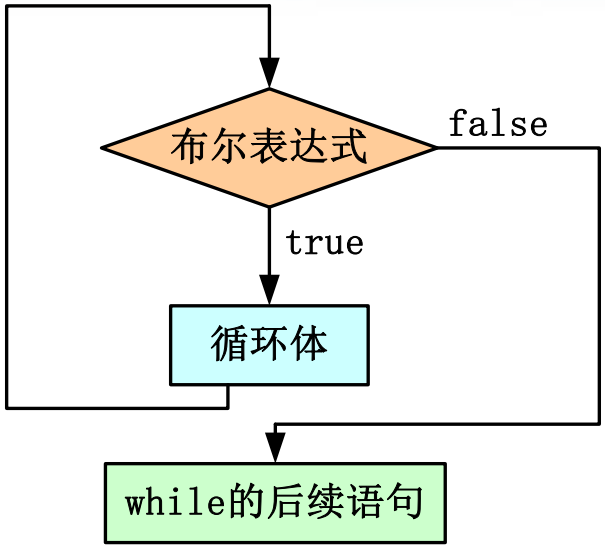
```
public class SwitchDemo2 {
        public static void main(String[] args) {
        int month = 8;
        switch (month) {
                case 1:
                case 2:
                case 3:  System.out.println("Spring"); break;
                case 4:
                case 5:
                case 6:  System.out.println("Summer"); break;
                case 7:
                case 8:
                        case 9:  System.out.println("Autumn"); break;
                case 10:
                case 11:
                case 12: System.out.println("Winter"); break;
                default: System.out.println("Invalid month.");break;
                }
        }
}
```

- 对String的支持，其中传递给switch的值以及case后面的值不能为null。式：

```
import java.util.*;
public class StringSwitch {
    public static void main(String args[]){
            Scanner s=new Scanner(System.in);
            String name=s.next();
            switch(name){
                    case "administrater":
                            System.out.println("你是管理员");
                            break;
                    case "guest":
                            System.out.println("你是访客");
                            break;
                    default:
                            System.out.println("用户名非法");
            }
    }
}
```

3.3 循环语句

- while语句

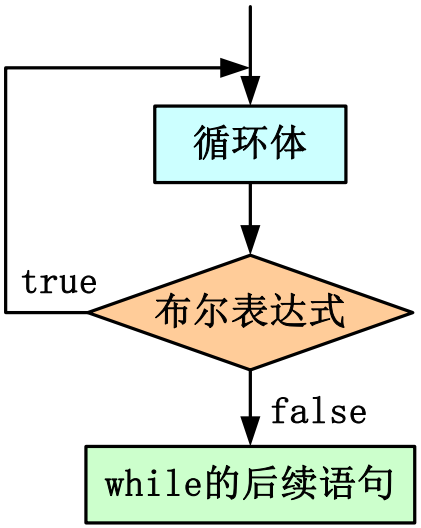


while语句

```
while(条件表达式)
{
        循环体;
}
```

- 其中，while是关键字，布尔表达式是循环条件，语句为循环体。
- 执行过程如下：首先判断布尔表达式，若为真，则执行循环体，然后再判断条件，直到布尔表达式的值为假，停止执行语句。注意：
  - 该语句是先判断后执行，若一开始条件就不成立，则不执行循环体；
  - 在循环体内部一定要有改变条件的语句，否则是死循环。

- do-while语句
  - 格式：




```
do-while语句
do
{
        循环体;
}
while(条件表达式);
```

  - do-while语句是“先执行后判断”型，先执行一次循环体中的语句，然后测试布尔表达式的值，若为true，则继续执行循环，否则执行do-while语句的后续语句。

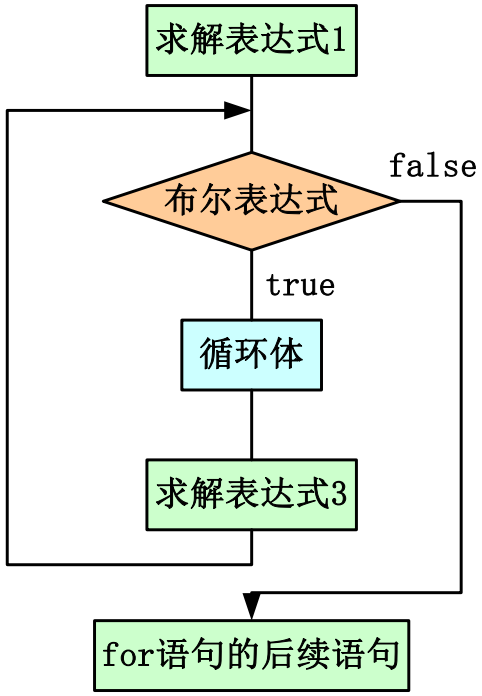
while例程

```
public class WhileDemo {
        public static void main(String[] args) {
                int count = 1;
                while (count < 11) {
                        System.out.println("Count is: " + count);
                        count++;
                }
        }
}
```

do-while例程

```
public class DoWhileDemo {
        public static void main(String[] args) {
                int count = 1;
                do {
                        System.out.println("Count is: " + count);
                        count++;
                } while (count <= 11);
        }
}
```

- for语句
  - 是三种循环语句中功能较强、形式灵活、使用较频繁的循环语句结构，尤其适合于循环次数清晰的场合。



for语句

```
for(表达式1;布尔表达式;表达式3)
{
        循环体;
}
```

- 执行过程：

  ①按表达式1将初值赋给循环控制变量；

  ②按布尔表达式判断循环是否成立，即判断控制变量的值是否满足布尔表达式的条件，若条件不成立，则转步骤⑤；若条件成立，则执行循环；

  ③按表达式3修改控制变量；

  ④返回步骤②；

  ⑤结束循环，执行for语句的后续语句。

- for语句中循环控制变量必须是有序类型。循环控制变量初值和终值通常是与控制变量类型相一致的常量，也可以是表达式。对控制变量的初始化只执行一次，而循环次数由初值和终值决定。

- 其他形式：
  - 当“表达式1”、“表达式2”、“表达式3”都为空的时候，相当于一个无限循环，如：

特殊的for语句

```
for( ; ; )
{
        循环体;
}
```

  - 有时，for语句在“表达式1”和“表达式3”的位置上需要包含多个语句，由于不能在for语句的“( )”中使用“{ }”来定义复合语句，因此提供了用逗号分隔的语句系列。如：

特殊的for语句

```
for(表达式1,表达式2[,…];布尔表达式;表达式I,表达式II[,…])
{
        循环体;
}
```

- 在for语句的圆括号的后面加个分号，则相当于for语句的循环体为空语句。例如：

特殊的for语句

```
for(表达式1;布尔表达式;表达式3);
```

- 例程

for例程

```
public class ForDemo
{
        public static void main(String[] args)
        {
                for(int i=1; i<11; i++)
                {
                        System.out.println("Count is: " + i);
                }
        }
}
```

注意：初始化表达式中声明变量的代码。该变量的作用域从其声明开始直到for循环控制块的结束。如果在循环之外不需要用到该变量，那么最好在初始化表达式中声明，这样就限制了它们的生存周期并且减少了错误的可能。

- 适用于简单遍历的foreach循环
  - 用于在集合和数组之中进行迭代，可以使循环更加紧凑和容易阅读。

foreach循环无需获得数组的长度，无需根据索引来访问数组元素，会自动遍历数组，当每个元素都被迭代一次后for循环自动终止。

增强的for例程

```
public class EnhancedForDemo {
        public static void main(String[] args) {
                int[] numbers = {1,2,3,4,5,6,7,8,9,10};
                for (int item : numbers) {
                        System.out.println("Count is: " + item);
                }
        }
}
```

- foreach循环一般用来输出数组(集合)元素，通常不要用foreach来对循环变量进行赋值，容易引起错误：

```
public class ForEachTest{
    public static void main(String args[ ]){
        int[ ] nums={1,2,3,4,5};
        for(int i:nums){
            i=5;
            System.out.println(i);
        }
        for(int i:nums){
            System.out.println(i);
        }
    }
}
```

```
run:
5
5
5
5
5
1
2
3
4
5
成功构建（总时间：0 秒）
```

- 循环嵌套
  - 一个循环体内又包含另一个完整的循环结构，称为循环的嵌套。内嵌的循环中还可以嵌套循环，这就是多重循环。while、do-while、for之间都可以互相嵌套使用。

循环嵌套例程

```
public class Mul99 {
        public static void main(String[] args) {
                int i,j,n=9;
                for(i=1;i<=n;i++){
                        for(j=1;j<=n;j++)
                                System.out.print("  "+i*j);
                        System.out.println();
                }
        }
}
```

3.4 跳转语句

- 跳转语句用来实现程序执行过程中流程的转移。
  - goto语句使程序结构混乱，可读性差；
  - Java语言取消了goto语句，通过break语句、continue语句和return语句实现流程的转移。
- break语句
  - 使程序的流程从一个封闭语句块（如switch）中跳出来，也可以用于退出一个循环（如do、for、while），此外，还可以跳到相应的标记位。
  - 分类：
    - 不带标号的break语句
      - 从它所在的switch分支或最内层循环中跳出来，执行分支或循环体后面的语句。
      - 不能用于循环语句和switch语句之外的其他任何语句中。

无标号break例程

```
public class BreakDemo {
    public static void main(String[] args) {
        int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076,2000, 8, 622, 127 };
        int searchfor = 12;
        int i;
        boolean foundIt = false;
        for (i = 0; i < arrayOfInts.length; i++) {
                if (arrayOfInts[i] == searchfor) {
                        foundIt = true;
                        break;
                }
        }
        if (foundIt) {
                System.out.println("Found " + searchfor + " at index " + i);
        }
        else {
                System.out.println(searchfor + " not in the array");
        }
        }
}
```

- 带标号的break语句
  - break语句的语法格式如下：

```
break 标号;
```

  - 设置标记的语句格式：

```
标号: 语句
```

  - “标号”是程序中设置好的标记名，程序跳到标号所在的语句或者语句块的下一句开始执行。
  - break语句必须在加标记的代码块的内部才可以跳到标记位所标记的语句(块)之后；
  - break终止带加标记的语句，不是把控制流转移到标记位置，而是转移到紧跟在标记的语句（块）后面的语句。
  - 可以把标号设置在任意的语句或语句块之前。

带标号break例程

```
public class BreakWithLabelDemo {
        public static void main(String[] args) {
                int[][] arrayOfInts = {{32,87,3,589},{12,1076,2000,8},{622,127,77,955}};
                int searchfor = 12;
                int i;
                int j = 0;
                boolean foundIt = false;
                search:
                        for (i = 0; i < arrayOfInts.length; i++) {
                                for (j = 0; j < arrayOfInts[i].length; j++) {
                                        if (arrayOfInts[i][j] == searchfor) {
                                                foundIt = true;
                                                break search;
                                        }
                                }
                        }
                if (foundIt) {
                        System.out.println("Found " + searchfor + " at " + i + ", " + j);
                }
                else {
                        System.out.println(searchfor + " not in the array");
                }
        }
}
```

- continue语句
  - continue语句只能在循环语句中使用。
  - 它和break语句的区别是：continue语句只终止本次循环，而不是终止整个循环；而break语句则是结束整个循环语句的执行。
  - continue语句同样有两种格式：
    - 不带标号
      - 终止本次循环，即跳过循环体中continue语句后面的语句，回到循环体的条件测试部分继续执行。注意：只能跳过本次循环的剩余语句。
    - 带标号的情况

continue 标号;　　标号：语句

      - 跳过标号指出的语句块中的所有余下语句部分，回到标号所指语句块的条件测试部分继续执行。
      - 标号应该定义在程序中外层循环语句的前面，用来标志这个循环结构。
      - 带标号的continue语句使程序的流程直接转入标号标明的循环层次。

continue例程

```
public class ContinueDemo {
        public static void main(String[] args) {
                String searchMe = "peter piper picked a peck of pickled peppers";
                int max = searchMe.length();
                int numPs = 0;
                for (int i = 0; i < max; i++) {
                        if (searchMe.charAt(i) != 'p')
                                continue;
                        numPs++;
                }
                System.out.println("Found " + numPs + " p's in the string.");
        }
}
```

带标号continue例程

```
public class ContinueWithLabelDemo {
        public static void main(String[] args) {
                String searchMe = "Look for a substring in me";
                String substring = "sub";
                boolean foundIt = false;
                int max = searchMe.length() - substring.length();
        test:
                for (int i = 0; i <= max; i++) {
                        int n = substring.length();
                        int j = i;
                        int k = 0;
                        while (n-- != 0) {
                                if (searchMe.charAt(j++) != substring.charAt(k++)) {
                                        continue test;
                                }
                        }
                        foundIt = true;
                        break test;
                }
                System.out.println(foundIt ? "Found it" : "Didn't find it");
        }
}
```

- return语句
  - 用来从当前方法中退出，可使正在执行的分支程序返回到调用它的方法的相应语句处，并从紧跟该语句的下一条语句继续执行，格式如下：

```
return [表达式];
```

  - 说明：
    - 表达式的值就是被调用方法的返回值，如果方法没有返回值，则return语句中的表达式可以省略；
    - 如果方法中没有出现return语句，则执行完方法中的最后一条语句后自动返回调用它的方法。