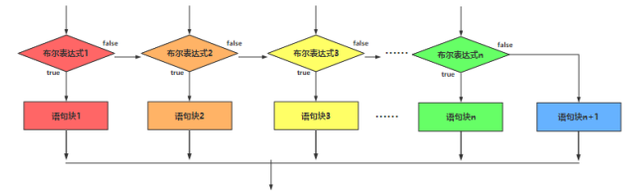
04 控制语句

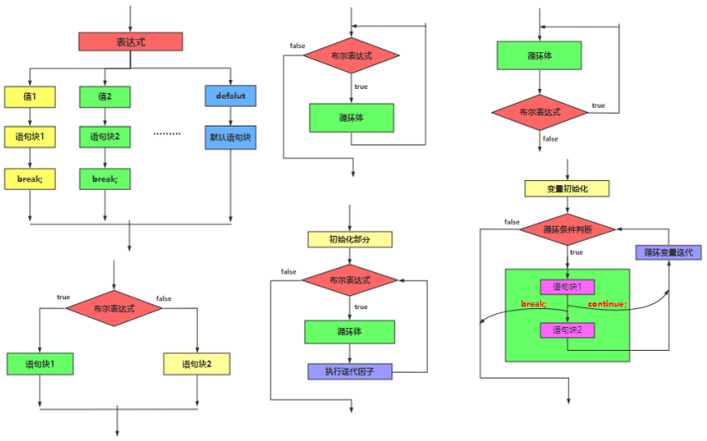
控制语句：把语句组合成能完成一定功能的小逻辑模块。它分为三类：顺序、选择和循环。学会控制语句，是真正跨入编程界的“门槛”，是成为“程序猿”的“门票”。

1. **“顺序结构”代表“先执行 a，再执行 b”的逻辑。**比如，先找个女朋友，再给女朋友打电话；先订婚，再结婚；

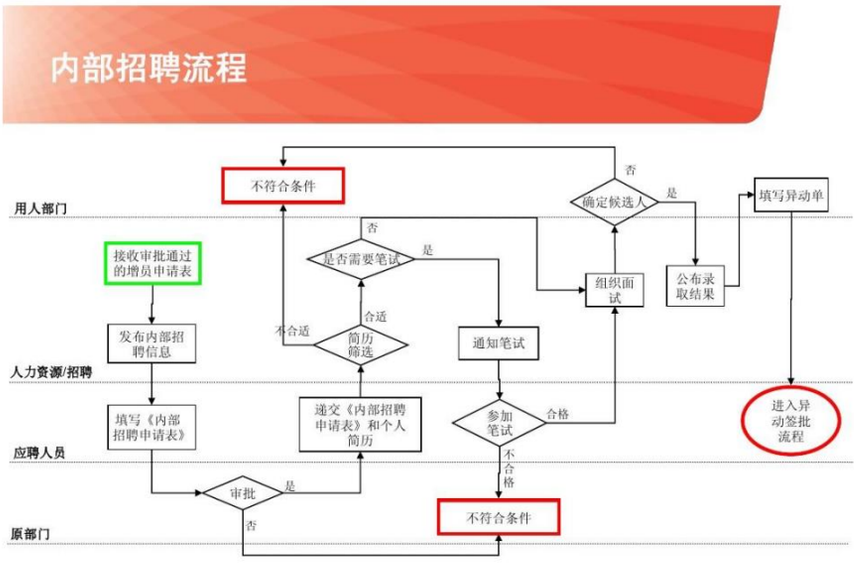
2. **“条件判断结构”代表“如果…，则…”的逻辑。**比如，如果女朋友来电，则迅速接电话；如果看到红灯，则停车；

3. **“循环结构”代表“如果…，则重复执行…”的逻辑。**比如，如果没打通女朋友电话，则再继续打一次； 如果没找到喜欢的人，则再继续找。



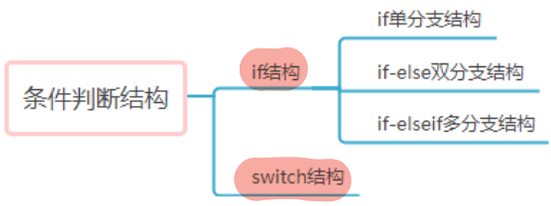


很神奇的是，三种流程控制语句就能表示所有的事情！不信，你可以试试拆分你遇到的各种事情。**实际上，任何软件和程序，小到一个练习，大到一个操作系统，本质上都是由“变量、选择语句、循环语句”组成。**

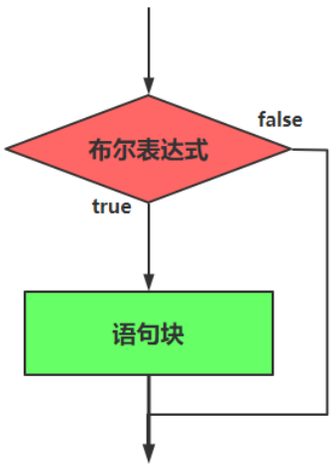


条件判断结构(选择结构)

条件判断结构有：if 结构和 switch 结构。而 if 结构又可以分为 if 单分支结构、if-else 双分支结构、if-else if-else 多分支结构。



if 单分支结构



语法结构:

```
if(布尔表达式){
    语句块
}
```

新手雷区

1. 如果 if 语句不写{}，则只能作用于后面的第一条语句。
2. 强烈建议，任何时候都写上{}，即使里面只有一句话！

【示例】if 单分支结构（掷骰子游戏）

Math 类的使用

- Math.random()该方法用于产生 0 到 1 区间的 double 类型的随机数，但是不包括 1。
  int i = (int) (6 * Math.*random*()); //产生：[0，5]之间的随机整数

```
public class Test1 {
    public static void main(String[ ] args) {
        //通过掷三个骰子看看今天的手气如何?
        int i = (int)(6 * Math.random()) + 1;//通过 Math.random()产生随机数
        int j = (int)(6 * Math.random()) + 1;
        int k = (int)(6 * Math.random()) + 1;
        int count = i + j + k;
        //如果三个骰子之和大于 15，则手气不错
        if(count > 15) {
            System.out.println("今天手气不错");
        }
        //如果三个骰子之和在 10 到 15 之间，则手气一般
        if(count >= 10 && count <= 15) { //错误写法：10<=count<=15
            System.out.println("今天手气很一般");
        }
        //如果三个骰子之和小于 10，则手气不怎么样
        if(count < 10) {
            System.out.println("今天手气不怎么样");
        }
```

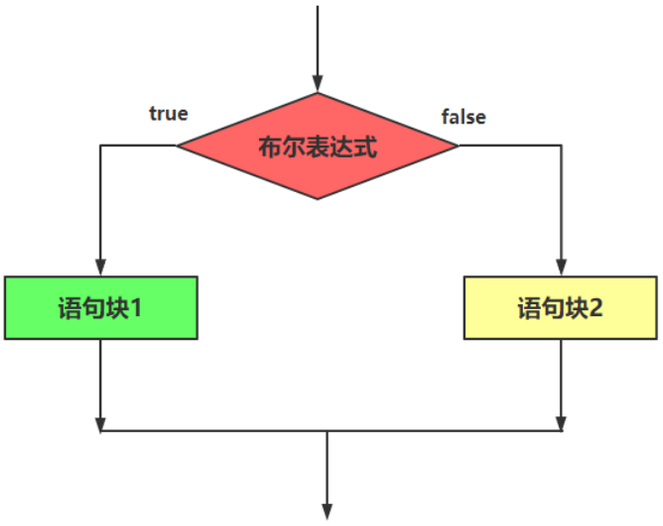
```
            System.out.println("得了" + count + "分");
        }
}
```

执行结果如图 3-2 所示：

```
今天手气不怎么样
得了8分
```

图 3-2 示例 3-1 运行效果图

if-else 双分支结构



语法结构：

```
if(布尔表达式){
    语句块 1
}else{
    语句块 2
}
```

当布尔表达式为真时，执行语句块 1，否则，执行语句块 2。也就是 else 部分。

【示例】if-else 双分支结构

```
public class Test2 {
    public static void main(String[ ] args) {
        //随机产生一个[0.0, 4.0)区间的半径，并根据半径求圆的面积和周长
```

```
            double r = 4 * Math.random();
          //Math.pow(r, 2)求半径 r 的平方
            double area = 3.14* r*r;
            double circle = 2 * Math.PI * r;
            System.out.println("半径为：  " + r);
            System.out.println("面积为：  " + area);
            System.out.println("周长为：  " + circle);
            //如果面积>=周长，则输出"面积大于等于周长"，否则，输出周长大于面积
            if(area >= circle) {
                System.out.println("面积大于等于周长");
            } else {
                System.out.println("周长大于面积");
            }
        }
}
```

执行结果如图 3-4 所示：

```
半径为：  2.3954358845679184
面积为：  18.026813888408498
周长为：  15.050967554207881
面积大于等于周长
```

图 3-4 示例 3-2 运行效果图

条件运算符有时候可用于代替 if-else。

【示例】if-else 与条件运算符的比较：使用 if-else

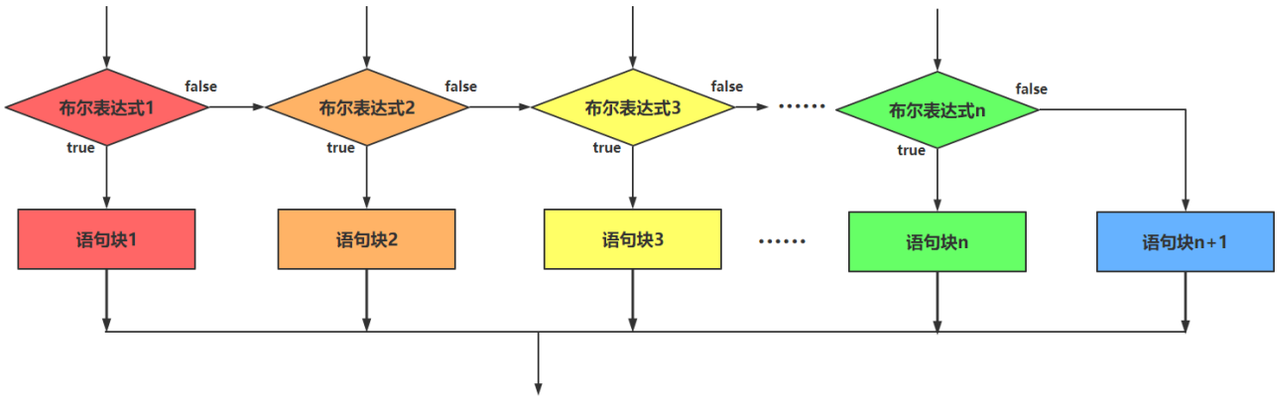
```
int a = 3;
int b = 4;

int c2 = 0;
if(a<b){
    c2 = b;
}else{
    c2 = a;
}
System.out.println(c2);
```

【示例】if-else 与条件运算符的比较：使用条件运算符

```
int a = 3;
int b = 4;
```

```
int c = a<b?b:a;          //返回 a 和 b 的最大值
System.out.println(c);
```

if-else if-else 多分支结构



语法结构：

```
if(布尔表达式 1) {
    语句块 1;
} else if(布尔表达式 2) {
   语句块 2;
}.....
else if(布尔表达式 n){
语句块 n;
} else {
语句块 n+1;
}
```

当布尔表达式 1 为真时，执行语句块 1；否则，判断布尔表达式 2，当布尔表达式 2 为真时，执行语句块 2；否则，继续判断布尔表达式 3；如果 1~n 个布尔表达式均判定为假时，则执行语句块 n+1，也就是 else 部分。

【示例】if-else if-else 多分支结构

```
public class Test5 {
    public static void main(String[ ] args) {
        int age = (int) (100 * Math.random());
```

```
            System.out.print("年龄是" + age + "，  属于");
        //15 岁以下儿童；15-24 青年；25-44 中年；45-64 中老年；65-84 老年；85 以上老寿星
            if (age < 15) {
                System.out.println("儿童，  喜欢玩！");
            } else if (age < 25) {
                System.out.println("青年，  要学习！");
            } else if (age < 45) {
                System.out.println("中年，  要工作！");
            } else if (age < 65) {
                System.out.println("中老年，  要补钙！");
            } else if (age < 85) {
                System.out.println("老年，  多运动！");
            } else {
                System.out.println("老寿星，  古来稀！");
            }
        }
}
```

执行结果如图 3-8 和图 3-9 所示：

```
年龄是45，  属于中老年，  要补钙！
```

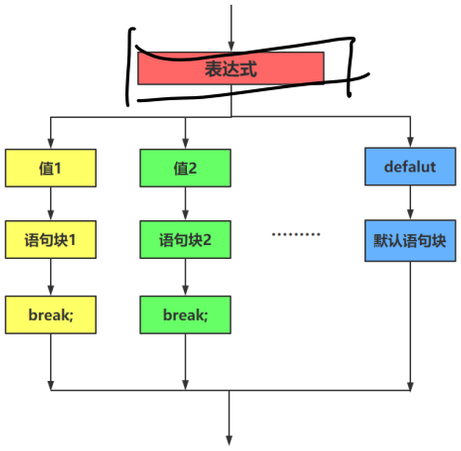
图 3-8 示例 3-5 运行效果图 1

本节作业

1. Math.random()是返回[0,1)之间的随机浮点数，需要[10,15]之间的随机数，如何通过Math.random()获取。
2. 完成课堂上的“掷骰子”游戏。
3. 键盘输入一个圆的半径或者随机生成一个半径，计算出圆的面积、周长。比较面积和周长的数字，并打印出不同的提示。
4. 生成[0,120]随机的年龄，然后根据不同的年龄段输出相应的语句（参考课上代码）
5. 生成[0,100]之间的分数。如果 90 以上(含 90)则输出“A+”，如果 80-89 输出“A”，如果 70-79 输出“B”,60-69 之间“C”，60 以下是“D”。

switch 语句

switch 多分支结构(多值情况)



语法结构：

```
switch (表达式) {
      case  值 1:
            语句块 1;
            [break];
      case  值 2:
            语句块 2;
            [break];
       … … …          … …
      [default:
            默认语句块;]
}
```

1. switch 会根据表达式的值从相匹配的 case 标签处开始执行，一直执行到 break 处或者是 switch 的末尾。如果表达式的值与任一 case 值不匹配，则进入 default 语句。

2. switch 中表达式的值，是 int(byte、short、char 也可，long 不行)、枚举，字符串。

练习一

```
//grade 表示大学年级
int  grade = 1;
```

```
if(grade==1) {
    System.out.println("大学一年级，可以放松一下，学着谈
谈恋爱");
}else if(grade==2){
    System.out.println("大学二年级，少玩点游戏，不空虚，
不慌嘛？");
}else if(grade==3) {
    System.out.println("大学三年级，专业课开始了，好好
学，找份好工作");
}else{
    System.out.println("大四了，要毕业了。因为跟着尚学堂
学习，好工作搞定！");
}


switch (grade){
    case 1:
        System.out.println("大学一年级");
        break;
    case 2:
        System.out.println("大学二年级");
        break;
    case 3:
```

```
        System.out.println("大学三年级");
        break;
    default:
        System.out.println("大四了，要毕业了");
        break;
}
```

switch 结构

练习二

```
public class TestSwitch02 {

    public static void main(String[] args){
        int  month = 2;  //1 表示 1 月，2 表示 2 月，....

        if(month==1||month==2||month==3){
            System.out.println("春季");
        }else if(month==4||month==5||month==6){
            System.out.println("夏季");
        }else if(month==7||month==8||month==9){
            System.out.println("秋季");
        }else{
            System.out.println("冬季");
        }


        System.out.println("========使用 switch 改造上面的代码，switch 特别适合多值判断
==============");
        switch (month){
            case 1:
            case 2:
            case 3:
                System.out.println("春季");
                break;
            case 4:
            case 5:
```

```
            case 6:
                System.out.println("夏季");
                break;
            case 7:
            case 8:
            case 9:
                System.out.println("秋季");
                break;
            default:
                System.out.println("冬季");
        }
    }
}
```

switch 接收字符串(JDK7 新特性，掌握)

【示例】JDK7.0 之后可以直接使用字符串。

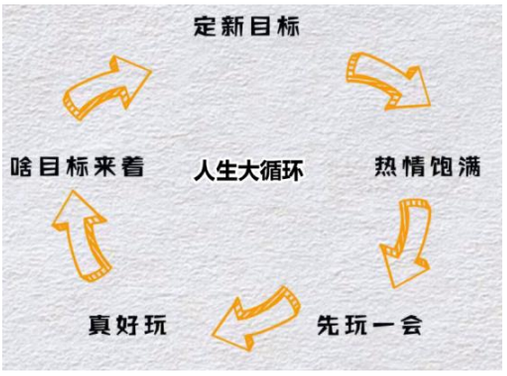
```
String  str = "audi";
switch (str){
    case "audi":
        System.out.println("我买了个奥迪车");
        break;
    case "benz":
        System.out.println("我买了个奔驰车");
        break;
    default:
        System.out.println("比亚迪，挺好！");
}
```

循环结构

每天一句吃了么"

直到你爱上我为止

爱情大循环



循环结构分两大类，一类是当型，一类是直到型。

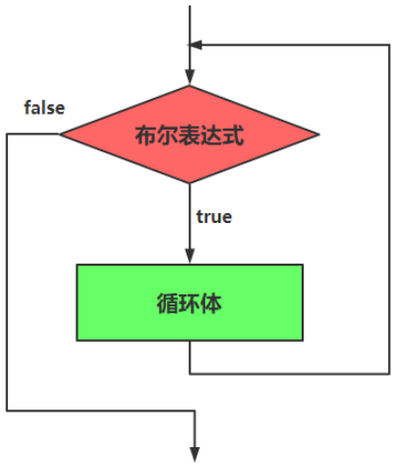
- **当型：**

当布尔表达式条件为 true 时，反复执行某语句，当布尔表达式的值为 false 时才停止循环，比如：while 与 for 循环。

- **直到型：**

先执行某语句， 再判断布尔表达式，如果为 true，再执行某语句，如此反复，直到布尔表达式条件为 false 时才停止循环，比如 do-while 循环。

while 循环



语法结构：

```
while (布尔表达式) {
    循环体;
}
```

1. 在循环刚开始时，会计算一次“布尔表达式”的值，若条件为真，执行循环体。而对于后来每一次额外的循环，都会在开始前重新计算一次。
2. 语句中应有使循环趋向于结束的语句，否则会出现无限循环——"死"循环。

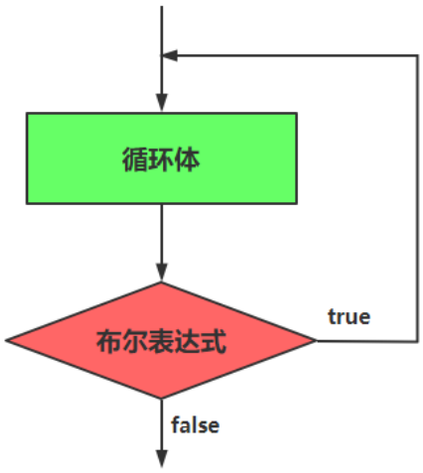
【示例】while 循环结构：求 1 到 100 之间的累加和

```
public class Test7 {
    public static void main(String[ ] args) {
        int  i = 0;
        int  sum = 0;
        // 1+2+3+...+100=?
        while (i <= 100) {
            sum += i;//相当于 sum = sum+i;
            i++;
        }
        System.out.println("Sum= " + sum);
    }
}
```

执行结果如图所示：

```
Sum= 5050
```

do-while 循环



语法结构：

```
do {
    循环体;
} while(布尔表达式) ;
```

do-while 循环结构会先执行循环体，然后再判断布尔表达式的值，若条件为真，执行循环体，当条件为假时结束循环。do-while 循环的循环体至少执行一次。

【示例】do-while 循环结构：求 1-100 之间的累加和

```
public class Test8 {
    public static void main(String[ ] args) {
        int i = 0;
        int sum = 0;
        do {
            sum += i; // sum = sum + i
            i++;
        } while (i <= 100);   //此处的；不能省略
        System.out.println("Sum= " + sum);
    }
}
```

执行结果如图 3-16 所示：

```
Sum= 5050
```

【示例】while 与 do-while 的区别

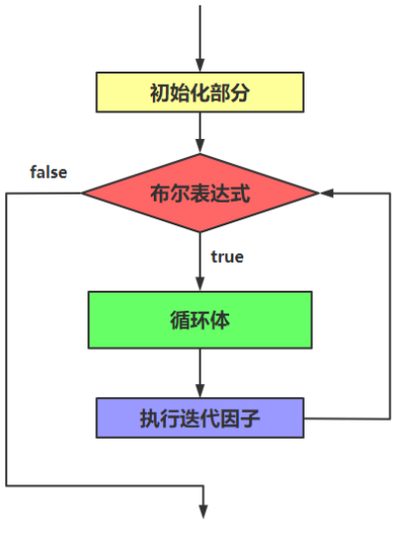
```java
public class Test9 {
    public static void main(String[ ] args) {
        //while 循环：先判断再执行
        int a = 0;
        while (a < 0) {
            System.out.println(a);
            a++;
        }
        System.out.println("-----");
        //do-while 循环：先执行再判断
        a = 0;
        do {
            System.out.println(a);
            a++;
        } while (a < 0);
    }
}
```

运行结构，可以看出 do-while 总是保证循环体至少会被执行一次！

课堂练习

1. 使用 while/for 循环：计算 1-100 之间所有数字的和，所有偶数的和，所有奇数的和。
2. 使用 while/for 循环 0-130 之间的数字，每行显示 5 个数字。

循环结构(for)



语法结构：

```
for (初始表达式; 布尔表达式; 迭代因子) {
        循环体;
}
```

- ❑ 初始化部分设置：循环变量的初值
- ❑ 条件判断部分为：布尔表达式
- ❑ 迭代因子：控制循环变量的增减

for 循环在执行条件判定后，先执行的循环体部分，再执行步进。

【示例】for 循环

```
public class Test10 {
    public static void main(String args[ ]) {
        int sum = 0;
        //1.求 1-100 之间的累加和
        for (int i = 0; i <= 100; i++) {
            sum += i;
        }
        System.out.println("Sum= " + sum);
        //2.循环输出 9-1 之间的数
        for(int i=9;i>0;i--){
            System.out.print(i+"、 ");
        }
        System.out.println();
        //3.输出 90-1 之间能被 3 整除的数
        for(int i=90;i>0;i-=3){
            System.out.print(i+"、 ");
        }
        System.out.println();
    }
}
```

执行结果如图 3-19 所示：

```
Sum= 5050
9、8、7、6、5、4、3、2、1、
90、87、84、81、78、75、72、69、66、63、60、57、54、51、48、45、42、39、36、33、30、27、24、21、18、15、12、9、6、3、
```

图 3-19 示例 3-10 运行效果图

【示例】逗号运算符

```
public class Test11 {
    public static void main(String[ ] args) {
        for(int i = 1, j = i + 10; i < 5; i++, j = i * 2) {
            System.out.println("i= " + i + " j= " + j);
        }
    }
}
```

执行结果如图 3-20 所示：

```
i= 1 j= 11
i= 2 j= 4
i= 3 j= 6
i= 4 j= 8
```

【示例 3-12】无限循环

```
public class Test12 {
    public static void main(String[ ] args) {
        for ( ; ; ) {      // 无限循环: 相当于 while(true)
        System.out.println("北京尚学堂");
        }
    }
}
```

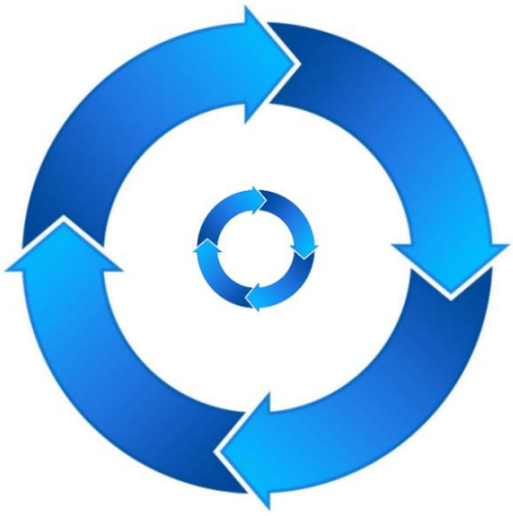
编译器将 while(true)与 for(;;)看作同一回事，都指的是无限循环。

【示例 3-13】初始化变量的作用域

```
public class Test13 {
    public static void main(String[] args) {
        for(int i = 1; i < 10; i++) {
            System.out.println(i+" ");
        }
        System.out.println(i);//编译错误，无法访问在 for 循环中定义的变量 i
    }
}
```

嵌套循环

循环语句内部，再写一个或多个循环，称为嵌套循环。一般工作中多见的就是两层。



【示例】嵌套循环

```
public class Test14 {
	public static void main(String args[ ]) {
		for (int i=1; i <=5; i++) {
			for(int j=1; j<=5; j++){
				System.out.print(i+"    ");
			}
			System.out.println();
		}
	}
}
```

执行结果如图所示：

```
1   1   1   1   1
2   2   2   2   2
3   3   3   3   3
4   4   4   4   4
5   5   5   5   5
```

【示例 3-15】使用嵌套循环实现九九乘法表


```
public class Test15 {
    public static void main(String args[ ]) {
        for (int i = 1; i < 10; i++) { // i 是一个乘数
            for (int j = 1; j <= i; j++) { // j 是另一个乘数
                System.out.print(j + "*" + i + "=" + (i * j < 10 ? (" " + i * j) : i * j) + "
");
            }
            System.out.println();
        }
    }
}
```

执行结果如图所示：

```
1*1= 1
1*2= 2  2*2= 4
1*3= 3  2*3= 6  3*3= 9
1*4= 4  2*4= 8  3*4=12  4*4=16
1*5= 5  2*5=10  3*5=15  4*5=20  5*5=25
1*6= 6  2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7= 7  2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8= 8  2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9= 9  2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

本节练习

1. 输出九九乘法表。

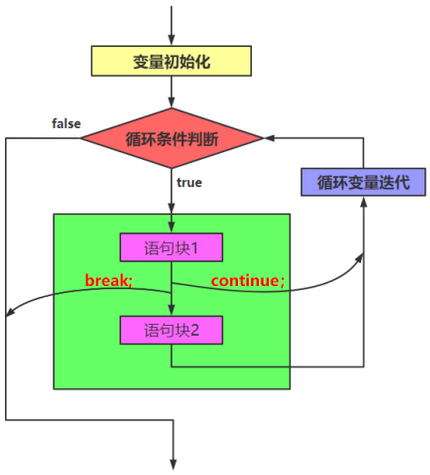
2. 使用嵌套循环，打印输出 5×5 的方阵，格式如下：

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

3. 使用嵌套循环，打印输出 5*5 的方阵，格式如下：

```
*   #   *   #   *
#   *   #   *   #
*   #   *   #   *
#   *   #   *   #
*   #   *   #   *
```

break 语句和 continue 语句



1. break 用于强行退出整个循环

2. continue 用于结束本次循环，继续下一次

【示例】break 语句

```
//产生 100 以内的随机数，直到随机数为 88 时终止循环
public class Test16 {
    public static void main(String[ ] args) {
        int total = 0;//定义计数器
```

```
        System.out.println("Begin");
        while (true) {
            total++;//每循环一次计数器加 1
            int i = (int) Math.round(100 * Math.random());
            //当 i 等于 88 时，退出循环
            if (i == 88) {
                break;
            }
        }
        //输出循环的次数
        System.out.println("Game over，  used " + total + " times.");
    }
}
```

执行结果如图所示：

```
Begin
Game over, used 110 times.
```

continue 语句用在循环语句体中，用于终止某次循环过程，即跳过循环体中尚未执行的语句，接着进行下一次是否执行循环的判定。

【示例】continue 语句

```
//把 100~150 之间不能被 3 整除的数输出，并且每行输出 5 个
public class Test17 {
    public static void main(String[ ] args) {
        int count = 0;//定义计数器
        for (int i = 100; i < 150; i++) {
            //如果是 3 的倍数，则跳过本次循环，继续进行下一次循环
            if (i % 3 == 0){
                continue;
            }
            //否则（不是 3 的倍数），输出该数
            System.out.print(i + "、");
            count++;//每输出一个数，计数器加 1
            //根据计数器判断每行是否已经输出了 5 个数
            if (count % 5 == 0) {
                System.out.println();
            }
        }
    }
```

```
}
```

执行结果如图所示：

```
100、101、103、104、106、
107、109、110、112、113、
115、116、118、119、121、
122、124、125、127、128、
130、131、133、134、136、
137、139、140、142、143、
145、146、148、149、
```

图 3-24 示例 3-17 运行效果图

带标签的 continue 语句

“标签”是指后面跟一个冒号的标识符，例如：“label:”。对 Java 来说唯一用到标签的地方是在循环语句之前。

“goto 有害”论中，最有问题的就是标签，而非 goto，随着标签在一个程序里数量的增多，产生错误的机会也越来越多。但 Java 标签不会造成这方面的问题，因为它们的活动场所已被限死，不可通过特别的方式到处传递程序的控制权。

【示例 3-18】带标签的 continue

```
//控制嵌套循环跳转(打印 101-150 之间所有的质数)
public class Test18 {
    public static void main(String args[ ]) {
        outer: for (int i = 101; i < 150; i++) {
            for (int j = 2; j < i / 2; j++) {
                if (i % j == 0){
                    continue outer;   //符合某条件，跳到外部循环继续
                }
            }
            System.out.print(i + "    ");
        }
    }
}
```

执行结果如图所示：

```
101   103   107   109   113   127   131   137   139   149
```

图 3-25 示例 3-18 运行效果图

课堂练习

1. 薪水计算器：

(1) 通过键盘输入用户的月薪，每年是几个月薪水。

(2) 输出用户的年薪

(3) 输出一行字“如果年薪超过 10 万，恭喜你超越 90%的国人”，“如果年薪超过 20 万，恭喜你超越 98%的国人”。

(4) 键盘输入数字 88，则退出程序（使用 break 退出循环）

(5) 键盘输入 66，直接显示“重新开始计算...”，然后算下一个用户的年薪。

```
/**
 * 薪水计算器
 * (1)通过键盘输入用户的月薪，每年是几个月薪水。
 * (2)输出用户的年薪
 * (3)输出一行字“如果年薪超过 10 万，恭喜你超越 90%的国人”，“如果年薪超过 20 万，恭喜
你超越 98%的国人”。
 * (4)直到键盘输入数字 88，则退出程序（使用 break 退出循环）
 * (5)键盘输入 66，直接显示“重新开始计算...”，然后算下一个用户的年薪。
 */
import java.util.Scanner;
public class SalaryCalculator {
```

```java
public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.println("**********我的薪水计算器**********");
    System.out.println("1.输入 88，退出程序\n2.输入 66，计算下一个年薪");

    while(true){
        System.out.println("请输入月薪：");
        int monthSalary = s.nextInt();
        System.out.println("请输入一年几个月薪资：");
        int months = s.nextInt();
        int yearSalary = monthSalary*months;  //年薪

        System.out.println("年薪是："+yearSalary);
        if(yearSalary>=200000){
            System.out.println("恭喜你超越 98%的国人");
        }else if(yearSalary>=100000){
            System.out.println("恭喜你超越 90%的国人");
        }

        System.out.println("输入 88，退出系统；输入 66，继续计算。");
        int comm = s.nextInt();
```

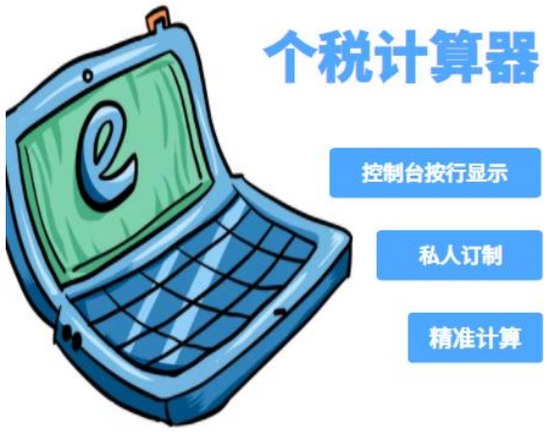
```
            if(comm==88){
                System.out.println("系统退出！");
                break;
            }
            if(comm==66) {
                System.out.println("继续计算下一个薪资");
                continue;
            }
        }
    }
}
```

2. 个税计算器：

(1) 通过键盘输入用户的月薪

(2) 百度搜索个税计算的方式，计算出应缴纳的税款

(3) 直到键盘输入 88，则退出程序（使用 break 退出循环）



学员刘佳作业：

```
import java.util.Scanner;

//个税计算器
public class TestPersonalIncomeTax {
```

```
    public static void main(String[] args) {
        //2. 个税计算器：
//(1)     通过键盘输入用户的月薪
//(2)     百度搜索个税计算的方式，计算出应缴纳的税款
//(3)     直到键盘输入“exit”，则退出程序（使用 break 退出循环）
        /*
        应纳税所得额=工资收入金额 - 各项社会保险费 - 起征点(5000 元)
        应纳税额=应纳税所得额 x 税率 - 速算扣除数
        级数 应纳税所得额                    税率(%)          速算扣除数
        1       不超过 3,000 元的部分          3              0
        2       超过 3,000 元至 12,000 元的部分  10              210
        3       超过 12,000 元至 25,000 元的部分  20              1410
        4       超过 25,000 元至 35,000 元的部分  25              2660
        5       超过 35,000 元至 55,000 元的部分  30              4410
        6       超过 55,000 元至 80,000 元的部分  35              7160
        7       超过 80,000 元的部分           45            15160
         */
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("请输入月薪：");
            double slary = scanner.nextInt();//月薪
            double jiao = slary-5000;//应纳税所得额（各项社会保险费=0）
            double shui = 0;//应纳税额
            if (jiao<0){
                System.out.println("个税起征点为 5000 元，不需要纳税");
            }else if(jiao<=3000){
                shui=jiao*0.03;
                slary-=shui;

            }else if (jiao<=12000){
                shui=jiao*0.1-210;
                slary-=shui;
            }else if (jiao<=25000){
                shui=jiao*0.2-1410;
                slary-=shui;
            }else if (jiao<=35000){
                shui=jiao*0.25-2660;
                slary-=shui;
            }else if (jiao<=55000){
                shui=jiao*0.3-4410;
```

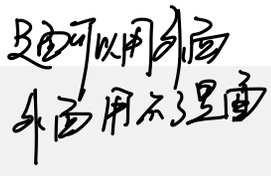
```
                slary-=shui;
            }else if (jiao<=80000){
                shui=jiao*0.35-7160;
                slary-=shui;
            }else {
                shui=jiao*0.45-15160;
                slary-=shui;
            }
            System.out.println("应纳税所得额："+jiao+"元\t"+"纳税税额"+shui+"元
\t"+"实发工资"+slary+"元");
            System.out.println("输入 exit 退出程序！其他继续计算！");
            int cmd =scanner.nextInt();
            if(cmd==88){
                System.out.println("程序结束，退出！");
                break;
            }else{
                continue;
            }
        }
    }
}
```

方法

语句块

语句块（也叫复合语句）。语句块中定义的变量只能用于自己，外部不能使用。语句块可以使用外部的变量，而外部不能使用语句块的变量；

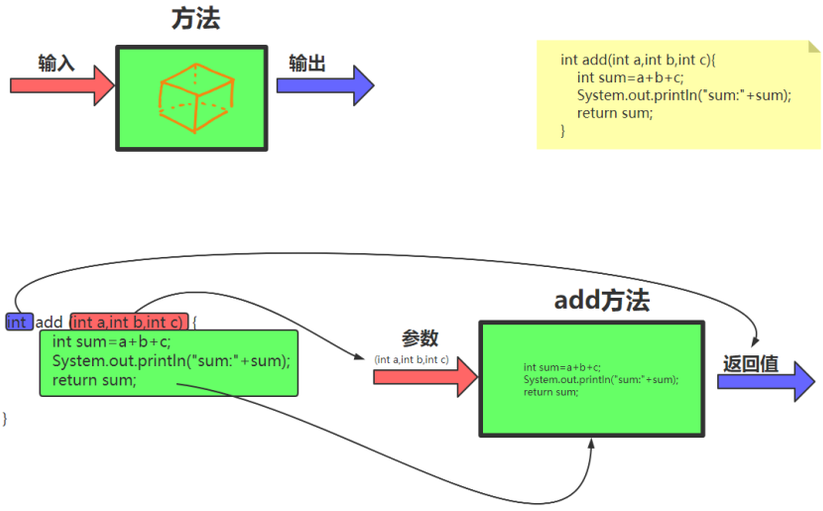
【示例】语句块



```
public class Test19 {
    public static void main(String[ ] args) {
        int n;
        int a;
        {
            int k;
            int n; //编译错误：不能重复定义变量 n
        } //变量 k 的作用域到此为止
    }
```

```
}
```

方法



1. 方法(method)：一段用来完成特定功能的代码片段，类似于其它语言的函数(function)。

2. 方法用于定义该类或该类的实例的行为特征和功能实现。

3. 面向过程中，函数是最基本单位，整个程序由一个个函数调用组成。

4. 面向对象中，整个程序的基本单位是类，方法是从属于类和对象的。

方法声明格式：

```
[修饰符 1　修饰符 2　...]　返回值类型　方法名(形式参数列表){
        Java 语句；... ... ...
}
```

方法的调用方式：

普通方法	对象名.方法名(实参列表)
静态方法	类名.方法名(实参列表)

方法的详细说明

- **形式参数**：在方法声明时用于接收外界传入的数据。（方法定义时）
- **实参**：调用方法时实际传给方法的数据。（方法调用时）
- **返回值**：执行完毕后，返还给调用它的环境的数据。

- **返回值类型：**事先约定的返回值的数据类型，如无返回值，则为 void。

【示例】方法的声明及调用

```
public class Test20 {
    /** main 方法：程序的入口 */
    public static void main(String[ ] args) {
        int num1 = 10;
        int num2 = 20;
        //调用求和的方法：将 num1 与 num2 的值传给 add 方法中的 n1 与 n2
        // 求完和后将结果返回，用 sum 接收结果
        int sum = add(num1, num2);
        System.out.println("sum = " + sum);//输出：sum = 30
        //调用打印的方法：该方法没有返回值
        print();
    }
    /** 求和的方法 */
    public static int add(int n1, int n2) {
        int sum = n1 + n2;
        return sum;//使用 return 返回计算的结果
    }
    /** 打印的方法 */
    public static void print() {
        System.out.println("北京尚学堂...");
    }
}
```

执行结果如图所示：

```
sum = 30
北京尚学堂...
```

运行效果图

注意事项

- 实参必须和形参列表匹配。
- return：终止方法运行，并返回的数据。
- Java 中传递参数，遵循值传递的原则(传递的都是数据的副本)：
  - 基本类型传递的是该数据值的 copy 值。
  - 引用类型传递的是该对象引用的 copy 值，但指向的是同一个对象。

方法的重载(overload)

重载：一个类中可以定义多个名称相同，但形式参数列表不同的方法。

> **新手雷区**
>
> 重载的方法，实际是完全不同的方法，只是名称相同而已！

构成方法重载的条件：

1. 形参列表不同的含义：形参类型、形参个数、形参顺序不同
2. 只有返回值不同不构成方法的重载

   如：int a(String str){}与 void a(String str){}不构成方法重载
3. 只有形参的名称不同，不构成方法的重载

   如：int a(String str){}与 int a(String s){}不构成方法重载

【示例】方法重载

```
public class Test21 {
    public static void main(String[ ] args) {
        System.out.println(add(3, 5));// 8
        System.out.println(add(3, 5, 10));// 18
        System.out.println(add(3.0, 5));// 8.0
        System.out.println(add(3, 5.0));// 8.0
        // 我们已经见过的方法的重载
        System.out.println();// 0 个参数
        System.out.println(1);// 参数是 1 个 int
        System.out.println(3.0);// 参数是 1 个 double
    }
    /** 求和的方法 */
```

```
    public static int add(int n1, int n2) {
        int sum = n1 + n2;
        return sum;
    }
    // 方法名相同，参数个数不同，构成重载
    public static int add(int n1, int n2, int n3) {
        int sum = n1 + n2 + n3;
        return sum;
    }
    // 方法名相同，参数类型不同，构成重载
    public static double add(double n1, int n2) {
        double sum = n1 + n2;
        return sum;
    }
    // 方法名相同，参数顺序不同，构成重载
    public static double add(int n1, double n2) {
        double sum = n1 + n2;
        return sum;
    }
    //编译错误：只有返回值不同，不构成方法的重载
    public static double add(int n1, int n2) {
        double sum = n1 + n2;
        return sum;
    }
    //编译错误：只有参数名称不同，不构成方法的重载
    public static int add(int n2, int n1) {
        double sum = n1 + n2;
        return sum;
    }
}
```

课堂练习

1、 定义一个方法处理公司的迟到问题：

(1) 输入：迟到时间，月薪。

(2) 处理逻辑：

① 迟到 1-10 分钟，警告。

② 迟到 11-20 分钟，罚款 100 元。

③ 迟到 21 分钟-30 分钟，罚款 200 元。

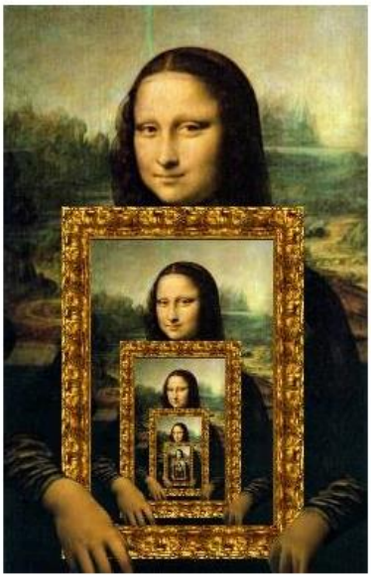
④ 迟到 30 分钟以上，扣除半日工资。

⑤ 迟到 1 小时以上，按照旷工计算，扣除 3 日工资。

(3) 输出：罚款金额

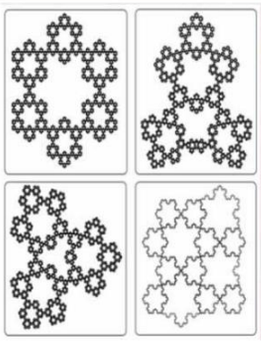
```
public class TestMethod2 {



    /**
     * (1)输入参数：迟到时间，月薪。
     * (2)处理逻辑：
     * ①迟到1-10 分钟，警告。
     * ②迟到11-20 分钟，罚款100 元。
     * ③迟到21 分钟-30 分钟，罚款200 元。
     * ④迟到30 分钟以上，扣除半日工资。
     * ⑤迟到1 小时以上，按照旷工计算，扣除3 日工资。
     * (3)输出罚款金额
     */
    public static int late(int lateMinute,double salary){
        int  fakuan = 0;        //罚款

        if(lateMinute<11){
            System.out.println("警告！！不能迟到！！");
        }else if(lateMinute<21){
            fakuan = 100;
        }else if(lateMinute<31){
            fakuan = 200;
        }else if(lateMinute<61){
            fakuan = (int)(salary/(21.75*2));      //21.75 指的
是：月平均工作日
        }else{
            fakuan = (int)(salary*3/(21.75));
        }

        System.out.println("迟到时间："+lateMinute+",罚款：
"+fakuan);

        return fakuan;

    }

    public static void main(String[] args) {
        late(45,42000);
    }

}
```

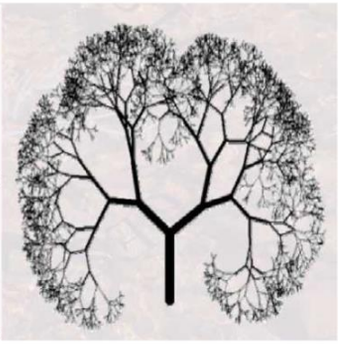
递归结构



1. 递归是一种常见的算法思路，在很多算法中都会用到。比如：深度优先搜索(DFS:*Depth First Search*）等。

2. 递归的基本思想就是“自己调用自己”。





递归结构包括两个部分：

- **定义递归头。** 解决：什么时候不调用自身方法。如果没有头，将陷入死循环，也就是递归的结束条件。
- **递归体。** 解决：什么时候需要调用自身方法。

【示例】使用递归求 n!

```
public class Test22 {
    public static void main(String[ ] args) {
        long d1 = System.currentTimeMillis();
        factorial(10);
        long d2 = System.currentTimeMillis();
```

```
            System.out.printf("递归费时:"+(d2-d1));   //耗时：32ms
        }
        /** 求阶乘的方法*/
        static long   factorial(int n){
            if(n==1){//递归头
                return 1;
            }else{//递归体
                return n*factorial(n-1);//n! = n * (n-1)!
            }
        }
}
```

执行结果如图所示：

```
10阶乘的结果：3628800
递归费时：558
```

执行结果如图所示：

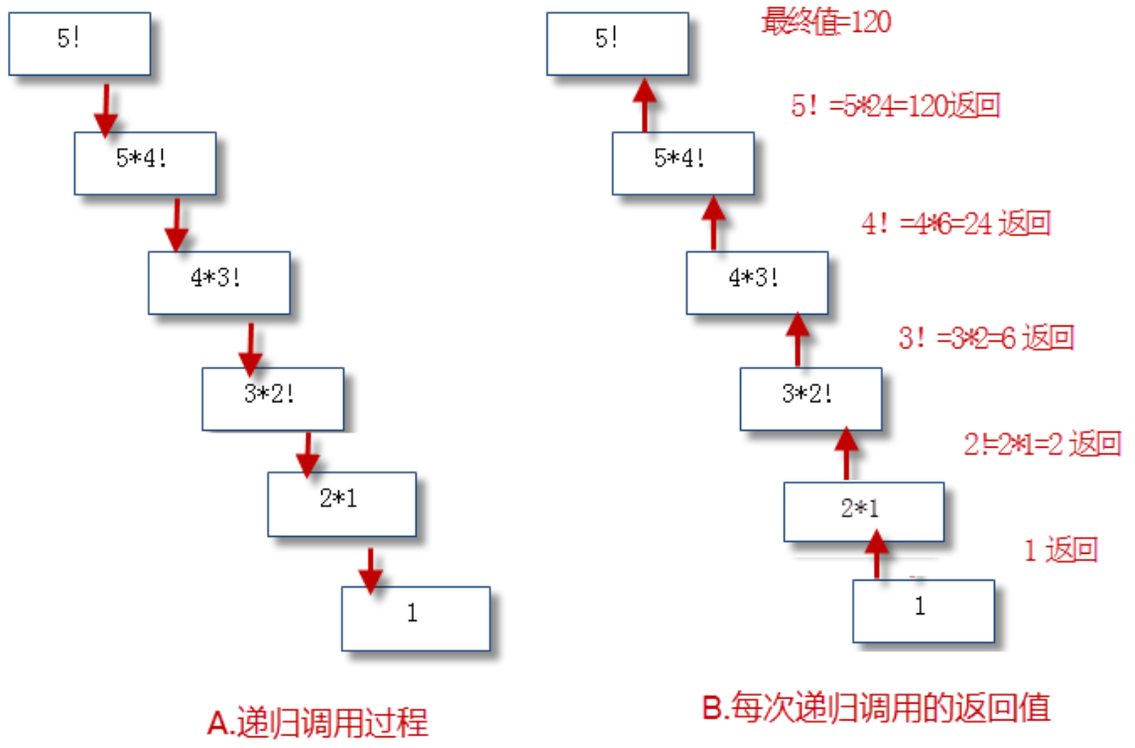


图 3-28 递归原理分析图

递归的缺陷

算法简单是递归的优点之一。但是递归调用会占用大量的系统堆栈，内存耗用多，在递归调用层次多时速度要比循环慢的多，所以在使用递归时要慎重。

比如上面的递归耗时 558ms(看电脑配置)。但是用普通循环的话快得多，如下所示。

【示例】使用循环求 n!

```
public class Test23 {
    public static void main(String[ ] args) {
        long d3 = System.currentTimeMillis();
        int a = 10;
        int result = 1;
        while (a > 1) {
            result *= a * (a - 1);
            a -= 2;
        }
        long d4 = System.currentTimeMillis();
        System.out.println(result);
        System.out.printf("普通循环费时："+(d4 - d3));
    }
}
```

执行结果如图 3-29 所示：

```
3628800
普通循环费时：0
```