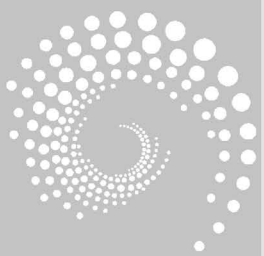
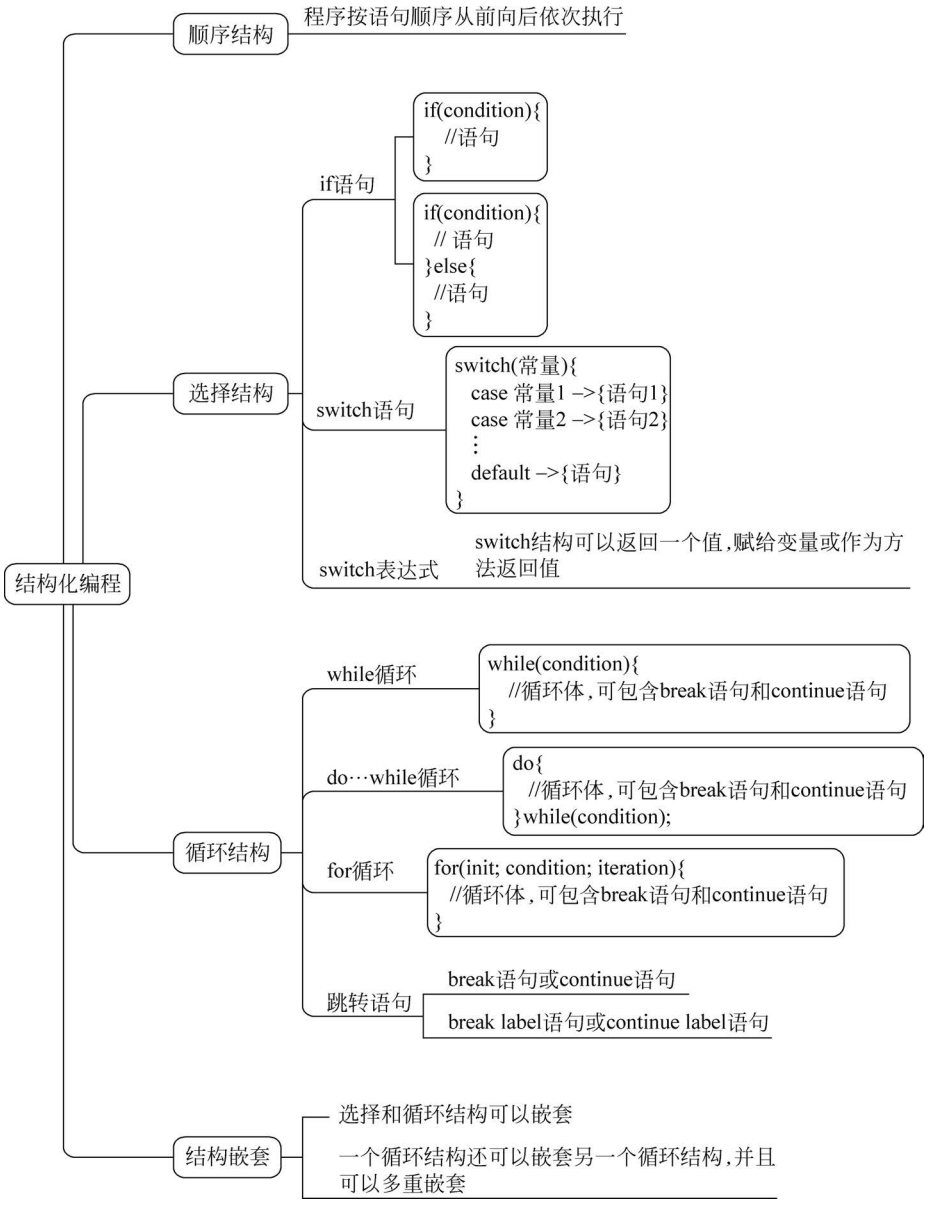
第3章

结构化编程

CHAPTER 3



本章知识点思维导图



扫一扫

视频讲解

3.1　编程方法

随着计算机硬件技术的不断发展,编程方法也在不断地发展和改进。有多种编程方法,比如结构化编程方法、面向对象编程方法、函数式编程、反应式编程等。本节简要介绍结构化编程的基本概念。

结构化编程(structured programming)方法在 1965 年提出,是软件发展的一个重要的里程碑。在结构化编程中,只允许使用 3 种基本的程序结构,它们是顺序结构、分支结构(包

括多分支结构)和循环结构,这 3 种基本结构的共同特点是只允许有一个入口和一个出口,仅由这 3 种基本结构组成的程序称为**结构化程序**。

- **顺序结构**。顺序结构表示程序中的各操作是按照它们出现的先后顺序执行的。如图 3-1 所示,其中 A、B 可以是一个语句(甚至是空语句),也可以是这里介绍的 3 种结构中的一种结构。顺序结构比较简单,程序按语句的顺序依次执行。在前 2 章中介绍的程序都是顺序结构的。

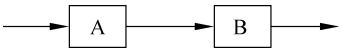


图 3-1 顺序结构

- **选择结构**。选择结构是指程序的处理步骤出现了分支,它需要根据某一特定的条件选择其中的一个分支执行。简单的选择结构如图 3-2(a)和图 3-2(b)所示,这里 P 表示谓词条件。它们通常称为单分支、双分支。图 3-2(c)一般称为多分支,它可以通过嵌套的图 3-2(b)得到。

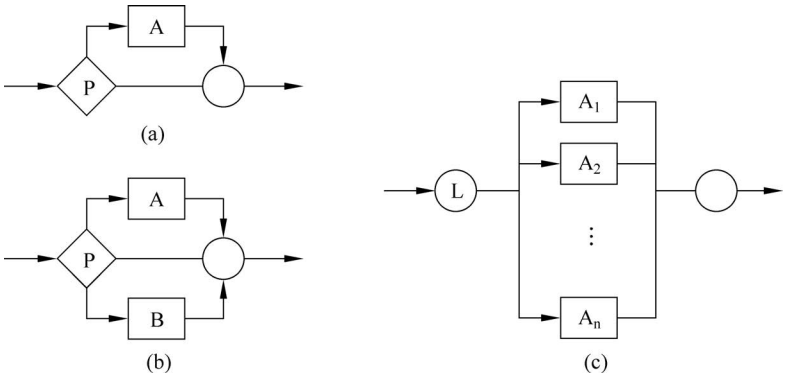


图 3-2 选择结构

- **循环结构**。循环结构是指程序反复执行某个或某些操作,直到某条件为假(或为真)时才可终止循环。在循环结构中最主要的是:什么情况下执行循环?哪些操作需要执行循环?循环结构有 3 种,如图 3-3 所示。这 3 种结构依次为 While 循环、Repeat 循环和 N+1/2 循环。前两种循环可以看作第 3 种循环的特殊情形。这 3 种循环虽然形式不同,但它们有一个共同点,即在一定条件下,可以将控制转移到本结构的入口点。

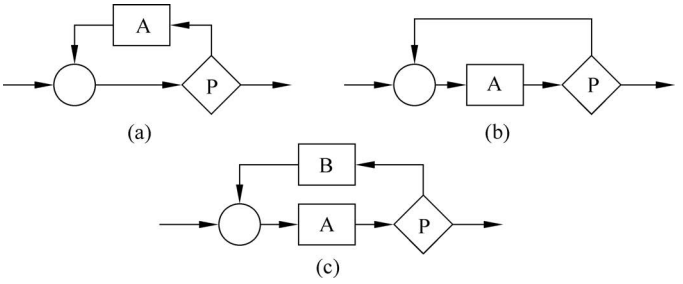


图 3-3 循环结构

按照结构化程序设计的观点,任何算法功能都可以通过由程序模块组成的 3 种基本程序结构的组合来实现。

采用结构化编程方法的软件开发需要遵循的主要原则包括:

- 自顶向下。设计程序时，应先考虑总体，后考虑细节；先考虑全局目标，后考虑局部目标。不要一开始就过多追求众多的细节，先从最上层的总目标开始设计，逐步使问题具体化。
- 逐步求精。对复杂问题，通常应设计一些子目标作为过渡，然后逐步求精。
- 模块化设计。一个复杂问题，是由若干简单的问题构成的。模块化是把程序要解决的总目标分解为子目标，再进一步分解为具体的小目标，每一个小目标被称为一个模块。
- 限制 goto 语句使用。很多编程语言支持 goto 语句，但使用 goto 语句是有害的，是造成程序混乱的根源，程序的质量与 goto 语句的数量成反比。

Java 是面向对象编程语言，它支持面向对象基本特征，同时它也吸收了结构化编程的思想精华，支持结构化编程方法。本章主要讨论 Java 对结构化编程的支持。关于 Java 面向对象的概念，我们从第 4 章开始讨论。

扫一扫

视频讲解

3.2　选择结构

3.1 节介绍了结构化编程的 3 种基本结构，其中顺序结构比较简单，即程序按语句的顺序依次执行。前面章节编写的程序都是顺序结构的，本章将重点讨论选择结构和循环结构。

Java 有多种类型的选择语句：简单 if 语句、双分支 if…else 语句、嵌套 if 语句、多分支 if…else 语句、switch 语句和条件表达式。

3.2.1　简单 if 语句

简单 if 语句的格式如下：

```
if (条件){
      语句(组);
}
```

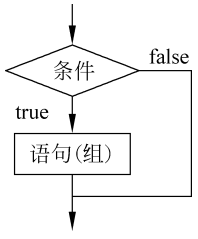


图 3-4　if 语句的执行流程

其中，条件为一个布尔表达式，它的值为 true 或 false。布尔表达式应该使用括号括住，后面是一对花括号。程序执行的流程是：首先计算条件表达式的值，若其值为 true，则执行语句(组)语句序列，否则转去执行 if 结构后面的语句，如图 3-4 所示。

下面程序 3-1，要求用户从键盘输入两个整数，分别存入变量 a 与 b，如果 a 大于 b，则交换 a 和 b 的值，也就是保证 a 小于或等于 b，最后输出 a 和 b 的值。

程序 3-1　ExchangeDemo.java

```
package com.boda.xy;
import java.util.Scanner;
public class ExchangeDemo{
   public static void main(String[] args){
      Scanner input = new Scanner(System.in);
      System.out.print("请输入整数 a:");
      int a = input.nextInt();
```

```
        System.out.print("请输入整数 b:");
        int b = input.nextInt();
        if(a > b){
            int t = b;
            b = a;          ←  交换 a 与 b 的值
            a = t;
        }
        System.out.println(" a = " + a) ;
        System.out.println(" b = " + b) ;
    }
}
```

执行程序输入 a 为 30,b 为 20,运行结果如图 3-5 所示。

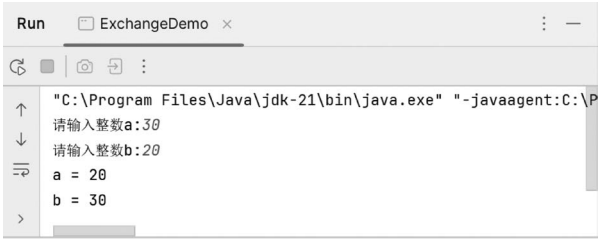


图 3-5 程序 3-1 的运行结果

注意 在 if 语句中,如果花括号内只有一条语句,则可以省略花括号。省略花括号可以使代码更简洁,但也容易产生错误。将来需要为代码块增加语句时,容易忘记加上花括号。这是初学者常犯的错误。

3.2.2 双分支 if…else 语句

双分支 if…else 语句是最常用的选择结构,它根据条件是 true 还是 false,决定执行的路径。if…else 结构的一般格式如下:

```
if (条件){
     语句(组)1;
}else{
     语句(组)2;
}
```

该结构的执行流程是:首先计算条件的值,如果为 true,则执行语句(组)1,否则执行语句(组)2,如图 3-6 所示。当 if 或 else 部分只有一条语句时,花括号可以省略,但推荐使用花括号。

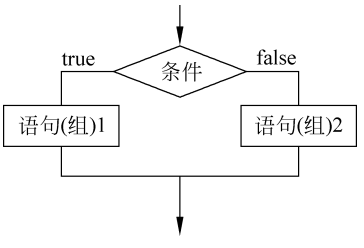


图 3-6 双分支 if…else 语句的执行流程

下面程序 3-2,要求用户从键盘输入一个年份,输出该年是否是闰年。符合下面两个条件之一的年份即为闰年:①能被 400 整除;②能被 4 整除,但不能被 100 整除。

程序 3-2　LeapYear.java

```
package com.boda.xy;
import java.util.Scanner;
public class LeapYear{
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        System.out.print("请输入年份:");
        int year = scan.nextInt() ;
        if(year % 400 == 0||(year % 4 == 0 && year % 100 != 0)){
            System.out.println(year + " 年是闰年。") ;
        }else{
            System.out.println(year + " 年不是闰年。") ;
        }
    }
}
```

运行程序输入年份 2024 的结果如图 3-7 所示。

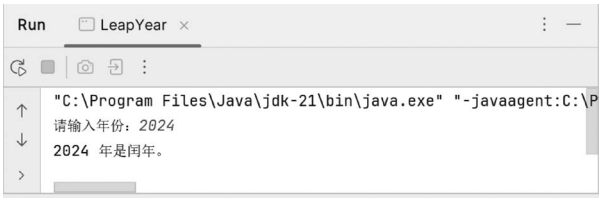


图 3-7　程序 3-2 的运行结果

3.2.3　多分支 if…else 语句

if 或 if…else 结构中的语句可以是任意合法的 Java 语句，甚至可以是其他的 if 或 if…else 结构。内层的 if 结构称为嵌套在外层的 if 结构。内层的 if 结构还可以包含其他的 if 结构。嵌套的深度没有限制。例如，下面就是一个嵌套的 if 结构，其功能是求 a、b 和 c 中最大值并将其保存到 max 中。

```
if(a > b){
   if( a > c)
       max = a;
   else                ←——  一个嵌套的 if…else 结构
       max = c;
}else{
   if( b > c)
       max = b;
   else                ←——  一个嵌套的 if…else 结构
       max = c;
}
```

注意，把每个 else 同与它匹配的 if 对齐排列，这样做很容易辨别嵌套层次。

如果程序逻辑需要多个选择，可以在 if 语句中使用一系列的 else if 语句，这种结构称为**多分支 if…else 语句**或阶梯式 if…else 结构。

编写程序，要求输入一个人的身高和体重，计算并打印出他的 BMI，同时显示 BMI 是高

还是低。对于一个成年人,BMI 值的含义如下所示。

- BMI<18.5,表示偏瘦;
- 18.5≤BMI<25.0,表示正常;
- 25.0≤BMI<30.0,表示超重;
- BMI≥30.0,表示过胖。

程序 3-3 ComputeBMI.java

```
package com.boda.xy;
import java.util.Scanner;
public class ComputeBMI{
   public static void main(String[]args){
      Scanner input = new Scanner(System.in);
      double weight,height;
      double bmi;
      System.out.print("请输入你的体重(单位:kg):");
      weight = input.nextDouble();
      System.out.print("请输入你的身高(单位:m):");
      height = input.nextDouble();
      bmi = weight / (height * height);
      System.out.println("你的身体质量指数是:" + bmi);
      if(bmi < 18.5){
          System.out.println("你的体重偏瘦。");
      }else if(bmi < 25.0) {
          System.out.println("你的体重正常。");
      }else if(bmi < 30.0) {
          System.out.println("你的体重超重。");
      }else {
          System.out.println("你的体重过胖。");
      }
   }
}
```

程序的一次运行结果如图 3-8 所示。

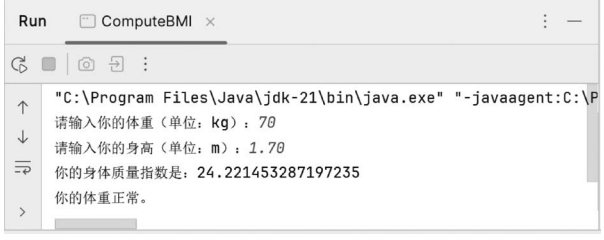


图 3-8 程序 3-3 的运行结果

3.2.4 条件运算符

条件运算符(conditional operator)的格式如下:

```
<条件> ? <表达式 1> : <表达式 2>
```

因为有三个操作数,又称为**三元运算符**。这里<条件>为关系或逻辑表达式,其计算结

果为布尔值。如果该值为 true，则计算<表达式 1 >的值，并将计算结果作为条件表达式的结果；如果该值为 false，则计算<表达式 2 >的值，并将计算结果作为条件表达式的结果。

条件运算符可以实现 if…else 结构。例如，若 max、a 和 b 是 int 型变量，要计算 a 与 b 的最大值 max，下面左侧的条件运算符结构与右侧使用的 if…else 结构等价。

```
max = (a > b)? a : b ;        等价于        if (a > b) {
                                                max = a;
                                            }else {
                                                max = b;
                                            }
```

从上面可以看到使用条件运算符会使代码简洁，但是不容易理解。现代的编程，程序的可读性变得越来越重要，因此推荐使用 if…else 结构，毕竟并没有多输入多少代码。

扫一扫

视频讲解

3.3　案例学习——两位数加减运算

1. 问题描述

开发一个让小学生练习两位整数加减法的程序，要求程序运行随机生成两个两位数及加减号（要保证减法算式的被减数大于减数），显示题目让学生输入计算结果，程序判断结果是否正确。

2. 运行结果

案例的运行结果如图 3-9 所示，这里产生一个减法题目。

图 3-9　案例的运行结果

3. 设计思路

该案例的设计思路主要如下。

（1）要实现加减法运算，首先应该随机产生两个两位整数。随机生成整数有多种方法，可以使用 Math.random()方法生成一个随机浮点数，然后将它扩大再取整。random()方法返回 0.0～1.0（不包括）的浮点数，要得到 10～99 的整数，可以使用下面表达式：

```
int number1 = 10 + (int)(Math.random() * 90);
```

（2）确定加或减运算。这也可以通过产生 2 个随机数（比如，0 和 1，0 表示加法，1 表示减法）确定。

```
int operator = (int)(Math.random() * 2);
```

(3) 设学生没有学过负数概念,如果做减法运算,要保证第一个数大于第二个数。也就是如果 number1 小于 number2,应该交换这两个数。

```
if(number1 < number2){
    int temp = number2;
    number2 = number1;
    number1 = temp;
}
```

(4) 最后根据运算符决定做何种运算。将计算结果保存到 result 变量中,然后与用户输入的答案 answer 比较,判断用户答题是否正确。

4. 代码实现

两位数加减法运算的代码如程序 3-4 所示。

程序 3-4 Calculator.java

```
package com.boda.xy;
import java.util.Scanner;
public class Calculator{
   public static void main(String[] args) {
       Scanner input = new Scanner(System.in);
       int number1 =  10 + (int)(Math.random() * 90);
       int number2 =  10 + (int)(Math.random() * 90);
       int operator =  (int)(Math.random() * 2);
       int result, answer;
       if(operator == 0){                    ←┤ operator 为 0 表示加法
           result = number1 + number2;
           System.out.print(number1 + " + " + number2 + " = ");
       }else{                                ←┤ operator 为 0 表示减法
          if(number1 < number2){
            int temp = number2;              ←┐
            number2 = number1;               ←┤ 交换 number1 和 number2
            number1 = temp;                  ←┘
          }
         result = number1 - number2;
         System.out.print(number1 + " - " + number2 + " = ");
       }
       answer = input.nextInt();             ←┤ 接收用户输入答案
       if(answer == result){
           System.out.print("恭喜你,回答正确!");
       }else{
           System.out.print("对不起,回答错误!");
       }
    }
}
```

✍多学一招

要产生随机数除了可以使用 Math.random()方法外,Java 语言还提供了一个 java.util.Random 类,使用该类的实例也可以随机产生整数、浮点数或布尔型值。如下面代码随

机产生一个两位整数。

```
Random rand = new Random();
int number = 10 + rand.nextInt(90);
```

Random 类除定义了 nextInt()方法，还定义了 nextBoolean()、nextDouble()、nextLong()、等方法，分别产生随机布尔值、浮点数和长整数等。

扫一扫

视频讲解

3.4　switch 语句与 switch 表达式

Java 语言从一开始就提供了 switch 语句实现多分支结构。从 Java 12 开始对 switch 语句进行了修改并支持 switch 表达式。尽管 Java 仍然支持旧的 switch 结构，但建议读者熟悉并使用新的 switch 语句和 switch 表达式。

3.4.1　switch 语句

如果需要从多个选项中选择其中一个，可以使用 switch 语句。switch 语句主要实现多分支结构，一般格式如下：

```
switch (表达式){
   case 值 1 ->  语句(组)1;
   case 值 2 ->  语句(组)2;
   …
   case 值 n ->  语句(组)n;
   [default ->  语句(组)n + 1;]
}
```

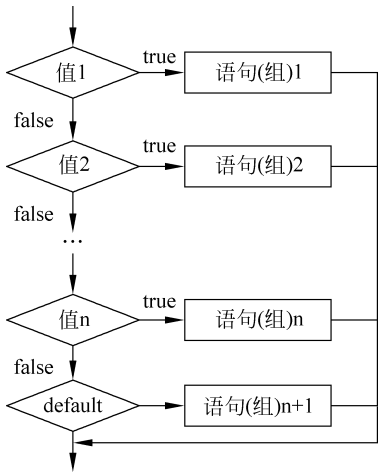


图 3-10　switch 语句的执行流程

switch 语句表达式的值必须是 byte、short、int、char、enum 或 String 类型。case 子句用来设定每一种情况，后面的值必须与表达式值类型相容。switch 语句的执行流程如图 3-10 所示。

当程序执行到 switch 语句时，首先计算表达式的值，然后用该值依次与每个 case 中的常量(或常量表达式)值比较，如果等于某个值，则执行该 case 子句中后面的语句，之后结束 switch 结构。每个 case 后面使用箭头号(->)指定要执行的语句，这里可以是一条语句，也可以包含多条语句。如果包含多条语句，需使用花括号。

default 子句是可选的，当表达式的值与每个 case 子句中的值都不匹配时，就执行 default 后的语句。如果表达式的值与每个 case 子句中的值都不匹配，且又没有 default 子句，则程序不执行任何操作，而是直接跳出 switch 结构，执行后面的语句。

用 switch 结构可以实现多重选择。下面程序 3-5 要求从键盘输入一个季节数字(1,2,3,4)，程序根据输入的数输出一句话。

程序 3-5 SwitchDemo.java

```
package com.boda.xy;
import java.util.Scanner;
public class SwitchDemo{
   public static void main(String[] args) {
       Scanner input = new Scanner(System.in);
       System.out.print("输入一个季节(1,2,3,4):");
       int season = input.nextInt();
       switch (season) {
           case 1 -> System.out.println("春雨惊春清谷天");
           case 2 -> System.out.println("夏满芒夏暑相连");
           case 3 -> System.out.println("秋处露秋寒霜降");
           case 4 -> System.out.println("冬雪雪冬小大寒");
           default ->
                System.out.println("季节输入非法.");
         }
    }
}
```

图 3-11 是程序的一次运行结果。

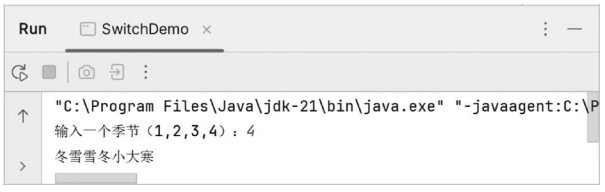


图 3-11 程序 3-5 的运行结果

从 Java 7 开始,可以在 switch 语句的表达式中使用 String 对象,下面程序 3-6 根据输入的英文季节名称(Spring、Summer、Autumn 和 Winter)输出中文季节名。

程序 3-6 StringSwitchDemo.java

```
package com.boda.xy;
import java.util.Scanner;
public class StringSwitchDemo {
   public static void main(String[] args) {
        String season = "";
        Scanner input = new Scanner(System.in);
        System.out.print("请输入英文季节名称:");
        season = input.next();
        switch (season.toLowerCase()) {
            case "spring" -> System.out.println("春天");
            case "summer" -> System.out.println("夏天");
            case "autumn" -> System.out.println("秋天");
            case "winter" -> System.out.println("冬天");
            default       -> System.out.println("输入名称错误!");
        }
    }
}
```

运行结果如图 3-12 所示。

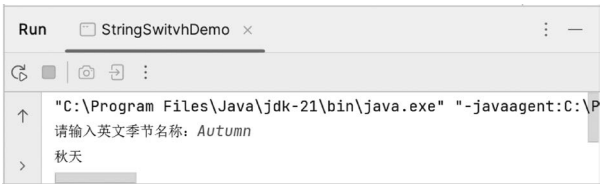


图 3-12　程序 3-6 的运行结果

程序中的 season.toLowerCase()是将字符串转换成小写字符串。switch 表达式中的字符串与每个 case 中的字符串进行比较。

3.4.2　switch 表达式

在 switch 结构中，除了可以执行 switch 语句外，还可以使用 switch 表达式，即通过 switch 结构返回一个值，并将该值赋给变量。例如，下面代码根据 day 的值返回一个数值赋给变量 numLetters。

```
DayOfWeek day = DayOfWeek.SATURDAY;
int numLetters = switch(day){
   case MONDAY, FRIDAY,SUNDAY -> 6;
   case TUESDAY               -> 7;          ←┤ 根据表示星期的枚举常量返回单词的字母数
   case THURSDAY, SATURDAY    -> 8;
   case WEDNESDAY             -> 9;
  };       ←┤ 分号是赋值语句的结束
System.out.println(numLetters);          // 输出 8
```

下面程序 3-7 要求从键盘输入一个年份(如 2000 年)和一个月份(如 2 月)，用 switch 表达式返回该月的天数(29)，将其存入一个变量。

程序 3-7　SwitchExprDemo.java

```
package com.boda.xy;
import java.util.Scanner;
public class SwitchExprDemo {
   public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.print("输入一个年份:");
      int year = input.nextInt();
      System.out.print("输入一个月份:");
      int month = input.nextInt();
      int numDays = switch (month) {          ←┤ switch 表达式
         case 1, 3, 5, 7, 8, 10, 12 -> 31;
         case 4, 6, 9, 11 -> 30;
         // 对 2 月需要判断是否是闰年
         case 2 -> {
            if(((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0))
               yield 29;                      ←┤ yield 是受限标识符，生成一个值
            else
               yield 28;
         }
         default -> 0;
      };                                      ←┤ 分号是赋值语句的结束
```

```
            System.out.println("该月的天数为:" + numDays);
        }
    }
```

图 3-13 是程序的一次运行结果。

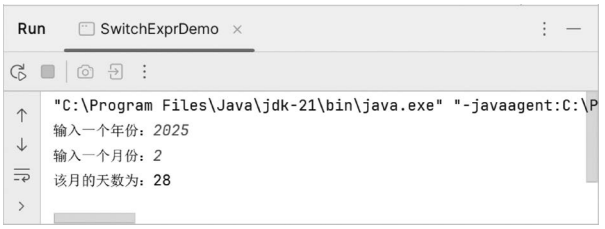


图 3-13 程序 3-7 的运行结果

yield 关键字用于跳出 switch 块,主要用于返回一个值。yield 和 return 的区别在于:return 会直接跳出当前循环或方法,而 yield 只会跳出当前 switch 块。

另外注意,如果某个 case 直接返回一个值,不能使用 yield 关键字,但可以在花括号中使用,如下所示:

```
case 4, 6, 9, 11 ->{ yield 30;}
```

switch 表达式还可以用在方法返回值中,如下方法所示:

```
private static String descLang(String name){
    return switch(name){
        case "Java"   ->{yield  "object-oriented, platform independent.";}
        case "Ruby"   ->{yield  "a programmer's best friend.";}
        default       ->{yield  name + " is a good language.";}
    };
}
```

注意 在使用 switch(expr)表达式时,case 子句中必须包含 expr 的所有可能的值,否则必须带一个 default 子句。

Java 仍然支持传统的 switch 语法结构,传统的 switch 结构如下:

```
switch (expression){
   case 值 1: 语句(组)1; [break;]
   case 值 2: 语句(组)2; [break;]
   …
   case 值 n: 语句(组)n; [break;]
   [default:   语句(组)n+1;]
}
```

← 不建议使用这种结构

使用这种语法结构很容易忘记 break 语句而产生错误,建议使用新的 switch 结构。

扫一扫

视频讲解

3.5 循环结构

在程序设计中,有时需要反复执行一段相同的代码,这时就需要使用循环结构来实现。Java 语言提供了 4 种循环结构:while 循环、do…while 循环、for 循环和增强的 for 循环。

一般情况下，一个循环结构包含如下 4 部分内容。

(1) 初始化部分：设置循环开始时的变量初值。

(2) 循环条件：一般是一个布尔表达式，当表达式值为 true 时执行循环体，为 false 时退出循环。

(3) 迭代部分：改变变量的状态。

(4) 循环体：需要重复执行的代码。

3.5.1　while 循环

while 循环是 Java 最基本的循环结构，这种循环是在某个条件为 true 时，重复执行一个语句或语句块。它的一般格式如下：

```
[初始化部分]
while(条件){
     // 循环体
    [迭代部分]
}
```

花括号内为循环体

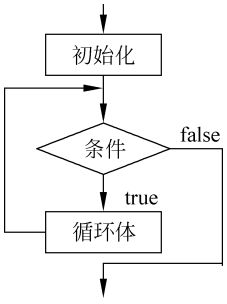


图 3-14　while 循环的执行流程

其中，初始化部分通常定义变量并赋初值，条件为一个布尔表达式，它是循环条件。中间的部分为循环体，用一对花括号定界。迭代部分也是可选的。

该循环首先判断循环条件，当条件为 true 时，一直反复执行循环体。这种循环一般称为“当循环”。一般用在循环次数不确定的情况下。while 循环的执行流程如图 3-14 所示。

下面程序 3-8 使用 while 循环计算 1 到 100 之和。

程序 3-8　WhileDemo.java

```
package com.boda.xy;
public class WhileDemo{
   public static void main(String[] args){
      int n = 1;                                    ← 初始化部分
      int sum = 0;
      while(n <= 100){
          sum = sum + n;
          n = n + 1;                                ← 迭代语句
      }
      System.out.println("sum = " + sum);          // 输出:sum = 5050
   }
}
```

运行结果如图 3-15 所示。

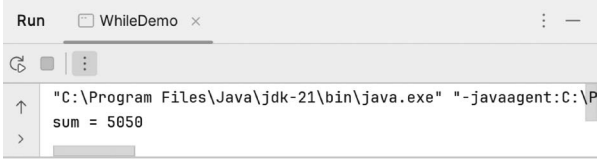


图 3-15　程序 3-8 的运行结果

下面编写一个简单的猜数程序,要求程序运行随机产生一个 1～100 的整数,用户从键盘输入所猜的数,程序显示是否猜中的消息,如果没有猜中要求用户继续猜,直到猜中为止。

程序 3-9 GuessNumber.java

```
package com.boda.xy;
import java.util.Scanner;
public class GuessNumber{
   public static void main(String[] args){
      int magic = (int)(Math.random() * 100) + 1;          ←┤ 随机产生的整数
      Scanner sc = new Scanner(System.in);
      System.out.print("我想出一个 1 - 100 的数,请你猜:");
      int guess = sc.nextInt();                            ←┤ 用户猜的数
      while(guess != magic){
         if(guess > magic)
           System.out.print("对不起!太大了,请重猜:");
         else
           System.out.print("对不起!太小了,请重猜:");
         guess = sc.nextInt();                             ←┤ 输入下一次猜的数
      }
      System.out.println("恭喜你,答对了!\n 该数是:" + magic);
   }
}
```

运行结果如图 3-16 所示。

图 3-16 程序 3-9 的运行结果

程序中使用了 java.lang.Math 类的 random()方法,该方法返回一个 0.0 到 1.0(不包括 1.0)的 double 型的随机数。程序中该方法乘以 100 再转换为整数,得到 0 到 99 的整数,再加上 1,则 magic 的范围就为 1 到 100 的整数。

✍多学一招

在上面猜数游戏中,最多猜多少次应该猜中随机生成的数呢?实际上,这个猜数游戏是一个查找过程,并且是二分查找。它是在有序列表中查找指定的数(87),由于每次查找都去掉一半的元素,所以查找的次数最多是 $\log_2 N$,这里 N 是 100,$\log_2 N$ 的结果是 6.6439,所以最多不超过 7 次就应该猜中。

3.5.2 do…while 循环

do…while 循环的一般格式如下。

```
[初始化部分]
do{
```

```
    // 循环体
    [迭代部分]
}while(条件);
```

花括号内为循环体

do…while 循环的执行过程如图 3-17 所示。

该循环首先执行循环体，然后计算条件表达式。如果表达式的值为 true，则返回到循环的开始继续执行循环体，直到条件的值为 false 循环结束。这种循环一般称为“直到型”循环。该循环结构与 while 循环结构的不同之处是，do…while 循环至少执行一次循环体。

下面程序 3-10 用 do…while 循环计算 1 到 100 之和。

程序 3-10　Sum100.java

```
package com.boda.xy;
public class Sum100 {
   public static void main(String[] args) {
      int n = 1;
      int sum = 0;
      do{
         sum = sum + n;
         n = n + 1;
      }while(n <= 100);
      System.out.println("sum = " + sum);    // 输出 sum = 5050
   }
}
```

3.5.3　for 循环

for 循环是 Java 程序中使用最广泛的，也是功能最强的循环结构。它的一般格式如下所示：

```
for (初始化部分; 循环条件; 迭代部分){
    // 循环体
}
```

这里，初始化部分、循环条件和迭代部分用分号隔开，花括号内为循环体。for 循环的执行流程如图 3-18 所示。

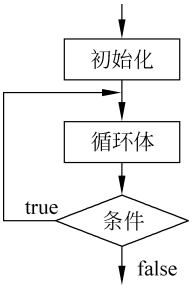


图 3-17　do…while 循环结构

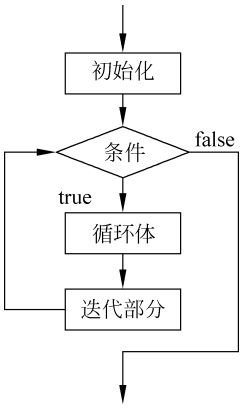


图 3-18　for 循环的执行流程

循环开始时首先执行初始化部分,该部分在整个循环中只执行一次。在这里通常定义循环变量并赋初值。接下来判断循环条件,若为 true 则执行循环体部分,若为 false 则退出循环。循环体执行结束后,控制返回到迭代部分,执行迭代,然后再次判断循环条件,若为 true 则反复执行循环体。

下面代码使用 for 循环计算 1 到 100 之和。

```
int sum = 0;
for(int n = 1; n <= 100; n++){
    sum = sum + n;
}
System.out.println("sum = " + sum);   // 输出:sum = 5050
```

在初始化部分可以声明多个变量,中间用逗号分隔,它们的作用域在循环体内。在迭代部分也可以有多个表达式,中间也用逗号分隔。下面循环中声明了两个变量 i 和 j。

```
for(int i = 0, j = 10 ; i < j ; i++, j--) {
     System.out.println("i = "+ i + " ,j = " + j);
}
```

for 循环中的一部分或全部可为空,循环体也可为空,但分号不能省略,如下所示:

```
for ( ;  ; ){
    // 这实际是一个无限循环,循环体中应包含结束循环代码
}
```

for 循环和 while 循环及 do…while 循环有时可相互转换,例如有下面的 for 循环:

```
for(int i = 0, j = 10 ; i < j ; i++, j--){
    System.out.println("i = "+ i + " ,j = " + j);
}
```

可以转换为下面等价的 while 循环结构:

```
int i = 0, j = 10;
while(i < j){
    System.out.println("i = "+ i + " ,j = " + j) ;
    i ++;
    j -- ;
}
```

提示 在 Java 5 中增加了一种新的循环结构,称为增强的 for 循环,它主要用于对数组和集合元素迭代。关于增强的 for 循环将在 5.1.4 节讨论。

3.5.4 循环的嵌套

在一个循环的循环体中可以嵌套另一个完整的循环结构,称为**循环的嵌套**。内嵌的循环还可以嵌套循环,这就是多层循环。同样,在循环体中也可以嵌套另一个选择结构,选择结构中也可以嵌套循环。

下面编写一个程序,用嵌套的 for 循环打印输出如图 3-19 所示的图形。这里,第 1 行输

出 1 个星号，第 2 行输出 2 个星号，……，第 8 行输出 8 个星号。

图 3-19　输出若干星号

程序 3-11　PrintStars.java

```
package com.boda.xy;
public class PrintStars {
   public static void main(String[] args) {
      // n 记录行数
      for (int n = 1; n <= 8; n++) {
         // 打印每行的前导空格
         for (int k = 1; k <= (8 - n); k++) {
            System.out.print(" ");
         }
         // 每行打印 n 个星号
         for (int j = 1; j <= n; j++) {
            System.out.print("*");
         }
         System.out.println();       // 换行
      }
   }
}
```

这个程序在一个 for 循环内嵌套了 2 个 for 循环，第一个 for 循环用于打印若干个空格，每行打印的空格数为 8－n 个。第二个 for 循环用于打印若干个星号，每行打印 n 个星号。

3.5.5　break 语句和 continue 语句

在 Java 循环体中可以使用 continue 语句和 break 语句。

1. break 语句

break 语句用来结束 while、do、for 结构的执行，该语句有下面 2 种格式：

```
break;
break 标签名;
```

break 语句的功能是结束本次循环，控制转到其所在循环的后面执行。对各种循环均直接退出，不再计算循环控制表达式。

下面程序 3-12 演示了 break 语句的使用。

程序 3-12 BreakDemo.java

```
package com.boda.xy;
public class BreakDemo{
   public static void main(String[] args){
      int n = 1;
      int sum = 0;
      while(n <= 100){
        sum = sum + n;
        if(sum > 100){
          break;                              // 若条件成立退出循环
        }
        n = n + 2;
      }
     System.out.println("n = " + n);          // 输出:n = 21
     System.out.println("sum = " + sum);      // 输出:sum = 121
     }
}
```

使用 break 语句只能跳出当前的循环体。如果程序使用了多重循环,则又需要从内层循环跳出或者从某个循环开始重新执行,此时可以使用带标签的 break。

考虑下面代码:

```
start:                          ←┤ 定义一个标签
for(int i = 0; i < 3; i++){
   for(int j = 0; j < 4; j++){
      if(j == 2){
         break start;           ←┤ 跳出 start 标签标识的循环
      }
      System.out.println(i +":" + j);
   }
}
```

这里,标签 start 用来标识外层的 for 循环,因此语句 break start;跳出了外层循环。上述代码的运行结果如图 3-20 所示。

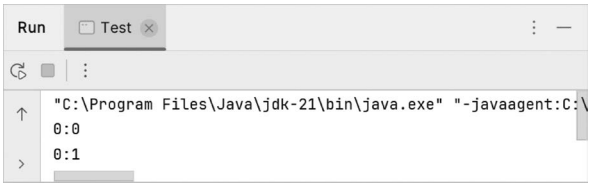


图 3-20 代码的运行结果

2. continue 语句

continue 语句与 break 语句类似,但它只终止执行当前的迭代,导致控制权从下一次迭代开始。该语句有下面两种格式:

```
continue;
continue  标签名;
```

以下代码会输出 0～9 的数字，但不会输出 5。

```
for(int i = 0; i < 10; i++){
   if(i == 5){
       continue;        ←┤ 控制转到迭代部分，即 i++处
   }
   System.out.println(i);
}
```

当 i 等于 5 时，if 语句的表达式运算结果为 true，使得 continue 语句得以执行。因此，后面的输出语句不能执行，控制权从下一次循环处继续，即 i 等于 6 的时候。

continue 语句也可以带标签，用来标识从那一层循环继续执行。下面是使用带标签的 continue 语句的例子。

```
start:                          ←┤ 定义一个标签
for(int i = 0; i < 3; i++){
   for(int j = 0; j < 4; j++){
      if(j == 2){
          continue start;       ←┤ 返回到 start 标签标识的循环的条件处，即 i++处
      }
      System.out.println(i + " : " + j);
   }
}
```

代码的运行结果如图 3-21 所示。

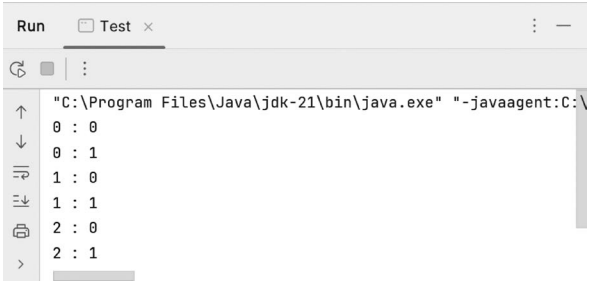


图 3-21　代码的运行结果

注意：

(1) 带标签的 break 语句可用于循环结构和带标签的语句块，而带标签的 continue 语句只能用于循环结构。

(2) 标签命名遵循标识符的命名规则，相互包含的块不能用相同的标签名。

(3) 带标签的 break 和 continue 语句不能跳转到不相关的标签块。

扫一扫

视频讲解

3.6　案例学习——求最大公约数

1. 问题描述

两个整数的**最大公约数**(Greatest Common Divisor，GCD)是能够同时被两个数整除的

最大整数。例如,4 和 2 的最大公约数是 2,16 和 24 的最大公约数是 8。

编写程序,要求从键盘输入两个整数,程序计算并输出这两个数的最大公约数。

2. 运行结果

案例的运行结果如图 3-22 所示,这里输入第 1 个整数 16,第 2 个整数 24,它们的最大公约数为 8。

图 3-22 案例的运行结果

3. 设计思路

求两个整数的最大公约数有多种方法。一种方法是,假设求两个整数 m 和 n 的最大公约数,显然 1 是一个公约数,但它可能不是最大的。可以依次检查 k(k=2,3,4,…)是否是 m 和 n 的最大公约数,直到 k 大于 m 或 n 为止。

该案例的设计思路主要如下:

(1) 从键盘输入两个整数,分别存入变量 m 和变量 n。

(2) 用下面循环结构计算能同时被 m 和 n 整除的数,循环结束后得到的 gcd 就是最大公约数。

```
while(k <= m && k <= n){
    if(m % k == 0 && n % k == 0)
        gcd = k;
    k++;
}
```

4. 代码实现

求最大公约数的代码如程序 3-13 所示。

程序 3-13 GCDDemo.java

```
package com.boda.xy;
import java.util.Scanner;
public class GCDDemo{
   public static void main(String[] args){
      Scanner input = new Scanner(System.in);
      System.out.print("请输入第 1 个整数:");
      int m = input.nextInt();
      System.out.print("请输入第 2 个整数:");
      int n = input.nextInt();
      // 求 m 和 n 的最大公约数
```

```
        int gcd = 1;
        int k = 2;
        while(k <= m && k <= n){
            if(m % k == 0 && n % k == 0)        ←┤ 判断 k 是否能同时被 m 和 n 整除
                gcd = k;
            k++;
        }
        System.out.println(m + " 与 " + n + " 的最大公约数是" + gcd);
    }
}
```

✍**多学一招**

计算两个整数 m 与 n 的最大公约数还有一个更有效的方法，称为辗转相除法或称欧几里得算法，其基本步骤如下：计算 r = m%n，若 r == 0，则 n 是最大公约数。若 r != 0，执行 m = n，n = r，再次计算 r = m%n，直到 r==0 为止，最后一个 n 即为最大公约数。请读者自行编写程序实现上述算法。

扫一扫

视频讲解

3.7　案例学习——打印输出若干素数

1. 问题描述

素数(prime number)又称**质数**，有无限个，在计算机密码学中有重要应用。素数定义为在大于 1 的自然数中，除了 1 和它本身以外不再有其他因数的数。编写程序计算并输出前 50 个素数，每行输出 10 个。

2. 运行结果

案例的运行结果如图 3-23 所示。

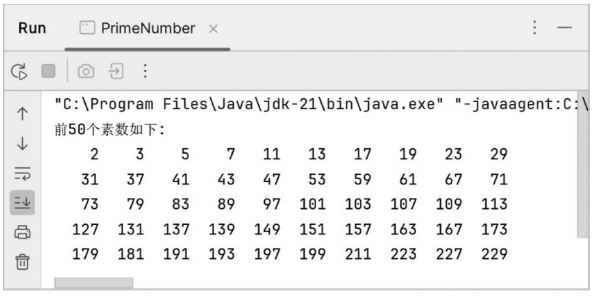


图 3-23　案例的运行结果

3. 设计思路

该案例的设计思路主要如下。

(1) 要输出 50 个素数，首先用 while 循环对素数计数(用 count 变量)。

(2) 要判断的数 number 从 2 开始，这里用一个 for 循环。根据定义，使用从 2 开始的数(divisor)去除要判断的数，如果直到 divisor=number−1 仍不能整除，则 number 就是一

个素数。

(3) 打印输出素数。为了输出美观,可以使用格式化输出方法,这里使用了 System 类的 printf()方法,它可以对输出数据进行格式化。

```
System.out.printf("%5d%n",number);
```

该语句用宽度 5 个字符位置(%5d)输出 number,输出数字右对齐,输出后换行(%n)。

4. 代码实现

计算并输出前 50 个素数的代码如程序 3-14 所示。

程序 3-14 PrimeNumber.java

```
package com.boda.xy;
public class PrimeNumber{
    public static void main(String[] args){
       int count = 0;          // 记录素数个数
       int number = 2;
       boolean isPrime;
       System.out.printf("前 50 个素数如下:%n");
       while(count < 50){
          isPrime = true;
          for(int divisor = 2; divisor < number; divisor ++){
             if(number % divisor == 0){
               isPrime = false;                    判断 number 是否是素数,
               break;                  <---        若循环结束 number 仍不能被 divisor 整除,
             }                                     则 number 是一个素数
           }
           if(isPrime){
             count ++;        <--- 如果 number 是素数,计数器加 1,并打印素数
             if(count % 10 == 0)
                System.out.printf("%5d%n",number);        <--- 格式化输出
             else
                System.out.printf("%5d",number);
          }
          number ++;          <--- 判断下一个数
       }
   }
}
```

说明,这里的循环条件 divisor < number 表示一直检测到 divisor=number-1。实际上,根据数论的理论,判断一个数是否是素数,其因子只需判断到 number 的平方根即可。因此这里的条件也可以写成如下形式:

```
divisor <= Math.sqrt(number)
```

或者

```
divisor * divisor <= number
```

3.8　本章小结

本章首先介绍了结构化编程方法和面向对象编程方法，然后重点介绍了结构化编程的三种控制结构，通过案例演示了这些结构的使用。

下一章将学习面向对象编程的基本概念。主要包括类的定义、对象的创建、方法设计以及对象初始化和销毁。

3.9　习题与实践

扫一扫

习题

扫一扫

自测题

3.10　上机实验

扫一扫

上机实验