

## 第4章

Introduction to Java Programming, 8E

## 循 环

## 学习目标

- 使用while循环编写重复执行某些语句的程序 (4.2节)。
- 开发程序GuessNumber (4.2.1节)。
- 遵循循环设计策略来开发循环 (4.2.2节)。
- 开发程序SubtractionQuizLoop (4.2.3节)。
- 使用标志值控制循环 (4.2.4节)。
- 使用输入重定向而不是从键盘输入以获取大量输入 (4.2.4节)。
- 使用do-while语句编写循环 (4.3节)。
- 使用for语句编写循环 (4.4节)。
- 了解三种类型循环语句的相似处和不同点 (4.5节)。
- 编写嵌套循环 (4.6节)。
- 学习最小化数值误差的技术 (4.7节)。
- 从多种多样的例子 (GCD、FutureTution、MonteCarloSimulation) 中学习循环 (4.8节)。
- 使用break和continue来实现程序的控制 (4.9节)。
- (GUI) 使用确定对话框控制循环 (4.10节)。

## 4.1 引言

假如你需要打印一个字符串 (例如: "Welcome to Java!") 100次, 就需要把下面的输出语句重复写100遍, 这是相当繁琐的:

```

100次 { System.out.println("Welcome to Java!");
        System.out.println("Welcome to Java!");
        ...
        System.out.println("Welcome to Java!");
    
```

那该如何解决这个问题呢?

Java提供了一种称为循环 (loop) 的功能强大的结构, 用来控制一个操作或操作序列重复执行的次数。使用循环语句时, 只要简单地告诉计算机输出字符串100次, 而无须重复打印输出语句100次, 如下所示:

```

int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java!");
    count++;
}
    
```

变量count的初值为0。循环检查 (count<100) 是否为true。若为true, 则执行循环体以输出消息 "Welcome to Java!", 然后给count加1。重复执行这个循环, 直到 (count<100) 变为false为止, 当 (count<100) 变为false (例如: count达到100), 此时循环终止然后执行循环语句之后的下一条语句。

循环是用来控制语句块重复执行的一种结构。循环的概念是程序设计的基础, Java 提供了三种类型的循环语句: while循环、do-while循环和for循环。

## 4.2 while循环

while循环的语法如下：

```
while (循环继续条件) {
    //循环体
    语句 (组);
}
```

while循环的流程图如图4-1a所示。循环中包含的重复执行的语句部分称为循环体 (loop body)。循环体的每一次执行都被认为是一次循环的迭代 (iteration of the loop)。每个循环都含有循环继续条件，循环继续条件是一个布尔表达式，控制循环体的执行。在循环体执行前总是先计算循环条件以决定是否执行它。若条件为true，则执行循环体；若条件为false，则终止整个循环并且程序控制转移到while循环后的下一条语句。

4.1节介绍的循环打印Welcome to Java! 100次就是while循环的一个例子。它的流程图如图4-1b所示。循环继续条件是 (count<100)，而且循环体包含如下两条语句：

```
int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java!");
    count++;
}
```

循环继续条件

循环体

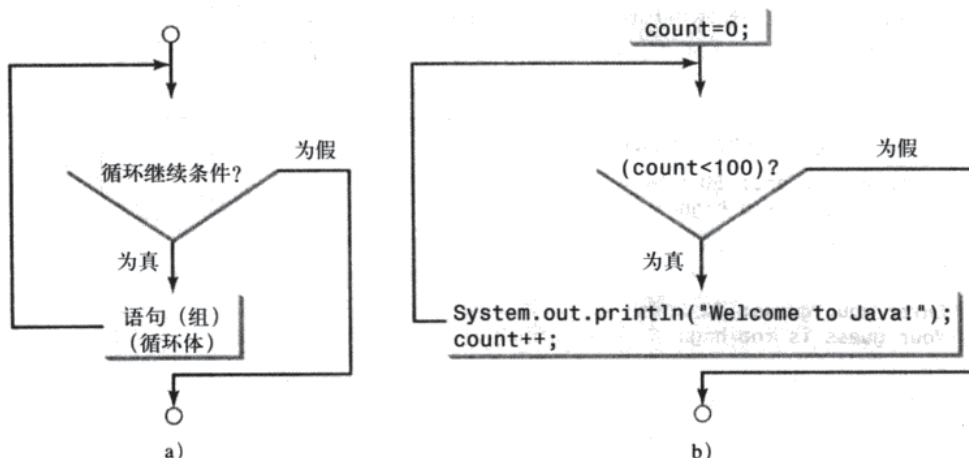


图4-1 当循环继续条件为true时，while循环重复执行循环体中的语句

在本例中，确切地知道循环体需要执行的次数。所以，使用一个控制变量count来对执行次数计数。这种类型的循环称为计数器控制的循环 (counter-controlled loop)。

**注意** 循环继续条件应该总是放在圆括号内。只有当循环体只包含一条语句或不包含语句时，循环体的花括号才可以省略。

下面是另外一个例子，有助于理解循环是如何工作的。

```
int sum = 0, i = 1;
while (i < 10) {
    sum = sum + i;
    i++;
}
System.out.println("sum is " + sum); // sum is 45
```

如果i<10为true，那么程序将i加入sum。变量i被初始化为1，然后自增为2、3、直到10。当i为

10时,  $i < 10$  为 `false`, 退出循环。所以, 和就是  $1+2+3+\dots+9=45$ 。

如果循环被错误地写为如下所示, 那会出现什么情况?

```
int sum = 0, i = 1;
while (i < 10) {
    sum = sum + i;
}
```

该循环就会成为无限循环, 因为  $i$  总是1而  $i < 10$  永远都为 `true`。

**警告** 要保证循环继续条件最终可以变为 `false`, 以便程序能够结束。一个常见的程序设计错误是无限循环。也就是说, 由于循环继续条件出错而使程序不能结束。


程序员经常会犯的错误就是使循环多执行一次或少执行一次。这种情况通常称为出一错误 (off-by-one error)。例如: 下面的循环会将 `Welcome to Java` 显示101次, 而不是100次。这个错误出在条件部分, 所以条件应该是 `count < 100` 而不是 `count <= 100`。

```
int count = 0;
while (count <= 100) {
    System.out.println("Welcome to Java!");
    count++;
}
```


#### 4.2.1 举例: 猜数字

本节的问题是猜测计算机“脑子”里想的是什么数。可以编写一个程序, 随机产生一个0到100之间且包含0和100的整数。程序提示用户连续输入一个数字, 直到它和计算机随机产生的数字相匹配为止。对用户每次输入的数字, 程序都要告诉用户该输入值是太大了, 还是太小了, 这样用户可以明智地进行下一轮的猜测。下面是一个运行示例:


Guess a magic number between 0 and 100

Enter your guess: 50 


Your guess is too high

Enter your guess: 25 


Your guess is too high

Enter your guess: 12 


Your guess is too high

Enter your guess: 6 

Your guess is too low

Enter your guess: 9 

Yes, the number is 9



这个魔法数在0到100之间。为了减小猜测的次数, 首先输入50。如果猜测值过高, 那么这个魔法数就在0到49之间。如果猜测值过低, 那么这个魔法数就在51到100之间。因此, 经过一次猜测之后, 下一次猜测时可以少考虑一半的数字。

该如何编写这个程序呢? 要立即开始编码吗? 不! 编码前的思考 (think before coding) 是非常重要的。思考一下, 在没有编写程序时你会如何解决这个问题。首先需要产生一个0到100之间且包含0和100的随机数, 然后提示用户输入一个猜测数, 最后将这个猜测数和随机数进行比较。

一次增加一个步骤地逐步编码 (code incrementally) 是一个很好的习惯。对涉及编写循环的程序而言, 如果不知道如何立即编写循环, 可以编写循环只执行一次的代码, 然后规划如何在循环中重复执行这些代码。为了编写这个程序, 可以打一个初稿, 如程序清单4-1所示。

程序清单4-1 `GuessNumberOneTime.java`

```
1 import java.util.Scanner;
2
3 public class GuessNumberOneTime {
```

```

4 public static void main(String[] args) {
5     // Generate a random number to be guessed
6     int number = (int)(Math.random() * 101);
7
8     Scanner input = new Scanner(System.in);
9     System.out.println("Guess a magic number between 0 and 100");
10
11    // Prompt the user to guess the number
12    System.out.print("\nEnter your guess: ");
13    int guess = input.nextInt();
14
15    if (guess == number)
16        System.out.println("Yes, the number is " + number);
17    else if (guess > number)
18        System.out.println("Your guess is too high");
19    else
20        System.out.println("Your guess is too low");
21 }
22 }

```

运行这个程序时，它只提示用户输入一次猜测值。为使用户重复输入猜测值，可将第11~20行的代码放入循环里，如下所示：

```

while (true) {
    // Prompt the user to guess the number
    System.out.print("\nEnter your guess: ");
    guess = input.nextInt();

    if (guess == number)
        System.out.println("Yes, the number is " + number);
    else if (guess > number)
        System.out.println("Your guess is too high");
    else
        System.out.println("Your guess is too low");
} // End of loop

```

这个循环重复提示用户输入猜测值。但是，这个循环是不正确的，因为它永远都不会结束。当 guess 和 number 匹配时，该循环就应该结束。所以，可对这个循环做如下修改：

```

while (guess != number) {
    // Prompt the user to guess the number
    System.out.print("\nEnter your guess: ");
    guess = input.nextInt();

    if (guess == number)
        System.out.println("Yes, the number is " + number);
    else if (guess > number)
        System.out.println("Your guess is too high");
    else
        System.out.println("Your guess is too low");
} // End of loop

```

程序清单4-2给出完整的代码。

程序清单4-2 GuessNumber.java

```

1 import java.util.Scanner;
2
3 public class GuessNumber {
4     public static void main(String[] args) {
5         // Generate a random number to be guessed
6         int number = (int)(Math.random() * 101);
7
8         Scanner input = new Scanner(System.in);
9         System.out.println("Guess a magic number between 0 and 100");
10
11        int guess = -1;
12        while (guess != number) {

```



```

13      // Prompt the user to guess the number
14      System.out.print("\nEnter your guess: ");
15      guess = input.nextInt();
16
17      if (guess == number)
18          System.out.println("Yes, the number is " + number);
19      else if (guess > number)
20          System.out.println("Your guess is too high");
21      else
22          System.out.println("Your guess is too low");
23      // End of loop
24  }
25 }

```

	line#	number	guess	output
	6	9		
iteration 1	11		-1	
	15		50	
	20			Your guess is too high
iteration 2	15		25	
	20			Your guess is too high
iteration 3	15		12	
	20			Your guess is too high
iteration 4	15		6	
	22			Your guess is too low
iteration 5	15		9	
	18			Yes, the number is 9

程序在第6行创建一个魔法数，然后提示用户在一个循环中连续输入猜测值（第12~23行）。对每一次猜测，程序检查该猜测数是否正确，是太高还是太低了（第17~22行）。当某次猜测正确时，程序就退出这个循环（第12行）。注意：`guess`被初始化为-1。将它初始化为0到100之间的值会出错，因为它很可能就是要猜的数。

#### 4.2.2 循环设计策略

编写一个正确的循环对编程新手来说，并不是件容易的事。编写循环时应该考虑如下三个步骤：

第一步：确定需要重复的语句。

第二步：将这些语句放在一个循环中，如下所示：

```

while (true) {
    语句组;
}

```

第三步：为循环继续条件编码，并为控制循环添加适合的语句。

```

while (循环继续条件) {
    语句组;
    用于控制循环的附件语句;
}

```

#### 4.2.3 问题：高级数学学习工具

每次运行程序清单3-4中的数学减法学习工具的程序只能产生一道题目。可以使用一个循环重复产生



题目。如何编写能产生5道题目的代码呢？遵循循环设计策略。首先，确定需要重复的语句。这些语句包括：获取两个随机数，提示用户对两数做减法然后给试题打分。然后，将这些语句放在一个循环里。最后，增加一个循环控制变量和循环继续条件，然后执行循环五次。

程序清单4-3给出的程序可以产生5道问题，在学生回答完所有5个问题后，报告回答正确的题数。这个程序还显示该测试所花的时间，并列出的所有的题目。

程序清单4-3 SubtractionQuizLoop.java

```

1 import java.util.Scanner;
2
3 public class SubtractionQuizLoop {
4     public static void main(String[] args) {
5         final int NUMBER_OF_QUESTIONS = 5; // Number of questions
6         int correctCount = 0; // Count the number of correct answers
7         int count = 0; // Count the number of questions
8         long startTime = System.currentTimeMillis();
9         String output = ""; // output string is initially empty
10        Scanner input = new Scanner(System.in);
11
12        while (count < NUMBER_OF_QUESTIONS) {
13            // 1. Generate two random single-digit integers
14            int number1 = (int)(Math.random() * 10);
15            int number2 = (int)(Math.random() * 10);
16
17            // 2. If number1 < number2, swap number1 with number2
18            if (number1 < number2) {
19                int temp = number1;
20                number1 = number2;
21                number2 = temp;
22            }
23
24            // 3. Prompt the student to answer "What is number1 - number2?"
25            System.out.print(
26                "What is " + number1 + " - " + number2 + "? ");
27            int answer = input.nextInt();
28
29            // 4. Grade the answer and display the result
30            if (number1 - number2 == answer) {
31                System.out.println("You are correct!");
32                correctCount++;
33            }
34            else
35                System.out.println("Your answer is wrong.\n" + number1
36                    + " - " + number2 + " should be " + (number1 - number2));
37
38            // Increase the count
39            count++;
40
41            output += "\n" + number1 + "-" + number2 + "=" + answer +
42                ((number1 - number2 == answer) ? " correct" : " wrong");
43        }
44
45        long endTime = System.currentTimeMillis();
46        long testTime = endTime - startTime;
47
48        System.out.println("Correct count is " + correctCount +
49            "\nTest time is " + testTime / 1000 + " seconds\n" + output);
50    }
51 }

```

```

What is 9 - 2? 7 Enter
You are correct!

What is 3 - 0? 3 Enter
You are correct!

What is 3 - 2? 1 Enter
You are correct!

What is 7 - 4? 4 Enter
Your answer is wrong.
7 - 4 should be 3

What is 7 - 5? 4 Enter
Your answer is wrong.
7 - 5 should be 2

Correct count is 3
Test time is 1021 seconds

9-2=7 correct
3-0=3 correct
3-2=1 correct
7-4=4 wrong
7-5=4 wrong

```

程序使用控制变量`count`来控制循环的执行。`count`被初始化为0（第7行），并在每次迭代中加1（第39行）。每次迭代都显示并处理一个减法题目。程序中第8行代码获得测试开始的时间，第45行获得测试结束的时间，然后在第46行计算出测试所用时间。测试时间以毫秒为单位并且在第49行被转换为秒。

#### 4.2.4 使用标志值控制循环

另一种控制循环的常用技术是在读取和处理一个集合的值时指派一个特殊值。这个特殊的输入值也称为标志值（sentinel value），用以表明循环的结束。如果一个循环使用标志值来控制它的执行，它就称为标志位控制的循环（sentinel-controlled loop）。

程序清单4-4编写程序，用来读取和计算个数不确定的整数之和。输入0则表示输入结束。需要为每次输入值声明新变量吗？答案是：不需要。只需要使用名为`data`的变量（第12行）存储输入值，并使用名为`sum`的变量（第15行）存储和。每当读取一个数，就将其赋值给`data`，如果它不为0，则将该`data`加到`sum`中（第17行）。

程序清单4-4 SentinelValue.java

```





1 import java.util.Scanner;
2
3 public class SentinelValue {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Read an initial data
10        System.out.print(
11            "Enter an int value (the program exits if the input is 0): ";
12        int data = input.nextInt();
13
14        // Keep reading data until the input is 0
15        int sum = 0;
16        while (data != 0) {
17            sum += data;
18
19            // Read the next data
20            System.out.print(
21                "Enter an int value (the program exits if the input is 0): ";

```

```

22     data = input.nextInt();
23 }
24
25     System.out.println("The sum is " + sum);
26 }
27 }

```

Enter an int value (the program exits if the input is 0): 2   
Enter an int value (the program exits if the input is 0): 3   
Enter an int value (the program exits if the input is 0): 4   
Enter an int value (the program exits if the input is 0): 0   
The sum is 9



	line#	data	sum	output
	12	2		
	15		0	
iteration 1 {	17		2	
	22	3		
iteration 2 {	17		5	
	22	4		
iteration 3 {	17		9	
	22	0		
	25			The sum is 9



如果data不为0，则将它加到总和sum中（第17行），然后读取下一条输入数据（第20~22行）。若data为0，则不再执行循环体并且终止while循环。输入值0是该循环的标志值。注意：若第一个读取到的输入值就是0，则永远不会执行循环体，最终的和sum为0。

**警告** 在循环控制中，不要使用浮点值来比较值是否相等。因为浮点值都是某些值的近似值，使用它们可能导致不精确的循环次数和不准确的结果。

考虑下面计算 $1+0.9+0.8+\dots+0.1$ 的代码：

```

double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);

```

变量item从1开始，每执行一次循环体就减去0.1。当item变为0时循环应该终止。但是，因为浮点数在算术上是近似的，所以不能确保item会变成真正的0。从表面上看，这个循环似乎没问题，但实际上它是一个无限循环。

#### 4.2.5 输入和输出重定向

在前面的例子中，如果要输入大量的数值，那么从键盘上输入是非常乏味的事。可以将这些数据用空格隔开，保存在一个名为input.txt的文本文件中，然后使用下面的命令运行这个程序：

```
java SentinelValue < input.txt
```

这个命令称为输入重定向（input redirection）。程序从文件input.txt中读取输入，而不是让用户在运行时从键盘输入数据。假设文件内容是：



```

2 3 4 5 6 7 8 9 12 23 32
23 45 67 89 92 12 34 35 3 1 2 4 0

```

程序将得到sum值为518。

类似地，还有输出重定向（output redirection），输出重定向将输出发送给文件，而不是将它们显示在控制台上。输出重定向的命令为：

```
java ClassName > output.txt
```

可以在同一命令中同时使用输入重定向和输出重定向。例如，下面的命令从文件input.txt中获取输入，并将输出发送给文件output.txt：

```
java SentinelValue < input.txt > output.txt
```

请运行这个程序，查看一下output.txt中的内容是什么。

### 4.3 do-while循环

do-while循环是while循环的变体。它的语法如下：

```

do {
    // 循环体；
    语句（组）；
} while（循环继续条件）；

```

它的执行流程图如图4-2所示。

首先执行循环体，然后计算循环继续条件。如果计算结果为true，则重复执行循环体；如果为false，则终止do-while循环。while循环与do-while循环的差别在于：计算循环继续条件和执行循环体的先后顺序不同。while循环和do-while循环具有相同的表达能力。有时候，选择其中一种会比另一种更方便。例如，可以采用do-while循环改写程序清单4-4中的while循环，如程序清单4-5所示。

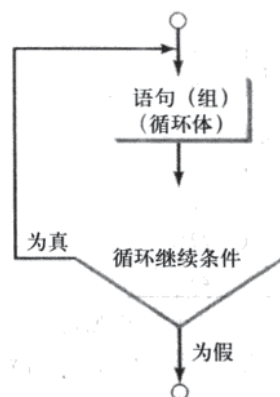


图4-2 do-while循环首先执行循环体，然后检查循环继续条件，以确定继续执行循环还是终止循环

#### 程序清单4-5 TestDoWhile.java

```

1 import java.util.Scanner;
2
3 public class TestDoWhile {
4     /** Main method */
5     public static void main(String[] args) {
6         int data;
7         int sum = 0;
8
9         // Create a Scanner
10        Scanner input = new Scanner(System.in);
11
12        // Keep reading data until the input is 0
13        do {
14            // Read the next data
15            System.out.print(
16                "Enter an int value (the program exits if the input is 0): ";
17            data = input.nextInt();
18
19            sum += data;
20        } while (data != 0);
21
22        System.out.println("The sum is " + sum);

```

```
23 }
24 }
```

Enter an int value (the program exits if the input is 0): 3 Enter  
 Enter an int value (the program exits if the input is 0): 5 Enter  
 Enter an int value (the program exits if the input is 0): 6 Enter  
 Enter an int value (the program exits if the input is 0): 0 Enter  
 The sum is 14



**提示** 同前面程序TestDoWhile中do-while循环的情形一样，如果循环中的语句至少需要执行一次，建议使用do-while循环。如果使用while循环，那么这些语句必须在循环前和循环内都出现。

## 4.4 for循环

经常会用到下面的通用形式编写循环：

```
i = initialValue; // Initialize loop control variable
while (i < endValue) {
    // Loop body
    ...
    i++; // Adjust loop control variable
}
```

可以使用for循环简化前面的循环：

```
for (i = initialValue; i < endValue; i++) {
    // Loop body
    ...
}
```

通常，for循环的语法如下所示：

```
for (初始操作; 循环继续条件; 每次迭代后的操作) {
    // 循环体;
    语句 (组);
}
```

for循环的流程图如图4-3a所示。

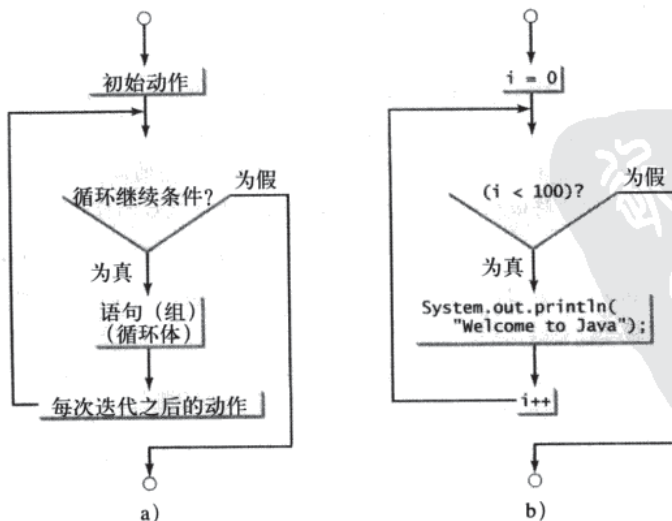


图4-3 for循环只执行初始动作一次，当循环继续条件为真时，重复执行循环体中的语句，然后完成每次迭代后的操作

**for**循环语句从关键字**for**开始,然后是用双括号括住的循环控制结构体。这个结构体包括初始动作、循环继续条件和每次迭代后的动作。控制结构体后紧跟着花括号括起来的循环体。初始动作、循环继续条件和每次迭代后的动作都要用分号分隔。

一般情况下,**for**循环使用一个变量来控制循环体的执行次数,以及什么时候循环终止。这个变量称为控制变量(control variable)。初始化动作是指初始化控制变量,每次迭代后的动作通常会对控制变量做自增或自减,而循环继续条件检验控制变量是否达到终止值。例如,下面的**for**循环打印Welcome to Java! 100次:

```
int i;
for (i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```

语句的流程图如图4-3b所示。**for**循环将控制变量*i*初始化为0,当*i*小于100时,重复执行println语句并计算*i++*。

初始化动作*i*=0初始化控制变量*i*。循环继续条件*i*<100是一个布尔表达式。这个表达式在初始化之后和每次迭代开始之前都要计算一次。如果这个条件为true,则执行该循环体。如果它为false,则循环终止,并且将程序控制转移到循环后的下一行。

每次迭代后的动作*i++*是一个调整控制变量的语句。每次迭代结束后执行这条语句。它自增控制变量的值。最终,控制变量的值应该使循环继续条件变为false,否则循环将成为无限循环。

循环控制变量可以在**for**循环中声明和初始化。下面就是一个例子:

```
for (int i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```

如果像这个例子一样,循环体内只有一条语句,则可以省略花括号。

**提示** 控制变量必须在循环控制结构体内或循环前说明。如果循环控制变量只在循环内使用而不在其他地方使用,那么在**for**循环的初始动作中声明它是一个很好的编程习惯。如果在循环控制结构体内声明变量,那么在循环外不能引用它。例如,不能在前面代码的**for**循环外引用变量*i*,因为它是在**for**循环内声明的。

**注意** **for**循环中的初始动作可以是0个或是多个以逗号隔开的变量声明语句或赋值表达式。例如:

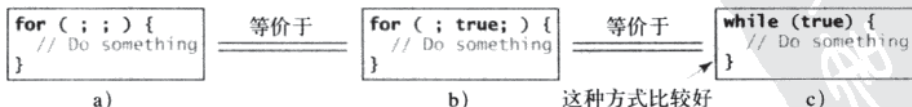
```
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```

**for**循环中每次迭代后的动作可以是0个或多个以逗号隔开的语句。例如:

```
for (int i = 1; i < 100; System.out.println(i), i++);
```

这个例子是正确的,但是它不是一个好例子,因为它增加了程序的阅读难度。通常,将声明和初始化一个变量作为初始动作,将增加或减少控制变量作为每次迭代后的操作。

**注意** 如果省略**for**循环中的循环继续条件,则隐含地认为循环继续条件为true。因此,下面图a中给出的语句和图b中给出的语句一样,它们都是无限循环。但是,为了避免混淆,最好还是使用图c中的等价循环:



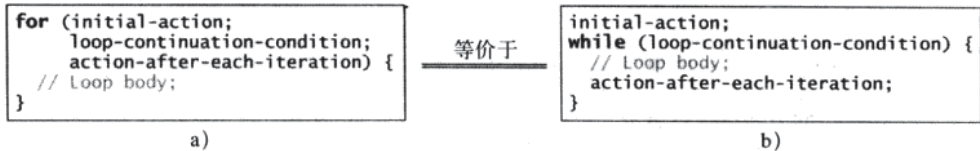
## 4.5 采用哪种循环

**while**循环和**for**循环都称为预测式循环(pretest loop),因为继续条件是在循环体执行之前检测的,

do-while循环称为后测试循环 (posttest loop)，因为循环条件是在循环体执行之后检测的。三种形式的循环语句：while、do-while和for，在表达上是等价的。也就是说，可以使用这三种形式之一来编写一个循环。例如，下面图a中while循环总能转化为图b中的for循环：

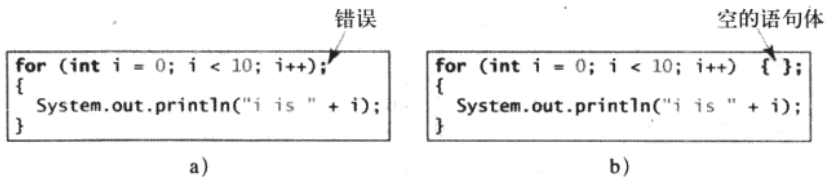


除了某些特殊情况外 (参见复习题4.17中的情况)，下面图a中的for循环通常都能转化为图b中的while循环：

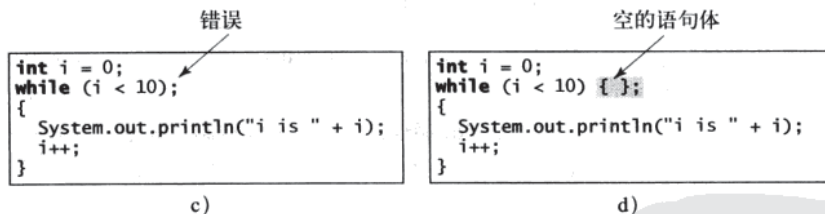


建议使用自己觉得最自然、最舒服的一种循环语句。通常，如果已经提前知道重复次数，那就采用for循环，例如，需要打印一条信息100次时，如果无法确定重复次数，就采用while循环，就像读入一些数值直到读入0为止的这种情况。如果在检验继续条件前需要执行循环体，就用do-while循环替代while循环。

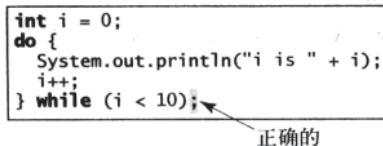
**警告** 在for子句的末尾和循环体之间多写分号是一个常见的错误，如下面的图a中所示。图a中分号过早地表明循环的结束。循环体实际上都是为空的，如图b所示。图a和图b是等价的。



类似地，图c中的循环也是错的，图c与图d等价。



通常在使用次行块格式时容易发生这些错误。使用行尾块风格可以避免这种类型的错误。在do-while循环中，需要分号来结束这个循环。



## 4.6 嵌套循环

嵌套循环是由一个外层循环和一个或多个内层循环组成的。每当重复执行一次外层循环时再次进入内部循环，然后重新开始。

程序清单4-6是使用嵌套for循环打印一个乘法表的程序。

程序清单4-6 MultiplicationTable.java

```

1 public class MultiplicationTable {
2     /** Main method */
3     public static void main(String[] args) {
4         // Display the table heading
5         System.out.println("      Multiplication Table");
6
7         // Display the number title
8         System.out.print("      ");
9         for (int j = 1; j <= 9; j++)
10            System.out.print("    " + j);
11
12        System.out.println("\n-----");
13
14        // Print table body
15        for (int i = 1; i <= 9; i++) {
16            System.out.print(i + " | ");
17            for (int j = 1; j <= 9; j++) {
18                // Display the product and align properly
19                System.out.printf("%4d", i * j);
20            }
21            System.out.println();
22        }
23    }
24 }

```

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



程序在输出的第一行显示标题（第5行）。第一个for循环（第9~10行）在第二行显示从1到9的数字。在第三行显示横线（-）（第12行）。

下一个循环（第15~22行）是一个嵌套的for循环，其外层循环控制变量是*i*，而内层循环控制变量是*j*。在内层循环中，针对每个*i*，随着*j*取遍1, 2, 3, ..., 9，内层循环在每一行显示乘积*i*\**j*的值。

## 4.7 最小化数值误差

涉及浮点数的数值误差是不可避免的。本节将通过实例讨论如何最小化这种误差。

程序清单4-7给出的例子，计算从0.01到1.0的数列之和，该数列中的数值以0.01递增，如下所示：

0.01+0.02+0.03 + ...。

程序清单4-7 TestSum.java

```

1 public class TestSum {
2     public static void main(String[] args) {
3         // Initialize sum
4         float sum = 0;
5
6         // Add 0.01, 0.02, ..., 0.99, 1 to sum
7         for (float i = 0.01f; i <= 1.0f; i = i + 0.01f)
8             sum += i;
9     }
10 }

```



```

9
10 // Display result
11 System.out.println("The sum is " + sum);
12 }
13 }

```

The sum is 50.499985



for循环（第7~8行）重复地将控制变量*i*加到sum中。变量*i*从0.01开始，每次迭代增加0.01。当*i*超过1.0时循环终止。

for循环初始动作可以是任何语句，但是，它经常用来初始化控制变量。从本例中可以看到，控制变量可以是float型。事实上，它可以为任意数据类型。

sum的精确结果应该是50.50，但是答案是50.499985。这个结果是不精确的，因为计算机使用固定位数表示浮点数，因此，它就不能精确表示某些浮点数。如果如下所示，将程序中的float型改成double型，应该可以看到精度有一些小小的改善，因为double型变量占64位而float型变量只占32位。

```

// Initialize sum
double sum = 0;

// Add 0.01, 0.02, ..., 0.99, 1 to sum
for (double i = 0.01; i <= 1.0; i = i + 0.01)
    sum += i;

```

可是你会吃惊地看到，实际的结果是49.50000000000003。究竟哪里出错了呢？如果将循环中每次迭代的*i*打印出来，会发现最后一个*i*比1稍微大一点（不是精确的1）。这就会造成最后一个*i*不能加到sum中。根本问题就是浮点数是用近似值表示的。为了解决这个问题，使用整数计数器以确保所有数字都被加到sum中。下面是一个新的循环：

```

double currentValue = 0.01;
for (int count = 0; count < 100; count++) {
    sum += currentValue;
    currentValue += 0.01;
}

```

这个循环结束后，sum的值是50.50000000000003。这个循环从小到大添加数字。如果如下所示从大到小（即以1.0, 0.99, 0.98, ..., 0.02, 0.01的顺序）添加，那会发生什么呢？

```

double currentValue = 1.0;
for (int count = 0; count < 100; count++) {
    sum += currentValue;
    currentValue -= 0.01;
}

```

在这个循环之后，sum的值是50.49999999999995。从大到小添加数字没有从小到大添加数字得到的值精确。这种现象是有限精度算术的产物。如果结果值需要比变量所能存储的容量值更高的精度，那么添加一个非常小的数给一个非常大的数可能没有什么影响。例如，100000000.0+0.000000001的精确结果是100000000.0。为了得到更精确的结果，仔细选择计算的顺序。在大数之前先增加小数是减小误差的一种方法。

## 4.8 实例学习

循环语句是程序设计的基础，编写循环语句的能力在学习Java程序设计时是非常必要的。如果能够使用循环编写程序，你就懂得如何编程了。正是由于这个原因，本节提供三个附加的例子，学习如何使用循环来解决问题。

### 4.8.1 举例：求最大公约数

两个整数4和2的最大公约数是2。两个整数16和24的最大公约数是8。如何求最大公约数呢？设输入的两个整数为n1和n2。已知1是一个公约数，但是它可能不是最大公约数。所以，可以检测k (k=2, 3, 4...) 是否为n1和n2的最大公约数，直到k大于n1或n2。公约数存储在名为gcd的变量中，gcd的初值设为1。当找到一个新的公约数时，它就成为新的gcd。当检查完在2到n1或n2之间所有可能的公约数后，变量gcd的值就是最大公约数。这个思路可以翻译成下面的循环：

```
int gcd = 1; // Initial gcd is 1
int k = 2; // Possible gcd
while (k <= n1 && k <= n2) {
    if (n1 % k == 0 && n2 % k == 0)
        gcd = k; // Update gcd
    k++; // Next possible gcd
}
```

// After the loop, gcd is the greatest common divisor for n1 and n2

程序清单4-8给出的程序，提示用户输入两个正整数，然后找到它们的最大公约数。

程序清单4-8 GreatestCommonDivisor.java

```
1 import java.util.Scanner;
2
3 public class GreatestCommonDivisor {
4     /** Main method */
5     public static void main(String[] args) {
6         // Create a Scanner
7         Scanner input = new Scanner(System.in);
8
9         // Prompt the user to enter two integers
10        System.out.print("Enter first integer: ");
11        int n1 = input.nextInt();
12        System.out.print("Enter second integer: ");
13        int n2 = input.nextInt();
14
15        int gcd = 1; // Initial gcd is 1
16        int k = 2; // Possible gcd
17        while (k <= n1 && k <= n2) {
18            if (n1 % k == 0 && n2 % k == 0)
19                gcd = k; // Update gcd
20            k++;
21        }
22
23        System.out.println("The greatest common divisor for " + n1 +
24            " and " + n2 + " is " + gcd);
25    }
26 }
```

Enter first integer: 125   
 Enter second integer: 2525   
 The greatest common divisor for 125 and 2525 is 25



如何编写这个程序呢？应该立即开始写代码吗？不，输代码之前先思考一下是很重要的。思考能够使你在不涉及怎样编写代码的情况下，得到问题的逻辑方案。一旦有了逻辑方案，再输入代码，将解决方案翻译成Java语言。翻译方式不是唯一的。例如，可以使用for循环改写代码，如下所示：

```
for (int k = 2; k <= n1 && k <= n2; k++) {
    if (n1 % k == 0 && n2 % k == 0)
        gcd = k;
}
```

一个问题常常有多种解决方案。最大公约数 (GCD) 问题就有许多解决方法。练习题4.15就给出了另一种解决方案。一个更有效的方法是使用经典的欧几里得算法。参考网站<http://www.cut-the-knot.org/blue/Euclid.shtml>可以得到相关的更多信息。

考虑到 $n_1$ 的除数不可能大于 $n_1/2$ ，所以，你可能会尝试用下面的循环改进该程序：

```
for (int k = 2; k <= n1 / 2 && k <= n2 / 2; k++) {
    if (n1 % k == 0 && n2 % k == 0)
        gcd = k;
}
```

上面的修改是错误的，你能找出原因吗？参见复习题4.14找出答案。

#### 4.8.2 问题：预测未来学费

假设某个大学今年的学费是10000美金，而且以每年7%的速度增加。多少年之后学费会翻倍？

在编写解决这个问题的程序之前，首先考虑如何手工解决它。第二年的学费是第一年的学费乘以

1.07。未来一年的学费都是前一年的学费乘以1.07。所以，每年的学费可以如下计算：

```
double tuition = 10000;   int year = 1   // Year 1
tuition = tuition * 1.07; year++;         // Year 2
tuition = tuition * 1.07; year++;         // Year 3
tuition = tuition * 1.07; year++;         // Year 4
...
```

不断地计算新一年的学费，直到学费至少是20000美金为止。到那时，就知道学费翻倍需要几年的时间。现在，可以将这个逻辑翻译成下面的循环：

```
double tuition = 10000;   // Year 1
int year = 1;
while (tuition < 20000) {
    tuition = tuition * 1.07;
    year++;
}
```

完整的程序如程序清单4-9所示。

程序清单4-9 FutureTuition.java

```
1 public class FutureTuition {
2     public static void main(String[] args) {
3         double tuition = 10000;   // Year 1
4         int year = 1;
5         while (tuition < 20000) {
6             tuition = tuition * 1.07;
7             year++;
8         }
9
10        System.out.println("Tuition will be doubled in "
11            + year + " years");
12    }
13 }
```

Tuition will be doubled in 12 years

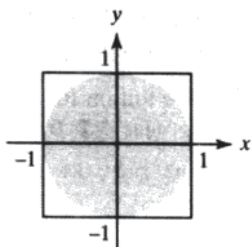


使用while循环（第5~8行）重复计算新一年的学费。当学费大于或等于20000美金时，循环结束。

#### 4.8.3 问题：蒙特卡罗模拟

蒙特卡罗模拟使用随机数和概率来解决问题。这个方法在计算数学、物理、化学和财经方面有很广泛的应用。本节给出使用蒙特卡罗模拟来估算 $\pi$ 值的例子。

为了使用蒙特卡罗方法来估算 $\pi$ ，画出一个圆的外接正方形，如下所示：



假设这个圆的半径是1。那么圆面积就是 $\pi$ 而外接正方形的面积是4。随便产生正方形中的一个点。该点落在这个圆内的概率是 $\text{circleArea/squareArea}$  (圆面积/正方形面积)  $=\pi/4$ 。

编写程序，在正方形内随机产生1000000个点，用`numberOfHits`表示落在圆内的点。因此，`numberOfHits`大约是 $1000000 * (\pi/4)$ 。可以近似估算 $\pi$ 为 $4 * \text{numberOfHits} / 1000000$ 。完整的程序如程序清单4-10所示。

程序清单4-10 MonteCarloSimulation.java

```
1 public class MonteCarloSimulation {
2     public static void main(String[] args) {
3         final int NUMBER_OF_TRIALS = 10000000;
4         int numberOfHits = 0;
5
6         for (int i = 0; i < NUMBER_OF_TRIALS; i++) {
7             double x = Math.random() * 2.0 - 1;
8             double y = Math.random() * 2.0 - 1;
9             if (x * x + y * y <= 1)
10                numberOfHits++;
11        }
12
13        double pi = 4.0 * numberOfHits / NUMBER_OF_TRIALS;
14        System.out.println("PI is " + pi);
15    }
16 }
```

PI is 3.14124



程序的第7~8行重复地产生落在正方形内的随机点  $(x, y)$ ：

```
double x = Math.random() * 2.0 - 1;
double y = Math.random() * 2.0 - 1;
```

如果 $x^2 + y^2 \leq 1$ ，那么这个点就在圆内，并给`numberOfHits`自增1。 $\pi$ 就近似为 $4 * \text{numberOfHits} / \text{NUMBER\_OF\_TRIALS}$  (第13行)。

## 4.9 关键字break和continue

**教学注意** 关键字break和continue都可以在循环语句中使用，为循环提供额外的控制。在某些情况下，使用break和continue可以简化程序设计。但是，过度使用或者不正确地使用它们会使得程序难以读懂也难以调试。(提醒教师：可以跳过本节，对本书的其他内容没有任何影响)。

你已经在switch语句中使用过关键字break，你也可以在一个循环中使用break立即终止该循环。

程序清单4-11给出的程序演示在循环中使用break的效果。

程序清单4-11 TestBreak.java

```
1 public class TestBreak {
2     public static void main(String[] args) {
3         int sum = 0;
4         int number = 0;
```



```

5
6   while (number < 20) {
7       number++;
8       sum += number;
9       if (sum >= 100)
10          break;
11   }
12
13   System.out.println("The number is " + number);
14   System.out.println("The sum is " + sum);
15 }
16 }

```

```

The number is 14
The sum is 105

```



程序清单4-11中的程序将从1到20的整数依次加到sum中，直到sum大于或等于100。如果没有if语句（第9行），该程序计算从1到20之间整数的和。但是，有了if语句，那么当总和大于或等于100时，这个循环就会终止。没有if语句，程序输出结果将会是：

```

The number is 20
The sum is 210

```



也可以在循环中使用关键字continue。当程序遇到continue时，它会结束当前的迭代。程序控制转向该循环体的末尾。换句话说，continue只是跳出了一次迭代，而关键字break是跳出了整个循环。程序清单4-12给出的程序演示在循环中使用continue的效果。

#### 程序清单4-12 TestContinue.java

```

1 public class TestContinue {
2     public static void main(String[] args) {
3         int sum = 0;
4         int number = 0;
5
6         while (number < 20) {
7             number++;
8             if (number == 10 || number == 11)
9                 continue;
10            sum += number;
11        }
12
13        System.out.println("The sum is " + sum);
14    }
15 }

```

```

The sum is 189

```



程序清单4-12中的程序，将1到20中除去10和11外的整数都加到sum中。程序中有了if语句（第8行），当number为10或11时就会执行continue语句。continue语句结束了当前迭代，就不再执行循环体中的其他语句，因此，当number为10或11时，它就没有被加到sum中。若程序中没有if语句，程序的输出就会如下所示：

```

The sum is 210

```



在这种情况下，即使当number为10或11时，也要将所有的数都加到sum中。因此，结果为210，这个值比有if语句的情况获取的值大了21。



注意 `continue`语句总是在一个循环内。在`while`和`do-while`循环中，`continue`语句之后会马上计算循环继续条件；而在`for`循环中，`continue`语句之后会立即先执行每次迭代后的动作，再计算循环继续条件。

总是可以编写在循环中不使用`break`和`continue`的程序，参见复习题4.18。通常，只有在能够简化代码并使程序更容易阅读的情况下，才可以适当地使用`break`和`continue`。

程序清单4-2给出猜数字的程序，可以使用`break`语句改写它，如程序清单4-13所示。

程序清单4-13 `GuessNumberUsingBreak.java`

```

1 import java.util.Scanner;
2
3 public class GuessNumberUsingBreak {
4     public static void main(String[] args) {
5         // Generate a random number to be guessed
6         int number = (int)(Math.random() * 101);
7
8         Scanner input = new Scanner(System.in);
9         System.out.println("Guess a magic number between 0 and 100");
10
11         while (true) {
12             // Prompt the user to guess the number
13             System.out.print("\nEnter your guess: ");
14             int guess = input.nextInt();
15
16             if (guess == number) {
17                 System.out.println("Yes, the number is " + number);
18                 break;
19             }
20             else if (guess > number)
21                 System.out.println("Your guess is too high");
22             else
23                 System.out.println("Your guess is too low");
24         } // End of loop
25     }
26 }

```

使用`break`语句可以使程序更简单和更易读。但是，应该谨慎使用`break`和`continue`。过多使用`break`和`continue`会使循环有很多退出点，使程序很难阅读。

注意 很多程序设计语言都有`goto`语句。`goto`语句可以随意地将控制转移到程序中的任意一条语句上，然后执行它。这使程序很容易出错。Java中的`break`语句和`continue`语句是不同于`goto`语句的。它们只能运行在循环中或者`switch`语句中。`break`语句跳出整个循环，而`continue`语句跳出循环的当前迭代。

## 问题：显示素数

大于1的整数，如果它的正因子只有1和它自身，那么该整数就是素数。例如：2、3、5、7都是素数，而4、6、8、9不是。

现在的问题是在5行中显示前50个素数，每行包含10个数。该问题可分解成以下任务：

- 判断一个给定数是否是素数。
- 针对`number=2, 3, 4, 5, 6, …`，测试它是否为素数。
- 统计素数的个数。
- 打印每个素数，每行打印10个。

显然，需要编写循环，反复检测新的`number`是否是素数。如果`number`是素数，则给计数器加1。计数器`count`被初始化为0。当它等于50时，循环终止。

下面是该问题的算法：

```

设置打印出来的素数个数为常量NUMBER_OF_PRIMES;
使用count来对素数个数进行计数并将其初值设为0;
设置number初始值为2;
while (count<NUMBER_OF_PRIMES) {
    测试该数是否是素数;
    if该数是素数{
        打印该素数并给count增加1;
    }
    给number加1;
}

```

为了测试某个数是否是素数，就要检测它是否能被2、3、4，一直到 $\text{number}/2$ 的整数整除。如果能被整除，那它就不是素数。这个算法可以描述如下：

```

使用布尔变量isPrime表示number是否是素数；设置isPrime的初值为true；
for (int divisor = 2; divisor<=number/2; divisor++) {
    if (number%divisor==0) {
        将isPrime设置为false
        退出循环；
    }
}

```

完整的程序在程序清单4-14中给出。

程序清单4-14 PrimeNumber.java

```

1 public class PrimeNumber {
2     public static void main(String[] args) {
3         final int NUMBER_OF_PRIMES = 50; // Number of primes to display
4         final int NUMBER_OF_PRIMES_PER_LINE = 10; // Display 10 per line
5         int count = 0; // Count the number of prime numbers
6         int number = 2; // A number to be tested for primeness
7
8         System.out.println("The first 50 prime numbers are \n");
9
10        // Repeatedly find prime numbers
11        while (count < NUMBER_OF_PRIMES) {
12            // Assume the number is prime
13            boolean isPrime = true; // Is the current number prime?
14
15            // Test whether number is prime
16            for (int divisor = 2; divisor <= number / 2; divisor++) {
17                if (number % divisor == 0) { // If true, number is not prime
18                    isPrime = false; // Set isPrime to false
19                    break; // Exit the for loop
20                }
21            }
22
23            // Print the prime number and increase the count
24            if (isPrime) {
25                count++; // Increase the count
26
27                if (count % NUMBER_OF_PRIMES_PER_LINE == 0) {
28                    // Print the number and advance to the new line
29                    System.out.println(number);
30                }
31                else
32                    System.out.print(number + " ");
33            }
34
35            // Check if the next number is prime
36            number++;

```

```

37     }
38 }
39 }

```

The first 50 prime numbers are

```

2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229

```



对编程新手而言这是一个复杂的例子。开发程序方案来解决这个问题或其他很多问题的关键之处在于要把问题分解成子问题，然后逐个地开发出每个子问题的解决方案。不要一开始就试图开发出一个完整的解决方案。而是应该首先编写代码判断一个给定的数是否是素数，然后扩展这个程序，再在循环中判断其他数是否是素数。

为了判断一个数是否是素数，就该检验该数是否能被2到 $\text{number}/2$ 之间并包括2和 $\text{number}/2$ 的整数整除（第16行）。如果能被整除，那它就不是素数（第18行）；否则，它就是一个素数。若是素数，就显示该数。若 $\text{count}$ 能被10整除（第27~30行），就转入一个新行。当计数器 $\text{count}$ 达到50时，程序终止。

程序在第19行使用`break`语句，一旦发现 $\text{number}$ 不是素数，就立即退出`for`循环。也可以不用`break`语句，改写这个循环（第16~21行），如下所示：

```

for (int divisor = 2; divisor <= number / 2 && isPrime;
    divisor++) {
    // If true, the number is not prime
    if (number % divisor == 0) {
        // Set isPrime to false, if the number is not prime
        isPrime = false;
    }
}

```

然而，在本例中，使用`break`语句可以使程序更简单、更易读。

## 4.10 (GUI) 使用确认对话框控制循环

使用确认对话框可以实现一个标志值控制的循环。答案Yes或者No决定是继续还是终止这个循环。这个循环的模板可能看起来如下所示：

```

int option = JOptionPane.YES_OPTION;
while (option == JOptionPane.YES_OPTION) {
    System.out.println("continue loop");
    option = JOptionPane.showConfirmDialog(null, "Continue?");
}

```

程序清单4-15使用一个确认对话框改写程序清单4-14。运行示例如图4-4所示。

程序清单4-15 SentinelValueUsingConfirmationDialog.java

```

1 import javax.swing.JOptionPane;
2
3 public class SentinelValueUsingConfirmationDialog {
4     public static void main(String[] args) {
5         int sum = 0;
6
7         // Keep reading data until the user answers No
8         int option = JOptionPane.YES_OPTION;
9         while (option == JOptionPane.YES_OPTION) {
10             // Read the next data
11             String dataString = JOptionPane.showInputDialog(

```

```

12     "Enter an int value: ");
13     int data = Integer.parseInt(dataString);
14
15     sum += data;
16
17     option = JOptionPane.showConfirmDialog(null, "Continue?");
18 }
19
20 JOptionPane.showMessageDialog(null, "The sum is " + sum);
21 }
22 }

```

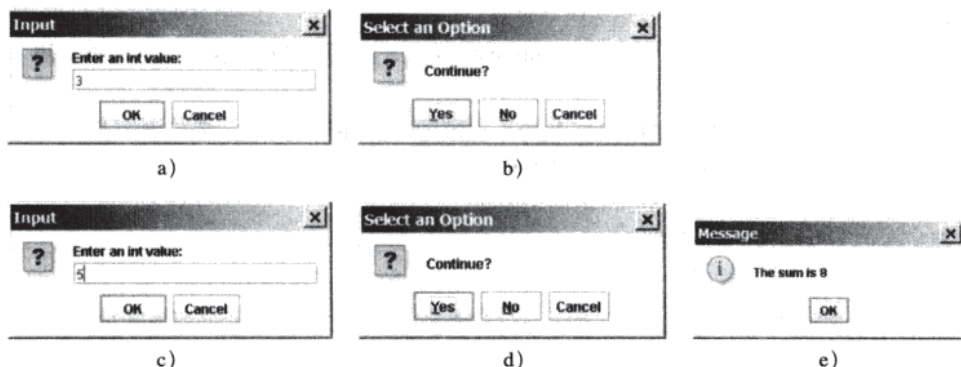


图4-4 用户在a中输入3，在b中点击Yes，在c中输入5，在d中点击Yes，结果显示在e中

程序显示一个输入对话框，提示用户输入一个整数（第11行），并将它加到sum中（第15行）。第17行显示一个确认对话框，让用户来决定是否继续输入。如果用户点击Yes，那么循环继续；否则循环终止。最后程序会在消息对话框中显示结果（第20行）。

## 关键术语

break statement (break语句)	loop (循环)
continue statement (continue语句)	loop-condition condition (循环继续条件)
do-while loop (do-while循环)	loop body (循环体)
for loop (for循环)	nested loop (嵌套循环)
loop control structure (循环控制结构)	off-by-one error (出一错误)
infinite loop (无限循环、死循环)	output redirection (输出重定向)
input redirection (输入重定向)	sentinel value (标志值)
iteration (迭代)	while loop (while循环)
labeled continue statement (带标号的continue语句)	

## 本章小结

- 循环语句有三类：while循环、do-while循环和for循环。
- 循环中需要重复执行的语句所构成的整体称为循环体。
- 循环体执行一次称为循环的一次迭代。
- 无限循环是指循环语句被无限次执行。
- 在设计循环时，既需要考虑循环控制结构体，还需要考虑循环体。
- while循环首先检查循环继续条件。如果条件为true，则执行循环体；如果条件为false，则循环结束。



- **do-while**循环与**while**循环类似，只是**do-while**循环先执行循环体，然后再检查循环继续条件，以确定是继续还是终止。
- 由于**while**循环和**do-while**循环都包含循环继续条件，这个条件是和循环体相关的，重复的次数就是由循环体决定的。因此，**while**和**do-while**循环常用于循环次数不确定的情况。
- 标志值是一个特殊的值，用来标志循环的结束。
- **for**循环一般用在循环体执行次数预知的情况，这个执行次数不是由循环体确定的。
- **for**循环控制由三部分组成。第一部分是初始操作，通常用于初始化控制变量。第二部分是循环继续条件，决定是否执行循环体。第三部分是每次迭代后执行的操作，经常用于调整控制变量。通常，在控制结构中初始化和修改循环控制变量。
- **while**循环和**for**循环都称为前测循环 (pretest loop)，因为在循环体执行之前，要检测一下循环继续条件。
- **do-while**循环称为后测循环 (posttest loop)，因为在循环体执行之后，要检测一下这个条件。
- 在循环中可以使用关键字**break**和**continue**。
- 关键字**break**立即终止包含**break**的最内层循环。
- 关键字**continue**只是终止当前迭代。

## 复习题

### 4.2~4.4节

4.1 分析下面的代码。在Point A处、Point B处和Point C处，**count < 0**总是**true**还是总是**false**，还是有时是**true**有时是**false**？

```
int count = 0;
while (count < 100) {
    // Point A
    System.out.println("Welcome to Java!\n");
    count++;
    // Point B
}
// Point C
```

4.2 在程序清单4-2中的第11行中，如果**guess**初始化为0，会出现什么错误？

4.3 下面的循环体要重复多少次？这个循环的输出是什么？

```
int i = 1;
while (i < 10)
    if (i % 2 == 0)
        System.out.println(i);
```

a)

```
int i = 1;
while (i < 10)
    if (i % 2 == 0)
        System.out.println(i++);
```

b)

```
int i = 1;
while (i < 10)
    if ((i++) % 2 == 0)
        System.out.println(i);
```

c)

4.4 **while**循环和**do-while**循环之间的区别是什么？将下面的**while**循环转换成**do-while**循环。

```
int sum = 0;
int number = input.nextInt();
while (number != 0) {
    sum += number;
    number = input.nextInt();
}
```

4.5 做完下列两个循环之后，**sum**是否具有相同的值？

```
for (int i = 0; i < 10; ++i) {
    sum += i;
}
```

a)

```
for (int i = 0; i < 10; i++) {
    sum += i;
}
```

b)



4.6 for循环控制的三个部分是什么？编写一个for循环，输出从1到100的整数。

4.7 假设输入是2 3 4 5 0，那么下面代码的输出结果是什么？

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int number, max;
        number = input.nextInt();
        max = number;

        while (number != 0) {
            number = input.nextInt();
            if (number > max)
                max = number;
        }

        System.out.println("max is " + max);
        System.out.println("number " + number);
    }
}
```

4.8 假设输入是2 3 4 5 0，那么下面代码的输出结果是什么？

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int number, sum = 0, count;

        for (count = 0; count < 5; count++) {
            number = input.nextInt();
            sum += number;
        }

        System.out.println("sum is " + sum);
        System.out.println("count is " + count);
    }
}
```

4.9 假设输入是2 3 4 5 0，那么下面代码的输出结果是什么？

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int number, max;
        number = input.nextInt();
        max = number;

        do {
            number = input.nextInt();
            if (number > max)
                max = number;
        } while (number != 0);

        System.out.println("max is " + max);
        System.out.println("number " + number);
    }
}
```



## 114 • 第4章 循 环

4.10 下面的语句做什么？

```
for ( ; ; ) {
    do something;
}
```

4.11 如果在for循环控制中声明一个变量，在退出循环后还可以使用它吗？

4.12 可以将for循环转换为while循环吗？列出使用for循环的好处？

4.13 将下面的for循环语句转换为while循环和do-while循环：

```
long sum = 0;
for (int i = 0; i <= 1000; i++)
    sum = sum + i;
```

4.14 如果将程序清单4-8中第17行的n1和n2用n1/2和n2/2来替换，程序还会工作吗？

4.9节

4.15 关键字break的作用是什么？关键字continue的作用是什么？下列程序能够结束吗？如果能，给出结果。

```
int balance = 1000;
while (true) {
    if (balance < 9)
        break;
    balance = balance - 9;
}

System.out.println("Balance is "
    + balance);
```

a)

```
int balance = 1000;
while (true) {
    if (balance < 9)
        continue;
    balance = balance - 9;
}

System.out.println("Balance is "
    + balance);
```

b)

4.16 是否总能将while循环转换为for循环？将下列while循环转换为for循环。

```
int i = 1;
int sum = 0;
while (sum < 10000) {
    sum = sum + i;
    i++;
}
```

4.17 将下面左边的for循环转换成右边的while循环，有什么错误？改正该错误。

```
for (int i = 0; i < 4; i++) {
    if (i % 3 == 0) continue;
    sum += i;
}
```

转换为

错误转换

```
int i = 0;
while (i < 4) {
    if (i % 3 == 0) continue;
    sum += i;
    i++;
}
```

4.18 不使用关键字break和continue，改写程序清单4-11和程序清单4-12的程序TestBreak和TestContinue。

4.19 在下面的循环中，执行完break语句之后，执行哪条语句？显示输出。

```
for (int i = 1; i < 4; i++) {
    for (int j = 1; j < 4; j++) {
        if (i * j > 2)
            break;

        System.out.println(i * j);
    }
    System.out.println(i);
}
```

4.20 在下面的循环中，执行continue语句之后，执行哪条语句？显示输出。

```
for (int i = 1; i < 4; i++) {
    for (int j = 1; j < 4; j++) {
        if (i * j > 2)
```

```

        continue;
    System.out.println(i * j);
}
System.out.println(i);
}

```

### 综合题

4.21 查找并修改下面代码中的错误：

```

1 public class Test {
2     public void main(String[] args) {
3         for (int i = 0; i < 10; i++);
4         sum += i;
5
6         if (i < j);
7         System.out.println(i)
8     else
9         System.out.println(j);
10
11     while (j < 10);
12     {
13         j++;
14     };
15
16     do {
17         j++;
18     } while (j < 10)
19 }
20 }

```

4.22 下面程序的错误在哪里？

```

1 public class ShowErrors {
2     public static void main(String[] args) {
3         int i;
4         int j = 5;
5
6         if (j > 3)
7             System.out.println(i + 4);
8     }
9 }

```

a)

```

1 public class ShowErrors {
2     public static void main(String[] args) {
3         for (int i = 0; i < 10; i++);
4             System.out.println(i + 4);
5     }
6 }

```

b)

4.23 给出下面程序的输出结果（提示：画一个表格，将变量放在某一列中，跟踪这些程序）。

```

public class Test {
    /** Main method */
    public static void main(String[] args) {
        for (int i = 1; i < 5; i++) {
            int j = 0;
            while (j < i) {
                System.out.print(j + " ");
                j++;
            }
        }
    }
}

```

a)

```

public class Test {
    /** Main method */
    public static void main(String[] args) {
        int i = 0;
        while (i < 5) {
            for (int j = i; j > 1; j--)
                System.out.print(j + " ");
            System.out.println("*****");
            i++;
        }
    }
}

```

b)

```
public class Test {
    public static void main(String[] args) {
        int i = 5;
        while (i >= 1) {
            int num = 1;
            for (int j = 1; j <= i; j++) {
                System.out.print(num + "xxx");
                num *= 2;
            }
            System.out.println();
            i--;
        }
    }
}
```

c)

```
public class Test {
    public static void main(String[] args) {
        int i = 1;
        do {
            int num = 1;
            for (int j = 1; j <= i; j++) {
                System.out.print(num + "G");
                num += 2;
            }
            System.out.println();
            i++;
        } while (i <= 5);
    }
}
```

d)

4.24 下面程序的输出是什么？解释原因。

```
int x = 80000000;
while (x > 0)
    x++;
System.out.println("x is " + x);
```

4.25 统计下面循环的迭代次数。

```
int count = 0;
while (count < n) {
    count++;
}
```

a)

```
for (int count = 0;
     count <= n; count++) {
}
```

b)

```
int count = 5;
while (count < n) {
    count++;
}
```

c)

```
int count = 5;
while (count < n) {
    count = count + 3;
}
```

d)

## 编程练习题

**教学注意** 对每个问题，都应该多读几次，直到理解透彻为止。在编码之前，思考一下如何解决这个问题。然后将你的逻辑翻译成程序。

一个问题经常是有多种解决方案的。鼓励学生探索多种解决方案。

### 4.2~4.7节

\*4.1 (统计正数和负数的个数然后计算这些数的平均值) 编写程序，读入未指定个数的整数，判断读入的正数有多少个，读入的负数有多少个，然后计算这些输入值的总和及其平均值（不对0计数）。当输入为0时，表明程序结束。将平均值以浮点数显示。下面是一个运行示例：

```
Enter an int value, the program exits if the input is 0:
1 2 -1 3 0
The number of positives is 3
The number of negatives is 1
The total is 5
The average is 1.25
```



PDG

4.2 (重复加法) 程序清单4-3产生了5个随机减法问题。改写该程序，使它产生10个随机加法问题，加

数是两个1到15之间的整数。显示正确答案的个数和测验时间。

- 4.3 (将千克转换成磅) 编写程序, 显示下面的表格 (注意: 1千克为2.2磅):

千克	磅
1	2.2
3	6.6
...	
197	433.4
199	437.8

- 4.4 (将英里转换成千米) 编写程序, 显示下面的表格 (注意: 1英里为1.609千米):

英里	千米
1	1.609
2	3.218
...	
9	14.481
10	16.090

- 4.5 (千克与磅之间的互换) 编写一个程序, 并排显示下列两个表格 (注意: 1千克为2.2磅):

千克	磅	磅	千克
1	2.2	20	9.09
3	6.6	25	11.36
...			
197	433.4	510	231.82
199	437.8	515	234.09

- 4.6 (英里与千米之间的互换) 编写一个程序, 并排显示下列两个表格 (注意: 1英里为1.609千米):

英里	千米	千米	英里
1	1.609	20	12.430
2	3.218	25	15.538
...			
9	14.481	60	37.290
10	16.090	65	40.398

- \*\*4.7 (财务应用程序: 计算将来的学费) 假设今年某大学的学费为10000美元, 学费的年增长率为5%。编写程序, 计算10年后的学费以及从现在开始10年后算起, 4年内总学费是多少?

- 4.8 (找出最高分) 编写程序, 提示用户输入学生的个数、每个学生的名字及其分数, 最后显示得最高分的学生的名字。

- \*4.9 (找出两个最高分) 编写程序, 提示用户输入学生的个数、每个学生的名字及其分数, 最后显示获得最高分的学生和第二高分的学生。

- 4.10 (找出能被5和6整除的数) 编写程序, 显示从100到1000之间所有能被5和6整除的数, 每行显示10个。

- 4.11 (找出能被5或6整除, 但不能被两者同时整除的数) 编写程序, 显示从100到200之间所有能被5或6整除, 但不能被两者同时整除的数, 每行显示10个数。

- 4.12 (求满足 $n^2 > 12\,000$ 的n的最小值) 使用while循环找出满足 $n^2$ 大于12 000的最小整数n。

- 4.13 (求满足 $n^3 < 12\,000$ 的n的最大值) 用while循环找出满足 $n^3$ 小于12 000的最大整数n。

- \*4.14 (显示ASCII码字符表) 编写一个程序, 打印ASCII字符表从'!'到'~'的字符。每行打印10个字符。ASCII码表如附录B所示。

#### 4.8节

- \*4.15 (计算最大公约数) 下面是求两个整数n1和n2的最大公约数的程序清单4-8的另一种解法: 首先找出n1和n2的最小值d, 然后依次检验d, d-1, d-2, ..., 2, 1是否是n1和n2的公约数。第一个满足条件的公约数就是n1和n2的最大公约数。编写程序, 提示用户输入两个正整数, 然后显示最大公约数。

- \*\*4.16 (找出一个整数的因子) 编写程序, 读入一个整数, 然后以升序显示它的所有最小因子。例如, 若输入的整数是120, 那么输出就应该是: 2, 2, 2, 3, 5。

- \*\*4.17 (显示金字塔) 编写程序, 提示用户输入一个在1到15之间的整数, 然后显示一个金字塔形状的



图案，如下面的运行示例所示：

Enter the number of lines: 7

```

      1
    2 1 2
  3 2 1 2 3
4 3 2 1 2 3 4
5 4 3 2 1 2 3 4 5
6 5 4 3 2 1 2 3 4 5 6
7 6 5 4 3 2 1 2 3 4 5 6 7
  
```



\*4.18 (使用循环语句打印4个图案) 使用嵌套的循环语句，用四个独立的程序打印下面的图案：

图案1	图案2	图案3	图案4
1	1 2 3 4 5 6	1	1 2 3 4 5 6
1 2	1 2 3 4 5	2 1	1 2 3 4 5
1 2 3	1 2 3 4	3 2 1	1 2 3 4
1 2 3 4	1 2 3	4 3 2 1	1 2 3
1 2 3 4 5	1 2	5 4 3 2 1	1 2
1 2 3 4 5 6	1	6 5 4 3 2 1	1

\*\*4.19 (打印金字塔形的数字) 编写一个嵌套的for循环，打印下面的输出：

```

      1
    1 2 1
  1 2 4 2 1
1 2 4 8 4 2 1
1 2 4 8 16 8 4 2 1
1 2 4 8 16 32 16 8 4 2 1
1 2 4 8 16 32 64 32 16 8 4 2 1
  
```

\*4.20 (打印2到1000之间的素数) 修改程序清单4-14，打印2到1000之间，包括2和1000的所有素数，每行显示8个素数。

## 综合题

\*\*4.21 (财务应用程序：比较不同利率下的贷款) 编写程序，让用户输入贷款总额和以年为单位的贷款期限，然后显示利率从5%到8%，每次递增1/8的过程中，每月的支付额和总偿还额。下面是一个运行示例：

Loan Amount: 10000

Number of Years: 5

Interest Rate	Monthly Payment	Total Payment
5%	188.71	11322.74
5.125%	189.28	11357.13
5.25%	189.85	11391.59
...		
7.875%	202.17	12129.97
8.0%	202.76	12165.83



计算月支付额的公式，请参见程序清单2-8。

\*\*4.22 (财务应用程序：显示分期还贷时间表) 对于给定的贷款额的月支付额包括偿还本金及利息。月利息是通过月利率乘以余额(剩余本金)计算出来的。因此，每月偿还的本金等于月支付额减去月利息。编写一个程序，让用户输入贷款总额、贷款年数以及利率，然后显示分期还贷时间表。下面是一个运行示例：

Loan Amount: 10000  
 Number of Years: 1  
 Annual Interest Rate: 7%

Monthly Payment: 865.26  
 Total Payment: 10383.21

Payment#	Interest	Principal	Balance
1	58.33	806.93	9193.07
2	53.62	811.64	8381.43
...			
11	10.0	855.26	860.27
12	5.01	860.25	0.01



**注意** 最后一次偿还后，余额可能不为0。如果是这样的话，最后一个月支付额应当是正常的月支付额加上最后的余额。

**提示** 编写一个循环来打印该表。由于每个月的还贷额都是相同的，因此，应当在循环之前计算它。开始时，余额就是贷款总额。在循环的每次迭代中，计算利息及本金，然后更新余额。这个循环可能会是这样的：

```
for (i = 1; i <= numberOfYears * 12; i++) {
    interest = monthlyInterestRate * balance;
    principal = monthlyPayment - interest;
    balance = balance - principal;
    System.out.println(i + "\t\t" + interest
        + "\t\t" + principal + "\t\t" + balance);
}
```

\*4.23 (获取更精确的结果) 在计算下面的数列时，从右到左计算要比从左到右计算得到的结果更精确：

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

编写程序对上面的数列从左到右和从右到左计算的结果进行比较，这里取 $n=50000$ 。

\*4.24 (数列求和) 编写程序，计算下面数列的和：

$$\frac{1}{3} + \frac{3}{5} + \frac{5}{7} + \frac{7}{9} + \frac{9}{11} + \frac{11}{13} + \cdots + \frac{95}{97} + \frac{97}{99}$$

\*\*4.25 (计算 $\pi$ ) 使用下面的数列可以近似计算 $\pi$ ：

$$\pi = 4 \left( 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{1}{2i-1} - \frac{1}{2i+1} \right)$$

编写程序，显示当 $i=10000, 20000, \dots, 100000$ 时 $\pi$ 的值。

\*\*4.26 (计算 $e$ ) 使用下面的数列可以近似计算 $e$ ：

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots + \frac{1}{i!}$$

编写程序，显示当 $i=10000, 20000, \dots, 100000$ 时 $e$ 的值。

**提示** 由于 $i! = i \times (i-1) \times \cdots \times 2 \times 1$ ，那么  $\frac{1}{i!} = \frac{1}{i(i-1)!}$ 。

将 $e$ 和通项 $item$ 初始化为1，反复将新的 $item$ 加到 $e$ 上。新的 $item$ 由前一个 $item$ 除以 $i$ 得到，其中 $i=2, 3, 4, \dots$ 。

\*\*4.27 (显示闰年) 编写程序，显示21世纪(2001年~2100年)中所有的闰年，每行显示10个。

\*\*4.28 (显示每月第一天是星期几) 编写程序，提示用户输入年份和代表该年第一天是星期几的数字，然后在控制台上显示该年每月第一天的星期。例如，如果用户输入的年份是2005和代表2005年1月1日为星期六的6，程序应该显示如下输出(注意：0表示星期天)：

January 1, 2005 is Saturday

...

December 1, 2005 is Thursday

**\*\*4.29** (显示日历) 编写程序, 提示用户输入年份和代表该年第一天是星期几的数字, 然后在控制台上显示该年的日历表。例如, 如果用户输入年份2005和代表2005年1月1日为星期六的6, 程序应该显示该年每个月的日历, 如下所示:

January 2005						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					
...						

December 2005						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

**\*4.30** (财务应用程序: 复利值) 假设你每月在储蓄账户上存100美元, 年利率是5%。那么每月利率是  $0.05/12=0.00417$ 。在第一个月之后, 账户上的值变成:

$$100 * (1 + 0.00417) = 100.417$$

第二个月之后, 账户上的值变成:

$$(100 + 100.417) * (1 + 0.00417) = 201.252$$

第三个月之后, 账户上的值变成:

$$(100 + 201.252) * (1 + 0.00417) = 302.507$$

依此类推。

编写程序提示用户输入一个数目 (例如: 100)、年利率 (例如: 5) 以及月份数 (例如: 6), 然后显示给定月份后账户上的钱数。

**\*4.31** (财务应用程序: 计算CD价值) 假设你投资10 000美元买一张CD, 年获利率为5.75%。一个月后, 这张CD价值为

$$10000 + 10000 * 5.75 / 1200 = 10047.91$$

两个月之后, 这张CD价值为

$$10047.91 + 10047.91 * 5.75 / 1200 = 10096.06$$

三个月之后, 这张CD价值为

$$10096.06 + 10096.06 * 5.75 / 1200 = 10144.43$$

依此类推。

编写程序, 提示用户输入一个总数 (例如: 10000)、年获利率 (例如: 5.75) 以及月份数 (例如: 18), 然后显示一个表格, 如下面的运行示例所示:

```

Enter the initial deposit amount: 10000
Enter annual percentage yield: 5.75
Enter maturity period (number of months): 18

Month      CD Value
1          10047.91
2          10096.06
...
17         10846.56
18         10898.54

```



**\*\*4.32 (游戏: 彩票)** 修改程序清单3-9, 产生一个两位整数的彩票。这个数中的两个整数是两个不同的数。

**提示** 产生第一个数, 使用循环不断产生第二个数, 直到它和第一个数不同为止。

**\*\*4.33 (完全数)** 如果一个正整数等于除它本身之外其他所有除数之和, 就称之为完全数。例如: 6是第一个完全数, 因为 $6=1+2+3$ 。下一个完全数是 $28=1+2+4+7+14$ 。10000以下的完全数有四个。编写程序, 找出这四个完全数。

**\*\*\*4.34 (游戏: 石头、剪刀、布)** 练习题3.17给出玩石头-剪刀-布游戏的程序。修改这个程序, 让用户可以连续地玩这个游戏, 直到用户或者计算机连续赢两次以上为止。

**\*4.35 (加法)** 编写程序, 计算下面的和。

$$\frac{1}{1+\sqrt{2}} + \frac{1}{\sqrt{2}+\sqrt{3}} + \frac{1}{\sqrt{3}+\sqrt{4}} + \cdots + \frac{1}{\sqrt{624}+\sqrt{625}}$$

**\*\*4.36 (商业应用程序: 检测ISBN)** 使用循环简化练习题3.19。

**\*\*4.37 (十进制到二进制)** 编写程序, 提示用户输入一个十进制整数, 然后显示对应的二进制值。在这个程序中不要使用Java的`Integer.toString(int)`方法。

**\*\*4.38 (十进制到十六进制)** 编写程序, 提示用户输入一个十进制整数, 然后显示对应的十六进制值。在这个程序中不要使用Java的`Integer.toHexString(int)`方法。

**\*4.39 (财务应用程序: 求出销售总额)** 假设你已经在某百货商店开始销售工作。你的工资包括基本工资和提成。基本工资是5000美金。使用下面的方案确定你的提成率。

销售额	提成率
0.01 ~ 5000美金	8%
5000.01 ~ 10 000美金	10%
10 000.01及以上	12%

你的目标是一年挣30 000美金。编写程序找出为挣到30 000美金, 你所必须完成的最小销售额。

**4.40 (模拟: 正面或反面)** 编写程序, 模拟抛硬币一百万次, 显示出现正面和反面的次数。

**\*\*4.41 (最大数的出现次数)** 编写程序读取整数, 找出它们的最大数, 然后计算该数的出现次数。假设输入是以0结束的。假定输入是3 5 2 5 5 5 0, 程序找出最大数5, 而5出现的次数是4。

**提示** 维护max和count两个变量。max存储当前最大数, 而count存储它的出现次数。初始状态时, 将第一个数赋值给max而将count赋值为1。然后将接下来的每个数字逐个地和max进行比较。如果这个数大于max, 就将它赋值给max, 同时将count重置为1。如果这个数等于max, 就给count加1。

```

Enter numbers: 3 5 2 5 5 5 0
The largest number is 5
The occurrence count of the largest number is 4

```



\*4.42 (财务应用程序: 求出销售额) 如下改写练习题4.39:

(1) 使用for循环替代do-while循环。

(2) 允许用户自己输入COMMISSION\_SOUGHT而不是将它固定为一个常量。

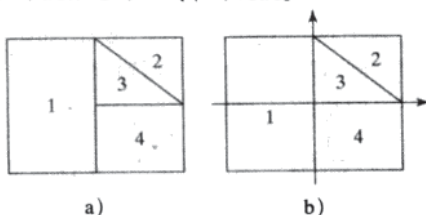
\*4.43 (模拟: 表的倒计时) 编写程序, 提示用户输入秒数, 每一秒都显示一条消息, 直到时间用完时结束程序。下面是一个运行示例:

```
Enter the number of second: 3
2 seconds remaining
1 second remaining
Stopped
```



\*\*4.44 (蒙特卡罗模拟) 一个正方形被分为更小的四部分, 如下面的图a中所示。如果将一个飞镖掷入这个正方形1 000 000次, 那么这个飞镖掷入奇数标记的区域的概率有多大? 编写程序模拟这个过程, 然后显示结果。

提示 将这个正方形的中点放在直角坐标系的原点, 如图b所示。随机产生一个正方形中的点, 然后计算这个点落在奇数标记的区域中的次数。



\*4.45 (数学方面: 组合) 编写程序, 显示从整数1到7中选择两个数字的所有组合, 还要显示所有组合的总数。

```
1 2
1 3
...
...
```



\*4.46 (计算机体系结构方面: 比特级的操作) 一个short型值用16位比特存储。编写程序, 提示用户输入一个短整型, 然后显示这个整数的16比特形式。下面是一个运行示例:

```
Enter an integer: 5
The bits are 0000000000000101
```



```
Enter an integer: -5
The bits are 1111111111111011
```



提示 需要使用按位右移运算符(>>)以及按位AND运算符(&), 它们都放在配套网站上的补充材料III.D中。