

方 法

学习目标

- 定义方法 (5.2节)。
- 调用带返回值的方法 (5.3节)。
- 调用无返回值的方法 (5.4节)。
- 按值传参 (5.5节)。
- 开发模块化的、易读、易调试和易维护的可重用代码 (5.6节)。
- 编写方法将十进制数转换为十六进制数 (5.7节)。
- 使用方法重载, 理解歧义重载 (5.8节)。
- 确定变量的作用域 (5.9节)。
- 使用Math类中的方法解决数学问题 (5.10~5.11节)。
- 在软件开发中应用方法抽象的概念 (5.12节)。
- 使用逐步求精的办法设计和实现方法 (5.12节)。

5.1 引言

假如需要分别求出从1到10、从20到30以及从35到45的整数和, 可以编写如下代码:

```
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
    sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
    sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

你会发现计算从1到10、从20到30以及从35到45的整数和, 除了开始的数和结尾的数不同之外, 其他都是非常类似的。如果可以一次性地编写好通用的代码而无须重新编写, 难道不是更好吗? 可以通过定义方法实现该功能。方法就是用来创建可重用的代码。

上面的代码可以简化为如下所示:

```
1 public static int sum(int i1, int i2) {
2     int sum = 0;
3     for (int i = i1; i <= i2; i++)
4         sum += i;
5
6     return sum;
7 }
8
9 public static void main(String[] args) {
10     System.out.println("Sum from 1 to 10 is " + sum(1, 10));
11     System.out.println("Sum from 20 to 30 is " + sum(20, 30));
12     System.out.println("Sum from 35 to 45 is " + sum(35, 45));
13 }
```

第1~7行定义一个名为sum的方法，该方法带有两个参数i1和i2。main方法中的语句调用sum(1,10)计算从1到10的整数和，sum(20,30)计算从20到30的整数和，而sum(35,45)计算从35到45的整数和。

方法是完成一个操作而组合在一起的语句组。在前面的章节里，已经使用过预定义的方法，例如：System.out.println、JOptionPane.showMessageDialog、JOptionPane.showInputDialog、Integer.parseInt、Double.parseDouble、System.exit、Math.pow和Math.random，这些方法都在Java库中定义。在本章里，将学习如何定义自己的方法以及应用方法抽象来解决复杂问题。

5.2 定义方法

定义方法的语法如下所示：

```
修饰符    返回值类型    方法名 (参数列表) {
//方法体;
}
```

我们一起来看看一个创建的方法，该方法找出两个整数中哪个数比较大。这个名为max的方法有两个int型参数：num1和num2，方法返回两个数中较大的一个。图5-1解释了这个方法的组成。

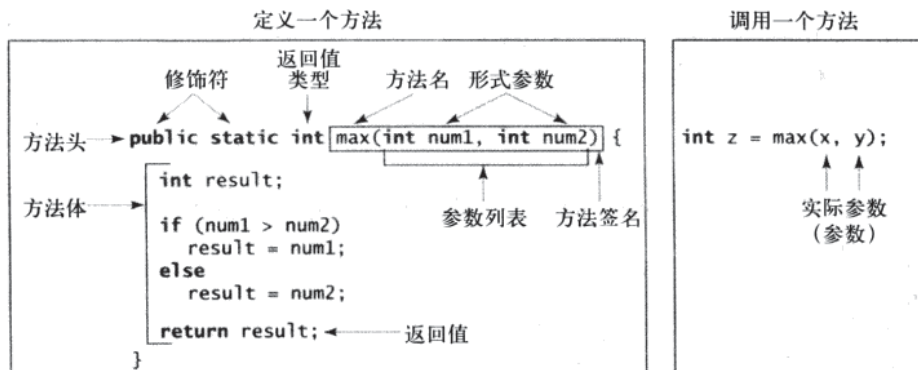


图5-1 方法定义包括方法头和方法体

方法头（method header）是指方法的修饰符（modifier）、返回值类型（return value type）、方法名（method name）和方法的参数（parameter）。本章的所有方法都使用静态修饰符，使用它的理由将在第8章中深入讨论。

方法可以返回一个值。returnValueType是方法返回值的数据类型。有些方法只是完成某些要求的操作，而不返回值。在这种情况下，returnValueType为关键字void。例如：在main方法中returnValueType就是void，在System.exit、System.out.println和JOptionPane.showMessageDialog方法中返回值类型也是如此。如果方法有返回值，则称之为带返回值的方法（value-returning method），否则就称这个方法为void方法（void method）。

定义在方法头中的变量称为形式参数（formal parameter）或者简称为形参（parameter）。参数就像占位符。当调用方法时，就给参数传递一个值，这个值称为实际参数（actual parameter）或实参（argument）。参数列表（parameter list）指明方法中参数的类型、顺序和个数。方法名和参数列表一起构成方法签名（method signature）。参数是可选的，也就是说，方法可以不包含参数。例如：Math.random()方法就没有参数。

方法体中包含一个定义方法做什么的语句集合。max方法的方法体使用一个if语句来判断哪个数较大，然后返回该数的值。为使带返回值的方法能返回一个结果，必须要使用带关键字return的返回语句，执行return语句意味着方法终止。

注意 在其他某些语言中，方法称为过程（procedure）或函数（function）。带返回值的方法称为函数，返回值类型为void的方法称为过程。

警告 在方法头中，需要对每一个参数进行独立的数据类型声明。例如：`max(int num1, int num2)`是正确的，而`max(int num1, num2)`是错误的。

注意 我们经常会说“定义方法”和“声明变量”，这里我们谈谈两者的细微差别。定义是指被定义的条目是什么，而声明通常是指为被声明的条目分配内存来存储数据。

5.3 调用方法

在创建方法时，定义方法要做什么。为了使用方法，必须调用（call或invoke）它。根据方法是否有返回值，调用方法有两种途径。

如果方法返回一个值，对方法的调用通常就当作一个值处理。例如：

```
int larger = max(3, 4);
```

调用方法`max(3,4)`并将其结果赋给变量`larger`。另一个把它当作值处理的调用例子是：

```
System.out.println(max(3, 4));
```

这条语句打印调用方法`max(3,4)`后的返回值。

如果方法返回void，对方法的调用必须是一条语句。例如，`println`方法返回void。下面的调用就是一条语句：

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

注意 在Java中，带返回值的方法也可以当作语句调用。这种情况下，函数调用者只需忽略返回值即可。虽然这种情况很少见，但是，如果调用者对返回值不感兴趣，这样也是允许的。

当程序调用一个方法时，程序控制就转移到被调用的方法。当执行完`return`语句或执行到表示方法结束的右括号时，被调用的方法将程序控制还给调用者。

程序清单5-1给出了测试`max`方法的完整程序。

程序清单5-1 TestMax.java

```
1 public class TestMax {
2     /** Main method */
3     public static void main(String[] args) {
4         int i = 5;
5         int j = 2;
6         int k = max(i, j);
7         System.out.println("The maximum between " + i +
8             " and " + j + " is " + k);
9     }
10
11     /** Return the max between two numbers */
12     public static int max(int num1, int num2) {
13         int result;
14
15         if (num1 > num2)
16             result = num1;
17         else
18             result = num2;
19
20         return result;
21     }
22 }
```

The maximum between 5 and 2 is 5



数字资源
PDF

	line#	i	j	k	num1	num2	result
	4	5					
	5		2				
Invoking max	12				5	2	
	13						undefined
	16						5
	6			5			

这个程序包括main方法和max方法。main方法与其他方法的唯一区别在于它是由Java虚拟机调用的。

main方法的方法头永远都是一样的。就像这个例子中的main方法一样，它包括修饰符public和static，返回值类型void，方法名main，String[]类型的参数。String[]表明参数是一个String型数组，数组是第6章将介绍的主题。

main中的语句可以调用main方法所在类中定义的其他方法，也可以调用别的类中定义的方法。在本例中，main方法调用与main方法在同一个类中定义的方法max(i,j)。

当调用max方法时（第6行），变量i的值5传递给max方法中的num1，变量j的值2传递给max方法中的num2。控制流程转向max方法。执行max方法。当执行max方法中的return语句时，max方法将程序控制返还给它的调用者（在此例中，调用者是main方法）。这个过程如图5-2所示。

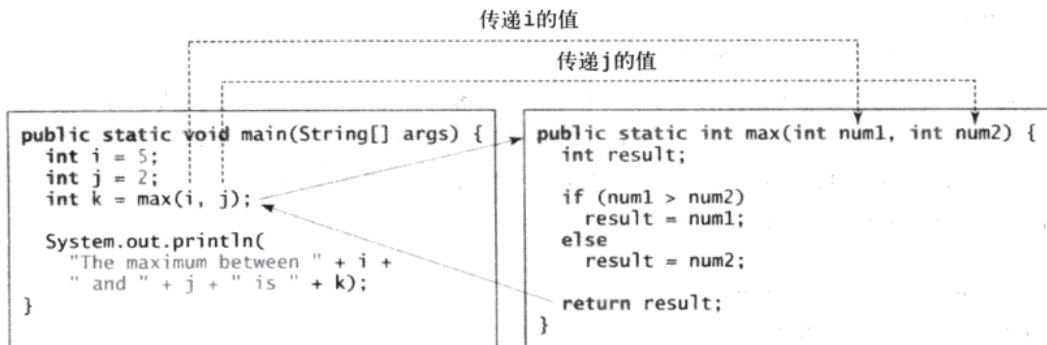
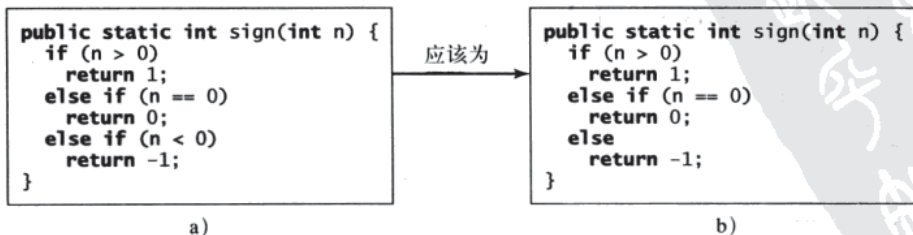


图5-2 当调用max方法时，控制流程转向max方法。一旦max方法结束，将控制返还给调用者

警告 对带返回值的方法而言，return语句是必需的。下面图a中显示的方法在逻辑上是正确的，但它会有编译错误，因为Java编译器认为该方法有可能不会返回任何值。

为了解决这个问题，删除图a中的if(n<0)，这样，编译器将发现不管if语句如何执行，总可以执行到return语句。



注意 方法能够带来代码的共享和重用。除了可以在TestMax中调用max方法，还可以在其他类中调用它。如果创建了一个新类，可以通过使用“类名.方法名”（即TestMax.max）来调用max方法。

调用堆栈

每当调用一个方法时，系统都会将参数、局部变量存储在一个称为堆栈（stack）的内存区域中，它用后进先出的方式存储数据。当一个方法调用另一个方法时，调用者的堆栈空间保持不动，新开辟的空间处理新方法的调用。一个方法结束返回到调用者时，其相应的空间也被释放。

理解调用堆栈有助于理解方法是如何调用的。`main`方法中定义的变量是*i*、*j*和*k*。`max`方法中定义的变量是*num1*、*num2*和*result*。定义在方法签名中的变量*num1*和*num2*都是方法的参数。它们的值通过方法调用进行传递。图5-3描述了堆栈中的变量。

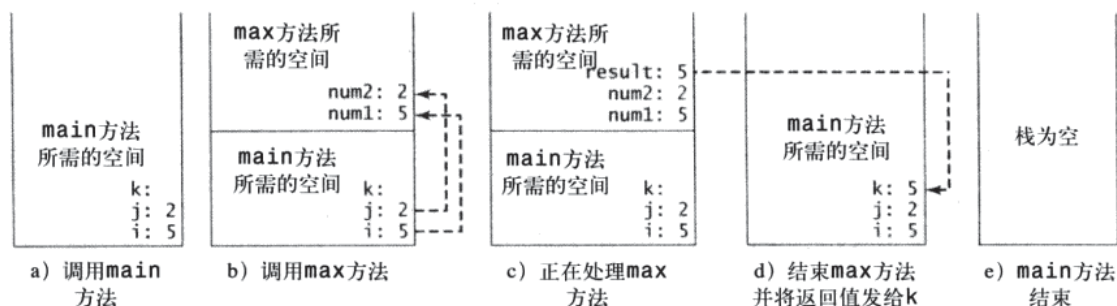


图5-3 调用`max`方法时，程序控制流转到`max`方法。一旦`max`方法结束，就将程序控制还给调用者

5.4 void方法举例

前一节给出一个带返回值方法的例子，本节将介绍如何定义和调用`void`方法。程序清单5-2给出的程序定义了一个名为`printGrade`的方法，然后调用它打印出给定分数的等级。

程序清单5-2 TestVoidMethod.java

```
1 public class TestVoidMethod {
2     public static void main(String[] args) {
3         System.out.print("The grade is ");
4         getGrade(78.5);
5
6         System.out.print("The grade is ");
7         getGrade(59.5);
8     }
9
10    public static void printGrade(double score) {
11        if (score >= 90.0) {
12            System.out.println('A');
13        }
14        else if (score >= 80.0) {
15            System.out.println('B');
16        }
17        else if (score >= 70.0) {
18            System.out.println('C');
19        }
20        else if (score >= 60.0) {
21            System.out.println('D');
22        }
23        else {
24            System.out.println('F');
25        }
26    }
27 }
```

The grade is C
The grade is F



`printGrade`方法是一个`void`方法，它不返回任何值。对`void`方法的调用必须是一条语句。因此，在`main`方法的第4行，`printGrade`方法作为一条语句调用，这条语句同其他Java语句一样，以分号结束。

为了区分`void`方法和带返回值的方法，我们重新设计`printGrade`方法使之返回一个值。称新方法为`getGrade`，返回等级，如程序清单5-3所示。

程序清单5-3 `TestReturnGradeMethod.java`

```

1 public class TestReturnGradeMethod {
2     public static void main(String[] args) {
3         System.out.print("The grade is " + getGrade(78.5));
4         System.out.print("\nThe grade is " + getGrade(59.5));
5     }
6
7     public static char getGrade(double score) {
8         if (score >= 90.0)
9             return 'A';
10        else if (score >= 80.0)
11            return 'B';
12        else if (score >= 70.0)
13            return 'C';
14        else if (score >= 60.0)
15            return 'D';
16        else
17            return 'F';
18    }
19 }

```

The grade is C
The grade is F



第7~18行定义的`getGrade`方法返回一个基于数字分值的字符等级。调用者在第3~4行调用这个方法。

调用者会在任何出现字符的地方调用`getGrade`方法。`printGrade`方法不返回任何值，它也必须作为一条语句被调用。

注意 `void`方法不需要`return`语句，但它能用于终止方法并返回到方法的调用者。它的语法是：

`return;`

这种用法很少，但是对于改变`void`方法中的正常流程控制是很有用的。例如：当分数是无效值时，下列代码就用`return`语句结束函数。

```

public static void printGrade(double score) {
    if (score < 0 || score > 100) {
        System.out.println("Invalid score");
        return;
    }

    if (score >= 90.0) {
        System.out.println('A');
    }
    else if (score >= 80.0) {
        System.out.println('B');
    }
    else if (score >= 70.0) {
        System.out.println('C');
    }
    else if (score >= 60.0) {
        System.out.println('D');
    }
    else {
        System.out.println('F');
    }
}

```

数字图书馆
PDG

5.5 参数的值传递

方法的威力在于它处理参数的能力。可以使用方法println打印任意字符串,用max方法求任意两个int值的最大值。调用方法时,需要提供实参,它们必须与方法签名中所对应的形参次序相同。这称作参数顺序匹配 (parameter order association)。例如,下面的方法打印message信息n次:

```
public static void nPrintln(String message, int n) {
    for (int i = 0; i < n; i++)
        System.out.println(message);
}
```

可以用nPrintln("Hello",3)打印"Hello" 3遍。语句nPrintln("Hello",3)把实际的字符串参数"Hello"传给参数message,把3传给n,然后打印"Hello" 3次。然而,语句nPrintln(3,"Hello")就是错的。3的数据类型不匹配第一个参数message的数据类型,第二个参数"Hello"不匹配第二个参数n。

警告 实参必须与方法签名中定义的参数在次序和数量上匹配,在类型上兼容。类型兼容是指不需要经过显式的类型转换,实参的值就可以传递给形参,例如,将int型的实参值传递给double型形参。

当调用带参数的方法时,实参的值传递给形参,这个过程称为通过值传递 (pass-by-value)。如果实参是变量而不是直接量,则将该变量的值传递给形参。无论形参在方法中是否改变,该变量都不受影响。如程序清单5-4所示,x(1)的值传给参数n,用以调用方法increment (第5行)。在该方法中n自增1 (第10行),而x的值是不论方法做了什么保持不变。

程序清单5-4 Increment.java

```
1 public class Increment {
2     public static void main(String[] args) {
3         int x = 1;
4         System.out.println("Before the call, x is " + x);
5         increment(x);
6         System.out.println("after the call, x is " + x);
7     }
8
9     public static void increment(int n) {
10        n++;
11        System.out.println("n inside the method is " + n);
12    }
13 }
```

Before the call, x is 1
n inside the method is 2
after the call, x is 1



程序清单5-5给出另一个演示值传递效果的程序。程序创建了一个能实现两个变量互换的swap方法。调用swap方法时传递两个实参。有趣的是,调用方法后,这两个实参并未改变。

程序清单5-5 TestPassByValue.java

```
1 public class TestPassByValue {
2     /** Main method */
3     public static void main(String[] args) {
4         // Declare and initialize variables
5         int num1 = 1;
6         int num2 = 2;
7
8         System.out.println("Before invoking the swap method, num1 is " +
9             num1 + " and num2 is " + num2);
10
11        // Invoke the swap method to attempt to swap two variables
12        swap(num1, num2);
```

```

13
14     System.out.println("After invoking the swap method, num1 is " +
15         num1 + " and num2 is " + num2);
16 }
17
18 /** Swap two variables */
19 public static void swap(int n1, int n2) {
20     System.out.println("\t\tInside the swap method");
21     System.out.println("\t\tBefore swapping n1 is " + n1
22         + " n2 is " + n2);
23
24     // Swap n1 with n2
25     int temp = n1;
26     n1 = n2;
27     n2 = temp;
28
29     System.out.println("\t\tAfter swapping n1 is " + n1
30         + " n2 is " + n2);
31 }
32 }

```

Before invoking the swap method, num1 is 1 and num2 is 2
 Inside the swap method
 Before swapping n1 is 1 n2 is 2
 After swapping n1 is 2 n2 is 1
 After invoking the swap method, num1 is 1 and num2 is 2



在调用swap方法（第12行）前，num1为1而num2为2。在调用swap方法后，num1仍为1，num2仍为2。它们的值没有因为调用swap方法而交换。如图5-4所示，实参num1和num2的值传递给n1和n2，但是n1和n2有自己独立于num1和num2的存储空间。所以，n1和n2的改变不影响num1和num2的内容。

另一个转变是把swap中形参的名称n1改为num1。这样做有什么效果呢？什么也不变，因为形参和实参是否同名是没有任何分别的。形参是方法中具有自己存储空间的变量。局部变量是在调用方法时分配的，当方法返回到调用者后它就消失了。

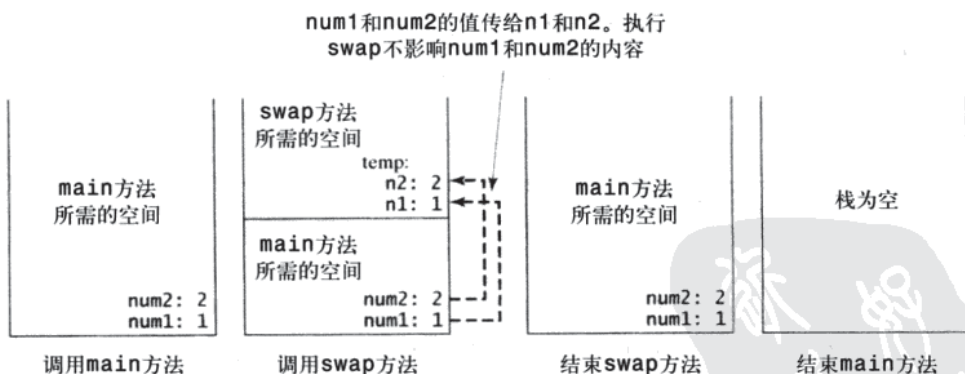


图5-4 变量的值传递给方法中的形参

注意 为了简便，Java程序员经常说将实参x传给形参y，实际含义是指将x的值传递给y。

5.6 模块化代码

使用方法可以减少冗余的代码，提高代码的复用性。方法也可以用来模块化代码，以提高程序的质量。

程序清单4-8给出的程序提示用户输入两个整数，然后显示它们的最大公约数。可以使用一种方法改写这个程序，如程序清单5-6所示。

程序清单5-6 GreatestCommonDivisorMethod.java

```

1 import java.util.Scanner;
2
3 public class GreatestCommonDivisorMethod {
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        System.out.println("The greatest common divisor for " + n1 +
16            " and " + n2 + " is " + gcd(n1, n2));
17    }
18
19    /** Return the gcd of two integers */
20    public static int gcd(int n1, int n2) {
21        int gcd = 1; // Initial gcd is 1
22        int k = 2; // Possible gcd
23
24        while (k <= n1 && k <= n2) {
25            if (n1 % k == 0 && n2 % k == 0)
26                gcd = k; // Update gcd
27            k++;
28        }
29
30        return gcd; // Return gcd
31    }
32 }

```

Enter first integer: 45
 Enter second integer: 75
 The greatest common divisor for 45 and 75 is 15



通过将求最大公约数的代码封装在一个方法中，这个程序就具备了以下几个优点：

1) 它将计算最大公约数的问题和main方法中的其他代码分隔开，这样做会使逻辑更加清晰而且程序的可读性更强。

2) 计算最大公约数的错误就限定在gcd方法中，这样就缩小了调试的范围。

3) 现在，其他程序就可以重复使用gcd方法。

程序清单5-7应用了代码模块化的概念来对程序清单4-14进行改进。

程序清单5-7 PrimeNumberMethod.java

```

1 public class PrimeNumberMethod {
2     public static void main(String[] args) {
3         System.out.println("The first 50 prime numbers are \n");
4         printPrimeNumbers(50);
5     }
6
7     public static void printPrimeNumbers(int numberOfPrimes) {
8         final int NUMBER_OF_PRIMES_PER_LINE = 10; // Display 10 per line
9         int count = 0; // Count the number of prime numbers
10        int number = 2; // A number to be tested for primeness
11

```

```

12 // Repeatedly find prime numbers
13 while (count < numberOfPrimes) {
14     // Print the prime number and increase the count
15     if (isPrime(number)) {
16         count++; // Increase the count
17
18         if (count % NUMBER_OF_PRIMES_PER_LINE == 0) {
19             // Print the number and advance to the new line
20             System.out.printf("%-5s\n", number);
21         }
22         else
23             System.out.printf("%-5s", number);
24     }
25
26     // Check whether the next number is prime
27     number++;
28 }
29 }
30
31 /** Check whether number is prime */
32 public static boolean isPrime(int number) {
33     for (int divisor = 2; divisor <= number / 2; divisor++) {
34         if (number % divisor == 0) { // If true, number is not prime
35             return false; // number is not a prime
36         }
37     }
38
39     return true; // number is prime
40 }
41 }

```

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



将一个大问题分成两个子问题，这样，新的程序会更易读，也更易于调试。而且，其他程序也可以使用方法 `printPrimeNumbers` 和 `isPrime`。

5.7 问题：将十进制数转换为十六进制数

计算机系统的程序设计中会经常用到十六进制数。附录F介绍数字系统。本节介绍将十进制数转换为十六进制数的程序。

将十进制数 d 转换为十六进制数就是找到满足以下条件的十六进制数 $h_n, h_{n-1}, h_{n-2}, \dots, h_2, h_1$ 和 h_0 ：

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \dots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

这些数可以通过不断地用 d 除以16直到商为零而得到。依次得到的余数是 $h_n, h_{n-1}, h_{n-2}, \dots, h_2, h_1$ 和 h_0 。

例如：十进制数123被转换为十六进制数7B。这个转换过程如下所示：

$$\begin{array}{r}
 \begin{array}{r}
 0 \\
 16 \overline{) 123} \\
 \underline{0} \\
 123 \\
 \underline{112} \\
 11
 \end{array}
 \qquad
 \begin{array}{r}
 7 \longleftarrow \text{商} \\
 16 \overline{) 123} \\
 \underline{112} \\
 11 \longleftarrow \text{余数} \\
 \downarrow \\
 h_1
 \end{array}
 \end{array}$$


程序清单5-8给出程序，提示用户输入一个十进制数，然后将它转换为一个字符串形式的十六进制数。

程序清单5-8 **Decimal2HexConversion.java**

```

1 import java.util.Scanner;
2
3 public class Decimal2HexConversion {
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 a decimal integer
10        System.out.print("Enter a decimal number: ");
11        int decimal = input.nextInt();
12
13        System.out.println("The hex number for decimal " +
14            decimal + " is " + decimalToHex(decimal) );
15    }
16
17    /** Convert a decimal to a hex as a string */
18    public static String decimalToHex(int decimal) {
19        String hex = "";
20
21        while (decimal != 0) {
22            int hexValue = decimal % 16;
23            hex = toHexChar(hexValue) + hex;
24            decimal = decimal / 16;
25        }
26
27        return hex;
28    }
29
30    /** Convert an integer to a single hex digit in a character */
31    public static char toHexChar(int hexValue) {
32        if (hexValue <= 9 && hexValue >= 0)
33            return (char)(hexValue + '0');
34        else // hexValue <= 15 && hexValue >= 10
35            return (char)(hexValue - 10 + 'A');
36    }
37 }

```

Enter a decimal number: 1234 
The hex number for decimal 1234 is 4D2



	line#	decimal	hex	hexValue	toHexChar(hexValue)
	19	1234	""		
iteration 1	22			2	
	23		"2"		2
	24	77			
iteration 2	22			13	
	23		"D2"		D
	24	4			
iteration 3	22			4	
	23		"4D2"		4
	24	0			

程序使用`decimalToHex`方法（第18~28行）将一个十进制整数转换为一个字符串形式的十六进制数。该方法得到这个十进制整数整除16之后的余数（第22行）。通过调用`toHexChar`方法将得到的余数转换为字符串（第23行）。接下来，这个字符被追加在表示十六进制数的字符串的后面（第23行）。这个表示十六进制数的字符串初始时空（第19行）。对这个十进制数除以16，就从该数中去掉一个十六进制数字（第24行）。方法在一个循环中重复执行这些操作，直到商是0为止（第21~25行）。

方法`toHexChar`（第31~36行）将0到15之间的十六进制数转换为一个十六进制字符。如果`hexValue`在0到9之间，那它就被转换为`(char)(hexValue+'0')`（第33行）。回顾一下，当一个字符和一个整数相加时，计算时使用的是字符的统一码。例如：如果`hexValue`为5，那么`(char)(hexValue+'0')`返回'5'。类似地，如果`hexValue`在10到15之间，那么它就被转换为`(char)(hexValue+'A')`（第35行）。例如，如果`hexValue`是11，那么`(char)(hexValue+'A')`返回'B'。

5.8 重载方法

前面用到的`max`方法只能用于`int`型数据类型。但是，如果需要决定两个浮点数中哪个较大，该怎么办呢？解决办法是创建另一个方法名相同但参数不同的方法，代码如下所示：

```
public static double max(double num1, double num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}
```

如果调用带`int`型参数的`max`方法，就将调用需要`int`型参数的`max`方法；如果调用带`double`型参数的`max`方法，就将调用需要`double`型参数的`max`方法。这称为方法重载（method overloading）。也就是说，在一个类中有两个方法，它们具有相同的名字，但有不同的参数列表。Java编译器根据方法签名决定使用哪个方法。

程序清单5-9是一个创建了三个方法的程序。第一个方法求最大整数，第二个方法求最大双精度数，而第三个方法求三个双精度数中的最大值。所有这三个方法都被命名为`max`。

程序清单5-9 TestMethodOverloading.java

```
1 public class TestMethodOverloading {
2     /** Main method */
3     public static void main(String[] args) {
4         // Invoke the max method with int parameters
5         System.out.println("The maximum between 3 and 4 is "
6             + max(3, 4));
7
8         // Invoke the max method with the double parameters
9         System.out.println("The maximum between 3.0 and 5.4 is "
10            + max(3.0, 5.4));
11
12        // Invoke the max method with three double parameters
13        System.out.println("The maximum between 3.0, 5.4, and 10.14 is "
14            + max(3.0, 5.4, 10.14));
15    }
16
17    /** Return the max between two int values */
18    public static int max(int num1, int num2) {
19        if (num1 > num2)
20            return num1;
21        else
22            return num2;
23    }
}
```

```

24
25  /** Find the max between two double values */
26  public static double max(double num1, double num2) {
27      if (num1 > num2)
28          return num1;
29      else
30          return num2;
31  }
32
33  /** Return the max among three double values */
34  public static double max(double num1, double num2, double num3) {
35      return max(max(num1, num2), num3);
36  }
37 }

```

The maximum between 3 and 4 is 4
 The maximum between 3.0 and 5.4 is 5.4
 The maximum between 3.0, 5.4, and 10.14 is 10.14



当调用`max(3,4)`（第6行）时，调用的是求两个整数中较大值的`max`方法。当调用`max(3.0,5.4)`（第10行）时，调用的是求两个双精度数中较大值的`max`方法。当调用`max(3.0,5.4,10.14)`（第14行）时，调用的是求三个双精度数中最大数的`max`方法。

可以调用像`max(2,2.5)`这样带一个`int`值和一个`double`值的`max`方法吗？如果能，该调用哪一个`max`方法呢？第一个问题的答案是肯定的。第二个问题的答案是调用求两个`double`数中较大值的方法。实参值2被自动转换为`double`值，然后传递给这个方法。

你可能会感到奇怪，为什么调用`max(3,4)`时不会使用`max(double,double)`呢？其实，`max(double,double)`和`max(int,int)`与`max(3,4)`都是可能的匹配。调用方法时，Java编译器寻找最精确匹配的方法。因为方法`max(int,int)`比`max(double,double)`更精确，所以调用`max(3,4)`时使用的是`max(int,int)`。

注意 被重载的方法必须具有不同的参数列表。不能基于不同修饰符或返回值类型来重载方法。

注意 有时调用一个方法时，会有两个或更多可能的匹配，但是，编译器无法判断哪个是最精确的匹配。这称为歧义调用（ambiguous invocation）。歧义调用会产生一个编译错误。考虑如下代码：

```

public class AmbiguousOverloading {
    public static void main(String[] args) {
        System.out.println(max(1, 2));
    }

    public static double max(int num1, double num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }

    public static double max(double num1, int num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }
}

```

`max(int,double)`和`max(double,int)`都有可能与`max(1,2)`匹配。由于两个方法谁也不比谁更精确，所以这个调用是有歧义的，它会导致一个编译错误。

PDF

5.9 变量的作用域

变量的作用域 (scope of a variable) 是指变量可以在程序中引用的范围。在方法中定义的变量称为局部变量 (local variable)。

局部变量的作用域从声明变量的地方开始, 直到包含该变量的块结束为止。局部变量都必须在使用之前进行声明和赋值。

参数实际上就是一个局部变量。一个方法的参数的作用域涵盖整个方法。

在for循环头中初始动作部分声明的变量, 其作用域是整个for循环。但是在for循环体内声明的变量, 其作用域只限于循环体内, 是从它的声明处开始, 到包含该变量的块结束为止, 如图5-5所示。

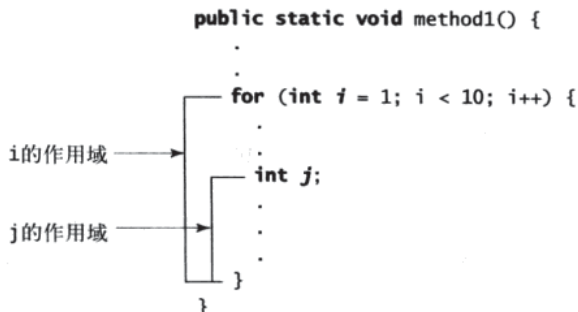


图5-5 在for循环头中初始动作部分声明的变量, 其作用域是整个for循环

可以在一个方法中的不同块里声明同名的局部变量, 但是, 不能在嵌套块中或同一块中两次声明同一个局部变量, 如图5-6所示。

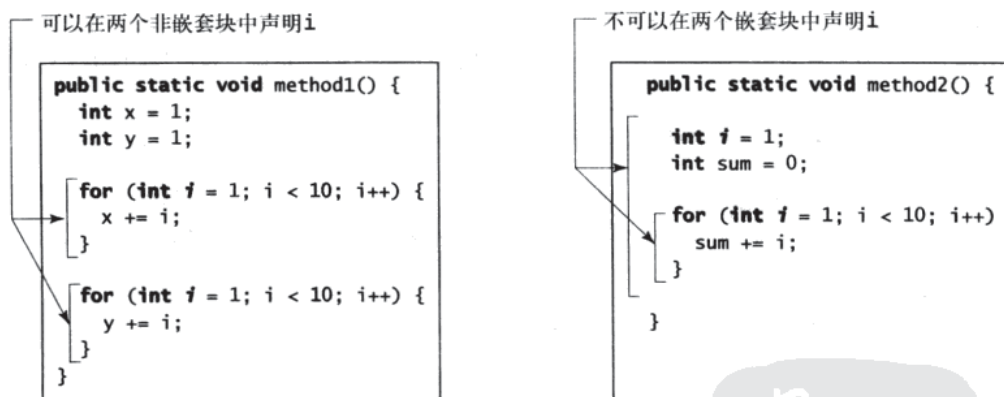


图5-6 一个变量可以在非嵌套的块中多次声明, 而在嵌套块中只能声明一次

警告 不要在块内声明一个变量然后企图在块外使用它。下面是一个常见错误的例子:

```

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

```

```

System.out.println(i);

```

因为变量*i*没有在for循环外定义, 所以最后一条语句就会产生一个语法错误。

5.10 Math数学类

Math类包含完成基本数学函数所需的方法。在程序清单2-8中已经使用过方法`pow(a,b)`计算 a^b , 在程序清单3-4中也使用过方法`Math.random()`。本节介绍Math类中其他有用的方法。这些方法分为三类: 三角函数方法 (trigonometric method)、指数函数方法 (exponent method) 和服务方法 (service method)。

除了这些方法之外，Math类还提供了两个很有用的double型常量，PI和E（自然对数的底）。可以在任意程序中用Math.PI和Math.E的形式来使用这两个常量。

5.10.1 三角函数方法

Math类包含下面的三角函数方法：

```
/** Return the trigonometric sine of an angle in radians */
public static double sin(double radians)

/** Return the trigonometric cosine of an angle in radians */
public static double cos(double radians)

/** Return the trigonometric tangent of an angle in radians */
public static double tan(double radians)

/** Convert the angle in degrees to an angle in radians */
public static double toRadians(double degree)

/** Convert the angle in radians to an angle in degrees */
public static double toDegrees(double radians)

/** Return the angle in radians for the inverse of sin */
public static double asin(double a)

/** Return the angle in radians for the inverse of cos */
public static double acos(double a)

/** Return the angle in radians for the inverse of tan */
public static double atan(double a)
```

sin、cos和tan的参数都是以弧度为单位的角。asin、acos和atan的返回值是在 $-\pi/2$ 到 $\pi/2$ 之间的一个弧度值。1度相当于 $\pi/180$ 弧度，90度相当于 $\pi/2$ 弧度，而30度相当于 $\pi/6$ 弧度。

例如：

```
Math.toDegrees(Math.PI / 2) returns 90.0
Math.toRadians(30) returns  $\pi/6$ 
Math.sin(0) returns 0.0
Math.sin(Math.toRadians(270)) returns -1.0
Math.sin(Math.PI / 6) returns 0.5
Math.sin(Math.PI / 2) returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6) returns 0.866
Math.cos(Math.PI / 2) returns 0
Math.asin(0.5) returns  $\pi/6$ 
```

5.10.2 指数函数方法

Math类中有五个与指数函数有关的方法：

```
/** Return e raised to the power of x ( $e^x$ ) */
public static double exp(double x)

/** Return the natural logarithm of x ( $\ln(x) = \log_e(x)$ ) */
public static double log(double x)

/** Return the base 10 logarithm of x ( $\log_{10}(x)$ ) */
public static double log10(double x)

/** Return a raised to the power of b ( $a^b$ ) */
public static double pow(double a, double b)

/** Return the square root of x ( $\sqrt{x}$ ) for  $x \geq 0$  */
public static double sqrt(double x)
```

例如：



```

Math.exp(1) returns 2.71828
Math.log(Math.E) returns 1.0
Math.log10(10) returns 1.0
Math.pow(2, 3) returns 8.0
Math.pow(3, 2) returns 9.0
Math.pow(3.5, 2.5) returns 22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5) returns 3.24

```

5.10.3 取整方法

Math类包括五个取整方法:

```

/** x is rounded up to its nearest integer. This integer is
 * returned as a double value. */
public static double ceil(double x)

/** x is rounded down to its nearest integer. This integer is
 * returned as a double value. */
public static double floor(double x)

/** x is rounded to its nearest integer. If x is equally close
 * to two integers, the even one is returned as a double. */
public static double rint(double x)

/** Return (int)Math.floor(x + 0.5). */
public static int round(float x)

/** Return (long)Math.floor(x + 0.5). */
public static long round(double x)

```

例如:

```

Math.ceil(2.1) returns 3.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -3.0
Math.rint(2.1) returns 2.0
Math.rint(-2.0) returns -2.0
Math.rint(-2.1) returns -2.0
Math.rint(2.5) returns 2.0
Math.rint(3.5) returns 4.0
Math.rint(-2.5) returns -2.0
Math.round(2.6f) returns 3 // Returns int
Math.round(2.0) returns 2 // Returns long
Math.round(-2.0f) returns -2
Math.round(-2.6) returns -3

```

5.10.4 min、max和abs方法

重载min和max方法以返回两个数(int、long、float或double型)的最小值和最大值。例如, max(3.4,5.0)返回5.0, 而min(3,2)返回2。

重载abs方法以返回一个数(int、long、float或double型)的绝对值。例如:

```

Math.max(2, 3) returns 3
Math.max(2.5, 3) returns 3.0
Math.min(2.5, 3.6) returns 2.5
Math.abs(-2) returns 2
Math.abs(-2.1) returns 2.1

```

5.10.5 random方法

你已经使用过`random()`方法,生成大于等于0.0且小于1.0的`double`型随机数 ($0.0 \leq \text{Math.random()} < 1.0$)。这个方法十分有用。可以使用它编写简单的表达式,生成任意范围的随机数。例如:

`(int) (Math.random() * 10)` → 返回0到9之间的一个随机整数

`50 + (int) (Math.random() * 50)` → 返回50到99之间的一个随机整数

通常,

`a + Math.random() * b` → 返回一个在`a`到`a+b`之间但不包括`a+b`的随机数

提示 可以在网站<http://java.sun.com/javase/6/docs/api/index.html>上在线查阅`Math`类的完整文档,如图5-7所示。

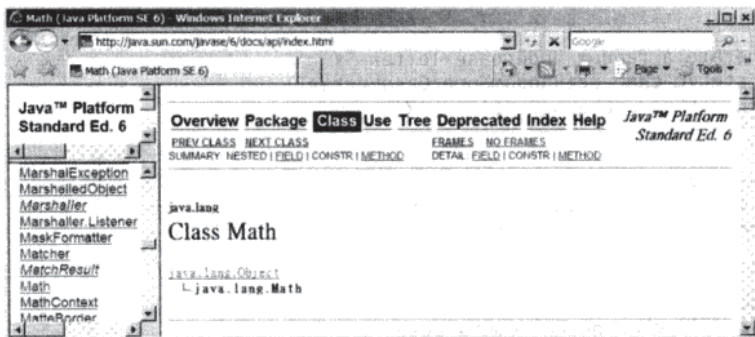


图5-7 可以在线查看Java API文档

注意 并非所有的类都需要`main`方法。`Math`类和`JOptionPane`类都没有`main`方法。这些类中所包含的方法主要是为了供其他类使用。

5.11 实例学习：生成随机字符

计算机程序处理的是数值数据和字符。前面已经看到了许多涉及数值数据的例子。了解字符和如何处理字符也是很重要的。本节给出生成随机字符的例子。

正如2.13节所介绍的,每个字符都有一个唯一的在十六进制数0到FFFF(即十进制的65 535)之间的统一码。生成一个随机字符就是使用下面的表达式,生成从0到65 535之间的一个随机整数(注意:因为 $0 \leq \text{Math.random()} < 1.0$, 必须给65 535上加1):

`(int)(Math.random() * (65535 + 1))`

现在让我们来考虑如何生成一个随机小写字母。小写字母的统一码是一串连续的整数,从小写字母'a'的统一码开始,然后是'b'、'c'、...和'z'的统一码。'a'的统一码是:

`(int)'a'`

所以, `(int)'a'`到`(int)'z'`之间的随机整数是:

`(int)((int)'a' + Math.random() * ((int)'z' - (int)'a' + 1))`

正如2.13.3节中所讨论的,所有的数字操作符都可以应用到`char`操作数上。如果另一个操作数是数字或字符,那么`char`型操作数就会被转换成数字。这样,前面的表达式就可以简化为如下所示:

`'a' + Math.random() * ('z' - 'a' + 1)`

这样,随机的小写字母是:

`(char)('a' + Math.random() * ('z' - 'a' + 1))`

推广前面的讨论, 可以生成任意两个字符ch1和ch2之间的随机字符, 其中 $ch1 < ch2$, 如下所示:

```
(char)(ch1 + Math.random() * (ch2 - ch1 + 1))
```

这是一个简单但却很有用的发现。在程序清单5-10中创建一个名为RandomCharacter的类, 它有五个重载的方法, 随机获取某种特定类型的字符。可以在以后的项目中使用这些方法。

程序清单5-10 RandomCharacter.java

```
1 public class RandomCharacter {
2     /** Generate a random character between ch1 and ch2 */
3     public static char getRandomCharacter(char ch1, char ch2) {
4         return (char)(ch1 + Math.random() * (ch2 - ch1 + 1));
5     }
6
7     /** Generate a random lowercase letter */
8     public static char getRandomLowerCaseLetter() {
9         return getRandomCharacter('a', 'z');
10    }
11
12    /** Generate a random uppercase letter */
13    public static char getRandomUpperCaseLetter() {
14        return getRandomCharacter('A', 'Z');
15    }
16
17    /** Generate a random digit character */
18    public static char getRandomDigitCharacter() {
19        return getRandomCharacter('0', '9');
20    }
21
22    /** Generate a random character */
23    public static char getRandomCharacter() {
24        return getRandomCharacter('\u0000', '\uFFFF');
25    }
26 }
```

程序清单5-11给出一个测试程序, 显示175个随机的小写字母。

程序清单5-11 TestRandomCharacter.java

```
1 public class TestRandomCharacter {
2     /** Main method */
3     public static void main(String[] args) {
4         final int NUMBER_OF_CHARS = 175;
5         final int CHARS_PER_LINE = 25;
6
7         // Print random characters between 'a' and 'z', 25 chars per line
8         for (int i = 0; i < NUMBER_OF_CHARS; i++) {
9             char ch = RandomCharacter.getRandomLowerCaseLetter();
10            if ((i + 1) % CHARS_PER_LINE == 0)
11                System.out.println(ch);
12            else
13                System.out.print(ch);
14        }
15    }
16 }
```

```
gmjssohezfkg tazqgmswfc lrao
pnrunu lnw mazt l fjedmpchcif
la lqdgiv xkxpbzu lrmqbhikr
lbnrjlsopfxahssqh wuuljvbe
xbhdotzhpehbqmuwsfktwsoli
cbuwkzgxpm tzi h gatds l vwbwz
bfesok lwbhnooygi gzd x uqni
```



第9行调用定义在RandomCharacter类中的方法getRandomLowerCaseLetter()。注意,虽然方法getRandomLowerCaseLetter()没有任何参数,但是在定义和调用这类方法时仍然需要使用括号。

5.12 方法抽象和逐步求精

开发软件的关键在于应用抽象的概念。从本书中将学习到多种层次的抽象。方法抽象(method abstraction)是通过将方法的使用和它的实现分离来实现的。用户在不知道方法是如何实现的情况下,就可以使用方法。方法的实现细节封装在方法内,对使用该方法的用户来说是隐藏的。这就称为信息隐藏(information hiding)或封装(encapsulation)。如果决定改变方法的实现,但只要不改变方法签名,用户的程序就不受影响。方法的实现对用户隐藏在“黑匣子”中,如图5-8所示。

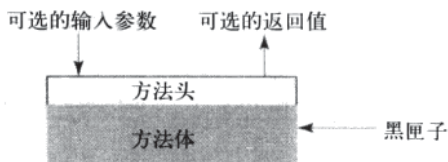


图5-8 方法体可以看做是一个包括该方法实现细节的黑匣子

前面已经使用过方法System.out.print来显示一个字符串,使用方法JOptionPane.showInputDialog从对话框中读入一个字符串,然后用max方法求最大数也知道了怎样在程序中编写代码来调用这些方法。但是作为这些方法的使用者,你并不需要知道它们是怎样实现的。

方法抽象的概念可以应用于程序的开发过程中。当编写一个大程序时,可以使用“分治”(divid-and-conquer)策略,也称之为逐步求精(stepwise refinement),将大问题分解成子问题。子问题又分解成更小、更容易处理的问题。

假设要编写一个程序,显示给定年月的日历。程序提示用户输入年份和月份,然后显示该月的整个日历,如下面的运行示例所示:

Enter full year (e.g., 2001):

Enter month in number between 1 and 12:

June 2006

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	

让我们用这个例子演示分治法。

5.12.1 自顶向下的设计

如何开始编写这样一个程序呢?你会立即开始编写代码吗?程序员新手常常一开始就想解决每一个细节。尽管细节对最终程序很重要,但在前期过多关注细节会阻碍解决问题的进程。为使解决问题的流程尽可能地流畅,本例先用方法抽象把细节与设计分离,只在最后才实现这些细节。

对本例来说,先把问题拆分成两个子问题:读取用户输入和打印该月的日历。在这一步,应该考虑还能分解成什么子问题,而不是用什么方法来读取输入和打印整个日历。可以画一个结构图,这有助于看清楚问题的分解过程(参见图5-9a)。

打印给定月份的日历问题可以分解成两个子问题:打印日历的标题和日历的主体,如图5-9b所示。月历的标题由三行组成:年月、虚线、每周七天的星期名称。需要通过表示月份的数字(例如:1)来确

定该月的全称（例如：January）。这个步骤是由getMonthName来完成的（参见图5-10a）。

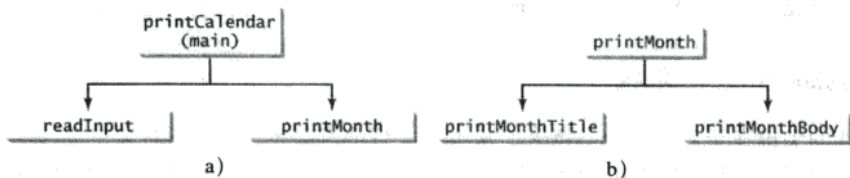


图5-9 结构图显示将打印日历printCalendar问题分解成两个子问题——读取输入readInput和打印日历printMonth，而将printMonth分解成两个更小的问题——打印日历头printMonthTitle和打印日历体printMonthBody

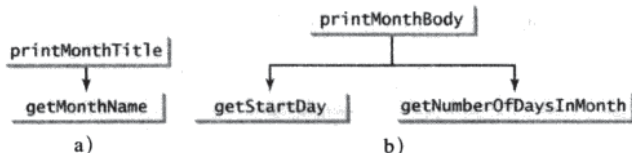


图5-10 a) 需要getMonthName才能完成printMonthTitle,
b) printMonthBody被细化成几个更小的问题

为了打印日历的主体，需要知道这个月的第一天是星期几（getStartDay），以及该月有多少天（getNumberOfDaysInMonth），如图5-10b所示。例如：2005年12月有31天，2005年12月1号是星期四。

怎样才能知道一个月的第一天是星期几呢？有几种方法可以求得。这里，我们采用下面的方法。假设知道1800年1月1日是星期三（startDay 1800=3），然后计算1800年1月1日和日历月份的第一天之间相差的总天数（totalNumberOfDays）。因为每个星期有7天，所以日历月份第一天的星期就是 $(totalNumberOfDays + startDay 1800) \% 7$ 。这样getStartDay问题就可以进一步细化为getTotalNumberOfDays，如图5-11a所示。

要计算总天数，需要知道该年是否是闰年以及每个月的天数。所以，getTotalNumberOfDays可以进一步细化成两个子问题：isLeapYear和getNumberOfDaysInMonth，如图5-11b所示。完整的结构图如图5-12所示。

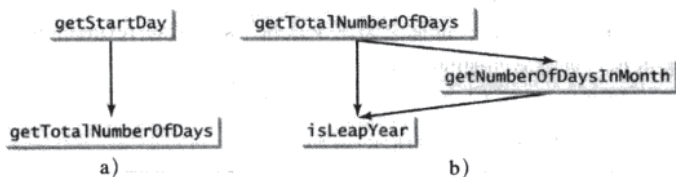


图5-11 a) 需要getTotalNumberOfDays才能得到getStartDay,
b) 问题getTotalNumberOfDays可以细化成两个更小的问题

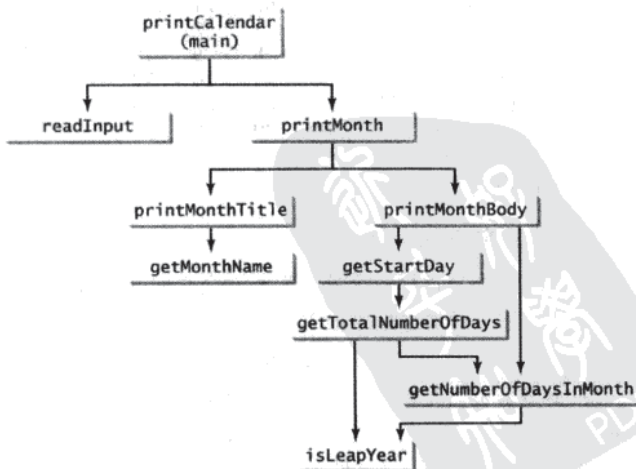


图5-12 结构图显示程序中子问题之间的层次关系

5.12.2 自顶向下和自底向上的实现

现在我们把注意力转移到实现上。通常，一个子问题对应于实现中的一个方法，即使某些子问题太简单，以至于都不需要方法来实现。需要决定哪些模块要用方法实现，而哪些模块要结合在另一个方法中。这种决策应该基于所做的选择，整个程序是否更易读做出的。在本例中，子问题`readInput`只要在`main`方法中即可实现。

可以采用“自顶向下”或“自底向上”的办法。“自顶向下”方法是自上而下，每次实现结构图中的一个方法。等待实现的方法可以用待完善方法代替。待完善方法（stub）是方法的一个简单但不完整的版本。使用待完善方法可以快速地构建程序的框架。首先实现`main`方法，然后使用`printMonth`方法的stub。例如，让`printMonth`中的待完善部分显示年份和月份，那么程序就以下面的形式开始：

```
public class PrintCalendar {
    /** Main method */
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter year
        System.out.print("Enter full year (e.g., 2001): ");
        int year = input.nextInt();

        // Prompt the user to enter month
        System.out.print("Enter month as number between 1 and 12: ");
        int month = input.nextInt();

        // Print calendar for the month of the year
        printMonth(year, month);
    }

    /** A stub for printMonth may look like this */
    public static void printMonth(int year, int month) {
        System.out.print(month + " " + year);
    }

    /** A stub for printMonthTitle may look like this */
    public static void printMonthTitle(int year, int month) {
    }

    /** A stub for getMonthBody may look like this */
    public static void printMonthBody(int year, int month) {
    }

    /** A stub for getMonthName may look like this */
    public static String getMonthName(int month) {
        return "January"; // A dummy value
    }

    /** A stub for getMonthName may look like this */
    public static int getStartDay(int year, int month) {
        return 1; // A dummy value
    }

    /** A stub for getTotalNumberOfDays may look like this */
    public static int getTotalNumberOfDays(int year, int month) {
        return 10000; // A dummy value
    }

    /** A stub for getNumberOfDaysInMonth may look like this */
    public static int getNumberOfDaysInMonth(int year, int month) {
        return 31; // A dummy value
    }

    /** A stub for getTotalNumberOfDays may look like this */
    public static boolean isLeapYear(int year) {
    }
}
```

```

        return true; // A dummy value
    }
}

```

编译和测试这个程序，然后修改所有的错误。现在，可以实现printMonth方法。对printMonth中调用的方法，可以继续使用待完善方法。

自底向上方法是从下向上每次实现结构图中的一个方法，对每个实现的方法都写一个测试程序进行测试。自顶向下和自底向上都是不错的方法。它们都是逐步地实现方法，这有助于分离程序设计错误，使调试变得容易。有时，这两种方法可以一起使用。

5.12.3 实现细节

方法isLeapYear(int year)可以使用下列代码实现：

```
return (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0));
```

使用下面的事实实现getTotalNumberOfDaysInMonth(int year,int month)方法：

- 1) 一月、三月、五月、七月、八月、十月和十二月都有31天。
- 2) 四月、六月、九月和十一月都有30天。
- 3) 二月通常有28天，但是在闰年有29天。因此，一年通常有365天，闰年有366天。

要实现getTotalNumberOfDays(int year, int month)方法，需要计算1800年1月1日和日历月份的第一天之间的总天数 (totalNumberOfDays)。可以求出1800年到该日历年的总天数，然后求出在该日历年中在日历月份之前的总天数。这两个总天数相加就是totalNumberOfDays。

要打印日历体，首先在第一天之前填充一些空格，然后为每个星期打印一条线。

完整的程序见程序清单5-12。

程序清单5-12 PrintCalendar.java

```

1 import java.util.Scanner;
2
3 public class PrintCalendar {
4     /** Main method */
5     public static void main(String[] args) {
6         Scanner input = new Scanner(System.in);
7
8         // Prompt the user to enter year
9         System.out.print("Enter full year (e.g., 2001): ");
10        int year = input.nextInt();
11
12        // Prompt the user to enter month
13        System.out.print("Enter month in number between 1 and 12: ");
14        int month = input.nextInt();
15
16        // Print calendar for the month of the year
17        printMonth(year, month);
18    }
19
20    /** Print the calendar for a month in a year */
21    public static void printMonth(int year, int month) {
22        // Print the headings of the calendar
23        printMonthTitle(year, month);
24
25        // Print the body of the calendar
26        printMonthBody(year, month);
27    }
28
29    /** Print the month title, e.g., May, 1999 */
30    public static void printMonthTitle(int year, int month) {
31        System.out.println("        " + getMonthName(month)
32            + " " + year);
33        System.out.println("-----");
34        System.out.println(" Sun Mon Tue Wed Thu Fri Sat");

```

数字图书馆
PDG


```

35 }
36
37 /** Get the English name for the month */
38 public static String getMonthName(int month) {
39     String monthName = "";
40     switch (month) {
41         case 1: monthName = "January"; break;
42         case 2: monthName = "February"; break;
43         case 3: monthName = "March"; break;
44         case 4: monthName = "April"; break;
45         case 5: monthName = "May"; break;
46         case 6: monthName = "June"; break;
47         case 7: monthName = "July"; break;
48         case 8: monthName = "August"; break;
49         case 9: monthName = "September"; break;
50         case 10: monthName = "October"; break;
51         case 11: monthName = "November"; break;
52         case 12: monthName = "December";
53     }
54
55     return monthName;
56 }
57
58 /** Print month body */
59 public static void printMonthBody(int year, int month) {
60     // Get start day of the week for the first date in the month
61     int startDay = getStartDay(year, month);
62
63     // Get number of days in the month
64     int numberOfDaysInMonth = getNumberOfDaysInMonth(year, month);
65
66     // Pad space before the first day of the month
67     int i = 0;
68     for (i = 0; i < startDay; i++)
69         System.out.print(" ");
70
71     for (i = 1; i <= numberOfDaysInMonth; i++) {
72         System.out.printf("%4d", i);
73
74         if ((i + startDay) % 7 == 0)
75             System.out.println();
76     }
77
78     System.out.println();
79 }
80
81 /** Get the start day of month/1/year */
82 public static int getStartDay(int year, int month) {
83     final int START_DAY_FOR_JAN_1_1800 = 3;
84     // Get total number of days from 1/1/1800 to month/1/year
85     int totalNumberOfDays = getTotalNumberOfDays(year, month);
86
87     // Return the start day for month/1/year
88     return (totalNumberOfDays + START_DAY_FOR_JAN_1_1800) % 7;
89 }
90
91 /** Get the total number of days since January 1, 1800 */
92 public static int getTotalNumberOfDays(int year, int month) {
93     int total = 0;
94
95     // Get the total days from 1800 to 1/1/year
96     for (int i = 1800; i < year; i++)
97         if (isLeapYear(i))
98             total = total + 366;
99     else

```



```

100         total = total + 365;
101
102         // Add days from Jan to the month prior to the calendar month
103         for (int i = 1; i < month; i++)
104             total = total + getNumberOfDaysInMonth(year, i);
105
106         return total;
107     }
108
109     /** Get the number of days in a month */
110     public static int getNumberOfDaysInMonth(int year, int month) {
111         if (month == 1 || month == 3 || month == 5 || month == 7 ||
112             month == 8 || month == 10 || month == 12)
113             return 31;
114
115         if (month == 4 || month == 6 || month == 9 || month == 11)
116             return 30;
117
118         if (month == 2) return isLeapYear(year) ? 29 : 28;
119
120         return 0; // If month is incorrect
121     }
122
123     /** Determine if it is a leap year */
124     public static boolean isLeapYear(int year) {
125         return year % 400 == 0 || (year % 4 == 0 && year % 100 != 0);
126     }
127 }

```

该程序没有检测用户输入的有效性。例如：如果用户输入的月份不在1到12之间，或者年份在1800年之前，那么程序就会显示出错误的日历。为避免出现这样的错误，可以添加一个if语句在打印日历前检查输入。

该程序可以打印一个月的日历，还可以很容易地修改为打印整年的日历。尽管它现在只能处理1800年1月以后的月份，但是可以稍作修改，以便能够追溯到1800年之前的月份。

注意 方法抽象将程序模块化为整齐、层次明显的形式。将程序写成由简洁的方法构成的集合，比其他方式更容易编写、调试、维护和修改。这样的编写风格也会提高方法的可重用性。

提示 编写大型程序时，可以使用自顶向下或自底向上的方法。不要一次性地编写整个程序。使用这些方法似乎浪费了更多的开发时间（因为要反复编译和运行程序），但实际上，它会更节省时间并使调试更容易。

关键术语

actual parameter (实际参数即实参)

argument (实参)

ambiguous invocation (歧义调用)

divide and conquer (分治)

formal parameter (ie.parameter) (形式参数即形参)

information hiding (信息隐藏)

method (方法)

method abstraction (方法抽象)

method overloading (方法重载)

method signature (方法签名)

modifier (修饰符)

pass-by-value (按值传递)

parameter (参数)

return type (返回值类型)

return value (返回值)

scope of variable (变量的作用域)

stepwise refinement (逐步求精)

stub (待完善方法)

本章小结

- 程序模块化和可重用性是软件工程的中心目标之一。Java提供了很多有助于完成这一目标的有效结构。方法就是一个这样的结构。
- 方法头指定方法的修饰符、返回值类型、方法名和参数。本章所有的方法都使用静态修饰符static。
- 方法可以返回一个值。返回值类型returnValueType是方法要返回的值的数据类型。如果方法不返回值，则返回值类型就是关键字void。
- 参数列表是指方法中参数的类型、次序和数量。方法名和参数列表一起构成方法签名（method signature）。参数是可选的，也就是说，一个方法可以不包含参数。
- return语句也可以用在void方法中，用来终止方法并返回到方法的调用者。在方法中，用于偶尔改变正常流程控制是很有用的。
- 传递给方法的实际参数应该与方法签名中的形式参数具有相同的数目、类型和顺序。
- 当程序调用一个方法时，程序控制就转移到被调用的方法。当执行到该方法的return语句或到达方法结束的右括号时，被调用的方法将程序控制还给调用者。
- 在Java中，带返回值的方法也可以当作语句调用。在这种情况下，调用函数只要忽略返回值即可。
- 每次调用一个方法时，系统都会将参数和局部变量存储在一个称为堆栈（stack）的区域中。当一个方法调用另一个方法时，调用者的堆栈空间保持不动，开辟新的空间处理新方法的调用。一个方法完成它的工作之后返回到它的调用者时，就释放其相应的空间。
- 方法可以重载。这就意味着两个方法可以拥有相同的方法名，只要它们的方法参数列表不同即可。
- 在方法中声明的变量称作局部变量。局部变量的作用域是从声明它的地方开始，到包含这个变量的块结束为止。局部变量在使用前必须声明和初始化。
- 方法抽象是把方法的应用和实现分离。用户可以在不知道方法是如何实现的情况下使用方法。方法的实现细节封装在方法内，对调用该方法的用户隐藏。这就称为信息隐藏或封装。
- 方法抽象将程序模块化为整齐、层次分明的形式。将程序写成简洁的方法构成的集合，会比其他方式更容易编写、调试、维护和修改。这种编写风格也会提高方法的可重用性。
- 当实现一个大型程序时，可以使用自顶向下或自底向上的编码方法。不要一次性编写完整个程序。这种方式似乎浪费了更多的编码时间（因为要反复编译和运行这个程序），但实际上，它会更节省时间并使调试更容易。

复习题

5.2~5.4节

5.1 使用方法的优点有哪些？如何定义一个方法？如何调用一个方法？

5.2 main方法的返回类型是什么？

5.3 可以使用条件运算符简化程序清单5-1中的max方法吗？

5.4 下面的说法是否正确？对返回值类型为void的方法的调用总是单独的一条语句，但是对带返回值类型的方法的调用总是表达式的一部分。

5.5 如果在一个带返回值的方法中，不写return语句会发生什么错误？在返回值类型为void的方法中可以有return语句吗？下面方法中的return语句是否会导致语法错误？

```
public static void xMethod(double x, double y) {  
    System.out.println(x + y);  
    return x + y;  
}
```

5.6 给出术语形参、实参和方法签名的定义。

5.7 写出下列方法的方法头：

- 给定销售额和提成率, 计算销售提成。
- 给定月份和年份, 打印该月的日历。
- 计算平方根。
- 测试一个数是否是偶数, 如果是, 则返回true。
- 按指定次数打印一条信息。
- 给定贷款额、还款年数和年利率, 计算月支付额。
- 对于给定的小写字母, 给出相应的大写字母。

5.8 确定并更正下面程序中的错误:

```

1 public class Test {
2     public static method1(int n, m) {
3         n += m;
4         method2(3.4);
5     }
6
7     public static int method2(int n) {
8         if (n > 0) return 1;
9         else if (n == 0) return 0;
10        else if (n < 0) return -1;
11    }
12 }

```

5.9 根据2.16节提出的程序设计风格和文档指南, 使用花括号的次行风格重新编排下面的程序。

```

public class Test {
    public static double method1(double i, double j)
    {
        while (i < j) {
            j--;
        }

        return j;
    }
}

```

5.5~5.7节

5.10 实参是如何传递给方法的? 实参可以和形参同名吗?

5.11 什么是值传递? 给出下面程序的运行结果:

```

public class Test {
    public static void main(String[] args) {
        int max = 0;
        max(1, 2, max);
        System.out.println(max);
    }

    public static void max(
        int value1, int value2, int max) {
        if (value1 > value2)
            max = value1;
        else
            max = value2;
    }
}

```

a)

```

public class Test {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 6) {
            method1(i, 2);
            i++;
        }

        public static void method1(
            int i, int num) {
            for (int j = 1; j <= i; j++) {
                System.out.print(num + " ");
                num *= 2;
            }

            System.out.println();
        }
    }
}

```

b)

```

public class Test {
    public static void main(String[] args) {
        // Initialize times
        int times = 3;
        System.out.println("Before the call,"
            + " variable times is " + times);

        // Invoke nPrintln and display times
        nPrintln("Welcome to Java!", times);
        System.out.println("After the call,"
            + " variable times is " + times);
    }

    // Print the message n times
    public static void nPrintln(
        String message, int n) {
        while (n > 0) {
            System.out.println("n = " + n);
            System.out.println(message);
            n--;
        }
    }
}

```

c)

```

public class Test {
    public static void main(String[] args) {
        int i = 0;
        while (i <= 4) {
            method1(i);
            i++;
        }

        System.out.println("i is " + i);
    }

    public static void method1(int i) {
        do {
            if (i % 3 != 0)
                System.out.print(i + " ");
            i--;
        } while (i >= 1);

        System.out.println();
    }
}

```

d)

5.12 在前面问题的图a中，分别给出调用max方法之前、刚进入max方法、max方法刚要返回之前以及max方法返回之后堆栈的内容。

5.8节

5.13 什么是方法重载？可以定义两个同名但参数类型不同的方法吗？可以在一个类中定义两个名称和参数列表相同，但返回值类型不同或修饰符不同的方法吗？

5.14 下面的程序有什么错误？

```

public class Test {
    public static void method(int x) {
    }

    public static int method(int y) {
        return y;
    }
}

```

5.9节

5.15 指出并改正下面程序中的错误：

```

1 public class Test {
2     public static void main(String[] args) {
3         nPrintln("Welcome to Java!", 5);
4     }
5
6     public static void nPrintln(String message, int n) {
7         int n = 1;
8         for (int i = 0; i < n; i++)
9             System.out.println(message);
10    }
11 }

```

5.10节

5.16 下述说法是否正确？三角函数方法中的参数是以弧度为单位的角。

5.17 编写一个表达式，返回34到55之间的一个随机整数。编写一个表达式，返回0到999之间的一个随机整数。编写一个表达式，返回5.5到55.5之间的一个随机数。编写一个表达式，返回任意一个小写字母。

5.18 计算调用下列方法后的结果：

- (1) `Math.sqrt(4)`
- (2) `Math.sin(2*Math.PI)`
- (3) `Math.cos(2*Math.PI)`
- (4) `Math.pow(2,2)`
- (5) `Math.log(Math.E)`
- (6) `Math.exp(1)`
- (7) `Math.max(2,Math.min(3,4))`
- (8) `Math rint(-2.5)`
- (9) `Math.ceil(-2.5)`
- (10) `Math.floor(-2.5)`
- (11) `Math.round(-2.5F)`
- (12) `Math.round(-2.5)`
- (13) `Math.rint(2.5)`
- (14) `Math.ceil(2.5)`
- (15) `Math.floor(2.5)`
- (16) `Math.round(2.5F)`
- (17) `Math.round(2.5)`
- (18) `Math.round(Math.abs(-2.5))`

编程练习题

5.2~5.9节

5.1 (数学方面: 五角数) 一个五角数被定义为 $n(3n-1)/2$, 其中 $n=1, 2, \dots$ 。所以, 开始的几个数字就是1, 5, 12, 22, ..., 编写下面的方法返回一个五角数:

```
public static int getPentagonalNumber(int n)
```

编写一个测试程序显示前100个五角数, 每行显示10个。

*5.2 (求一个整数各位数字之和) 编写一个方法, 计算一个整数各位数字之和:

```
public static int sumDigits(long n)
```

例如: `sumDigits(234)` 返回`9(2+3+4)`。

提示 使用求余运算符`%`提取数字, 用除号`/`去掉提取出来的数字。例如: 使用`234%10 (=4)`抽取4。然后使用`234/10 (=23)`从234中去掉4。使用一个循环来反复提取和去掉每位数字, 直到所有的位数都提取完为止。编写程序提示用户输入一个整数, 然后显示这个整数所有数字的和。

**5.3 (回文整数) 编写下面两个方法:

```
// Return the reversal of an integer, i.e. reverse(456) returns 654
public static int reverse(int number)
```

```
// Return true if number is palindrome
public static boolean isPalindrome(int number)
```

使用`reverse`方法实现`isPalindrome`。如果一个数字的反向倒置数和它的顺向数一样, 这个数就称作回文数。编写一个测试程序, 提示用户输入一个整数值, 然后报告这个整数是否是回文数。

*5.4 (反向显示一个整数) 编写下面的方法, 反向显示一个整数:

```
public static void reverse(int number)
```


例如: `reverse(3456)` 返回6543。编写一个测试程序,提示用户输入一个整数,然后显示它的反向数。

*5.5 (对三个数排序) 编写下面的方法,按升序显示三个数:

```
public static void displaySortedNumbers(  
    double num1, double num2, double num3)
```

*5.6 (显示模式) 编写方法显示如下模式:

```
      1  
     2 1  
    3 2 1  
    ...  
n n-1 ... 3 2 1
```

方法头为:

```
public static void displayPattern(int n)
```

*5.7 (财务应用程序: 计算未来投资价值) 编写一个方法,计算按照给定的年数和利率计算未来投资值,未来投资是用练习题2.13中的公式计算得到的。

使用下面的方法头:

```
public static double futureInvestmentValue(  
    double investmentAmount, double monthlyInterestRate, int years)
```

例如: `futureInvestmentValue(10000,0.05/12,5)` 返回12833.59。

编写一个测试程序,提示用户输入投资额(例如1000)、利率(例如9%),然后打印年份从1到30年的未来值,如下所示:

```
The amount invested: 1000  
Annual interest rate: 9%  
Years      Future Value  
1          1093.80  
2          1196.41  
...  
29         13467.25  
30         14730.57
```

5.8 (摄氏度和华氏度之间的转换) 编写一个类,包含下面两个方法:

```
/** Converts from Celsius to Fahrenheit */  
public static double celsiusToFahrenheit(double celsius)  
  
/** Converts from Fahrenheit to Celsius */  
public static double fahrenheitToCelsius(double fahrenheit)
```

转换公式如下:

$$\text{华氏度} = (9.0 / 5) * \text{摄氏度} + 32$$

编写一个测试程序,调用这两个方法来显示如下表格:

摄氏度	华氏度	华氏度	摄氏度
40.0	104.0	120.0	48.89
39.0	102.2	110.0	43.33
...			
32.0	89.6	40.0	4.44
31.0	87.8	30.0	-1.11

5.9 (英尺和米之间的转换) 编写一个类,包含如下两个方法:

```
/** Converts from feet to meters */  
public static double footToMeter(double foot)  
  
/** Converts from meters to feet */  
public static double meterToFoot(double meter)
```

转换公式如下:



$$1\text{米} = 0.305 \times \text{英尺数}$$

编写一个测试程序，调用这两个方法以显示下面的表格：

英尺	米	米	英尺
1.0	0.305	20.0	65.574
2.0	0.61	25.0	81.967
...			
9.0	2.745	60.0	196.721
10.0	3.05	65.0	213.115

5.10 (使用isPrime方法) 程序清单5-7提供了测试某个数字是否是素数的方法isPrime(int number)。

使用这个方法求小于10000的素数个数。

5.11 (财务应用程序：计算佣金) 编写一个方法，利用练习4.39中的方案计算佣金。方法头如下所示：

```
public static double computeCommission(double salesAmount)
```

编写一个测试程序，显示下面表格：

销售总额	佣金
10000	900.0
15000	1500.0
...	
95000	11100.0
100000	11700.0

5.12 (显示字符) 使用下面的方法头，编写一个打印字符的方法：

```
public static void printChars(char ch1, char ch2, int
    numberPerLine)
```

该方法打印ch1到ch2之间的字符，每行按指定个数打印。编写一个测试程序，打印从'1'到'Z'的字符，每行打印10个。

*5.13 (数列求和) 编写一个方法计算下列级数：

$$m(i) = \frac{1}{2} + \frac{2}{3} + \cdots + \frac{i}{i+1}$$

编写一个测试程序显示下面的表格：

i	m(i)
1	0.5000
2	1.1667
...	
19	16.4023
20	17.3546

*5.14 (计算数列) 编写一个方法计算下面的数列：

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

编写一个测试程序显示下面的表格：

i	m(i)
10	3.04184
20	3.09162
...	
90	3.13048
100	3.13159

*5.15 (财务应用程序：打印税表) 程序清单3-6给出计算税款的程序。使用下面的方法头编写一个计算税款的方法：

```
public static double computetax(int status, double taxableIncome)
```

使用这个方法编写程序，打印可征税收入从50 000美元到60 000美元，收入间隔为50美元的所有四种纳税人的纳税表，如下所示：

Taxable Income	Single	Married Joint	Married Separate	Head of a House
50000	8688	6665	8688	7353
50050	8700	6673	8700	7365
...				
59950	11175	8158	11175	9840
60000	11188	8165	11188	9853

*5.16 (一年的天数) 使用下面的方法头编写一个方法, 返回一年的天数:

```
public static int numberOfDaysInAYear(int year)
```

编写一个测试程序, 显示从2000年到2010年每年的天数。

5.10~5.11节

*5.17 (显示0和1构成的矩阵) 编写一个方法, 使用下面的方法头显示 $n \times n$ 的矩阵:

```
public static void printMatrix(int n)
```

每个元素都是随机产生的0或1。编写一个测试程序, 打印如下所示的 3×3 矩阵:

```
0 1 0
0 0 0
1 1 1
```

5.18 (使用Math.sqrt方法) 使用Math类中的sqrt方法编写程序, 打印如下表格:

数字	平方根
0	0.0000
2	1.4142
...	
18	4.2426
20	4.4721

*5.19 (MyTriangle类) 创建一个名为MyTriangle的类, 它包含如下两个方法:

```
/** Returns true if the sum of any two sides is
 * greater than the third side. */
```

```
public static boolean isValid(
    double side1, double side2, double side3)
```

```
/** Returns the area of the triangle. */
```

```
public static double area(
    double side1, double side2, double side3)
```

编写一个测试程序, 读入三角形三边的值, 若输入有效, 则计算面积; 否则显示输入无效。三角形面积的计算公式在练习题2.21中给出。

5.20 (使用三角函数方法) 打印下面的表格, 显示 0° 到 360° 之间, 角度间隔为 10° 的正弦和余弦值, 保留小数点后4位。

角度	正弦值	余弦值
0	0.0000	1.0000
10	0.1736	0.9848
...		
350	-0.1736	0.9848
360	0.0000	1.0000

**5.21 (统计学方面: 计算均值和标准差) 在商业应用中, 经常需要计算数据的均值和标准差。均值就是这些数据的平均值。标准差是一个统计数据, 它会告诉你数据集之中的数据距离平均值的远近程度。例如: 一个班学生的平均年龄是多少? 这些年龄的集中程度如何? 如果所有学生的年龄相同, 则标准差为0。编写一个程序, 提示用户输入十个数字, 然后使用下面的公式显示它们的均值和标准差:

$$\text{均值} = \frac{\sum_{i=1}^n x_i}{n} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

$$\text{标准差} = \sqrt{\frac{\sum_{i=1}^n x_i^2 - \frac{\left(\sum_{i=1}^n x_i\right)^2}{n}}{n-1}}$$

下面是一个运行示例的结果：

```
Enter ten numbers: 1 2 3 4.5 5.6 6 7 8 9 10
The mean is 5.61
The standard deviation is 2.99794
```



****5.22** (数学方面：平方根的近似求法) 实现sqrt方法。数num的平方根可以通过对下面公式的反复计算近似地得到：

```
nextGuess = (lastGuess + (num / lastGuess)) / 2
```

当nextGuess和lastGuess几乎相同时，nextGuess就是平方根的近似值。

最初的猜测值可以是任意一个正值(例如1)。这个值就是lastGuess的初始值。如果nextGuess和lastGuess的差小于一个很小的数，比如0.0001，就可以认为nextGuess是num平方根的近似值；否则，nextGuess就成为lastGuess，近似过程持续执行。

5.10~5.11节

***5.23** (生成随机字符) 使用程序清单5-10中RandomCharacter里的方法，打印100个大写字母和100个一位数，每行打印10个。

****5.24** (显示当前日期和时间) 程序清单2-9显示当前时间。改进这个例子，显示当前的日期和时间。在程序清单5-12中的日历例子应该会给你提供一些如何求年、月和日的思路。

****5.25** (将毫秒数转换成小时数、分钟数和秒数) 使用下面的方法头，编写一个将毫秒数转换成小时数、分钟数和秒数的方法。

```
public static String convertMillis(long millis)
```

该方法返回形如“小时：分钟：秒”的字符串。例如：convertMillis(5500)返回字符串0:0:5，convertMillis(100000)返回字符串0:1:40，convertMillis(555550000)返回字符串154:19:10。

综合题

****5.26** (回文素数) 回文素数是指一个数同时为素数和回文数。例如：131是一个素数，同时也是一个回文素数。数字313和757也是如此。编写程序，显示前100个回文素数。每行显示10个数并且准确对齐，如下所示：

```
  2    3    5    7   11   101   131   151   181   191
313  353  373  383  727  757  787  797  919  929
...
```

****5.27** (反素数) 反素数(逆向拼写的素数)是指一个将其逆向之后也是一个素数的非回文素数。例如：17是一个素数，而71也是一个素数，所以17和71是反素数。编写程序，显示前100个素数。每行显示10个，并且准确对齐，如下所示：

```
 13  17  31  37  71  73  79  97 107 113
149 157 167 179 199 311 337 347 359 389
...
```

****5.28** (梅森素数) 如果一个素数可以写成 $2^p - 1$ 的形式，其中 p 是某个正整数，那么这个素数就称作梅森素数。编写程序，找出 $p \leq 31$ 的所有梅森素数，然后显示如下的输出结果：

```
p      2p - 1
2          3
3          7
5         31
...
```

****5.29** (游戏：掷骰子游戏) 掷骰子游戏是赌场中非常流行的骰子游戏。编写程序，玩这个游戏的另一种玩法，如下所示：

掷两个骰子。每个骰子有六个面，分别表示值1, 2, ..., 6。检查这两个骰子的和。如果和为2、

3或12 (称为掷骰子), 你就输了; 如果和是7或者11 (称作自然), 你就赢了; 但如果和是其他数字 (例如: 4、5、6、8、9或者10), 就确定了一个点。继续掷骰子, 直到掷出一个7或者掷出和刚才相同的点数。如果掷出的是7, 你就输了。如果掷出的点数和你前一次掷出的点数相同, 你就赢了。

程序扮演一个独立的玩家。下面是一些运行示例。

You rolled 5 + 6 = 11
You win



You rolled 1 + 2 = 3
You lose



You rolled 4 + 4 = 8
point is 8
You rolled 6 + 2 = 8
You win



You rolled 3 + 2 = 5
point is 5
You rolled 2 + 5 = 7
You lose



****5.30 (双素数)** 双素数是指一对差值为2的素数。例如: 3和5就是一对双素数, 5和7是一对双素数, 而11和13也是一对双素数。编写程序, 找出小于1000的所有双素数。显示结果如下所示:

(3, 5)
(5, 7)
...

****5.31 (财务应用程序: 信用卡号的合法性)** 信用卡号遵循下面的模式。一个信用卡号必须是13到16位的整数。它的开头必须是:

- 4, 指Visa卡
- 5, 指Master卡
- 37, 指American Express卡
- 6, 指Discover卡

在1954年, IBM的Hans Luhn提出一种算法, 该算法可以验证信用卡号的有效性。这个算法在确定输入的卡号是否正确, 或者这张信用卡是否被扫描仪正确扫描方面是非常有用的。遵循这个合法性检测, 可以生成所有的信用卡号, 通常称之为Luhn检测或者Mod 10检测, 可以如下描述 (为了方便解释, 假设卡号为4388576018402626):

(1) 从左到右对每个数字翻倍。如果对某个数字翻倍之后的结果是一个两位数, 那么就将这两位加在一起得到一位数。

$2 \times 2 = 4$	$6 \times 2 = 12 \ (1+2=3)$
$2 \times 2 = 4$	$5 \times 2 = 10 \ (1+0=1)$
$4 \times 2 = 8$	$8 \times 2 = 16 \ (1+6=7)$
$1 \times 2 = 2$	$4 \times 2 = 8$

(2) 现在将第一步得到的所有一位数相加。

$4+4+8+2+3+1+7+8=37$

(3) 将卡号里从左到右在奇数位上的所有数字相加。

$6+6+0+8+0+7+8+3=38$

(4) 将第二步和第三步得到的结果相加。

$$37+38=75$$

(5) 如果第四步得到的结果能被10整除, 那么卡号是合法的; 否则, 卡号是不合法的。例如, 号码4388576018402626是不合法的, 但是号码4388576018410707是合法的。

编写程序, 提示用户输入一个long型整数的信用卡号码, 显示这个数字是合法的还是非法的。使用下面的方法设计程序:

```
/** Return true if the card number is valid */
public static boolean isValid(long number)

/** Get the result from Step 2 */
public static int sumOfDoubleEvenPlace(long number)

/** Return this number if it is a single digit, otherwise, return
 * the sum of the two digits */
public static int getDigit(int number)

/** Return sum of odd place digits in number */
public static int sumOfOddPlace(long number)

/** Return true if the digit d is a prefix for number */
public static boolean prefixMatched(long number, int d)

/** Return the number of digits in d */
public static int getSize(long d)

/** Return the first k number of digits from number. If the
 * number of digits in number is less than k, return number. */
public static long getPrefix(long number, int k)
```

**5.32 (游戏: 赢取双骰子赌博游戏的机会) 修改练习题5.29使该程序运行10000次, 然后显示赢得游戏的次数。

***5.33 (当前日期和时间) 调用System.currentTimeMillis()返回从1970年1月1号0点开始的毫秒数。编写程序, 显示日期和时间。下面是运行示例:

Current date and time is May 16, 2009 10:34:23



**5.34 (打印日历) 练习题3.21使用Zeller一致性原理来计算某天是星期几。使用Zeller的算法简化程序清单5-12以获得每月开始的第一天是星期几。

5.35 (几何问题: 五边形的面积) 使用下面的公式计算五边形的面积:

$$\text{面积} = \frac{5 \times s^2}{4 \times \tan\left(\frac{\pi}{5}\right)}$$

编写程序, 提示用户输入五边形的边, 然后显示它的面积。

*5.36 (几何问题: 正多边形的面积) 正多边形是一个n条边的多边形, 它的每个边的长度都相等, 而且所有角的角度也相等(即多边形既是等边又等角的)。计算正多边形面积的公式是:

$$\text{面积} = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

使用下面的方法头编写方法, 返回正多边形的面积:

```
public static double area(int n, double side)
```

编写一个main方法, 提示用户输入边的个数以及正多边形的边长, 然后显示它的面积。