

第2章 |

Introduction to Java Programming, 8E

基本程序设计

学习目标

- 编写Java程序完成简单的计算 (2.2节)。
- 使用Scanner类从控制台获取输入 (2.3节)。
- 使用标识符命名变量、常量、方法和类 (2.4节)。
- 使用变量存储数据 (2.5~2.6节)。
- 用赋值语句和赋值表达式编写程序 (2.6节)。
- 使用常量存储永久数据 (2.7节)。
- 声明Java基本数据类型: `byte`、`short`、`int`、`long`、`float`、`double`和`char` (2.8.1节)。
- 使用Java运算符书写数学表达式 (2.8.2~2.8.3节)。
- 显示当前时间 (2.9节)。
- 使用简捷运算符 (2.10节)。
- 将一种类型的值强制转换为另一种类型 (2.11节)。
- 计算贷款支付额 (2.12节)。
- 使用`char`类型表示字符 (2.13节)。
- 计算整钱兑零 (2.14节)。
- 使用`String`类型表示字符串 (2.15节)。
- 熟悉Java的文档管理、程序设计风格和命名习惯 (2.16节)。
- 区分语法错误、运行时错误、逻辑错误和调试错误 (2.17节)。
- (GUI) 使用JOptionPane输入对话框获取输入 (2.18节)。

2.1 引言

在第1章里,学习了如何创建、编译和运行一个Java程序。现在,将学习如何编程解决实际问题。通过这些问题,你将学到如何利用基本数据类型、变量、常量、运算符、表达式以及输入/输出来进行基本的程序设计。

2.2 编写简单的程序

首先,我们来看一个计算圆面积的简单问题。该如何编写程序解决这个问题呢?

编写程序涉及如何设计算法以及如何将算法翻译成程序代码两部分内容。算法 (algorithm) 描述的是: 如果要解决问题,所需要执行的动作以及这些动作执行的顺序。算法可以帮助程序员在使用程序设计语言编写程序之前做一个规划。算法可以用自然语言或者伪代码 (即自然语言和程序设计代码混在一起使用) 描述。这个程序的算法描述如下:

- 1) 读入半径。
- 2) 利用下面的公式计算面积:

$$\text{面积} = \text{半径} \times \text{半径} \times \pi$$

- 3) 显示面积。

在学习程序设计引论课程时,遇到的绝大多数问题都可以用简单、直观的算法描述。

当你编写代码时，就是要把算法翻译成程序。你已经知道每个Java程序都是以一个类的声明开始，在声明里类名紧跟在关键字`class`后面。假设你选择`ComputeArea`作为这个类的类名。这个程序的概貌就如下所示：

```
public class ComputeArea {  
    // Details to be given later  
}
```

如你所知，每一个Java应用程序都必须有一个`main`方法，程序从该方法处开始执行。所以该程序被扩展为如下所示：

```
public class ComputeArea {  
    public static void main(String[] args) {  
        // Step 1: Read in radius  
  
        // Step 2: Compute area  
  
        // Step 3: Display the area  
    }  
}
```

这个程序需要读取用户从键盘输入的半径。这就产生了两个重要问题：

- 1) 读取半径。
- 2) 将半径存储在程序中。

我们先来解决第二个问题。为了存储半径，在程序中需要声明一个称作变量（variable）的符号。变量指定在程序中用于存储数据和计算结果的内存位置。每个变量都有自己的名字，可以用来访问它在内存的位置。

变量名应该尽量选择描述性的名字（descriptive name），而不是用`x`和`y`这样的名字：在这里的例子中，用`radius`表示半径、用`area`表示面积。为了让编译器知道`radius`和`area`是什么，需要指明它们的数据类型。Java提供简单数据类型来表示整数、浮点数（即带小数点的数）、字符以及布尔类型。这些类型称为原始数据类型（primitive data type）或基本类型（fundamental type）。

将`radius`和`area`声明为双精度型浮点数。程序可被扩展为如下所示：

```
public class ComputeArea {  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Step 1: Read in radius  
  
        // Step 2: Compute area  
  
        // Step 3: Display the area  
    }  
}
```

程序将`radius`和`area`声明为变量。保留字`double`表明`radius`和`area`是以双精度型浮点数形式存储在计算机中的。

第一步是读取半径`radius`。从键盘读取数字并不是一件简单的事情。在当前情况下，我们就先给程序中的`radius`赋一个固定值。

第二步是计算`area`，这是通过将表达式`radius*radius*3.14159`的值赋给`area`来实现的。

在最后一步里，使用`System.out.println`方法在控制台上显示`area`的值。

完整的程序如程序清单2-1所示。该程序的示例运行如图2-1所示。

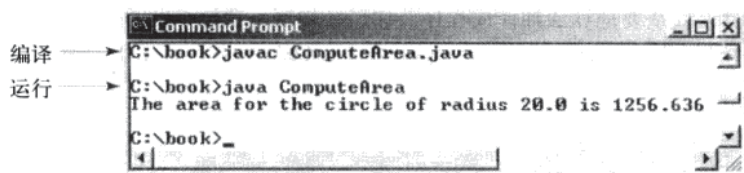


图2-1 程序显示圆面积

程序清单2-1 ComputeArea.java

```

1 public class ComputeArea {
2     public static void main(String[] args) {
3         double radius; // Declare radius
4         double area; // Declare area
5
6         // Assign a radius
7         radius = 20; // New value is radius
8
9         // Compute area
10        area = radius * radius * 3.14159;
11
12        // Display results
13        System.out.println("The area for the circle of radius " +
14            radius + " is " + area);
15    }
16 }

```

像radius和area这样的变量对应于它们在内存的位置。每个变量都有名字、类型、大小和值。第3行声明radius可以存储一个double型的数值。直到给它赋一个数值时，该变量才被定义。第7行将radius赋值为20。类似地，第4行声明变量area，第10行将10赋值给area。下面的表格显示的是随着程序的执行，area和radius在内存中的值。该表中的每一行显示的是程序中对应的每行语句执行之后变量的值。手动跟踪有助于理解一个程序是如何工作的，而且它也是一个查找程序错误的非常有用的工具。

行号	半径	面积
3	无值	
4		无值
7	20	
10		1256.636

加号(+)有两种意义：一种用途是做加法，另一种用途是做字符串的连接。第13~14行中的加号(+)称为字符串连接符(string concatenation operator)。如果两个操作数都是字符串，字符串连接符就把两个字符串连接起来。如果其中一个操作数非字符串(例如，一个数字)，这个非字符串值就会先被转换为一个字符串，然后再与另一个字符串相连。所以，第13~14行的加号(+)会将几个字符串连成一个更长的字符串，然后将它在输出结果中显示出来。关于字符串以及字符串连接的更多内容将在2.15节中讨论。

警告 在源代码中，字符串常量不能跨行。因此，下面的语句会造成编译错误：

```
System.out.println("Introduction to Java Programming,
    by Y. Daniel Liang");
```

为了改正错误，将该字符串分成几个单独的子串，然后再用连接符(+)将它们组合起来：

```
System.out.println("Introduction to Java Programming, " +
    "by Y. Daniel Liang");
```

提示 这个例子包括三个步骤。通过一次添加一个步骤，逐步开发和测试每一个步骤是一种很好的方法。

2.3 从控制台读取输入

在程序清单2-1中，源代码中的半径是固定的。为了能使用不同的半径，必须修改源代码然后重新编译它。很显然，这是非常不方便的。可以使用Scanner类从控制台输入。

Java使用System.out来表示标准输出设备，而用System.in来表示标准输入设备。默认情况下，输出设备是显示器，而输入设备是键盘。为了完成控制台输出，只需使用println方法就可以在控制台上显示基本值或字符串。Java并不直接支持控制台输入，但是可以使用Scanner类创建它的对象，以读取来自System.in的输入。如下所示：

```
Scanner input=new Scanner (System.in);
```

语法new Scanner (System.in)表明创建了一个Scanner类型的对象。语法Scanner input声明input是一个Scanner类型的变量。整行的Scanner input=new Scanner (System.in) 表明创建了一个Scanner对象，并且将它的引用值赋值给变量input。对象可以调用它自己的方法。调用对象的方法就是让这个对象完成某个任务。可以调用表2-1中的方法以读取各种不同类型的输入。


表2-1 Scanner对象的方法


方 法	描 述
nextByte()	读取一个byte类型的整数
nextShort()	读取一个short类型的整数
nextInt()	读取一个int类型的整数
nextLong()	读取一个long类型的整数
nextFloat()	读取一个float类型的数
nextDouble()	读取一个double类型的数
next()	读取一个字符串，该字符在一个空白符之前结束
nextLine()	读取一行文本（即以按下回车键为结束标志）

现在，我们将会看到如何通过调用nextDouble()方法来读取带小数点的数。其他的方法将在用到时再讨论。程序清单2-2改写了程序清单2-1，提示用户来输入一个半径值。

程序清单2-2 ComputeAreaWithConsoleInput.java

```
1 import java.util.Scanner; // Scanner is in the java.util package
2
3 public class ComputeAreaWithConsoleInput {
4     public static void main(String[] args) {
5         // Create a Scanner object
6         Scanner input = new Scanner(System.in);
7
8         // Prompt the user to enter a radius
9         System.out.print("Enter a number for radius: ");
10        double radius = input.nextDouble();
11
12        // Compute area
13        double area = radius * radius * 3.14159;
14
15        // Display result
16        System.out.println("The area for the circle of radius " +
17            radius + " is " + area);
18    }
19 }
```

Enter a number for radius: 2.5 
The area for the circle of radius 2.5 is 19.6349375

Enter a number for radius: 23 
The area for the circle of radius 23.0 is 1661.90111



Scanner类在包java.util里。它在第1行被导入。第6行创建一个Scanner对象。

第9行的语句显示一个消息，提示用户输入。

```
System.out.print("Enter a number for radius: ");
```

print方法和println方法很类似，两者的不同之处在于：当显示完字符串之后，println会将光标移到下一行，而print不会将光标移到下一行。

第10行的语句是从键盘读入一个输入。

```
double radius = input.nextDouble();
```

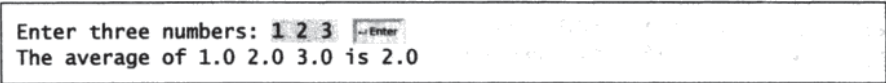
在用户键入一个数值然后点击回车键之后，该数值就被读入并赋值给radius。

更多关于对象的细节将在第8章中介绍。目前，只要知道这就是如何从控制台获取输入的方式就可以了。

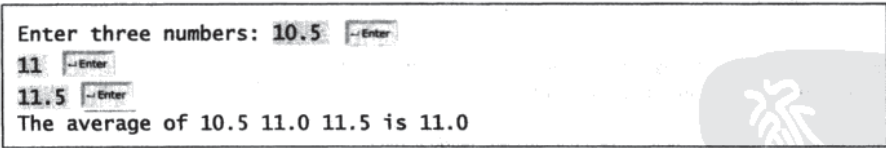
程序清单2-3给出从键盘读取输入的另一个例子。这个例子读取三个数值，然后显示它们的平均值。

程序清单2-3 ComputeAverage.java

```
1 import java.util.Scanner; // Scanner is in the java.util package
2
3 public class ComputeAverage {
4     public static void main(String[] args) {
5         // Create a Scanner object
6         Scanner input = new Scanner(System.in);
7
8         // Prompt the user to enter three numbers
9         System.out.print("Enter three numbers: ");
10        double number1 = input.nextDouble();
11        double number2 = input.nextDouble();
12        double number3 = input.nextDouble();
13
14        // Compute average
15        double average = (number1 + number2 + number3) / 3;
16
17        // Display result
18        System.out.println("The average of " + number1 + " " + number2
19            + " " + number3 + " is " + average);
20    }
21 }
```



Enter three numbers: 1 2 3
The average of 1.0 2.0 3.0 is 2.0

Enter three numbers: 10.5
11
11.5
The average of 10.5 11.0 11.5 is 11.0



导入Scanner类的代码（第1行）以及创建Scanner对象的代码（第6行）都是和前一个例子一样的，而且在你将编写的所有新程序中，这两行也都是同样的。

第9行提示用户输入三个数值。这些数值在第10~12行被读取。可以输入三个用空格符分隔开的数值，然后按回车键，或者每输入一个数值之后就按一次回车键，如该程序的示例运行所示。

2.4 标识符

正如在程序清单2-3中看到的，ComputeAverage、main、input、number1、number2、number3等都是出现在程序中事物的名字。这样的名字称为标识符（identifier）。所有的标识符必须遵从以下规则：

- 标识符是由字母、数字、下划线(_)和美元符号(\$)构成的字符序列。
- 标识符必须以字母、下划线(_)或美元符号(\$)开头, 不能以数字开头。
- 标识符不能是保留字。(参见附录A中的保留字列表)
- 标识符不能是true、false或null。
- 标识符可以为任意长度。

例如, \$2、ComputeArea、area、radius和showMessageDialog都是合法的标识符, 而2A和d+4都是非法的, 因为它们不符合标识符的命名规则。Java编译器会检测出非法标识符, 并且报语法错误。

注意 由于Java是区分大小写的, 所以area、Area和AREA是完全不同的标识符。

提示 标识符是为了命名变量、常量、方法、类和包。描述性的标识符可提高程序的可读性。

提示 不要用字符\$命名标识符。习惯上, 字符\$只用在机器自动产生的源代码中。

2.5 变量

正如在前几节的程序中看到的, 使用变量来存储将在程序中用到的数据。它们被称为变量是因为它们的值可能会被改变。在程序清单2-2中, radius和area都是双精度浮点型变量。可以将任意数值赋给radius和area, 这样, 就可以对它们重新赋值。例如, 可以使用如下所示的代码计算不同半径的面积:

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);
```

变量用于表示特定类型的数据。为了使用变量, 可以通过告诉编译器变量的名字及其他可以存储的数据类型来声明该变量。变量声明 (variable declaration) 告知编译器根据数据类型为变量分配合适的存储空间。声明变量的语法如下:

```
datatype variableName;
```

下面是一些变量声明的例子:

```
int count;           // Declare count to be an integer variable;
double radius;       // Declare radius to be a double variable;
double interestRate; // Declare interestRate to be a double variable;
```

这个例子中使用了数据类型int和double。后面还将介绍更多的数据类型, 例如, byte、short、long、float、char和boolean。

如果几个变量为同一类型, 允许一起声明它们, 如下所示:

```
datatype variable1, variable2, ..., variablen;
```

变量之间用逗号分隔开。例如:

```
int i, j, k; // Declare i, j, and k as int variables
```

注意 按照习惯, 变量名使用小写字母表示。如果一个名字由多个词组成, 那么将所有的词连接起来, 而且除了第一个词以外, 其他词的第一个字母都要大写, 例如radius和interestRate。

变量通常都有初始值。可以一步完成变量的声明和初始化。例如, 考虑下面的代码:

```
int count = 1;
```

它等同于下面的两条语句:

```
int count;
count = 1;
```

也可以使用简捷的方式来同时声明和初始化同一类型的变量。例如：

```
int i = 1, j = 2;
```

提示 在赋值给变量之前，必须声明变量。方法中声明的变量在使用之前必须被赋值。

任何时候，都要尽可能一步完成变量的声明和赋初值。这会使得程序易读，同时避免程序设计错误。

2.6 赋值语句和赋值表达式

声明变量之后，可以使用赋值语句（assignment statement）给它赋一个值。在Java中，将等号（=）作为赋值运算符（assignment operator）。赋值语句的语法如下所示：

```
variable = expression; (变量 = 表达式;)
```

表达式（expression）表示涉及值、变量和运算符的一个运算，它们组合在一起计算出一个新值。例如，考虑下面的代码：

```
int x = 1;           // Assign 1 to variable x
double radius = 1.0; // Assign 1.0 to variable radius
x = 5 * (3 / 2) + 3 * 2; // Assign the value of the expression to x
x = y + 1;           // Assign the addition of y and 1 to x
area = radius * radius * 3.14159; // Compute area
```

变量也可用在表达式中。例如：

```
x = x + 1;
```

在这个赋值语句中， $x+1$ 的结果赋值给 x 。假设在语句执行前 x 为1，那么语句执行后它就变成了2。

要给一个变量赋值，变量名必须在赋值运算符的左边。因此， $1=x$ 是错误的。

注意 在数学运算中， $x=2*x+1$ 表示一个等式。但是，在Java中， $x=2*x+1$ 是一个赋值语句，它计算表达式 $2*x+1$ ，并且将结果赋值给 x 。

在Java中，赋值语句本质上就是计算出一个值并将它赋给运算符左边变量的一个表达式。由于这个原因，赋值语句常常称作赋值表达式（assignment expression）。例如，下面的语句是正确的：

```
System.out.println(x = 1);
```

它等价于语句：

```
x = 1;
System.out.println(x);
```

下面的语句也是正确的：

```
i = j = k = 1;
```

该语句等价于：

```
k = 1;
j = k;
i = j;
```

注意 在赋值语句中，左边变量的数据类型必须与右边值的数据类型兼容。例如，`int x=1.0`是非法的，因为 x 的数据类型是整型`int`。在不使用类型转换的情况下，是不能把`double`值（1.0）赋给`int`变量的。类型转换将在2.11节介绍。

2.7 定名常量

一个变量的值在程序执行过程中可能会发生变化，但是定名常量（named constant）或简称常量则表示从不改变的永久数据。在`ComputeArea`程序中， π 是一个常量。如果频繁使用它，但又不想重复地输入3.14159，代替的方式就是声明一个常量 π 。下面就是声明常量的语法：

```
final datatype CONSTANTNAME = VALUE;
```

常量必须在同一条语句中声明和赋值。单词`final`是声明常量的Java关键字。例如，可以将 π 声明为

常量，然后将程序清单2-1改写如下：

```
// ComputeArea.java: Compute the area of a circle
public class ComputeArea {
    public static void main(String[] args) {
        final double PI = 3.14159; // Declare a constant

        // Assign a radius
        double radius = 20;

        // Compute area
        double area = radius * radius * PI;

        // Display results
        System.out.println("The area for the circle of radius " +
            radius + " is " + area);
    }
}
```

警告 按照习惯，常量用大写字母命名：用PI，而不是pi或Pi。

注意 使用常量有三个好处：1) 不必重复输入同一个值；2) 如果必须修改常量值（例如，将PI的值从3.14改为3.14159），只需在源代码中的一个地方做改动；3) 给常量赋一个描述性名字会提高程序易读性。

2.8 数值数据类型及其运算

每个数据类型都有它的取值范围。编译器会根据每个变量或常量的数据类型为其分配内存空间。Java为数值、字符值和布尔值数据提供了八种基本数据类型。本节介绍数值数据类型。

表2-2列出了六种数值数据类型、它们的范围以及所占存储空间。

表2-2 数值数据类型

类型名	范 围	存 储 大 小
byte	-2^7 (-128) \sim 2^7-1 (127)	8位带符号数
short	-2^{15} (-32 768) \sim $2^{15}-1$ (32 767)	16位带符号数
int	-2^{31} (-2 147 483 648) \sim $2^{31}-1$ (2 147 483 647)	32位带符号数
long	$-2^{63} \sim 2^{63}-1$ (即-9 223 372 036 854 775 808 \sim 9 223 372 036 854 775 807)	64位带符号数
float	负数范围：-3.4028235E+38 \sim -1.4E-45 正数范围：1.4E-45 \sim 3.4028235E+38)	32位，标准IEEE 754
double	负数范围：-1.7976931348623157E+308 \sim -4.9E-324 正数范围：4.9E-324 \sim 1.7976931348623157E+308	64位，标准IEEE 754

注意 IEEE 754是美国电气电子工程师协会通过的一个标准，用于在计算机上表示浮点数。该标准已被广泛采用。Java采用32位IEEE 754表示float型，64位IEEE 754表示double型。IEEE 754标准还定义了一些特殊值，这些值都在附录E中给出。

Java使用四种类型的整数：byte、short、int和long。为变量选择最适合的数据类型。例如：如果知道存储在变量中的整数是在字节范围内，就应该将该变量声明为byte型。为了简单和一致性，我们在本书的大部分内容中都使用int来表示整数。

Java使用两种类型的浮点数：float和double。double型是float型的两倍。所以，double型又称为双精度（double precision），而float称为单精度（single precision）。通常情况下，应该使用double型，因为它比float型更精确。

警告 当被赋值的变量的值太大（在大小方面）以至于无法存储时，就会造成上溢（overflow）。例如，执行以下语句就会造成上溢，因为可以存储在int类型变量中的最大值是2147483647。2147483648对int型而言就太大了。

```
int value = 2147483647 + 1; // value will actually be -2147483648
```

类似地，执行下面的语句也会造成上溢，因为可以存储在int类型变量中的最小值是-2147483648，而值-2147483649太大，不能存储在一个int变量中。

```
int value = -2147483648 - 1; // value will actually be 2147483647
```

Java不会报关于上溢的警告或错误。所以，当使用接近给定类型最大范围或最小范围的数时要小心。当浮点数太小（即特别接近0时）而不能被存储时，会造成下溢（underflow）。Java将它近似认为是0。所以通常无须考虑下溢问题。

2.8.1 数值运算符

数值数据类型的运算符包括标准的算术运算符：加号（+）、减号（-）、乘号（*）、除号（/）和求余号（%），如表2-3所示。

当除法的操作数都是整数时，除法的结果就是整数，小数部分被舍去。例如：5/2的结果是2而不是2.5，而-5/2的结果是-2而不是-2.5。为了实现通常意义的算术除法，其中一个操作数必须是浮点数。例如：5.0/2的结果是2.5。

运算符%可以求得除法的余数。左边的操作数是被除数，右边的操作数是除数。因此，7%3的结果是1，12%4的结果是0，26%8的结果是2，20%13的结果是7。

表2-3 算术运算符

名称	含义	示例	结果
+	加	34+1	35
-	减	34.0-0.1	33.9
*	乘	300*30	9000
/	除	1.0/2.0	0.5
%	求余	20%3	2

$$\begin{array}{r}
 2 \\
 3 \overline{) 7} \\
 \underline{6} \\
 1
 \end{array}
 \quad
 \begin{array}{r}
 3 \\
 4 \overline{) 12} \\
 \underline{12} \\
 0
 \end{array}
 \quad
 \begin{array}{r}
 3 \\
 8 \overline{) 26} \\
 \underline{24} \\
 2
 \end{array}
 \quad
 \begin{array}{r}
 \text{除数} \rightarrow 13 \overline{) 20} \\
 \underline{13} \\
 7 \leftarrow \text{余数}
 \end{array}$$

1 ← 商
 20 ← 被除数
 7 ← 余数

运算符%经常用在正整数上，实际上，它也可用于负整数和浮点值。只有当被除数是负数时，余数才是负的。例如：-7%3结果是-1，-12%4结果是0，-26%-8结果是-2，20%-13结果是7。

在程序设计中余数是非常有用的。例如：偶数%2的结果总是0而奇数%2的结果总是1。所以，可以利用这一特性来判定一个数是偶数还是奇数。如果今天是星期六，7天之后就又是星期六。假设你和你的朋友计划10天之后见面，那么10天之后是星期几呢？使用下面的表达式你就能够发现那天是星期二：

$$\begin{array}{c}
 \text{一周的第6天是星期六} \\
 \downarrow \\
 (6 + 10) \% 7 = 2 \\
 \uparrow \\
 \text{10天后}
 \end{array}$$

一周有7天
 一周的第2天是星期二
 注意：第0天是指星期天

程序清单2-4给出计算一个以秒为单位的时间量所包含的分钟数和剩余秒数的程序。例如，500秒就是8分钟20秒。

程序清单2-4 DisplayTime.java


```

1 import java.util.Scanner;
2
3 public class DisplayTime {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         // Prompt the user for input
    
```

```

7   System.out.print("Enter an integer for seconds: ");
8   int seconds = input.nextInt();
9
10  int minutes = seconds / 60; // Find minutes in seconds
11  int remainingSeconds = seconds % 60; // Seconds remaining
12  System.out.println(seconds + " seconds is " + minutes +
13    " minutes and " + remainingSeconds + " seconds");
14 }
15 }

```

Enter an integer for seconds: 500 
 500 seconds is 8 minutes and 20 seconds



line#	seconds	minutes	remainingSeconds
8	500		
10		8	
11			20



`nextInt()` 方法（第8行）读取整数 `seconds`。第4行使用 `seconds/60` 获取分钟数。第10行（`seconds%60`）获取在减去分钟数之后得到的剩余秒数。

运算符 `+` 和 `-` 可以是一元的也可以是二元的。一元操作符仅有一个操作数；而二元操作符有两个操作数。例如，在 `-5` 中，负号（`-`）可以认为是一元操作符，是对正数 `5` 取反，而在表达式 `4-5` 中，负号（`-`）是二元操作符，是从 `4` 中减去 `5`。

注意 涉及浮点数的计算都是近似的，因为这些数没有以准确的精度来存储。例如：

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
```

显示的是 `0.5000000000000001`，而不是 `0.5`，而

```
System.out.println(1.0 - 0.9);
```

显示的是 `0.09999999999999998`，而不是 `0.1`。

整数可以精确地存储。因此，整数计算得到的是精确的整数运算结果。

2.8.2 数值直接量

直接量（literal）是指在程序中直接出现的常量值。例如，下面的语句中 `34` 和 `0.305` 都是直接量：

```
int numberOfYears = 34;
double weight = 0.305;
```

1. 整型直接量

只要整型直接量与整型变量相匹配，就可以将整型直接量赋值给该整型变量。如果直接量太大，超出该变量的存储范围，就会出现编译错误。例如：语句 `byte b=128` 就会造成一个编译错误，因为 `byte` 型变量存放不下 `128`（注意：`byte` 型变量的范围是 `-128~127`）。

整型直接量默认为 `int` 型的，它的值在 -2^{31} （`-2 147 483 648`） $\sim 2^{31}-1$ （`2 147 483 647`）。为了表示一个 `long` 型的整型直接量，需要在其后追加字母 `L` 或 `l`（例如：`2147483648L`）。推荐使用 `L`，因为 `l`（`L` 的小写）很容易与 `1`（数字 `1`）混淆。为了在 Java 程序中写整数 `2147483648`，必须将它写成 `2147483648L`，因为 `2147483648` 超出了 `int` 型的范围。

注意 默认情况下，整型直接量是一个十进制整数。要表示一个八进制整数直接量，就用 `0`（零）开头，而要表示一个十六进制整数直接量，就用 `0x` 或 `0X`（零 `x`）开头。例如，下面的代码将十六进制数 `FFFF` 显示为十进制数 `65535`：

```
System.out.println(0xFFFF);
```

十六进制数、二进制数和八进制数都将在附录F中介绍。

2. 浮点型直接量

浮点型直接量带小数点，默认情况下是double型的。例如：5.0被认为是double型而不是float型。可以通过在数字后面加字母f或F表示该数为float型直接量，也可以在数字后面加d或D表示该数为double型直接量。例如：可以使用100.2f或100.2F表示float型值，用100.2d或100.2D表示double型值。

注意 double型值比float型值更精确。例如：

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

显示结果为：1.0/3.0 is 0.3333333333333333。

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

显示结果为：1.0F/3.0F is 0.33333334。

3. 科学记数法

浮点型直接量也可以用科学记数法表示，例如，1.23456e+2也可以写成1.23456e2，相当于 $1.23456 \times 10^2 = 123.456$ ，而1.23456e-2等于 $1.23456 \times 10^{-2} = 0.0123456$ 。E（或e）表示指数，既可以是写大的也可以是写小的。

注意 float型和double型都是用来表示带有小数点的数。为什么把它们称为浮点数呢？因为这些数都是以科学记数法的形式存储的。当一个像50.534的数被转换成科学记数法的形式时，它就是5.0534e+1，它的小数点就移到（即浮动到）一个新的位置。

2.8.3 计算Java表达式

用Java编写数值表达式就是使用Java运算符对算术表达式进行直接的翻译。例如，下述算术表达式：

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

可以翻译成如下所示的Java表达式：

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x +
9 * (4 / x + (9 + x) / y)
```

尽管Java有自己在后台计算表达式的方法，但是，Java表达式的结果和它对应的算术表达式的结果是一样的。因此，可以放心地将算术运算规则应用在计算Java表达式上。首先执行的是包括在圆括号里的运算。圆括号可以嵌套，嵌套时先计算内层括号。接下来，执行乘法、除法和求余运算。如果表达式中包含若干个乘法、除法和求余运算符，可按照从左到右的顺序执行。最后执行加法和减法运算。如果表达式中包含若干个加法和减法运算符，则按照从左到右的顺序执行。下面是一个如何计算表达式的例子：

$$\begin{array}{rcl}
 3 + 4 * 4 + 5 * (4 + 3) - 1 & & \\
 \uparrow & \text{————— (1) 圆括号里的先计算} & \\
 3 + 4 * 4 + 5 * 7 - 1 & & \\
 \uparrow & \text{————— (2) 乘法} & \\
 3 + 16 + 5 * 7 - 1 & & \\
 \uparrow & \text{————— (3) 乘法} & \\
 3 + 16 + 35 - 1 & & \\
 \uparrow & \text{————— (4) 加法} & \\
 19 + 35 - 1 & & \\
 \uparrow & \text{————— (5) 加法} & \\
 54 - 1 & & \\
 \uparrow & \text{————— (6) 减法} & \\
 53 & &
 \end{array}$$


程序清单2-5给出了利用公式 $celsius = \left(\frac{5}{9}\right)(fahrenheit - 32)$ 将华氏温度转换成摄氏温度的程序。

程序清单2-5 FahrenheitToCelsius.java

```

1 import java.util.Scanner;
2
3 public class FahrenheitToCelsius {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         System.out.print("Enter a degree in Fahrenheit: ");
8         double fahrenheit = input.nextDouble();
9
10        // Convert Fahrenheit to Celsius
11        double celsius = (5.0 / 9) * (fahrenheit - 32);
12        System.out.println("Fahrenheit " + fahrenheit + " is " +
13            celsius + " in Celsius");
14    }
15 }

```

Enter a degree in Fahrenheit: 100 
 Fahrenheit 100.0 is 37.777777777778 in Celsius



line#	fahrenheit	celsius
8	100	
11		37.777777777778



使用除法时要特别小心。在Java中，两个整数相除商为整数。在第11行，将 $\left(\frac{5}{9}\right)$ 转换为5.0/9而不是5/9，因为在Java中5/9的结果是0。

2.9 问题：显示当前时间

本节的问题是开发一个显示当前GMT（格林威治标准时间）的程序，格林威治时间的格式为小时：分钟：秒（hour:minute:second），例如13：19：8。

System类中的方法currentTimeMillis返回从GMT 1970年1月1日00:00:00开始到当前时刻的毫秒数，如图2-2所示。因为1970年是UNIX操作系统正式发布的时间，所以这一时间也称为UNIX时间戳（UNIX epoch）。

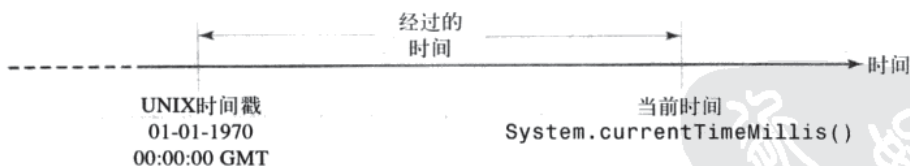


图2-2 System.currentTimeMillis()返回自UNIX时间戳以来的毫秒数

可以使用这个方法获取当前时间，然后按照如下步骤计算出当前的秒数、分钟数和小时数：

- 1) 调用System.currentTimeMillis()方法获取存放在变量totalMilliseconds中从1970年1月1日午夜到现在的毫秒数（例如：1203183086328毫秒）。
- 2) 通过将总毫秒数totalMilliseconds除以1000得到总秒数totalSeconds（例如：1203183086328毫秒/1000=1203183086秒）。
- 3) 通过totalSeconds%60得到当前的秒数（例如：1203183086秒%60=26，这个值就是当前秒数）。
- 4) 通过将totalSeconds除以60得到总的分钟数totalMinutes（例如：1203183086秒/60=20053051分钟）。
- 5) 通过totalMinutes%60得到当前分钟数（例如：20053051分钟%60=31，这个值就是当前分

30 • 第2章 基本程序设计

钟数)。

6) 通过将总分钟数totalMinutes除以60获得总的小时数totalHours (例如: 20053051分钟/60=334217小时)。

7) 通过totalHours%24得到当前的小时数 (例如: 334217小时%24=17, 该值就是当前小时数)。程序清单2-6给出完整的程序。

程序清单2-6 ShowCurrentTime.java

```

1 public class ShowCurrentTime {
2     public static void main(String[] args) {
3         // Obtain the total milliseconds since midnight, Jan 1, 1970
4         long totalMilliseconds = System.currentTimeMillis();
5
6         // Obtain the total seconds since midnight, Jan 1, 1970
7         long totalSeconds = totalMilliseconds / 1000;
8
9         // Compute the current second in the minute in the hour
10        long currentSecond = totalSeconds % 60;
11
12        // Obtain the total minutes
13        long totalMinutes = totalSeconds / 60;
14
15        // Compute the current minute in the hour
16        long currentMinute = totalMinutes % 60;
17
18        // Obtain the total hours
19        long totalHours = totalMinutes / 60;
20
21        // Compute the current hour
22        long currentHour = totalHours % 24;
23
24        // Display results
25        System.out.println("Current time is " + currentHour + ":"
26            + currentMinute + ":" + currentSecond + " GMT");
27    }
28 }

```

Current time is 17:31:26 GMT



line#	4	7	10	13	16	19	22
variables							
totalMilliseconds	1203183086328						
totalSeconds		1203183086					
currentSecond			26				
totalMinutes				20053051			
currentMinute					31		
totalHours						334217	
currentHour							17

当调用`System.currentTimeMillis()` (第4行) 时, 它返回GMT当前时间与1970年1月1日0点之间单位为毫秒的差值。这个方法返回一个`long`型的毫秒值。因此, 在该程序中的所有变量都被声明为`long`型。

2.10 简捷运算符

经常会出现变量的当前值被使用、修改, 然后再重新赋值给该变量的情况。例如, 下面的语句将就是给`i`的当前值加8, 再将结果值赋值给`i`:

```
i = i + 8;
```

Java允许使用简捷赋值运算符合并赋值符号和加号的功能。例如, 上面的语句可以改写成:

```
i += 8;
```

符号`+=`称为加法赋值运算符 (addition assignment operator)。其他简捷赋值运算符如表2-4中所示。

表2-4 简捷赋值运算符

运 算 符	名 称	举 例	等 价 于
<code>+=</code>	加法赋值运算符	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	减法赋值运算符	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	乘法赋值运算符	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	除法赋值运算符	<code>i /= 8</code>	<code>i = i / 8</code>
<code>%=</code>	求余赋值运算符	<code>i %= 8</code>	<code>i = i % 8</code>

警告 在简捷运算符中是没有空格的。例如: `+=`应该是`+=`。

注意 就像赋值运算符 (`=`) 一样, 运算符 (`+=`, `-=`, `*=`, `/=`, `%=`) 既可以构成赋值语句, 也可以构成赋值表达式。例如, 在下面的代码中, 第1行的`x+=2`是一个语句, 而在第2行中它就是一个表达式:

```
x += 2; // Statement
System.out.println(x += 2); // Expression
```

这里还有两个更简捷的赋值运算符, 它们是对变量进行自增1和自减1的操作。由于经常需要多次改变某个值, 所以这两个运算符使用起来很方便。这两个运算符分别是`++`和`--`。例如, 下面的代码是对`i`自增1, 而对`j`自减1:

```
int i = 3, j = 3;
i++; // i becomes 4
j--; // j becomes 2
```

运算符`++`和`--`都有前置和后置两种方式, 如表2-5所示。

表2-5 自增和自减运算符

运算符	名 称	说 明	举例 (假设 <code>i=1</code>)
<code>++var</code>	前置自增运算符	变量 <code>var</code> 的值加1且使用 <code>var</code> 增加后的新值	<code>int j=++i;</code>
<code>var++</code>	后置自增运算符	变量 <code>var</code> 的值加1但使用 <code>var</code> 原来的值	<code>int j=i++;</code>
<code>--var</code>	前置自减运算符	变量 <code>var</code> 的值减1且使用 <code>var</code> 减少后的新值	<code>int j=--i;</code>
<code>var--</code>	后置自减运算符	变量 <code>var</code> 的值减1但使用 <code>var</code> 原来的值	<code>int j=i--;</code>

若运算符是在变量的前面 (前置于变量), 则该变量自增1或自减1, 然后返回的是变量的新值。若运算符是在变量的后面 (后置于变量), 则变量也会自增1或自减1, 但是返回的是变量原来的旧值。因此, 前置的`++x`和`--x`分别称为前置自增运算符 (preincrement operator) 和前置自减运算符

(predecrement operator)，而后置 $x++$ 和 $x--$ 分别称为后置自增运算符 (postincrement operator) 和后置自减运算符 (postdecrement operator)。如果单独使用某种前置形式 $++$ (或 $--$) 和后置形式 $++$ (或 $--$)，那么不管前置还是后置其效果是一样的，但是用在表达式中它们就会产生不同的效果。下面的代码就解释了这一点：

```
int i = 10;
int newNum = 10 * i++;    等效于    int newNum = 10 * i;
                                i = i + 1;
```

在此例中，首先对 i 自增1，然后返回 i 的旧值来参与乘法运算。这样， $newNum$ 的值就成为100。如果如下所示将 $i++$ 换为 $++i$ ：

```
int i = 10;
int newNum = 10 * (++i);    等效于    i = i + 1;
                                int newNum = 10 * i;
```

i 自增1，然后返回 i 的新值，并参与乘法运算。这样， $newNum$ 的值就成为110。

下面是另一个例子：

```
double x = 1.0;
double y = 5.0;
double z = x-- + (++y);
```

在这三行程序执行完之后， y 的值为6.0， z 的值为7.0，而 x 的值为0.0。

自增运算符 $++$ 和自减运算符 $--$ 可以用于所有的整型和浮点型。这些运算符经常用在循环语句中。循环语句是控制一个操作或一系列操作连续执行多少次的结构体。这种结构以及循环语句相关的主题都将在第4章中介绍。

提示 使用自增运算符和自减运算符可以使表达式更加简短，但也会使它们比较复杂且难以读懂。应该避免在同一个表达式中使用这些运算符修改多个变量或多次修改同一个变量，如：

```
int k=++i + i.
```

2.11 数值类型转换

可以完成两个不同类型操作数的二元运算吗？当然可以。如果在一个二元运算中，其中一个操作数是整数，而另一个操作数是浮点数，Java会自动地将整数转换为浮点值。这样， $3*4.5$ 就成了 $3.0*4.5$ 。

总是可以将一个数值赋给支持更大数值范围类型的变量，例如，可以将 $long$ 型的值赋给 $float$ 型变量。但是，如果不进行类型转换，就不能将一个值赋给范围较小类型的变量。类型转换是一种将一种数据类型的数据类型的值转换成另一种数据类型的操作。将一个小范围类型的变量转换为大范围类型的变量称为拓宽类型 (widening a type)，把大范围类型的变量转换为小范围类型的变量称为缩窄类型 (narrowing a type)。拓宽类型不需要显式转换，可以自动执行转换。缩窄类型必须显式完成。

类型转换的语法要求目标类型放在括号内，紧跟其后的是要转换的变量名或值。例如，下面的语句：

```
System.out.println((int)1.7);
```

显式结果为1。当 $double$ 型值被转换为 int 型值时，小数部分被截去。

下面的语句：

```
System.out.println((double)1 / 2);
```

显式结果为0.5，因为1首先被转换为1.0，然后用2除1.0。但是，语句

```
System.out.println(1 / 2);
```

显式结果为0，因为1和2都是整数，那么对它们做除法的结果也必须是整数。

警告 如果要将一个值赋给一个范围较小类型的变量，例如：将 $double$ 型的值赋给 int 型变量，就必须进行类型转换。如果在这种情况下没有使用类型转换，就会出现编译错误。使用类型转

换时必须小心,丢失的信息也许会导致不精确的结果。

注意 类型转换不改变被转换的变量。例如,下面代码中的d在类型转换之后值不变:

```
double d = 4.5;
int i = (int)d; // i becomes 4, but d is not changed, still 4.5
```

注意 将一个int型变量赋值给short型或byte型变量,必须显式地使用类型转换。例如,下述语句就会有一个编译错误:

```
int i = 1;
byte b = i; // Error because explicit casting is required
```

然而,只要整型直接量是在目标变量允许的范围内,那么将整型直接量赋给short型或byte型变量时,就不需要显式的类型转换。请参考2.8.2节。

程序清单2-7给出保留营业税小数点后两位的程序。

程序清单2-7 SalesTax.java

```
1 import java.util.Scanner;
2
3 public class SalesTax {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         System.out.print("Enter purchase amount: ");
8         double purchaseAmount = input.nextDouble();
9
10        double tax = purchaseAmount * 0.06;
11        System.out.println("Sales tax is " + (int)(tax * 100) / 100.0);
12    }
13 }
```

Enter purchase amount: 197.55
Sales tax is 11.85



line#	purchaseAmount	tax	output
8	197.55		
10		11.853	
11			11.85



变量purchaseAmount是197.55 (第8行)。营业税是销售额的6%,所以,计算得到的税款tax为11.853 (第10行)。注意

```
tax * 100 is 1185.3
(int)(tax * 100) is 1185
(int)(tax * 100) / 100.0 is 11.85
```

因此,第11行的语句显示营业税是保留小数点后两位的11.85。

2.12 问题: 计算贷款支付额

本节的问题是编写程序计算贷款支付额。这里的贷款可以是购车贷款、学生贷款或者房屋抵押贷款。程序要求用户输入利率、年数以及贷款总额,并要求显示月支付金额和总偿还金额。

计算月支付额的公式如下:

$$\text{月支付额} = \frac{\text{贷款总额} \times \text{月利率}}{1 - \frac{1}{(1 + \text{月利率})^{\text{年数} \times 12}}}$$

不必知道这个公式是如何推导出来的。但是, 给定了月利率、年数和贷款总额, 就可以利用这个公式计算出月支付额。

在这个公式中, 必须计算 $(1 + \text{月利率})^{\text{年数} \times 12}$ 。可以使用Math类中的方法pow(a,b)来计算 a^b 。Java API中的Math类适用于所有的Java程序。例如:

```
System.out.println(Math.pow(2, 3)); // Display 8.0
System.out.println(Math.pow(4, 0.5)); // Display 2.0
```

使用Math.pow(1+monthlyInterestRate, numberOfYears*12)可以计算 $(1 + \text{月利率})^{\text{年数} \times 12}$ 。

下面是开发该程序的步骤:

- 1) 提示用户输入年利率、年数和贷款总额。
- 2) 利用年利率获取月利率。
- 3) 使用前面的公式计算月支付额。
- 4) 计算总支付额, 它是月支付额乘以12再乘以年数。
- 5) 显示月支付额和总支付额。

程序清单2-8给出完整的程序。

程序清单2-8 ComputeLoan.java

```
1 import java.util.Scanner;
2
3 public class ComputeLoan {
4     public static void main(String[] args) {
5         // Create a Scanner
6         Scanner input = new Scanner(System.in);
7
8         // Enter yearly interest rate
9         System.out.print("Enter yearly interest rate, for example 8.25: ");
10        double annualInterestRate = input.nextDouble();
11
12        // Obtain monthly interest rate
13        double monthlyInterestRate = annualInterestRate / 1200;
14
15        // Enter number of years
16        System.out.print(
17            "Enter number of years as an integer, for example 5: ");
18        int numberOfYears = input.nextInt();
19
20        // Enter loan amount
21        System.out.print("Enter loan amount, for example 120000.95: ");
22        double loanAmount = input.nextDouble();
23
24        // Calculate payment
25        double monthlyPayment = loanAmount * monthlyInterestRate / (1
26            - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
27        double totalPayment = monthlyPayment * numberOfYears * 12;
28
29        // Display results
30        System.out.println("The monthly payment is " +
31            (int)(monthlyPayment * 100) / 100.0);
32        System.out.println("The total payment is " +
33            (int)(totalPayment * 100) / 100.0);
34    }
35 }
```

```
Enter yearly interest rate, for example 8.25: 5.75 [Enter]
Enter number of years as an integer, for example 5: 15 [Enter]
Enter loan amount, for example 120000.95: 250000 [Enter]
The monthly payment is 2076.02
The total payment is 373684.53
```



	line#	10	13	18	22	25	27
variables							
annualInterestRate		5.75					
monthlyInterestRate			0.0047916666666				
numberOfYears				15			
loanAmount					250000		
monthlyPayment						2076.0252175	
totalPayment							373684.539

第10行读取年利率，在第13行将它转为月利率。如果输入的不是一个数值，就会出现运行错误。

需要为变量选择最合适的数据类型。例如，尽管可以将numberOfYears声明为long、float或double型，但是最好还是将其声明为int型（第18行）。注意，可能最适合于numberOfYears的类型是byte。但是为了简便起见，本书中的例子都是用int来表示整数，而用double来表示浮点值。

计算月支付额的公式被翻译成第25~27行的Java代码。

第31行和33行使用类型转换得到新的monthlyPayment和totalPayment，两个变量都保留到小数点后两位。

程序使用在第一行导入的Scanner类。程序还要用到Math类。为什么不导入Math类呢？这是因为Math类在java.lang包中。java.lang包中所有的类都被隐式导入，所以，是无须显式导入Math类的。

2.13 字符数据类型及运算

字符数据类型char用来表示单个字符。字符型直接量用单引号括住。考虑以下代码：

```
char letter = 'A';
char numChar = '4';
```

第一条语句将字符A赋值给char型变量letter。第二条语句将数字字符4赋值给char型变量numChar。

警告 字符串直接量必须括在双引号中。而字符直接量是括在单引号中的单个字符。因此“A”是一个字符串，而‘A’是一个字符。

2.13.1 统一码和ASCII码

计算机内部使用二进制数。一个字符在计算机中是以0和1构成的序列的形式来存储的。将字符映射为它的二进制形式的过程称为编码（encoding）。字符有多种不同的编码方式，编码表（encoding scheme）定义该如何编码每个字符。

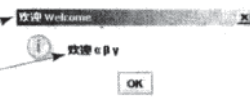
Java支持统一码（Unicode），统一码是由统一码协会（Unicode Consortium）建立的一种编码方案，它支持世界不同语言的可书写文本的交换、处理和显示。统一码一开始被设计为16比特的字符编码。基本数据类型char本身就能够利用这个设计思路来提供一种能够存放任意字符的简单数据类型。但是，这样的16比特的编码所能产生的字符只有65536个，它是不足以表示全世界所有字符的。因此，统一码标准被扩展为1112064个字符。这些字符都远远超过了原来16位的限制，它们称为补充字符（supplementary character）。Java支持这些补充字符。对补充字符的处理和表示都超过了本书的范围。为了简化问题，本书只考虑原来的16位统一码字符。这些字符都可以存储在一个char型变量中。

一个16比特位统一码占两个字节，用以\u开头的4位十六进制数表示，范围从‘\u0000’到‘\uFFFF’。例如：单词“welcome”被翻译成中文需要两个字符“欢迎”。这两个字符的统一码为“\u6B22\u8FCE”。

程序清单2-9给出显示两个中文字符和三个希腊字母的程序。

程序清单2-9 DisplayUnicode.java

```
1 import javax.swing.JOptionPane;
2
3 public class DisplayUnicode {
4     public static void main(String[] args) {
5         JOptionPane.showMessageDialog(null,
6             "\u06B2\u08FCE \u03b1 \u03b2 \u03b3",
7             "\u06B2\u08FCE Welcome",
8             JOptionPane.INFORMATION_MESSAGE);
9     }
10 }
```



如果系统中没有安装中文字体，是无法看到中文字符的。希腊字母的统一码是 \u03b1\u03b2\u03b3。

大多数计算机采用ASCII码（美国标准信息交换码），它是表示所有大小写字母、数字、标点符号和控制字符的7位编码表。Unicode码包括ASCII码，从 '\u0000' 到 '\u007F' 对应128个ASCII字符（参见附录B中的ASCII字符列表以及它们的十进制和十六进制编码）。在Java程序中，可以使用像 'X'、'1' 和 '\$' 等这样的ASCII字符，也可以使用统一码。例如，下面的语句是等价的：

```
char letter = 'A';
char letter = '\u0041'; // Character A's Unicode is 0041
```

两条语句都将字符A赋值给char型变量letter。

注意 自增和自减运算符也可用在char型变量上，这会得到该字符之前或之后的统一码字符。

例如，下面的语句显示字符b：

```
char ch = 'a';
System.out.println(++ch);
```

2.13.2 特殊字符的转义序列

假如你想在输出时打印如下带引号的信息，你能编写如下所示的这条语句吗？

```
System.out.println("He said \"Java is fun\"");
```

答案是不能，这条语句有语法错误。编译器会认为第二个引号字符就是这个字符串的结束标志，而不知道如何处理剩余的字符。

为了解决这个问题，Java定义了转义序列来表示特殊的字符，如表2-6所示。一个转义序列以反斜杠（\）开始，后面跟一个对编译器而言具有特殊意义的字符。

表2-6 Java转义字符序列

转义序列	名 称	Unicode
\b	退格键	\u0008
\t	Tab键	\u0009
\n	换行符号	\u000A
\f	进纸	\u000C
\r	回车键	\u000D
\\	反斜杠	\u005C
\'	单引号	\u0027
\"	双引号	\u0022

所以，现在可以使用下面的语句打印带引号的消息：

```
System.out.println("He said \"Java is fun\"");
```

它的输出才是：

```
He said "Java is fun"
```

2.13.3 字符型char数据与数值型数据之间的转换

char型数据可以转换成任意一种数值类型，反之亦然。将一个整数转换成一个char型数据时，只用到该数据的低十六位，其余部分都被忽略。例如：

```
char ch = (char)0xAB0041; // the lower 16 bits hex code 0041 is
                          // assigned to ch
System.out.println(ch); // ch is character A
```

要将一个浮点值转换成char型时，首先将浮点值转换成int型，然后就将这个整型值转换为char型。

```
char ch = (char)65.25; // decimal 65 is assigned to ch
System.out.println(ch); // ch is character A
```

当一个char型数据转换成数值型时，这个字符的统一码就被转换成某个特定的数值。

```
int i = (int)'A'; // the Unicode of character A is assigned to i
System.out.println(i); // i is 65
```

如果转换结果适用于目标变量，就可以使用隐式转换方式；否则，必须使用显式转换方式。例如，因为'a'的统一码是97，它是在一个字节的范围内，所以就可以使用隐式转换方式：

```
byte b = 'a';
int i = 'a';
```

但是，因为统一码\uFFFF4不适用于一个字节范围内，下面的转换就是不正确的：

```
byte b = '\uFFFF4';
```

为了强制赋值，就必须使用显式转换方式，如下所示：

```
byte b = (byte)\uFFFF4';
```

在0到FFFF之间的任何一个十六进制正整数都可以隐式地转换成字符型数据。而任何不在此范围内的其他数值都必须显式地转换为char型。

注意 所有数值运算符都可以用在char型操作数上。如果另一个操作数是一个数字或字符，那么char型操作数就会被自动转换成一个数字。如果另一个操作数是一个字符串，字符就会与该字符串相连。例如，下面的语句：

```
int i = '2' + '3'; // (int)'2' is 50 and (int)'3' is 51
System.out.println("i is " + i); // i is 101

int j = 2 + 'a'; // (int)'a' is 97
System.out.println("j is " + j); // j is 99
System.out.println(j + " is the Unicode for character "
    + (char)j);

System.out.println("Chapter " + '2');
```

显示结果

```
i is 101
j is 99
99 is the Unicode for character c
Chapter 2
```

注意 小写字母的统一码是从'a'的统一码开始，然后是'b'、'c'、...、'z'的统一码所构成的连续整数。大写字母的情况也是一样的。此外，'a'的统一码比'A'的统一码大，所以，'a'-'A'与'b'-'B'相等。因此，对应于小写字母ch的大写字母是(char)('A'+(ch-'a'))。

2.14 问题：整钱兑零

假如你希望开发一个程序，将给定的钱数分类成较小的货币单位。这个程序要求用户输入一个double型的值，该值是用美元和美分表示的总钱数，然后输出一个清单，列出和总钱数等价的dollar（1美元）、quarter（2角5分）、dime（1角）、nickel（5分）和penny（1分）的数目，如运行示例所示。

38 • 第2章 基本程序设计

该程序应该能够列出1美元的最大数, 其次是2角5分的最大数等, 依此类推。

下面是开发这个程序的步骤:

- 1) 提示用户输入十进制数作为总钱数, 例如11.56。
- 2) 将该钱数 (例如11.56) 转换为1分币的个数 (例如1156)。
- 3) 通过将1分币的个数除以100, 求出1美元的个数。通过对1分币的个数除以100求余数, 得到剩余1分币的个数。
- 4) 通过将剩余的1分币的个数除以25, 求出2角5分币的个数。通过对剩余的1分币的个数除以25求余数, 得到剩余1分币的个数。
- 5) 将剩余的1分币的个数除以10, 求出1角币的个数。通过对剩余的1分币的个数除以10求余数, 得到剩余1分币的个数。
- 6) 将剩余的1分币的个数除以5, 求出5分币的个数。通过对剩余的1分币的个数除以5求余数, 得到剩余1分币的个数。
- 7) 剩余1分币的个数即为所求。
- 8) 显示结果。

完整的程序如程序清单2-10所示。

程序清单2-10 ComputeChange.java

```

1 import java.util.Scanner;
2
3 public class ComputeChange {
4     public static void main(String[] args) {
5         // Create a Scanner
6         Scanner input = new Scanner(System.in);
7
8         // Receive the amount
9         System.out.print(
10             "Enter an amount in double, for example 11.56: ");
11         double amount = input.nextDouble();
12
13         int remainingAmount = (int)(amount * 100);
14
15         // Find the number of one dollars
16         int numberOfOneDollars = remainingAmount / 100;
17         remainingAmount = remainingAmount % 100;
18
19         // Find the number of quarters in the remaining amount
20         int numberOfQuarters = remainingAmount / 25;
21         remainingAmount = remainingAmount % 25;
22
23         // Find the number of dimes in the remaining amount
24         int numberOfDimes = remainingAmount / 10;
25         remainingAmount = remainingAmount % 10;
26
27         // Find the number of nickels in the remaining amount
28         int numberOfNickels = remainingAmount / 5;
29         remainingAmount = remainingAmount % 5;
30
31         // Find the number of pennies in the remaining amount
32         int numberOfPennies = remainingAmount;
33
34         // Display results
35         System.out.println("Your amount " + amount + " consists of \n" +
36             "\t" + numberOfOneDollars + " dollars\n" +
37             "\t" + numberOfQuarters + " quarters\n" +
38             "\t" + numberOfDimes + " dimes\n" +
39             "\t" + numberOfNickels + " nickels\n" +
40             "\t" + numberOfPennies + " pennies");
41     }

```

42 }

```

Enter an amount in double, for example 11.56: 11.56 --Enter
Your amount 11.56 consists of
    11 dollars
    2 quarters
    0 dimes
    1 nickels
    1 pennies

```



	line#	11	13	16	17	20	21	24	25	28	29	32
variables												
Amount		11.56										
remainingAmount			1156		56		6		6		1	
numberOfOneDollars				11								
numberOfQuarters						2						
numberOfDimes								0				
numberOfNickles										1		
numberOfPennies												1



变量amount存储的是从控制台上输入的钱数（第11行）。由于在程序结尾处显示结果时要用到该金额，所以该变量的值是不变的。程序引入变量remainingAmount（第13行）来存储变化的余额remainingAmount。

变量amount是一个double型的以美元和美分形式表示的十进制数。它被转换为一个int型变量remainingAmount以代表总的1美分的个数。例如：如果amount为11.56，那么remainingAmount的初始值为1156。除法运算得到除法的整数部分，所以1156/100的结果是11。求余运算可以得到除法的余数，所以1156%100的结果是56。

程序从总钱数中去掉了1美元币的最大数，得到的余额存储在变量remainingAmount中（第16~17行）。接着从remainingAmount中去掉2角5分币的最大数，得到一个新的余额remainingAmount（第20~21行）。继续同样的过程，程序就找到了余额所能包括的1角的最多个数、5分的最多个数和1分的最多个数。

本例的一个严重问题是将一个double型的总钱数转换为int型数remainingAmount时可能会损失精度，这会导致不精确的结果。如果输入的总额值为10.03，那么10.03*100就会变成1002.9999999999999，程序会显示10个1美元和2个1美分。为了解决这个问题，应该输入用美分表示的整型值（参见练习题2.9）。

结果如运行示例所示，显示0个1角、1个5美分和1个1美分。最好不要显示0个1角，也最好能用字的单数形式显示1个5美分和1个1美分。下一章将学习如何使用选择语句修改这个程序（参见练习题3.7）。

2.15 String类型

char类型只能表示一个字符。为了表示一串字符，使用称为String（字符串）的数据类型。例如，下述代码将消息声明为一个字符串，其值为“Welcome to Java”：

```
String message = "Welcome to Java";
```

String实际上与System类、JOptionPane类和Scanner类一样，都是一个Java库中预定义的类。String

类型不是基本类型，而是引用类型（reference type）。任何Java类都可以将变量表示为引用类型。引用数据类型将在第8章中详细讨论。目前，只需要知道如何声明一个String类型的变量，如何将字符串赋值给该变量以及如何连接字符串就可以了。

如前面的程序清单2-1中所示，可以进行两个字符串的连接。如果操作数之一是字符串，加号（+）就是连接运算符。如果操作数之一不是字符串（例如是一个数字），非字符串值先转换为字符串，再与另一个字符串连接起来。下面是几个例子：

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

如果操作数都不是字符串，那么加号（+）就是将两个数值加起来的加法运算符。

简捷运算符+=也可以用于字符串连接。例如，下面的代码将字符串“and Java is fun”追加到message中的字符串“Welcome to Java”之后：

```
message += " and Java is fun";
```

这样，新的message就成为“Welcome to Java and Java is fun”。

假设i=1且j=2，下列语句的输出结果是什么？


```
System.out.println("i + j is " + i + j);
```

由于“i + j is”首先与i的值连接，所以，输出结果是“i + j is 12”。要强制先执行i + j，就需要将i + j用括号括起来，如下所示：

```
System.out.println("i + j is " + (i + j));
```

为了从控制台读取字符串，调用Scanner对象上的next()方法。例如，下面的代码就可以从键盘读取三个字符串：


```
Scanner input = new Scanner(System.in);
System.out.println("Enter three strings: ");
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

```
Enter a string: Welcome to Java 
s1 is Welcome
s2 is to
s3 is Java
```

next()方法读取以空白字符结束的字符串（即' '、'\t'、'\f'、'\r'或'\n'）。

可以使用nextLine()方法读取一整行文本。nextLine()方法读取以按下回车键为结束标志的字符串。例如，下面的语句读取一行文本：

```
Scanner input = new Scanner(System.in);
System.out.println("Enter a string: ");
String s = input.nextLine();
System.out.println("The string entered is " + s);
```

```
Enter a string: Welcome to Java 
The string entered is "Welcome to Java"
```

重要警告 为了避免输入错误，不要在`nextByte()`、`nextShort()`、`nextInt()`、`nextLong()`、`nextFloat()`、`nextDouble()`和`next()`之后使用`nextLine()`，原因将在9.7.3节中解释。

2.16 程序设计风格和文档

编程风格 (programming style) 决定程序的外观。如果把整个程序写在一行，它也会被正确地编译和运行，但是这样是非常不好的程序设计风格，因为这会使程序的可读性很差。文档 (documentation) 是嵌在程序中的解释性评注和注释的一个结构体。程序设计风格与文档和编写代码的作用一样重要。良好的程序设计风格和适当的文档可以减少出错的几率，并且提高程序的可读性。到现在为止，已经学习了一些好的编程风格。本节把它们总结出来，并给出几条指导原则。关于Java程序设计风格和文档更详细的指南，可以在本书配套网站上的补充材料ID中找到。

2.16.1 适当的注释和注释风格

在程序的开头写一个摘要，解释一下这个程序是做什么的、其主要特点以及所用到的独特技术。在较大的程序中还要加上注释，介绍每一个主要步骤并解释每个难以读懂之处。注释写得简明扼要是很重要的，不能让整个程序都充满注释而使程序很难读懂。

除了行注释`//`和块注释`/*`之外，Java还支持一种称为Java文档注释 (javadoc comment) 的特殊注释形式。javadoc注释以`/**`开始，以`*/`结尾。它们能被JDK的javadoc命令提取出来，放入一个HTML文件。为了得到更多信息，参见java.sun.com/j2se/javadoc。

使用javadoc注释 (`/**...*/`) 来注释整个类或整个方法。为了将这些注释提取出来放在一个javadoc的HTML文件中，这些注释必须放在类或者方法头的前面。要注释方法中的某一步骤，建议使用行注释 (`//`)。

2.16.2 命名习惯

应该确保程序中为变量、常量、类和方法所选择的描述性名字是直观易懂的。名字是区分大小写的。下面列出变量、常量、方法和类的命名习惯。

- 1) 使用小写字母命名变量和方法。如果一个名字包含多个单词，就将它们连在一起，第一个单词的字母小写，而后面的每个单词的首字母大写，例如，变量`radius`和`area`以及方法`showInputDialog`。
- 2) 类名中的每个单词的首字母大写，例如，类名`ComputeArea`、`Math`和`JOptionPane`。
- 3) 大写常量中的所有字母，两个单词间用下划线连接，例如，常量`PI`和常量`MAX_VALUE`。

警告 对类命名时不要选择Java库中已经使用的名称。例如，因为Java已定义了`Math`类，就不要用`Math`来命名自己的类。

提示 应避免对标识符使用缩写。使用完整的单词更具描述性。例如，`numberOfStudents`比`numStuds`、`numOfStuds`或`numOfStudents`好。

2.16.3 适当的缩进和空白

保持一致的缩进风格会使程序更加清晰、易读、易于调试和维护。缩进 (indentation) 用于描述程序中组件或语句之间的结构性关系。即使将程序的所有语句都写在一行，对Java而言，读懂这样的程序也是没问题的，但是人们发现正确的对齐能够使我们更易读懂和维护代码。在嵌套结构中，每个内层的组件或语句应该比外层缩进两格，这会比仅仅只做嵌套好得多。

二元运算符的两边应该各加一个空格，如下面语句所示：


```
int i = 3+4 * 4; ← 不好的风格
```

```
int i = 3 + 4 * 4; ← 良好的风格
```

应该使用一个空行把代码分段，以使程序更易阅读。

2.16.4 块的风格

块是由大括号围起来的一组语句。块的写法有两种常用方式，次行（next-line）风格和行尾（end-of-line）风格，如下所示。

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Block Styles");
    }
}
```

次行风格

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

行尾风格

次行风格将括号垂直对齐，因而使程序容易阅读，而行尾风格更节省空间，并有助于避免犯一些细小的程序设计错误。这两种风格都是可以采纳的。选择哪一种完全依赖于个人或组织的喜好。应该持续采用一种风格，建议最好不要将这两种风格混合使用。本书与Java API源代码保持一致，都采用行尾风格。

2.17 程序设计错误

即使是非常有经验的程序员，也不能避免程序设计错误。错误可以分为三类：语法错误、运行错误和逻辑错误。

2.17.1 语法错误

在编译过程中出现的错误称为语法错误（syntax error）或编译错误（compile error）。语法错误是由代码结构体中的错误引起的，例如：拼错关键字，忽略了一些必要的标点符号，或者左花括号没有对应的右花括号。这些错误通常很容易检测到，因为编译器会告诉你这些错误在哪儿，以及是什么原因造成的。例如：编译下面的程序会出现语法错误，如图2-3所示。

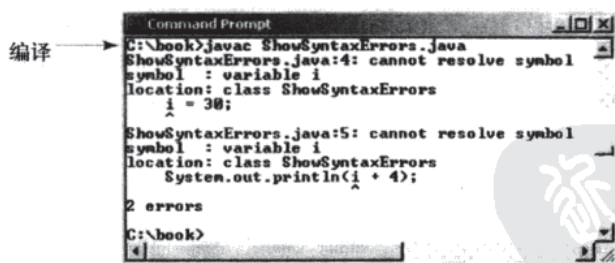


图2-3 编译器报告语法错误

```
1 // ShowSyntaxErrors.java: The program contains syntax errors
2 public class ShowSyntaxErrors {
3     public static void main(String[] args) {
4         i = 30;
5         System.out.println(i + 4);
6     }
7 }
```

检测出的错误有两个。它们都是因为没有声明变量*i*造成的。因为一个错误常常会显示很多行的编译错误。因此，从最上面的行开始向下调试是一个很好的习惯。解决程序前面出现的错误，可能就改正了程序中后面出现的其他错误。

2.17.2 运行错误

运行错误 (runtime error) 是引起程序非正常中断的错误。运行应用程序时, 当环境检测到一个不可能执行的操作时, 就会出现运行错误。输入错误就是典型的运行错误。

当用户输入一个程序不期望的输入, 导致程序不能处理该值时, 就会发生输入错误。例如: 如果程序希望读入的是一个数, 而用户输入的却是一个字符串, 就会导致程序出现数据类型错误。为了防止输入错误, 程序应该提醒用户输入正确类型的值。在从键盘读入整数之前, 最好能显示 “Please enter an integer” (请输入一个整数) 之类的提示信息。

另一个常见的运行错误是0作除数。当整数除法中除数为0时可能引发这种情况。例如: 下面的程序将会导致运行错误, 如图2-4所示。

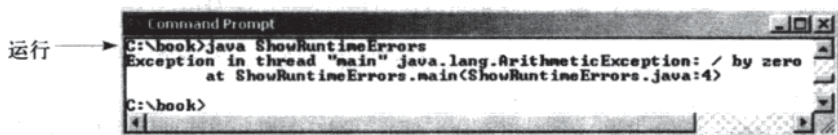


图2-4 运行错误导致程序非正常中断

```
1 // ShowRuntimeErrors.java: Program contains runtime errors
2 public class ShowRuntimeErrors {
3     public static void main(String[] args) {
4         int i = 1 / 0;
5     }
6 }
```

2.17.3 逻辑错误

当程序没有按预期的方式执行时就会发生逻辑错误 (logic error)。这种错误发生的原因多种多样。例如: 假设你编写下面的程序将number1和number2相加。

```
// ShowLogicErrors.java: The program contains a logic error
public class ShowLogicErrors {
    public static void main(String[] args) {
        // Add number1 to number2
        int number1 = 3;
        int number2 = 3;
        number2 += number1 + number2;
        System.out.println("number2 is " + number2);
    }
}
```

该程序既没有语法错误也没有运行错误, 但是它不能打印出number2的正确结果。看看你能否找到错误。

2.17.4 调试

通常情况下, 因为编译器可以明确指出错误的位置以及出错的原因, 所以语法错误是很容易发现和纠正的。运行错误也不难找, 因为在程序异常中止时, 错误的原因和位置都会显示在控制台上。然而, 查找逻辑错误就很富有挑战性。

逻辑错误也称为小虫子 (bug)。查找和改正错误的过程称为调试 (debugging)。调试的一般途径是采用各种方法逐步缩小程序中bug所在的范围。可以手工跟踪 (hand trace) 程序 (即通过读程序找错误), 也可以插入打印语句, 显示变量的值或程序的执行流程。这种方法适用于短小、简单的程序。对于庞大、复杂的程序, 最有效的调试方法还是使用调试工具。

教学注意 IDE不仅有助于调试错误, 而且也是一个有效的教学工具。补充材料II给出如何使用调试器来追踪程序, 以及调试过程是如何帮助你有效学习Java的。

2.18 (GUI) 从输入对话框获取输入

可以从控制台获取输入。还有一种可以选择的方法，那就是通过调用 `JOptionPane.showInputDialog` 方法从一个输入对话框中获取输入，如图2-5所示。

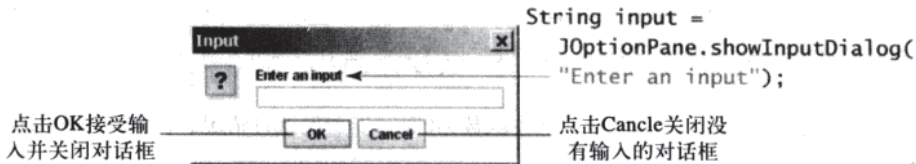


图2-5 输入对话框可以让用户输入一个字符串

当执行这个方法时，就会显示一个对话框，让你键入输入值。输入一个字符串后，单击OK接收输入并关闭该对话框。从该方法中返回的输入是一个字符串。

使用 `showInputDialog` 方法的途径有很多种。目前，你只需要知道两种调用方式即可。

一种是使用像下面例子中的语句：

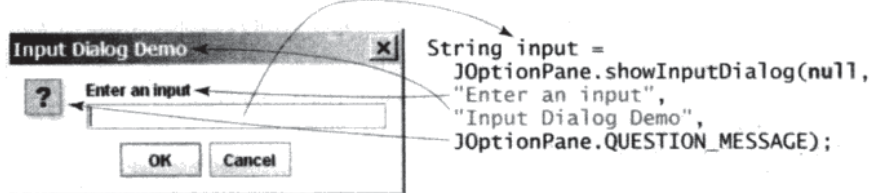
```
JOptionPane.showInputDialog(x);
```

其中 `x` 是表示提示信息的字符串。

而另一种是使用如下所示的语句：

```
String string = JOptionPane.showInputDialog(null, x,
    y, JOptionPane.QUESTION_MESSAGE);
```

其中 `x` 是表示提示信息的字符串，`y` 是表示输入对话框标题的字符串，如下图所示。



2.18.1 将字符串转换为数字

输入对话框返回的输入是一个字符串。如果你输入的是一个数字值123，它会返回"123"。必须把字符串转化为数字值以得到数字型的输入。

要把一个字符串转换为一个 `int` 型值，使用 `Integer` 类中的 `parseInt` 方法，如下所示：

```
int intValue = Integer.parseInt(intString);
```

这里的 `intString` 是一个数值字符串，例如："123"。

要将一个字符串转换为一个 `double` 型的值，使用 `Double` 类中的 `parseDouble` 方法，如下所示：

```
double doubleValue = Double.parseDouble(doubleString);
```

这里的 `doubleString` 是一个数值字符串，例如："123.45"。

`Integer` 类和 `Double` 类都包含在包 `java.lang` 中，因此，它们都是自动导入的。

2.18.2 使用输入对话框

程序清单2-8是从控制台读取输入。还可以使用输入对话框。

程序清单2-11给出完整的程序。图2-6给出程序的示例运行。

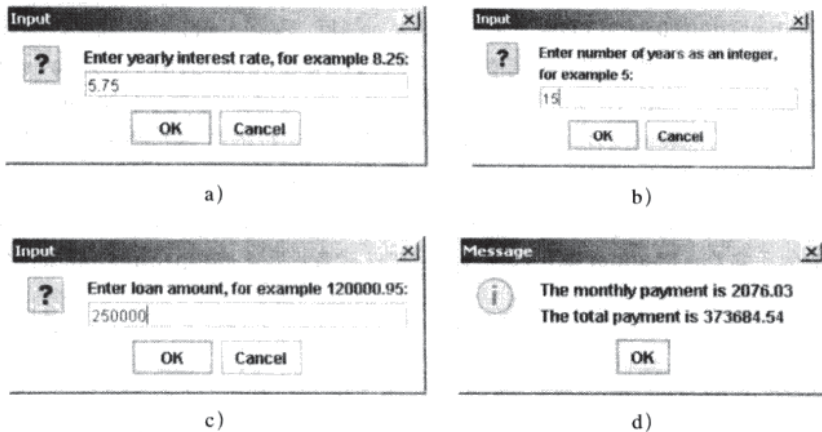


图2-6 程序接受年利率a、年数b、贷款总额c，然后显示月支付额和总支付额d

程序清单2-11 ComputeLoanUsingInputDialog.java

```

1 import javax.swing.JOptionPane;
2
3 public class ComputeLoanUsingInputDialog {
4     public static void main(String[] args) {
5         // Enter yearly interest rate
6         String annualInterestRateString = JOptionPane.showInputDialog(
7             "Enter yearly interest rate, for example 8.25:");
8
9         // Convert string to double
10        double annualInterestRate =
11            Double.parseDouble(annualInterestRateString);
12
13        // Obtain monthly interest rate
14        double monthlyInterestRate = annualInterestRate / 1200;
15
16        // Enter number of years
17        String numberOfYearsString = JOptionPane.showInputDialog(
18            "Enter number of years as an integer, \nfor example 5:");
19
20        // Convert string to int
21        int numberOfYears = Integer.parseInt(numberOfYearsString);
22
23        // Enter loan amount
24        String loanString = JOptionPane.showInputDialog(
25            "Enter loan amount, for example 120000.95:");
26
27        // Convert string to double
28        double loanAmount = Double.parseDouble(loanString);
29
30        // Calculate payment
31        double monthlyPayment = loanAmount * monthlyInterestRate / (1
32            - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
33        double totalPayment = monthlyPayment * numberOfYears * 12;
34
35        // Format to keep two digits after the decimal point
36        monthlyPayment = (int)(monthlyPayment * 100) / 100.0;
37        totalPayment = (int)(totalPayment * 100) / 100.0;
38
39        // Display results
40        String output = "The monthly payment is " + monthlyPayment +
41            "\n\nThe total payment is " + totalPayment;
42        JOptionPane.showMessageDialog(null, output);
43    }
44 }

```


第6~7行的showInputDialog方法显示一个输入对话框。输入double值的利率，然后点击OK接受这个输入。返回值是一个字符串，该值被赋值给一个String变量annualInterestRateString。使用Double.parseDouble(annualInterestRateString)（第11行）将字符型转换为double型值。如果在输入对话框中输入的不是数值或者点击了Cancel，都会出现运行错误。在第13章中，将会学习如何处理这些异常，以使程序能够继续运行。

教学注意 可以使用JOptionPane或Scanner来获取输入，这是很方便的。为保持一致性，本书中的很多例子都是用Scanner来获取输入的。可以很容易地使用JOptionPane来改写这些例子以获取输入。

关键术语

algorithm (算法)	incremental development and testing (递进式开发和测试)
assignment operator (=) (赋值运算符)	indentation (缩进)
assignment statement (赋值语句)	int type (整数类型)
backslash (\) (反斜杠)	literal (直接量)
byte type (字节类型)	logic error (逻辑错误)
casting (类型转换)	long type (长整型类型)
char type (字符类型)	narrowing (of types) ((类型的) 缩窄)
constant (常量)	operator (操作符)
data type (数据类型)	overflow (上溢)
debugger (调试器)	pseudocode (伪代码)
debugging (调试)	primitive data type (基本数据类型)
declaration (声明)	runtime error (运行错误)
decrement operator (--) (自减运算符)	short type (短整型类型)
double type (双精度类型)	syntax error (语法错误)
encoding (编码)	supplementary Unicode (补充统一码)
final (final关键字)	underflow (下溢)
float type (浮点类型)	Unicode (统一码)
floating-point number (浮点数)	UNIX epoch (UNIX时间戳)
expression (表达式)	variable (变量)
identifier (标识符)	widening (of types) ((类型的) 拓宽)
increment operator (++) (自增运算符)	whitespace (空白符)

本章小结

- 标识符是程序中的事物的名称。
- 标识符是由字母、数字、下划线(_)和美元符号(\$)构成的字符序列。
- 标识符必须以字母或下划线(_)开头，不能以数字开头。
- 标识符不能是保留字。
- 标识符可以为任意长度。
- 选择描述性的标识符可提高程序的可读性。
- 使用变量存储在程序中使用的数据。
- 声明变量就是告诉编译器何种数据类型可以存储变量。
- 按照习惯，变量名是小写字母。

- 在Java中，等号(=)被用作赋值运算符。
- 方法中声明的变量必须在使用前被赋值。
- 定名常量(或简称为常量)表示从不改变的永久数据。
- 用关键字final声明定名常量。
- 按照习惯，常量用大写字母命名。
- Java提供四种整数类型(byte、short、int、long)表示四种不同长度范围的整数。
- Java提供两种浮点类型(float、double)表示两种不同精度的浮点数。
- Java提供运算符完成数值运算：加号(+)、减号(-)、乘号(*)、除号(/)和求余符号(%)。
- 整数运算(/)得到的结果是一个整数。
- Java表达式中的数值运算符和算术表达式中应用的数值运算符的使用方法是完全一致的。
- Java提供简捷运算符：+= (加法赋值)、-= (减法赋值)、*= (乘法赋值)、/= (除法赋值)以及%= (求余赋值)。
- 自增运算符(++)和自减运算符(--)分别对变量加1或减1。
- 当计算的表达式中有不同类型的值，Java会自动地将操作数转换为正确类型。
- 可以使用(type)exp这样的表示法显式地将数值从一个类型转换到另一个类型。
- 将一个小范围类型的变量转换为大范围类型的过程称为拓宽类型。
- 将一个大范围类型的变量转换为小范围类型的过程称为缩窄类型。
- 拓宽类型不需要显式转换，可以自动完成。缩窄类型必须显式完成。
- 字符类型(char)表示单个字符。
- 字符\称为转义字符。
- Java允许使用转义序列来表示特殊字符，例如：'\t'和'\n'。
- 字符' '、'\t'、'\f'、'\r'和'\n'都称为空白字符。
- 在计算机科学中，1970年1月1日午夜零点称为UNIX时间戳。
- 程序设计错误可归纳为三类：语法错误、运行错误和逻辑错误。
- 编译过程中出现的错误为语法错误或编译错误。
- 运行错误是指引起程序非正常中断的错误。
- 逻辑错误是指程序没有按预期的方式完成。

复习题

2.2~2.7节

2.1 下列哪些标识符是合法的？哪些是Java的关键字？

**applet, Applet, a++, --a, 4#R, \$4, #44, apps
class, public, int, x, y, radius**

2.2 将下面的算法翻译成Java代码：

- 第一步：声明一个名为miles初始值为100的double型变量。
- 第二步：声明一个名为KILOMETERS_PER_MILE值为1.609的double型常量。
- 第三步：声明一个名为kilometers的double型变量，将miles和KILOMETERS_PER_MILE相乘的结果赋值给kilometers。
- 第四步：将kilometers显示在控制台上。

在第四步之后，kilometers的值是多少？

2.3 使用常量的好处是什么？声明一个值为20的int型常量SIZE。

2.8~2.10节

2.4 假设int a=1和double d=1.0，并且每个表达式都是独立的，那么下面表达式的结果是什么？

48 • 第2章 基本程序设计

```

a = 46 / 9;
a = 46 % 9 + 4 * 4 - 2;
a = 45 + 43 % 5 * (23 * 3 % 2);
a %= 3 / a + 3;
d = 4 + d * d + 4;
d += 1.5 * 3 + (++a);
d -= 1.5 * 3 + a++;

```

2.5 给出下面求余运算的结果。

```

56 % 6
78 % -4
-34 % 5
-34 % -5
5 % 1
1 % 5

```

2.6 如果今天是星期二，那么100天后是星期几？

2.7 分别找出byte、short、int、long、float和double中的最大数和最小数，哪种数据类型所需的存储空间最小？

2.8 25/4的结果是什么？如果希望得到的结果是浮点数，应该怎样改写这个表达式？

2.9 下列语句正确吗？如果正确，写出其输出值。

```

System.out.println("25 / 4 is " + 25 / 4);
System.out.println("25 / 4.0 is " + 25 / 4.0);
System.out.println("3 * 2 / 4 is " + 3 * 2 / 4);
System.out.println("3.0 * 2 / 4 is " + 3.0 * 2 / 4);

```

2.10 如何用Java书写下面的算术表达式？

$$\frac{4}{3(r+34)} - 9(a+bc) + \frac{3+d(2+a)}{a+bd}$$

2.11 假设m和r都是整数。编写mr²的Java表达式，得到一个浮点数。

2.12 下列说法哪些是正确的？

- (1) 任何表达式都可以用作语句。
- (2) 表达式x++可以用作语句。
- (3) 语句x=x+5也是一个表达式。
- (4) 语句x=y=x=0是非法的。

2.13 下列哪些是正确的浮点数直接量？

12.3, 12.3e+2, 23.4e-2, -334.4, 20, 39F, 40D

2.14 找出并修改下列代码中的错误：

```

1 public class Test {
2     public void main(string[] args) {
3         int i;
4         int k = 100.0;
5         int j = i + 1;
6
7         System.out.println("j is " + j + " and
8             k is " + k);
9     }
10 }

```

2.15 如何使用System.currentTimeMillis()方法获取当前分钟数？

2.11节

2.16 不同类型的数值能在一起计算吗？

2.17 如果显式地将double型转换为int型，那么对double型值的小数部分是如何处理的？类型转换是否改变被转换变量的值？

2.18 写出下面语句段的输出：



```
float f = 12.5F;
int i = (int)f;
System.out.println("f is " + f);
System.out.println("i is " + i);
```

2.13节

2.19 使用打印语句求出 '1'、'A'、'B'、'a'、'b' 的ASCII码。使用打印语句求出ASCII码为十进制数 40、59、79、85、90 的字符。使用打印语句求出ASCII码为十六进制数 40、5A、71、72、7A 的字符。

2.20 下列哪些是正确的字符直接量？

'1', '\u345dE', '\u3fFa', '\b', \t

2.21 如何显示字符 \ 和 "？

2.22 执行下述代码：

```
int i = '1';
int j = '1' + '2';
int k = 'a';
char c = 90;
```

2.23 下面哪些类型转换是允许的？如果允许，写出转换后的结果。

```
char c = 'A';
i = (int)c;
float f = 1000.34f;
int i = (int)f;
```

```
double d = 1000.34;
int i = (int)d;
```

```
int i = 97;
char c = (char)i;
```

2.24 给出下面程序的输出结果：

```
public class Test {
    public static void main(String[] args) {
        char x = 'a';
        char y = 'c';

        System.out.println(++x);
        System.out.println(y++);
        System.out.println(x - y);
    }
}
```

2.15节

2.25 给出下面语句的输出结果（编写程序验证你的结果）：

```
System.out.println("1" + 1);
System.out.println('1' + 1);
System.out.println("1" + 1 + 1);
System.out.println("1" + (1 + 1));
System.out.println('1' + 1 + 1);
```

2.26 计算下面表达式的结果（编写程序验证你的结果）：

```
1 + "Welcome " + 1 + 1
1 + "Welcome " + (1 + 1)
1 + "Welcome " + ('\u0001' + 1)
1 + "Welcome " + 'a' + 1
```

2.16~2.17节

2.27 类名、方法名、常量和变量的命名习惯是什么？根据Java的命名习惯，下面哪些是常量、方法、变量或类？



MAX_VALUE、Test、read、readInt

2.28 根据编程风格和文档指南，使用次行花括号方式，重新布局下列程序的格式。

```
public class Test
{
    // Main method
    public static void main(String[] args) {
        /** Print a line */
        System.out.println("2 % 3 = "+2%3);
    }
}
```

2.29 描述何谓语法错误、运行错误和逻辑错误。

2.18节

2.30 为什么必须导入JOptionPane类而无须导入Math类？

2.31 如何使用对话框提示用户输入一个整数？

2.32 如何将字符串转换为整数？如何将字符串转换为double型？

编程练习题

注意 学生可以从www.cs.armstrong.edu/liang/intro8e/exercise8e.zip上下载 exercise8e.zip运行所有的练习，可以使用命令 `java -cp exercise8e.zip Exercisei_j` 来运行练习Exercisei_j。例如：要运行练习Exercise2_1，就使用

```
java -cp exercise8e.zip Exercise2_1
```

这也给你一种如何运行程序的思路。

调试提示 编译器经常会给出语法错误出现的原因。如果不知道如何改正它，就在文本里将程序同与其相类似的例子逐字地进行比较。

2.2~2.9节

2.1 (将摄氏温度转换为华氏温度) 编写程序，从控制台读入double型的摄氏温度，然后将其转换为华氏温度，并且显示结果。转换公式如下所示：

$$\text{fahrenheit} = (9/5) * \text{celsius} + 32 \quad (\text{华氏度} = (9/5) * \text{摄氏度} + 32)$$

提示 在Java中，9/5的结果是1，但是9.0/5的结果是1.8。

下面是一个运行示例：

```
Enter a degree in Celsius: 43
43 Celsius is 109.4 Fahrenheit
```



2.2 (计算圆柱体的体积) 编写程序，读入圆柱体的半径和高，并使用下列公式计算圆柱的体积：

$$\text{面积} = \text{半径} \times \text{半径} \times \pi$$

$$\text{体积} = \text{面积} \times \text{高}$$

下面是一个运行示例：

```
Enter the radius and length of a cylinder: 5.5 12
The area is 95.0331
The volume is 1140.4
```



2.3 (将英尺转换为米) 编写程序，读入英尺数，将其转换为米数并显示结果。一英尺等于0.305米。

下面是运行示例：

```
Enter a value for feet: 16 Enter
16 feet is 4.88 meters
```



2.4 (将磅转换为千克) 编写程序, 将磅数转换为千克数。程序提示用户输入磅数, 然后转换成千克并显示结果。一磅等于0.454千克。下面是一个运行示例:

```
Enter a number in pounds: 55.5 Enter
55.5 pounds is 25.197 kilograms
```



*2.5 (财务应用程序: 计算小费) 编写一个程序, 读入一笔费用与酬金率, 计算酬金和总钱数。例如, 如果用户输入10作为费用, 15%作为酬金率, 计算结果显示酬金为\$1.5, 总费用为\$11.5。下面是一个运行示例:

```
Enter the subtotal and a gratuity rate: 15.69 15 Enter
The gratuity is 2.35 and total is 18.04
```



**2.6 (求一个整数各位数的和) 编写程序, 读取一个在0和1000之间的整数, 并将该整数的各位数字相加。例如: 整数是932, 各位数字之和为14。

提示: 利用运算符%分解数字, 然后使用运算符/去掉分解出来的数字。例如: $932\%10=2$, $932/10=93$ 。下面是一个运行示例:

```
Enter a number between 0 and 1000: 999 Enter
The sum of the digits is 27
```



*2.7 (求出年数) 编写程序, 提示用户输入分钟数 (例如十亿) 然后显示这些分钟代表多少年和多少天。为了简化问题, 假设一年有365天。下面是一个运行示例:

```
Enter the number of minutes: 1000000000 Enter
1000000000 minutes is approximately 1902 years and 214 days.
```



2.13节

*2.8 (求ASCII码对应的字符) 编写程序接收一个ASCII码 (从0到128的整数), 然后显示它所代表的字符。例如, 如果用户输入的是97, 程序显示的是字符a。下面是一个运行示例:

```
Enter an ASCII code: 69 Enter
The character for ASCII code 69 is E
```



*2.9 (财务应用程序: 货币单位) 改写程序清单2-10, 解决将double型值转换为int型值时可能会造成精度损失的问题。输入的输入值是一个整数, 其最后两位代表的是分币值。例如: 1156就表示的是11美元56美分。

2.18节

*2.10 (使用图形用户界面输入) 改写程序清单2-10, 使用图形用户界面进行输入和输出。

综合题

*2.11 (财务应用程序: 工资单) 编写程序, 读入下列信息并打印工资单:

- 雇员的名字 (例如Smith)
- 每周工作小时数 (例如10)
- 每小时工资 (例如6.75)
- 联邦所得税税率 (例如20%)

州所得税税率（例如9%）

编写两种版本的程序：（1）使用对话框获取输入并显示输出；（2）使用控制台进行输入和输出。从控制台进行输入和输出的示例运行如下所示：

```

Enter employee's name: Smith
Enter number of hours worked in a week: 10
Enter hourly pay rate: 6.75
Enter federal tax withholding rate: 0.20
Enter state tax withholding rate: 0.09
Employee Name: Smith
Hours Worked: 10.0
Pay Rate: $6.75
Gross Pay: $67.5
Deductions:
    Federal Withholding (20.0%): $13.5
    State Withholding (9.0%): $6.07
    Total Deduction: $19.57
Net Pay: $47.92
  
```



*2.12（财务应用程序：计算利息）如果你知道收支余额和年利率的百分比，你就可以使用下面的公式计算下个月要支付的利息额：

$$\text{利息额} = \text{收支余额} \times (\text{年利率}/1200)$$

编写程序，读取收支余额和年百分利率，显示两个版本的下月利息：（1）使用对话框获取输入并显示输出；（2）使用控制台进行输入和输出。下面是一个运行示例：

```

Enter balance and interest rate (e.g., 3 for 3%): 1000 3.5
The interest is 2.91667
  
```



*2.13（财务应用程序：计算未来投资值）编写程序，读取投资总额、年利率和年数，然后使用下面的公式显示未来投资金额：

$$\text{futureInvestmentValue} = \text{investmentAmount} \times (1 + \text{annuallyInterestRate})^{\text{numberOfYears} \times 12}$$

例如：如果输入的投资金额为1000，年利率为3.25%，年数为1，那么未来投资额为1032.98。

提示 使用方法Math.pow(a, b)来计算a的b次幂。

下面是一个运行示例：

```

Enter investment amount: 1000
Enter monthly interest rate: 4.25
Enter number of years: 1
Accumulated value is 1043.34
  
```



*2.14（医疗应用程序：计算BMI）身体质量指数（BMI）是对体重的健康测量。它的值可以通过将体重（以公斤为单位）除以身高（以米为单位）的平方值得到。编写程序，提示用户输入体重（以磅为单位）以及身高（以英寸为单位），然后显示BMI。注意：一磅是0.45359237公斤而一英寸是0.0254米。下面是一个运行示例：

```

Enter weight in pounds: 95.5
Enter height in inches: 50
BMI is 26.8573
  
```



- **2.15 (财务应用程序: 复利值)** 假设你每月向银行账户存100美元, 年利率为5%, 那么每月利率是 $0.05/12=0.00417$ 。第一个月之后, 账户上的值就变成:

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

第二个月之后, 账户上的值就变成:

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

第三个月之后, 账户上的值就变成:

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

依此类推。

编写程序显示六个月后账户上的钱数。(在练习题4.30中, 你将使用循环来简化这里的代码, 并能显示任何一个月之后的账户值。)

- 2.16 (科学方面: 计算能量)** 编写程序, 计算将水从初始温度加热到最终温度所需的能量。程序应该提示用户输入水的重量 (以千克为单位), 以及水的初始温度和最终温度。计算能量的公式是:

$$Q=M \times (\text{最终温度}-\text{初始温度}) \times 4184$$

这里的M是以千克为单位的水的重量, 温度以摄氏度为单位, 而能量Q以焦耳为单位。下面是一个运行示例:

```
Enter the amount of water in kilograms: 55.5
Enter the initial temperature: 3.5
Enter the final temperature: 10.5
The energy needed is 1.62548e+06
```



- *2.17 (科学方面: 风寒温度)** 外面到底有多冷? 只有温度是不足以提供答案的, 包括风速、相对湿度以及阳光等其他的因素在确定室外是否寒冷方面都起了很重要的作用。2001年, 国家气象服务 (NWS) 利用温度和风速, 使用新的风寒温度来测量寒冷程度。计算公式如下所示:

$$t_{wc} = 35.74 + 0.6215t_a - 35.75v^{0.16} + 0.4275t_av^{0.16}$$

这里的 t_a 是室外的温度, 以华氏摄氏度为单位, 而 v 是速度, 以每小时英里数为单位。 t_{wc} 是风寒温度。该公式不适用于风速低于2mph或温度在 -58°F 以下或 41°F 以上的情况。

编写程序, 提示用户输入在 -58°F 和 41°F 之间的度数, 同时大于或等于2的风速, 然后显示风寒温度。使用 $\text{Math.pow}(a,b)$ 来计算 $v^{0.16}$ 。下面是一个运行示例:

```
Enter the temperature in Fahrenheit: 5.3
Enter the wind speed miles per hour: 6
The wind chill index is -5.56707
```



- 2.18 (打印表格)** 编写程序, 显示下面的表格:

a	b	pow(a, b)
1	2	1
2	3	8
3	4	81
4	5	1024
5	6	15625

- 2.19 (随机字符)** 编写程序, 使用`System.currentTimeMillis()`显示任意一个大写字母。

- 2.20 (几何方面: 两点间距离)** 编写程序, 提示用户输入两个点 (x_1, y_1) 和 (x_2, y_2) , 然后显示两点间的距离。计算两点间距离的公式是 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 。注意: 可以使用 `Math.pow(a, 0.5)` 来计算。下面是一个运行示例:


```
Enter x1 and y1: 1.5 -3.4 Enter
Enter x2 and y2: 4 5 Enter
The distance of the two points is 8.764131445842194
```



- *2.21 (几何方面: 三角形的面积) 编写程序, 提示用户输入三角形的三个点 (x_1, y_1) 、 (x_2, y_2) 和 (x_3, y_3) , 然后显示它的面积。计算三角形面积的公式是:

$$s = (side1 + side2 + side3) / 2;$$

$$area = \sqrt{s(s - side1)(s - side2)(s - side3)}$$

下面是一个运行示例:

```
Enter three points for a triangle: 1.5 -3.4 4.6 5 9.5 -3.4 Enter
The area of the triangle is 33.6
```



- 2.22 (几何方面: 六边形面积) 编写程序, 提示用户输入六边形的边长, 然后显示它的面积。计算六边形面积的公式是:

$$area = \frac{3\sqrt{3}}{2} s^2$$

这里的 s 就是边长。下面是一个运行示例:

```
Enter the side: 5.5 Enter
The area of the hexagon is 78.5895
```



- 2.23 (物理方面: 加速度) 平均加速度定义为速度的变化量除以这个变化所用的时间, 如下式所示:

$$a = \frac{v_1 - v_0}{t}$$

编写程序, 提示用户输入以米/秒为单位的起始速度 v_0 , 以米/秒为单位的终止速度 v_1 , 以及以秒为单位的时间段, 最后显示平均加速度。下面是一个运行示例:

```
Enter v0, v1, and t: 5.5 50.9 4.5 Enter
The average acceleration is 10.0889
```



- 2.24 (物理方面: 求出跑道长度) 假设一个飞机的加速度是 a 而起飞速度是 v , 那么可以使用下面的公式计算出飞机起飞所需的最短跑道长度:

$$length = \frac{v^2}{2a}$$

编写程序, 提示用户输入以米/秒为单位的速度 v 和以米/秒的平方 (m/s^2) 为单位的加速度 a , 然后显示最短跑道长度。下面是一个运行示例:

```
Enter v and a: 60 3.5 Enter
The minimum runway length for this airplane is 514.286
```



- *2.25 (当前时间) 程序清单2-6给出了显示当前格林威治时间的程序。修改这个程序, 使之能够做到程序提示用户输入相对于GMT的时区偏移量, 然后显示在这个特定时区的时间。下面是一个运行示例:

```
Enter the time zone offset to GMT: -5 Enter
The current time is 4:50:34
```

