第 3 章 Java 使用初步

3.1 Java 应用程序的组成元素

Java 应用程序一般由 3 部分组成：注释、import 语句和类声明。本节结合实例对这 3 部分进行简要介绍。

【例 3-1】 设想我们需要编写一个 Java 程序计算任意两个整数之和。在程序运行时首先提示用户输入两个数的值，程序计算出结果，并在屏幕上显示出来。

下面是计算两整数之和的程序代码：

```
/*
File: Addition.java                                          }
功能：计算两个整数之和。                                      } 多行注释
*/                                                           }
import java.util.Scanner;                    <-------------- import 语句
//以下为类声明部分
public class Addition {
    //开始运行程序的主方法                    <-------------- 单行注释
    public static void main(String args[]){
        //创建 Scanner 对象 input
        Scanner input = new Scanner(System.in);
        int number1;                //第一个加数   <-------- 单行注释
        int number2;                //第二个加数
        int sum;                    //sum 存储累加和
        System.out.print("输入第一个加数(整数)：");
        number1 = input.nextInt();
        System.out.print("输入第二个加数(整数)：");
        number2 = input.nextInt();
        sum = number1 + number2;  //进行相加运算
        //输出运算结果
        System.out.print("累加结果：");
        System.out.println(sum);
    }
}                                   //类声明结束
```

其中，第一部分为注释，第二部分为 import 语句，第三部分为类声明，类声明中也包含注释。

1. 注释

程序中除了计算机要执行的指令，还包含注释。我们在注释中陈述程序的目的，解释代

码的含义，并提供任何其他描述来帮助程序员理解程序。

Java 程序的注释分为 3 种，分别是多行注释、单行注释和文档注释。

顾名思义，多行注释可以占多行，以标记/* 开始，以另一个标记 */结束，中间的部分就是注释。开始和结束注释标记是成对匹配的，即每一个开始标记必须有一个对应的结束标记。

单行注释的标记是双斜线//。位于双斜线标记和行末的任何文本都是注释。单行注释放在被注释语句的上面或右侧。

第三种类型的注释称为 Java 文档注释，它是一种专门的注释，可以出现在类声明之前，也可以出现在其他程序元素之前。文档注释由标记/** 开始，由标记 */结束，可以占一行，也可以占多行，具体使用见 4.7 节。

注释只供程序员使用，而被计算机所忽略。在程序中不写注释不影响程序的运行，但会影响程序的阅读和理解。编写正确运行的程序是不够的，还需要编写程序文档，给程序加注释是程序文档的重要部分。程序文档的其他部分包括程序图、程序员的工作日志、设计文档和用户手册。如果能够一次编写完程序，并永远使用，而不需要修改，那么编写不带注释的程序还是可以忍受的。然而，在现实世界中，使用从未做任何变更的程序几乎是不存在的。例如，你可能决定增加新的特性和功能，或者要修改用户与程序交互的方式。即使不改进程序，当检测到程序中的某些错误时，你还是不得不修改程序。另外，对于商用程序，大多数情况下，修改程序的人通常并不是开发程序的人。当程序员需要修改自己或其他人的程序时，首先必须理解程序，而程序文档对于理解程序起到了绝对必要的辅助作用。

2. import 语句

只要可能，我们就通过使用预先定义的类开发面向对象的程序，包括系统定义的类和程序员定义的类。当没有合适的预先定义的类可以使用时，就需要定义自己的类。在 Java 中类被组织到包(packages)中，Java 系统具有很多包。我们还可以将自己的类逻辑组合到包中，这样就可以被其他程序方便地复用。

为了使用包中的类，通过使用下面的格式将类引入到程序中：

```
import <package name>.<class name>
```

包可以包括子包，形成包的层次。在引用嵌入包中的类时，使用多个点。例如，编写语句

```
import java.util.Scanner;
```

来引用 java.util 包中的 Scanner 类。即 util 包在 java 包中。带有类所属的所有包名的点符号表示法称为类的全限定名。使用类的全限定名常常很麻烦，特别是当我们不得不在程序中多次引用相同的类时。可以使用 import 语句来避免这个问题。

在 Addition.java 程序中，由于使用了 import 语句将 java.util.Scanner 类引入到了程序中，在使用 Scanner 类时只写类名即可，其所属的包可以省略。如果没有使用 import 语句，则创建 Scanner 对象 input 的语句需要改为：

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

如果想从同一个包中引入多个类，则可以使用星号将包中的所有类都引入，而不需要分

别为每一个类使用一个 import 语句。使用星号引入类的语句如下：

```
import <package name> . * ;
```

例如，语句

```
import java.util. * ;
```

从 java. util 包中引入了所有类。

3. 类声明

Java 程序由一个类或多个类组成。有些是预先定义的类，而有些是我们自己定义的类。在实例程序 Addition. java 中有两个类：Scanner 和 Addition。Scanner 是一个标准类，而 Addition 是我们自己定义的类。为了定义一个新类，必须在程序中进行类声明。类声明的语法如下：

```
class  <class name>  {
    <class member declarations>
 }
```

其中，<class name>是类名，<class member declarations>是类成员声明的一个序列。单词 class 是保留字，用于标记类声明的开始。类成员或者是数据成员，或者是方法成员。我们可以使用不是保留字的任何合法标识符来命名类。

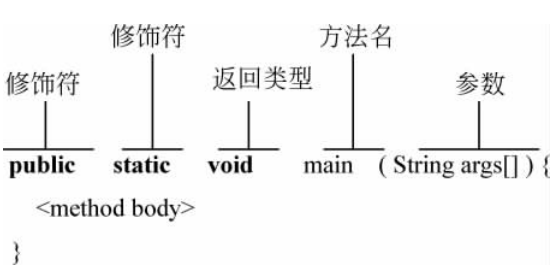
程序中的一个类必须被指定为主类。实例程序的主类是 Addition。如果将一个类指定为主类，则必须定义一个称为 main 的方法。因为当 Java 程序运行时，首先运行主类的 main 方法。要定义一个方法，就必须在类中声明它。

声明方法的语法是：

```
<modifiers> <return type> <method name> ( <parameters> ) {
    <method body>
}
```

其中，<modifiers>是一系列修饰符，指定不同种类的方法；<return type>是方法返回值的类型，<method name>是方法名，<parameters>是传给方法的值序列，<method body>是指令序列。

主方法的声明如下：



这里不解释修饰符、返回类型和参数的含义，在本书的后续章节中将逐步给出详细解释。

3.2 基本数据类型与表达式

3.2.1 基本数据类型

在 Java 中，数字、字符和布尔值都不属于对象，而是一种基本类型值。其中数字型有 6 种，分别为字节型(byte)、短整型(short)、整型(int)、长整型(long)、单精度浮点型(float)和双精度浮点型(double)。表 3-1 列出了 Java 语言中的 8 种基本类型。

表 3-1 Java 语言的基本类型

类 型	长 度	范 围
byte	1B	−128～127
short	2B	−32768～32767
int	4B	−2147483648～2147483647
long	8B	−9223372036854775808L～9223372036854775807L
float	4B	−3.40282347E+38F～3.40282347E+38F
double	8B	−1.79769313486231570E+308～1.79769313486231570E+308
char	2B	'\u0000'～'\uFFFF'
boolean	1b	true,false

在表示 long 常量时，需要使用一个后缀 L；在表示 float 常量时使用后缀 F，例如 3.14159F。

字符类型数据(char)用于存储单个字符，字符以数字形式存储。字符常量包含在一对单引号中，例如'a'。在 ASCII 字符集中，数字 65 对应于字母 A，49 对应于数字 1。Java 字符数据类型是 16 位的，最小值 0，最大值 65 535，放置 Unicode 符号。Unicode 是 ASCII 字符集的扩展集，用于处理多种语言。

Java 也提供转义字符，它以反斜杠(\)开头，将其后的字符转变成另外的含义。表 3-2 列出了 Java 中常用的转义字符。

表 3-2 Java 中常用的转义字符

转义字符	含 义	转义字符	含 义
\'	单引号	\f	送页符
\"	双引号	\t	水平制表符
\\	反斜杠	\b	退格
\r	回车	\un1n2n3n4	Unicode 码
\n	换行符		

3.2.2 变量与常量

变量名和常量名必须是 Java 语言中的合法标识符，因此这里先介绍 Java 语言中的标识符。

1. 标识符(Identifier)

标识符是一个名称,其第一个字符必须是下列字符中的一个:大写字母(A～Z)、小写字母(a～z)、下画线(_)或者($),后面的字符可以是上述字母或者数字(0～9)。

在标识符中有一部分被系统定义,被称为保留字或关键字,用户不能使用。关键字列表如下:

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

$var1、_var2、aInt、student_Number 都是合法标识符,而下列标识符是不合法的:

2student　　(数字不能作为标识符第一个字符)

try　　(关键字不能作为标识符)

var#　　(含有非法字符#)

2. 变量(Variables)

假设我们想要计算两个整数的和与差,这两个整数分别是 x 和 y,则可以使用下面的语句声明整型变量 x 和 y:

```
int  x , y ;
```

一旦进行了声明,就为 x 和 y 分配了内存单元来存储数据值,这些内存单元就称为变量(variable),并且 x 和 y 就是我们给这些内存单元命名的名称。可以使用任何合法有效的标识符作为变量名。当完成上述的变量声明以后,只能给 x 和 y 赋整型数据。

声明变量的一般语法格式为:

```
<data type>  <variables>;
```

<variables>是用逗号隔开的标识符序列。程序中用到的每个变量都必须声明。

可以根据需要以多个声明语句来定义变量。下面是声明不同数据类型的例子:

```
int      i, j, k;
int      x, y;
float    f1, f2;
long     bigInteger;
double   bigNumber;
```

也可以在声明变量时进行初始化。例如,可以在下面的声明语句中将整型变量 x 和 y 分别初始化为 10 和 34。

```
int x = 10,y = 34;
```

3. 常量(Final Variables)

变量的值是可以变化的,在程序运行的过程中,在不同的时刻可以将不同的值赋给同一个变量。但是,在有些情况下,我们希望某些值保持不变,这时就可以使用常量(constant)。声明常量的方式与变量相似,但要增加保留字 final。在声明常量时,必须要给它赋值。下面是常量声明的例子:

```
final int       CAPACITY = 35;            //容量限制
final double    RATE_OF_CALL = 0.4;       //活期存款利率
```

常量 CAPACITY 及 RATE_OF_CALL 称为命名常量(named constant)或符号常量(symbolic constant)。第二种常量称为字面常量(literal constant),并使用实际的值来描述它。上面声明语句中的 35 和 0.4 都是字面常量。

在程序中使用符号常量能够提高程序的可读性,并使程序易于修改。例如,如果活期存款利率改变了,那么只需要修改 RATE_OF_CALL 的声明语句即可,程序中用到活期存款利率的地方都不需要修改。如果没有声明即使用符号常量 RATE_OF_CALL,在程序中使用活期存款利率都使用了字面常量 0.4,修改起来就比较麻烦。一方面,可能有很多地方都用到了 0.4;另一方面,程序中有 0.4 的地方代表的含义不一定是活期存款利率。这样,在修改程序中可能发生漏改或错改的隐患。

3.2.3 表达式

表达式由运算符和运算数组成。运算数可以是常量、变量、方法调用或者是另一个用括号括起来的表达式。

运算符也称为操作符,指明对操作数所进行的运算。按照功能,可以把运算符分为算术运算符、赋值运算符、关系运算符、逻辑运算符、位运算符和条件运算符。

1. 算术运算符

表 3-3 列出了 Java 中所用到的算术运算符。从表 3-3 中可以看到,当两个整数相除时,结果是整数商,即小数部分都被截掉了。两个整数之间的除法称为整除。当两个运算数中任意一个或两个都是浮点数或双精度数时,相除的结果是实数。

表 3-3 算术运算符

运　　算	Java 运算符	例　　子	表达式的结果 (x=15,y=6,z=2.5)
加	+	x+y	21
减	−	x−y	9
乘	*	x*y	90
		y*z	15.0
除	/	x/y	2
		x/z	6.0
模(求余)	%	x%y	3
		y%x	6

续表

运　　算	Java 运算符	例　　子	表达式的结果 (x=15,y=6,z=2.5)
递增	++	++x	16
		x++	15
递减	--	--y	5
		y--	6
求正数	+	+z	2.5
求负数	-	-z	-2.5

模运算的结果是除法的余数。尽管可以对实数进行模运算,但是,大多数情况下,模运算只涉及整数。

加、减、乘、除及模运算都为二元运算,递增、递减、求正数、求负数都为一元运算符,所不同的是,递增和递减运算修改操作数本身的值,++x 和 x++都使 x 的值递增 1,--y 和 y--都使 y 的值递减 1,而其他运算不改变操作数本身的值。

当两个以上的运算符出现在表达式中时,按照先乘除后加减的规则确定它们的计算顺序。例如,在表达式 x+3 * y 中,首先进行乘法运算,然后再进行加法运算。表达式中可以用括号()改变运算次序,适当地使用括号可以使表达式结构清晰,增强程序的可读性。图 3-1 给出了算术运算符及括号的优先规则。

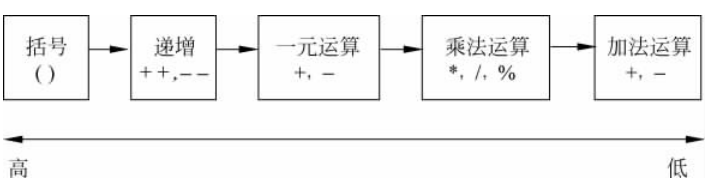


图 3-1　算术运算符及括号的优先规则

2. 赋值运算符

赋值运算符(=)的左侧为变量,右侧为表达式。赋值运算符的作用是将右侧表达式的结果赋值给左侧的变量,例如:

```
int a,b;
a = 8;
b = 3 + 5 * a;
```

赋值运算之后,a 的值为 8,b 的值为 43。

在编写程序时,经常以一个确定的值对变量值递增或递减。例如,要将 sum 的值增加 10,可以写成:

```
sum = sum + 10;
```

可以用复合赋值运算符来改写这个语句,从而不用在赋值号的左右两边重复同一个变量:

```
sum += 10;
```

Java 中常用的复合赋值运算符有+=、-=、*=、/=、%=。

如果想将一个值赋给多个变量，可以串联使用赋值号，例如：

```
a = b = c = 3;
```

与下面 3 个语句等价：

```
c = 3;
b = 3;
a = 3;
```

赋值运算符是从右向左计算的。赋值运算符的优先级低于任何其他运算符。

赋值运算符也可以出现在表达式中，例如：

```
a = 3 + (b = 10);                    //表达式的值是 13,a 的值是 13,b 的值是 10
a = (b = 14)/(c = 7);                //表达式的值是 2,a 的值是 2,b 的值是 14,c 的值是 7
```

3. 关系运算符

关系运算符分为算术比较运算符和类型比较运算符。关系表达式的结果只能是布尔型。算术比较运算符如表 3-4 所示。

表 3-4　算术比较运算符

运　　算	Java 运算符	例　　子	结果(x=15,y=6)
大于	>	x > y	true
大于等于	>=	x >= y	true
小于	<	x < y	false
小于等于	<=	x <= y	false
等于	==	x == y	false
不等于	!=	x != y	true

算术比较运算符的优先级低于算术运算符。在算术比较运算符中，运算符<、>、<=、>=的优先级高于运算符==、!=。

类型比较运算符只有一个：instanceof，用法举例如下：

```
e instanceof Point                   //Point 是一个类
```

如果 e 是 Point 类的一个实例，结果为 true，否则结果为 false。

4. 逻辑运算符

逻辑运算符有 3 个："与"运算符(&&)、"或"运算符(||)和"非"运算符(!)，具体见表 3-5。

表 3-5　逻辑运算符

运算	Java 运算符	说　　明	例　　子	结果 (x=15,y=6)
与运算	&&	两个操作数的值都为 true，运算结果为 true，否则结果为 false	(x>10)&&(y<10)	true
			(x>10)&&(y<5)	false
或运算	||	两个操作数的值都为 false，运算结果为 false，否则结果为 true	(x>10)||(y<5)	true
			(x<10)||(y<5)	false
非运算	!	操作数的值为 false，运算结果为 true；操作数的值为 true，运算结果为 false	!(x>10)	false
			!(y<5)	true

非运算！为一元运算符，其优先级与++、--相同，与运算 && 的优先级高于或运算 ||，两者的优先级都低于算术比较运算符。

5. 条件运算符

条件运算符为三目运算符，语法形式如下：

```
expression ? statement1: statement2
```

首先计算表达式 expression，它的结果应该为一个布尔值，如果该值为 true，则执行语句 statement1，否则执行语句 statement2。语句 statement1 和 statement2 需要返回相同的数据类型。下面是条件运算符的应用举例：

```
boolean isStudent;
int salary;
isStudent = true;
salary = (isStudent?500:1000);
```

3.2.4 类型转换

在很多时候都需要进行类型转换。当不同类型的数据进行混合运算时，在运算之前，系统会进行类型转换；在进行赋值运算时，当表达式运算结果的类型和被赋值的变量类型不一致时，则需要将表达式结果的类型转换成变量所对应的类型。

类型转换或者塑型(typecasting)是将一种数据类型的值转换成另一种数据类型的过程。在 Java 中有隐式(implicit)和显式(explicit)两种类型的塑型。有些类型转换可以自动进行，通常将这种类型转换称为隐式类型转换。图 3-2 箭头所示方向表示可以自动进行隐式转换。按照这个方向，从一种类型转换到另一种类型，不损失任何信息，例如，将 short 转换为 int 或将 float 转换为 double。但从整型转换到 float 或者 double 将损失精度。

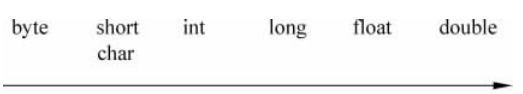


图 3-2 隐式类型转换

如果需要按照图 3-2 箭头相反的方向进行类型转换，如将 double 类型的数据转换成 int 类型或 float 类型，则不能进行隐式转换，需要进行显示转换。显示转换使用塑型运算符(typecast operator)来实现运算数的类型转换。塑型运算符的语法格式为：

```
(<data type>) <expression>
```

例如：

```
double x = 10.0/3.0;          //x 的值为 3.3333333333333335
int n = (int) x;              //n 的值为 3
float f = (float) x           //f 的值为 3.3333333
```

布尔类型和数字类型之间不能进行转换。当字符串(不属于基本数据类型)与数值进行混合运算时(通常为“字符串＋操作数”)，操作数会被自动转换为字符串类型，这种情况在输

出语句中经常遇到。例如，要输出前面 3 个变量的值，可以使用下面的输出语句：

```
System.out.println("x = " + x);
System.out.println("n = " + n);
System.out.println("f = " + f);
```

也可以使用换行符，将上面要输出的内容写在一个输出语句中：

```
System.out.println("x = " + x + "\n" + "n = " + n + "\n" +"f = " + f);
```

但如果需要将字符串转换成数值时，如将字符串"123"转换成数字 123，则不能使用塑型运算符，需要使用包裹类进行转换。包裹类的介绍见 3.4.4 节。

3.3 控制流程语句

Java 语言中的 if 语句和 switch 语句用于选择条件的执行；while 语句、do-while 语句和 for 语句用于循环。

3.3.1 选择结构

1. if 语句

if 语句是 Java 程序中最常见的分支控制语句。if 语句的语法形式如下（其中 else 部分是可选的）：

```
if (boolean - expression)  {
    //statement1;
}
else {
    //statement2;
  }
```

if 语句的一种特殊形式为：

```
if (boolean - expression1) {
    //statement1;
    }
else if (boolean - expression2) {
    //statement2;
    }
          …
else {
    //statement n;
  }
```

注意：else 子句不能单独作为语句使用，它必须和 if 配对使用。else 总是与离它最近的 if 匹配，可以通过使用大括号{}来改变匹配关系。

【例 3-2】 计算每个月的天数。

假设暂不考虑闰年，则 2 月份为 28 天。系统提示输入月份后，系统显示此月有多少天。代码编写如下：

```
…
    if (month == 2)
        days = 28;
    else if ((month == 4) || (month == 6) || (month == 9) || (month == 11))
        days = 30;
    else days = 31;
…
```

2. switch 语句

switch 语句是多分支的选择结构,它的一般格式如下:

```
switch (switch-expression) {
    case value1:    statements for case1; break;
    case value2:    statements for case2; break;
    …
    case valueN:    statements for caseN; break;
    default:        statements for default case; break;
}
```

switch 语句中表达式的值(switch-expression)必须是整型或字符型;常量值 value1 到常量值 valueN 也必须是整型或字符型[①]。switch 语句首先计算表达式的值,如果表达式的值和某个 case 后面的值相同,则从该 case 之后开始执行,直到 break 语句为止;若没有一个常量与表达式的值相同,则从 default 之后开始执行。default 是可有可无的,如果它不存在,并且所有的常量值都和表达式不相同,那么 switch 语句就不会进行任何处理。另外,在同一个 switch 语句中,case 后的值必须互不相同。

在例 3-2 中,计算每个月的天数用的是 if 语句,也可以换成 switch 语句来实现,代码如下:

```
switch(month) {
        case 2: days = 28; break;
        case 4:
        case 6:
        case 9:
        case 11: days = 30; break;
        default: days = 31;
}
```

需要注意的是,switch 语句的每一个 case 判断,都只负责指明分支的入口点,而不指定分支的出口点。分支的出口点需要编程人员用相应的跳转语句 break 来标明。

3.3.2 循环结构

1. while 语句

while 语句实现“当型”循环,其一般语法格式如下:

```
while (check-expression) {
    //body of the loop;
}
```

① JDK 7 以上可以使用字符串。

while 语句的执行过程是先判断条件表达式(check-expression)的值,若为真,则执行循环体,循环体执行完后再无条件转向条件表达式做计算与判断;当计算出条件表达式的值为假时,跳过循环体执行 while 语句后面的语句。

【例 3-3】 计算前 100 个正整数 1,2,…,100 的累加和。

采用 while 语句,可以编写下面的代码计算:

```
int sum = 0, number = 1;
while (number <= 100) {
  sum = sum + number;
  number = number + 1;
}
System.out.println("sum = " + sum);
```

2. do-while 语句

do-while 语句实现"直到型"循环,它的一般语法结构如下:

```
do {
   //body of the loop;
} while (check-expression);
```

do-while 语句的使用与 while 语句很类似,不同的是它不像 while 语句是先计算条件表达式的值,而是无条件地执行一遍循环体,再来判断条件表达式的值,若表达式的值为真,则再运行循环体,否则跳出 do-while 循环,执行下面的语句。可以看出,do-while 语句的特点是它的循环体将至少执行一次。

【例 3-4】 用 do-while 循环计算正整数 1,2,…,100 的累加和。

代码如下:

```
int sum = 0, number = 1;
do{
    sum = sum + number;
    number = number + 1;
}while (number <= 100);
System.out.println("sum = " + sum);
```

3. for 语句

for 循环结构特别适用于计数控制循环,它的一般语法格式如下:

```
for (start-expression; check-expression; update-expression) {
    //body of the loop;
}
```

start-expression 完成循环变量和其他变量的初始化工作;check-expression 是返回布尔值的条件表达式,用于判断循环是否继续;update-expression 用来修整循环变量,改变循环条件。3 个表达式之间用分号隔开。

for 语句的执行过程是这样的:首先根据初始表达式 start-expression,完成必要的初始化工作;再判断表达式 check-expression 的值,若为真,则执行循环体,执行完循环体后再返回表达式 update-expression,计算并修改循环条件,这样一轮循环就结束了。第二轮循环从

计算并判断表达式 check-expression 开始,若表达式的值仍为真,则循环继续,否则跳出整个 for 语句执行 for 循环下面的句子。

【例 3-5】 用 for 循环计算正整数 1,2,…,100 的累加和。

```
int i, sum = 0;
for (i = 1; i <= 100; i++){
    sum += i;
}
```

也可以在初始化部分说明控制变量,如上面的代码可以改为:

```
int sum = 0;
for (int i = 1; i <= 100; i++){
    sum += i;
}
```

这种情况下,变量 i 的生命周期仅限于 for 语句及其循环体中,一旦程序的执行跳出了 for 循环,变量 i 就不能再使用了。

for 循环的循环体中也可以包含另一个或多个 for 循环,这称为 for 循环的嵌套。

【例 3-6】 打印九九乘法表的程序。

```
/*
 File: MultiTable.java
 功能: 打印九九乘法表。
 */
public class MultiTable {
    public static void main(String[] args){
        for (int i = 1; i <= 9;i++) {
            for (int j = 1; j <= i;j++)
                System.out.print("  " + i + "*" + j + "=" + i * j);
            System.out.println();
        }
    }
}
```

在循环体中也可以使用 break 语句终止循环,执行 break 语句将跳出 break 所在的最内层循环。

【例 3-7】 在循环体中使用 break 语句终止循环。

打印九九乘法表的程序也可以改写如下:

```
/*
 File: MultiTable.java
 功能: 打印九九乘法表。
 */
public class MultiTable {
    public static void main(String[] args){
        for (int i = 1; i <= 9;i++) {
            for (int j = 1; j <= 9;j++){
                if (j > i)
```

```
                    break;
                System.out.print("  " + i + " * " + j + " = " + i * j);
            }
            System.out.println();
        }
    }
}
```

在循环体中，除了可以使用 break 语句，还可以使用 continue 语句。continue 语句的作用是终止当前这一轮的循环，跳过本轮循环剩余的语句，直接进入下一轮循环。在 while 或 do-while 循环中，continue 语句会使流程直接跳转至条件表达式；在 for 循环中，continue 语句会使流程跳转至表达式 update-expression，计算并修改循环变量后再判断循环条件。

【例 3-8】 在循环体中使用 continue 语句。

```
public class ContinueTest {
    public static void main( String args[] ) {
         String output = "";
         int i;
        for ( i = 1; i <= 10; i++ ) {
          if ( i == 5 )
             continue;          //skip remaining code in this loop
          output += i + " ";
        }
        output += "\nUsing continue to skip printing 5";
        output += "\ni = " + i;
        System.out.println(output);
    }
}
```

程序运行结果如下：

```
1 2 3 4 6 7 8 9 10
Using continue to skip printing 5
i = 11
```

从上面的例子看到：continue 语句并没有跳出循环体，而是跳过本轮循环，直接进入下一轮循环。

3.4 Java 标准类实例

Java 提供了用于程序开发的类库，称为 Java 基础类(Java Foundational Class，JFC)库，也称为应用程序编程接口(Application Programming Interface，API)，分别放在不同的包中。Java 提供的包主要有 java.lang、java.io、java.util、java.applet、java.awt、java.awt.event、java.awt.image、java.beans、java.net、java.rmi、java.security、java.sql 等。

语言包 java.lang 提供了 Java 语言最基础的类，包括数据类型包裹类(The Data Type Wrapper)、字符串类(String、StringBuffer)、数学类(Math)、系统和运行时类(System、Runtime)等；实用包 java.util 提供了实现各种不同实用功能的类，包括日期类、集合类等；

Java 文本包 java.text 中的 Format、DateFormat、SimpleDateFormat 等类提供各种文本或日期格式。

在能够熟练定义自己的类之前,必须学习如何使用已有的类。本节介绍几个常用的标准类,包括 Math、String、System 类、包裹类、JOptionPane、Date 和 SimpleDateFormat。此处的目的不是充分解释这些类,而是通过对某些标准类的简单解释,使你能够开始编写实际的 Java 程序。更详细的描述,请参考有关标准类的文档。有关标准类的文档通常被称为 Java API 文档。

3.4.1 Math 类

许多计算需要使用数学函数,如三角函数,java.lang 包中的 Math 类以类方法的形式实现了一些有用的数学函数,它同时也包含两个类常量 PI 和 E,分别代表 π 和自然数 e。表 3-6 列出了一些最常用的方法。Math 类中的方法不能作用于对象,只有数字型可以作为它的参数。

表 3-6 常用数学方法

类 方 法	描 述
abs(x)	计算 x 的绝对值
round(x)	计算 x 的四舍五入值
sqrt(x)	计算 x 的平方根
pow(x,y)	计算 x^y(x>0; x=0 且 y>0; x<0 且 y 是整数)
toRadians(x)	将 x 转换为弧度(x 以角度表示)
toDegrees(x)	将 x 转换为角度(x 以弧度表示)
max(a,b)	返回 a 和 b 的较大者
min(a,b)	返回 a 和 b 的较小者
sin(a)	返回 a 的正弦值,a 为弧度
cos(a)	返回 a 的余弦值,a 为弧度
tan(a)	返回 a 的正切值,a 为弧度
asin(a)	返回 a 的反正弦
acos(a)	返回 a 的反余弦
atan(a)	返回 a 的反正切
ceil(a)	返回大于或等于 a 的最小整数
floor(a)	返回小于或等于 a 的最大整数
exp(a)	返回自然数 e(2.718…)的 a 次方
log(a)	返回以 e 为底的 a 的对数
random()	返回一个大于等于 0.0 且小于 1.0 的随机数

在表达式中,使用类方法和类常量的语法格式如下所示:

```
<类名> . <方法名> (< 参数 >)
```

或者

```
<类名> . <类常量>
```

例如，为了表达数学公式

$$\frac{1}{2}\sin\left(x-\frac{\pi}{\sqrt{y}}\right)$$

使用 Math 类中的常量和方法表示如下：

```
(1.0/2.0) * Math.sin(x - Math.PI / Math.sqrt(y))
```

3.4.2 String 类

由双引号分隔的字符序列是 String 常量。由于 String 是一个类，可以生成一个实例，并给这个实例取一个名字。例如，

```
String name;
 name = new String("Java Programming");
```

与其他类不同，明确使用 new 来生成实例对于 String 类是可选的。例如，可以使用下面的方式生成新的 String 对象：

```
String name;
 name = "Java Application Development";
```

在 String 类中定义了将近 50 个方法，这里介绍常用的 charAt、substring、length 和 indexOf 方法。

1. charAt 方法

charAt 方法可以得到一个字符串中的单个字符。字符串的起始位置为 0。例如，可以编写下面的代码：

```
String greeting = "Hello!";
char ch = greeting.charAt(1);                    //将 ch 设定为'e'
```

Java 字符串一旦创建就不能改变，因此，不存在 setCharAt 方法。这似乎有一些局限性，但在实际使用时，给一个字符串变量赋值后，仍然可以给它赋另外一个值，例如：

```
greeting = "Goodbye!";
```

在这种情况下，并没有改变字符串对象"Hello!"的值，只是现在 greeting 不再指向"Hello!"对象了，而是指向了另一个字符串对象"Goodbye!"。

2. substring 方法

substring 方法可以计算一个字符串中的子字符串。其中，需要指出包含在子字符串中的第一个字符的位置以及子字符串中最后一个字符的下一个位置。例如：

```
String greeting = "Hello!";
System.out.println(greeting.substring(1,4));      //结果为"ell"
greeting = "Goodbye!";
System.out.println(greeting.substring(4,7));      //结果为"bye"
```

由于字符串是对象，需要使用 equals 方法比较两个字符串是否包含相同的内容。

【例 3-9】 测试==及 equals 的用法。

```
/*
 File:StringTest.java
 功能: 测试 == 及 equals 的用法。
 */
public class StringTest {
    public static void main(String args[]){
        String greeting, subs;
        greeting = "Goodbye!";
        subs = greeting.substring(4,7);
        System.out.println(subs);

        System.out.print(" == 测试结果:");
        if (subs == "bye")
            System.out.println("true");
        else
            System.out.println("false");
        System.out.print("equals 测试结果:");
        if (subs.equals("bye"))
            System.out.println("true");
        else
            System.out.println("false");
    }
}
```

程序运行结果如下:

```
bye
 == 测试结果:false
equals 测试结果:true
```

3. length 方法

通过使用 length(求长度)方法,可以得到 String 对象中字符的个数。例如,如果变量名 text 指向字符串"China",则

```
text.length()
```

将返回 5,因为在字符串中有 5 个字符。

4. indexOf 方法

要确定一个子字符串在另一个字符串中的位置序号,使用 indexOf (求位置序号)方法。例如,如果变量名 text 指向字符串"I Love Java",则

```
text.indexOf("Love")
```

将返回 2,也就是指定的字符串"Love"的第一个字符的位置序号。如果查找的子字符串在字符串中不存在,则返回-1。注意:查找是以区分大小写的方式进行的。因此,

```
text.indexOf("java")
```

会返回-1。如果相同的子字符串出现两次及以上,则返回第一次匹配子串中第一个字符的

位置序号。

3.4.3 System 类

在程序运行过程中,经常要进行输入/输出操作。本节介绍如何通过 System 类提供的方法进行输入与输出操作。

1. 标准输出

有很多不同的方法可以显示多行文本,本节介绍一种最简单的方法。当执行前面的例子程序时,会看到屏幕上出现一个黑色背景的窗口,类似于第 2 章中图 2-3 和图 2-4 所显示的窗口。这个窗口称为标准输出窗口(Standard Output Window)。通过 System.out 可以将多行文本(也可以将任何数值型的值转换成文本)输出到这个窗口上。使用的 Java 开发工具不同,这个标准输出窗口的实际外观就可能不同。

System 类包含许多有用的类属性,其中之一是 PrintStream 类实例,称为 out。由于这是一个类属性,可以通过类名来引用,即 System.out,而且这个 PrintStream 对象依赖于标准输出窗口。传给 System.out 的每个文本都会出现在标准输出窗口上。使用 System.out 输出数据的技术称为标准输出(Standard Output)。可以混合使用 print 和 println 方法进行文本输出。

2. 标准输入

如同 System.out 用来输出一样,System.in 用来实现输入。使用 System.in 输入数据的技术称为标准输入(Standard Input)。有时也用术语——控制台输入(Console Input)来指标准输入。使用 System.in 输入比使用 System.out 输出稍微复杂一些。System.in 是 InputStream 类的一个实例,使用它的 read 方法一次只能输入 1 个字节。然而,原始数据类型或字符串需要占用多个字节(例如,用 4 个字节存储一个整型值)。所以,为了方便地输入原始数据值或字符串,需要输入设施将多字节作为一个整体来处理。

使用 System.in 实现输入的最常用方法是将 Scanner 对象和 System.in 对象结合在一起使用。Scanner 对象允许读入原始数据类型的值和字符串,它属于 java.util 包。首先,通过传递 System.in 作为参数来生成一个新的 Scanner 对象:

```
import java.util.*;
…
Scanner input;
input = new Scanner(System.in);
```

一旦有了 Scanner 对象 input,就可以调用它的 nextInt 方法输入整型数据了。下面的例子说明如何输入一个人的年龄:

```
int age;
age = input.nextInt( );
```

也可以在一行中输入两个以上的值,下面的代码就是读入两个整数,并输出累加和:

```
int n1, n2;
System.out.print("输入两个整数: ");
n1 = input.nextInt( );
n2 = input.nextInt( );
```

```
System.out.print("sum = ");
System.out.println(n1 + n2);
```


代码运行时,系统提示用户输入两个整数。用户必须在输入完第二个值之后按 Enter 键。注意:两个值之间用空格隔开。任意多个空白符号都可以用来分隔两个输入的值,如空格、跳格(tabs)或新行。

输入其他原始数值型数据时,使用相对应的方法,例如 readDouble 方法和 readLong 方法。表 3-7 列出了 6 种输入方法来输入数值型数据。

表 3-7　6 种数值型数据的输入方法

方　　法	举　　例
nextInt()	int i = input.nextInt();
nextShort()	short s = input.nextShort();
nextLong()	long l = input.nextLong();
nextByte()	byte b = input.nextByte();
nextFloat()	float f = input.nextFloat();
nextDouble()	double d = input.nextDouble ();

在默认情况下,空格是输入值之间的分隔符,用户可以在一行中输入多个值。如果想限制用户在一行中只输入一个值,可以将行分隔符作为定界符,这样的话,用户就必须按 Enter 键来分隔输入值。

可以通过调用 useDelimiter 方法并传递合适的参数来改写默认的定界符。下面的代码描述了如何将行分隔符作为定界符:

```
Scanner input = new Scanner(System.in);
String lineSeparator = System.getProperty("line.separator");
input.useDelimiter(lineSeparator);
```

一旦改写了默认的定界符,空格将不再是定界符。因此,如果在一行中输入多个值,就会产生错误。

读入一个字符串比读入一个数值型的数据稍微复杂一些。为了输入一个单词,使用 next()方法如下:

```
Scanner input = new Scanner(System.in);
String name;
System.out.print("请输入姓名: ");
name = input.next();
```

如果想读入一行作为一个输入的话,就必须改写定界符,并且将行分隔符作为定界符。

如果有多个输入,输入中既有字符串数据,又有原始类型的数据,建议将行分隔符作为定界符,并且每行只输入一个值。

3.4.4　包裹类

对应 Java 的每一个基本数据类型(Primitive Data Type)都有一个数据包裹类,如表 3-8 所示。

表 3-8　基本数据类型及数据包裹类

基本数据类型	数据包裹类	基本数据类型	数据包裹类
boolean	Boolean	int	Integer
byte	Byte	long	Long
char	Character	float	Float
short	Short	double	Double

每个包裹类都有从基本数据类型的变量或常量生成包裹类对象的构造方法，如可以使用下面的代码生成 Double 类的对象：

```
double x = 1.2;
Double a = new Double(x);
Double b = new Double( - 5.25);
```

除了 Character 类以外，其他每个包裹类都有从字符串生成包裹类对象的构造方法，只要字符串中包含的是合法的基本数据类型。例如：

```
Double c = new Double(" - 2.34");
Integer i = new Integer("1234");
```

有时，已知字符串，可使用 valueOf()方法将其转换成包裹类对象，例如：

```
Integer.valueOf("125")                  //返回表示 125 的 Integer 类对象
Double.valueOf("5.15")                  //返回表示 5.15 的 Double 类对象
Float.valueOf("2.43513")                //返回表示 2.43513 的 Float 类对象
Boolean.valueOf("true")                 //返回表示 true 的 Boolean 类对象
Character.valueOf("G")                  //返回表示字符 G 的 Character 类对象
```

每个包裹类都提供相应的方法将包裹类对象转换回基本数据类型的数据。例如：

```
anIntegerObject.intValue()              //返回 int 类型的数据
aFloatObject.floatValue()               //返回 float 类型的数据
aDoubleObject.doubleValue()             //返回 double 类型的数据
aLongObject.longValue()                 //返回 long 类型的数据
aCharacterObject.charValue()            //返回 char 类型的数据
aBooleanObject.booleanValue()           //返回 boolean 类型的数据
aShortObject.shortValue()               //返回 short 类型的数据
aByteObject.byteValue()                 //返回 byte 类型的数据
```

Integer、Float、Double、Long、Byte 及 Short 类提供了特殊的方法能够将字符串类型的对象直接转换成对应的 int、float、double、long、byte 或 short 类型的数据。例如：

```
Integer.parseInt("234")                 //返回 int 类型的数据
Double.parseDouble("234.6576533254")    //返回 double 类型的数据
Float.parseFloat("234.78")              //返回 float 类型的数据
Long.parseLong("23454654")              //返回 long 类型的数据
```

3.4.5　JOptionPane 类

JOptionPane 类提供的 showMessegeDialog 方法可以用来输出信息，showInputDialog

方法可以用来输入信息。

1. showMessegeDialog 方法的使用

当程序计算出结果时,需要一种方法将结果显示给用户。除了使用前面用到的 System.out.print 方法,还可以使用 JOptionPane 类提供的方法。

【例 3-10】 showMessegeDialog 方法举例。

```
/*
 File:ShowMessage.java
 功能:显示消息对话框。
 */
import javax.swing.*;
public class ShowMessage {
    public static void main(String args[]){
        JOptionPane.showMessageDialog(null, "Hello, I'm Marry!");
    }
}
```

程序的运行结果如图 3-3 所示。如图 3-3 所示的对话框就显示在屏幕的中央。

图 3-3 JOptionPane 类生成的"消息"对话框

在图形用户界面(GUI)环境中,大体上有两种类型的窗体:框架窗体和对话窗体。在 Java 中,使用 JFrame 对象作为框架窗体,使用 JDialog 对象作为对话窗体。showMessageDialog 方法的第一个参数是控制这个对话的框架对象,第二个参数是要显示的文本。在上面的样例语句中,传送了保留字 null,意思是没有框架对象。如果传送 null 作为第一个参数,则对话框显示在屏幕的中央;如果传送一个框架对象,则对话框被放置在此框架的中央。

下面对 ShowMessege 类进行修改,使其在一个框架对象中放置对话框:

```
/*
 File:ShowMessage.java
 功能:在框架对象中显示消息对话框。
 */
import javax.swing.*;
public class ShowMessage {
    public static void main(String args[]){
        JFrame jFrame;
        jFrame = new JFrame();
        jFrame.setSize(400,300);
        jFrame.setVisible(true);
        JOptionPane.showMessageDialog(jFrame, "Hello, I'm Marry!");        }
}
```

注意:showMessageDialog 是类方法,因此没有生成 JOptionPane 对象。如果需要更复杂的对话,则应该生成 JDialog 实例。但是,对于文本的简单显示,调用 JOptionPane 的类方法 showMessageDialog 就足够了。

2. showInputDialog 方法的使用

除了使用 JOptionPane 的类方法 showMessageDialog 显示字符串,也可以使用这个类

输入字符串。要将输入字符串赋给变量名 input，可编写下面的语句：

```
String input;
 input = JOptionPane.showInputDialog(null, "Enter text: ");
```

【例 3-11】 showInputDialog 方法举例。

```
/*
 File: ShowInput.java
 功能：从对话框中读入字符串。
 */
import javax.swing.*;
public class ShowInput {
    public static void main(String args[]){
        String name;
        name = JOptionPane.showInputDialog(null,"请输入你的姓名：");
    JOptionPane.showMessageDialog(null, "欢迎你," + name + "!");
    }
}
```

运行上面的程序，系统显示的输入对话框如图 3-4 所示。输入姓名后，单击“确定”按钮，观察系统弹出的输出对话框。

图 3-4 JOptionPane 类生成的输入对话框

使用 showInputDialog 方法输入数值型数据要麻烦一些。由于 showInputDialog 方法返回一个 String 对象，因此，不能将它直接赋值给一个数值型变量，需要使用 3.3.4 节介绍的包裹类进行类型转换。

例如，为了输入一个整型值，如年龄，可以编写如下代码：

```
String str = JOptionPane.showInputDialog(null,"请输入年龄：");
int age = Interger.parseInt(str);
```

3.4.6 Date 类和 SimpleDateFormat 类

Date 类用于表示时间实例，精确到毫秒（千分之一秒）。Date 类在 java.util 包中。当生成一个新的 Date 对象时，将此对象设置为生成它的时间（通过读取本机操作系统所维护的时间来决定当前时间）。Date 类包括 toString 方法，此方法将时间的内部格式转换成字符串表示，之后使用字符串表示来显示时间。

【例 3-12】 测试 Date 类。

```
/*
 File: DateTest.java
 功能：测试 Date 类的使用
 */
import java.util.*;
public class DateTest {
    public static void main(String args[]){
        Date today;
        today = new Date();
```

```
        System.out.println(today.toString());
    }
}
```


将以下面的格式显示当前时间:

```
Fri Apr 01 11:16:44 CST 2017
```

注意:将当前时间转换成字符串格式时,还包括日期信息。在内部,时间以从标准时间开始所流逝的毫秒时间保存,将此标准时间称为新纪元,即格林尼治时间时间(Greenwich Mean Time,GMT):1970-1-1 00:00:00。

如果不喜欢默认格式,例如只想显示月份和年度,或以 AM/PM 标准只显示时和分,则可以使用 SimpleDateFormat 类。SimpleDateFormat 类在 java.text 包中。

【例 3-13】 测试 Date 类及 SimpleDateFormat 类。

如果想以 MM/dd/yy 简写格式显示月、日和年,如 04/01/17,则可以如下改写 DateTest.java 程序:

```
/*
 File: DateTest.java
 功能:测试 Date 类及 SimpleDateFormat 类的使用。
 */
import java.util.*;
import java.text.*;
public class DateTest {
    public static void main(String args[]){
        Date today;
        SimpleDateFormat sdf;
        today = new Date();
        sdf = new SimpleDateFormat("MM/dd/yy");
        System.out.println(sdf.format(today));
    }
}
```

如果是 2017 年 4 月 1 日,上面程序的运行结果为

```
04/01/17
```

注意:当生成新的 SimpleDateFormat 类对象时,是通过传送格式化字符串来确定格式标准的。格式化字符串中的字母是区分大小写的。此例中的格式化字符串必须是 MM/dd/yy,并且字母 d 和 y 必须小写。通过增加格式化字母的数量,可以改变信息的长度,如用 2017 代替 17。对于月份,可以将数字改为月份名。例如,当将 sdf 修改为

```
sdf = new SimpleDateFormat("MMMM dd, yyyy");
```

时,此运行结果将显示:

```
4 月 01,2017
```

当想显示星期几时,可以使用字母 E,将 sdf 修改为:

```
sdf = new SimpleDateFormat("EEEE");
```

本章小结

本章介绍了 Java 编程的基础知识，使读者了解 Java 应用程序的组成元素、基本数据类型与表达式，以及不同数据类型之间的转换；讲解了 Java 语言中的选择结构与循环结构，以及如何使用 Java 提供的标准类进行编程。

习　　题

3.1　注释的作用是什么？你认为注释是不是可有可无的？请说出可以用到的注释的种类。

3.2　import 语句的作用是什么？Java 程序是否总要包括 import 语句？说明引入包中的一个类和引入包中的所有类的语法。

3.3　描述开发和运行 Java 程序的步骤及在每一步中所使用的工具。

3.4　描述对象声明和对象生成之间的区别，使用内存状态图举例说明这种区别。

3.5　编写程序，接收用户从键盘上输入的 3 个整数 x、y、z，从中选出最大和最小者，并编程实现。

3.6　编写应用程序，将输入的摄氏(Celsius)温度转换为华氏(Fahrenheit)温度，分别使用 System.in 和 System.out 进行输入输出。从摄氏温度转换为华氏温度的公式是：

$$\text{Fahrenheit}=1.8*\text{Celsius}+32$$

3.7　编写应用程序求解下面的二次方程式：

$$Ax^2+Bx+C=0$$

其中，系数 A、B、C 都是实数，它的两个实数解可以根据下面的公式得到：

$$x=\frac{-B\pm\sqrt{B^2-4AC}}{2A}$$

当然对于这个习题，可以假定 $A\neq 0$ 并且

$$B^2\geqslant 4AC$$

成立，因此 x 有实数解。使用标准的输入输出。

3.8　分别用 for、do-while 和 while 语句计算下面公式的结果。

(1) $1+2+3+\cdots+100$

(2) $1+3+7+15+31+\cdots+(2^{20}-1)$

(3) $1\times 2\times 4\times 8\times\cdots\times 2^{20}$

3.9　求出 100 以内的素数，并将这些数在屏幕上 5 个一行地显示出来。

3.10　从键盘上输入一件物品的价格(范围在 0.10～5.00 元)，假设用户付了一张 5 元纸币，请列出一种找零的方案，使得纸币及硬币的个数最少。如 3.68 元，应为：2 元 1 张，1 元 1 张，5 角 1 个，1 角 1 个，5 分 1 个，2 分 1 个，1 分 1 个。

3.11　编写一个程序，对于给定的年份，回答“Leap Year”(闰年)或者“Not a Leap Year”(非闰年)。如果一个年份能被 4 整除，但不能被 100 整除，它就是闰年(例如，1976 就是闰年，因为它能被 4 整除，但不能被 100 整除)。如果一个年份能同时被 4 和 100 整除，即

能被 400 整除,也是闰年(例如,2000 是闰年,而 1800 就不是闰年)。

3.12 编写 Java 应用程序,以下面的格式显示一个日期:

```
Monday May 9,2011
```


3.13 编写应用程序,确定给定学期的天数。将一学期的第一天和最后一天的年、月、日信息输入程序。

提示:为学期的开始日期和结束日期各生成一个 GregorianCalendar 对象,并对它们的 DAY_OF_YEAR 数据进行操作。