第3章 Java语言基础

本章主要内容：

- 数据类型
- 变量
- 基本类型变量
- 数据类型的转换规则
- 从键盘输入数据的语句格式
- 运算符

本章主要介绍编写 Java 程序必须掌握的若干语言基础知识，包括数据类型、变量、常量、表达式等。掌握这些基础知识，是编写正确 Java 程序的前提条件。

3.1 数据类型

程序在执行的过程中，需要对数据进行运算，也需要存储数据。这些数据可能是由使用者输入的，也可能是从文件中取得的，甚至是由网络上得到的。在程序运行的过程中，这些数据通过变量存储在内存中，以便程序随时取用。

数据存储在内存中的一块空间中，为了取得数据，必须知道这块内存空间的位置，为了方便使用，程序设计语言用变量名来代表该数据存储空间的位置。将数据指定给变量，就是将数据存储到对应的内存空间，调用变量，就是将对应的内存空间中的数据取出来使用。

一个变量代表一个内存空间，数据就存储在这个空间中，使用变量名来取得数据非常方便，然而由于数据在存储时所需要的容量各不相同，不同的数据就必须要分配不同大小的内存空间来存储，因此在 Java 语言中对不同的数据用不同的数据类型来区分。

在程序设计中，数据是程序的必要组成部分，也是程序处理的对象。不同的数据有不同的数据类型，不同的数据类型有不同的数据结构、不同的存储方式，并且参与的运算也不相同。通常计算机语言将数据按其性质进行分类，每一类称为一种数据类型(data type)。数据类型定义了数据的性质、取值范围、存储方式以及对数据所能进行的运算和操作。程序中的每一个数据都属于一种类型，定义了数据的类型也就相应决定了数据的性质以及对数据进行的操作，同时数据也受到类型的保护，确保对数据不进行非法操作。

Java 语言中的数据类型分为两大类，一类是基本数据类型(primitive types)，另一类是引用数据类型(reference types)，简称引用类型。基本数据类型是由程序设计语言系统所定义、不可再划分的数据类型。基本数据类型的数据所占内存的大小是固定的，与软硬件环境无关。基本数据类型在内存中存放的是数据值本身。引用数据类型在内存中存放的是指向该数据的地址，不是数据值本身，它往往由多个基本数据组成，因此，对引用数据类型的应用称为对象引

用,引用数据类型也被称为复合数据类型,在有的程序设计语言中称为指针。

基本数据类型有整型、浮点型、逻辑型和字符型;引用数据类型包括类、数组和接口等。本节只介绍基本数据类型,引用数据类型在后面章节中再进行介绍。

Java 语言的数据类型实际上都是用类实现的,即引用对象的使用方式,同时 Java 语言也提供了类似 C 语言中简单类型的使用方式,即声明类型的变量。

Java 语言定义了 4 类共 8 种基本类型,其中有 4 种整型、2 种浮点型、1 种逻辑型和 1 种字符型,它们的分类及关键字如下。

- 整型　byte、short、int、long。
- 浮点型　float、double。
- 逻辑型　boolean。
- 字符型　char。

1. 整型

整数有正整数、零、负整数,其含义与数学中的含义相同。Java 语言的整数有 3 种进制的表示形式。

- 十进制　用多个 0～9 之间的数字表示,如 123 和－100,其首位不能为 0。
- 八进制　以 0 打头,后跟多个 0～7 之间的数字,如 0123。
- 十六进制　以 0x 或 0X 打头,后跟多个 0～9 之间的数字或 a～f 之间的小写字母或 A～F 之间的大写字母,a～f 或 A～F 分别表示值 10～15,如 0X123E。

Java 语言定义了 4 种表示整数的类型:字节型(byte)、短整型(short)、整型(int)、长整型(long)。每种整型的数据都是带符号位的。Java 语言的每种数据类型都对应一个默认的数值,使得这种数据类型变量的取值总是确定的,体现了其安全性。它们的特性如表 3.1 所示。

表 3.1　Java 语言的整数类型

类　　型	数 据 位	范　　围
byte(字节型)	8	－128～127,即$-2^7 \sim 2^7-1$
short(短整型)	16	－32 768～32 767,即$-2^{15} \sim 2^{15}-1$
int(整型)	32	－2 147 483 648～2 147 483 647,即$-2^{31} \sim 2^{31}-1$
long(长整型)	64	－9 223 372 036 854 775 808～9 223 372 036 854 775 807,即$-2^{63} \sim 2^{63}-1$

一个整数隐含为整型(int 型)。当要将一个整数强制表示为长整数时,需在后面加字母 l 或 L。

2. 浮点型

Java 语言用浮点型表示数学中的实数,也就是说既有整数部分又有小数部分的数。浮点数有两种表示方式。

- 标准记数法　由整数部分、小数点和小数部分构成,如 3.0、3.1415 等。
- 科学记数法　由十进制整数、小数点、小数和指数部分构成,指数部分由字母 E 或 e 跟上带正负号的整数表示,如 123.45 可表示为 1.2345E＋2。

浮点数用于需要小数位精确度高的计算。例如,计算平方根或三角函数等,都会产生浮点型的值。Java 语言的浮点型有单精度浮点(float)和双精度浮点(double)两种。它们的宽度和范围如表 3.2 所示。

一个浮点数隐含为 double 型。若在一个浮点数后加字母 f 或 F,将其强制转换为 float 型。double 型占 8 个字节,有效数字最长为 15 位,之所以称它为 double 型,是因为它的精度

是 float 型精度的两倍，所以又称为双精度型。

表 3.2 Java 语言的浮点数类型

类 型	数 据 位	范 围
float(单精度浮点)	32	1.4E－45～3.4E＋38
double(双精度浮点)	64	4.9E－324～1.8E＋308

3. 逻辑型

逻辑型(boolean)用来表示逻辑值，也称为布尔型。它只有 true 和 false 两个取值。其中，true 代表“真”，false 代表“假”，true 和 false 不能转换成数字表示形式。

所有关系运算(如 a＞b)的返回值都是逻辑型的值。逻辑型也用于控制语句中的条件表达式，如 if、while、for 等语句。

4. 字符型

字符型(char)用来存储单个字符。Java 语言中的字符采用的是 Unicode 字符集编码方案，在内存中占两个字节，是 16 位无符号的整数，一共有 65 536 个，字符的取值范围从 0～65 535，表示其在 Unicode 字符集中的排序位置。Unicode 字符是用\u0000 到\uFFFF 之间的十六进制数值来表示的，前缀\u 表示一个 Unicode 值，后面的 4 个十六进制值表示是哪个 Unicode 字符。Unicode 字符表的前 128 个字符刚好是 ASCII 表。每个国家的字母表的字母都是 Unicode 表中的一个字符。由于 Java 语言的字符类型采用了 Unicode 这种新的国际标准编码方案，因而便于东方字符和西方字符的处理。因此，与其他语言相比，Java 语言处理多语种的能力大大加强。

说明：

(1) 字符型数据的声明只能表示单个字符，且必须使用单引号将字符括上。

(2) Java 语言中所有可见的 ASCII 字符都可以用单引号括起来成为字符，如'a'、'B'、'＊'等。要想得到一个字符在 Unicode 字符集中的取值，必须强制转换成 int 类型，如(int)'a'。

(3) 由于字符型用来表示 Unicode 编码中的字符，所以字符型数据可以转化为整数，其值介于 0～65 535 之间。但要取得该取值范围的数所代表的 Unicode 表中相应位置上的字符，必须强制转换成 char 型，如 int c＝20320; char s＝(char)c;。

Java 语言的 4 类 8 种基本数据类型总结归纳如表 3.3 所示。

表 3.3 Java 语言的基本数据类型

数据类型	关键字	占用字节数	默认数值	取 值 范 围
布尔型	boolean	1	false	true,false
字节型	byte	1	0	－128～127
短整型	short	2	0	－32 768～32 767
整型	int	4	0	－2 147 483 648～2 147 483 647
长整型	long	8	0L	－9 223 372 036 854 775 808～9 223 372 036 854 775 807
单精度浮点型	float	4	0.0F	1.4×10^{-45}～3.4×10^{38}
双精度浮点型	double	8	0.0D	4.9×10^{-324}～1.8×10^{308}
字符型	char	2	'\u0000'	'\u0000'～'\uffff'

为了使用上方便，Java 语言提供了数值型数据的最大值与最小值的标识符及常量值，如表 3.4 所示。

表 3.4 数值型常量的特殊值代码

数据类型	所在的类	最小值代码	最大值代码
byte	java.lang.Byte	Byte.MIN_VALUE	Byte.MAX_VALUE
short	java.lang.Short	Short.MIN_VALUE	Short.MAX_VALUE
int	java.lang.Integer	Integer.MIN_VALUE	Integer.MAX_VALUE
long	java.lang.Long	Long.MIN_VALUE	Long.MAX_VALUE
float	java.lang.Float	Float.MIN_VALUE	Float.MAX_VALUE
double	java.lang.Double	Double.MIN_VALUE	Double.MAX_VALUE

3.2 关键字与标识符

1. 关键字

关键字(keyword)是Java语言中被赋予特定含义的一些单词,它们在程序中有着不同的用途,因此Java语言不允许用户对关键字赋予其他的含义。Java语言定义的关键字如表3.5所示。

表 3.5 Java语言定义的关键字

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

2. 标识符

标识符(identifier)是用来表示变量名、类名、方法名、数组名和文件名的有效字符序列。也就是说,任何一个变量、常量、方法、对象和类都需要有名字,这些名字就是标识符。标识符可以由编程者自由指定,但是需要遵循一定的语法规定。标识符要满足如下的规定。

(1) 标识符可以由字母、数字和下划线(_)、美元符号($)等组合而成。

(2) 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。

在实际应用标识符时,应该使标识符能在一定程度上反映它所表示的变量、常量、对象或类的意义,这样程序的可读性会更好。例如,i1、i2、count、value_add等都是合法的标识符,因关键字不能当作标识符使用,所以2count、high#、null等都是非法的标识符。

同时,应注意Java语言是大小写敏感的语言。例如,class和Class,System和system分别代表不同的标识符,在定义和使用时要特别注意这一点。

用Java语言编程时,经常遵循以下的编码习惯(不是强制性的):类名首字母大写;变量、方法及对象的首字母小写。对于所有标识符,其中包含的所有单词都应紧靠在一起,而且大写中间单词的首字母。例如,ThisIsAClassName,thisIsMethodOrFieldName。若定义常量时,则大写所有字母,这样便可标志出它们属于编译期的常数。Java包(Package)属于一种特殊情况,它们全都是小写字母,即便中间的单词亦是如此。

3.3 常量

常量存储的是在程序中不能被修改的固定值，也就是说，常量是在程序运行的整个过程中保持其值不改变的量。Java 语言中的常量也是有类型的，包括整型、浮点型、逻辑型、字符型和字符串型常量。

1. 整型常量

整型常量可以用来给整型变量赋值，整型常量可以采用十进制、八进制和十六进制表示。十进制的整型常量用非 0 开头的数值表示，如 80，－30；八进制的整型常量用以 0 开头的数字表示，如 016 代表十进制的数字 14；十六进制的整型常量用 0x 或 0X 开头的数值表示，如 0x3E 代表十进制的数字 62。

整型常量按照所占用的内存长度，又可分为一般整型常量和长整型常量，其中一般整型常量占用 32 位，长整型常量占用 64 位，长整型常量的尾部有一个字母 l 或 L，如－32L，0l，3721l。

2. 浮点型常量

浮点型常量表示的是可以含有小数部分的数值常量。根据占用内存长度的不同，可以分为一般浮点(单精度)常量和双精度浮点常量两种。其中，单精度常量后跟一个字母 f 或 F，双精度常量后跟一个字母 d 或 D。双精度常量后的 d 或 D 可以省略。

浮点型常量可以有普通的书写方法，如 3.14f，－2.17d，也可以用指数形式，如 2.8e－2 表示 2.8×10^{-2}，58E3D 代表 58×10^{3}(双精度)。

3. 逻辑型常量

逻辑型常量也称为布尔常量，包括 true 和 false，分别代表真和假。

4. 字符型常量

字符型常量是用一对单引号括起的单个字符，如'a'，'9'。字符可以直接是字母表中的字符，也可以是转义符，还可以是要表示的字符所对应的八进制数或 Unicode 码。

转义符是一些有特殊含义、很难用一般方式来表达的字符，如回车、换行等。为了表达清楚这些特殊字符，Java 语言中引入了一些特别的定义。所有的转义符都用反斜线(\)开头，后面跟着一个字符来表示某个特定的转义符，如表 3.6 所示。

表 3.6 常用的转义字符

转义字符	所代表的意义
\f	换页(form feed)，走纸到下一页
\b	退格(backspace)，后退一格
\n	换行(new line)，将光标移到下一行的开始
\r	回车(carriage return)，将光标移到当前行的行首，但不移到下一行
\t	横向跳格(tab)，将光标移到下一个制表符位置
\\	反斜线字符(backslash)，输出一个反斜杠
\'	单引号字符(single quote)，输出一个单引号
\"	双引号字符(double quote)，输出一个双引号
\uxxxx	1～4 位十六进制数(xxxx)所表示的 unicode 字符
\ddd	1～3 位八制数(ddd)所表示的 unicode 字符，范围在八进制的 000～377

5. 字符串常量

字符串常量是用双引号括起的一串若干个字符(可以是 0 个)。字符串中可以包括转义符,标志字符串开始和结束的双引号必须在源代码的同一行上。例如:

```
"您好,刘女士!\n".
```

6. 常量声明

常量声明的形式与变量声明基本一样,只需用关键字 final 标识,通常 final 写在最前面。例如:

```
final int MAX = 10;
final float PI = 3.14f;
```

Java 语言建议常量标识符全部用大写字母表示。上式 MAX 声明为值是 10 的整型常量,PI 声明为浮点数常量。

程序中使用常量有两点好处:一是增加可读性,从常量名可知常量的含义;二是增强可维护性,程序中多处使用常量时,当要对它们进行修改时,只需在声明语句中修改一处即可。

3.4 变量

在程序中使用的值大多是需要经常变化的数据,用常数值表示显然是不够的。因此,每一种计算机语言都使用变量(variable)来存储数据,变量的值在程序中是可以改变的,使用变量的原则是“先声明后使用”,即变量在使用前必须先声明。

1. 变量声明

计算机程序是通过内存变量来操纵内存中的数据,所以程序在使用任何变量之前首先应该在该变量和内存单元之间建立联系,这个过程称为变量的声明或称变量的定义。因此,也可以说变量存储的是在程序中可以修改的值。变量具有四个基本要素:名字、类型、值和作用域。Java 语言的每个变量都有一个名字,称为变量的标识符,所以对变量的命名一定要遵守标识符的规定。每个变量都具有一种类型,变量的类型决定了变量的数据性质和范围、变量存储在内存中所占空间的大小(字节数)以及对变量可以进行的合法操作等。声明变量包括指明变量的数据类型和变量的名称,必要时还可以指定变量的初始数值。变量声明语句后要加分号“;”。

1) 变量声明格式

一个变量由标识符、类型和可选的初始化值共同定义。变量声明的格式如下:

```
类型 变量名[ = 初值][,变量名[ = 初值]…]
```

其中,“变量名”是一个合法的标识符,变量名的长度没有限制,“类型”是变量所属的数据类型,[]中的是可选项。例如,int i;表示声明了标识符 i 是 int 类型的变量。声明后,系统将给变量分配内存空间,每一个被声明的变量都有一个内存地址值。当有多个变量同属一个类型时,各变量可在同一行定义,且它们之间用逗号分隔。例如:

```
int  i,j,k;
```

表示同时声明了 3 个 int 类型的变量 i,j,k。

2) 变量初始化

在声明变量的同时也可以对变量进行初始化,即赋初值。例如:

```
int  i = 0;
```

表示声明的 i 是 int 类型的变量，且 i 的初值为 0。此时 i 称为已初始化的变量。一个变量被初始化后，它将保存此值直到被改变时为止。

Java 语言程序中可以随时定义变量，不必集中在执行语句之前。

同样也可声明其他类型变量。例如：

```
float   x = 3.14f;
double   v = 3.1415926;
boolean truth = true;
char   c = 'A';
```

2. 变量的赋值

当声明一个变量并没有赋初值或需要重新对变量赋值时，就需要使用赋值语句。Java 语言的赋值语句同其他计算机语言的赋值语相同，其格式为：

```
变量名 = 值;
```

下面举例来说明。

```
boolean b = true;                    //声明 boolean 型变量并赋值
int x,y = 8;                         //声明 int 型变量
float f = 2.718f;                    //声明 float 型变量并赋值
double d = 3.1415;                   //声明 double 型变量并赋值
char c;                              //声明 char 型变量
c = '\u0031' ;                       //为 char 型变量赋值
x = 12;                              //为 int 型变量 x 赋值
```

3.5 数据类型转换

Java 语言的数据类型在定义时就已经决定，因此不能随意转换成其他的数据类型，但 Java 语言容许用户有限度地做类型转换处理，这就是所谓的数据类型转换，简称类型转换。类型转换就是在 Java 程序中，常数或变量从一种数据类型转换到另一种数据类型，但这种转换是有条件的，并不是一种数据类型能任意的转换为另一种数据类型。

1. 数值型不同类型数据的转换

由于数值型也分为不同的类型，所以数值型数据也有类型转换问题。数值型数据的类型转换分为隐含类型转换（或称默认类型转换）和强制类型转换两种。凡是把占用位数较少的数据（简称较短的数据）转换成占用位数较多的数据（简称较长的数据），都使用隐含类型转换，即类型转换由编译系统自动完成，不需要程序做特别的说明。但如果把较长的数据转换成较短的数据时，就要使用强制类型转换，否则就会产生编译错误。

1）自动类型转换

在程序中已经定义好的数值型的变量，若是想以另一种数值类型表示时，Java 语言会在下列条件同时成立时，自动进行数据类型的转换。

（1）转换前的数据类型与转换后的类型兼容。

（2）转换后的数据类型的表示范围比转换前的类型大。

条件（2）说明不同类型的数据进行运算时，需先转换为同一类型，然后进行运算。转换从“短”到“长”的优先关系为：

byte→short→char→int→long→float→double

低————————————————→高

举例来说，若是想将 short 类型的变量 a 转换为 int 类型，由于 short 与 int 皆为整数类型，

符合上述条件(1),而 int 的表示范围比 short 来得大,亦符合条件(2),因此 Java 语言会自动将原为 short 类型的变量 a 转换为 int 类型。

值得注意的是,类型的转换只限该语句本身,并不会影响原先变量的类型定义,而且通过自动类型的转换,可以保证数据的精确度,它不会因为类型转换而损失数据的内容。这种类型的转换方式也称为扩大转换(augmented conversion)。

在一个表达式中若有整数类型为 short 或 byte 的数据参加运算,为了避免溢出,Java 会将表达式中的 short 或 byte 类型的数据自动转换成 int 类型,这样就可以保证其运算结果的正确性,这也是 Java 语言所提供的"扩大转换"功能。

由于 boolean 类型只能存放 true 或 false,与整数及字符不兼容,因此是不可能做类型的转换。接下来看一看当两个数中有一个为浮点数时,其运算的结果会变成如何?

【例 3.1】 数据类型的自动转换。

```
1   //filename: App3_1.java          类型自动转换
2   public class App3_1                                  //定义类 App3_1
3   {
4     public static void main(String[] args)
5     {
6       int a = 155;
7       float b = 21.0f;
8       System.out.println("a = " + a + ",b = " + b); //输出 a,b 的值
9       System.out.println("a/b = " + (a/b));        //输出 a/b 的值
10     }
11  }
```

输出的结果为:

```
a = 155,b = 21.0
a/b = 7.3809524
```

由运行的结果可以看到,当两个数中有一个为浮点数时,其运算的结果会直接转换为浮点数。当表达式中变量的类型不同时,Java 会自动将较少的表示范围转换成较大的表示范围后,再作运算。也就是说,假设有一个整数和双精度浮点数作运算时,Java 会把整数转换成双精度浮点数后再作运算,运算结果也会变成双精度浮点数。

2) 强制类型转换

如果要将较长的数据转换成较短的数据时,就要进行强制类型转换。强制类型转换的格式如下:

```
(欲转换的数据类型)变量名
```

这种强制类型的转换,因为是直接编写在程序代码中,所以也称为显性转换(explicit cast)。经过强制类型转换,将得到一个括号里声明的数据类型的数据,该数据是从指定变量名中所包含的数据转换而来的,但是指定的变量及其数据本身将不会因此而转变。下面的程序说明了在 Java 语言中是如何进行数据类型强制转换的。

【例 3.2】 整型与浮点数据类型的转换。

```
1   //filename: App3_2.java          整数与浮点数的类型转换
2   public class App3_2
3   {
4     public static void main(String[] args)
5     {
6       int a = 155;
```

```
7        int b = 9;
8        float g,h;
9        System.out.println("a = " + a + ",b = " + b); //输出 a,b 的值
10       g = a/b;                                     //将 a 除以 b 的结果放在 g 中
11       System.out.println("a/b = " + g + "\n");     //输出 g 的值
12       System.out.println("a = " + a + ",b = " + b); //输出 a,b 的值
13       h = (float)a/b;                              //先将 a 强制转换成 float 类型后再参加运算
14       System.out.println("a/b = " + h);           //输出 h 的值
15     }
16   }
```

程序运行结果如下：

```
a = 155,b = 9
a/b = 17.0

a = 155,b = 9
a/b = 17.222221
```

当两个整数相除时，小数点之后的数字会被截断，使得运算的结果保持为整数。但由于这并不是预期的计算结果，因此想要使运算的结果为浮点数，就必须将两个整数中的一个或两个强制转换为浮点数类型，例如下面的三种写法均成立。

```
(float)a/b                    //将整数 a 强制转换成浮点数，再与整数 b 相除
a/(float)b                    //将整数 b 强制转换成浮点数，再以整数 a 除之
(float)a/(float)b             //将整数 a 与 b 同时强制转换成浮点数
```

只要在变量名前面加上欲转换的类型，程序运行时就会自动将此行语句里的变量做类型转换的处理，并不影响原先定义的类型。

此外，若是将一个大于变量可表示范围的值赋值给这个变量时，这种转换称为缩小转换(narrowing conversion)。由于缩小转换在转换的过程中可能会因此损失数据的精确度，Java并不会自动做这种类型的转换，此时就必须要由程序员做强制性的转换。

注意：*在程序设计过程中，不推荐从较长数据向较短数据的转换，因为在较长数据向较短数据转换的过程中，由于数据存储位数的缩小，将导致计算数据精度的降低。*

2. 字符串型数据与整型数据相互转换

1) 字符串转换成数值型数据

数字字符串型数据转换成 byte、short、int、float、double、long 等数据类型，或将字符串 true、false 转换成相应的逻辑类型，可以分别使用表 3.7 所提供的 Byte、Short、Integer、Float、Double、Long 和 Booleab 类的 parseXXX()方法完成。

表 3.7 字符串转换成数值型数据的方法

转换的方法	功 能 说 明
Byte. parseByte()	将数字字符串转换为字节型数据
Short. parseShort()	将数字字符串转换为短整型数据
Integer. parseInt()	将数字字符串转换为整型数据
Long. parseLong()	将数字字符串转换为长整型数据
Float. parseFloat()	将数字字符串转换为浮点型数据
Double. parseDouble()	将数字字符串转换为双精度型数据
Boolean. parseBoolean()	将字符串转换为逻辑型数据

例如：

```
String myNumber = "1234.567";                    //定义字符串型变量 myNumber
float myFloat = Float.parseFloat(myNumber);
```

第 2 条语句是将字符串型变量 myNumber 的值转换成浮点型数据后，赋给变量 myFloat。

2）数值型数据转换成字符串

在 Java 语言中，字符串可用加号＋来实现连接操作。所以若其中某个操作数不是字符串，该操作在连接之前会自动将其转换成字符串。所以可用加号来实现自动的转换。例如：

```
int myInt = 1234;                                //定义整形变量 myInt
String  myString = "" + myInt;                   //将整型数据转换成字符串
```

其他数值型数据类型也可以利用同样的方法转换成字符串。

3.6 由键盘输入数据

在程序设计中，经常需要从键盘上读取数据，这时就需要用户从键盘输入数据，从而可以增加与用户之间的交互。利用键盘输入数据，Java 语言提供了两种方式。

方式 1：利用键盘输入数据，其基本格式如下：

```
import java.io.*;
public class class_name    //类名称
{
  public static void main(String[] args) throws IOException
  {
     ⋮
    String str;            //声明 str 为 String 类型的变量
    BufferedReader buf;    //声明 buf 为 BufferedReader 类的变量，该类在 java.io 类库中
    buf = new BufferedReader(new InputStreamReader(System.in));  //创建 buf 对象
     ⋮
    str = buf.readLine();  //用 readLine()方法读入字符串存入 str 中，且须处理 IOException 异常
     ⋮
  }
}
```

这个输入数据的基本结构是固定的格式，其中的有关输入语句的功能将在第 10 章介绍。

该格式由键盘输入的数据，不管是文字还是数字，Java 皆视为字符串，因此若是要由键盘输入数值则必须再经过转换。该格式中的相应语句可写成如下的格式，其作用完全相同。

```
import java.io.*;
public class class_name    //类名称
{
  public static void main(String[] args) throws IOException
  {
     ⋮
    String str;            //声明 str 为 String 类型的变量
    InputStreamReader inp;//声明 inp 为 InputStreamReader 类的变量，该类在 java.io 类库中
    inp = new InputStreamReader(System.in);      //创建 inp 对象
    BufferedReader buf;    //声明 buf 为 BufferedReader 类的变量，该类在 java.io 类库中
    buf = new BufferedReader(inp);               //创建 buf 对象
     ⋮
    str = buf.readLine();  //用 readLine()方法读入字符串存入 str 中，且须处理 IOException 异常
     ⋮
  }
}
```