第2章

数据类型和运算符

本章重点

※ 标识符和关键字

※ 数据类型和运算符

本章目标

※ 数据类型转换

本章任务

学习本章，需要完成以下 2 个工作任务。请记录学习过程中所遇到的问题，可以通过自己的努力或访问 kgc.cn 解决。

任务 1：实现个人简历信息输出

在控制台中输出一个同学的个人简历信息，图 2.1 所示为本任务的输出结果。

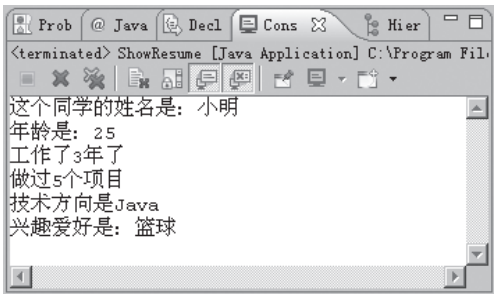

图 2.1　个人简历信息输出

任务 2：实现模拟幸运抽奖

输入 4 位会员卡号，判断是否中奖，并输出中奖结果，图 2.2 所示为本任务的输出结果。

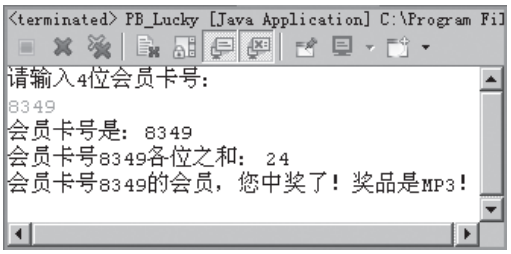

图 2.2　实现模拟幸运抽奖

任务 1　实现个人简历信息输出

关键步骤如下：

- 将个人信息保存在变量中。
- 使用输出语句输出变量中的内容。

2.1.1　使用规范的标识符为变量命名

在 Java 中，标识符用来为程序中的常量、变量、方法、类、接口和包命名。

1. 标识符的命名规则

Java 中的标识符有以下 4 个命名规则。

- 标识符由字母、数字、下划线（_）或美元符号（$）组成。
- 标识符的首字母以字母、下划线或美元符号开头，不能以数字开头。
- 标识符的命名不能与关键字、布尔值（true、false）和 null 相同。
- 标识符区分大小写，没有长度限制，坚持见名知义的原则。

示例 1

请从以下标识符中找出错误的标识符：

$name、_name、1name、name1、name、name$、null、Name、@beijing

分析：

根据标识符命名规则，1name 是错误的，标识符不能以数字开头；null 是错误的，标识符不能是关键字；@beijing 是错误的，标识符只能由字母、数字、下划线或美元符号组成，并且标识符只能以字母、下划线或美元符号开头。

2. 关键字

关键字是 Java 语言保留的，为其定义了固定含义的特殊标识符。

注意：

关键字全部为小写字母，程序员不能将关键字定义为标识符，否则会出现编译错误。

Java 定义的常用 48 个关键字如表 2-1 所示。

表 2-1　Java 中常用的关键字

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

扩充阅读

见名知义原则与驼峰命名法

此部分内容是对平台内容的补充。

见名知义原则是指在使用标识符命名时，要使用能反映被定义者的含义或作用的字符。这样，其他人在阅读代码时通过名称就可以对程序有所理解。

例如，定义姓名时使用 name，定义年龄时使用 age，在定义学生姓名时使用 studentName，在定义老师年龄时使用 teacherAge，一看便能知道其代表的含义，是推荐的用法。如果定义为 a、A1、s 等名称，虽然没有错，但是对于理解程序没有任何意义，应该避免使用。

驼峰命名法就是当使用标识符命名时，如果由一个或多个单词连接在一起，第一个单词以小写字母开始，第二个单词及后续每一个单词的首字母都采用大写字母，这样的变量名看上去就像驼峰一样此起彼伏，故因此得名，如 fileUtil、fileName、dataManager、studentInfo。

驼峰命名法的命名规则可视为一种惯例，并不绝对强制，为的是增强程序的可读性。

2.1.2 使用注释对代码进行解释说明

注释是程序开发人员和程序阅读者之间交流的重要手段，是对代码的解释和说明。好的注释可以提高软件的可读性，减少软件的维护成本。

在 Java 中，提供了 3 种类型的注释：单行注释、多行注释和文档注释。

1. 单行注释

单行注释指的是只能书写在一行的注释，是最简单的注释类型，用于对代码进行简单的说明。当只有一行内容需要注释时，一般使用单行注释。在 MyEclipse 中默认按 Ctrl+/ 快捷键，可以自动产生单行注释。

单行注释的语法格式如下：

```
// 单行注释
```

单行注释以“//”开头。“//”后面的内容都被认为是注释。

示例 2

```
// 年龄
// 姓名
// 工作时间
// 技术方向
// 爱好
// 做过的项目个数
```

注意：

①单行注释不会被编译。

②“//”不能放到被解释代码的前面，否则这行代码会被注释掉。

2. 多行注释

多行注释一般用于说明比较复杂的内容，如复杂的程序逻辑和算法的实现原理等。当有多行内容需要被注释时，一般使用多行注释。

在 MyEclipse 中，选中代码块并按 Ctrl+Shift+/ 快捷键可以生成多行注释；输入“/*”并按 Enter 键将会自动补全多行注释符。

多行注释的语法格式如下：

```
/*
* 个人简历信息输出
*/
```

- 多行注释以“/*”开头，以“*/”结尾。
- “/*”和“*/”之间的内容都被认为是注释。

示例 3

请使用多行注释。

关键代码：

```
/*
* ShowResume.java
* 2016 年 12 月 12 日
 * 个人简历信息输出
*/
public class ShowResume {
  public static void main(String[] args){
   //……此处实现代码省略
  }
}
```

提示：

①注释简单来说就是一种说明，不被当成语句执行，既可以增强代码的可读性，又可以为自己理清思路。

②单行注释添加方便，随处可以添加，只能作用于一行代码。

③当有多行代码需要注释时，如 1000 行代码需要注释，仍然可以采用在 1000 行代码前面添加“//”进行 1000 个单行注释。但是操作起来比较复杂，且不是很美观。于是，可以采用以“/*”开始，以“*/”结尾的多行注释，只需要简单的操作就可以对这 1000 行代码进行注释。

④有时，需要注释的代码行数不多时，可以将其合并为一行，并使用单行注释。但是，当这些行代码中包含不同的含义时，建议使用多行注释，保持各行代码间的不同含义。

3. 文档注释

如果想为程序生成像官方 API 帮助文档一样的文件，可以在编写代码时使用文档注释。使用 JDK 提供的 javadoc 命令，将代码中的文档注释提取出来，可自动生成一份 HTML 格式的 API 帮助文档，其风格与官方 API 帮助文档完全一样，省去了枯燥、烦琐的手动编写帮助文档的工作。

在 MyEclipse 中，输入“/**”，然后按 Enter 键，MyEclipse 会自动显示文档注释格式。

文档注释的语法格式如下：

```
/**
* 文档注释
*/
```

- 文档注释以“/**”开头，以“*/”结尾。
- 每个注释包含一些描述性的文本及若干个文档注释标签。
- 文档注释标签一般以“@”为前缀，常用的文档注释标签如表 2-2 所示。

表 2-2　Java 中常用的文档注释标签

标签	含义	标签	含义
@author	作者名	@version	版本
@parameter	参数及其意义	@since	最早使用该方法、类、接口的 JDK 版本
@return	返回值	@throws	异常类及抛出条件

例如：

```
/**
* 课工场类
*@author kgc
*@version 2.0
*/
```

2.1.3　数据类型

1. 了解 Java 中的数据类型

Java 是强类型语言，在定义变量前需要声明数据类型。在 Java 中主要分为两种数据类型：基本数据类型和引用数据类型。

（1）基本数据类型

Java 中的 8 种基本数据类型如图 2.3 所示。

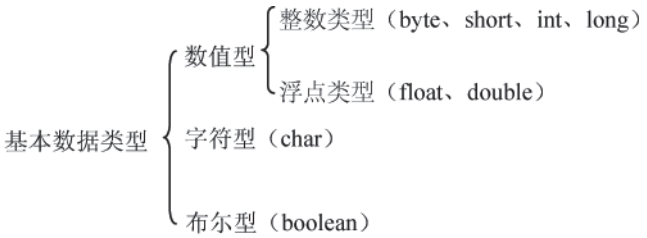

图 2.3　Java 中的基本数据类型分类

其中，int、double、char 等都是 Java 定义的关键字。Java 中的基本数据类型取值范围如表 2-3 所示。

表 2-3　Java 中的基本数据类型取值范围

基本类型	大小	示例	取值范围
boolean	1 字节 8 位	true	true、false
byte	1 字节 8 位有符号整数	-12	-128 ～ +127
short	2 字节 16 位有符号整数	100	-32768 ～ +32767
int	4 字节 32 位有符号整数	12	-2147483648 ～ +2147483647
long	8 字节 64 位有符号整数	10000	-2^{63} ～ $+2^{63}-1$
char	2 字节 16 位 Unicode 字符	'a'	0 ～ 65535
float	4 字节 32 位浮点数	3.4f	-3.4E38 ～ 3.4E38
double	8 字节 64 位浮点数	-2.4e3D	-1.7E308 ～ 1.7E308

注意：

① char 类型占 2 字节，采用 Unicode 码。

② byte 类型占 1 字节，是整数类型的一种。

③所有的数据类型长度固定，不会因为硬件、软件系统不同而发生变化。

④ String 类型不是基本数据类型，而是引用数据类型，它是 Java 提供的一个类。

（2）引用数据类型

Java 中的引用数据类型主要包含类、接口和数组等。

2. 常量

前面认识了 Java 中的标识符及数据类型，下面介绍 Java 中的常量。Java 中的常量指在程序运行中值不能改变的量，举例说明如表 2-4 所示。

表 2-4　Java 中的常量

名称	举例	说明
整型常量	789 // 十进制整型常量	超过 int 类型取值范围的，必须在整数后面加大写的英文字母“L”或小写的英文字母“l”，才能作为 long 类型处理。由于小写“l”容易和数字“1”混淆，一般选用大写字母“L”
浮点型常量	3.4f //float(32bit) -45.9F //float(32bit) -2.4e3D //double(64bit) 3.4 // double(64bit)	Java 的浮点型常量默认是 double，float 需要在数字后面加大写的“F”或小写的“f”
布尔常量	true // 真 false // 假	布尔常量只能为 true 和 false
字符常量	'A'、'8'、'a' // 普通字符常量 '\n' // 转义字符常量：表示换行 '\t' // 转义字符常量：表示按 Tab 键 '\b' // 转义字符常量：表示按 Backspace 键 '\\' // 特殊字符常量：表示反斜杠 '\'' // 特殊字符常量：表示单引号 '\"' // 特殊字符常量：表示双引号	字符常量占用 2 字节内存空间； 转义字符常量都是不可显示字符； 表示单引号、双引号、反斜杠时，再加一个“\”即可
字符串常量	" 课工场 "、"A"	要注意字符和字符串的区别，字符用单引号，字符串用双引号。例如，"A" 和 'A' 是不一样的，前者是字符串，后者是字符
null 常量	null	null 常量只有 null 一个值，可以把 null 常量赋给任意类型的引用类型变量

符号常量	final double PI=3.14;
	double area=PI * r * r; // 计算面积
	double length=PI * r * 2; // 计算周长
		final 含义是指最终的、最后的，代表不能再变了。PI 的值在下面的运算中不能被修改，如果要改变 PI 的值，只能修改第一行定义中 PI 的值

3. 变量

前面讲解了 Java 中的常量，与常量对应的就是变量。变量是在程序运行中其值可以改变的量，它是 Java 程序的一个基本存储单元。

变量的基本格式与常量有所不同。

变量的语法格式如下：

[访问修饰符] 变量类型　变量名 [= 初始值];

- “变量类型”可从数据类型中选择。
- “变量名”是定义的名称变量，要遵循标识符命名规则。
- 中括号中的内容为初始值，是可选项。

示例 4

使用变量存储数据，实现个人简历信息的输出。

分析：

（1）将常量赋给变量后即可使用。

（2）变量必须先定义后使用。

关键代码：

```
/*
 * ShowResume.java
 * 2016 年 12 月 12 日
 * 个人简历信息输出
 */ 造词
public class ShowResume {
  public static void main(String[] args) {
    int age = 25;                          // 年龄
    String name = " 小明 ";                 // 姓名
    int workTime = 3;                      // 工作时间
    String way = "Java";                   // 技术方向
    String favorite = " 篮球 ";             // 爱好
    String projectCount = "5";             // 做过的项目个数

    System.out.println(" 这个同学的姓名是："+name);
    System.out.println(" 年龄是："+age);
    System.out.println(" 工作了 "+workTime+" 年了 ");
    System.out.println(" 做过 "+projectCount+" 个项目 ");
    System.out.println(" 技术方向是 "+way);
    System.out.println(" 兴趣爱好是："+favorite);
  }
}
```

输出结果如下所示：

```
这个同学的姓名是：小明
年龄是：25
工作了 3 年了
做过 5 个项目
技术方向是 Java
兴趣爱好是：篮球
```

4. 数据类型转换

不同的基本数据类型之间进行运算时需要进行类型转换。除布尔类型外，所有基本数据类型进行运算时都要考虑类型转换，主要应用在算术运算时和赋值运算时。

（1）算术运算时

存储位数越多，类型的级别越高。类型转换图如图 2.4 所示。

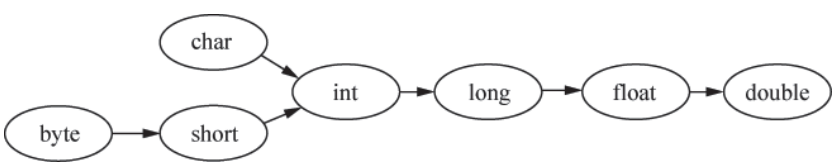

图 2.4　类型转换图

示例 5

```
5+6+7L+'A'
5+5.6*4+'A'
```

分析：

数字 5 和 6 是 int 类型，7L 是 long 类型，而 'A' 是 char 类型。首先两个 int 类型相加，结果还是 int 类型，然后 int 类型和 long 类型相加，自动转换为 long 类型，而 long 类型和 char 类型相加，结果依然是 long 类型。所以第一个表达式结果为 long 类型。同理，第二个表达式结果为 double 类型。不同类型的操作数，首先自动转换为表达式中最高级别的数据类型然后进行运算，运算的结果是最高级别的数据类型，简称低级别自动转换为高级别。

（2）赋值运算时

转换方式有自动类型转换和强制类型转换。

1）自动类型转换

将低级别的类型赋值给高级别类型时将进行自动类型转换。

示例 6

```
byte b=7;
int  i=b;                    //b 自动转换成 int 型
```

分析：

byte 级别比 int 低，所以进行自动类型转换，其转换过程如图 2.5 所示。

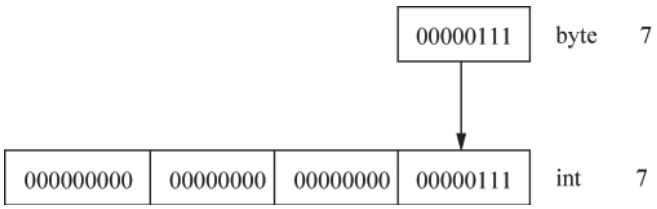

图 2.5　类型转换

2）强制类型转换

将高级别的类型赋值给低级别类型时，必须进行强制类型转换。在 Java 中，使用一对小括号进行强制类型转换。

示例 7

```
int num=786;
byte by=num;                 // 错误
byte by =(byte)num;          // 正确，为强制类型转换
short sh=num;                // 错误
short sh =(short)num;        // 正确，为强制类型转换
```

分析：

byte 和 short 级别比 int 低，所以必须进行强制类型转换，“byte by =(byte)num;”语句的强制类型转换过程如图 2.6 所示。

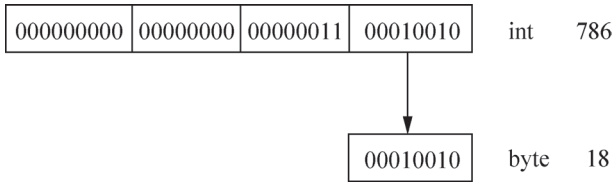

图 2.6　int 类型强制转换为 byte 类型

注意：

进行强制类型转换时，可能会丢失数据。int 类型强制转换为 byte 时，int 的低位第一字节中的数据 00000011 在强制类型转换中会丢失。

“short sh =(short)num;”语句的强制类型转换过程如图 2.7 所示。

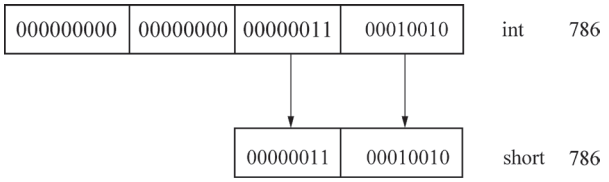

图 2.7　int 类型强制转换为 short 类型

当将 int 类型变量 num 赋值给 short 类型变量 sh 时，将直接取最后 2 字节的内容复制到 sh 的内存空间中，完成强制类型转换，此时并没有丢失数据。

提示：

不仅基本数据类型可以进行类型转换，存在继承关系的引用数据类型也可以进行自动类型转换和强制类型转换，这些内容将在类的继承和多态性章节讲解。

任务 2　实现模拟幸运抽奖

关键步骤如下：

- 获得键盘输入的会员卡号。
- 将会员卡号存储在变量中。
- 使用运算符分解会员卡号的各个位上的数字。
- 将分解后的数字相加判断是否中奖。

要实现任务 2 的幸运抽奖程序，首先要使用 Scanner 类的方法获得用户从键盘输入的数据。

Scanner 类是用于扫描输入文本的实用程序。如果使用 Scanner 类，必须使用

import 语句导入 Scanner 类，即指定 Scanner 类的位置，它位于 java.util 包中。

使用 Scanner 类可以接收用户键盘输入的字符，这是完成任务 2 的第一个关键步骤，实现步骤如下。

（1）导入 Scanner 类。

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

（2）创建 Scanner 对象。

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

（3）获得键盘输入的数据。

表 2-5 中列出了 Scanner 类的常用方法，通过这些方法可以接收用户在键盘输入的字符串、整型数值等。

表 2-5　Scanner 的常用方法

方法	说明
String next( )	获得一个字符串
int nextInt( )	获得一个整型数值
double nextDouble( )	获得一个双精度类型数值
boolean hasNext( )	判断是否有输入数据，如果有输入数据，则返回 true；否则，返回 false

示例 8

使用 Scanner 类获取键盘输入的会员卡号，并将该数据存储在变量中，同时输出这个变量的信息。这是完成任务 2 的第二个关键步骤。

分析：

（1）导入 Scanner 类。

（2）创建 Scanner 对象，获取键盘输入的数据。

（3）将数据存入变量，输出这个变量。

关键代码：

```
import java.util.Scanner;                          // 导入 Scanner 类
public class Lucky{
  public static void main(String[] args){
    int custNo;                                    // 客户会员号
    // 输入会员卡号
    System.out.println(" 请输入 4 位会员卡号：");
    Scanner input=new Scanner(System.in);          //System.in 代表系统输入，如键盘输入
    custNo=input.nextInt( );       //nextInt( ) 获取从键盘输入的一个整数，并赋值给 num 变量
    System.out.println(" 会员卡号是："+custNo);
  }
}
```

注意：

关于 Scanner 类的更多内容，请查看 JDK 文档。

2.2.1　Java 中的运算符

运算符就是告诉程序执行特定的运算操作的符号。Java 中提供了 6 类运算符，分别是赋值运算符、算术运算符、关系运算符、逻辑运算符、位运算符和条件运算符。

1. 赋值运算符

赋值运算符“=”用于给变量指定变量值，并可以和算术运算符结合，组成复合赋值运算符。复合赋值运算符主要包括“+=”“-=”“*=”“/=”“%=”。

示例 9

```
int i=5;
int j=15;
i=i+j;  // 可以替代为 i+=j;
```

分析：

推荐使用复合赋值运算符，将“i=i+j”换为“i+=j”，此写法便于程序编译处理，具有更好的性能。

2. 算术运算符

算术运算符包括“+”“-”“*”“/”“%”“++”“--”，如表 2-6 所示。

表 2-6　算术运算符

运算符	含义	范例	结果
+	加法运算符	5+3	8
-	减法运算符	5-3	2
*	乘法运算符	5*3	15
/	除法运算符	5/3	1
%	取模（取余）运算符	5%3	2
++	自增运算符	i=2; j=i++;	i=3; j=2
--	自减运算符	i=2; j=i--;	i=1; j=2

注意：

①对于除法运算符，如果两个操作数均是整数，结果也是整数，会舍弃小数部分；如果两个操作数中有一个是浮点数，将进行自动类型转换，结果也是浮点数，保留小数部分。

②对于取模运算符，如果两个操作数均是整数，结果也是整数；如果两个操作数中有一个是浮点数，结果也是浮点数，保留小数部分。

③自增运算符有 i++、++i 两种使用方式，它们的相同点是都相当于 i=i+1；不同点是 i++ 是先进行表达式运算再加 1，而 ++i 是先加 1 再进行表达式运算。

➲ 示例 10

完成任务 2，需要使用“/”和“%”运算符分解获得会员卡各个位上的数字，得到分解后的数字之和。

分析：

这是完成任务 2 的第三个关键步骤。

实现步骤：

（1）4 位会员卡号和 10 求余可得个位数。

（2）4 位会员卡号除以 10 再和 10 求余可得十位数。

（3）4 位会员卡号除以 100 再和 10 求余可得百位数。

（4）4 位会员卡号除以 1000 可得千位数。

（5）计算各位之和。

关键代码：

```
import java.util.Scanner;                              // 导入 Scanner 类
public class Lucky{
  public static void main(String[] args){
    int custNo;                                        // 客户会员号
    // 输入会员卡号
    System.out.println(" 请输入 4 位会员卡号：");
    Scanner input=new Scanner(System.in);              //System.in 代表键盘
    custNo=input.nextInt( );       //nextInt( ) 获取从键盘输入的一个整数，并赋值给 num 变量
    System.out.println(" 会员卡号是：" + custNo);
    // 利用“/”和“%”运算符获得每位数字
    int gewei=custNo % 10;                             // 分解获得个位数
    int shiwei=custNo / 10 % 10;                       // 分解获得十位数
    int baiwei=custNo / 100 % 10;                      // 分解获得百位数
    int qianwei=custNo / 1000;                         // 分解获得千位数
    System.out.println(" 千位数：" + qianwei+"，百位数：" + baiwei+"，十位数："
                + shiwei+"，个位数：" + gewei);
    // 利用 "+" 运算符计算各位数字之和
    int sum=gewei + shiwei + baiwei + qianwei;
    System.out.println(" 会员卡号 " + custNo + " 各位之和： " + sum);
  }
}
```

3. 关系运算符

关系运算符有时又称比较运算符，用于比较两个变量或常量的大小，运算结果是布尔值 true 或 false。Java 中共有 6 个关系运算符，分别为“==”“!=”“>”“<”“>=”“<=”。关系运算符的说明如表 2-7 所示。

表 2-7　关系运算符

运算符	含义	范例	结果
==	等于	5==6	false

续表

运算符	含义	范例	结果
!=	不等于	5!=6	true
>	大于	5>6	false
<	小于	5<6	true
>=	大于等于	5>=6	false
<=	小于等于	5<=6	true

注意：

① “=” 为赋值运算符，“==” 为等于运算符。

② “>” “<” “>=” “<=” 只支持数值类型的比较。

③ “==” “!=” 支持所有数据类型的比较，包括数值类型、布尔类型、引用类型。

④关系表达式运算的结果为布尔值。

⑤ “>” “<” “>=” “<=” 优先级别高于 “==” “!=” 。

示例 11

完成任务 2，根据会员卡各个位上的数字之和，判断用户是否中奖。

分析：

使用关系运算符中的“>”判断。这是完成任务 2 的第四个关键步骤。

关键代码：

```
import java.util.Scanner;                          // 导入 Scanner 类
public class Lucky{
  public static void main(String[] args){
    int custNo;                                    // 客户会员号
    // 输入会员卡号
    System.out.println(" 请输入 4 位会员卡号：");
    Scanner input=new Scanner(System.in);          //System.in 代表键盘
    custNo=input.nextInt( );//nextInt( ) 获取从键盘输入的一个整数，并赋值给 num 变量
    System.out.println(" 会员卡号是：" + custNo);
    // 利用 “/” 和 “%” 运算符获得每位数字
    int gewei=custNo % 10;                         // 分解获得个位数
    int shiwei=custNo / 10 % 10;                   // 分解获得十位数
    int baiwei=custNo / 100 % 10;                  // 分解获得百位数
    int qianwei=custNo / 1000;                     // 分解获得千位数
    System.out.println(" 千位数："+qianwei+"，百位数："+baiwei+"，十位数："+shiwei+"，
                个位数："+gewei);
    // 利用 "+" 运算符计算数字之和
    int sum=gewei+shiwei + baiwei+qianwei;
```

```
        System.out.println(" 会员卡号 "+custNo+" 各位之和 :"+sum);
        // 判断是否中奖
        if(sum > 20){
          System.out.println(" 会员卡号 "+custNo + " 的会员，您中奖了！奖品是 MP3 ！ ");
        }else{
          System.out.println(" 会员卡号 "+custNo+" 的会员，您没有中奖 ");
        }
      }
    }
```

提示：

if-else 类型语句称为选择结构，在后续课程中会讲解，这里只要简单了解即可。

4. 逻辑运算符

逻辑运算符用于对两个布尔型操作数进行运算，其结果还是布尔值。逻辑运算符如表 2-8 所示。

表 2-8　逻辑运算符

运算符	含义	运算规则
&	逻辑与	两个操作数都是 true，结果才为 true；不论左边取值，右边的表达式都会进行运算
|	逻辑或	两个操作数一个是 true，结果为 true；不论左边取值，右边的表达式都会进行运算
^	逻辑异或	两个操作数相同，结果为 false；两个操作数不同，结果为 true
!	逻辑反（逻辑非）	操作数为 true，结果为 false；操作数为 false，结果为 true
&&	短路与	运算规则同“&”，不同在于如果左边为 false，右边的表达式不会进行运算
||	短路或	运算规则同“|”，不同在于如果运算符左边的值为 true，右边的表达式不会进行运算

注意：

①操作数类型只能是布尔类型，操作结果也是布尔值。

②优先级别：“!”>“&”>“^”>“|”>“&&”>“||”。

③“&”和“&&”的区别：当“&&”的左侧为 false 时，将不会计算其右侧的表达式，即左 false 则 false；无论任何情况，“&”两侧的表达式都会参与计算。

④“|”和“||”的区别与“&”和“&&”的区别类似。

5. 位运算符

位运算符及运算规则如表 2-9 所示。

表 2-9　位运算符

运算符	含义	运算规则
&	按位与	两个操作数都是 1，结果才为 1
|	按位或	两个操作数一个是 1，结果为 1
^	按位异或	两个操作数相同，结果为 0；两个操作数不同，结果为 1
~	按位非 / 取反	操作数为 1，结果为 0；操作数为 0，结果为 1
<<	左移	右侧空位补 0
>>	右移	左侧空位补最高位，即符号位
>>>	无符号右移	左侧空位补 0

示例 12

计算 5&6 的结果。

实现步骤：

（1）把 5、6 分别转变为二进制数（应该为 32 位二进制数，这里只显示最后 8 位）。

（2）根据按位与运算符的运算规则，两个操作数都是 1，结果才为 1（全 1 得 1）。

（3）最终结果为 00000100，转变为十进制数就是 4。

实现过程如图 2.8 所示。

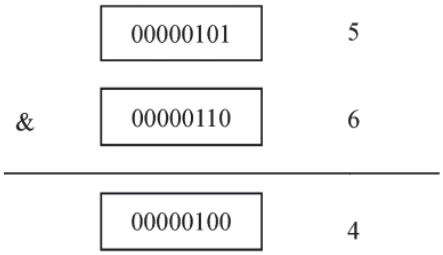

图 2.8　计算 5&6 的结果

示例 13

计算 5|6 的结果。

实现步骤：

（1）把 5、6 分别转变为二进制数（应该为 32 位二进制数，这里只显示最后 8 位）。

（2）根据按位或运算符的运算规则，两个操作数只要有一个是 1，结果就是 1（有 1 得 1）。

（3）最终结果为 00000111，转变为十进制数就是 7。

实现过程如图 2.9 所示。

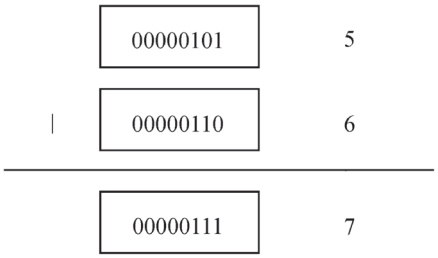

图 2.9　计算 5|6 的结果

示例 14

计算 6<<2 的结果。

实现步骤：

（1）把 6 转变为 32 位二进制数。

（2）让所有二进制位向左移动两位，最高两位溢出，空出的低位一律补 0。

（3）最终结果转变为十进制数就是 24。

实现过程如图 2.10 所示。

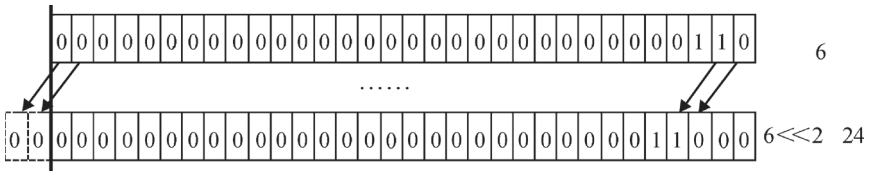

图 2.10　计算 6<<2 的结果

提示：

①一个整数每向左移动 1 位，其值扩大两倍，前提是移出位数不包含有效数字。在示例 14 中表现为移出位中没有 1，且最高位为 0。

② a=a * 4 和 a=a << 2 的作用和结果是相同的，但是使用位运算符执行效率更高。

示例 15

计算 12>>2、12>>3 的结果。

实现步骤：

（1）把 12 转变为 32 位二进制数。

（2）让所有二进制位向右移动两位，最低两位溢出，空出的高位一律补充和最高位相同的数字，此处补 0（12>>3 同理，按 3 位移动）。

（3）最终结果转变为十进制数就是 3（12>>3 为 1）。

实现过程如图 2.11 所示。

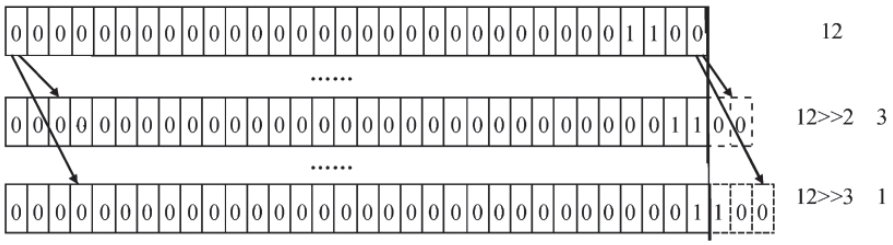

图 2.11　计算 12>>2、12>>3 的结果

提示：

①一个整数每向右移动 1 位，其值缩小 1/2，前提是溢出位中不包含有效数字。

②位运算符对操作数以二进制位为单位进行运算。

③位运算符的操作数是整型数，包括 int、short、long、byte 和 char。

④位运算符的运算结果也是整型数，包括 int、long。

⑤如果操作数是 char、byte、short，位运算前其值会自动晋升为 int，运算结果也为 int。

6. 条件运算符

条件运算符是 Java 中唯一的需要 3 个操作数的运算符，所以又称三目运算符或三元运算符。

条件运算符的语法格式如下：

条件 ？ 表达式 1 ： 表达式 2

- 首先对条件进行判断，如果结果为 true，则返回表达式 1 的值。
- 如果结果为 false，返回表达式 2 的值。

提示：

条件表达式实现的功能和后面要讲解的 if-else 选择结构类似，可以转变为 if-else 语句。

示例 16

```
int min;
min=5<7?5:7;
System.out.println(min);
min=10<7?10:7;
```

System.out.println(min);

分析：

（1）在表达式“min=5<7?5:7;”中，首先判断 5<7 的值，结果为 true，则取表达式 1 的值 5 赋给变量 min，所以 min 的值是 5。

（2）在表达式“min=10<7?10:7;”中，首先判断 10<7 的值，结果为 false，则取表达式 2 的值 7 赋给变量 min，所以 min 的值是 7。

2.2.2 优先级和结合性

Java 中的各种运算符都有自己的优先级和结合性。所谓优先级就是在表达式运算中的运算顺序。优先级越高，在表达式中运算顺序越靠前。

结合性可以理解为运算的方向，大多数运算符的结合性都是从左向右，即从左向右依次进行运算。

各种运算符的优先级如表 2-10 所示，优先级别从上而下逐级降低。

表 2-10 运算符的优先级

优先级	运算符	结合性
1	()、[]、.	从左向右
2	!、～、++、--	从右向左
3	*、/、%	从左向右
4	+、-	从左向右
5	<<、>>、>>>	从左向右
6	<、<=、>、>=、instanceof	从左向右
7	==、!=	从左向右
8	&	从左向右
9	^	从左向右
10	|	从左向右
11	&&	从左向右
12	||	从左向右
13	?:	从右向左
14	=、+=、-=、*=、/=、%=、&=、|=、^=、～=、<<=、>>=、>>>=	从右向左

提示：

①优先级别最低的是赋值运算符，其次是条件运算符。

②单目运算符包括“!”　“~”　“++”　“--”，优先级别高。

③可以通过“( )”控制表达式的运算顺序，“( )”优先级最高。

④总体而言，优先顺序为算术运算符 > 关系运算符 > 逻辑运算符。

⑤结合性为从右向左的只有赋值运算符、三目运算符和单目运算符（一个操作数）。

本章总结

本章介绍了以下知识点：

- Java 中的标识符和使用标识符时需要遵循的规则。
- Java 中的注释分为单行注释、多行注释和文档注释，同时明确了几种注释的使用场合和使用方法。
- Java 中丰富的数据类型及 7 种类型的常量。
- Java 中与常量对应的变量的作用和使用方法。
- Java 中数据类型之间的转换，主要包含自动类型转换和强制类型转换。
- Java 中六大类运算符，分别是赋值运算符、算术运算符、关系运算符、逻辑运算符、位运算符和条件运算符。

本章练习

1．商场为员工提供了基本工资、物价津贴及房租津贴。其中，物价津贴为基本工资的 40%，房租津贴为基本工资的 25%。要求：从控制台输入基本工资，并计算输出实领工资，输出结果如图 2.12 所示。

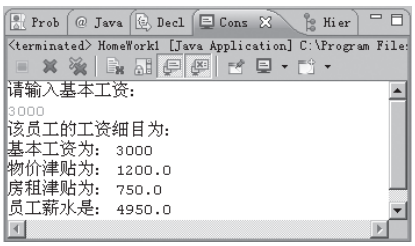

图 2.12　实领工资输出

2．小明左右手分别拿两张纸牌：黑桃 10 和红心 8，现在交换手中的牌。编写一

个程序模拟这一过程：两个整数分别保存在两个变量中，将这两个变量的值互换，并输出互换后的结果，输出结果如图 2.13 所示。

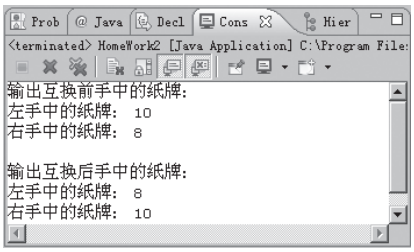

图 2.13　实现牌的交换

3．银行提供了整存整取定期储蓄业务，其存期分为一年、两年、三年、五年，到期凭存单支取本息。年利率如表 2-11 所示。

表 2-11　年利率

存期	年利率
一年	2.25%
两年	2.7%
三年	3.24%
五年	3.6%

编写一个程序，输入存入的本金数目，计算假设存一年、两年、三年或五年，到期取款时，银行应支付的本息分别是多少，输出结果如图 2.14 所示。

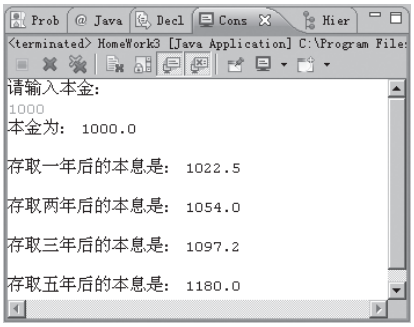

图 2.14　实现本息输出