巍巍交大　百年书香
www.jiaodapress.com.cn
bookinfo@sjtu.edu.cn

丛书策划　张荣昌
责任编辑　王　清　孟海江
封面设计　唐韵设计

软件开发类人才培养系列教材

- .NET开发综合实战
- PHP开发综合实战
- Web前端开发综合实战
- Java开发综合实战（第2版）
- Java Web开发技术任务教程
- Java Web应用与开发（第2版）
- PHP应用与开发
- Bootstrap应用与开发
- Vue.js应用与开发
- jQuery应用与开发
- PHP网络编程入门与进阶
- 项目管理
- 软件测试基础
- 软件工程与测试
- 计算机程序设计语言（VC++）
- C语言程序设计（第2版）
- C#程序设计
- R语言程序设计
- Android程序设计
- Python程序设计项目化教程
- Java语言程序设计实践教程
- Java程序设计实用案例教程
- **Java程序设计基础案例教程**
- 面向对象程序设计项目教程
- JavaScript基础教程
- 网页设计基础教程（HTML5+CSS3+JavaScript）
- HTML5+CSS3 Web前端设计案例教程
- HTML5网页设计与实战

本书提供教学资源包

网址：https://www.sjhtbook.com

扫描二维码
关注上海交通大学出版社
官方微信

ISBN 978-7-313-32948-6
9 787313 329486 >
定价：48.00元

软件开发类人才培养系列教材

Java程序设计基础案例教程

主编　陈　暄　邢红刚　俞立峰

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

软件开发类人才培养系列教材
“互联网+”新形态一体化教材

Java
程序设计基础案例教程

主编　陈　暄　邢红刚　俞立峰

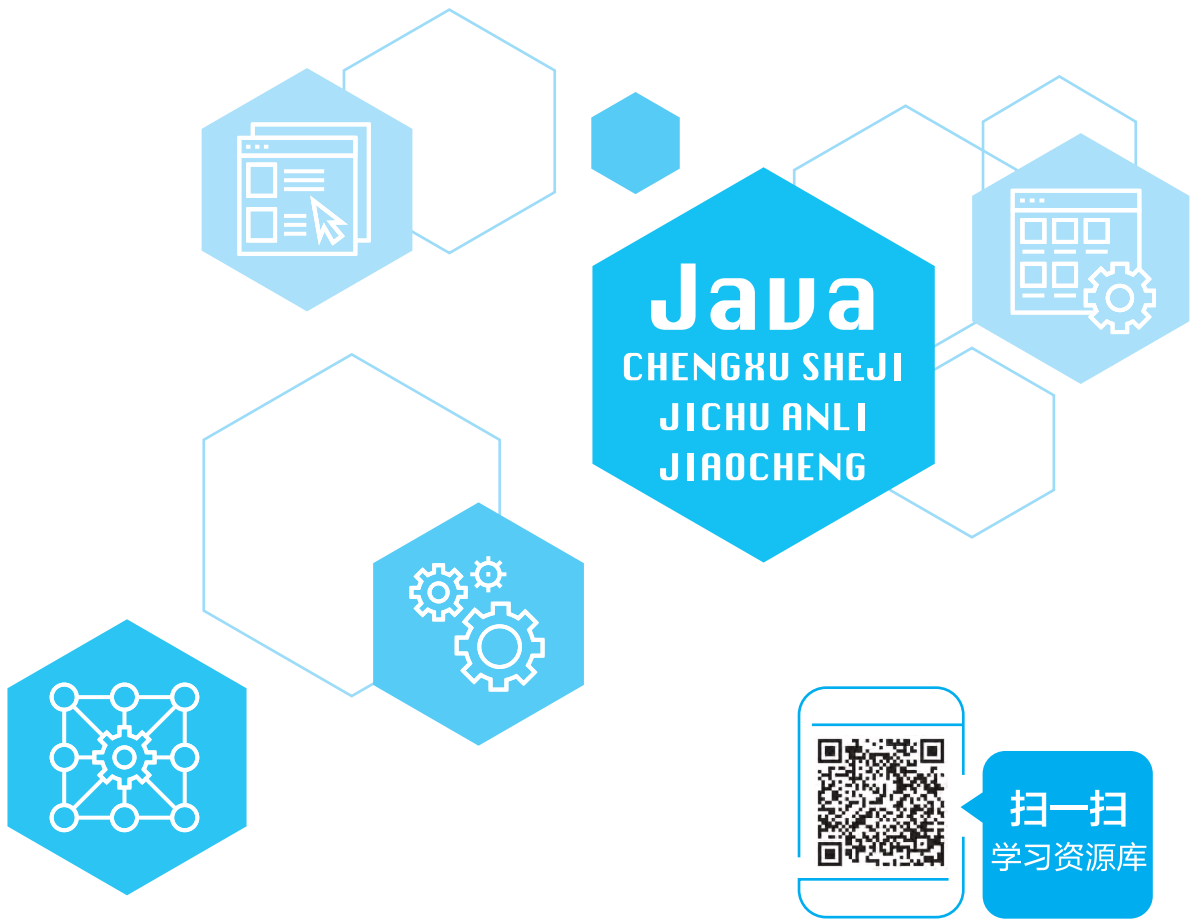
Java
CHENGXU SHEJI
JICHU ANLI
JIAOCHENG

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

软件开发类人才培养系列教材
“互联网+”新形态一体化教材

Java
程序设计基础案例教程

主编　陈　暄　邢红刚　俞立峰



上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内容提要

本书定位于Java程序设计的基础学习阶段，以将基本知识点讲解透彻为宗旨，通过丰富的案例，分8个模块讲解了Java概述、Java基础知识、Java流程控制、数组和字符串、Java面向对象（上）、Java面向对象（下）、Java异常处理和Java常用类等内容。本书可作为职业院校计算机相关专业程序设计类课程的教学用书，也可作为相关技术人员培训或工作的参考用书。

图书在版编目（CIP）数据

Java程序设计基础案例教程 / 陈暄，邢红刚，俞立峰主编. -- 上海 : 上海交通大学出版社，2025. 7.

ISBN 978-7-313-32948-6

Ⅰ. TP312.8

中国国家版本馆CIP数据核字第20257PM375号

Java 程序设计基础案例教程

Java CHENGXU SHEJI JICHU ANLI JIAOCHENG

主　　编：陈　暄　邢红刚　俞立峰
出版发行：上海交通大学出版社
邮政编码：200030
印　　制：北京荣玉印刷有限公司
开　　本：889 mm × 1194 mm　1/16
字　　数：432千字
版　　次：2025年7月第1版
书　　号：ISBN 978-7-313-32948-6
定　　价：48.00元

地　　址：上海市番禺路951号
电　　话：021-6407 1208
经　　销：全国新华书店
印　　张：14
印　　次：2025年7月第1次印刷
电子书号：ISBN 978-7-89564-330-7

编写委员会

主　编　陈　暄　邢红刚　俞立峰

副主编　许炜熔　张　敏　宋雯斐
沈艳洪　王秋月　俞群群

前　言

随着信息技术的飞速发展，编程已成为诸多行业所需的基本技能。作为一种广泛使用的编程语言，Java在全球范围内被广泛应用于软件开发、移动互联网、数据处理等多个领域，已经成为现代计算机科学与技术的重要组成部分。近年来，国家大力贯彻落实数字化、智能化相关战略，培养高素质的信息技术人才成为教育领域的重要任务。

本书积极响应国家关于加强信息技术人才培养的号召，结合当代信息技术最新发展趋势，在帮助学生掌握Java编程的基本理论与实战技能的同时，培养其创新思维和解决实际问题的能力。此外，在习近平新时代中国特色社会主义思想和党的二十大精神的指引下，本书也将课程思政与专业知识有机融合，在专业知识教学中渗透思想政治教育，潜移默化地引导学生树立正确的世界观、人生观、价值观。

本书在考虑学生能力需求的基础上，力争基础理论知识“以实用为准则，以够用为尺度”，增加实践性和应用型的案例，旨在推动理论向实践的转化。本书充分体现了“中高职一体化”的培养理念，绍兴市职教中心为本书提供了案例支持。

本书体现如下特色。

1. 理论与实践相结合

本书不仅注重基础知识的讲解，还通过大量的实际案例与编程习题帮助学生巩固所学知识，使其能够将理论应用到实际的项目开发中。

2. 紧跟技术前沿

本书内容紧密追踪Java语言的最新发展趋势，增加诸如Java 23版本所引入的创新特性与改进内容，旨在深入引导学生熟练掌握现代编程语言Java的核心技术与前沿动态，确保他们能够在快速变化的编程领域中保持竞争力。

3. 加强综合素质教育

本书通过丰富的案例和综合实训，培养学生的团队协作、问题解决、自主创新等软技能，提高其综合素质和全面发展能力。

本书内容由浅入深、循序渐进，针对Java语言中一些核心、基础及高频应用的概念，进行科学的知识体系编排与重难点分散布局，从而充分契合学生的认知规律与教师的教学逻辑。本书秉持理论与实践并重的编写理念，通过通俗易懂的用例，体现程序设计中最基本的思想和方法。

本书由浙江工业职业技术学院陈暄、邢红刚、俞立峰担任主编，绍兴市职教中心许炜熔，浙江工业职业技术学院张敏、宋雯斐、沈艳洪，绍兴职业技术学院的王秋月，以及绍兴冠群信息技术有限公司俞群群担任副主编。书中模块1由宋雯斐、许炜熔编写，模块2由沈艳洪编写，模块3由俞立峰编写，模块4由陈暄编写，模块5由张敏、俞群群编写，模块6由邢红刚编写，模块7由王秋月编写，模块8由陈暄、邢红刚编写。全书由陈暄和邢红刚统稿。

此外，编者为广大一线教师提供了服务于本书的教学资源库，有需要者可发邮件至2393867076@qq.com领取。

由于编者水平有限、编写时间仓促，书中若存在不妥或疏漏之处，敬请广大读者给予批评和指正。

案例导航

目　录

模块 2 Java 基础知识

问题导入

在编程语言的学习过程中，除了关注技术本身，我们也可以结合社会发展、国家需求，以及思想政治教育来深化理解。例如，Java 作为一种强大的编程语言，在现代信息技术领域有着广泛的应用，而这种技术背后往往与国家的科技进步和创新密切相关。党的二十大报告强调了创新驱动发展战略的重要性，并提出了加快实现高水平科技自立自强的目标。其实，这与学习 Java 等编程技术是相辅相成的。通过学习 Java 语言的基本语法和应用开发，不仅能够提升个人的技术能力，也能够为国家的技术创新贡献力量。特别是随着人工智能、大数据、云计算等前沿技术的蓬勃发展，Java 作为重要的开发语言，承担着支撑数字经济、促进社会发展的使命。

学习目标

知识目标

（1）掌握 Java 中各运算符的使用及优先级。
（2）掌握变量和常量的声明。

能力目标

（1）能够熟练使用 Java 运算符完成程序设计。
（2）能够对 Java 变量进行熟练的使用。

素养目标

（1）培养精益求精的工匠精神。
（2）培养遵守社会规范与社会秩序的价值观。
（3）培养环保意识，践行绿色生活。

2.1 编码规范

在编码时要养成良好的编码规范。编码规范是指在编写代码时希望编程人员遵守的一些规定。当然，如果编写人员不遵守，代码并不会出错，但会降低程序的可读性。常见编码规范如下。

2.1.1 注释

注释是对程序语句的说明，有助于开发者之间的交流，方便理解和维护程序。注释不是编程语句，不会被编译器执行。对于代码量较少的程序，加不加注释对理解和修改代码没有太大影响，但如果是一些中大型应用程序，没有代码注释就会增加后台程序运行及维护的难度。在平时的编码过程中养成规范代码注释的习惯，也是能够学好 Java 的重要因素之一。

Java 支持三种注释方式：单行注释、多行注释、文档注释。

（1）单行注释。它主要用于注释少量代码或者说明内容，格式如下：

```
// 单行注释
```

（2）多行注释。它主要用于注释大量的代码或者说明内容，格式如下：

```
/*
 多行注释
 多行注释
*/
```

（3）文档注释。它主要用在类、方法和变量上面，用来描述其作用，说明类的编写时间、作者、用法、参数和返回值等信息，格式如下：

```
/**
 * @author
 * @version jdk23
 */
public class Test1 {
    /**
     * 求输入的两个参数之间的整数的和
     * @param n 接收第一个参数，范围起点
     * @param m 接收第二个参数，范围终点
     * @return 两个参数之间的整数的和
     */
    public int add(int n, int m) {
        int sum = 0;
        for (int i = n; i <= m; i++) {
            sum = sum + i;
        }
```

```
        return sum;
    }
}
```

注意

文档注释能嵌套单行注释，不能嵌套多行注释和文档注释，一般首行和尾行不写注释信息。

2.1.2 代码缩进

在 Java 代码中，每一个层级的代码都应该使用 Tab 键（制表键）进行缩进，缩进 4 个英文字符的长度，比如下面的代码：

```
package com.first;
public class HelloJava {
    public static void main(String[] args) {
        System.out.print(" 你好，Java");
    }
}
```

不同层级的代码要缩进 4 个字符，并用大括号“{}”分割，而且“{}”要采用上面代码所示的配对方式，注意不要采用如下方式。这种大括号配对方式，不是 Java 的编码风格，虽然没有错误，但不符合绝大多数 Java 程序员的编码习惯和审美要求，所以不建议这样写。

```
package com.first;
public class HelloJava
{
    public static void main(String[] args)
    {
        System.out.print(" 你好，Java");
    }
}
```

2.2 标识符与关键字

在学习常量与变量前，要先了解标识符与关键字。标识符是用来表示类名、变量名、方法名、类型名、数组名、文件名的有效字符序列。简单地说，标识符就是一个由数字、字母、美元符（$）和下划线[①]（_）组成的字符序列。

① “下划线”应为“下画线”，为与计算机专业领域使用习惯一致，本书采用“下划线”。

标识符需要同时满足以下条件：

（1）标识符只能由数字、字母、美元符（$）和下划线（_）组成；

（2）标识符不能以数字开头，也不能包含空格；

（3）标识符可以是任意个数的字符，但只有前 32 个字符有效。

有效的标识符：myname、ict_network、Hello、_sys_path、$bill、int、float。

无效的标识符：3abc（第一个字符是数字）、root-1（有“-”）、my=you（有“=”）、100%（有“%”）。

有些标识符是电脑语言里事先定义的，称为关键字或保留字。定义标识符时，应避开关键字，因为每个关键字都有特定的意义。例如：char、int、float 用于声明数据类型；interface 用于声明接口；class 用于声明类；等等。

Java 关键字如图 2-1 所示（按首字母顺序排列）。

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

图 2-1　Java 关键字

2.3 命名规范

在编写 Java 程序的时候，需要先建立一个项目，而项目名称优先考虑英文，如“testjavaproject”“studentmanagement”等，当然也可以用中文，如“五子棋游戏 v01”“java 考试系统”等。

2.3.1 包名

包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词，如“cn.zjipc.sjxy.mycode”“cn.edu.zju.myutil”等。

另外需要注意的是，包名最好统一使用单数形式。但是如果类名有复数含义，类名可以使用复数形式。在创建一个包项目时可能会涉及多层设计，每层的包名要遵循包名全部小写的规范。

2.3.2 类名

类名采用大驼峰式命名方式，遵循首字母大写原则。类的名字必须由大写字母开头，单词中的其他字母均为小写。如果一个类名是由多个单词组成的，则每个单词的首字母均应为大写，如“ModelAction”；如果类名中包含单词缩写，则这个缩写词的每个字母均应为大写，如“ XMLExample”。此外，由于类是用来代表对

象的，因此在命名类时应尽量选择名词。

点亮

变量的命名方式

1. 驼峰式（camel case）

驼峰式的特点是名称中间没有空格和标点，除第一个单词外后面的单词首字母均大写。

如果第一个单词首字母大写，称为大驼峰式（upper camel case），如“GetUserName”。

如果第一个单词首字母小写，称为小驼峰式（lower camel case），如“getUserName”。

2. 蛇式（snake case）

蛇式的特点是名称中间的标点被替换成下划线“_”。

如果所有单词都大写，称之为大蛇式（upper snake case），如“GET_USER_NAME”。

如果所有单词都小写，称之为小蛇式（lower snake case），如“get_user_name”。

2.3.3 方法名、参数名和变量名

方法名采用小驼峰式命名方式，首字母小写，往后的每个单词首字母都要大写。和类名不同的是，方法名一般为动词或动词短语，与参数或参数名共同组成动宾短语，即“动词 + 名词”。一个好的函数名（即方法名）一般能通过名字直接体现该函数能实现什么样的功能，如 add()、getStudentName() 等。

除了方法名，参数名、成员变量名、局部变量名也都统一采用小驼峰式命名方式，即除第一个单词首字母小写外（若只有一个单词，就全部小写），其余单词首字母均大写。

2.3.4 对象名和属性名

对象名和属性名的命名方法相同，采用“名词或形容词 + 名词”的形式，如 name、dbClassName、dbUser、dbPassword、dbUrl 等。

2.4 常量和变量

2.4.1 常量值

常量值又称为字面常量，它是通过数据直接表示的，因此有很多种数据类型，如整型、浮点型（也叫实型）、布尔型、字符型、字符串型等。下面一一介绍这些常量值。

1. 整型常量值

整型（int）常量值默认在内存中占 32 位，是整数类型的值，当运算过程中所需值超过 32 位长度时，可

以把它表示为长整型（long）数值。长整型数值要在数字后面加上“L”或“l”，如“697L”就表示一个长整型数，它在内存中占 64 位。

Java 的整型常量值主要有如下 3 种形式。

（1）十进制数形式：如 54、−67、0。

（2）八进制数形式：Java 中的八进制数以“0”开头表示，如“0125”表示十进制数 85，“−013”表示十进制数 −11。

（3）十六进制数形式：Java 中的十六进制数以“0x”或“0X”开头，如“0x100”表示十进制数“256”，“−0x16”表示十进制数“−22”。

2. 浮点型常量值

Java 浮点型常量值默认在内存中占 64 位，是双精度型（double）的值。如果需要节省运行时的系统资源，而运算时的数据值取值范围并不大且运算精度要求不太高，可以把它表示为单精度型（float）的数值。单精度型数值一般要在该数字后面加“F”或“f”，如“69.7f”就表示一个单精度型实数，它在内存中占 32 位（取决于系统的版本高低）。

Java 浮点型常量值主要有如下两种形式。

（1）十进制数形式：由数字和小数点组成，且必须有小数点，如 12.34、−98.0。

（2）科学记数法形式：如 1.75e5、32E3，其中“e”或“E”之前必须有数字，且之后的数字必须为整数。

3. 布尔型常量值

Java 的布尔型常量值只有两个，即 true（真）和 false（假）。

4. 字符型和字符串型常量值

Java 的字符型常量值是用单引号引起来的一个字符，如 'e'、'E'。需要注意的是，Java 字符串型常量值中的单引号和双引号不可混用。双引号用来表示字符串，如 "11"、"d" 等都是表示单个字符的字符串。

除了以上所述形式的字符型常量值，Java 还允许使用一种特殊形式的字符型常量值来表示一些难以用一般字符表示的字符，这种特殊形式的字符是以“\”开头的字符序列，称为转义字符。Java 中常用的转义字符如表 2−1 所示。

表 2−1　Java 中常用的转义字符

转义字符	说明
\ddd	1 ～ 3 位八进制数所表示的 ASCII 字符或扩展字符（ddd 是八进制数）
\uxxxx	1 ～ 4 位十六进制数所表示的 Unicode 字符（xxxx 是十六进制数）
\'	单引号字符
\"	双引号字符
\\	反斜杠字符本身
\r	回车
\n	换行
\b	退格
\t	横向跳格

2.4.2 声明常量

常量不同于常量值，它可以在程序中用符号来代替常量值使用，因此在使用前必须先声明。常量与变量类似，也需要初始化，即在声明常量的同时要赋予一个初始值。常量一旦初始化就不可以被修改。Java 语言使用 final 关键字来声明常量，其语法如下所示：

```
final 数据类型 常量名 = 常量初始值 ;
```

例如：

```
final String LOVE ="java";
```

final 关键字表示最终的，可以修饰很多元素，修饰变量就可以将其变成常量。例如，以下语句使用 final 关键字声明常量。

```
public class HelloJava {
    // 静态常量
    public static final double PI = 3.14;
    // 声明成员常量
    final int y = 10;
    public static void main(String[] args) {
        // 声明局部常量
        final double x = 3.3;
    }
}
```

注意

所声明的常量若为类的成员常量，必须在声明的同时进行初始化，否则会出现编译错误。

2.4.3 变量

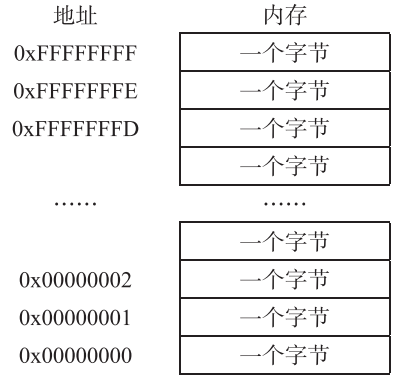


图 2-2 内存编址

在日常生活中，人们会用到大量数据，比如去超市购物、餐厅吃饭时，使用支付宝或者微信进行支付时需要扫描商家的二维码，这个过程其实就是在获取商家账号的数据。而后，输入密码，这个过程也是在获取数据，也就是密码。在后续进行的业务处理中，像钱的数额、对方的二维码账号、密码或者指纹会被反复、频繁地使用，那就需要一个存储这些数据的地方。

在软件系统开发过程中，要将系统获取的用户输入数据、计算过程中产生的中间数据，以及系统准备的一些初始数据存储在内存之中。内存单元有地址编号，而且数量极大，如图 2-2 所示。

为了方便写入数据和读出内存中的数据，需要为内存空间取一个有

意义、好记的名字，来引用内存中的数据，这种引用就是变量，可以理解为变量就是内存中数据的代词。简单来说，变量就是指在内存中开辟的，用于存放运算过程中需要用到的数据的存储空间。变量的声明如下所示：

```
int a = 5;
int b = 6;
int c = a + b;
```

在上面的代码中，变量 a、b、c 指代内存中 3 块用于存储整数的存储空间，这 3 块存储空间分别用于存储两个整数及这两个整数之和，int 意为整数数据类型，后续会详细介绍。

对于变量，我们需要关注如下几个方面。

（1）变量的声明：用特定语法声明一个变量，让运行环境为其分配空间。

（2）变量的命名：需要有个见名知义的名字，而且要符合 Java 语言规范。

（3）变量的初始化：变量声明后，要为其赋一个确定的初值后再使用。

（4）变量的访问：可以对变量中的数据进行存取及其他操作，但必须和其类型匹配。

2.4.4 声明变量

当需要使用一个变量时，必须对该变量进行声明。变量的声明包含两点——数据类型和变量名，代码如下所示：

```
int a;
```

在上面的代码中，int 为变量的数据类型，a 为变量的名称（名称要符合前面讲的变量命名规范）。当声明如上语句时，Java 虚拟机会为该变量在内存中开辟存储空间，不同的变量类型决定了存储空间的结构。

注意

Java 语言语法规定，变量使用之前必须声明，否则会有编译错误。

因变量未声明而出现编译错误的代码如下：

```
public class VarTest {
    public static void main(String[] args) {
        a = 1; // 编译错误，变量没有声明
        int score = 0; // 正确，声明同时赋值
        scord = 100; // 编译错误，变量没有声明
        System.out.println(score);
    }
}
```

在上面的代码中，可以看到两处编译错误。第一个错误是“a = 1”，因为该变量没有声明。第二个错误是“scord = 100”，因为前面声明的变量为“score”，编译器并未找到“scord”变量，该错误是由拼写错误造成的。

如果多个变量的类型一样，可以在一条语句中声明，中间使用逗号分隔，代码如下所示：

```
public static void main(String args[]) {
    int a = 1, b = 2;
    int c, d = 3;
}
```

在上面的代码中，可以看到第一条语句声明了两个整型变量，分别赋值为 1 和 2，中间使用逗号分隔，最后以“;”结尾；第二条语句声明了两个整型变量，c 没有赋初始值，d 初始值为 3。

2.5 数据类型

2.5.1 Java 基本数据类型

Java 基本数据类型包括 int（整型）、long（长整型）、double（双精度浮点型）、float（单精度浮点型）、char（字符型）、byte（字节型）、short（短整型）、boolean（布尔型）8 种，如表 2-2 所示。8 种数据类型中包括 6 种数值型（4 种整数型、2 种浮点型），1 种字符型和 1 种布尔型。

表 2-2 Java 基本数据类型

类型名称	关键字	占用内存	取值范围
整型	int	4 字节	−2147483648 ～ 2147483647
长整型	long	8 字节	−9223372036854775808L ～ 9223372036854775807L
双精度浮点型	double	8 字节	+/−1.8E+308（15 个有效位）
单精度浮点型	float	4 字节	+/−3.4E+38F（6 ～ 7 个有效位）
字符型	char	2 字节	ISO 单一字符集
字节型	byte	1 字节	−128 ～ 127
短整型	short	2 字节	−32768 ～ 32767
布尔型	boolean	1 字节	true 或 false

所有的基本数据类型的大小（所占用的字节数）都已明确规定，在各种不同的平台上保持不变，这一特性有助于提高 Java 程序的可移植性。

Java 数据类型的结构如图 2-3 所示。

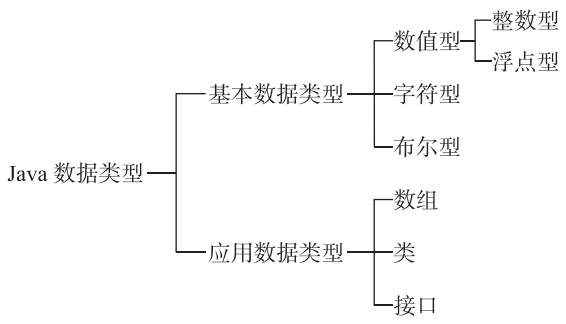


图 2-3 Java 数据类型的结构

案例 2-1 输出个人信息

请输出姓名、性别、年龄、住址等信息。

```
public class Example2_1 {
    public static void main(String[] args) {
        String BEGIN = " 大家好！下面是我的个人基本信息：";
        String END = " 谢谢！ ";
        String name;
        String sex;
        int age;
        String address;
        name = " 张三 ";
        sex = " 男 ";
        age = 23;
        address = " 越城区曲屯路 151 号 ";
        System.out.println(BEGIN);
        System.out.println(" 姓名：" + name);
        System.out.println(" 性别：" + sex);
        System.out.println(" 年龄：" + age);
        System.out.println(" 住址：" + address);
        System.out.println("----------" + END + "----------");
    }
}
```

执行结果：

```
大家好！下面是我的个人基本信息：
姓名：张三
性别：男
年龄：23
住址：越城区曲屯路 151 号
---------- 谢谢！ ----------
```

性别的赋值采用的是双引号，如果变量定位为字符串类型，赋值的无论是一个汉字，还是一个字符、一个数字，或者其他符号，都必须用双引号括起来，否则编译不会通过。单引号、双引号是有严格区别的，单引号中是字符，双引号中则是字符串，即单引号中的数据一般是 char 类型的，双引号中的数据是 String 类型的。单引号里面只能放一个字母、数字或符号，双引号里面可有 0 到多个字符。字符串需要使用 charAt(n) 方法（该方法是 String 类的一个方法），获取指定位置的字符。charAt(n) 方法返回字符串中指定索引位置“n”的字符。

2.5.2 数据类型转换

在开发过程中，有时需要对数据内置的类型进行转换，数据类型转换必须满足以下规则。

（1）不能对 boolean 类型进行类型转换。

（2）不能把对象类型转换成不相关类的对象。

（3）在把容量大的类型转换为容量小的类型时必须使用强制类型转换。

（4）转换过程中可能导致溢出或损失精度，例如：

```
int i = 128;
byte b = (byte)i;
```

因为 byte 类型能表示的最大值为 127，所以当 int 类型强制转换为 byte 类型时，值 128 就会溢出。

1. 自动类型转换

从小类型到大类型的转换称为自动类型转换，因为大类型肯定可以表示小类型的数据，所以不会出现精度丢失的情况。编译器会帮我们自动地进行自动类型转换。类型间的大小关系（由小到大）如下：

byte、short、char → int → long → float → double

在多种类型的运算中，结果会自动向较大的类型转换。且在条件运算中，结果也会向较大的数据类型转换。例如，对于语句“ int a=(5>4)?20:6.0”来说，此时 a 为 20.0，因为在条件运算中有浮点数，结果会自动向大类型转换。

下面的语句可以在 Java 中直接通过：

```
byte b;
int i = b;
long l = b;
float f = b;
double d = b;
```

如果小类型是 char 型，向大类型（整型）转换时，会转换为对应 ASCII 码值，例如：

```
char c = 'c';
int i = c;
System.out.println("output:" + i);
```

输出：

```
output:99
```

2. 强制类型转换

对于 byte、short、char 三种类型而言，它们是平级的，不能相互自动转换，但可以使用强制类型转换，格式如下：

```
(type)value
```

type 为强制转换后的数据类型，注意转换的数据类型必须是兼容的。

平级的转换：

```
short i = 99;
char c = (char)i;
```

```
System.out.println("output:" + c);
```

输出：

```
output:c
```

从大类型到小类型的转换需要强制转换，且可能出现精度丢失或者精度溢出的情况。

精度丢失指的是小类型无法准确地描述大类型的数据，尤其是浮点数，很可能小数点后面的数据就不见了。精度丢失不会四舍五入，而是直接抹去后面无法表示的内容。例如：

```
int a = (int) 20.56484615;
```

此时，a=20。

精度溢出是指大类型的数据超出了小类型的容量，此时编译器不会报错，但是结果就会不如意了。例如：

```
int a = (int)1024^4*20;
```

此时，a=0。

注意

（1）整数的默认类型是 int。

（2）小数的默认类型是 double，在声明 float 类型时必须在数字后面跟上“F”或者“f”。

案例 2-2 变量的“换装”之旅

请完成下列变量的类型转换。

```
public class Example2_2 {
    public static void main(String[] args) {
        double num = 32;
        byte b = 123;
        short s = 123;
        int num1 = 5;
        double num2 = 7.34;
        double sum = num1 + num2;
        final byte v1 = 5;
        final short v2 = 5;
        final char v3 = 97;
        final int v6 = 100;
        final byte v7 = v6;
        System.out.println("v7 = " + v7);
        final byte v8 = 127;
        float ma = 123.23f;
        byte mb = (byte)ma;
        System.out.println(mb);
```

```
            System.out.println(ma);
        }
    }
```

代码运行结果如下：

```
v7 = 100
123
123.23
```

Java 是强类型的语言，因此在参与“赋值运算”和“算数运算”时，要求参与运算的数据类型必须保持一致，否则就需要做数据类型转换。

赋值运算中低字节向高字节自动提升。

特例：把 int 类型的常量赋值给 byte、short 和 char 类型的变量或 final 修饰的常量，属于隐式类型转换的特例，赋值的数据没有超出其数据类型的表示范围即可。

对两个操作数做运算时，如果其中一个操作数为 double 类型，则另外一个操作数也会隐式转换为 double 类型，最终计算结果就是 double 类型。

2.6 运算符

2.6.1 算术运算符

表 2-3 列出了所有的 Java 算术运算符。

表 2-3 Java 算术运算符

操作符	描述
+	加法，将运算符两侧的值相加
-	减法，左操作数减去右操作数
*	乘法，将运算符两侧的值相乘
/	除法，左操作数除以右操作数
%	取余，左操作数除以右操作数所得余数
++	自增，操作数的值增加 1
--	自减，操作数的值减少 1

案例 2-3　领略 Java 算术运算符

请输出下列 4 个变量进行不同运算所得的结果。

```
public class Example2_3 {
    public static void main(String[] args) {
        int a = 10;
```

```
        int b = 20;
        int c = 25;
        int d = 25;
        System.out.println("a + b = " + (a + b) );
        System.out.println("a - b = " + (a - b) );
        System.out.println("a * b = " + (a * b) );
        System.out.println("b / a = " + (b / a) );
        System.out.println("b % a = " + (b % a) );
        System.out.println("c % a = " + (c % a) );
        System.out.println("a++ = " +  (a++) );
        System.out.println("a-- = " +  (a--) );
        // 查看 d++ 与 ++d 的不同
        System.out.println("d++ = " +  (d++) );
        System.out.println("++d = " +  (++d) );
    }
}
```

运行结果如下：

```
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++ = 10
a-- = 11
d++ = 25
++d = 27
```

自增（++）、自减（--）运算符是一种特殊的算术运算符，其他算术运算符需要两个操作数来进行运算，而自增、自减运算符只需要一个操作数。“++d”这样的前缀自增自减法是先进行自增或者自减运算，再进行表达式运算。“d++”这样的后缀自增自减法是先进行表达式运算，再进行自增或者自减运算。

案例 2-4　奇妙的加 1 和减 1

输出下列变量的自增和自减结果。

```
public class Example2_4 {
    public static void main(String[] args) {
        int a = 3; // 定义一个变量
        int b = ++a; // 自增运算
        int c = 3;
        int d = c--; // 自减运算
```

```
            System.out.println(" 进行自增运算后的值等于 " + b);
            System.out.println(" 进行自减运算后的值等于 " + d);
        }
}
```

运行结果如下：

```
进行自增运算后的值等于 4
进行自减运算后的值等于 3
```

在上述代码中，变量 b 的值来自变量 a 先进行自增运算后的值，变量 d 的值来自变量 c 还没有进行自减运算时的值。

2.6.2 关系运算符

Java 支持的关系运算符主要包括 ==、!=、>、<、>=、<=6 种，具体如表 2-4 所示。

表 2-4 Java 关系运算符

操作符	描述
==	检查两个操作数的值是否相等，如果值相等则结果为真
!=	检查两个操作数的值是否相等，如果值不相等则结果为真
>	检查左操作数的值是否大于右操作数的值，如果是那么结果为真
<	检查左操作数的值是否小于右操作数的值，如果是那么结果为真
>=	检查左操作数的值是否大于或等于右操作数的值，如果是那么结果为真
<=	检查左操作数的值是否小于或等于右操作数的值，如果是那么结果为真

案例 2-5 代码中的“是非”判官大揭秘

请输出以下 3 个变量进行关系运算的结果。

```
public class Example2_5 {
    public static void main(String[] args) {
        int a = 60;
        int b = 13;
        boolean c;
        c=a==b;
        System.out.println("a == b = " + c );
        c = a !=b;
        System.out.println("a != b = " + c );
        c = a >= b;
        System.out.println("a >= b = " + c );
        c = a > b;
        System.out.println("a > b = " + c );
```

```
            c = a <= b;
            System.out.println("a <= b = " + c );
            c = a < b;
            System.out.println("a < b  = " + c );
        }
}
```

运行结果如下：

```
a == b = false
a != b = true
a >= b = true
a > b = true
a <= b = false
a < b = false
```

2.6.3 位运算符

Java 定义了位运算符，应用于整型（int）、长整型（long）、短整型（short）、字符型（char）和字节型（byte）等数据类型。位运算是对二进制位进行操作的运算。位运算符作用在所有的位上，并且按位运算。表 2-5 列出了 Java 位运算符的基本运算。

表 2-5 Java 位运算符

操作符	描述
&	如果相对应位都是 1，则结果为 1，否则为 0
|	如果相对应位都是 0，则结果为 0，否则为 1
^	如果相对应位置相同，则结果为 0，否则为 1
~	按位取反运算符。翻转操作数的每一位，即 0 变成 1，1 变成 0
<<	按位左移运算符。左操作数按位左移右操作数指定的位数
>>	按位右移运算符。左操作数按位右移右操作数指定的位数
>>>	按位右移补零运算符。左操作数按右操作数指定的位数右移，移动得到的空位以零填充

案例 2-6 探秘计算机的 0、1 位运算

请对以下 2 个变量进行不同的位运算并输出结果。

```
public class Example2_6 {
    public static void main(String[] args) {
        int a = 10;
        int b = 3;
        System.out.println("a & b = " + (a & b));
```

```
        System.out.println("a | b = " + (a | b) );
        System.out.println("a ^ b = " + (a ^ b) );
        System.out.println("~ a = " + ~ a );
        System.out.println("a << 2 = " + (a << 2) );
        System.out.println("a >> 2 = " + (a >> 2) );
        System.out.println("a >>> 2 = " + (a >>> 2) );
    }
}
```

运行结果如下：

```
a & b = 2
a | b = 11
a ^ b = 9
~ a= -11
a << 2 = 40
a >> 2 = 2
a >>> 2 = 2
```

2.6.4 逻辑运算符

Java 语言提供了 3 种逻辑运算符，如表 2-6 所示。

表 2-6 Java 逻辑运算符

操作符	描述
&&	逻辑与运算符。当且仅当两个操作数都为真，结果才为真
||	逻辑或运算符。两个操作数中任何一个为真，结果就为真
!	逻辑非运算符，用来反转操作数的逻辑状态。如果操作数为 true，则进行逻辑非运算将得到 false

案例 2-7 代码逻辑链的“关键锁扣”

输出下面 2 个变量的 3 种逻辑运算的结果。

```
public class Example2_7 {
    public static void main(String[] args) {
        boolean a = true;
        boolean b = false;
        System.out.println("a && b = " + (a && b));
        System.out.println("a || b = " + (a || b) );
        System.out.println("!(a && b) = " + !(a && b));
    }
}
```

运行结果如下：

```
a && b = false
a || b = true
!(a && b) = true
```

2.6.5 赋值运算符

在 Java 语言中，赋值号“=”是一个运算符，称为赋值运算符。赋值运算符所在的表达式称为赋值表达式，其形式如下：

```
变量名 = 表达式;
```

赋值运算符的左侧必须是一个表达某一存储单元的变量名，赋值运算符的右侧必须是 Java 语言中合法的表达式。赋值运算的功能是先求出右侧表达式的值，然后把此值赋给赋值运算符左侧的变量。

对于赋值运算符，有以下几处注意事项。

（1）表达式“x=3”表示将 3 的值赋值给变量 x[①]，即 x 的值是 3。

（2）表达式“x=y”的作用是，将变量 y 所代表的存储单元中的内容赋给变量 x 所代表的存储单元，x 中原有的数据会被替换掉。赋值后，变量 y 中的内容保持不变。

（3）表达式“m=m+1”也是合法的赋值表达式，其作用是取变量 m 中的值加 1 后再放回到变量 m 中，即使变量 m 中的值加 1。

（4）赋值运算符的左侧只能是变量，不能是常量或表达式。“a+b=c”就是非法的赋值表达式。

（5）赋值运算符右侧的表达式也可以是一个赋值表达式。如“a=b=2+3”，按照运算符的优先级，将首先计算出 2+3 的值 5，然后按照赋值运算符自右向左的结合性，把 5 赋给变量 b，最后再把变量 b 的值赋给变量 a。

（6）当赋值运算符左侧的变量为短整型，右侧的值为长整型时，短整型变量只能接受长整型数低位上两个字节中的数据，高位上两个字节中的数据将丢失。也就是说，右侧的值不能超出短整型的数值范围，否则将得不到预期的结果。

（7）当赋值运算符左侧的变量为无符号整型，右侧的值为有符号整型时，则把内存中的内容原样复制（二进制内容直接复制）。右侧数值的范围不应超出左侧变量可以接受的数值范围。同时需要注意，这时负数将转换为无符号正整数（无符号类型会忽略符号位）。

（8）当赋值运算符左侧的变量为有符号整型，右侧的值为无符号整型时，复制的机制同上。这时若符号位为 1，将按负数处理。

Java 中常见的赋值运算符如表 2-7 所示。

表 2-7 Java 中常见的赋值运算符

操作符	描述
=	简单的赋值运算符，将右操作数的值赋给左操作数
+=	加法赋值运算符，将左操作数和右操作数相加的得数赋值给左操作数

① 为与代码格式保持一致，本书科技符号统一采用正体表示。

续表

操作符	描述
-=	减法赋值运算符，将左操作数和右操作数相减的得数赋值给左操作数
*=	乘法赋值运算符，将左操作数和右操作数相乘的得数赋值给左操作数
/=	除法赋值运算符，将左操作数和右操作数相除的得数赋值给左操作数
%=	取模赋值运算符，将左操作数和右操作数相除得到的余数赋值给左操作数
<<=	左移位赋值运算符
>>=	右移位赋值运算符
&=	按位与赋值运算符
^=	按位异或赋值运算符
|=	按位或赋值运算符

案例 2-8　变量蜕变的“神秘力量”揭秘

输出下面 3 个变量的赋值运算的结果。

```
public class Example2_8 {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int c = 0;
        c = a + b;
        System.out.println("c = a + b = " + c );
        c += a;
        System.out.println("c += a = " + c );
        c -= a;
        System.out.println("c -= a = " + c );
        c *= a;
        System.out.println("c *= a = " + c );
        a = 10;
        c = 15;
        c /= a;
        System.out.println("c /= a = " + c );
        a = 10;
        c = 15;
        c %= a;
        System.out.println("c %= a = " + c );
        c <<= 2;
        System.out.println("c <<= 2 = " + c );
```

```
            c >>= 2;
            System.out.println("c >>= 2 = " + c );
            c >>= 2;
            System.out.println("c >>= 2 = " + c );
            c &= a;
            System.out.println("c &= a = " + c );
            c ^= a;
            System.out.println("c ^= a = " + c );
            c |= a;
            System.out.println("c |= a = " + c );
        }
}
```

运行结果如下：

```
c = a + b = 30
c += a = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a = 5
c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a = 0
c ^= a = 10
c |= a = 10
```

2.6.6 条件运算符

条件运算符也被称为三元运算符、三目运算符。该运算符有 3 个操作数，并且需要判断布尔表达式的值。条件运算符主要决定哪个值应该赋值给变量，语法格式如下：

```
variable x = (expression) ? value if true : value if false
```

对于表达式“布尔表达式 ? 表达式 1: 表达式 2”来说，如果布尔表达式的值为 true，则返回表达式 1 的值，否则返回表达式 2 的值。

案例 2-9 代码岔路的“智能向导”秀

输出下面 2 个变量使用条件运算符运算的结果。

```
public class Example2_9 {
    public static void main(String[] args) {
```

```
        int a, b;
        a = 10;
        // 如果 a 等于 1 成立，则设置 b 为 20，否则为 30
        b = (a == 1) ? 20 : 30;
        System.out.println("Value of b is : " + b);
        // 如果 a 等于 10 成立，则设置 b 为 20，否则为 30
        b = (a == 10) ? 20 : 30;
        System.out.println("Value of b is : " + b);
    }
}
```

运行结果如下：

```
Value of b is : 30
Value of b is : 20
```

2.6.7 运算符优先级

表达式是符合语法规则的操作数、操作符和运算符的有效组合。每个表达式的运算结果都是一个值，表达式的值的数据类型就是表达式的类型。所有的数学运算都认为表达式是从左向右运算的，Java 语言中大部分运算符也是从左向右结合的，只有单目运算符、赋值运算符和条件运算符例外，是从右向左结合的，也就是从右向左运算。

单目运算符是编程语言中仅对单个操作数进行运算的运算符。Java 中常见的单目运算符有！、+（正号）、–（负号）、~、++、––。

乘法和加法是两个可结合的运算，也就是说，这两个运算符左右两边的操作数是可以互换位置而不影响结果的。

运算符有不同的优先级，所谓优先级就是在表达式运算中的运算顺序。一般来说，单目运算符优先级较高，赋值运算符优先级较低；算术运算符优先级较高，关系和逻辑运算符优先级较低。

Java 语言中运算符的优先级共分为 14 级，其中 1 级优先级最高，14 级最低。在同一个表达式中运算符优先级高的先执行。表 2–8 列出了所有的运算符的优先级及其结合性。

表 2–8　运算符的优先级及其结合性

优先级	操作符	结合性
1	()、[]	从左到右
2	！、+（正号）、–（负号）、~、++、––	从右到左
3	*、/、%	从左到右
4	+（加）、–（减）	从左到右
5	<<、>>、>>>	从左到右
6	<、<=、>、>=、instanceof	从左到右

续表

优先级	操作符	结合性
7	==、!=	从左到右
8	&	从左到右
9	^	从左到右
10	|	从左到右
11	&&	从左到右
12	||	从左到右
13	? :	从右到左
14	=、+=、-=、*=、/=、% =、<<=、>>=、&=、^=、|=	从右到左

案例 2-10 代码运算的“指挥交响”

输出下面 3 个变量采用不同运算符进行运算的运行结果。

```
public class Example2_10 {
    public static void main(String[] args) {
        // 声明三个整型变量 a、b、c
        int a , b , c;
        // 将变量 a、b、c 分别赋值为 4、5、6
        a = 4; b = 5; c =6;
        // 计算 (b / a) + c 的值 ， 并赋值给变量 m
        double m;
        m = ((double)b / a ) + c;
        // 计算 (c % b) * a - c 的值，并赋值给变量 n
        int n;
        n = ( c % b ) * a - c;
        // 输出 m 和 n 的值
        System.out.println(m); // 输出 7.25
        System.out.println(n); // 输出 -2
    }
}
```

运行结果如下：

```
7.25
-2
```

课堂练习 2-1 下面代码执行完成后 c 的值是多少？

```
public class Exercise2_1 {
```

```
    public static void main(String[] args) {
        int  a = 5;
        int  b = 4;
        int  c = a++ - b++ / b-- >> 2 % a--;
        System.out.println(c);
    }
}
```

运行结果如下：

```
1
```

课堂练习 2-2　下面代码执行完成后 a 和 b 的值分别是多少？

```
public class Exercise2_2 {
    public static void main(String[] args){
        int a = 10,b = 6;
        System.out.println(" 改变之前：a = " + a + ",b = " + b);
        a -= b++;
        System.out.println(" 改变之后：a = " + a + ",b = " + b);
    }
}
```

运行结果如下：

```
改变之前：a = 10,b = 6
改变之后：a = 4,b = 7
```

2.7 程序基本输入输出操作

2.7.1 控制台输出

Java 中常用 println() 方法输出数据到控制台并换行。该方法由 System 类的 PrintStream 型静态常量 out 调用执行。如果把 println() 改成 print()，则后续的内容会紧接在这一次的输出之后输出，即不换行。也可用 printf(" 输出格式 ", 输出数据列表) 方法实现增强的格式化输出，例如：

```
System.out.printf("%s%.1f\n"," 该圆的面积为：",area);
```

其中，“ %s”表示格式化输出字符串，“ %.1f”表示格式化输出浮点数，小数点后位数为 1，“ \n”表示输出后换行。不同的格式转换符对应不同的输出结果，如表 2-9 所示。

表 2-9 格式转换符

格式转换符	类型	使用方法	输出结果
d	十进制整数	("%d",10)	10
x	十六进制整数	("%x",10)	A
o	八进制整数	("%o",10)	12
f	定点浮点数	("%f",100f)	100.000000
e	指数浮点数	("%e",100f)	1.000000e+02
g	通用浮点数	("%g",100f)	100.000
a	十六进制浮点数	("%a",100f)	0x1.9p6
s	字符串	("%s",10)	10
c	字符	("%c",'1')	1
b	布尔值	("%b",10)	true
h	散列码	("%h",10)	a
%	百分号	("%.2f%%",2/7f)	0.29%

案例 2-11 程序心声的“传声筒”

编写程序演示不同控制台输出方法。

```
public class Example2_11 {
    public static void main(String[] args) {
        System.out.println("Hello Java"); // 输出后换行
        System.out.print("Hello Java\n"); // 与 println() 方法效果相同
        System.out.print("Hello Java"); // 输出不换行
    }
}
```

运行结果如下：

```
Hello Java
Hello Java
Hello Java
```

2.7.2 控制台输入

在程序开发过程中，用户时常需要对程序代码中某个或某些变量重新赋值，如果每次重新赋值都要停止程序运行，就会影响开发效率。这里介绍一个 Scanner 类的获取用户从键盘输入的数据的方法，格式如下：

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

其中，“scanner”表示 Scanner 类的一个对象，“scanner”使用多种方法来读取不同类型的数据。

Scanner 类是用于扫描输入文本的。Scanner 类位于 java.util 包中。如果要使用 Scanner 类，则必须使用 import 语句导入 Scanner 类，否则会出现“Scanner cannot be resolved to a type”异常。

案例 2-12　代码与用户的“对话桥梁”

从键盘上输入不同类型的数据给变量，并将这些结果输出。

```
import java.util.Scanner; // 导入 java.util 包中的 Scanner 类
public class Example2_12 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String str1 = input.next(); // 获取字符串
        System.out.println(str1); // 输出字符串 str1
        int number1 = input.nextInt(); // 获取整数
        System.out.println(number1); // 输出整型变量
        float number2 = input.nextFloat(); // 获取浮点数据
        System.out.println(number2); // 输出浮点型变量
    }
}
```

运行结果如下：

```
abc123
abc123
3
3
3.1415926
3.1415926
```

通常情况下，我们通过 Scanner 类的 next() 与 nextLine() 方法获取输入的字符串，在读取前我们一般使用 hasNext() 与 hasNextLine() 方法判断是否还有输入的数据，这样操作比较安全，能提高程序的健壮性。next() 与 nextLine() 的区别如下。

next()：

（1）一定要读取到有效字符后才可以结束输入；

（2）对输入有效字符之前遇到的空白，next() 方法会自动将其去掉；

（3）只有输入有效字符后才将其后面输入的空白作为分隔符或者结束符；

（4）不能得到带有空格的字符串。

nextLine()：

（1）以 Enter 为结束符，也就是说 nextLine() 方法返回的是输入回车之前的所有字符；

（2）可以获取空白。

综合实训：灵活掌握条件表达式

实训案例

近年来，国家电网建设取得了巨大的成就，提升了人民生活水平，但是节约用电依然是每一个公民应尽的责任和义务。假设家庭用电的费用是根据每月的用电量来计算的，电费的计算规则如下。

（1）用电量不超过 200 度：每度电收费 0.5 元。

（2）用电量超过 200 度但不超过 400 度：前 200 度按 0.5 元 / 度收费，超过部分按 0.75 元 / 度收费。

（3）用电量超过 400 度：前 200 度按 0.5 元 / 度收费，接下来的 200 度按 0.75 元 / 度收费，超过 400 度的部分按 1.0 元 / 度收费。

请编写一个程序，根据用户输入的用电量（单位：度）计算并输出当月的电费。

思路解析

基本数据类型：使用 int 来表示用电量（度），使用 double 来表示电费（元）。

条件判断：需要根据不同的用电量区间来计算电费，使用条件运算符进行条件判断。

加法计算：根据用电量和收费标准，逐步叠加总电费。

代码实现

```
import java.util.Scanner;
public class Training {
    public static void main(String[] args) {
        // 创建 Scanner 对象以便输入数据
        Scanner scanner = new Scanner(System.in);

        // 提示用户输入用电量
        System.out.print(" 请输入本月用电量（度）：");
        int usage = scanner.nextInt(); // 读取用电量

        // 计算电费：使用条件运算符
        double bill =
            (usage <= 200) ? (usage * 0.5) : // 如果用电量不超过 200 度
            (usage <= 400) ? (200 * 0.5 + (usage − 200) * 0.75) : // 如果用电量超过 200 度但不超过 400 度
            (200 * 0.5 + 200 * 0.75 + (usage − 400) * 1.0); // 如果用电量超过 400 度

        // 输出结果
        System.out.println(" 本月用电量：" + usage + " 度 ");
        System.out.println(" 本月电费：" + bill + " 元 ");

        // 关闭扫描器
        scanner.close();
```

```
        }
    }
```

拓展训练

请扫描下方二维码查看拓展训练内容。

拓展训练 2