目 录

CONTENTS

单元一　Java 程序设计基础 / 001

任务 1.1　Java 的基本语法的使用 / 002

任务 1.2　Java 中的常量和变量的使用 / 005

任务 1.3　Java 中的运算符的使用 / 011

任务 1.4　流程控制语句的使用 / 017

任务 1.5　数组的使用 / 026

思考与练习 / 034

单元二　Java 面向对象编程 / 039

任务 2.1　类与对象的创建与使用 / 040

任务 2.2　类的封装使用 / 044

任务 2.3　类的继承和多态的使用 / 054

任务 2.4　异常的使用 / 073

任务 2.5　常用类的使用 / 081

思考与练习 / 101

单元三　集合 / 107

任务 3.1　Collection 接口和 List 接口的使用 / 108

任务 3.2　Set 接口的使用 / 117

任务 3.3　Map 接口的使用 / 123

任务 3.4　常用工具类的使用 / 129

思考与练习 / 135

单元四　I/O 流 / 139

任务 4.1　字节流和字符流的使用 / 140

任务 4.2　File 类的使用 / 153

思考与练习 / 160

单元五　JDBC 的使用 / 163

任务 5.1　JDBC 常用 API 的使用 / 164

任务 5.2　JDBC 编程的实现 / 164

思考与练习 / 171

单元六　线程的使用 / 175

任务 6.1　线程的使用 / 176

任务 6.2　线程的调度 / 179

任务 6.3　多线程的同步 / 182

思考与练习 / 186

单元七　校园订餐系统的立项和需求分析 / 189

任务 7.1　项目的立项和规划 / 190

任务 7.2　项目的需求分析 / 190

思考与练习 / 201

单元八　校园订餐系统的软件设计 / 203

任务 8.1　软件的概要设计 / 204

任务 8.2　软件的详细设计 / 212

思考与练习 / 225

单元九　校园订餐系统的软件实现 / 227

任务 9.1　项目环境的搭建和前期准备 / 228

任务 9.2　用户功能模块的实现 / 233

任务 9.3　管理员功能模块的实现 / 249

思考与练习 / 256

单元十　校园订餐系统的测试与总结 / 259

任务 10.1　项目的测试 / 260

任务 10.2　项目的总结 / 273

思考与练习 / 277

参考文献 / 278

单元一

Java 程序设计基础

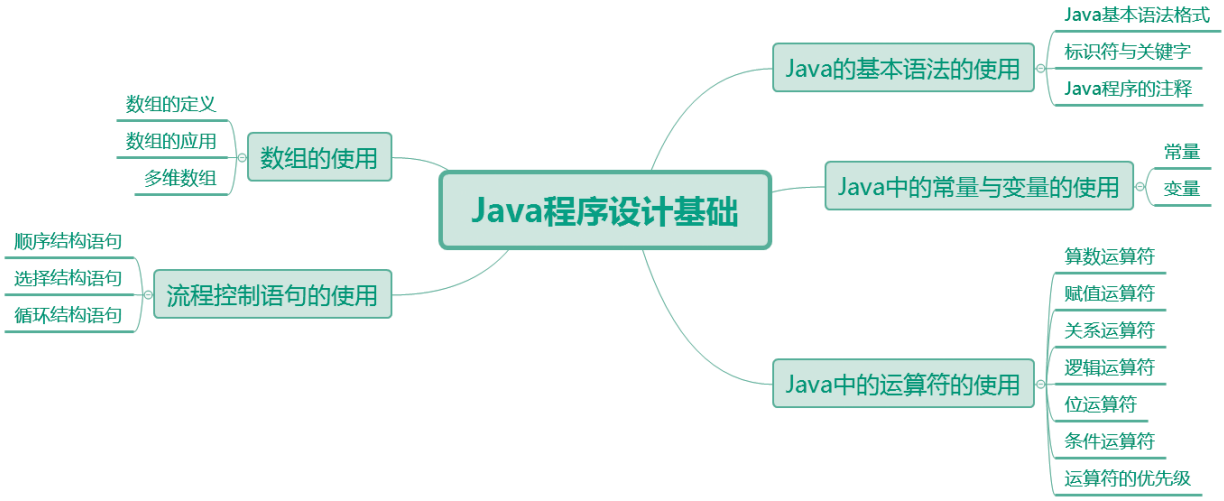
单元目标

- 了解 Java 的基本语法格式
- 熟悉 Java 语言中的变量与常量
- 掌握 Java 语言中运算符的使用
- 掌握 Java 结构语句的使用
- 掌握 Java 中数组的定义与使用

思政目标

1. 培养学生社会主义核心价值观
2. 培养学生勇攀科学高峰的责任感和使命感
3. 培养学生科技报国的家国情怀和使命担当

知识脉络图



任务 1.1

Java 的基本语法的使用

每一种编程语言都有一套自己的语法规范，Java 语言也不例外，同样需要遵从一定的语法规范，如代码的书写、程序的注释、标识符的定义、关键字的应用等。因此要学好 Java 语言，首先需要熟悉它的基本语法。

1.1.1 Java 基本语法格式

Java 基本语法格式：

```
[ 修饰符 ] class 类名 {
    程序代码
}
```

【例 1-1】HelloWorld.java

示例代码如下：

```
public class HelloWorld{
       public static void main(String[] args){
         System.out.println("Hello World!!!");
   }
}
```

语句解释说明：

（1）class 是一个关键字，用于定义一个类。在 Java 中，类就是一个程序的基本单元，所有的代码都需要在类中书写。

（2）HelloWorld 是类的名称，简称类名。class 关键字与类名之间需要用空格、制表符、换行符等任意的空白字符进行间隔。类名之后要写一对大括号，它定义了当前这个类的管辖范围。

（3）public static void main(String[] args){} 定义了一个 main 方法。该方法是 Java 程序的执行入口，程序将从 main() 方法所属大括号内的代码开始执行。

（4）在 main() 方法中编写了一条执行语句“System.out.println("Hello World!!!");”，它的作用是打印一段文本信息。

现在把【例 1-1】代码做一些修改，去掉 class 前面的 public，再次运行程序，仍然能够得到想要的结果。这说明程序没有错误，那么为什么还要使用 public 呢？这就是一种编码规范。可见不遵守编码规范的代码并不是错误的代码，但是一段好的代码不仅能够完成某项功能不应该遵守相应的编码规范，从开始就注意按照编码规范编写代码，这是成为一名优秀程序员的基本条件。请对照上面的代码并记住以下编码规范。

- 类名必须使用 public 修饰。
- 一行只写一条语句。
- 每条语句的结尾都必须用英文分号 (;) 结束。
- 用 { } 括起来的部分通常表示程序的某一层次结构。“{”一般放在这一结构开始行的最末，“}”

与该结构的第一个字母对齐，并单独占一行。

❑ 低一层次的语句或注释应该比高一层次的语句或注释缩进若干个空格后再书写，使程序更加清晰，增加程序的可读性。

❑ 大小写敏感：Java 是大小写敏感的，这就意味着标识符 Hello 与 hello 是不同的。

❑ 类名：对于所有的类来说，类名的首字母应该大写。如果类名由若干单词组成，那么每个单词的首字母应该大写，例如：MyFirstJavaClass 。

❑ 方法名：所有的方法名都应该以小写字母开头。如果方法名含有若干单词，则后面的每个单词首字母大写，例如：方法 callName( )。

❑ 源文件名：源文件名必须和类名相同。当保存文件的时候，你应该使用类名作为文件名保存（切记 Java 是大小写敏感的），文件名的后缀为 .java。（如果文件名和类名不相同则会导致编译错误）。

❑ 主方法入口：所有的 Java 程序由 public static void main(String args[]) 方法开始执行。

⚠ **注意：**在程序中所有的标点符号都是英文标点，千万不要在中文状态下输入，否则编译器会报告“Invalid Character（无效字符）”错误信息。

1.1.2　标识符与关键字

1. 标识符

在程序设计语言中存在的任何一个成分（如变量、常量、属性、方法、类、接口等）需要有一个名字表示它的存在和唯一性，这个名字就是标识符。也可以说，程序员对程序中的每个成分加以命名时使用的命名记号称为标识符（identifier）。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。Java 语言中，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。

（3）不能是关键字。

例如，下面的这些标识符都是合法的：

```
username
username123
userName
$username
Class
```

注意，下面的这些标识符都是不合法的。

```
123username      //不能以数字开头
Hello World      //不能包含空格特殊符号
class            //不能是关键字
```

在 Java 程序中定义的标识符必须要严格遵守上面列出的规范，否则程序在编译时会报错。除了上面列出的规范，为了增强代码的可读性，建议初学者在定义标识符时还应该遵循以下规则。

❑ 包名所有字母一律小写。例如：cn.test。

❑ 类名和接口名每个单词的首字母都要大写。如：ArrayList、Iterator。

❑ 常量名所有字母都大写，单词之间用下划线连接。例如：DAY_OF_MONTH、PI。

❑ 变量名和方法名的第一个单词首字母小写，从第二个单词开始每个单词首字母大写。例如：lineNumber、getLineNumber。

❑ 在程序中，应该尽量使用有意义的英文单词来定义标识符，使得程序便于阅读。例如使用 userName 表示用户名，password 表示密码。

2. 关键字

关键字 (keyword) 具有专门的意义和用途，不能当作一般的标识符使用。和其他语言一样，Java 中预留了许多关键字，如 class、public 等。下面列举的是 Java 中的 50 个关键字，见表 1–1。

表 1–1　Java 关键字列表

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

使用 Java 关键字时，有几个值得注意的地方。

❑ 所有的关键字都是小写的。

❑ 程序中的标识符不能以关键字命名。

❑ const 和 goto 是保留字关键字，虽然在 Java 中还没有任何意义，但在程序中不能用来作为自定义的标识符。

1.1.3　Java 程序的注释

在编写程序时，为了方便程序的阅读、使代码易于理解，通常会在实现功能的同时为代码加一些说明性的文字，这就是代码的注释。注释只在 Java 源文件中有效，在编译程序时，编译器并不处理这些注释。Java 中的注释有三种类型。

1. 单行注释

单行注释通常用于对程序中的某一行代码进行解释，用符号“//”表示，“//”后面为注解说明的内容。

【例 1–2】单行注释

```
public class HelloWorld{
    public static void main(String[] args){
        // 输出消息到控制台
```

```
            System.out.println("Hello World!!!");
        }
    }
```

2. 多行注释

多行注释以“/*”开头，以“*/”结尾，在“/*”和“*/”之间的内容都被看作注释。

【例 1-3】多行注释

```
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("Hello  World!!!");
        /*
        System.out.println("Hello  World!!!")
        System.out.println("Hello  World!!!");
        */
    }
}
```

3. 文档注释

文档注释：以“/**”开头，并在注释内容末尾以“*/”结束。通常在一个源文件开始之前，编写注释对整个文件做一些说明，包括文件的名称、功能、作者、创建日期等。文档注释是对一段代码概括性的解释，说明可以使用 javadoc 命令将文档注释提取出来生成帮助文档。

【例 1-4】文档注释

```
/**
 * HelloWorld.java
 * 2023-03-21
 * 第一个 Java 程序
 */
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("Hello  World!!!");
    }
}
```

任务 1.2

Java 中的常量和变量的使用

1.2.1　常量

常量代表程序运行过程中不能改变的值，如圆周率等。定义常量时候，需要使用 final。常量名需要大写，多个单词之间使用“_”分割。定义常量语法如下：

```
final 数据类型 常量名 = 值;
```

例如：

```
final double PI  =  3.14 ;                          // 定义表示圆周率的常量
final char MALE  =  'M',  FEMALE  =  'F' ;        // 定义表示性别的常量
final int STYDENT_MAX_NUM  =  50 ;             // 定义表示最多学生人数的常量
```

1.2.2 变量

1. 变量的定义

在程序运行期间，随时可能产生一些临时数据，应用程序会将这些数据保存在一些内存单元中，每个内存单元都用一个标识符来标识。这些内存单元被称为变量，定义的标识符就是变量名，内存单元中存储的数据就是变量的值。

变量基本语法格式：

```
变量类型 变量名 [= 初始值 ];
```

例如：

```
int x = 0, y;
y = x + 4;
```

2. 变量的数据类型

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。

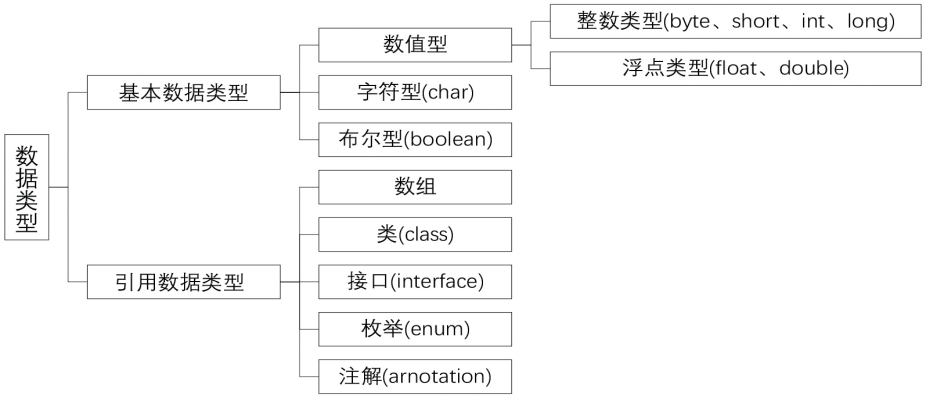


图 1–1　Java 数据类型

8 种基本数据类型是 Java 语言内嵌的，在任何操作系统中都具有相同大小和属性，而引用数据类型是在 Java 程序中由编程人员自己定义的变量类型。

Java 语言的基本数据类型有整型、浮点型、逻辑型和字符型，共 4 类 8 种基本数据类型。

（1）整型变量：用来存储整数数值，即没有小数部分的值。在 Java 中，为了给不同大小范围内的整数合理地分配存储空间，整数类型分为 4 种不同的类型：字节型（byte）、短整型（short）、整型（int）和长整型（long）。4 种类型所占存储空间的大小以及取值范围如表 1–2 所示。

表 1–2　Java 语言的整数类型

类　型	占用空间	范　围
byte（字节型）	8 位（1 个字节）	–128 ~ 127，即 -2^7 ~ 2^7-1

类　型	占用空间	范　围
short（短整型）	16 位（2 个字节）	−32 768 ~ 32 767，即 -2^{15} ~ $2^{15}-1$
int（整型）	32 位（4 个字节）	−2 147 483 648 ~ 2 147 483 647，即 -2^{31} ~ $2^{31}-1$
long（长整型）	64 位（8 个字节）	−9 223 372 036 854 775 808 ~ 9 223 372 036 854 775 807，即 -2^{63} ~ $2^{63}-1$

⚠ 注意：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。所以若声明 long 型变量的值超过 int 型的取值范围时，如果数的后面不加 l 或 L，系统会认为是 int 型而出错。

```
long x =2200000000L;          // 所赋值超过 int 型的取值范围，后面必须加上字母 L
long y =5732;                 // 所赋值未超过 int 型的取值范围，后面可省略字母 L
```

（2）浮点型变量：用来存储小数数值，Java 语言的浮点型有单精度浮点（float）和双精度浮点（double）两种。它们所占存储空间的大小以及取值范围如表 1–3 所示。

表 1–3　Java 语言的浮点数类型

类　型	数据位	范　围
float(单精度浮点)	32 位 （4 个字节）	负数范围：−3.4E+38 ~ −1.4E−45 正数范围：1.4E−45 ~ 3.4E+38
double(双精度浮点)	64 位 （8 个字节）	负数范围：−1.79E+308 ~ −4_9E−324 正数范围：4.9E−324 ~ 1.79E+308

⚠ 注意：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，所以若声明 float 型变量时如果数的后面不加 f 或 F，系统会认为是 double 型而出错。

```
float  x =123.54f;            //Java 中默认将小数视为 double 型，需在小数后加 F
```

（3）布尔型变量：用来存储布尔值，在 Java 中用 boolean 表示，该类型的变量只有两个值，即 true 和 false。其中，true 代表“真”，false 代表“假”，true 和 false 不能转换成数字表示形式。

```
boolean flag= false;      // 声明一个 boolean 类型的变量，初始值为 false
flag = true;              // 改变 flag 变量的值为 true
```

（4）字符型变量：用于存储一个单字符，在 Java 中用 char 表示。Java 中每个 char 类型字符变量都占 2 个字节。在给 char 类型的变量赋值时，需要用一对英文半角格式的单引号把字符括起来，如 'a'，也可以将 char 类型的变量赋值为 0~65535 范围内的整数，计算机会自动将这些整数转化为所对应的字符，如数值 97 对应的字符为 'a'。

```
char c='a';                   // 为一个 char 类型的变量赋值字符 a
char ch= 97;                  // 为一个 char 类型的变量赋值整数 97，相当于赋值字符 a
```

【例 1–5】输出 Java 课考试最高分：98.5。输出最高分学员姓名：张三。输出最高分学员性别：男。

```
public class Demo1_5 {
        public static void main(String[] args){
            double score=98.5;
            String name=" 张三 ";    // 字符串用双引号 ""
            char sex=' 男 ';        // 字符用单引号 ''
```

```
            System.out.println(" 本次考试成绩最高分："+score);
            System.out.println(" 最高分得主："+name);
            System.out.println(" 性别 "+sex);
        }
}
```

3. 变量的类型转换

在程序中，当把一种数据类型的值赋给另一种数据类型的变量时，需要进行数据类型转换。根据转换方式的不同，数据类型转换可分为两种：自动类型转换和强制类型转换。

（1）自动类型转换。

自动类型转换也叫隐式类型转换，指的是两种数据类型在转换过程中不需要显式地进行声明。要实现自动类型转换，必须同时满足两个条件，一是两种数据类型彼此兼容，二是目标类型的取值范围大于源类型的取值范围。

在 Java 中，将取值范围小的数据类型的变量值赋值给取值范围大的数据类型的变量时，程序也不会出现任何问题。Java 中支持的不同数据类型之间的自动转换如图 1–2 所示。

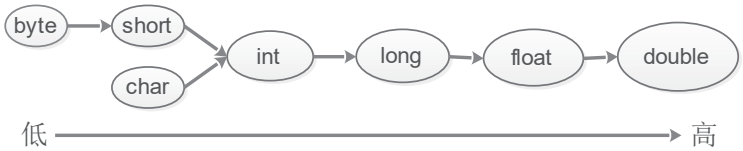


图 1–2　自动类型转换图

从上图可以看出，Java 中取值范围小的 byte、short、char 等类型数据都可以自动转换为取值范围大的数据类型 (如 int 类型)，并最终都可以自动转换为双精度浮点数类型。例如：

```
int a = 1;
double b = a;            // 程序把 int 类型的变量 a 转换成了 double 类型，无需特殊说明
byte c=3;
int d = c;               // 程序把 byte 类型的变量 c 转换成了 int 类型，无需特殊说明
```

上面的语句中，把 int 类型的变量 a 转换成了 double 类型，将 byte 类型的变量 c 的值赋给 int 类型的变量 d，由于 int 类型的取值范围大于 byte 类型的取值范围，编译器在赋值过程中不会造成数据丢失，所以编译器能够自动完成这种转换，在编译时不报告任何错误。

（2）强制类型转换。

强制类型转换也叫显式类型转换，指的是两种数据类型之间的转换需要显式地声明。当两种类型彼此不兼容，或者目标类型取值范围小于源类型时，自动类型转换无法进行，这时就需要进行强制类型转换。

将取值范围大的数据类型的变量值赋值给取值范围小的数据类型的变量时，就可能造成数据的丢失，所以系统默认不支持这种行为，只能由开发者自己决定是否进行强制类型转换。

接下来先演示一个错误类型转换的例子，如例 1–6 所示。

【例 1–6】强制类型转换

```
public class Demo1_6{
    public static void main(String[] args){
        int num =4;
        byte b =num ;
```

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

程序编译时报错，结果如图 1–3 所示。

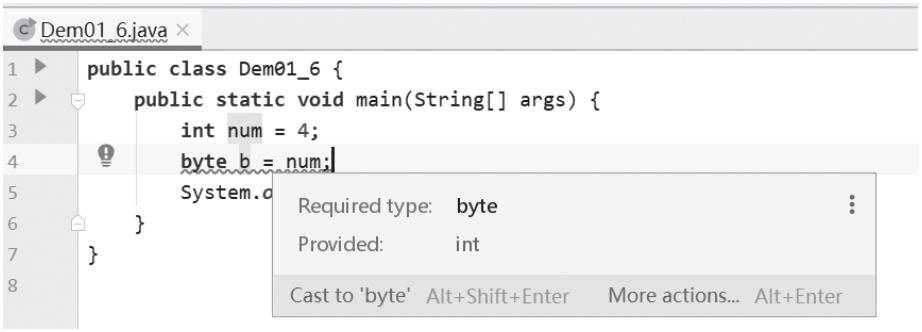


图 1–3　报错运行结果

在进行 Java 代码编写时，IntelliJ IDEA 开发工具会自动对已编写的代码进行检测，如果发现问题，它会以红色波浪线和红叉的形式进行提醒。将鼠标悬停在出现红色波浪线错误的位置时，会出现一个悬浮框，悬浮框内将提示错误信息以及快速解决方案。

从图 1–3 可以看出，程序编译过程中出现了类型转换异常，出现这样错误的原因是将一个 int 型的值赋给 byte 类型的变量 b 时，int 类型的取值范围大于 byte 类型的取值范围，这样的赋值可能会导致数值溢出，也就是说一个字节的变量无法存储 4 个字节的整数值。

在这种情况下，就需要进行强制类型转换，其语法格式如下：

```
目标类型 变量名 = ( 目标类型 ) 值 ;
```

将【例 1–6】中的第 4 行代码修改为下面的代码：

```
byte b = (byte) num;
```

修改后保存源文件程序将不再报错，程序的运行结果如下。

```
4
```

需要注意的是，在对变量进行强制类型转换时，会发生取值范围较大的数据类型向取值范围较小的数据类型的转换情况，如将一个 int 类型的数转为 byte 类型，这样做极容易造成数据精度的丢失。接下来通过一个案例来说明，如例 1–7 所示。

【例 1–7】强制类型转换 2

```
public class Demo1_7{
    public static void main(String[] args) {
        byte a; // 定义 byte 类型的变量 a
        int b =298; // 定义 int 类型的变量 b，其表现形式是十六进制
        a = (byte) b;
        System.out.println("b = " +b);
        System.out.println("a = "+a);
    }
}
```

运行结果如下所示。

```
b = 298
a = 42
```

例 1-7 中，第 5 行代码进行了强制类型转换，将一个 int 类型的变量 b 强制转换成 byte 类型，然后再将强转后的结果赋值给 byte 类型的变量 a。从运行结果可以看出，变量 b 本身的值为 298，然而在赋值给变量 a 后，变量 a 的值却为 42，这说明在强制转换过程中丢失了精度。出现这种现象的原因是，变量 b 为 int 类型，在内存中占用 4 个字节，而 byte 类型的数据在内存中占用 1 个字节。当将变量 b 的类型强转为 byte 类型后，前面 3 个高位字节的数据已经丢失，这样数值就发生了改变。int 类型转 byte 类型的过程如图 1-4 所示。

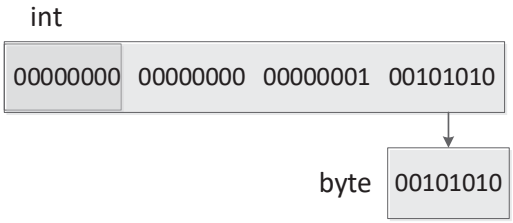


图 1-4　int 类型变量强制转换为 byte 类型

4. 变量的作用域

变量需要先定义后使用，但这并不意味着在变量定义之后的语句中一定可以使用该变量。变量需要在它的作用范围内才可以被使用，这个作用范围称为变量的作用域。在程序中，变量一定会被定义在某一对大括号中，该大括号所包含的代码区域便是这个变量的作用域。接下来通过一个代码片段来分析变量的作用域，具体如下：

```
public static void main(String[] args) {
    int x=4;
     {
        int y=9;
        ......
     }
        ......
}
```

y的作用域

x 的作用域

上面的代码中，有两层大括号。其中，外层大括号所标识的代码区域就是变量 x 的作用域，内层大括号所标识的代码区域就是变量 y 的作用域。变量的作用域在编程中尤为重要，接下来通过一个案例进一步熟悉变量的作用域，如例 1-8 所示。

【例 1-8】 变量的作用域

```
public class Demo1_8 {
    public static void main(String[] args) {
        int x = 3;                      // 定义了变量 x
        {
        int y = 5; // 定义了变量 y
        System.out.println("x =" + x); // 访问变量 x
        System.out.println("y =" + y); // 访问变量 y
        }
        y = x;                          // 访问变量 x，为变量 y 赋值
        System.out.println("x =" + x); // 访问变量 x
    }
}
```

编译程序报错，结果如图 1–5 所示。

```
Dem01_8.java
1  public class Dem01_8 {
2      public static void main(String[] args) {
3          int x = 3;                    // 定义了变量x
4          {
5              int y = 5; // 定义了变量y
6              System.out.println("x =" + x); // 访问变量x
7              System.out.println("y =" + y); // 访问变量y
8          }
9          y = x;                        // 访问变量x，为变量y赋值
10         S
11     }
12 }
13
Cannot resolve symbol 'y'
Create local variable 'y'  Alt+Shift+Enter    More actions...  Alt+Enter
```

图 1–5　程序编译报错

图 1–5 出现“y 不能被解析为一个变量”的错误。出错的原因在于，在给变量 y 赋值时超出了它的作用域。例 1–5 中的第 9 行代码中“y=x;”语句去掉，保存后文件将不再报错，程序的运行结果如下所示。

```
x =3
y =5
x =3
```

变量 x、y 都在各自的作用域中，因此都可以被访问到。

任务 1.3

Java 中的运算符的使用

1.3.1　算数运算符

在数学运算中最常见的就是加减乘除，被称作四则运算。Java 中的算数运算符就是用来处理四则运算的符号。Java 算数运算符如表 1–4 所示。

表 1–4　算数运算符

运 算 符	功 能	示 例
+	正值	+a
–	负值	–a
+	加运算	a+b
–	减运算	a–b
*	乘运算	a*b
/	除运算	a/b
%	取模（求余）运算	a%b

运 算 符	功 能	示 例
++	加 1（前）	++a
--	减 1（前）	--a
++	加 1（后）	a++
--	减（后）	a--

示例代码如下：

```
int a=4,b=3,c,d,e,f,g;
c=a+b;     //两个整数相加，则 c=7
d=a-b;     //两个整数相减，则 d=1
e=a*b;     //两个整数相乘，则 e=12
f=a/b;     //两个整数做除法，只保留整数部分而舍弃小数部分，则 f=1
g=a%b;     //两个数取模，则 g=1
-----------------------------------------------------------------------------
int i = 10,j,k,m,n,p,q;
j = +i;    //取原值，则 j=10
k = -i;    //取相反符号值，则 k = -10
m = i++;   //先 m = i，再 i = i+1，则 m = 10,i=11
n = ++i;   //先 i=i+1，再 n = i，则 i=12,n = 12
p = i--;   //先 p=i，再 i = i-1，则 p = 12,i=11
q = --i;   //先 i=i-1，再 q=i，则 i=10,q=10
```

⚠ 注意：

（1）在进行除法运算时，当除数和被除数都为整数时，得到的结果也是一个整数。如果除法运算有小数参与，得到的结果会是一个小数。

（2）在进行取模（%）运算时，运算结果的正负取决于被模数（% 左边的数）的符号，与模数（% 右边的数）的符号无关。如 :(–5) % 3=–2，而 5%(–3) = 2。

1.3.2 赋值运算符

赋值运算符的作用就是将常量、变量或表达式的值赋值给某一个变量。Java 赋值运算符如表 1–5 所示。

表 1–5 赋值运算符

运 算 符	功能	示 例	等效表达式
=	赋值	a=b	a= b
+=	加等于	a+=b	a=a+b
–=	减等于	a–=b	a=a–b
*=	乘等于	a*=b	a=a*b
/=	除等于	a/=b	a=a/b
%=	模等于	a%=b	a=a%b

示例代码如下：

```
int x,y;
x=y=3;     // 为两个数赋值，则 x=3,y=3
y+=x;      // 等效于 y=y+x，则 x=3,y=6
y-=x;      // 等效于 y=y-x，则 x=3,y=3
y*=x;      // 等效于 y=y*x，则 x=3,y=9
y/=x;      // 等效于 y=y/x，则 x=3,y=3
y%=x;     // 等效于 y=y%x，则 x=3,y=1
```

1.3.3　关系运算符

关系运算符用于比较两个值之间的大小，结果返回逻辑型值 true 或 false，关系运算符都是二元运算符，如表 1–6 所示。

表 1–6　关系运算符

运 算 符	功 能	示 例
>	大于	a>b
>=	大于或等于	a>=b
<	小于	a<b
<=	小于或等于	a<=b
==	等于	a==b
!=	不等于	a!=b

示例代码如下：

```
int x=3,y=4;
boolean z;
z=x>y;      //x=3,y=4，3 大于 4，结果为 false，则 z=false
z=x>=y;     //x=3,y=4，3 大于或等于 4，结果为 false，则 z=false
z=x<y;     //x=3,y=4，3 小于 4，结果为 true，则 z=true
z=x<=y;     //x=3,y=4，3 小于或等于 4，结果为 true，则 z=true
z=x==y;     //x=3,y=4，3 等于 4，结果为 false，则 z=false
z=x!=y;     //x=3,y=4，3 不等于 4，结果为 true，则 z=true
```

⚠ **注意：**不能在浮点数之间作“==”的比较，因为浮点数在表达上有难以避免的微小误差，精确的相等比较无法达到，所以这类比较没有意义。比较运算符在使用时需要注意一个问题，不能将比较运算符“==”误写成赋值运算符“=”。

1.3.4　逻辑运算符

逻辑运算与关系运算的关系非常密切，关系运算是运算结果为逻辑型量的运算，而逻辑运算是操作数与运算结果都是逻辑型量的运算。逻辑运算符如表 1–7 所示。

表 1-7 逻辑运算符

运算符	功 能	示 例	运 算 规 则
&	逻辑与	a&b	两个操作数均为 true 时，结果才为 true
|	逻辑或	a|b	两个操作数均为 false 时，结果才为 false
!	逻辑非（取反）	!a	将操作数取反
^	异或	a^b	两个操作数同真或同假时，结果才为 false
&&	短路与	a&&b	两个操作数均为 true 时，结果才为 true
||	短路或	a||b	两个操作数均为 false 时，结果才为 false

示例代码如下：

```
boolean a,b,c;
a=true;
b=false;
z=a&b;       //a 为 true，b 为 false，a 与 b 为 false，则 z=false
z=a|b;        //a 为 true，b 为 false，a 或 b 为 true，则 z=true
z=! a;        //a 为 true，a 取反为 false，则 z=false
z=a^b      //a 为 true，b 为 false，a 异或 b 为 false，则 z=false
z=a&&b;       //a 为 true，b 为 false，a 短路与 b 为 false，则 z=false
z=a||b;        //a 为 true，b 为 false，a 短路或 b 为 true，则 z=true
```

注意：

（1）逻辑运算符可以针对结果为布尔值的表达式进行运算。如：x > 3 && y != 0。

（2）运算符“&”和“&&”都表示与操作，当且仅当运算符两边的操作数都为 true 时，其结果才为 true，否则结果为 false。当运算符“&”和“&&”的右边为表达式时，两者在使用上还有一定的区别。在使用“&”进行运算时，不论左边为 true 或者 false，右边的表达式都会进行运算。如果使用“&&”进行运算，当左边为 false 时，右边的表达式不会进行运算，因此“&&”被称作短路与。

（3）运算符“|”和“||”都表示或操作，当运算符两边的操作数任何一边的值为 true 时，其结果为 true，当两边的值都为 false 时，其结果才为 false。同与操作类似，“||”表示短路或，当运算符“||”的左边为 true 时，右边的表达式不会进行运算。

【例 1-9】运算符“&”和“&&”

```
public class Demo1_9{
    public static void main(String[] args) {
        int x = 0;                          // 定义变量 x，初始值为 0
        int y = 0;                          // 定义变量 y，初始值为 0
        int z = 0;                          // 定义变量 z，初始值为 0
        boolean a, b;                       // 定义 boolean 变量 a 和 b
        a = x > 0 & y++ > 1;  // 逻辑运算符 & 对表达式进行运算，然后将结果赋值给 a
        System.out.println("a = " + a);
        System.out.println("y = " + y);
        b = x > 0 && z++ > 1;// 逻辑运算符 && 对表达式进行运算，然后将结果赋值给 b
        System.out.println("b = " + b);
```

```
            System.out.println("z = " + z);
        }
    }
```

语句解释：

（1）给整型 x，y，z 赋值为 0。

（2）使用 “&”对两个表达式进行运算的结果赋值给 a，x>0 结果为 false，无论 y++>1 的结果为什么，表达式 x>0& y++ >1 的结果都会是 false，即 a=false；使用运算符“&”，运算符两边的表达式都会进行运算，因此变量 y 会进行自增，即 y=1。

（3）使用 “&&”对两个表达式进行运算的结果赋值给 b，x>0 结果为 false，同样表达式结果为 false，即 b=false；使用运算符“&&”，当左边为 false 时，右边的表达式不进行运算，因此变量 z 的值仍为 0，即 z=0。

1.3.5　位运算符

位运算符是对操作数以二进制比特位为单位进行的操作和运算，Java 语言中提供了如表 1–8 所示的位运算符。

表 1–8　位运算符

运算符	功 能	示例	运 算 规 则
~	按位取反	~ a	将 a 按位取反
&	按位与	a & b	将 a 和 b 按比特位相与
|	按位或	a | b	将 a 和 b 按比特位相或
^	按位异或	a ^ b	将 a 和 b 按比特位相异或
>>	右移	a>>b	将 a 各比特位向右移 b 位
<<	左移	a<<b	将 a 各比特位向左移 b 位
>>>	0 填充右移	a>>>b	将 a 各比特位向右移 b 位，左边的空位一律填 0

Java 语言的位运算符可分为按位运算和移位运算两类。位运算符的操作数只能为整型或字符型数据。

⚠ **注意：**有的符号（如 &、|、^）与逻辑运算符的写法相同，但逻辑运算符的操作数为 boolean 型的量。用户在使用这种运算符时要注意它们的区别。

1.3.6　条件运算符

条件运算符，也称作三元条件运算符或三目运算符（?：）。该运算符的格式如下：

```
表达式 1 ? 表达式 2 : 表达式 3;
```

其中“表达式 1”是一个结果为逻辑值的逻辑表达式。该运算符的功能是：先计算“表达式 1”的值，当“表达式 1”的值为 true 时，则将“表达式 2”的值作为整个表达式的值；当“表达式 1”的值为 false 时，则将“表达式 3”的值作为整个表达式的值。

示例代码如下：

```
int a=1,b=2,max;
max = a > b ? a : b;                    //max 获得 a,b 之中的较大值
System.out.println("max="+max);         // 输出结果为 max = 2
```

如果要通过测试某个表达式的值来选择两个表达式中的一个进行计算时，用条件运算符来实现是一种简练的方法。这时，它实现了 if...else 语句的功能。

1.3.7 运算符的优先级

对一些较为复杂的表达式进行运算时，需要明确表达式中所有运算符参与运算的顺序，通常把这种顺序称作为运算符的优先级。通过表 1–9 来展示 Java 中运算符的优先级以及结合性，其中数字越小表示优先级越高。

表 1–9 运算符的优先级及结合性

优先级	运 算 符	运算符的结合性
1	. [] ()	左→右
2	++ -- ! ~ +（正号）-（负号）	右→左
3	* / %	左→右
4	+ -	左→右
5	<< >> >>>	左→右
6	< > <= >=	左→右
7	== !=	左→右
8	&	左→右
9	^	左→右
10	|	左→右
11	&&	左→右
12	||	左→右
13	? :	左→右
14	= += -= *= /= %= <<= >>= >>>= &= ^= |=	右→左

任务 1.4

流程控制语句的使用

1.4.1　顺序结构语句

顺序结构是最简单的程序结构，也是最常见的程序结构，只要按照解决问题的顺序写出相应的语句即可。它的执行顺序是自上而下，依次执行。

1.4.2　选择结构语句

1. if 选择结构

if 选择结构的语法为：

```
if（条件）{
      代码块          //条件成立后要执行的代码，可以是一条语句，也可以是一组语句
}
```

⚠ **注意：**关键字 if 后小括号里的条件是一个表达式，而且表达式的值必须为 true 或 false。程序执行时，先判断条件。

执行过程：当小括号的条件为 true 时，程序先执行大括号里的代码块，再执行 if 结构后面的代码。当结果为 false 时，不执行大括号里的代码块，而直接执行 if 结构后面的代码。

例如：判断成绩等级，如果高于 90 分则输出 Excellent，代码如下：

```
int score = 99;
if(score>90){
  System.out.println("Excellent!");
}
```

2.if…else 选择结构

if…else 选择结构的语法为：

```
if（条件）{
      //代码块 1
}else {
      //代码块 2
}
```

执行过程：当小括号的条件为 true 时，程序执行紧跟 if 语句的代码部分，否则执行跟在 else 语句后面的代码部分。

例如：判断成绩等级，如果大于等于 60 分则弹出提示框提示“考试通过！”，否则提示“不及格！”，代码如下：

```
int score = 99;
if(score>=60){
   System.out.println("考试通过！");
}else{
   System.out.println("不及格！");
}
```

3.if…else if…else 选择结构（多重 if 选择结构）

多重 if 选择结构的语法为：

```
if( 条件 1) {
  条件 1 为真（true）时执行的代码（代码块 1）
}else if( 条件 2){
  条件 2 为真（true）时执行的代码（代码块 2）
  ......
} else{
  所有条件都为假（false）时执行的代码（代码块 n+1）
}
```

⚠ **注意：**else 块最多有一个或没有，else 块必须放在 else if 块之后。

执行过程： 程序判断条件 1，如果成立，则执行代码块 1，然后直接跳出这个多重 if 选择结构，执行它后面的代码。这种情况下，其他代码块都不会被执行。如果条件 1 不成立，则判断条件 2。如果条件 2 成立，则执行代码块 2，然后跳出这个多重 if 选择结构，执行它后面的代码。这种情况下，代码块 1 和其他代码块都不会被执行。如果条件 2 也不成立，则代码块 1 和代码块 2 都不被执行，则继续往下判断，直到条件成立，如果所有条件都不成立，则执行 else 语句块的内容。其中的 else if 语句可以根据实际需要有一个或多个。

【例 1-10】 使用 if…else if…else 语句实现对学生考试成绩进行等级划分的程序

```
public class Demo1_10 {
     public static void main(String[] args) {
          int grade = 75; // 定义学生成绩
          if (grade > 80) {
          // 满足条件 grade > 80
          System.out.println(" 该成绩的等级为优 ");
          } else if (grade > 70) {
          // 不满足条件 grade > 80 ，但满足条件 grade > 70
          System.out.println(" 该成绩的等级为良 ");
          } else if (grade > 60) {
          // 不满足条件 grade > 70 ，但满足条件 grade > 60
          System.out.println(" 该成绩的等级为中 ");
          } else {
          // 不满足条件 grade > 60
          System.out.println(" 该成绩的等级为差 ");
          }
     }
}
```

语句解释：

（1）定义了学生成绩 grade 为 75。

（2）判断是否满足 grade > 80，如果满足，则成绩的等级为优。

（3）如果 grade > 80 不满足，则判断 grade > 70 是否满足，如果满足（不满足条件 grade > 80 ，但满足条件 grade > 70），即 70<grade<80，则成绩的等级为良。

（4）如果 grade > 70 不满足，则判断 grade > 60，如果满足（不满足条件 grade > 70 ，但满足条件

grade > 60 ），即 60<grade<70，则成绩的等级为中。

（5）如果 grade > 60 不满足，即 grade < 60，成绩的等级为差。

4. 嵌套 if 选择结构

if 语句嵌套指的是 if 语句内部包含 if 语句，其格式如下：

```
if （条件表达式 1){
    代码块 1
    if （条件表达式 2){
          代码块 2
  }
}
```

⚠ **注意：** if 语句可以多层嵌套，不仅限于两层；外层和内层的 if 判断都可以使用 if 语句、if...else 语句等。

执行过程：先判断外层 if 语句中条件表达式 1 的结果是否为 true，如果结果为 true，则执行代码块 1，再判断内层 if 的条件表达式 2 的结果是否为 true，如果条件表达式 2 的结果为 true，则执行代码块 2。

【例 1-11】 使用嵌套 if 选择结构语句实现根据年份和月份计算当月一共有多少天（四年一闰，百年不闰，四百年再闰）。

```
public class Demo1_11 {
     public static void main(String[] args) {
          int year =2023;
          int month =2;
          if (month == 1 || month == 3 || month == 5 || month == 7 ||
                   month == 8 || month == 10 || month == 12) {
                System.out.println(year + " 年 " + month + " 月有 31 天 ");
          } else if (month == 4 || month == 6 || month == 9 || month == 11) {
                System.out.println(year + " 年 " + month + " 月有 30 天 ");
          } else if (month == 2) {
                if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
                      System.out.println(year + " 年 " + month + " 月有 29 天 ");
                } else {
                     System.out.println(year + " 年 " + month + " 月有 28 天 ");
                }
          } else {
                System.out.println(" 月份输入非法！ ");
          }
     }
}
```

语句解释：

（1）定义了表示年份和月份的变量 year 和 month。

（2）对月份进行判断：若月份为 1、3、5、7、8、10、12，输出“* 年 * 月有 31 天”；若月份为 4、6、9、11，输出“* 年 * 月有 30 天”；若月份为 2 月，则需要对年份进行判断，年份为闰年时输出“* 年 * 月有 29 天”，年份为平年时输出“* 年 * 月有 28 天”。

5. switch 选择结构语句

switch 选择结构语句语法：

```
switch( 控制表达式 ){
  case 值 1：
      执行代码段 1
      break;
  case 值 2：
      执行代码段 2
      break;
  ……
  case 值 n-1：
      执行代码段 n-1
      break;
  [default： 以上条件均不符合时的执行代码段 n]
}
```

⚠ **注意：** switch 语句的控制表示式结果只能是 byte、short、char、int、enum 以及 String 类型，不能是 boolean 类型。

执行过程：switch 语句将控制表达式的值与每个 case 中的目标值进行匹配，如果找到了匹配的值，会执行对应 case 后的语句，并通过 break 关键字跳出 switch 语句；如果没找到任何匹配的值，就会执行 default 后的语句。

【例 1-12】 使用 switch 语句根据给出的数值来输出对应中文格式的星期。

```
public class Demo1_12 {
    public static void main(String[] args) {
        int week = 5;
        switch (week) {
        case 1:
            System.out.println("星期一");
            break;
        case 2:
            System.out.println("星期二");
            break;
        case 3:
            System.out.println("星期三");
            break;
        case 4:
            System.out.println("星期四");
            break;
        case 5:
            System.out.println("星期五");
            break;
        case 6:
            System.out.println("星期六");
            break;
```

```
            case 7:
                System.out.println(" 星期天 ");
                break;
            default:
                System.out.println(" 输入的数字不正确 ...");
                break;
            }
        }
    }
```

语句解释：由于变量 week 的值为 5，整个 switch 语句判断的结果满足示例代码的条件，因此打印出“星期五”。

1.4.3　循环结构语句

在 Java 中有 3 种类型的循环语句：while 循环语句，do...while 循环语句以及 for 循环语句。

1.while 循环语句

while 循环语句的语法结构：

```
while ( 循环条件 ) {
    执行语句
}
```

执行过程：关键字 while 后小括号中的循环条件是一个表达式，它的值为布尔类型，即 true 或 false。循环体是否执行取决于循环条件。当循环条件为 true 时，循环体就会被执行。循环体执行完毕后，程序会继续判断循环条件是否成立，如果条件仍为 true 则会继续执行循环体，直到循环条件的结果为 false 时，整个循环过程才会结束。

【例 1-13】　使用 while 循环语句实现打印 1~4 之间的自然数。

```
public class Demo1_13 {
    public static void main(String[] args) {
        int x = 1; // 定义变量 x，初始值为 1
        while (x <= 4) { // 循环条件
            System.out.println("x = " + x); // 条件成立，打印 x 的值
            x++; // x 进行自增
        }
    }
}
```

运行结果如下所示。

```
x=1
x=2
x=3
x=4
```

语句解释：x 初始值为 1，在满足循环条件 x<=4 的情况下，循环体会重复执行，打印 x 的值并让 x 进行自增。因此打印结果中 x 的值分别为 1、2、3、4。值得注意的是，示例中第 6 行代码用于在每次循环时改变变量 x 的值，从而达到最终改变循环条件的目的。如果没有这行代码，整个循环会进入

无限循环的状态，永远不会结束。

2.do…while 循环语句

do…while 循环语句又称为后测试循环，其语法结构：

```
do{
    循环体
} while( 条件表达式 )
```

执行过程：关键字 do 后面 { } 中的执行语句是循环体。do...while 循环语句将循环条件放在了循环体的后面。这也就意味着，循环体会无条件执行一次，然后再根据循环条件来决定是否继续执行。

⚠ **注意：** do…while 循环语句与 while 循环语句的区别：do…while 循环语句是先执行循环体，再进行循环条件判断；while 循环语句是先循环条件判断，再执行循环体。

【例 1-14】 使用 do…while 循环语句实现打印 1~4 之间的自然数。

```
public class Demo1_14{
     public static void main(String[] args) {
          int x = 1; // 定义变量 x，初始值为 1
          do {
               System.out.println("x = " + x); // 打印 x 的值
               x++; // 将 x 的值自增
          } while (x <= 4); // 循环条件
     }
}
```

运行结果如下所示：

```
x=1
x=2
x=3
x=4
```

3. for 循环语句

for 循环语句是最常用的循环语句，一般用在循环次数已知的情况下。for 循环语句的语法结构：

```
for( 语句 1; 语句 2; 语句 3){
    循环体
}
```

⚠ **注意：** 语句 1、语句 2、语句 3 之间用“;”分隔。

执行过程：①表示语句 1：初始化表达式；②表示语句 2：循环条件；③表示语句 3：操作表达式；④表示循环体。

```
for( ①; ②; ③ ){
④
}
```

第 1 步，执行①；

第 2 步，执行②，如果循环条件结果为 true，则执行第 3 步，如果结果为 false，则执行第 5 步；

第 3 步，执行④；

第 4 步，执行③，然后重复执行第 2 步；

第 5 步，退出循环。

【例 1-15】　使用 for 循环语句实现对自然数 1~4 进行求和

```
public class Demo1_15 {
    public static void main(String[] args) {
        int sum = 0; // 定义变量 sum，用于记住累加的和
        for (int i = 1; i <= 4; i++) { // i 的值会在 1~4 之间变化
            sum += i; // 实现 sum 与 i 的累加
        }
        System.out.println("sum = " + sum); // 打印累加的和
    }
}
```

运行结果如下所示。

```
sum = 10
```

语句解释：变量 i 的初始值为 1，在判断条件 i<=4 为 true 的情况下，会执行循环体 sum+=i，执行完毕后，会执行操作表达式 i++，i 的值变为 2，然后继续进行条件判断，开始下一次循环，直到 i=5 时，条件 i<=4 为 false，结束循环，执行 for 循环后面的代码，打印“sum= 10”。

4. 循环嵌套

相同或不同的循环结构之间可以互相嵌套，也可以与选择结构语句嵌套使用，用来实现更为复杂的逻辑。

【例 1-16】　实现使用“*” 打印边长为 4 的正方形

```
public class Demo1_16{
    public static void main(String[] args) {
        int i, j; // 定义两个循环变量
        for (i = 1; i <= 4; i++) { // 外层循环
            for (j = 1; j <= 4; j++) { // 内层循环
                System.out.print("* "); // 打印 *
            }
            System.out.print("\n"); // 换行
        }
    }
}
```

运行结果如下所示。

```
* * * *
* * * *
* * * *
* * * *
```

语句解释：外层循环用于控制打印的行数，内层循环用于打印“*”，每一行的“*” 个数逐行增加，最后输出一个正方形。

【例 1-17】　实现使用 * 打印直角三角形

```
public class Demo1_17{
    public static void main(String[] args) {
        int i, j; // 定义两个循环变量
```

```
            for (i = 1; i <= 4; i++) { // 外层循环
                    for (j = 1; j <= i; j++) { // 内层循环
                            System.out.print("* "); // 打印 *
                    }
                    System.out.print("\n"); // 换行
            }
        }
    }
```

运行结果如下所示。

```
*
* *
* * *
* * * *
```

语句解释：

第 1 步，在第 3 行代码定义了两个循环变量 i 和 j，其中，i 为外层循环变量，j 为内层循环变量。

第 2 步，在第 4 行代码将 i 初始化为 1，条件 i<=4 为 true，首次进入外层循环的循环体。

第 3 步，在第 5 行代码将 j 初始化为 1，由于此时 i 的值为 1，条件 j<=i 为 true， 首次进入。

第 4 步，执行第 5 行代码中内层循环的操作表达式 ++，将 j 的值自增为 2。

第 5 步，执行第 5 行代码中的判断条件 j<=i，判断结果为 false，内层循环结束。执行后面的代码，打印换行符。

第 6 步，执行第 4 行代码中外层循环的操作表达式 i++，将 i 的值自增为 2。

第 7 步，执行第 4 行代码中的判断条件 i<=4，判断结果为 true，进入外层循环的循环体，继续执行内层循环。

第 8 步，由于 i 的值为 2，内层循环会执行两次，即在第 2 行打印两个“*”。 在内层循环结束时会打印换行符。

第 9 步，依此类推，在第 3 行会打印 3 个“*”，逐行递增，直到 i 的值为 4 时，外层循环的判断条件 i<=4 结果为 false，外层循环结束，整个程序也就结束了。

5. 跳转语句

跳转语句用于实现循环语句执行过程中程序流程的跳转，Java 中的跳转语句有 break 语句和 continue 语句。

使用 break 语句和 continue 语句的注意点如下。

（1）共同点：break 语句和 continue 语句在循环语句中都可以使用，并且一般常与选择结构结合，以达到在特定条件得到满足时跳出循环的目的。

（2）区别：一旦 break 语句被执行，整个循环就提前结束。continue 语句的作用是终止本次循环，并忽略 continue 语句之后的所有语句，直接回到循环的顶端，提前进入下一次循环。

【例 1-18】对例 1-13 稍作修改，当变量 x 的值为 3 时，使用 break 语句跳出循环。

```
public class Demo1_18 {
    public static void main(String[] args) {
        int x = 1; // 定义变量 x，初始值为 1
```

```
            while (x <= 4) { // 循环条件
                System.out.println("x = " + x); // 条件成立，打印 x 的值
                if (x == 3) {
                    break;
                }
                x++; // x 进行自增
            }
        }
    }
```

运行结果如下所示：

```
x = 1
x = 2
```

语句解释：通过 while 循环打印 x 的值，当 x 的值为 3 时，使用 break 语句跳出循环。因此打印结果中并没有出现“x=4”和“x=5”。

⚠ **注意：**当 break 语句出现在嵌套循环中的内层循环时，它只能跳出内层循环，如果想使用 break 语句跳出外层循环，则需要对外层循环添加标记。

【例 1-19】　对例 1-17 进行修改，使用 break 语句跳出外层循环

```
public class Demo1_19{
    public static void main(String[] args) {
        int i, j; // 定义两个循环变量
        out: for (i = 1; i <= 4; i++) { // 外层循环
            for (j = 1; j <= i; j++) { // 内层循环
                if (i > 3) { // 判断 i 的值是否大于 3
                    break out; // 跳出外层循环
                }
                System.out.print("* "); // 打印 *
            }
            System.out.print("\n"); // 换行
        }
    }
}
```

运行结果如下所示：

```
*
* *
* * *
```

语句解释：在外层 for 循环前面增加了标记“out”。当 i>3 时，使用 break out; 语句跳出外层循环。因此程序只打印了 3 行“*”。

【例 1-20】continue 语句，求 1~100 之间个位数字不是 2、3、4、7，并且不能被 3 整除的整数之和。

```
public class Demo1_20{
    public static void main(String[] args) {
        int sum = 0;// 定义变量保存累加值
```

```
            for (int i = 1; i <= 100; i++) {
                // 判断 i 是否满足条件
                if (i % 10 == 2 || i % 10 == 3 || i % 10 == 4 || i % 10 == 7 || i % 3 == 0) {
                    continue;
                }
                sum = sum + i;// 进行累加
            }
            System.out.println("1~100 之间个位数不是 2、3、4、7,"
                    + " 并且不能被 3 整除的整数和是 :" + sum);
        }
    }
```

运行结果如下所示：

```
1~100 之间个位数不是 2、3、4、7, 并且不能被 3 整除的整数和是 :2058
```

语句解析：使用 if 语句进行条件判断。如果 i 对 10 取余等于 2，或 i 对 10 取余等于 3，或 i 对 10 取余等于 4，或 i 对 10 取余等于 7，或 i 对 3 取余等于 0，则说明 i 不是所需要的数，不进行累加，而进入下一次的循环。

任务 1.5

数组的使用

1.5.1 数组的定义

在 Java 中，数组的定义有三种方式，其基本语法格式如下：

```
数组类型 [ ] 数组名 = new 数组类型 [ 数组长度 ];
数组类型 [ ] 数组名 = new 数组类型 [ ]{ 数组元素 0, 数组元素 1,...};
数组类型 [ ] 数组名 ={ 数组元素 0, 数组元素 1,...};
```

示例如下：

```
int [ ] array = new int [100];
String [ ] name =new String [ ] { “张三” , “tom” ,...};
Object[ ] object = { “张三” , “tom” ,...};
```

语句解释：第一种方式定义了一个 int[ ] 数组类型的变量 array，使用 new int [100] 指定了该数组的长度（即元素的个数）为 100，但没有为数组中的元素赋值；第二种和第三种方式分别定义了一个 String [ ] 类型和 Object[ ] 类型的数组变量 names 和 object 并都通过 = 右边的大括号 {} 为数组完成了初始化赋值。

⚠ 注意：

（1）第一种定义方式，相当于在内存中定义了 100 个 int 类型的变量，第一个变量的名称为 array[0]，第 2 个变量的名称为 array [1]... 依此类推，第 100 个变量的名称为 x[99]，这些变量的初始值都是 0。

（2）数组中的每个元素都有一个索引（或称为角标），最小的索引值为 0，最大的索引值为“数组

的长度 −1”。访问数组中的元素可以通过“数组名 [角标]”，即“array[0]、…、array[99]”。

（3）在 Java 中，为获取数组的长度，提供 length 属性，可通过“数组名 .length”的方式获得数组的长度，即数组元素的个数。

【例 1-21】定义数组以及访问数组中的元素

```
public class Demo1_21{
    public static void main(String[] args) {
        int[] arr = new int[3]; // 创建数组对象
        System.out.println("arr[0]=" + arr[0]); // 访问数组中的第一个元素
        System.out.println("arr[1]=" + arr[1]); // 访问数组中的第二个元素
        System.out.println("arr[2]=" + arr[2]); // 访问数组中的第三个元素
        System.out.println(" 数组的长度是：" + arr.length); // 打印数组长度
    }
}
```

运行结果如下所示：

```
arr[0]=0
arr[1]=0
arr[2]=0
数组的长度是：3
```

语句分析：声明了一个 int[] 类型变量 arr，创建了一个长度为 3 的数组，然后通过下标来访问数组中的元素，通过 length 属性访问数组中元素的个数。从打印结果可以看出，数组的长度为 3，且 3 个元素初始值都为 0，这是因为当数组被成功创建后，数组中元素会被自动赋予一个默认值，根据元素类型的不同，默认初始化的值也是不一样的。具体如表 1-10 所示。

表 1-10　元素默认值

数据类型	默认初始化值
byte、short、int、long	0
float、Double	0.0
char	一个空字符，即 '\u0000'
boolean	false
引用数据类型	null, 表示变量不引用任何对象

注意：如果在使用数组时，不想使用这些默认初始值，也可以显式地为这些元素赋值。

【例 1-22】　为数组的元素赋值

```
public class Demo1_22 {
    public static void main(String[] args) {
        int[] arr = new int[4]; // 定义可以存储 4 个元素的整数类型数组
        arr[0] = 1; // 为第 1 个元素赋值 1
        arr[1] = 2; // 为第 2 个元素赋值 2
        // 依次打印数组中每个元素的值
        System.out.println("arr[0]=" + arr[0]);
        System.out.println("arr[1]=" + arr[1]);
```

```
            System.out.println("arr[2]=" + arr[2]);
            System.out.println("arr[3]=" + arr[3]);
        }
    }
```

运行结果如下所示：

```
arr[0]=1
arr[1]=2
arr[2]=0
arr[3]=0
```

语句解释：第 3 行代码定义了一个数组，此时数组中每个元素都为默认初始值 0。第 4、5 行代码通过赋值语句将数组中的元素 arr[0] 和 arr[1] 分别赋值为 1 和 2，而元素 arr[2] 和 arr[3] 没有赋值，其值仍为 0，因此打印结果中 4 个元素的值依次为 1、2、0、0。

1.5.2 数组的应用

1. 数组遍历

在操作数组时，经常需要依次访问数组中的每个元素，这种操作称作数组的遍历。接下来通过一个案例来学习如何使用 for 循环遍历数组，如例 1-23 所示。

【例 1-23】 for 循环遍历数组

```
public class Demo1_23 {
    public static void main(String[] args) {
        int[] arr = { 1, 2, 3, 4, 5 }; // 定义数组
        // 使用 for 循环遍历数组的元素
        for (int i = 0; i < arr.length; i++) {
            System.out.println(arr[i]); // 通过索引访问元素
        }
    }
}
```

运行结果如下所示。

```
1
2
3
4
5
```

语句解释：定义一个长度为 5 的数组 arr，数组的下标为 0~4。由于 for 循环中定义的变量 i 的值在循环过程中为 0~4，因此可以作为索引，依次去访问数组中的元素，并将元素的值打印出来。

2. 数组的最值

在操作数组时，经常需要获取数组中元素的最值。接下来通过一个案例来演示如何使用获取数组中元素的最大值，如例 1-24 所示。

【例 1-24】 求最大值

```
public class Demo1_24{
    /**
```

```
     * 求数组最大值
     */
    public static void main(String[] args) {
        //定义一个 int[ ] 数组
        int[] arr = new int[6]{7,1,6,3,5,8};
        //定义变量 max 用于记录最大值，并先假设第一个元素为最大值
        int max = arr [0];
        //遍历数组，查找最大值
        for(int i = 1; i < arr.length; i++){
        //比较 arr[i] 的值是否大于 max
            if(arr[i] > max){
            //条件成立，将 arr[i] 的值赋给 max
                max = arr[i];
            }
        }
        //打印最大值
        System.out.println("数组 arr 中的最大值为：" + max);
    }
}
```

运行结果如下所示。

```
数组 arr 中的最大值为：8
```

语句解释：先定义变量 max 存放数组的最大值，并假设数组中的第一个元素 arr[0] 为最大值，然后使用 for 循环对数组进行变量，遍历过程中如果遇到比 max 值大的元素，就将该元素赋值给 max。这样一来，变量 max 在循环结束时保存的是数组中的最大值。

3. 数组排序

在操作数组时，经常需要对数组中元素进行排序。排序的方法很多，本例用“冒泡法排序”。“冒泡法”的基本思路是：每次将相邻两个数比较，将小的调到前头（相当于气泡往上冒）。

【例 1-25】使用冒泡排序，实现 6 个数：9，8，5，4，2，0 的排序

思路分析：

第 1 次先将最前面的两个数 8 和 9 对调（如图 1-6 所示）。第 2 次将第 2 和第 3 个数（9 和 5）对调……如此共进行 5 次，得到 8-5-4-2-0-9 的顺序，可以看到：最大的数 9 已“沉底”，成为最下面一个数，而小的数“上升”。最小的数 0 已向上“浮起”一个位置。经过第 1 趟（共 5 次比较与交换）后，已得到最大的数 9。

然后进行第 2 趟比较，对余下的前面 5 个数（8，5，4，2，0）进行新一轮的比较，以便使次大的数“沉底”。按以上方法进行第 2 趟比较。经过这一趟 4 次比较与交换，得到次大的数 8。

按此规律进行下去，可以推知，对 6 个数要比较 5 趟，才能使 6 个数按大小顺序排列。在第 1 趟中要进行两个数之间的比较共 5 次，在第 2 趟过程中比较 4 次……第 5 趟只须比较 1 次。

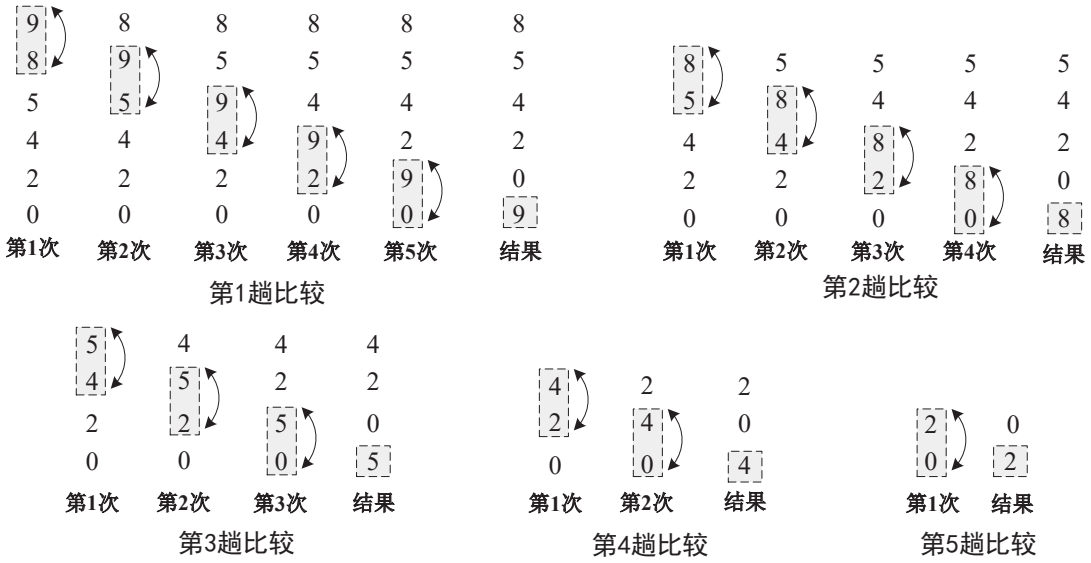


图 1–6　“冒泡排序”示意图

示例代码：

```
public class Demo1_25 {
     public static void main(String args[]) {
     int[] intArray = {9 , 8 , 5 , 4 , 2 , 0};
     System.out.println(" 排序前的数组 :");
     for (int i = 0; i < intArray.length; i++){
        System.out.print(intArray[i] + " ");
        System.out.println();
     }
     int temp;
     for (int i = 0; i < intArray.length; i++) {
        for (int j = i; j < intArray.length; j++) {
          if (intArray[j] < intArray[i]) {
             temp = intArray[i];
             intArray[i] = intArray[j];
             intArray[j] = temp;
          }
        }
     }
     System.out.println(" 排序后的数组 :");
     for (int i = 0; i < intArray.length; i++)
        System.out.print(intArray[i] + " ");
     }
}
```

运行结果如下所示。

排序前的数组：

```
9 8 5 4 2 0
```

排序后的数组：

```
0 2 4 5 8 9
```

语句解释：通过一个嵌套 for 循环实现了冒泡排序。其中外层循环用来控制进行多少轮比较，每一轮比较都可以确定一个元素的位置，由于最后一个元素不需要进行比较，所以外层循环的次数为 intArray.length−1。内层循环的循环变量用于控制每轮进行比较的相邻的两个数，它被作为角标去比较数组的元素，由于变量在循环过程中是自增的，这样就可以实现相邻元素依次进行比较，在每次比较时如果前者小于后者，就交换两个元素的位置。

1.5.3　多维数组

虽然一维数组可以处理一般简单的数据，但是在实际的应用中仍显不足，所以 Java 语言提供了多维数组。但在 Java 语言中并没有真正的多维数组，所谓多维数组，就是数组元素也是数组的数组。比较常见的多维数组是二维数组，接下来针对二维数组进行详细讲解。

二维数组可以被看作特殊的一维数组，其基本语法格式如下：

```
int[ ][ ] s = new int[3][5];
```

语句解释：定义了一个 3*5 的二维数组，二维数组的长度为 3，可以将它看作 3 个 int[ ] 类型的一维数组，每个一维数组中的元素又是一个长度为 5 的一维数组。即二维数组 s，包括 3 个元素，分别为 s[0]、s[1]、s[2]，每个元素是整型数组类型即一维数组类型。而 s[0] 又是一个数组的名称，包括 5 个元素，分别为 s[0][0]、s[0][1]、s[0][2]、s[0][3]、s[0][4]，每个元素都是整数类型。s[1]、s[2] 与 s[0] 的情况相同如图 1–7 所示。

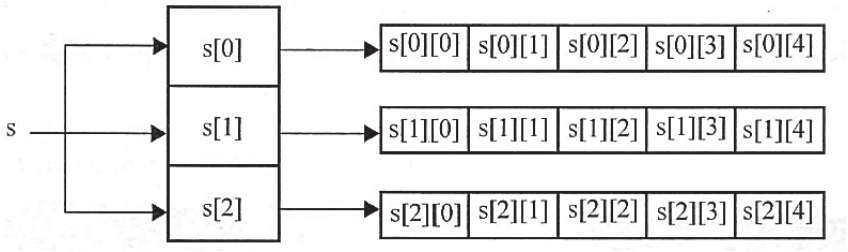


图 1–7　二维数组存储方式示意图

【例 1–26】　计算每个班级的学生总成绩

```
public class Demo1_26{
    public static void main(String[] args) {
        // 定义二维数组、分配空间、赋值
        int[][] array = new int[][] { { 80, 66 }, { 70, 54, 98 }, { 77, 59 } };
        int total;
        // 保存总成绩
        for (int i = 0; i < array.length; i++) {
            String str = (i + 1) + "班";
            total = 0;
            // 每次循环到此都将其归 0
            for (int j = 0; j < array[i].length; j++) {
                total += array[i][j];// 成绩叠加
            }
```

```
                    System.out.println(str + " 总成绩 :" + total);
            }
        }
    }
```

运行结果为：

```
1 班总成绩 : 146
2 班总成绩 : 222
3 班总成绩 : 136
```

知识拓展

1. 注释嵌套

在 Java 中，有的注释可以嵌套使用，有的则不可以。

（1）多行注释“/*…*/”中可以嵌套使用单行注释“//”。

例如：

```
 /*  int c = 10;   // 定义一个整型的 c
     int x = 5; */
```

（2）多行注释“/*…*/”中不能嵌套使用多行注释“/*…*/”。

例如：

```
 /*
    /*int c = 10;*/
    int x=5;
*/
```

2. 表达式类型自动提升

所谓表达式是指由变量和运算符组成的一个算式。变量在表达式中进行运算时，也有可能发生自动类型转换，这就是表达式数据类型的自动提升，如 byte、short 和 char 类型的变量在运算期间类型会自动提升为 int，然后再进行运算。下面通过一个具体的案例来演示，如例 1–27 所示。

【例 1–27】达式类型自动提升

```
public class Demo1_27 {
     public static void main(String[] args) {
          byte b =3;
          short s =4;
          char c =5;
          byte b2 = b+s+c;              // 将 byte、short、char 类型数值相加，再赋值给 byte 类型
          System.out.println("b2=" + b2);
     }
}
```

程序编译报错，如图 1–8 所示。

```
Dem01_27.java
1  public class Dem01_27 {
2      public static void main(String[] args) {
3          byte b = 3;
4          short s = 4;
5          char c = 5;
6          byte b2 = b + s + c;        //将byte、short、char类型数值相加，再赋值给byte
7          System.out.printl
8      }
9  }
10
```

Required type: byte
Provided: int
Cast to 'byte' Alt+Shift+Enter　More actions... Alt+Enter

图 1–8　程序编译报错

图 1–8 中，出现了和图 1–3 相同的类型转换错误，这是因为在表达式 b + s + c 运算期间，byte 类型的 b、short 类型的 s 和 char 类型的 c 都被自动提升为 int 型，表达式的运算结果也就成了 int 型，这时如果将该结果赋给 byte 型的变量就会报错，此时就需要进行强制类型转换。

要解决例中的错误，必须要将第 5 行的代码修改为：

```
byte b2 = (byte) (b+s+c);
```

再次编译后，程序不会报错，运行结果如下所示。

```
b2=12
```

⚠ **注意：** 在 Java 语言中，布尔类型数据不能进行类型转换。

3. 转义符——反斜杠（\）

在字符常量中，反斜杠（\）是一个特殊的字符，被称为转义符，它的作用是用来转义紧随其后的一个字符。转义后的字符通常用于表示一个不可见的字符或具有特殊含义的字符，例如“\n”表示换行。下面列出一些常见的转义字符。

\r：表示回车，将光标定位到当前行的开头，不会跳到下一行。

\n：表示换行，换到下一行的开头。

\t：表示制表符，将光标移到下一个制表符的位置，就像在文档中用 Tab 键一样。

\b：表示退格符号，就像键盘上的 Backspace。

\'：表示单引号字符，表示字符的开始和结束，如果直接写单引号字符（'），程序会认为前两个是一对，会报错，因此需要使用转义（\'）。

\"：表示双引号字符，表示字符串的开始和结束，包含在字符串中的双引号需要转义，比如 "he says,\"thank you\"."。

\\：表示反斜杠转义字符，由于在 Java 代码中的斜杠（\）是转义符，因此需要表示字面意义上的 \，就需要使用双斜杠 \\。

单元总结

本单元主要介绍了学习 Java 所需的基础知识。首先介绍了 Java 语言的基本语法、常量、变量的定义以及一些常见运算符的使用，然后介绍了条件选择结构语句和循环结构语句的概念和使用，最后介绍了数组的相关操作。通过本单元的学习，读者能够掌握 Java 程序的基本语法、格式以及变量和运算符的使用，能够掌握几种流程控制语句的使用方式，能够掌握数组的声明、初始化和使用等知识。

思考与练习

一、选择题（单选）

1. 若 x = 5,y = 8，则表达式 x|y 的值为（　　）。

A. 3　　B. 13　　C. 0　　D. 5

2. 以下（　　）是合法的变量名。

A. double　　B. 3x　　C. sum　　D. de$f

3. 基本数据类型 short 的取值范围是（　　）。

A.（−256）~255　　B.（−32 768）~32767　　C.（−128）~127　　D. 0~65535

4. 表达式 (11+3*8)/ 4 % 3 的值是（　　）

A. 31　　B. 0　　C. 1　　D. 2

5. 为一个 boolean 类型变量赋值时，可以使用（　　）方式

A. boolean a = 1;　　B. boolean a = (9>=10);

C. boolean a ="真";　　D. boolean a == false;

6. 若定义有变量 float f1,f2 =8.0F，则下列说法正确的是（　　）。

A. 变量 f1，f2 均被初始化为 8.0　　B. 变量 f1 没有被初始化，f2 被初始化为 8.0

C. 变量 f1，f2 均未被初始化　　D. 变量 f2 没有被初始化，f1 被初始化为 8.0

7. 若定义有 short s; byte b; char c; 则表达式 s * b + c 的类型为（　　）。

A. char　　B. short　　C. int　　D. byte

8. 下列（　　）是不能通过编译的语句。

A. double d = 545.0;　　B. char a1 = “c”;　　C. int i = 321;　　D. float f1 =45.0f;

9. 编译运行如下 Java 代码，输出结果是（　　）。

```
int num = 5;
if (num <= 5) {
num += 2;
System.out.println(num);
}
System.out.println(num + 5);
```

A. 10　　B. 5　10　　C. 7　12　　D. 运行出错

10. 下面这段代码的输出结果为（　　）。

```
int year = 2046;
```

```
if (!(year % 2 == 0)) {
    if (year / 10 == 0) {
        System.out.println("进入了 if");
    }
} else {
    System.out.println("进入了 else");
    System.out.println("退出");
}
```

A. 进入了 if　B. 退出　C. 进入了 else　退出　D. 进入了 if　退出

11. 关于以下代码，下面说法正确的是（　　）。

```
int k = 10;
while (k == 0) {
k = k - 1;
}
```

A. 循环将执行 10 次　B. 死循环，将一直执行下去

C. 循环将执行 1 次　D. 循环一次也不执行

12. 以下关于 break 语句和 continue 语句的说法正确的是（　　）。

A. continue 语句的作用是结束整个循环的执行

B. 在循环体内和 switch 结构体内可以使用 break 语句

C. 循环体内使用 break 语句或 continue 语句的作用相同

D. 在 switch 结构体内也可以使用 continue

13. 定义一个数组 Strirg[] cities={"北京","上海","天津","重庆","武汉","广州","香港"}，数组中的 cities[6] 指的是（　　）。

A. 北京　B. 广州　C. 香港　D. 数组越界

二、填空题

1. 已知：int a =8,b=6; 则：表达式 ++a-b++ 的值为_____。

2. 已知：boolean b1=true,b2; 则：表达式！b1 && b2 ||b2 的值为_____。

3. 已知：double x=8.5,y=5.8; 则：表达式 x++>y-- 的值为_____。

4. 执行 int x, a = 2,b = 3,c = 4; x = ++a + b++ + c++;x 的结果是_____。

5. Java 中的显式类型转换既能_____也能从高类型向低类型转换，而隐式类型转换只有前者。

6. 布尔常量即布尔类型的两个值，分别是_____和_____。

7. 执行下列程序代码的输出结果是_____。

int a = 10; int i, j; i = ++a; j = a--;

System.out.printf("%d,%d,%d", a, i, j);

8. 执行完 boolean x=false; boolean y=true; boolean z=(x&&y)&&(!y); int f=z==false?1:2; 这段代码后，z 与 f 的值分别是_____和_____。

9. 数组的元素通过_____来访问，求数组长度可以调用_____方法。

10. 假设一个数组的长度为 n，那么数组下标是从_____到_____。

三、判断题

1. “[][]”用于表明定义了一个二维数组，通过多个下标进行数据访问。(　　)
2. Java 语言中不同数据类型的长度是固定的，不随机器硬件不同而改变。(　　)
3. 所有的变量在使用前都必须进行初始化。(　　)
4. 已知 byte i = (byte)127; i = i +1;这两个语句能被成功编译。(　　)
5. 说明或声明数组时不分配内存大小，创建数组时分配内存大小。(　　)
6. 强制类型转换运算符的功能是将一个表达式的类型转换为所指定的类型。(　　)
7. 在使用 if 嵌套时，else 语句块中不能再出现 if 语句。(　　)
8. 在使用 switch 语句时，break 和 continue 的作用都是一样的。(　　)
9. 一维数组中可以存储不同数据类型的元素。(　　)
10. 数组元素的序号是从 0 开始。(　　)

四、简答题

1. 简述 Java 中变量的命名规则。

2. 说明在循环结构中 break 语句和 continue 语句的区别。

3. 阅读以下代码，找出其中的错误。

```
String[] scores =new String[5]{"Mike", "Lily","Sunny", "Tenny", "Ana"};
for (int i=0; i <= scores.length; i++){
        System.out.println(scores[i]);
}
```

五、编程题

1. 小明左手、右手中分别拿一张纸牌：黑桃 10 和红桃 8，现在交换手中的牌。用程序模拟这一过程：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。程序运行结果如下所示。

```
输出互换前手中的纸牌：
左手中的纸牌：10
右手中的纸牌： 8
输出互换后手中的纸牌：
左手中的纸牌： 8
右手中的纸牌： 10
```

提示：互换两个变量的值需要借助第三个变量。前两个变量用来存储两个整数，第三个变量用来作为中间变量，借助于这个中间变量，将两个变量的值进行互换。

2. 在马克思手稿中有一道趣味的数学问题：一共有 30 个人，可能包括男人、女人和小孩。他们在一家饭馆吃饭共花了 50 先令，其中每个男人花 3 先令，每个女人花 2 先令，每个小孩花 1 先令。请问男人、女人和小孩各几人？请编写一个程序来解决这个问题。

提示：

- 定义三个变量 men、women 和 kids 分别表示男人数、女人数和小孩数，可以得到如下两个关系。

```
men + women + kids = 30;
3×men + 2xwomen + kids = 50;
```

- 男人数 i 的范围是 0<=men<=10，利用循环实现。

3. 某百货商场当日消费积分最高的八名顾客，他们的积分分别是 18、25、7、36、13、2、89、63。编写程序找出最低的积分及它在数组中的原始位置（下标）。

提示：

- 创建数组 points[]，存储八名顾客的积分。
- 定义变量 min，存储最低积分；定义变量 index，存储最低积分的下标。
- 假设第一个元素为最低积分，下标为 0。
- 遍历数组，将数组元素和 min 的值进行比较。