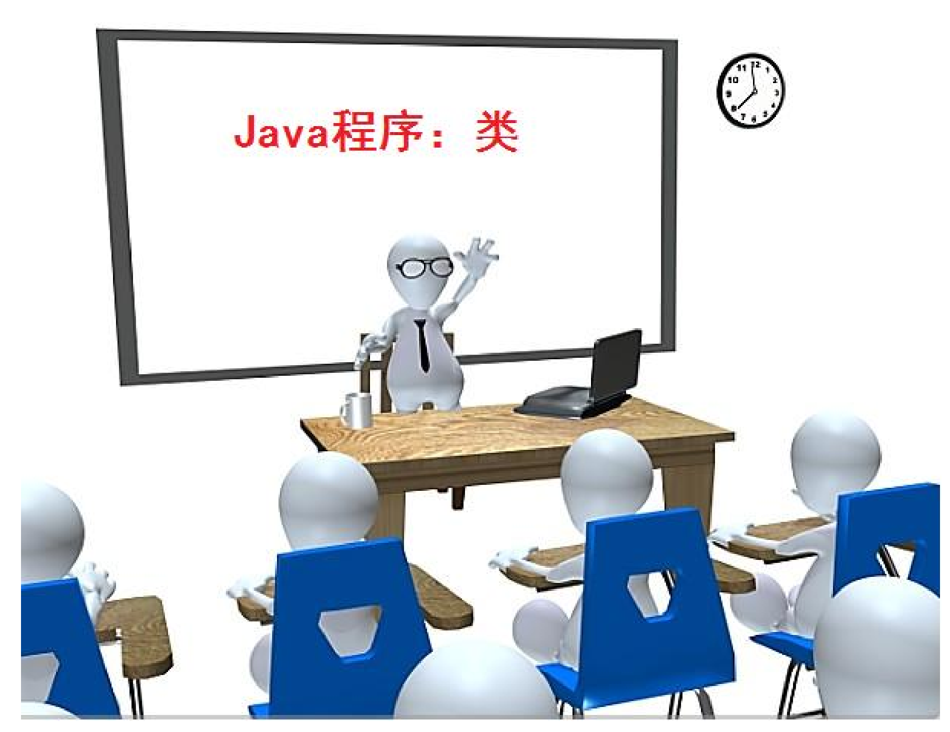
第2章 Java编程基础

- Java基本语法
- 常量、变量的定义与使用
- 运算符
- 选择结构与循环结构
- 方法
- 数组

2.1.1 Java程序的基本格式

Java程序代码必须放在一个类中，初学者可以简单地把一个类理解为一个Java程序。类使用class关键字定义，在class前面可以有类的修饰符，类的定义格式如下：

```
修饰符 class 类名{
        程序代码
}
```



2.1.1 Java程序的基本格式

（1）Java程序代码可分为结构定义语句和功能执行语句，其中，结构定义语句用于声明一个类或方法，功能执行语句用于实现具体的功能。每条功能执行语句的最后必须用分号(;)结束。如下面的语句：

```
System.out.println("这是第一个Java程序！");
```

需要注意的是，在程序中不要将英文的分号(;)误写成中文的分号（；），如果写成中文的分号，编译器会报告 “illegal character” （非法字符）错误信息。

2.1.1 Java程序的基本格式

（2）Java语言是严格区分大小写的。在定义类时，不能将class写成Class，否则编译器会报错。程序中定义一个computer的同时，还可以定义一个Computer，computer和Computer是两个完全不同的符号，在使用时务必注意。

2.1.1 Java程序的基本格式

（3）在编写Java程序时，为了便于阅读，通常会使用一种良好的格式进行排版，但这并不是必需的，我们也可以在两个单词或符号之间插入空格、制表符、换行符等任意的空白字符。例如，下面这段代码的编排方式也是可以的。

```
public class HelloWorld {public static void
            main(String [
] args){System.out.println("这是第一个Java程序！");}}
```

2.1.1 Java程序的基本格式

虽然Java没有严格要求用什么样的格式编排程序代码，但是，出于可读性的考虑，应该让自己编写的程序代码整齐美观、层次清晰。常用的编排方式是一行只写一条语句，符号“{”与语句同行，符号“}”独占一行，示例代码如下：

```
public class HelloWorld {
        public static void main(String[] args) {
                System.out.println("这是第一个Java程序！");
        }
}
```

2.1.1 Java程序的基本格式

（4）Java程序中一个连续的字符串不能分成两行书写。例如，下面这条语句在编译时将会出错：

```
System.out.println("这是第一个
                    Java程序！");
```

如果为了便于阅读，需要将一个比较长的字符串分两行书写，可以先将字符串分成两个字符串，然后用加号（+）将这两个字符串连起来，在加号（+）处换行。例如，可以将上面的语句可以修改成如下形式：

```
System.out.println("这是第一个" +
             "Java程序！");
```

2.1.2 Java中注释

1.单行注释

单行注释用于对程序中的某一行代码进行解释，一般用来注释局部变量。

单行注释用符号“//”表示，“//”后面为被注释的内容，具体示例如下：

```
int c = 10;      // 定义一个整型变量
```

2.多行注释

多行注释顾名思义就是注释的内容可以为多行，它以符号“/*”开头，以符号“*/”结尾。多行注释具体示例如下：

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

2.1.2 Java中注释

3.文档注释

文档注释是以“/**”开头，并在注释内容末尾以“*/”结束。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。文档注释具体示例如下：

```
/**
  name = “程序员”；
 */
```

2.1.3 Java中的标识符

在编程过程中，经常需要在程序中定义一些符号标记一些名称，如包名、类名、方法名、参数名、变量名等，这些符号被称为**标识符**。标识符可以由**字母**、**数字**、**下画线**（_）和**美元符号**（$）组成，但标识符不能以数字开头，不能是Java中的关键字。

2.1.3 Java中的标识符

```
//合法标识符
username
username123
user_name
userName
$username
```

```
123username
class
98.3
Hello World
```

2.1.3 Java中的标识符

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：

（1）包名所有字母一律小写。例如：cn.itcast.test。

（2）类名和接口名每个单词的首字母都要大写。例如：ArrayList、Iterator。

（3）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。

2.1.3 Java中的标识符

（4）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。驼峰

（5）在程序中，应该尽量使用有意义的英文单词定义标识符，使得程序便于阅读。例如，使用userName定义用户名，password定义密码。

2.1.4 Java中的关键字

Java中的关键字

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

2.1.4 Java中的关键字

编写Java程序时，需要注意以下几点：

（1）所有的关键字都是小写的。

（2）不能使用关键字命名标识符。

（3）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。

（4）true、false和null虽然不属于关键字，但它们具有特殊的意义，也不能作为标识符使用。

2.1.5 Java中的常量

常量就是在程序中固定不变的值，是不能改变的数据。例如，数字1、字符'a'、浮点数3.2等都是常量。在Java中，常量包括整型常量、浮点数常量、布尔常量、字符常量等。

2.1.5 Java中的常量

1．整型常量

整型常量是整数类型的数据，有二进制、八进制、十进制和十六进制4种表示形式。

● 二进制：由数字0和1组成的数字序列。从JDK7开始，允许使用字面值表示二进制数，前面要以0b或0B开头，目的是为了和十进制进行区分，如0b01101100、0B10110101。

2.1.5 Java中的常量

- 八进制：以0开头并且其后由0~7范围内（包括0和7）的整数组成的数字序列，如0342。
- 十进制：由数字0~9范围内（包括0和9）的整数组成的数字序列，如198。
- 十六进制：以0x或者0X开头并且其后由0~9、A~F(包括0和9、A和F，字母不区分大小写)组成的数字序列，如0x25AF。

2.1.5 Java中的常量

在程序中为了标明不同的进制，数据都有特定的标识，八进制必须以0开头，如0711、0123；十六进制必须以0x或0X开头，如0xaf3、0Xff；整数以十进制表示时，第一位不能是0。例如，十进制的127，用二进制表示为0b1111111或者0B1111111，用八进制表示为0177，用十六进制表示为0x7F或者0X7F。

2.1.5 Java中的常量

2．浮点数常量

浮点数常量就是在数学中用到的小数，浮点数分为单精度浮点数（float）和双精度浮点数（double）两种类型。

其中，单精度浮点数后面以F或f结尾，而双精度浮点数则以D或d结尾。当然，在使用浮点数时也可以在结尾处不加任何的后缀，此时JVM会默认浮点数为double类型的浮点数。浮点数常量还可以通过指数形式表示。

2.1.5 Java中的常量

3．字符常量

字符常量用于表示一个字符，一个字符常量要用一对英文半角格式的单引号（''）括起来。字符常量可以是英文字母、数字、标点符号、以及由转义序列表示的特殊字符。

```
'a'
'1'
'&'
'\r'
'\u0000'
```

2.1.5 Java中的常量

4．字符串常量

字符串常量用于表示一串连续的字符，一个字符串常量要用一对英文半角格式的双引号（“ ”）括起来。

```
"HelloWorld"
"123"
"Welcome \n XXX"
""
```

2.1.5 Java中的常量

5．布尔常量

布尔常量即布尔型的值，用于区分事物的真与假。布尔常量有true和false两个值。

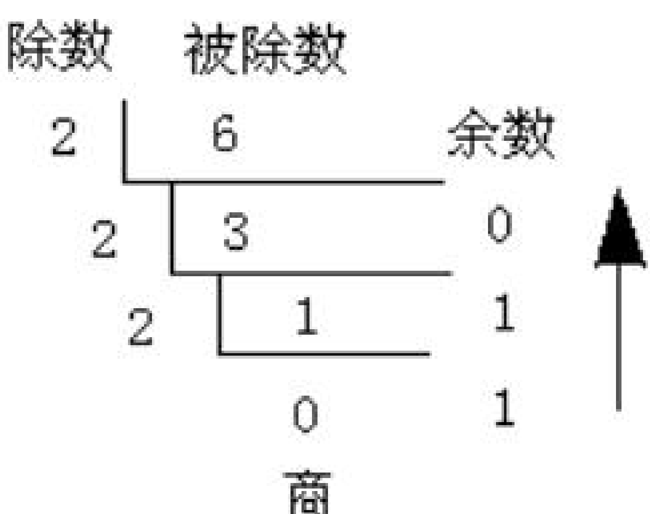
6．null常量

null常量只有一个值null，表示对象的引用为空。

🕮多学一招：十进制和二进制之间的转换

1. 十进制转二进制

十进制转换成二进制就是一个除以2取余数的过程。把要转换的数，除以2，得到商和余数，将商继续除以2，直到商为0。最后将所有余数倒序排列，得到的数就是转换结果。



📖多学一招：十进制和二进制之间的转换

十进制数据6三次除以2得到的余数依次是0、1、1，将所有余数倒序排列是110，所以十进制的6转换成二进制，结果是110。

📖多学一招：十进制和二进制之间的转换

2. 二进制转十进制

二进制转化成十进制要从右到左用二进制位上的每个数乘以2的相应次方。例如，将最右边第一位的数乘以2的0次方，第二位的数乘以2的1次方，第n位的数乘以2的n-1次方，然后把所有相乘后的结果相加，得到的结果就是转换后的十进制。

多学一招：十进制和二进制之间的转换

例如，把二进制数0110 0100转换为十进制，转换方式如下：

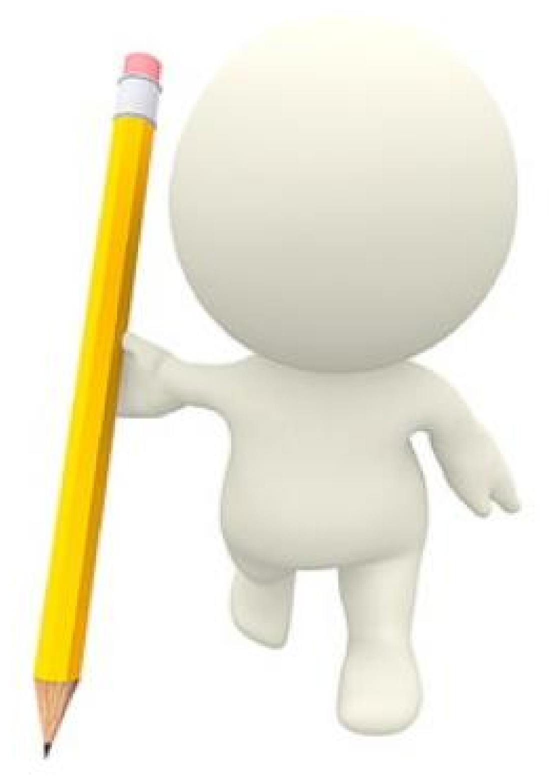
0 * 20 + 0 * 21 + 1 * 22 + 0 * 23 + 0 * 24 + 1 * 25 + 1 * 26+ 0 * 27 = 100

由于0乘以多少都是0，因此上述表达式也可以简写为：

1 * 22 + 1 * 25 + 1 * 26 = 100

得到的结果100就是二进制数0110 0100转化后的十进制数。

2.2.1 变量的定义



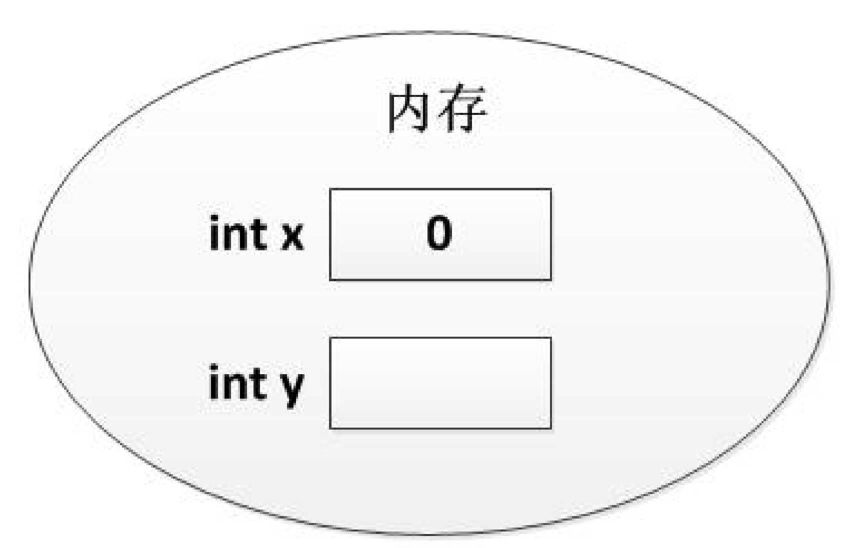
在程序运行期间，随时可能产生一些临时数据，应用程序会将这些数据保存在内存单元中，每个内存单元都用一个标识符标识，这些用于标识内存单元的标识符就称为变量，内存单元中存储的数据就是变量的值。

2.2.1 变量的定义

通过具体的代码学习变量的定义。

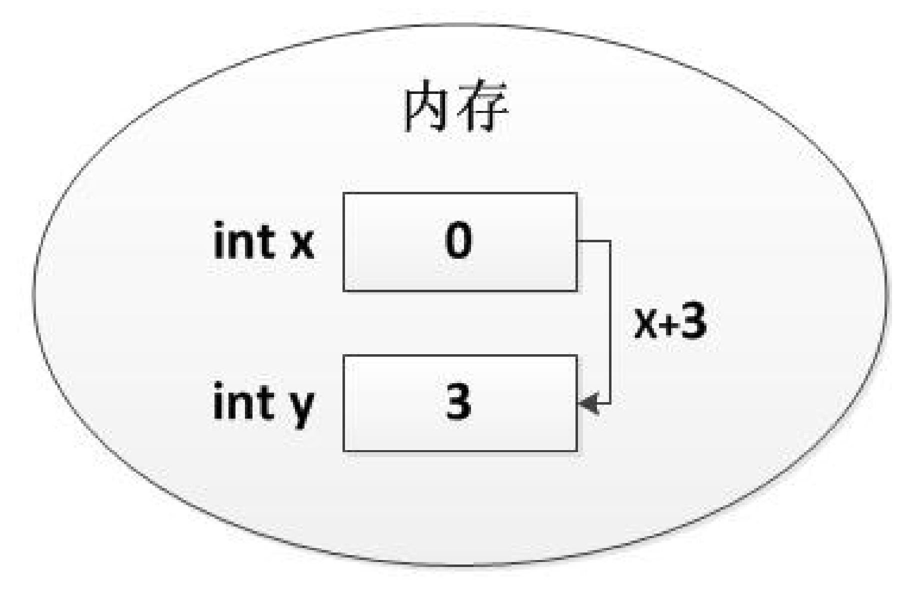
```
int x = 0,y;
y = x+3;
```

上面的代码中，第一行代码定义了两个变量x和y，也就相当于分配了两块内存单元，在定义变量的同时为变量x分配了一个初始值0，而变量y没有分配初始值，变量x和y在内存中的状态如右图。



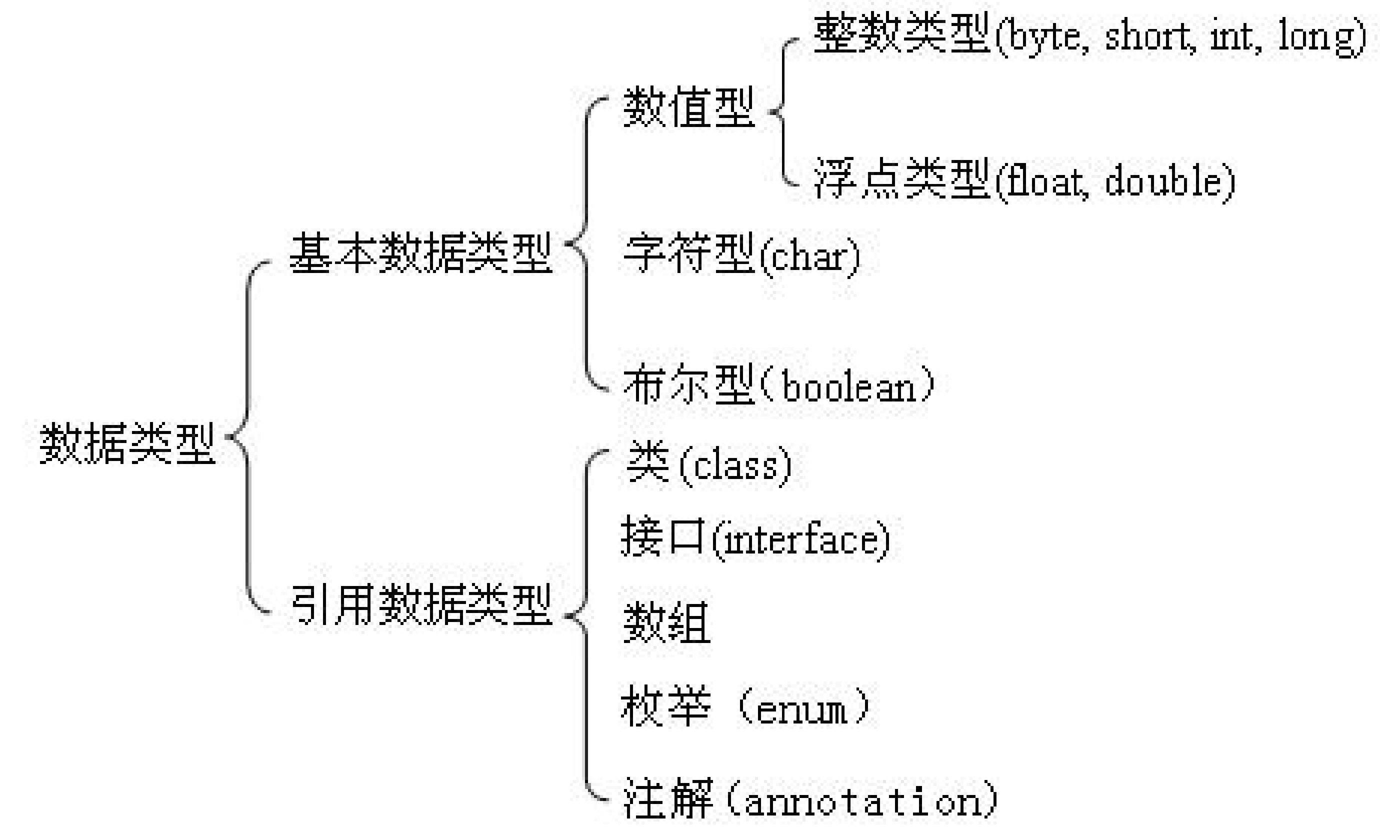
2.2.1 变量的定义

第二行代码的作用是为变量赋值，在执行第二行代码时，程序首先取出变量x的值，与3相加后，将结果赋值给变量y，此时变量x和y在内存中的状态发生了变化，如右图。



2.2.2 变量的数据类型

Java中的所有数据类型如下图。



2.2.2 变量的数据类型

1．整数类型变量

整数类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，分别是字节型（byte）、短整型（short）、整型（int）和长整型（long）。

2.2.2 变量的数据类型

类型	占用空间	取值范围
byte	8位（1个字节）	-2^{7} ～ $2^{7}-1$
short	16位（2个字节）	-2^{15} ～ $2^{15}-1$
int	32位（4个字节）	-2^{31} ～ $2^{31}-1$
long	64位（8个字节）	-2^{63} ～ $2^{63}-1$

2.2.2 变量的数据类型

在为一个long类型的变量赋值时，所赋值的后面要加上字母L（或小写l），说明赋值为long类型。如果赋的值未超出int型的取值范围，则可以省略字母L（或小写l）。

2.2.2 变量的数据类型

```
long num = 2200000000L;    // 所赋的值超出了int型的取值范围，后面必须加上字母L
long num = 198L;           // 所赋的值未超出int型的取值范围，后面可以加上字母L
long num = 198;            // 所赋的值未超出int型的取值范围，后面可以省略字母L
```

2.2.2 变量的数据类型

2．浮点数类型变量

浮点数类型变量用于存储小数数值。double类型所表示的浮点数比float类型更精确，两种浮点数所占存储空间的大小以及取值范围。

类型名	占用空间	取值范围
float	32位（4个字节）	1.4E-45 ～ 3.4E+38,-3.4E+38 ～ -1.4E-45
double	64位（8个字节）	4.9E-324 ～ 1.7E+308, -1.7E+308 ～ -4.9E-324

2.2.2 变量的数据类型

在Java中，一个小数会被默认为double类型的值，因此在为一个float类型的变量赋值时，在所赋值的后面一定要加上字母F（或者小写f），而为double类型的变量赋值时，可以在所赋值的后面加上字母D（或小写d），也可以不加。

2.2.2 变量的数据类型

```
float f = 123.4f;         // 为一个float类型的变量赋值，后面必须加上字母f
double d1 = 100.1;        // 为一个double类型的变量赋值，后面可以省略字母d
double d2 = 199.3d;       // 为一个double类型的变量赋值，后面可以加上字母d
```

在程序中也可以为一个浮点数类型变量赋予一个整数数值。

```
float f = 100;            // 声明一个float类型的变量并赋整数值
double d = 100;           // 声明一个double类型的变量并赋整数值
```

2.2.2 变量的数据类型

3．字符类型变量

在Java中，字符类型变量用char表示，用于存储一个单一字符。Java中每个char类型的字符变量都会占用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'。

2.2.2 变量的数据类型

在计算机的世界里，所有文字、数值都只是一连串的0与1，这些0与1是机器语言，人类难以理解，于是就产生了各种方式的编码，使用一个数值代表某个字符，如常用的字符编码系统ASCII。

Java使用Unicode字符码系统，Unicode为每个字符制定了一个唯一的数值，在计算时，计算机会自动将字符转化为所对应的数值，如用97表示小写英文字母a。

2.2.2 变量的数据类型

```
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
```

2.2.2 变量的数据类型

4．布尔类型变量

在Java中，使用boolean定义布尔类型变量，布尔类型变量只有true和false两个值。定义布尔类型变量，具体示例如下：

```
boolean flag = false;        // 定义一个boolean类型的变量flag，初始值为false
flag = true;                 // 改变变量flag的值为true
```

2.2.3 变量的类型转换

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的过程中不需要显式地进行声明，由编译器自动完成。自动类型转换必须同时满足两个条件，第一是两种数据类型彼此兼容，第二是目标类型的取值范围大于源类型的取值范围。

2.2.3 变量的类型转换

```
byte b = 3;
int x = b;
```

上面的代码中，使用byte类型的变量b为int类型的变量x赋值，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。

2.2.3 变量的类型转换

下面列出3种可以进行自动类型转换的情况，具体如下：

（1）整数类型之间可以实现转换。例如，byte类型的数据可以赋值给short、int、long类型的变量；short、char类型的数据可以赋值给int、long类型的变量；int类型的数据可以赋值给long类型的变量。

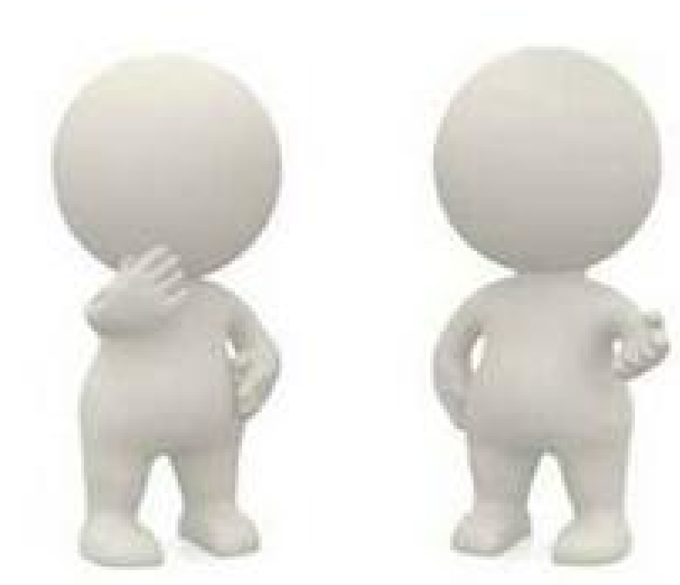
（2）整数类型转换为float类型。例如，byte、char、short、int类型的数据可以赋值给float类型的变量。

（3）其他类型转换为double类型。例如，byte、char、short、int、long、float类型的数据可以赋值给double类型的变量。

2.2.3 变量的类型转换

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进行显式地声明。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。



2.2.3 变量的类型转换

```
public class Example01 {
    public static void main(String[] args) {
        int num = 4;
        byte b = num;
        System.out.println(b);
    }
}
```

2.2.3 变量的类型转换

编译上述代码，程序报错。错误原因：数据类型不兼容，不能将int类型转换成byte类型。

解决办法：进行强制类型转换，即强制将int类型的值赋值给byte类型的变量。

强制类型转换格式：目标类型 变量 = （目标类型）值

将变量num赋值给变量b时，进行强制类型转换，代码如下：

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

2.2.3 变量的类型转换

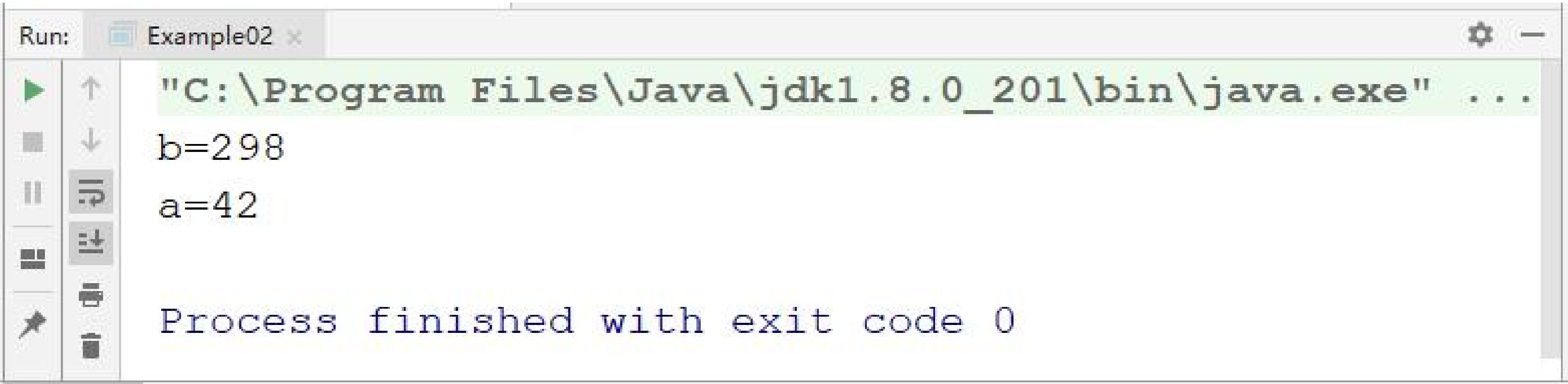
在对变量进行强制类型转换时，如果将取值范围较大的数据类型强制转换为取值范围较小的数据，如将一个int类型的数转为byte类型，极容易造成数据精度的丢失。

2.2.3 变量的类型转换

```
public class Example02 {
    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
        a = (byte) b;
        System.out.println("b=" + b);
        System.out.println("a=" + a);
    }
}
```

2.2.3 变量的类型转换

上述程序运行结果如下图。



2.2.3 变量的类型转换

从运行结果可知变量b本身的值为298，然而在赋值给变量a后，a的值为42。原因：变量b为int类型，在内存中占用4个字节；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个高位字节的数据丢失，数值发生改变。

int

00000000 00000000 00000001 00101010

byte

00101010

📖多学一招：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，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。例如，一个byte型的变量在运算期间类型会自动提升为int型。

多学一招：表达式类型自动提升

```
public class Example03 {
	public static void main(String[] args) {
		byte b1 = 3; // 定义一个byte类型的变量
		byte b2 = 4;
		// 两个byte类型变量相加，赋值给一个byte类型变量
		byte b3 = b1 + b2;
		System.out.println("b3=" + b3);
	}
}
```

📖多学一招：表达式类型自动提升

编译上述代码，程序报错，原因：表达式b1+b2运算期间，变量b1和b2都被自动提升为int型，表达式的运算结果也就成了int型，将该结果赋给byte型的变量，编译器报错。

解决办法：进行强制类型转换。

```
byte b3 = (byte) (b1 + b2);
```

2.2.4 变量的作用域

变量需要先定义后使用，但这并不意味着定义的变量，在之后所有语句中都可以使用。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。

2.2.4 变量的作用域

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

y的作用域

x 的作用域

上图中的代码有两层大括号。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。

2.2.4 变量的作用域

接下来通过一个案例进一步熟悉变量的作用域。

```
public class Example04 {
    public static void main(String[] args) {
        int x = 12;                  // 定义了变量x
        {
            int y = 96;              // 定义了变量y
            System.out.println("x is " + x);     // 访问变量x
            System.out.println("y is " + y);     // 访问变量y
        }
        y = x;                 // 访问变量x，为变量y赋值
        System.out.println("x is " + x);   // 访问变量x
    }
}
```

2.2.4 变量的作用域

编译上述代码，程序报错，

原因：变量y在作用域范围

之外被使用（y=x;)。

2.3 Java中的运算符

在程序中经常出现一些特殊符号，如+、-、*、=、>等，这些特殊符号称作运算符。运算符用于对数据进行算术运算、赋值运算和比较运算等。在Java中，运算符可分为算术运算符、赋值运算符、比较运算符、逻辑运算符等。

2.3.1 算术运算符

在数学运算中最常见的就是加减乘除，被称作四则运算。Java中的算术运算符就是用来处理四则运算的符号，算术运算符是最简单、最常用的运算符号。

运算符	运算	范例	结果
+	正号	+3	3
-	负号	b=4;-b;	-4
+	加	5+5	10
-	减	6-4	2
*	乘	3*4	12
/	除	5/5	1
%	取模（即算术中的求余数）	7%5	2
++	自增（前）	a=2;b=++a;	a=3;b=3;
++	自增（后）	a=2;b=a++;	a=3;b=2;
--	自减（前）	a=2;b=--a	a=1;b=1;
--	自减（后）	a=2;b=a--	a=1;b=2;

2.3.1 算术运算符

算术运算符在使用时需要注意的问题：

（1）在进行自增（++）和自减（--）运算时，如果运算符++或--放在操作数的前面，则先进行自增或自减运算，再进行其他运算。反之，如果运算符放在操作数的后面，则先进行其他运算再进行自增或自减运算。

2.3.1 算术运算符

```
int a = 1;
int b = 2;
int x = a + b++;
System.out.print("b=" b);
System.out.print("x=" x);
```

左边代码运行结果为：b=3、x=3。 在上述代码中，定义了三个int类型的变量a、b、x。其中a=1、b=2。当进行“a+b++”运算时，由于运算符++写在了变量b的后面，则先进行a+b运算，再进行变量b的自增，因此变量b在参与加法运算时其值仍然为2，x的值应为3。变量b在参与运算之后会进行自增，因此b的最终值为3。

2.3.1 算术运算符

（2）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。例如，2510/1000属于整数之间相除，会忽略小数部分，得到的结果是2，而2.5/10的结果为0.25。

2.3.1 算术运算符

（3）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。例如，(-5)%3=-2，而5%(-3)=2。

2.3.2 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给某一个变量。

运算符	运算	范例	结果
=	赋值	a=3;b=2;	a=3;b=2;
+=	加等于	a=3;b=2;a+=b;	a=5;b=2;
-=	减等于	a=3;b=2;a-=b;	a=1;b=2;
*=	乘等于	a=3;b=2;a*=b;	a=6;b=2;
/=	除等于	a=3;b=2;a/=b;	a=1;b=2;
%=	模等于	a=3;b=2;a%=b;	a=1;b=2;

2.3.2 赋值运算符

赋值运算符使用时的注意事项。

（1）在Java中可以通过一条赋值语句对多个变量进行赋值，具体示例如下：

```
int  x, y, z;
x = y = z = 5;          // 为三个变量同时赋值
```

在上述代码中，一条赋值语句将变量x，y，z的值同时赋值为5。需要注意的是，下面的这种写法在Java中是不可以的。

```
int  x = y = z = 5;          // 这样写是错误的
```

2.3.2 赋值运算符

（2）在表2-4中，除了“=”，其他的都是特殊的赋值运算符，以“+=”为例，x += 3就相当于x = x + 3，表达式首先会进行加法运算x+3，再将运算结果赋值给变量x。-=、*=、/=、%=赋值运算符都可以此类推。

📖多学一招：强制类型转换

在为变量赋值时，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，需要进行强制类型转换。例如，将一个int类型的值赋给一个short类型的变量，需要进行强制类型转换。然而在使用+=、-=、*=、/=、%= 运算符进行赋值时，强制类型转换会自动完成，程序不需要做任何显式地声明。

多学一招：强制类型转换

```
public class Example05 {
    public static void main(String[] args) {
        short s = 3;
        int i = 5;
        s += i;     //强制类型转换自动完成
        System.out.println("s = " + s);
    }
}
```

2.3.3 比较运算符

比较运算符用于对两个数值或变量进行比较，比较运算结果是一个布尔值，即true或false。

运算符	运算	范例	结果
==	相等于	4 == 3	false
!=	不等于	4 != 3	true
<	小于	4 < 3	false
>	大于	4 > 3	true
<=	小于等于	4 <= 3	false
>=	大于等于	4 >= 3	true

2.3.3 比较运算符

在比较运算中，不能将比较运算符“==”误写成赋值运算符“=”。

2.3.4 逻辑运算符

逻辑运算符用于对布尔型的数据进行操作，其结果仍是一个布尔值。

运算符	运算	范例	结果
&	与	true & true	true
		true & false	false
		false & false	false
		false &true	false
|	或	true | true	true
		true | false	true
		false| false	false
		false| true	true
^	异或	true ^ true	false
		true ^ false	true
		false ^ false	false
		false ^ true	true
!	非	!true	false
		!false	true
&&	短路与	true && true	true
		true && false	false
		false && false	false
		false && true	false
||	短路或	true || true	true
		true || false	true
		false|| false	false
		false|| true	true

2.3.4 逻辑运算符

使用逻辑运算符需要注意的问题。

（1）逻辑运算符可以针对结果为布尔值的表达式进行运算。例如，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。虽然运算符“&”和“&&”都表示与操作，但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。在使用“&&”进行运算，当左边为false时，右边的表达式就不再进行运算，因此“&&”被称作短路与。

2.3.4 逻辑运算符

```
1 public class Example06 {
2         public static void main(String[] args) {
3                 int x = 0;                      // 定义变量x，初始值为0
4                 int y = 0;                      // 定义变量y，初始值为0
5                 int z = 0;                      // 定义变量z，初始值为0
6                 boolean a, b;                   // 定义boolean变量a和b
7                 a = x > 0 & y++ > 1;            // 逻辑运算符&对表达式进行运算
8                 System.out.println(a);
9                 System.out.println("y = " + y);
10                b = x > 0 && z++ > 1;           // 逻辑运算符&&对表达式进行运算
11                System.out.println(b);
12                System.out.println("z = " + z);
13        }
14 }
```

2.3.4 逻辑运算符

上述代码中，第3~5行代码定义了三个整型变量x、y、z，初始值都为0；第6行代码定义了两个布尔类型的变量a和b。第7行代码使用“&”运算符对两个表达式进行逻辑运算，左边表达式x>0的结果为false，这时无论右边表达式y++>1的比较结果是什么，整个表达式x > 0 & y++ > 1的结果都会是false。由于使用的是运算符“&”，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，整个表达式运算结束之后，y的值为1。

2.3.4 逻辑运算符

第10行代码是逻辑“&&”运算，运算结果和第7行代码一样为false，区别在于，第10行代码使用了短路与“&&”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。

2.3.4 逻辑运算符

（3）运算符“|”和“||”都表示或操作，当运算符两边的任一表达式值为true时，其结果为true。只有两边表达式的值都为false时，其结果才为false。同逻辑与操作类似，“||”运算符为短路或，当运算符“||”的左边为true时，右边的表达式不再进行运算。

2.3.4 逻辑运算符

```
int x = 0;
int y = 0;
boolean b = x==0 || y++>0
```

上面的代码块执行完毕后，b的值为true，y的值仍为0。原因是运算符“||”的左边表达式x==0结果为true，那么右边表达式将不进行运算，y的值不发生任何变化。

2.3.4 逻辑运算符

（4）运算符“^”表示异或操作，当运算符两边的布尔值相同时（都为true或都为false），其结果为false。当两边表达式的布尔值不相同时，其结果为true。

2.3.5 运算符优先级

在对一些比较复杂的表达式进行运算时，要明确表达式中所有运算符参与运算的先后顺序，通常把这种顺序称作运算符的优先级。

<table>
<tr><th>优先级</th><th>运算符</th></tr>
<tr><td>1</td><td>. [] ()</td></tr>
<tr><td>2</td><td>++ -- ~ ! (数据类型)</td></tr>
<tr><td>3</td><td>* / %</td></tr>
<tr><td>4</td><td>+ -</td></tr>
<tr><td>5</td><td><< >> >>></td></tr>
<tr><td>6</td><td>< > <= >=</td></tr>
<tr><td>7</td><td>== !=</td></tr>
<tr><td>8</td><td>&</td></tr>
<tr><td>9</td><td>^</td></tr>
<tr><td>10</td><td>|</td></tr>
<tr><td>11</td><td>&&</td></tr>
<tr><td>12</td><td>||</td></tr>
<tr><td>13</td><td>?:</td></tr>
<tr><td>14</td><td>= *= /= %= += -= <<= >>= >>>= &= ^= |=</td></tr>
</table>

2.3.5 运算符优先级

在上表中，运算符前面的数字越小优先级越高。根据运算符优先级，分析下面代码的运行结果。

```
int a =2;
int b = a + 3*a;
System.out.println(b);
```

上述代码运行结果为8，由于运算符“*”的优先级高于运算符“+”，因此先运算3*a，得到的结果是6，再将6与a相加，得到最后的结果8。

2.3.5 运算符优先级

```
int a =2;
int b = (a+3) * a;
System.out.println(b);
```

上述代码运行结果为10，由于运算符“()”的优先级最高，因此先运算括号内的a+3，得到的结果是5，再将5与a相乘，得到最后的结果10。

【案例2-1】 商品入库

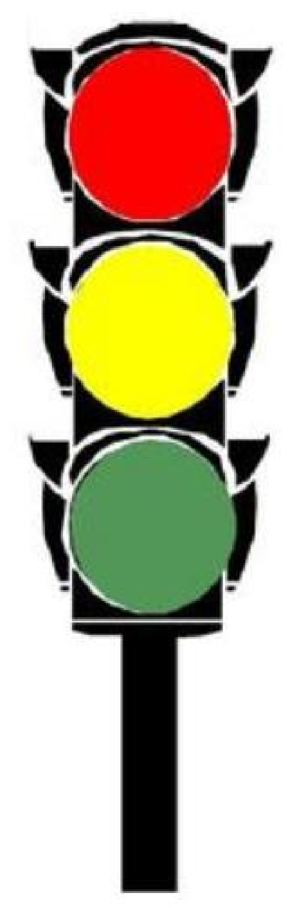
现要对华为和小米两种手机产品进行入库，本案例要求编写一个模拟商品入库的程序，可以在控制台输入入库商品的数量，最后打印出仓库中所有商品详细信息以及所有商品的总库存数和库存商品总金额。

商品信息如下：

- 品牌型号
- 尺寸
- 价格
- 配置
- 库存
- 总价

2.4 选择结构语句

2.4 选择结构语句



在实际生活中经常需要做出一些判断，比如开车来到一个十字路口，就需要对红绿灯进行判断，如果前面是红灯，就停车等候，如果是绿灯，就通行。Java中有一种特殊的语句叫做选择语句，它也需要对一些条件做出判断，从而决定执行哪一段代码。选择语句分为if条件语句和switch条件语句。

2.4.1 if条件语句

1．if语句

if语句是指如果满足某种条件，就进行某种处理。例如，小明妈妈跟小明说“如果你考试得了100分，星期天就带你去游乐场玩”。这句话可以通过下面的一段伪代码来描述。

如果小明考试得了100分

        妈妈星期天带小明去游乐场

2.4.1 if条件语句

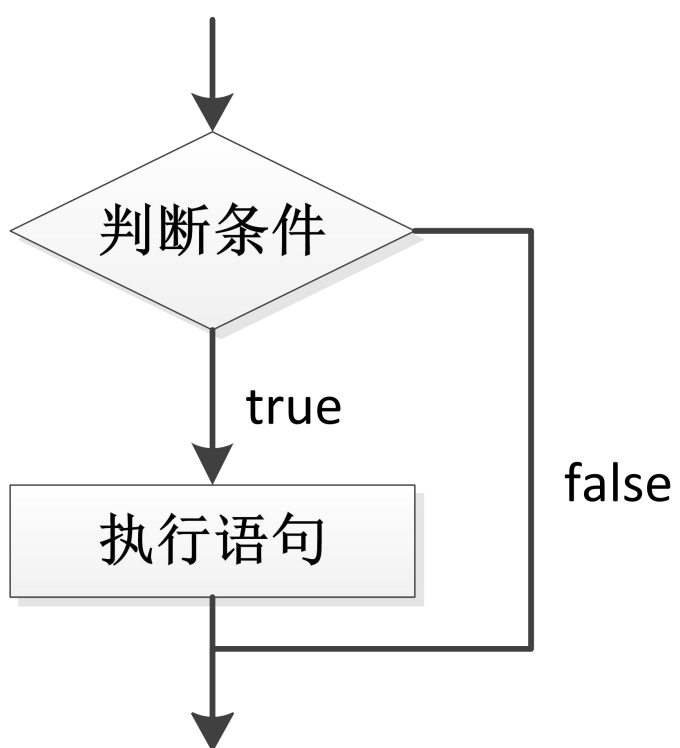
在上面的伪代码中，“如果”相当于Java中的关键字if，“小明考试得了100分”是判断条件，需要用()括起来，“妈妈星期天带小明去游乐场”是执行语句，需要放在{}中。修改后的伪代码如下：

```
if (小明考试得了100分) {
        妈妈星期天带小明去游乐场
}
```

2.4.1 if条件语句

通过上面的伪代码可以总结出if语句的格式：

```
if (条件语句)
{
        代码块
}
```



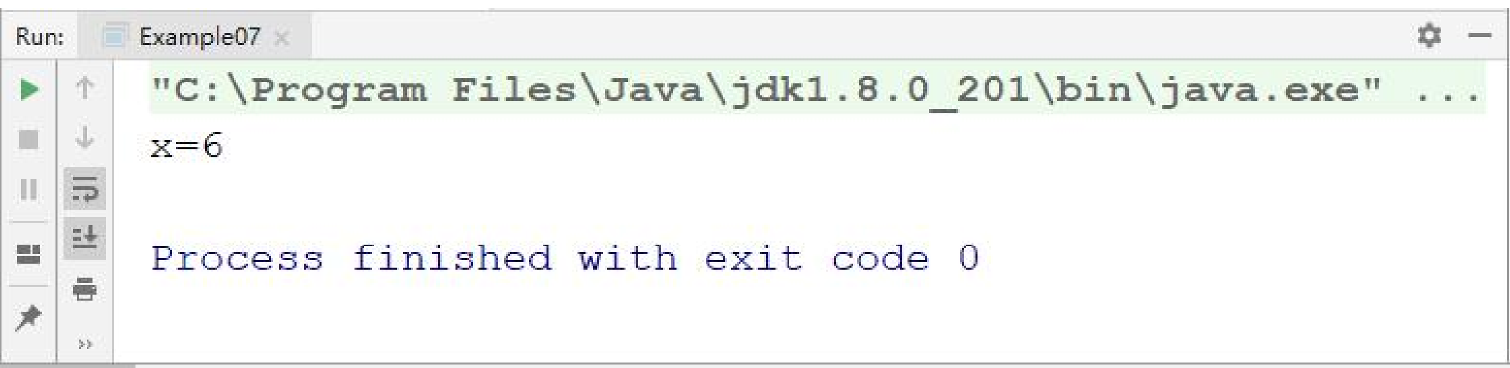
2.4.1 if条件语句

接下来通过一个案例学习if语句的具体用法。

```
1 public class Example07 {
2         public static void main(String[] args) {
3                 int x = 5;
4                 if (x < 10) {
5                         x++;
6                 }
7                 System.out.println("x=" + x);
8         }
9 }
```

2.4.1 if条件语句

案例运行结果如下图。



2.4.1 if条件语句

上述代码中，第3行代码定义了一个变量x，初始值为5。第4~6行代码在if语句中判断x的值是否小于10，如果x小于10，就执行x++。由于x值为5，x<10条件成立，{}中的语句会被执行，变量x的值进行自增。从运行结果可以看出，x的值已由原来的5变成了6。

2.4.1 if条件语句

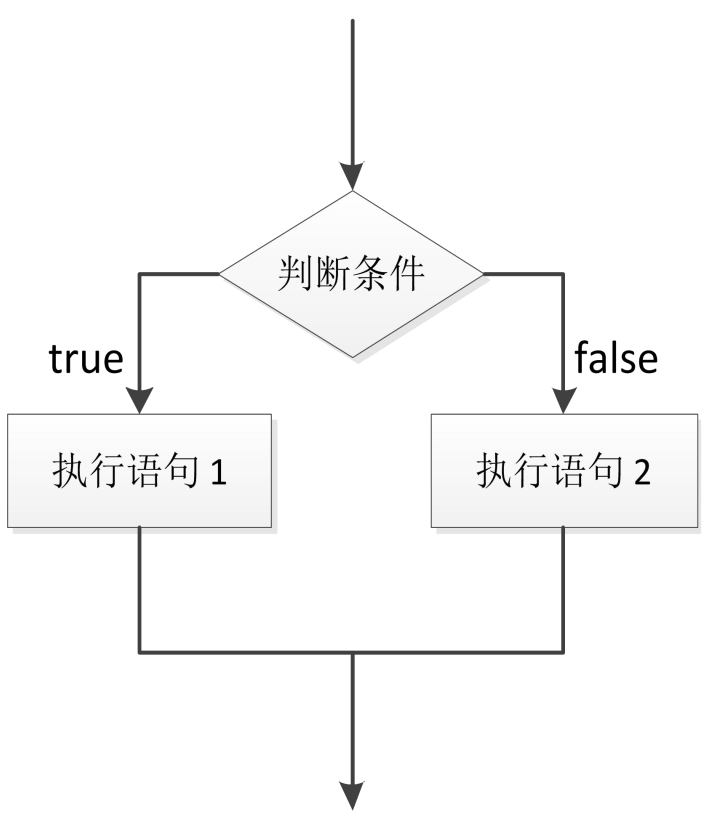
2．if...else语句

if...else语句是指如果满足某种条件，就进行某种处理，否则就进行另一种处理。例如，要判断一个正整数的奇偶，如果该数字能被2整除则是一个偶数，否则该数字就是一个奇数。

2.4.1 if条件语句

if…else语句具体语法格式如下：

```
if (判断条件){
        执行语句1
        …
}
else{
        执行语句2
        …
}
```



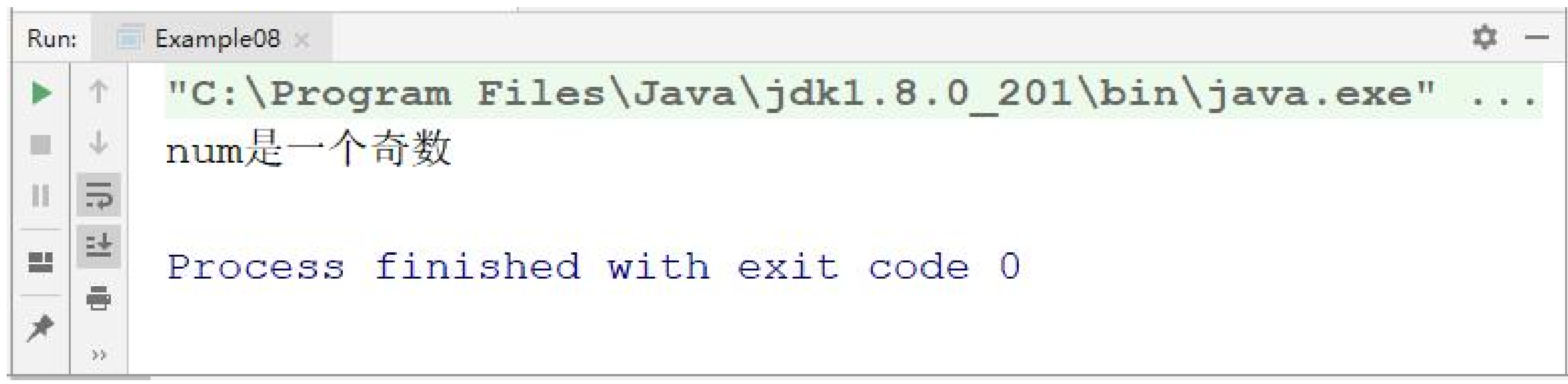
2.4.1 if条件语句

接下来通过一个案例实现判断奇偶数的程序。

```
1 public class Example08 {
2       public static void main(String[] args) {
3           int num = 19;
4           if (num % 2 == 0) {
5               // 判断条件成立，num被2整除
6               System.out.println("num是一个偶数");
7           } else {
8               System.out.println("num是一个奇数");
9               }
10      }
11 }
```

2.4.1 if条件语句

程序运行结果如下图。



2.4.1 if条件语句

上述代码中，第3行代码定义了变量num，num的初始值为19；第4~9行代码判断num%2的值是否为0，如果为0则输出“num是一个偶数”，否则输出“num的值是一个奇数”。由于num的值为19，与2取模的结果为1，不等于0，判断条件不成立。因此程序执行else后面{}中的语句，打印“num是一个奇数”。

2.4.1 if条件语句

3．if…else if…else语句

if…else if…else语句用于对多个条件进行判断，进行多种不同的处理。例如，对一个学生的考试成绩进行等级划分，如果分数大于80分，则等级为优；如果分数大于70分，则等级为良；如果分数大于60分，则等级为中；如果分数小于60分，则等级为差。

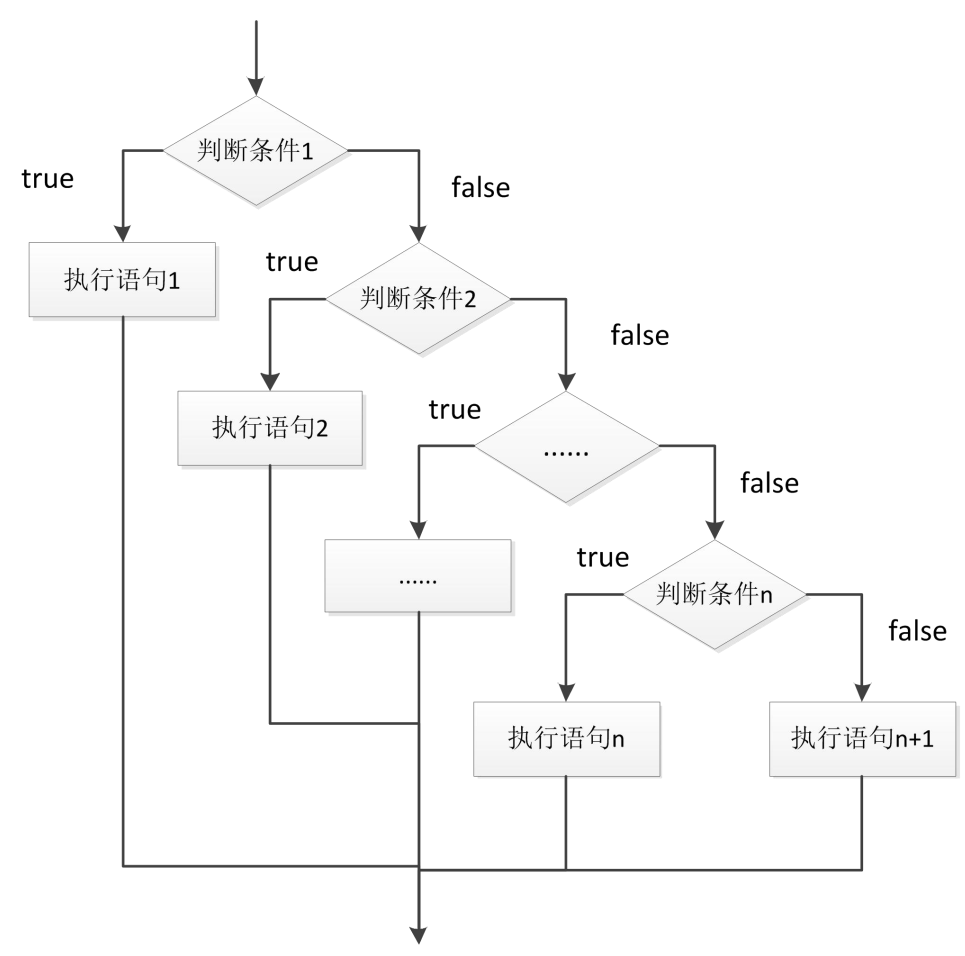
2.4.1 if条件语句

if…else if…else语句格式：

```
if (判断条件1) {
        执行语句1
}
else if (判断条件2) {
        执行语句2
}
...
else if (判断条件n) {
        执行语句n
}
else {
        执行语句n+1
}
```

2.4.1 if条件语句

上述格式中，判断条件是一个布尔值。当判断条件1为true时，if后面{}中的执行语句1会执行。当判断条件1为false时，会继续执行判断条件2，如果判断条件2为true则执行语句2…，以此类推，如果所有的判断条件都为false，则意味着所有条件均不满足，else后面{}中的执行语句n+1会执行。



2.4.1 if条件语句

接下来通过一个案例演示if…else if…else语句的用法，该案例实现对学生考试成绩进行等级划分的程序。

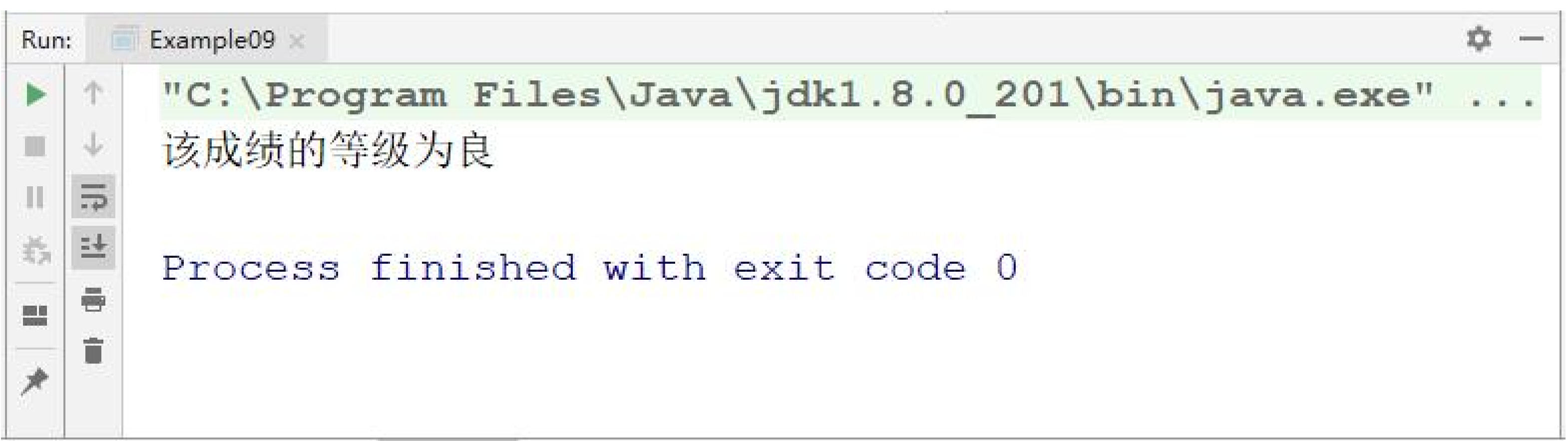
```
1 public class Example09 {
2         public static void main(String[] args) {
3                 int grade = 75;              // 定义学生成绩
4                 if (grade > 80) {
5                         // 满足条件 grade > 80
6                         System.out.println("该成绩的等级为优");
7                 } else if (grade > 70) {
```

2.4.1 if条件语句

```
8                          // 不满足条件 grade > 80 ，但满足条件 grade > 70
9                          System.out.println("该成绩的等级为良");
10                 } else if (grade > 60) {
11                         // 不满足条件 grade > 70 ，但满足条件 grade > 60
12                         System.out.println("该成绩的等级为中");
13                 } else {
14                         // 不满足条件 grade > 60
15                         System.out.println("该成绩的等级为差");
16                 }
17         }
18 }
```

2.4.1 if条件语句

程序运行结果如下图。



2.4.2 三元运算符

Java提供了一个三元运算符，可以同时操作三个表达式。三元运算符语法格式如下所示：

判断条件 ? 表达式1 : 表达式2

在上述语法格式中，当判断条件成立时，计算表达式1的值作为整个表达式的结果，否则计算表达式2的值作为整个表达式的结果。

2.4.2 三元运算符

三元运算符比if…else语句更简洁。例如，求两个数x、y中的较大者，分别使用if…else语句和三元运算符实现。

```
int x = 0;
int y = 1;
int max;
if (x > y) {
        max = x;
} else {
        max = y;}
System.out.println(max);
```

```
int x = 0;
int y = 1;
max = x > y? x : y;
System.out.println(max);
```

2.4.2 三元运算符

使用三元运算符时需要注意以下几点：

（1）条件运算符“？”和“：”是一对运算符，不能分开单独使用。

（2）条件运算符的优先级低于关系运算符与算术运算符，但高于赋值运算符。

（3）条件运算符可以进行嵌套，结合方向自右向左。例如，a>b?a:c>d?c:d应该理解为a>b?a:(c>d?c:d)，这也是条件运算符的嵌套情形，即三元表达式中的表达式2又是一个三元表达式。

2.4.3 switch条件语句

switch 条件语句也是一种很常用的选择语句，和if条件语句不同，它只能针对某个表达式的值做出判断，从而决定程序执行哪一段代码。例如，在程序中使用数字1~7表示星期一到星期天，如果想根据输入的数字输出对应中文格式的星期值，可以通过右边的一段伪代码来描述：

用于表示星期的数字

如果等于1，则输出星期一

如果等于2，则输出星期二

如果等于3，则输出星期三

如果等于4，则输出星期四

如果等于5，则输出星期五

如果等于6，则输出星期六

如果等于7，则输出星期天

2.4.3 switch条件语句

上述伪代码，如果使用if...else语句实现，则代码太过繁琐，不便于阅读。为此，Java提供了switch语句实现这种需求，switch语句使用switch关键字描述一个表达式，使用case关键字描述和表达式结果比较的目标值，当表达式的值和某个目标值匹配时，就执行对应case下的语句。

```
switch (表达式){
case 目标值1:
        执行语句1
        break;
case 目标值2:
        执行语句2
        break;
 . . . . . .
case 目标值n:
        执行语句n
        break;
default:
        执行语句n+1
        break;    }
```

2.4.3 switch条件语句

switch语句将表达式的值与每个case中的目标值进行匹配，如果找到了匹配的值，则执行对应case后面的语句；如果没找到任何匹配的值，则执行default后的语句。switch语句中的break关键字将在后面的小节中做具体介绍，此处，初学者只需要知道break的作用是跳出switch语句即可。

2.4.3 switch条件语句

接下来通过一个案例来演示switch语句的用法，在该案例中，使用switch语句根据给出的数值输出对应中文格式的星期。

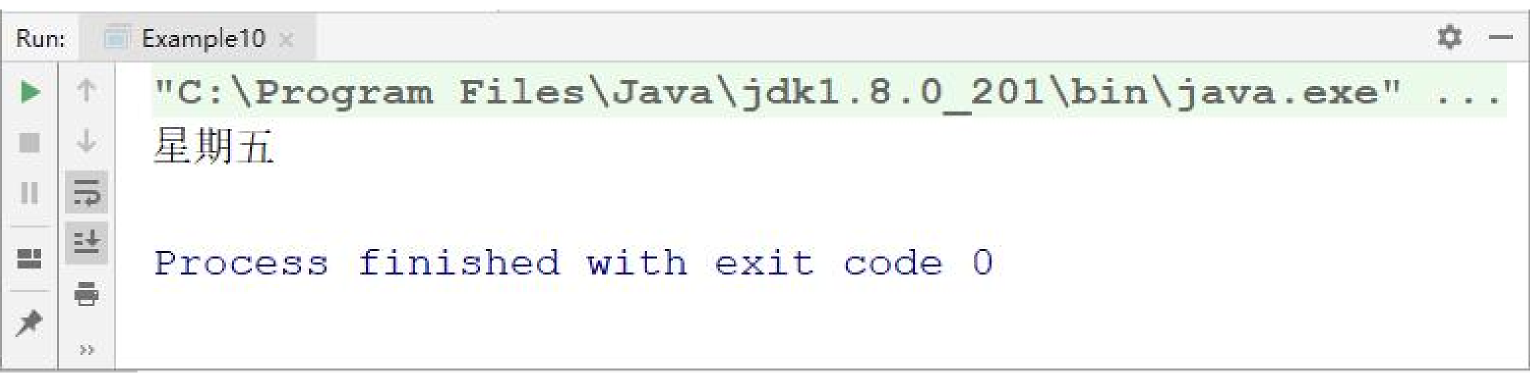
```
1 public class Example10{
2           public static void main(String[] args) {
3                   int week = 5;
4                   switch (week) {
5                   case 1:
6                           System.out.println("星期一");
7                                   break;
8                   case 2:
9                           System.out.println("星期二");
10                                  break;
11                  case 3:
12                          System.out.println("星期三");
13                                  break;
```

2.4.3 switch条件语句

```
14 case 4:
15          System.out.println("星期四");
16          break;
17 case 5:
18          System.out.println("星期五");
19          break;
20 case 6:
21          System.out.println("星期六");
22          break;
23 case 7:
24          System.out.println("星期天");
25          break;
26 default:
27          System.out.println("输入的数字不正确...");
28          break;
29          }
30     }
31  }
```

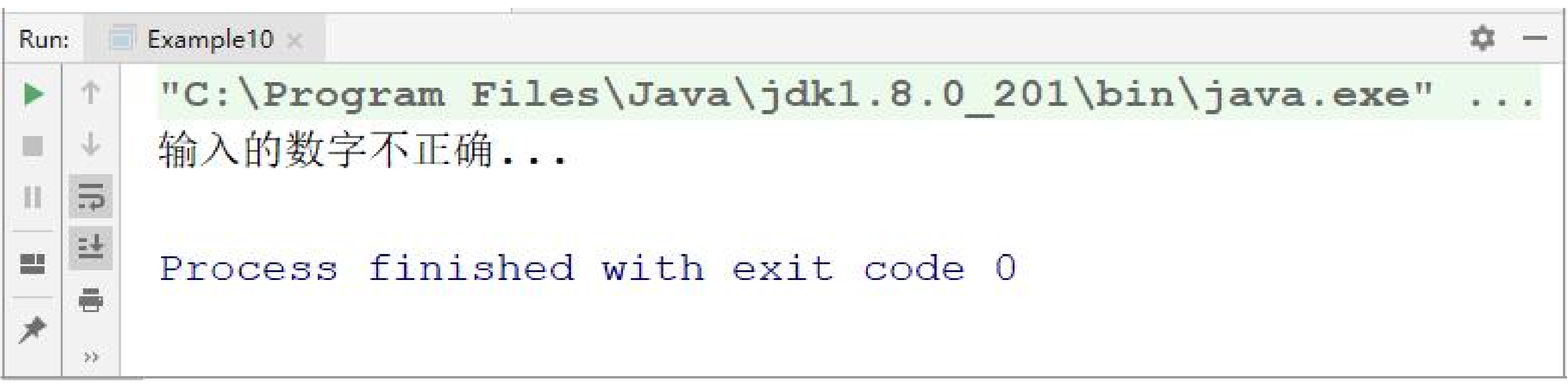
2.4.3 switch条件语句

上述程序运行结果如下图。



2.4.3 switch条件语句

将第3行代码替换为int week = 8，再次运行程序，则输出结果如下图。



2.4.3 switch条件语句

在使用switch语句时，如果多个case条件后面的执行语句是一样的，则执行语句只需书写一次即可。例如，要判断一周中的某一天是否为工作日，同样使用数字1~7表示星期一到星期天，当输入的数字为1、2、3、4、5时就视为工作日，否则就视为休息日。

2.4.3 switch条件语句

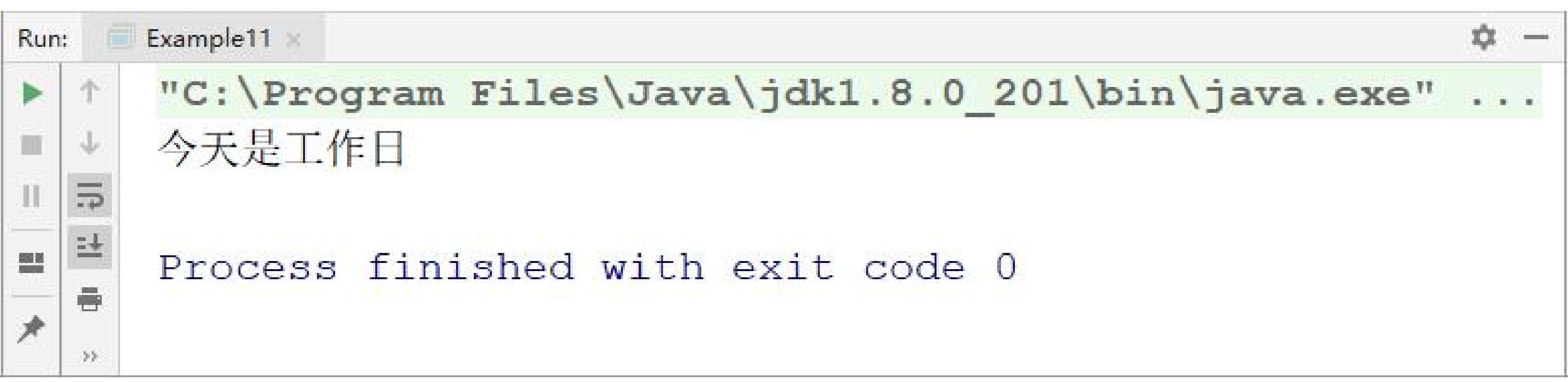
```
public class Example11 {
    public static void main(String[] args) {
        int week = 2;
        switch (week) {
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
```

2.4.3 switch条件语句

```
        // 当 week 满足值 1、2、3、4、5 中任意一个时，处理方式相同
                    System.out.println("今天是工作日");
                    break;
                case 6:
                case 7:
        // 当 week 满足值 6、7 中任意一个时，处理方式相同
                    System.out.println("今天是休息日");
                    break;
            }
        }
}
```

2.4.3 switch条件语句

上述程序运行结果如下图。



2.4.3 switch条件语句

在JDK5之前，switch语句中的表达式只能是byte、short、char、int类型的值，如果传入其他类型的值，程序会报错。JDK5引入了新特性，enum枚举可以作为switch语句表达式的值；JDK7中也引入了新特性，switch语句可以接收一个String类型的值。

【案例2-2】 小明都可以买什么

编写一个智能购物计算小程序，在一家商店有书本、铅笔、橡皮、可乐、零食五种商品，商品价格如下表所示。假如小明带了20元，且必须购买一本书，剩余的钱还可以购买哪种商品，可以购买几件，购买完后又能剩余多少钱？

商品名称	价格
书本	12元
铅笔	1元
橡皮	2元
可乐	3元
零食	5元

2.5.1 while循环语句

while循环语句与选择结构语句类似，都是根据判断条件决定是否执行大括号内的执行语句。区别在于，while语句会反复地进行条件判断，只要条件成立，{}内的执行语句就会执行，直到条件不成立，while循环结束。

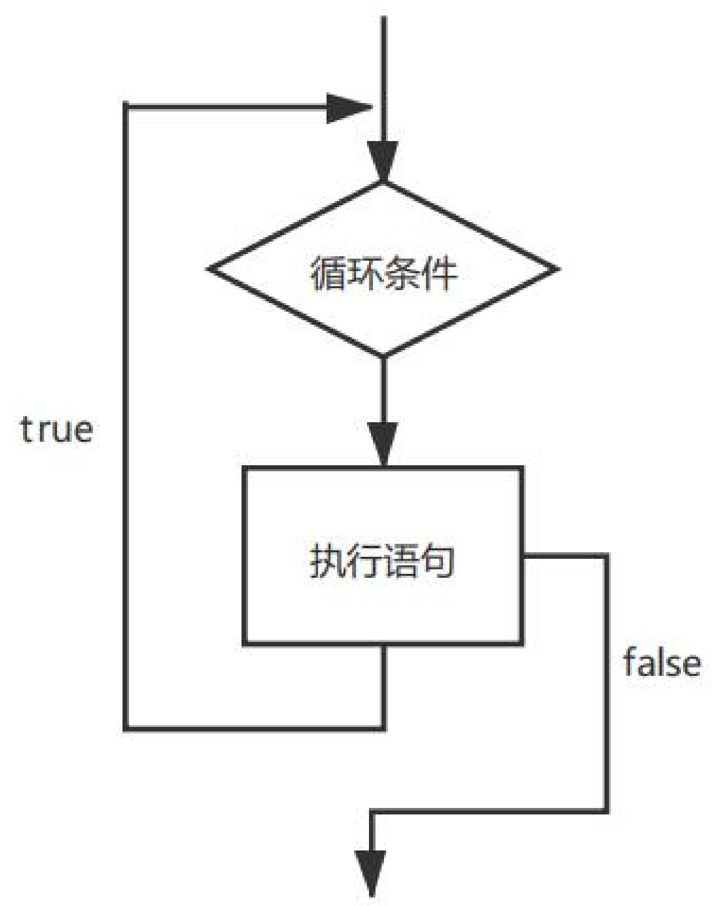
2.5.1 while循环语句

while循环语句的语法结构如下：

```
while(循环条件){
执行语句
...
}
```

2.5.1 while循环语句

在上面的语法结构中，{}中的执行语句被称作循环体，循环体是否执行取决于循环条件。当循环条件为true时，循环体就会执行。循环体执行完毕，程序继续判断循环条件，如果条件仍为true则继续执行循环体，直到循环条件为false时，整个循环过程才会结束。



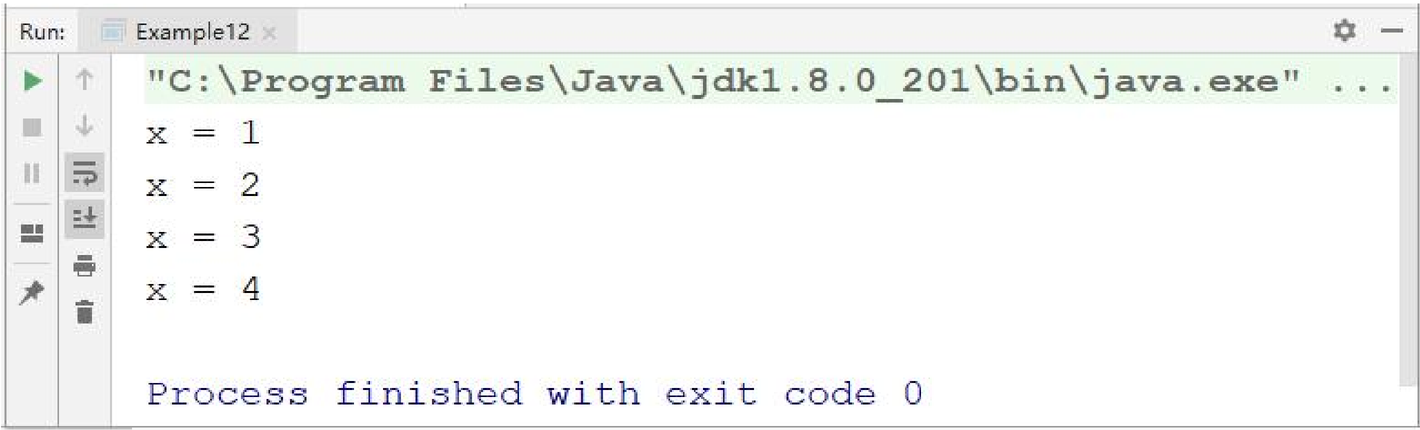
2.5.1 while循环语句

接下来通过打印1~4之间的自然数演示while循环语句的用法。

```
1public class Example12 {
2        public static void main(String[] args) {
3                int x = 1;                        // 定义变量x，初始值为1
4                while (x <= 4) {                  // 循环条件
5                        System.out.println("x = " + x);   // 条件成立，打印x的值
6                        x++;                      // x进行自增
7                }
8        }
9 }
```

2.5.1 while循环语句

程序运行结果如下图。



2.5.1 while循环语句

上述代码中，第3行代码定义了变量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自增。由运行结果可知，打印结果中x的值分别为1、2、3、4。需要注意的是，文件中第6行代码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，x的值一直为1，整个循环会进入无限循环的状态，永远不会结束。

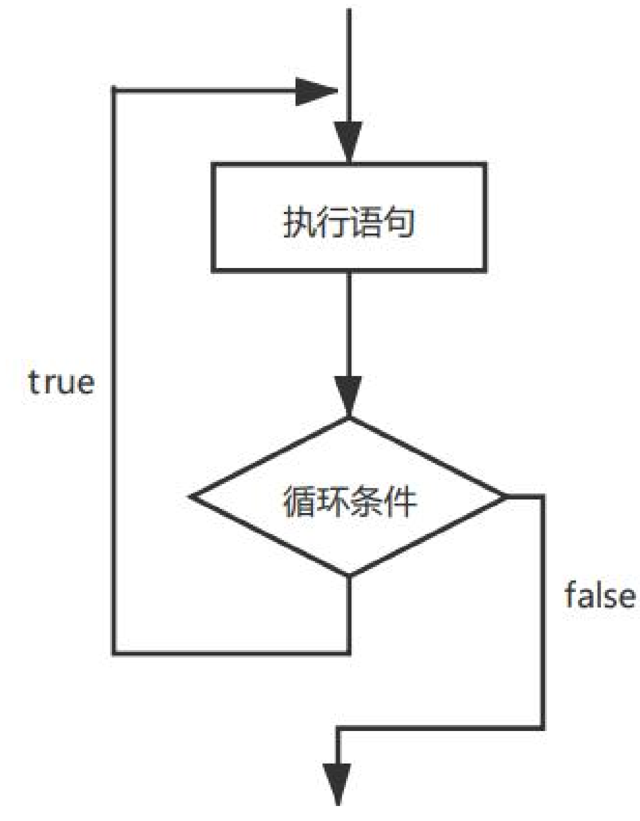
2.5.2 do...while循环语句

do…while循环语句和while循环语句功能类似，语法结构如下：

```
do {
执行语句
...
} while(循环条件);
```

2.5.2 do...while循环语句

在上面的语法结构中，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决定是否继续执行。



2.5.2 do...while循环语句

接下来使用do…while循环语句输出1~4之间的自然数。

```
1public class Example13 {
2       public static void main(String[] args) {
3              int x = 1;                     // 定义变量x，初始值为1
4              do {
5                     System.out.println("x = " + x); // 打印x的值
6                     x++;                    // 将x的值自增
7              } while (x <= 4);              // 循环条件
8       }
9 }
```

2.5.2 do...while循环语句

程序运行结果如下图。

```
Run:  Example13
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...
x = 1
x = 2
x = 3
x = 4

Process finished with exit code 0
```

2.5.3 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循环语句的语法格式如下：

```
for（初始化表达式; 循环条件; 操作表达式）
{
	执行语句
	…
}
```

2.5.3 for循环语句

在上面的语法格式中，for关键字后面()中包括了三部分内容：初始化表达式、循环条件和操作表达式，它们之间用（；）分隔，{}中的执行语句为循环体。接下来分别用①表示初始化表达式，②表示循环条件，③表示操作表达式，④表示循环体，通过序号分析for循环的执行流程。具体如下：

```
for（①；②；③）{
        ④
}
```

2.5.3 for循环语句

第一步，执行①

第二步，执行②，如果判断结果为true，执行第三步，如果判断结果为false，执行第五步

第三步，执行④

第四步，执行③，然后重复执行第二步

第五步，退出循环

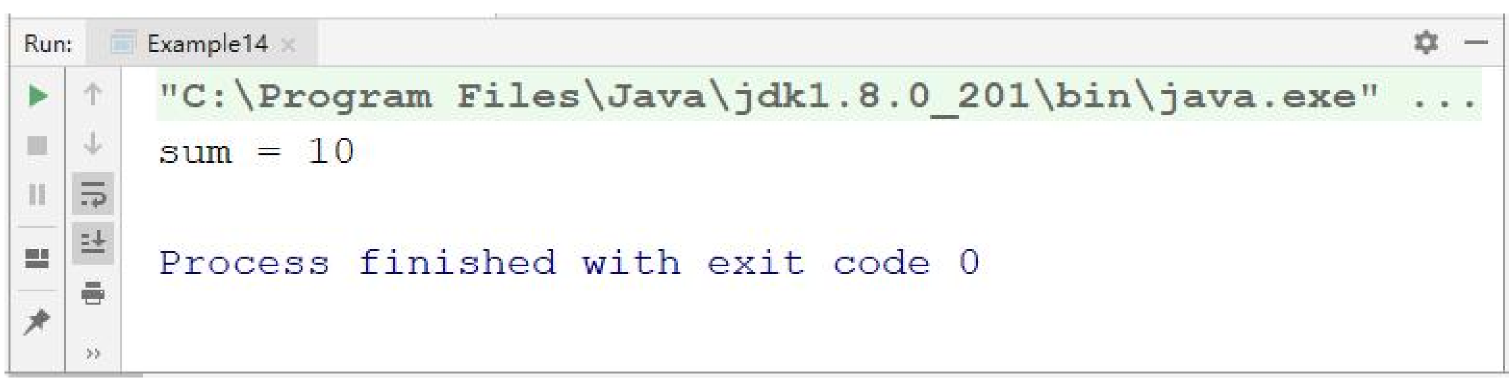
2.5.3 for循环语句

接下来对自然数1~4求和演示for循环的使用。

```
1 public class Example14 {
2 	public static void main(String[] args) {
3 		int sum = 0;			// 定义变量sum，用于记住累加的和
4 		for (int i = 1; i <= 4; i++) {	// i的值会在1~4之间变化
5 			sum += i;		// 实现sum与i的累加
6 		}
7 		System.out.println("sum = " + sum); // 打印累加的和
8 	}
9 }
```

2.5.3 for循环语句

程序运行结果如下图。



2.5.3 for循环语句

在上述程序中，变量i的初始值为1，在判断条件i＜＝4结果为true的情况下，执行循环体sum＋＝i；执行完毕后，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，判断条件i＜＝4结果为false，循环结束，执行for循环后面的代码，打印“sum＝10”。

2.5.3 for循环语句

每次循环中变量sum和i的值的变化如下表。

循环次数	i	sum
第一次	1	1
第二次	2	3
第三次	3	6
第四次	4	10

2.5.4 循环嵌套

循环嵌套是指在一个循环语句的循环体中再定义一个循环语句的语法结构。while、do…while、for循环语句都可以进行嵌套，并且它们之间也可以互相嵌套，其中最常见的是在for循环中嵌套for循环，格式如下：

```
for(初始化表达式; 循环条件; 操作表达式) {
        ...
        for(初始化表达式; 循环条件; 操作表达式) {
                执行语句
                ...
        }
        ...
}
```

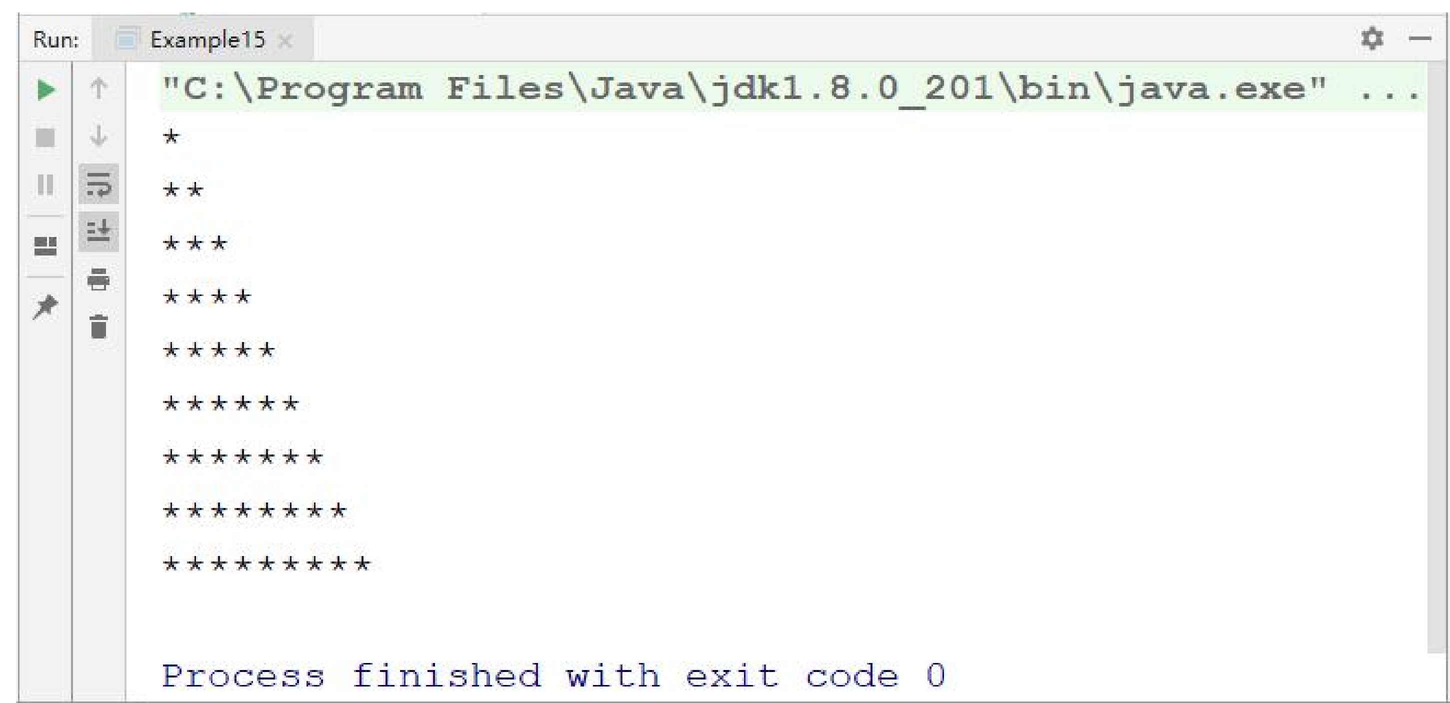
2.5.4 循环嵌套

接下来通过使用“*”打印直角三角形演示for循环嵌套的使用。

```
1 public class Example15 {
2         public static void main(String[] args) {
3                 int i, j;                                    // 定义两个循环变量
4                 for (i = 1; i <= 9; i++) {                   // 外层循环
5                         for (j = 1; j <= i; j++) {           // 内层循环
6                                 System.out.print("*");       // 打印*
7                         }
8                         System.out.print("\n");      // 换行
9                 }
10        }
11 }
```

2.5.4 循环嵌套

程序运行结果如下图。



2.5.4 循环嵌套

程序运行结果如下图。

下面分步骤讲解循环过程，具体如下：

第一步，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。

第二步，第4行代码将i初始化为1，判断条件为i <= 9为true，首次进入外层循环的循环体。

第三步，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，首次进入内层循环的循环体，打印一个“*”。

2.5.4 循环嵌套

第四步，执行第5行代码中内层循环的操作表达式j++，将j的值自增为2。

第五步，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。执行第8行代码，打印换行符。

第六步，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。

第七步，执行第4行代码中的判断条件i<=9，判断结果为true，进入外层循环的循环体，继续执行内层循环。

2.5.4 循环嵌套

第八步，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。

第九步，以此类推，在第3行会打印3个“*”，逐行递增，直到i的值为10时，外层循环的判断条件i <= 9结果为false，外层循环结束，整个循环也就结束了。

2.5.5 跳转语句（break、continue)

1．break语句

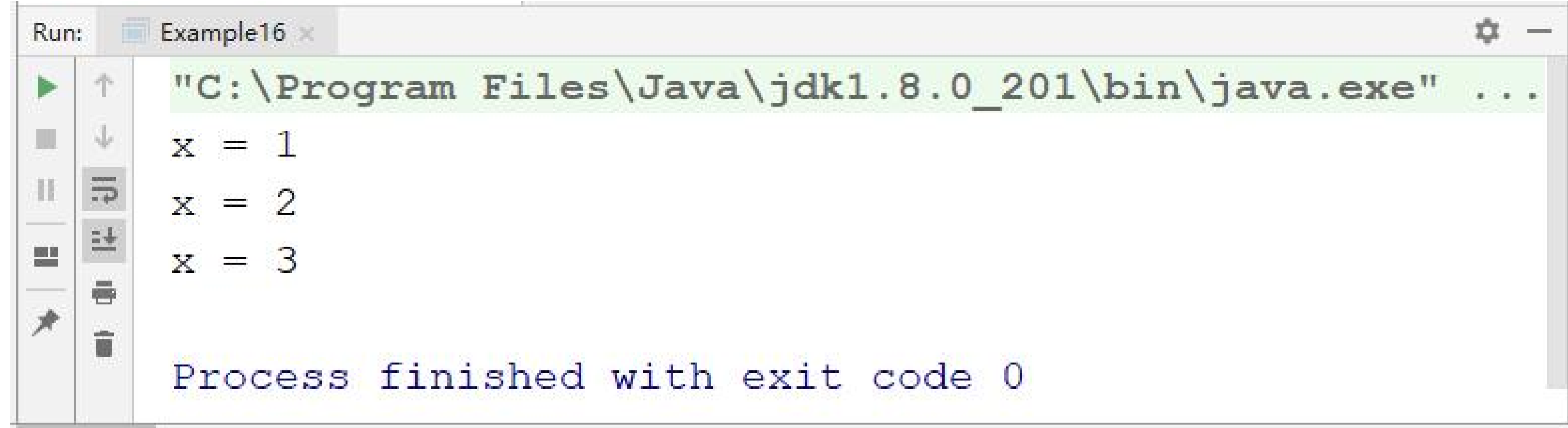
在switch条件语句和循环语句中都可以使用break语句。当它出现在switch条件语句中时，作用是终止某个case并跳出switch结构。当它出现在循环语句中，作用是跳出循环语句，执行循环后面的代码。

2.5.5 跳转语句（break、continue)

```
1 public class Example16 {
2       public static void main(String[] args) {
3             int x = 1;                    // 定义变量x，初始值为1
4             while (x <= 4) {              // 循环条件
5                   System.out.println("x = " + x); // 条件成立，打印x的值
6                   if (x == 3) {
7                         break;
8                   }
9                   x++;                    // x进行自增
10            }
11      }
12 }
```

2.5.5 跳转语句（break、continue)

程序运行结果如下图。



2.5.5 跳转语句（break、continue)

2.5.5 跳转语句（break、continue)

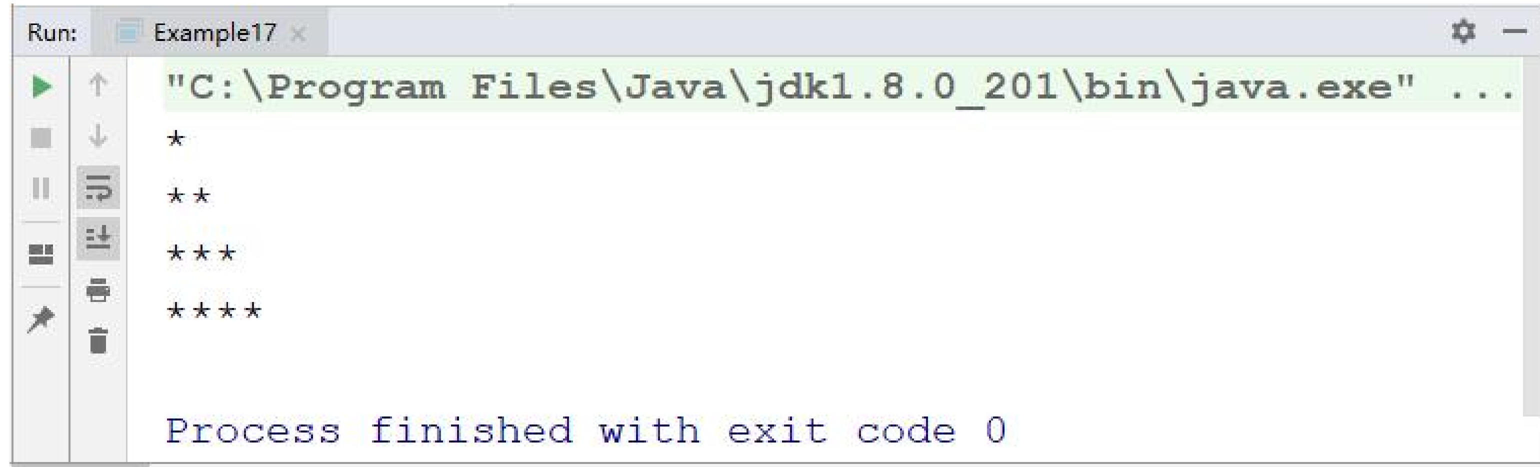
当break语句出现在嵌套循环中的内层循环时，它只能跳出内层循环，如果想使用break语句跳出外层循环，则需要在外层循环中使用break语句。

2.5.5 跳转语句（break、continue)

```
1 public class Example17 {
2     public static void main(String[] args) {
3         int i, j;                          // 定义两个循环变量
4         for (i = 1; i <= 9; i++) {         // 外层循环
5             for (j = 1; j <= i; j++) {     // 内层循环
6                 if (i > 4) {               // 判断i的值是否大于4
7                     break;                 // 跳出外层循环
8                 }
9                 System.out.print("*");     // 打印*
10            }
11            System.out.print("\n");        // 换行
12        }
13    }
14 }
```

2.5.5 跳转语句（break、continue)

程序运行结果如下图。



2.5.5 跳转语句（break、continue)

2.5.5 跳转语句（break、continue)

2．continue语句

continue语句用在循环语句中，它的作用是终止本次循环，执行下一次循环。

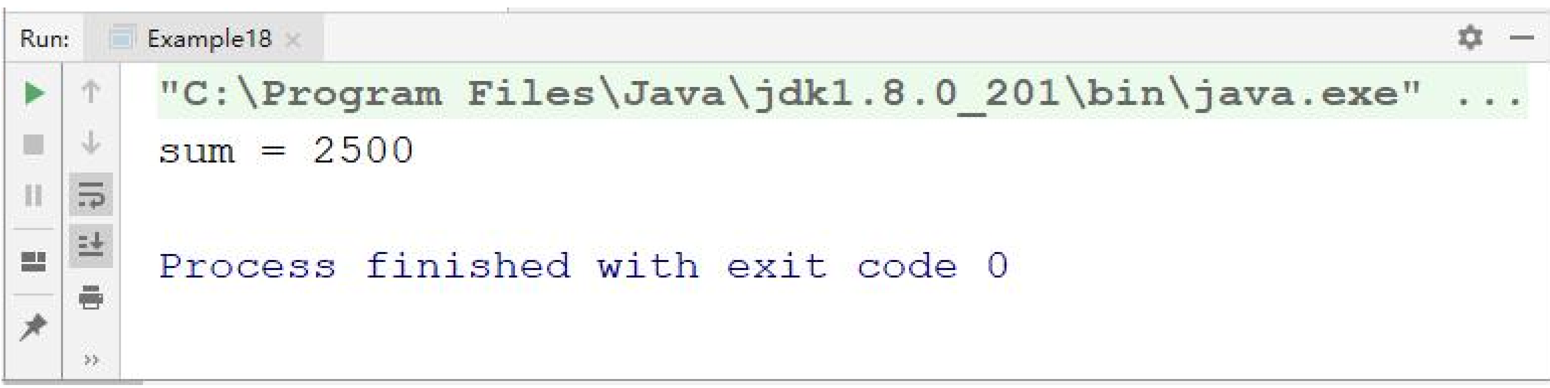
2.5.5 跳转语句（break、continue)

接下来通过对1~100之内的奇数求和演示continue的用法。

```
1 public class Example18 {
2 	public static void main(String[] args) {
3 		int sum = 0;                        // 定义变量sum，用于记住和
4 		for (int i = 1; i <= 100; i++) {
5 			if (i % 2 == 0) {           // i是一个偶数，不累加
6 				continue;           // 结束本次循环
7 			}
8 			sum += i;                   // 实现sum和i的累加
9 		}
10 		System.out.println("sum = " + sum);
11 	}
12 }
```

2.5.5 跳转语句（break、continue)

程序运行结果如下图。



2.5.5 跳转语句（break、continue)

上述代码使用for循环让变量i的值在1~100之间循环，在循环过程中，当i的值为偶数时，执行continue语句结束本次循环，进行下一次循环；当i的值为奇数时，sum和i进行累加，最终得到1~100之间所有奇数的和，打印“sum = 2500”。

【案例2-3】 超市购物

编写一个超市购物程序，在一家超市有牙刷、毛巾、水杯、苹果和香蕉五种商品，用户输入商品序列号进行商品购买，用户输入购买数量后计算出所需要花费的钱，一次购买结束后，需要用户输入“Y”或“N”，“Y”代表继续购买，“N”代表购物结束。商品价格如下表所示。

编号	商品名称	价格
1	牙刷	8.8元
2	毛巾	10.0元
3	水杯	18.8元
4	苹果	12.5元
5	香蕉	15.5元

【案例2-4】 为新员工分配部门

某公司现有java程序开发部门、C#程序开发部门、asp.net程序测试部门、前端程序开发部门四个部门。编写一个程序实现新入职的员工的部门分配，要求根据用户输入的员工姓名和应聘语言确定员工应该分配到哪个部门。若公司没有与输入的语言匹配的部门，则进行相关提示。

【案例2-5】剪刀石头布

“剪刀石头布”的游戏相信大家都不陌生，本案例要求编写一个剪刀石头布游戏的程序。程序启动后会随机生成1~3的随机数，分别代表剪刀、石头和布，玩家通过键盘输入剪刀、石头和布与电脑进行5轮的游戏，赢的次数多的一方为赢家。若五局皆为平局，则最终结果判为平局。

2.6.1 什么是方法

方法就是一段可以重复调用的代码。假设有一个游戏程序，程序在运行过程中，要不断地发射炮弹。发射炮弹的动作需要编写100行代码，在每次实现发射炮弹的地方都需要重复地编写这100行代码，这样程序会变得很臃肿，可读性也非常差。为了解决上述问题，通常会将发射炮弹的代码提取出来，放在一个{}中，并为这段代码起个名字，提取出来的代码可以被看作是程序中定义的一个方法。这样在每次发射炮弹的地方，只需通过代码的名称调用方法，就能完成发射炮弹的动作。需要注意的是，有些书中也会把方法称为函数。

2.6.1 什么是方法

在Java中，定义一个方法的语法格式如下：

```
修饰符 返回值类型 方法名(参数类型 参数名1,参数类型 参数名2,. . . ){
        执行语句
        ...
        return 返回值;
}
```

2.6.1 什么是方法

对于方法的语法格式，具体说明如下：

- 修饰符：方法的修饰符比较多，例如，对访问权限进行限定的修饰符，static修饰符，final修饰符等，这些修饰符在后面的学习过程中会逐步介绍。

2.6.1 什么是方法

- 返回值类型：用于限定方法返回值的数据类型。
- 参数类型：用于限定调用方法时传入参数的数据类型。
- 参数名：是一个变量，用于接收调用方法时传入的数据。
- return关键字：用于返回方法指定类型的值并结束方法。
- 返回值：被return语句返回的值，该值会返回给调用者。

2.6.1 什么是方法

方法中的“参数类型 参数名1，参数类型 参数名2”被称作参数列表，参数列表用于描述方法在被调用时需要接收的参数，如果方法不需要接收任何参数，则参数列表为空，即()内不写任何内容。方法的返回值类型必须是方法声明的返回值类型，如果方法没有返回值，返回值类型要声明为void，此时，方法中return语句可以省略。

2.6.1 什么是方法

接下来通过一个案例演示方法的定义与调用，在该案例中，定义一个方法，使用“*”符号打印矩形。

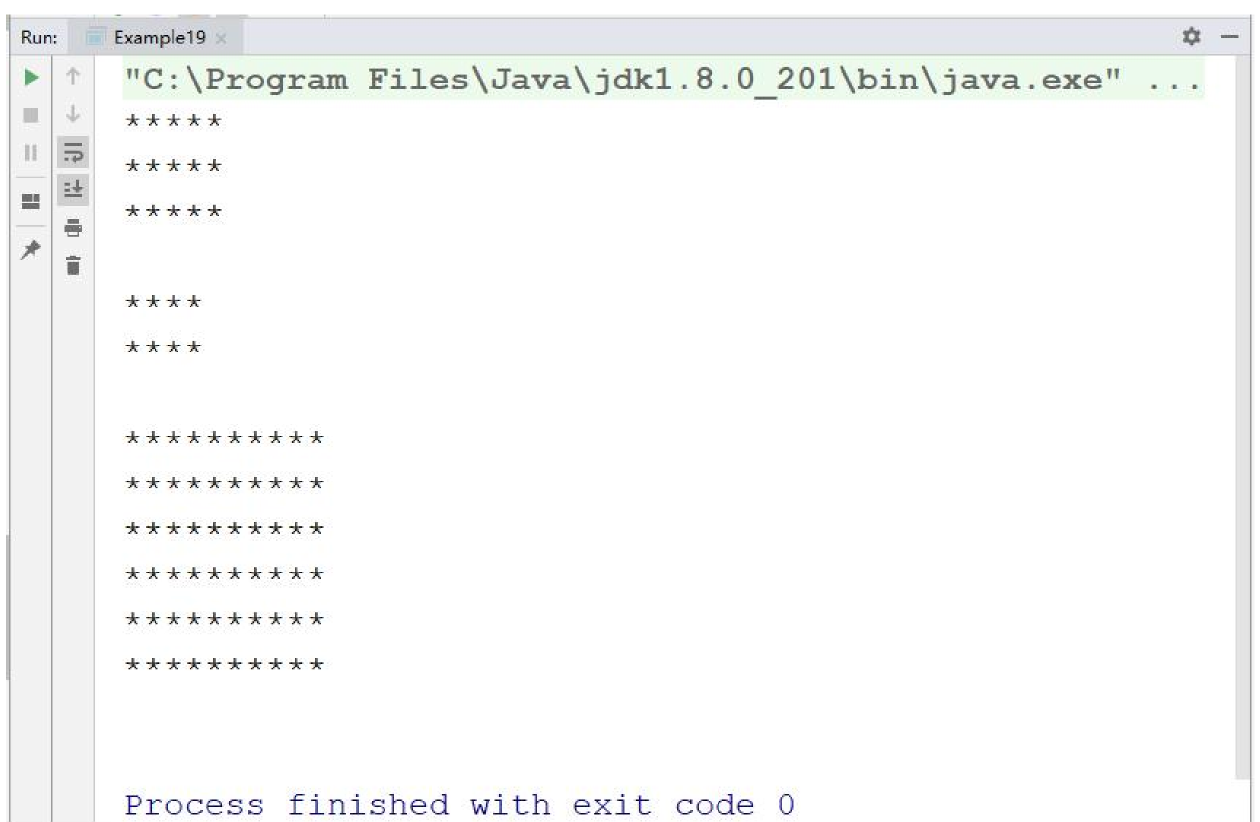
```
1 public class Example19 {
2         public static void main(String[] args) {
3                 printRectangle(3, 5);          // 调用 printRectangle()方法实现打印矩形
4                 printRectangle(2, 4);
5                 printRectangle(6, 10);
6         }
```

2.6.1 什么是方法

```
7        // 下面定义了一个打印矩形的方法，接收两个参数，其中height为高，width为宽
8        public static void printRectangle(int height, int width) {
9                // 下面是使用嵌套for循环实现*打印矩形
10               for (int i = 0; i < height; i++) {
11                       for (int j = 0; j < width; j++) {
12                               System.out.print("*");
13                       }
14                       System.out.print("\n");
15               }
16               System.out.print("\n");
17       }
18 }
```

2.6.1 什么是方法

程序运行结果如下图。



2.6.1 什么是方法

上述代码中，第8~17行代码定义了一个方法printRectangle()，{}内实现打印矩形的代码是方法体，printRectangle是方法名，方法名后面()中的height和width是方法的参数，方法名前面的void表示方法没有返回值。第3~5行代码调用printRectangle()方法传入不同的参数，分别打印出3行5列、2行4列和6行10列的矩形。由运行结果可知，程序成功打印出了三个矩形。

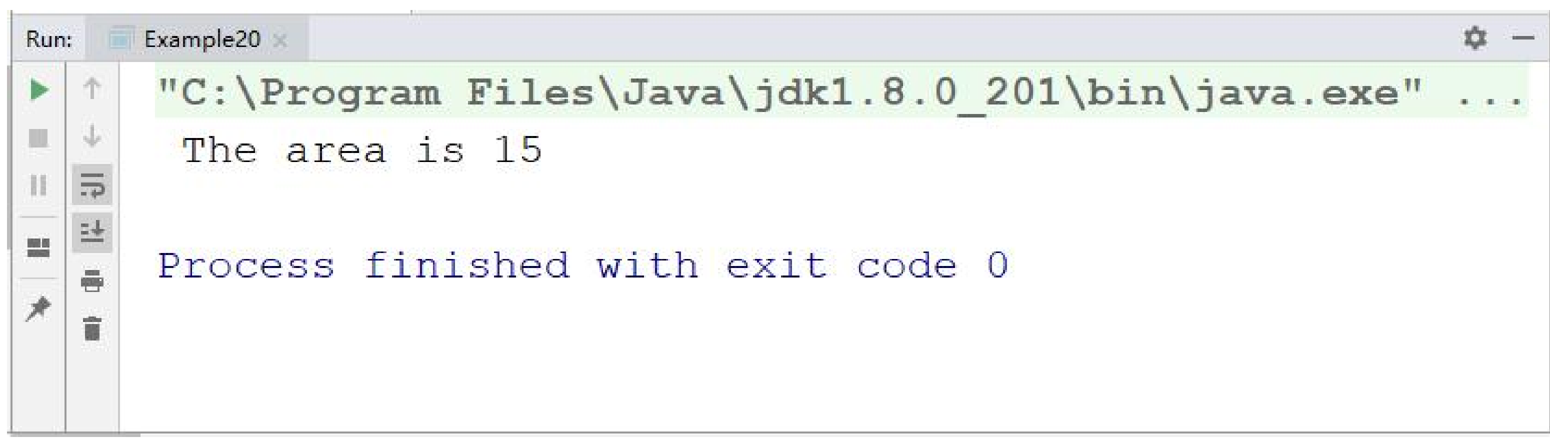
2.6.1 什么是方法

接下来通过一个案例演示有返回值方法的定义与调用。

```
public class Example20 {
    public static void main(String[] args) {
        int area = getArea(3, 5);           // 调用 getArea方法
        System.out.println(" The area is " + area);
    }
    // 下面定义了一个求矩形面积的方法，接收两个参数，其中x为高，y为宽
    public static int getArea(int x, int y) {
        int temp = x * y;                   // 使用变量temp记住运算结果
        return temp;                        // 将变量temp的值返回
    }
}
```

2.6.1 什么是方法

程序运行结果如下图。

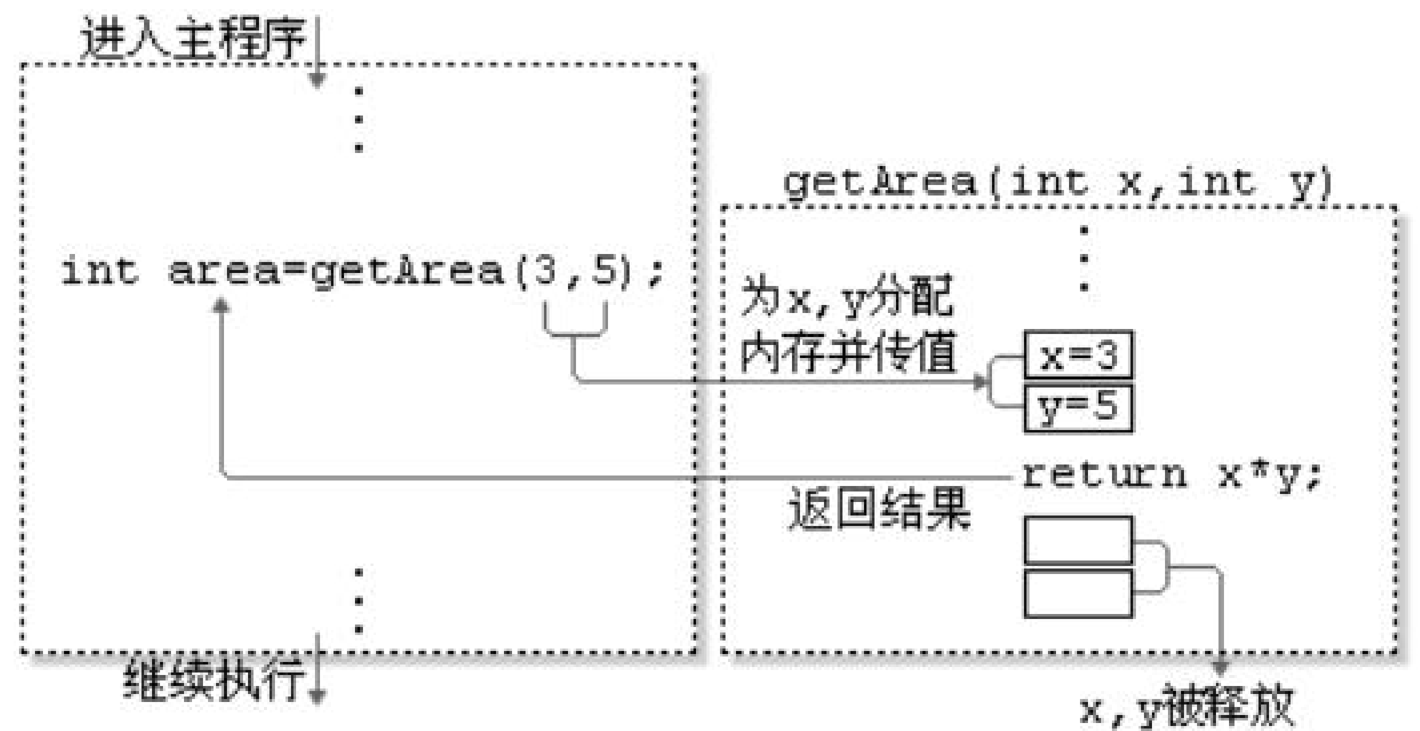


2.6.1 什么是方法

上述代码中，第7~10行代码定义了一个getArea()方法用于求矩形的面积，参数x和y分别用于接收调用方法时传入的长和宽，return语句用于返回计算所得的面积。在main()方法中调用getArea()方法，获得长为3宽为5的矩形的面积，并将结果打印出来。由运行结果可知，程序成功打印出了矩形面积15。

2.6.1 什么是方法

通过一个图展示getArea()方法的整调用过程如下图。



2.6.1 什么是方法

当调用getArea()方法时，程序执行流程从当前函数调用处跳转到getArea()内，程序为参数变量x和y分配内存，并将传入的参数3和5分别赋值给变量x和y。在getArea()函数内部，计算x*y的值，并将计算结果通过return语句返回，整个方法的调用过程结束，变量x和y被释放。程序执行流程从getArea()函数内部跳转回主程序的函数调用处。

2.6.2 方法的重载

在平时生活中经常会出现这样一种情况，一个班里可能同时有两个叫小明的同学，甚至有多个，但是他们的身高、体重、外貌等有所不同，老师点名时都会根据他们的特征来区分。在编程语言里也存在这种情况，参数不同的方法有着相同的名字，调用时根据参数不同确定调用哪个方法，这就是Java方法重载机制。

2.6.2 方法的重载

所谓方法重载，就是在同一个作用域内方法名相同但参数个数或者参数类型不同的方法。例如，在同一个作用域内同时定义三个add()方法，这三个add()方法就是重载函数。

2.6.2 方法的重载

下面一个案例演示重载方法的定义与调用，在该案例中，定义三个add()方法，分别用于实现两个整数相加、三个整数相加以及两个小数相加的功能。

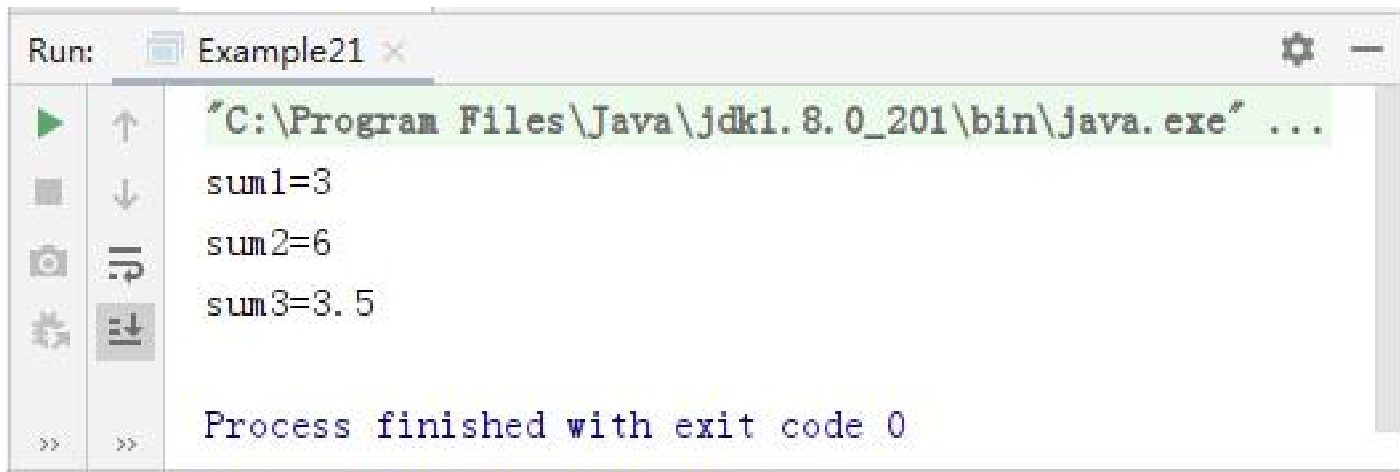
```
1  public class Example21 {
2      public static void main(String[] args) {
3          // 下面是针对求和方法的调用
4          int sum1 = add(1, 2);
5          int sum2 = add(1, 2, 3);
6          double sum3 = add(1.2, 2.3);
7          // 下面的代码是打印求和的结果
8          System.out.println("sum1=" + sum1);
9          System.out.println("sum2=" + sum2);
10         System.out.println("sum3=" + sum3);
11     }
```

2.6.2 方法的重载

```
12        // 下面的方法实现了两个整数相加
13        public static int add(int x, int y) {
14                return x + y;
15        }
16        // 下面的方法实现了三个整数相加
17        public static int add(int x, int y, int z) {
18                return x + y + z;
19        }
20        // 下面的方法实现了两个小数相加
21        public static double add(double x, double y) {
22                return x + y;
23        }
24 }
```

2.6.2 方法的重载

程序运行结果如下图。



2.6.2 方法的重载

上述代码中，定义了三个同名的add()方法，但它们的参数个数或类型不同，从而形成了方法的重载。在main()方法中调用add()方法时，通过传入不同的参数便可以确定调用哪个重载的方法，如add(1,2)调用的是第13~14行代码定义的add()方法。需要注意的是，方法的重载与返回值类型无关。

2.7 数组

数组是指一组类型相同的数据的集合，数组中的每个数据被称作元素。数组可以存放任意类型的元素，但同一个数组里存放的元素类型必须一致。

2.7.1 数组的定义

在Java中，声明数组的方式有两种。

第一种方式：

数据类型[] 数组名 ＝ null;

第二种方式：

数据类型[]数组名;

数组名＝ new数据类型[长度];

2.7.1 数组的定义

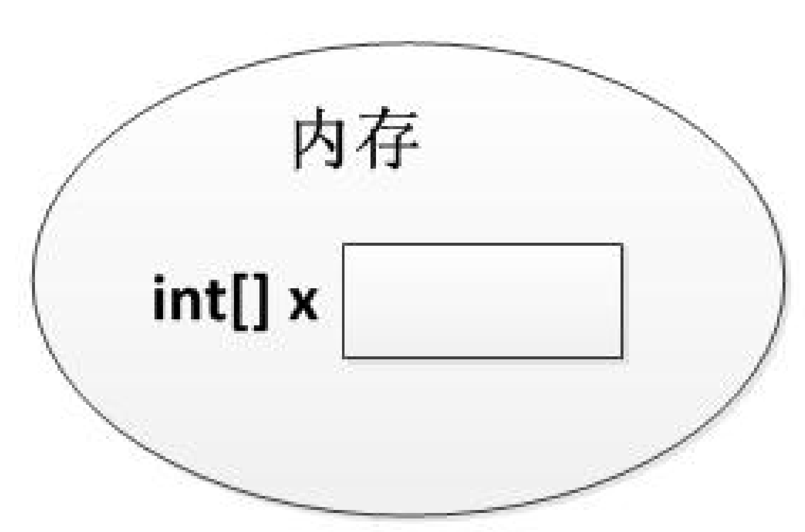
2.7.1 数组的定义

```
int[] x;                    // 声明一个int[]类型的变量
x = new int[100];           // 为数组x分配100个元素空间
```

上述语句就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x[0]，第二个变量的名称为x[1]，以此类推，第100个变量的名称为x[99]，这些变量的初始值都是0。

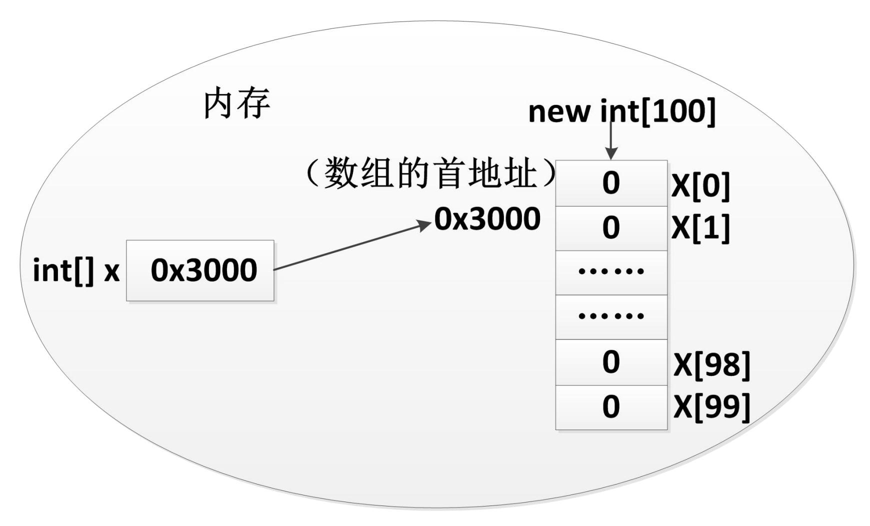
2.7.1 数组的定义

第一行代码声明了一个变量x，该变量的类型为int[]，即声明了一个int类型的数组。变量x会占用一块内存单元，它没有被分配初始值。变量x的内存状态如右图。



2.7.1 数组的定义

第二行代码 x = new int[100];创建了一个数组，将数组的地址赋值给变量x。在程序运行期间可以使用变量x引用数组，这时变量x在内存中的状态会发生变化，如右图。



2.7.1 数组的定义

上图描述了变量x引用数组的情况。该数组中有100个元素，初始值都为0。数组中的每个元素都有一个索引（也可称为角标），要想访问数组中的元素可以通过“x[0]、x[1]、……、x[98]、x[99]”的形式。需要注意的是，数组中最小的索引是0，最大的索引是“数组的长度-1”。在Java中，为了方便获得数组的长度，提供了一个length属性，在程序中可以通过“数组名.length”的方式获得数组的长度，即元素的个数。

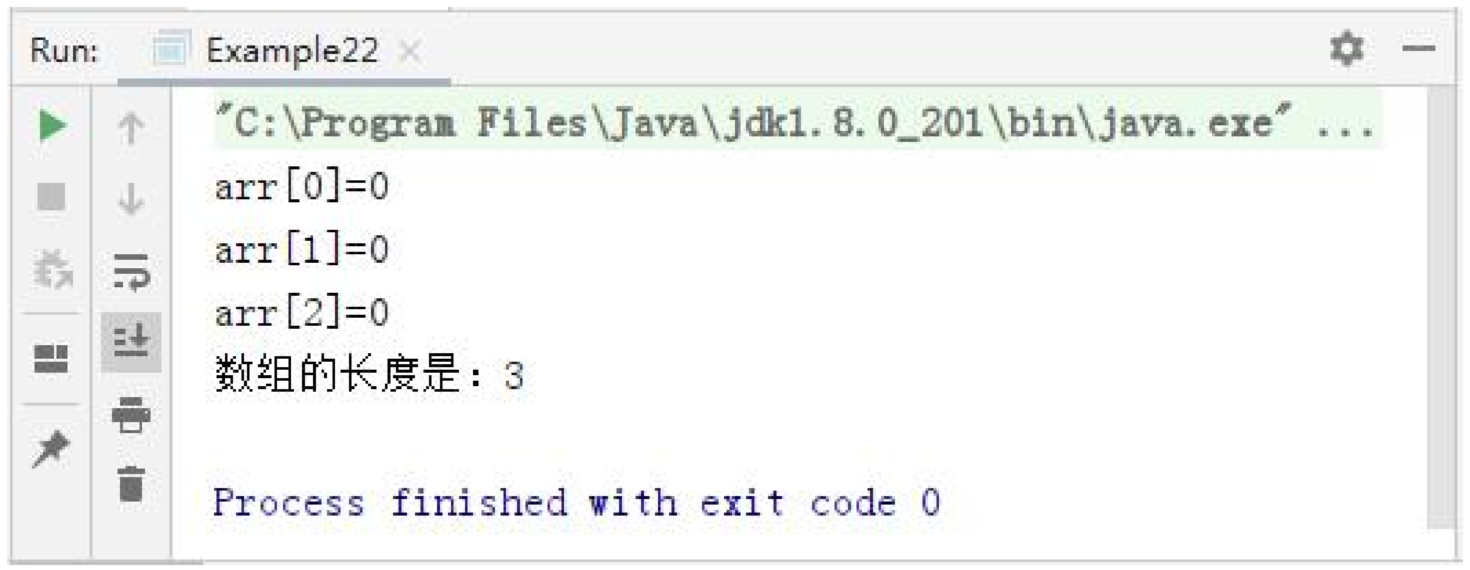
2.7.1 数组的定义

接下来通过一个案例来演示如何定义数组以及访问数组中的元素。

```
public class Example22 {
        public static void main(String[] args) {
                int[] arr;                                    // 声明变量
                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
        }
}
```

2.7.1 数组的定义

程序运行结果如下图。



2.7.1 数组的定义

上述代码中，第3行代码声明了一个int[]类型变量arr，第4行代码创建了一个长度为3的数组，并将数组在内存中的地址赋值给变量arr。在第5~7行代码中，通过索引访问数组中的元素，第8行代码通过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，且三个元素初始值都为0，这是因为当数组被成功创建后，如果没有给数组元素赋值，则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。

2.7.1 数组的定义

数组不进行初始化，系统默认初始化，不同类型数组元素的默认值如下表。

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

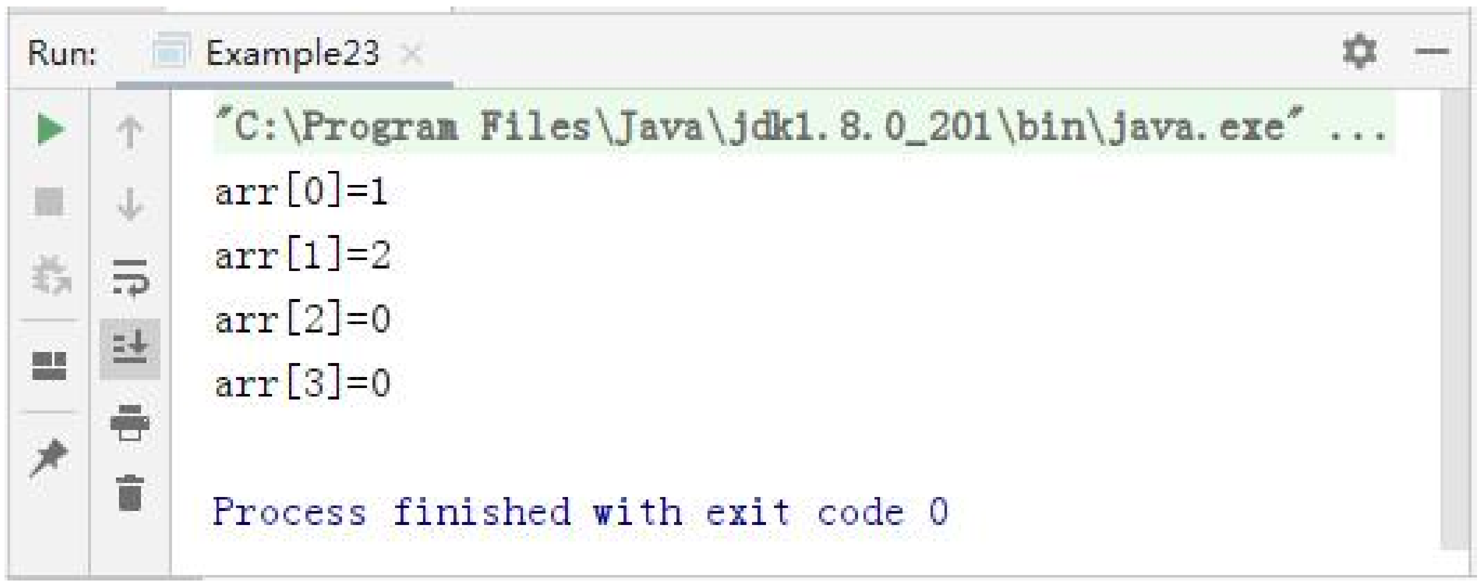
2.7.1 数组的定义

接下来通过一个案例来学习如何为数组的元素赋值。

```
public class Example23 {
        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]);
        }
}
```

2.7.1 数组的定义

程序运行结果如下图。



2.7.1 数组的定义

上述代码中，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，因此打印结果中四个元素的值依次为1、2、0、0。

2.7.1 数组的定义

在定义数组时只指定数组的长度，由系统自动为元素赋初值的方式称作动态初始化。在初始化数组时还有一种方式叫做静态初始化，就是在定义数组的同时就为数组的每个元素赋值。数组的静态初始化有两种方式，具体格式如下：

```
类型[] 数组名 = new 类型[]{元素,元素,……};
类型[] 数组名 = {元素,元素,元素,……};
```

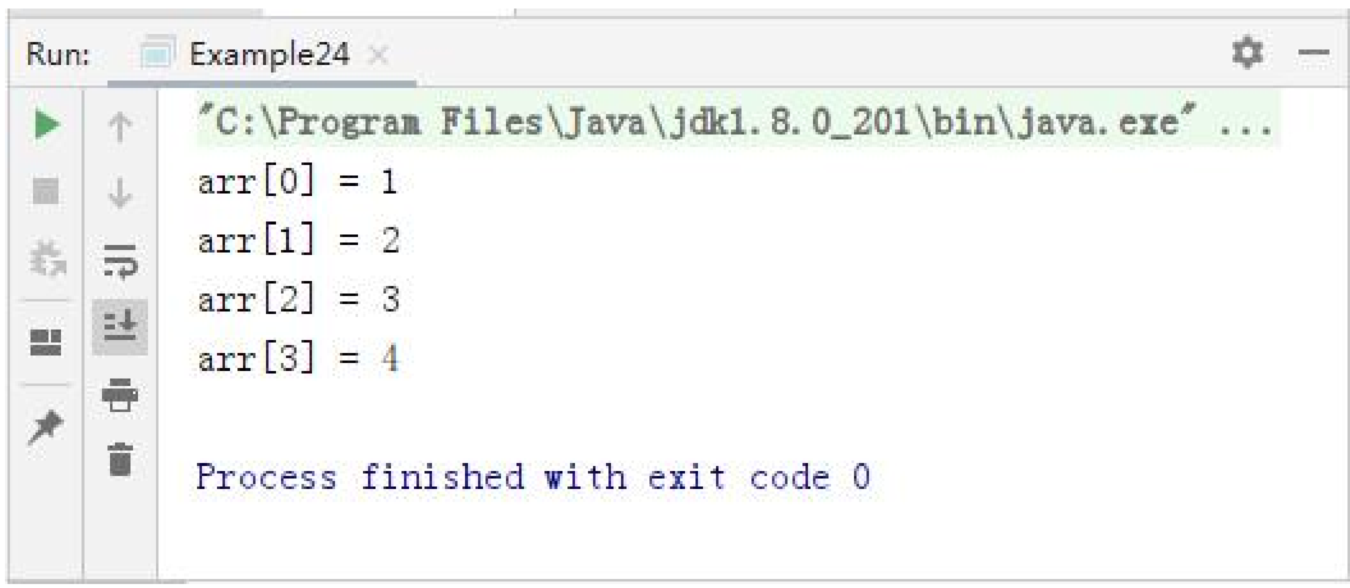
2.7.1 数组的定义

接下来通过一个案例演示数组静态初始化的效果。

```
1 public class Example24 {
2        public static void main(String[] args) {
3               int[] arr = { 1, 2, 3, 4 };        // 静态初始化
4               //依次访问数组中的元素
5               System.out.println("arr[0] = " + arr[0]);
6               System.out.println("arr[1] = " + arr[1]);
7               System.out.println("arr[2] = " + arr[2]);
8               System.out.println("arr[3] = " + arr[3]);
9        }
10 }
```

2.7.1 数组的定义

程序运行结果如下图。



2.7.1 数组的定义

上述代码中，采用静态初始化的方式为每个元素赋予初值，其值分别是1、2、3、4。需要注意的是，文件中的第3行代码千万不可写成int[] x = new int[4]{1,2,3,4};，这样写编译器会报错。原因在于编译器会认为数组限定的元素个数[4]与实际存储的元素{1,2,3,4}个数有可能不一致，存在一定的安全隐患。

脚下留心：数组索引

数组是一个容器，存储到数组中的每个元素，都有自己的自动编号，最小值为0，最大值为数组长度-1，如果要访问数组存储的元素，必须依赖于索引。在访问数组的元素时，索引不能超出0~length-1范围，否则程序会报错。

脚下留心：数组索引

接下来通过一个案例演示索引超出数组范围的情况。

```
public class Example25 {
	public static void main(String[] args) {
		int[] arr = new int[4];                    // 定义一个长度为4的数组
		System.out.println("arr[0]=" + arr[4]); // 通过索引4访问数组元素
	}
}
```

脚下留心：数组索引

程序运行结果如下图。

脚下留心：数组索引

运行结果中所提示的错误信息是数组越界异常ArrayIndexOutOfBoundsException，出现这个异常的原因是数组的长度为4，索引范围为0~3，文件2-25中的第4行代码使用索引4访问元素时超出了数组的索引范围。所谓异常指程序中出现的错误，它会报告出错的异常类型、出错的行号以及出错的原因。

脚下留心：数组索引

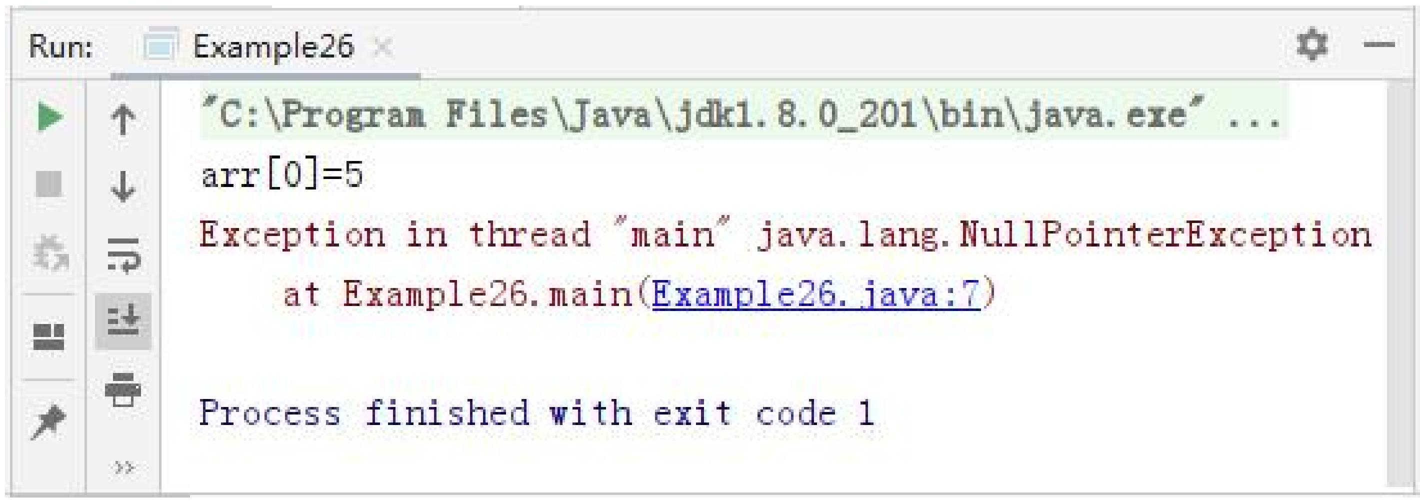
在使用变量引用一个数组时，变量必须指向一个有效的数组对象，如果该变量的值为null，则意味着没有指向任何数组，此时通过该变量访问数组的元素会出现空指针异常，接下来通过一个案例来演示这种异常。

脚下留心：数组索引

```
1 public class Example26 {
2         public static void main(String[] args) {
3                 int[] arr = new int[3];                  // 定义一个长度为3的数组
4                 arr[0] = 5;                              // 为数组的第一个元素赋值
5                 System.out.println("arr[0]=" + arr[0]); // 访问数组的元素
6                 arr = null;                              // 将变量arr置为null
7                 System.out.println("arr[0]=" + arr[0]); // 访问数组的元素
8         }
9 }
```

脚下留心：数组索引

程序运行结果如下图。



脚下留心：数组索引

从运行结果可以看出，文件2-26的第4~5行代码都能通过变量arr正常地操作数组。第6行代码将变量置为null，第7行代码再次访问数组时就出现了空指针异常NullPointerException。

2.7.2 数组的常见操作

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学习如何使用for循环遍历数组。

```
1 public class Example27 {
2     public static void main(String[] args) {
3         int[] arr = { 1, 2, 3, 4, 5 };     // 定义数组
4         // 使用for循环遍历数组的元素
5         for (int i = 0; i < arr.length; i++) {
6             System.out.println(arr[i]); // 通过索引访问元素
7         }
8     }
9 }
```

2.7.2 数组的常见操作

2．数组最值

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获取数组中元素的最大值。

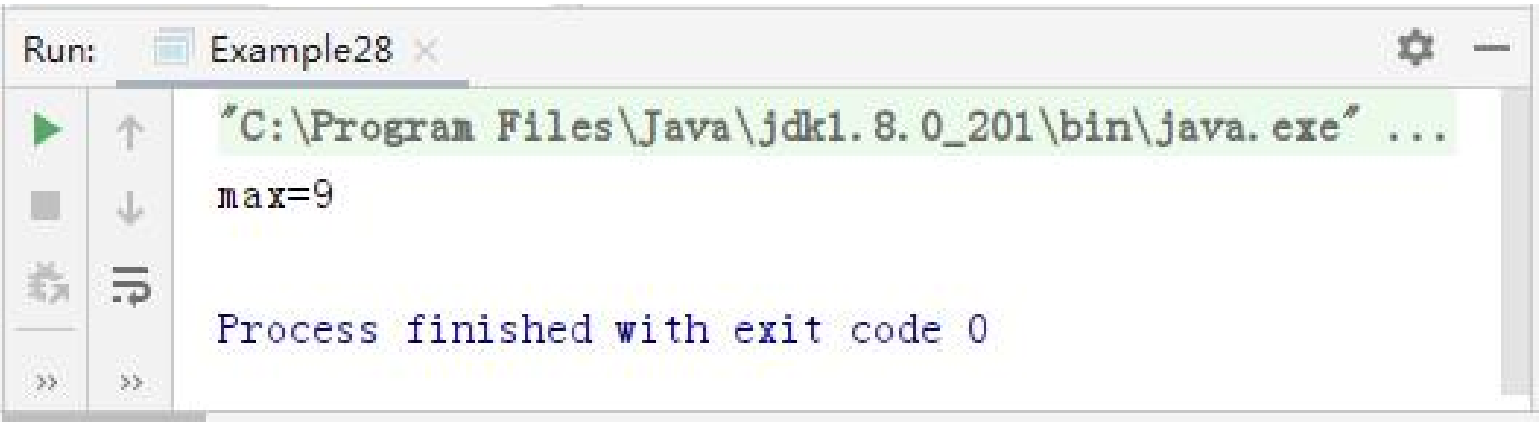
```
public class Example28 {
        public static void main(String[] args) {
                int[] arr = { 4, 1, 6, 3, 9, 8 };          // 定义一个数组
                int max = getMax(arr);                      // 调用获取元素最大值的方法
                System.out.println("max=" + max);   // 打印最大值
```

2.7.2 数组的常见操作

```
static int getMax(int[] arr) {
    int max = arr[0];  // 定义变量max用于记住最大数，首先假设第一个元素为最大值
    // 下面通过一个for循环遍历数组中的元素
    for (int x = 1; x < arr.length; x++) {
        if (arr[x] > max) {          // 比较 arr[x]的值是否大于max
            max = arr[x];            // 条件成立，将arr[x]的值赋给max
        }
    }
    return max;                      // 返回最大值max
}
}
```

2.7.2 数组的常见操作

程序运行结果如下图。



2.7.2 数组的常见操作

上述代码中，第7~16行代码定义的getMax()方法用于求数组中的最大值，该方法定义了一个临时变量max，用于记录数组的最大值。首先假设数组中第一个元素arr[0]为最大值，然后使用for循环对数组进行遍历，在遍历的过程中只要遇到比max值还大的元素，就将该元素赋值给max。这样一来，变量max就能够在循环结束时记录数组中的最大值。需要注意的是，for循环中的变量i是从1开始的，原因是程序已经假设第一个元素为最大值，for循环只需要从第二个元素开始比较。第4行代码调用getMax()函数获取数据arr的最大值，由图2-46可知，数组arr中的最大值为9。

2.7.2 数组的常见操作

3．数组排序

在操作数组时，经常需要对数组中的元素进行排序。下面为读者介绍一种比较常见的排序算法——冒泡排序。在冒泡排序的过程中，不断地比较数组中相邻的两个元素，较小者向上浮，较大者往下沉，整个过程和水中气泡上升的原理相似。

2.7.2 数组的常见操作

第一步，从第一个元素开始，将相邻的两个元素依次进行比较，如果前一个元素比后一个元素大，则交换它们的位置，直到最后两个元素完成比较。整个过程完成后，数组中最后一个元素自然就是最大值，这样也就完成了第一轮比较。

第二步，除了最后一个元素，将剩余的元素继续进行两两比较，过程与第一步相似，这样就可以将数组中第二大的元素放在倒数第二个位置。

第三步，以此类推，持续对越来越少的元素重复上面的步骤，直到没有任何一对元素需要比较为止。

2.7.2 数组的常见操作

接下来通过一个案例实现冒泡排序。

```
public class Example29 {
    public static void main(String[] args) {
        int[] arr = { 9, 8, 3, 5, 2 };
        System.out.print("冒泡排序前  : ");
        printArray(arr);              // 打印数组元素
        bubbleSort(arr);              // 调用排序方法
        System.out.print("冒泡排序后  : ");
        printArray(arr);              // 打印数组元素
    }
```

2.7.2 数组的常见操作

```
// 定义打印数组元素的方法
public static void printArray(int[] arr) {
        // 循环遍历数组的元素
        for (int i = 0; i < arr.length; i++) {
                System.out.print(arr[i] + " ");     // 打印元素和空格
        }
        System.out.print("\n");
}
// 定义对数组排序的方法
public static void bubbleSort(int[] arr) {
        // 定义外层循环
        for (int i = 0; i < arr.length - 1; i++) {
                //定义内层循环
```

2.7.2 数组的常见操作

```
                    for (int j = 0; j < arr.length - i - 1; j++) {
                        if (arr[j] > arr[j + 1]) {    // 比较相邻元素
                            // 下面的三行代码用于交换两个元素
                            int temp = arr[j];
                            arr[j] = arr[j + 1];
                            arr[j + 1] = temp;
                        }
                    }
                System.out.print("第" + (i + 1) + "轮排序后：");
                printArray(arr);              // 每轮比较结束打印数组元素
            }
        }
}
```

2.7.2 数组的常见操作

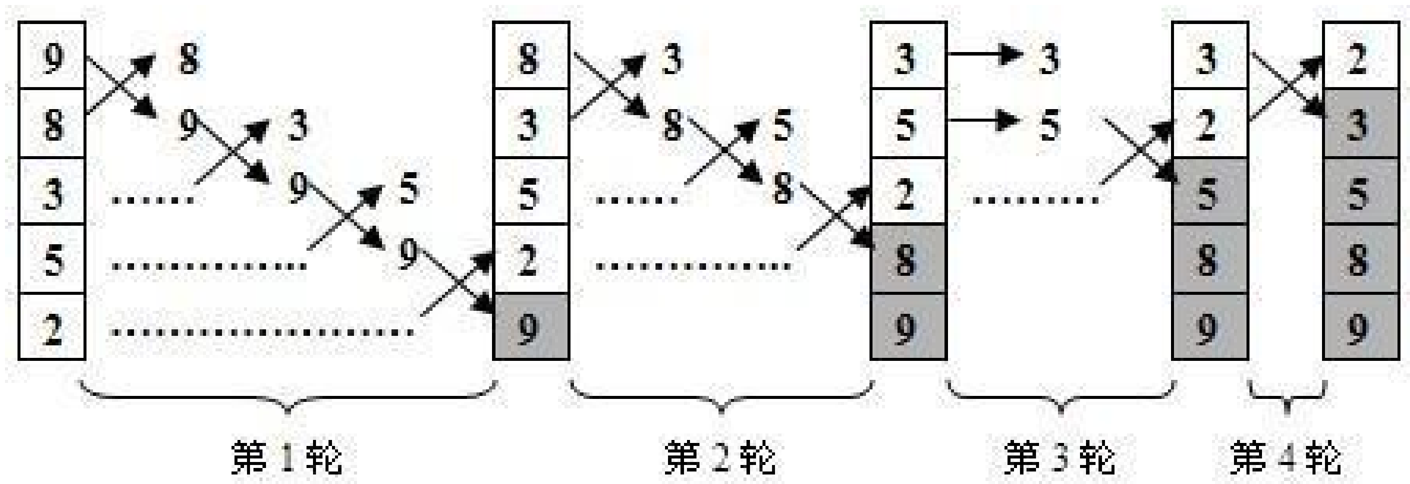
程序运行结果如下图。

2.7.2 数组的常见操作

在上述代码中，第19~34行代码定义了bubbleSort ()方法，在bubbleSort()方法中通过嵌套for循环实现数组元素的冒泡排序，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，因此外层循环的次数为arr.length-1。内层循环的循环变量用于控制每轮比较的次数，它被作为索引用于访问数组的元素。由于变量在循环过程中是自增的，因此可以实现相邻元素依次进行比较。

2.7.2 数组的常见操作

在每次比较时如果前者小于后者，就交换两个元素的位置，元素交换过程如下图。

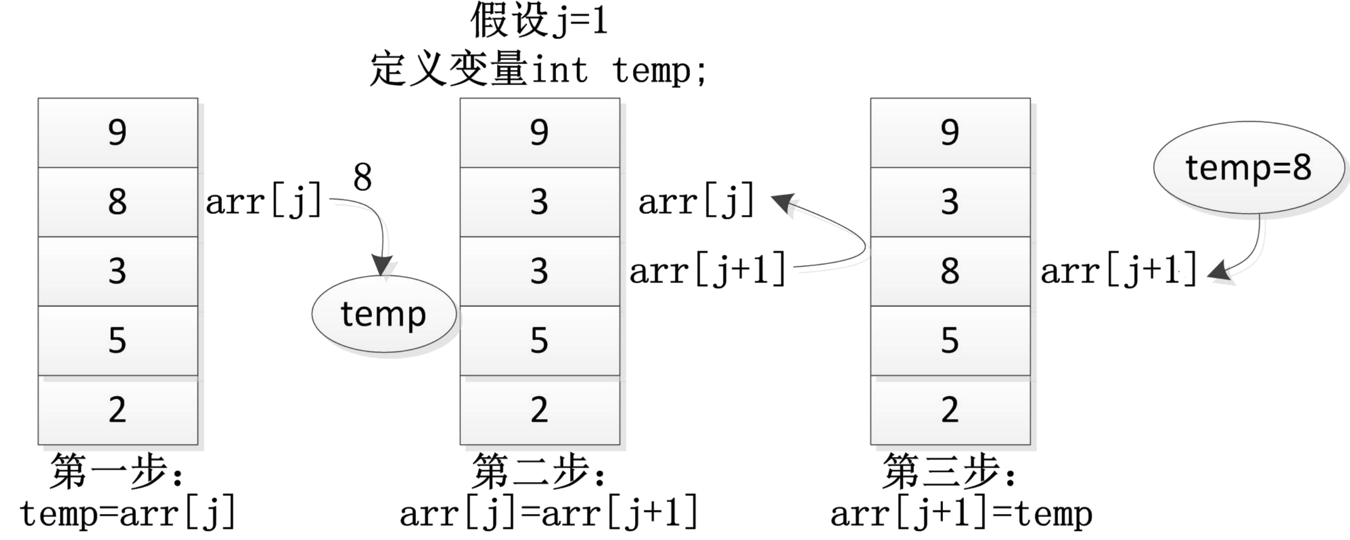


2.7.2 数组的常见操作

在上图中，第一轮比较中，第一个元素“9”为最大值，因此它在每次比较时都会发生位置的交换，被放到最后一个位置。第二轮比较与第一轮过程类似，元素“8”被放到倒数第二个位置。第三轮比较中，第一次比较没有发生位置的交换，在第二次比较时才发生位置交换，元素“5”被放到倒数第三个位置。第四轮比较只针对最后两个元素，它们比较后发生了位置的交换，元素“3”被放到第二个位置。通过四轮比较，数组中的元素已经完成了排序。

2.7.2 数组的常见操作

在交换两个元素时，首先定义了一个变量temp用于记录数组元素arr[j]的值，然后将arr[j+1]的值赋给arr[j],最后再将temp的值赋给arr[j+1]，这样便完成了两个元素的交换。交换过程如下图。



2.7.3 二维数组

二维数组的定义有很多方式。

第一种方式：

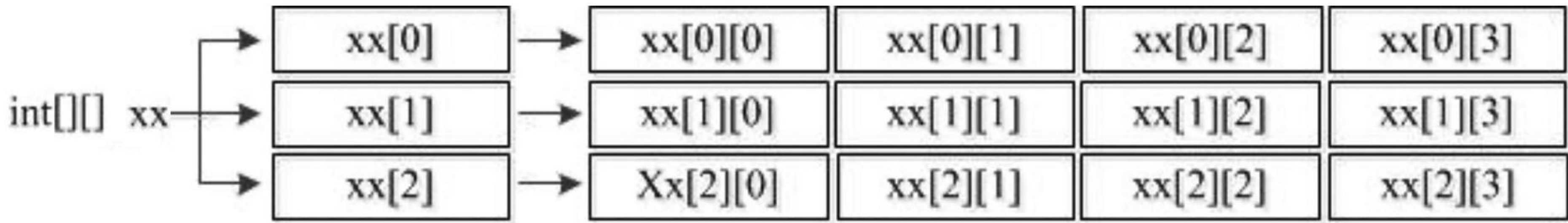
数据类型[][] 数组名 ＝ new数据类型[行的个数][列的个数];

下面以第一种方式声明一个数组，如下所示。

```
int[][] xx= new int[3][4];
```

2.7.3 二维数组

上面的代码相当于定义了一个3*4的二维数组，即3行4列的二维数组，接下来通过一个图表示xx[3][4]。



2.7.3 二维数组

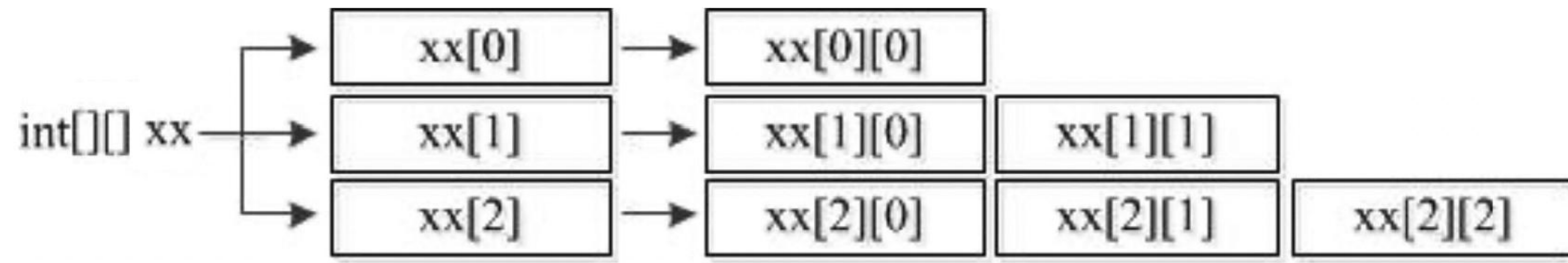
第二种方式：

数据类型[][] 数组名 ＝ new int[行的个数][];

下面以第二种方式声明一个数组，如下所示。

```
int[][] xx= new int[3][];
```

第二种方式和第一种类似，只是数组中每个元素的长度不确定。



2.7.3 二维数组

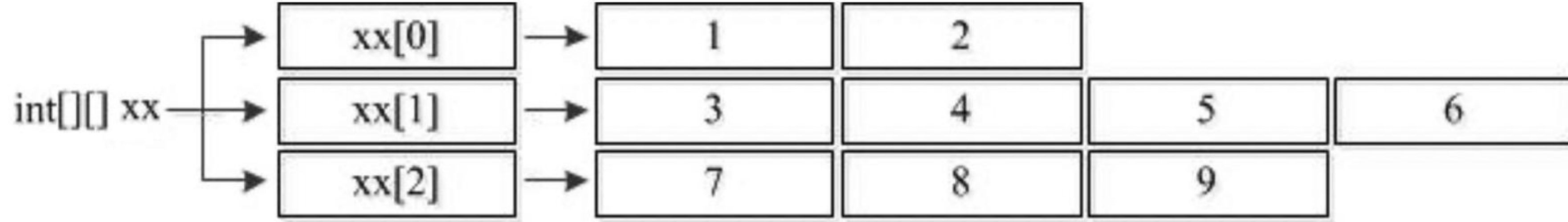
第三种方式：

数据类型[][] 数组名＝ {{第0行初始值},{第1行初始值},...,{第n行初始值}};

下面以第三种方式声明一个数组，如下所示。

```
int[][] xx= {{1,2},{3,4,5,6},{7,8,9}};
```

上面的二维数组arr中定义了三个元素，这三个元素都是数组，分别为{1,2}、{3,4,5,6}、{7,8,9}。



2.7.3 二维数组

二维数组中元素的访问也是通过索引的方式。例如，访问二维数组arr中第一个元素数组的第二个元素，具体代码如下：

```
arr[0][1];
```

2.7.3 二维数组

接下来通过一个案例演示二维数组的使用，该案例要统计一个公司三个销售小组中每个小组的总销售额以及整个公司的销售额。

```
1 public class Example30 {
2     public static void main(String[] args) {
3         int[][] arr = new int[3][];            // 定义一个长度为3的二维数组
4         arr[0] = new int[] { 11, 12 };          // 为数组的元素赋值
5         arr[1] = new int[] { 21, 22, 23 };
6         arr[2] = new int[] { 31, 32, 33, 34 };
7         int sum = 0;                            // 定义变量记录总销售额
8         for (int i = 0; i < arr.length; i++) {  // 遍历数组元素
```

2.7.3 二维数组

```
9                int groupSum = 0;            // 定义变量记录小组销售总额
10               for (int j = 0; j < arr[i].length; j++) { // 遍历小组内每个人的销售额
11                   groupSum = groupSum + arr[i][j];
12               }
13               sum = sum + groupSum;                // 累加小组销售额
14               System.out.println("第" + (i + 1) + "小组销售额为：" + groupSum + " 万元。");
15           }
16           System.out.println("总销售额为: " + sum + " 万元。");
17       }
18 }
```

2.7.3 二维数组

程序运行结果如下图。

2.7.3 二维数组

上述代码中，第3行代码定义了一个长度为3的二维数组arr；第4~6行代码为数组arr的每个元素赋值。文件中还定义了两个变量sum和groupSum，其中sum用于记录公司的总销售额，groupSum用于记录每个销售小组的销售额。第8~15行代码通过嵌套for循环统计销售额，外层循环对三个销售小组进行遍历，内层循环对每个小组员工的销售额进行遍历，内层循环每循环一次就相当于将一个小组员工的销售额累加到本小组的销售总额groupSum中。内层循环结束，相当于本小组销售总金额计算完毕，把groupSum的值累加到sum中。当外层循环结束时，三个销售小组的销售总额groupSum都累加到了sum中，统计出整个公司的销售总额。

【案例2-6】 登录注册

编写程序实现简单的登录注册功能。程序包含以下4个功能：

（1）登录功能，用户输入正确的账号密码进行成功；

（2）注册功能，输入用户名和密码进行注册；

（3）查看功能，查看所有的用户名与密码；

（4）退出功能，退出系统。

用户可以输入对应的编号进行相应的功能操作。例如，输入2进入注册功能，输入用户名和密码进行注册。

【案例2-7】 抽取幸运观众

在一些节目活动中，经常会有抽取幸运观众环节。本案例要求编写程序实现幸运观众的抽取，在指定人群中随机抽取一名幸运观众。

案例功能要求如下：

（1）从键盘输入3名观众。

（2）存储观众姓名。

（3）总览观众姓名。

（4）随机选取一名观众，并打印出该观众姓名。

2.8 本章小结

本章主要介绍了学习Java的基础知识。首先介绍了Java语言的基本语法，包括Java程序的基本格式、注释、标识符等；其次介绍了Java中的变量和运算符；接着介绍了选择结构语句、循环结构语句和跳转语句；然后介绍了方法，包括方法的概念、定义、调用以及重载；最后介绍了数组，包括数组的定义、数组的常见操作、多维数组。通过本章的学习，读者能够掌握Java程序的基本语法格式、变量和运算符的使用，能够掌握流程控制语句的使用，能够掌握方法的定义和调用方式，能够掌握数组的声明、初始化和使用等，为后面学习作好铺垫。