本章练习

第5章 数组

引 言

在Java程序中，经常需要存储大量的、具有相同性质的数据，例如，需要输入60名同学的成绩，并计算其平均值与方差。在这种情况下，Java程序需要实现这60个数的输入与存储，然后进行分析处理。为了完成这个任务，程序声明60个简单变量分别存储这些数据显然是不太现实的。对于这一问题，数组就是一种较好的解决方法。

5.1 Java数组

在程序开发设计中，经常需要存储大量相同类型的数据。针对这个问题，Java和大多数其他高级语言一样提供了数组来保存这组数据。数组是相同类型数据的集合，集合的名字就是数组名。

数组用一组连续的内存空间存储数据，每个数据都是数组中的一个元素，数组中的每个元素都有对应的下标，下标是从0开始的整数。由于数组是用一片连续的内存单元存放数据，通过数组名与下标就可以定位并访问数组中的任何一个元素。数组的这种能够被快速访问的特点，让程序代码拥有非常高的访问效率，也使得数组成为程序设计中最常用的数据结构。

在Java语言中，数组与对象一样都是引用类型的变量，需要用关键词new创建数组。Java数组具有如下特点。

(1) 数组中元素的类型相同。

(2) 数组中所有的元素存放在一块连续的内存空间中。

(3) 通过数组名与下标可以访问每个元素，下标从0开始。

(4) 数组的大小一旦定义以后，不可再动态增大或减小。

Java语言提供了一维数组与多维数组，在编程时可以根据需要创建与使用。

观看视频

5.2 一维数组

数组是用来存储批量数据的，一个数组中的元素应该都属于同一种数据类型。

同一个数组中的元素在内存中是按照顺序连续存放在一片空间内，因此可以按照它们在内存中的顺序进行编号，也就是每个数组元素对应一个下标，并按照这个数组下标来进行

存取访问。数组需要创建后才能访问，数组的下标从 0 开始，数组的长度就是数组中元素的个数，其大小在数组初始化之后就固定下来。

假设有一个具有 10 个元素的双精度浮点数数组 array，并对该数组完成数据初始化，则可以用图 5-1 展示一个双精度类型的数组变量 array 在内存空间中的存储方式。

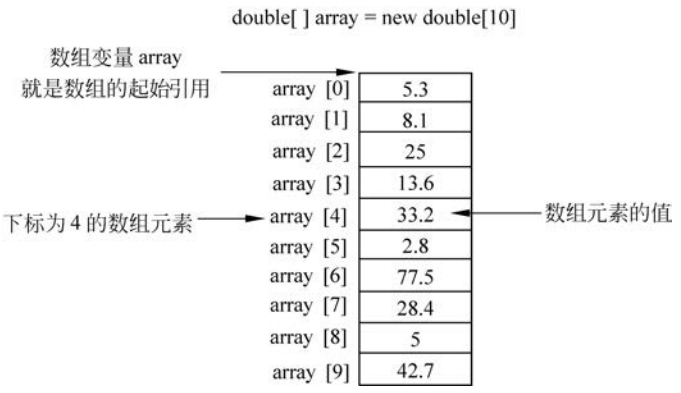

图 5-1 数组存储空间示意图

Java 的数组可以有多个维度，其中一维数组是数组应用的基础，有了一维数组的概念之后，可以很容易地把相关概念推广到多维数组。本节将以一维数组为例介绍如何声明数组、创建数组和存取访问数组中的数据。

5.2.1 数组的声明

Java 语言的数组属于引用类型，为了正确使用数组，必须声明一个数组变量来引用数组。在声明数组变量时，需要给出数组的名称，以及数组中的数据元素所属的数据类型。声明数组的语法格式有如下两种：

格式 1：

```
数组元素类型 数组名[];
```

格式 2：

```
数组元素类型[] 数组名;
```

其中格式 1 和 C 语言的数组声明语法兼容，这种格式从语法上来讲虽然没有什么问题，但在 Java 中推荐采用格式 2，即把[]放在数组名前面。

数组元素的类型本质上是定义数组中每个元素的类型，该类型可以是 Java 中的任意类型，既可以是基本数据类型，也可以是类等各种引用数据类型。数组中所有元素的数据类型都是完全一样的。例如，下面的代码声明了一个整数数组 numbers：

```
int[] numbers;
```

再比如，可以声明一个数组，其数据元素的类型为引用类型，例如使用用户自定义的 Student 类，定义一个 Students 数组的声明如下：

```
Student[] students;
```

和C语言等高级语言不同，Java在声明数组时，[]内不能指定长度，这时的数组大小尚不确定，也没有分配相应的数据存储空间。如果想要对数组进行正确的存取访问，必须要创建数组，也就是指定数组长度，分配相应的数组存储空间。

5.2.2 创建数组

在Java中，数组是一种引用类型，因此它和基本数据类型变量的使用不一样。在声明一个数组时，并不在内存中给数组分配任何空间来存放数组中的元素，仅仅声明了一个引用数组的地址变量（又被称为数组的引用）。

数组声明的目的只是告诉系统一个新的数组的名称和类型，数组名本身不能存放任何数组元素，这意味着该数组变量并没有引用任何数据空间，数组变量当前的值为空（null）。因此，使用数组之前，需要先使用new关键字创建数组，为数组分配指定长度的连续内存空间，并把这片连续内存空间的起始地址赋值给数组变量。

1. 创建数组的语法形式

通常在声明数组的同时可以进行创建数组的操作，声明并创建数组，分配内存空间的语句格式如下：

```
数组名 = new 数组元素类型[数组的长度];
```

数组名就是数组变量名，数组的长度就是数组的容量大小，也就是数组中元素的个数，它是一个整数。数组长度存储在数组的length属性中，可以通过数组名.length引用。

也可以单独进行创建数组的操作，例如在声明了数组numbers与students之后，可以使用下面的语句创建数组：

```
numbers = new int[4];
students = new Student[3];
```

new int[4]给数组numbers分配了4个整数的连续内存空间，用来保存4个int类型的数据。分配空间之前的numbers数组变量不引用任何空间，其值为null，如图5-2(a)所示。使用new关键字分配4个连续整数空间后，numbers变量将存放一个空间地址（又称为引用），也即被分配的4个整数的连续空间的起始位置，这个引用地址的取值由JVM自动分配，整个过程如图5-2(b)所示。

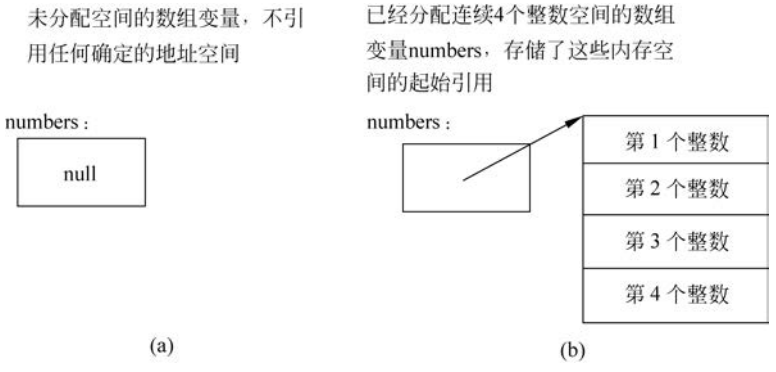

图5-2 创建整数数组分配内存空间

【例 5-1】 从键盘输入 10 个数，计算它们平均值和高于平均值的数量。

```
import java.util.*;
public class Example5_1 {
     public static void main(String[] args) {
         final int NUMBER_OF_ELEMENTS = 10;
         double[] numbers = new double[NUMBER_OF_ELEMENTS];
         double sum = 0;
         System.out.println("下面请输入" + NUMBER_OF_ELEMENTS + "个数");
         Scanner input = new Scanner(System.in);
         for (int i = 0; i < NUMBER_OF_ELEMENTS; i++) {
              System.out.print("请输入一个数: ");
              numbers[i] = input.nextDouble();
              sum += numbers[i];
         }
         double average = sum / NUMBER_OF_ELEMENTS;
         int count = 0;              // 存储高于平均值的数的个数
         for (int i = 0; i < NUMBER_OF_ELEMENTS; i++)
              if (numbers[i] > average)
                   count++;
        System.out.println("这些数字的平均值是:" + average);
        System.out.println("其中高于平均值的数字的数量是:" + count);
    }
}
```

例 5-1 声明了一个一维数组 numbers，数组元素的数量用一个常量 NUMBER_OF_ELEMENTS 表示，并给 NUMBER_OF_ELEMENTS 赋予初值 10，数组元素的数据类型是 double，程序首先用一个 for 循环从键盘读取 10 个数字并计算它们的平均值，然后再用另一个 for 循环计算高于平均值的数字的个数。

2. 对象数组的创建

numbers 数组中的元素都是基本数据类型 int，这种数组一旦完成创建工作，就可以马上对它进行存取访问操作。但是 students 数组比较特殊，数组中的元素本身也是引用类型 Student 类，数组中的元素是 3 个对象，它们都属于引用类型，这是一种对象数组。创建这种元素为引用类型的数组则需要额外进行操作。

当使用 students=new Student[3]创建数组时，系统为数组 students 分配了 3 个连续空间，但它们仅仅可以被用来存放 3 个 Student 对象的引用，3 个 Student 对象本身并没有被创建和分配空间。为了正确对数组中的数据进行存取，还需要为 students 数组中的这 3 个数组元素分别构造 Student 对象实例，否则数组中的每一个元素的引用为 null，并不能进行正确的数据存取访问，这时的 students 数组如图 5-3 所示。

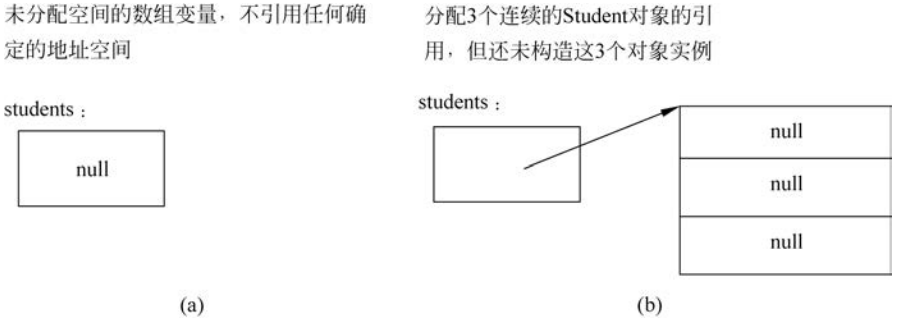

图 5-3 创建 students 数组分配内存空间

因此在使用数组元素为引用类型(如某种对象)的数组时,除了需要声明数组和创建数组,还必须进一步对数组中的每一个数据元素创建内存空间(如构造对象实例),否则在程序运行时将会抛出一个 NullPointerException 的异常。例 5-2 的程序展示了如何创建并存取数据元素为对象的数组。

【例 5-2】 创建一个元素为引用类型的数组,并进行访问存取。

```
class Student {
      String name;
      int age;
      String major;
      Student(String name,int age,String major){
          this.name = name;
          this.age = age;
          this. major = major;
      }
      void study() {
          System.out.println(name + "在学习" + major);
      }
}
public class Example5_2{
      public static void main(String[] args) {
          Student[] students = new Student[3];
          // students[0].age = 20; 这行代码将不能正常执行
          students [0] = new Student("张三",18,"计算机");
          students [1] = new Student("李四",20,"电子工程");
          students [2] = new Student("王五",19,"光学工程");
          students[1].age = 21;                          //修改第 2 名学生的年龄为 21
          for(int i = 0;i < students.length;i++) {       //遍历所有学生
              System.out.println(students[i].name + "," + students[i].age);
              students[i].study();
          }
      }
}
```

例 5-2 首先定义了一个学生类 Student,包含三个成员变量(name、age、major)、一个构造方法 Student(String name,int age,String major)和一个成员方法 study()。在 main()入口方法中则声明并创建了具有 3 个元素的 Student 类型的数组,但这时并不能立即直接对数组进行存取,例如对数组中下标为 0 的元素进行赋值操作,也就是程序中被注释的“students[0].age=20;”这行代码将不能正常运行。例 5-2 在创建完 Student 对象数组后,逐一为数组中的 3 个元素分别构造对象实例,分配空间,然后才能正常对数组进行遍历访问等存取操作。

5.2.3 数组长度

在创建数组的时候,数组长度可以是一个整数常量,也可以是整数变量,其值决定了数组中元素的个数。一旦创建好数组并分配了内存空间,就不能再改变它的长度。在使用数组的过程中,可以使用“数组变量名.length”的语法形式来获取数组的长度值,length 是数组对象的一个成员属性。

例如,在下面的代码中,通过使用 a.length 就可以得到数组变量 a 的长度。

```
int n = 4;
int[ ] a = new int[n];
int len = a.length;
System.out.prinln("数组长度为:" + len);
```

上面的代码首先声明并创建了具有 n 个元素的整数数组，n 的值为 4，然后再通过程序读取数组 a 的长度，并赋值给整数变量 len，最后打印输出 len 的值，输出结果为 4。

5.2.4 数组的初始化

数组创建后，如不对其进行初始化，系统会根据其类型自动为元素赋初始值。

如果数组的元素是基本类型，数组中元素默认初始化的值是基本类型的默认值，基本数据类型的数组元素的默认初值如表 5-1 所示。如果数组元素是对象等引用数据类型，数组元素的默认初值是 null。

表 5-1 基本数据类型的数组元素的默认初值

数据类型	默认初值	数据类型	默认初值
byte	0	char	\u0000
int	0	float	0.0
short	0	double	0.0
long	0	boolean	false

例如，在使用 int[] a=new int[4]创建数组 a 之后，数组 a 中就有 4 个整数，每个整数的值都初始化为 0。

也可以使用赋值语句对数组元素进行初始化赋值。例如下面的语句给数组 a 的 4 个元素分别进行了初始化赋值。

```
a[0] = 12;
a[1] = 30;
a[2] = 18;
a[3] = 55;
```

为了简化上面这种烦琐的赋值操作，Java 语言允许在声明数组的同时就完成数组的初始化操作，例如：

```
int[ ] a = { 12, 30, 18, 55 };
```

这种方法比逐一为数组中每个元素分别赋值要简洁得多，而且自动创建数组的存储空间，不再需要使用 new 创建数组，其数组长度由{ }中元素的个数决定，是一种很常见的初始化操作。

对于数组元素为引用类型的数组，同样也可以通过这种方法将数组的声明、创建和初始化操作合并在一起。例如将例 5-2 中的数组变量 students 的声明进行如下修改，就可以省去分别为数组中的每个元素赋初值的操作。

```
Student[ ] students = new Student[ ]{
            new Student("张三",18,"计算机"),
```

```
            new Student("李四",20,"电子工程"),
            new Student("王五",19,"光学工程")
        };
```

需要强调的是，数组的初始化操作并不是必需的。对于数组元素为基本类型的数组来说，没有经过初始化操作也能被正常地存取。

但是，对于数组元素为引用类型的数组则不同，正如例 5-2 所示，如果没有在数组声明的同时进行初始化，则需要在访问数组之前对数组中的每一个元素赋初值，分配相应的内存引用空间，否则不能对该数组元素进行正确的存取操作。这是初学者在使用数组时很容易忽视的一个地方。

5.2.5 访问数组

在创建数组并初始化之后，就可以对数组中的元素进行存取访问了。由于数组中的元素在内存中是连续有序存放的，因此数组的访问可以通过其在存储空间中的顺序编号，也就是下标来完成，数组下标是从 0 开始的。访问数组元素的语法格式如下：

```
数组名[数组元素下标]
```

例如下面的语句给 numbers 数组中的第 3 个元素赋值 10，然后打印输出到控制台：

```
numbers[2] = 10;
System.out.println("numbers 数组中的第 3 个元素取值是:" + numbers[2]);
```

使用数组时要注意下标值不要超出范围，数组元素的下标范围是[0，数组长度－1]。程序执行时如果访问数组超出这个范围将会抛出一个 ArrayIndexOutBoundException 异常。

数组是连续有序地存放在内存空间中的，在实际应用中，经常可以借助循环来控制对数组元素的访问，访问数组的下标随循环控制变量的变化而变化。因此，数组的访问往往采用 for 循环，这是因为数组的长度一般都是已知的。例如：

```
int n = 100;
int[] a = new int[n];
for (int i = 0; i <= n－1; i++) {
    a[i] = i * i;
}
```

在上面的程序中数组 a 采用循环变量 i 作为下标来遍历访问每一个数组元素，数组的长度为 100，因此数组下标 i 的取值范围为 0～99 的整数，数组的访问只能取 a[0]～a[99]的变量。

对于类似数组这种批量数据的遍历操作，Java 语言还提供了其他循环结构。可以用枚举的方法处理数组中的每个数据元素，而不必指定下标值。这种循环通常被称作 for-each 循环。

采用 for-each 循环来访问数组的语句格式为：

```
for(元素类型 变量名:数组名){
        //操作数组元素
}
```

其中，变量名代表一个临时变量，用来暂存数组中的每一个元素，并在循环体中执行相应语句来操作该临时变量。例如，要输出数组 a 中的所有元素值，可以用如下的代码段：

```
for (int element : a) {
System.out.println( element );
}
```

这段代码打印输出数组中每一个元素的值，每输出一个元素就换一行。

可以把上面的 for-each 循环代码理解为“依次循环访问数组 a 中的每一个元素，将该元素赋值给一个临时变量 element 并在循环体中进行访问或处理”。实际上，这种方法和下面传统的 for 循环执行的效果是等价的。

```
for (int i = 0; i < 100; i++) {
     System.out.println( a[i]);
}
```

对于数组操作，采用 for-each 循环更加简洁，更不容易出错，因为不需要为数组下标起始值和终止值操心。但是，在很多情况下仍然需要使用传统的 for 循环，例如有的时候可能并不需要遍历整个数组，在这种情况下使用下标值来指定访问数组中的部分元素可能更加方便。

当然，采用其他的循环语句来进行数组访问也是一种常见手段，例 5-3 采用了多种循环结构对数组中的批量数据进行存取。

【例 5-3】 从键盘输入全班同学的成绩，并计算平均值与标准差。

假设全班有 n 名同学，可以定义一个数组来存放 n 名同学的成绩。采用下面的公式计算均值与标准差。这里假定 n 小于 200，且当输入成绩小于 0 时，表示输入结束。

均值(avg)的计算：

$$avg = \frac{1}{n}\sum_{i=1}^{n} x_i$$

标准差(sd)的计算：

$$sd = \sqrt{\frac{1}{n-1}\sum_{i=0}^{n-1}(x_i - avg)^2}$$

程序代码如下：

```
import java.util.Scanner;
public class Example5_3 {
    static int size = 200;
    public static void main(String[] args) {
    float[] x = new float[size];
    float avg, sd, t,total;
    int i = 0,n;
    Scanner sc = new Scanner(System.in);
    System.out.println("输入一名同学的成绩");
    total = 0;
    t = sc.nextFloat();
    while (i < 200 && t >= 0){
```

```
            x[i] = t;
            i++;
            total = total + t;                    //计算总成绩
            System.out.println("输入一名同学的成绩,当输入小于 0 时,结束成绩输入");
            t = sc.nextFloat();
        }
        n = i;                                    //共输入了 n 个数据
        if(n > 0){
        avg = total/n;                            //计算平均成绩
        //计算标准差
        sd = 0;
        for (i = 0;i < n;i++)
              sd = sd + (x[i] - avg) * (x[i] - avg);
        if (n > 1){
           sd = (float)Math.sqrt(sd/(n - 1));
           System.out.println("学生人数为:" + n);
           System.out.println("平均成绩为:" + avg);
           System.out.println("成绩的标准差:" + sd);
        }
        }
    }
}
```

程序的一次运行过程及结果如下：

```
输入一名同学的成绩
88
输入一名同学的成绩,输入负数时,结束成绩输入
56
输入一名同学的成绩,输入负数时,结束成绩输入
0
输入一名同学的成绩,输入负数时,结束成绩输入
99
输入一名同学的成绩,输入负数时,结束成绩输入
-5
学生人数为: 4
平均成绩为: 60.75
成绩的标准差: 44.417526
```

在该例中，输入控制用 while 语句实现，在循环中用变量 i 记录数组元素的下标。

观看视频

5.3 数组应用

数组作为一种引用类型，其使用方法与基本数据类型，如 int、float 等存在许多差异，在很多时候，如数组之间赋值、数组被用作方法参数，数组之间的操作不是基本数据类型变量的“传值”，而是通过所谓的“传引用”方式来完成的。下面结合示例来说明数组的这些特殊应用。

5.3.1 数组的赋值

1. 数组赋值

在 Java 语言中，同类型的数组之间可以用“=”赋值，实现把一个数组变量赋值给另外

一个数组的功能。在这种情况下，由于数组本身是引用类型，其值是数组元素的内存空间首地址，因此，数组之间直接赋值，实际上就是数组引用的赋值。例如：

```
int[] num = {4, 6, 3, 7};
int[] numCopy = {8, 1, 0, 9};
numCopy = num;
numCopy[2] = 5;
System.out.println(num[2]);
```

上面的代码将数组 num 赋值给数组 numCopy，也就是把数组 num 的引用赋值给 numCopy 数组，赋值完成后，两个数组都引用了相同的存储空间，因此在修改了 numCopy 数组的第三个元素的值之后，num 数组相应也会发生变化。上面的语句运行后，最后的输出为 5。

如图 5-4(a)所示，在赋值前 num 和 numCopy 指向了内存中不同的空间，是两个完全不同的数组；而在赋值之后，num 和 numCopy 实际上指向了同一个内存空间，如图 5-4(b)所示。因此，当修改数组元素 numCopy[2]时，num[2]的值也就随之改变了。

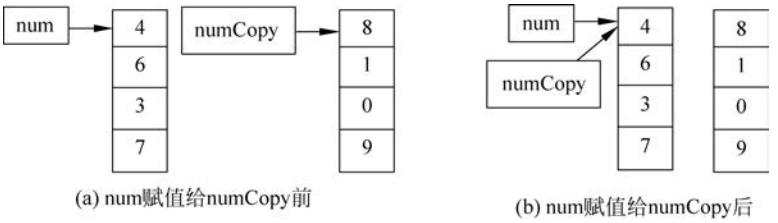

图 5-4 数组变量赋值过程示意图

2. 数组元素赋值

如果仅仅需要将数组中的元素的数值复制给另外一个数组，同时又要保证两个数组保持各自不同的内存引用空间，可以编写一个 for 循环，依次将原来数组中每一个元素的值赋值给新数组。例如下面的语句片段：

```
int[] num = {4, 6, 3, 7};
int[] numCopy = {8, 1, 0, 9};
for (int i = 0; i < num.length; i++) {
numCopy [i] = num [i];
}
```

和图 5-4 不同，图 5-5 中的 for 循环执行后，num 和 numCopy 两个数组中的元素内容完全一样，但数组变量仍然引用各自不同的内存空间。

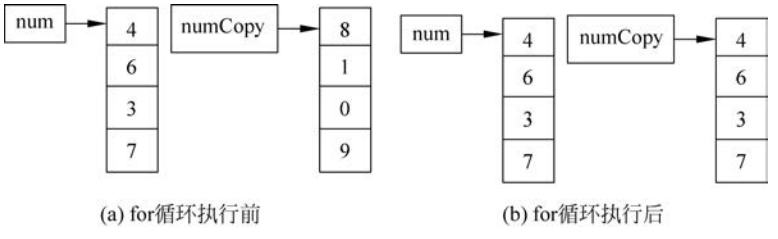

图 5-5 数组循环复制元素过程示意图

还可以采用更加简便的方法实现数组元素的内容复制，也就是采用 System 类的 arrayCopy()方法，其格式如下：

```
System.arraycopy(src, srcPos, dest, destPos, length)
```

该方法可以将 src 源数组中从 srcPos 开始的连续 length 个元素复制到 dest 数组的 destPos 开始的位置，且 src 和 dest 数组指向不同的内存空间。于是，上面的数组复制代码片段可以重写为如下：

```
int[] num = {4, 6, 3, 7};
int[] numCopy = {8, 1, 0, 9};
System.arraycopy(num, 0, numCopy, 0, num.length);
```

5.3.2 数组参数传递

和其他高级语言类似，在 Java 语言中数组变量也可以作为参数传递给方法。但是和整数、浮点数等基本类型的变量传值不同，数组是一种引用类型，传递的是数组的引用，在使用时需要注意两者之间的区别。

当一个方法的参数是基本类型（如整数、浮点数）时，方法中对形式参数的任何修改不会影响调用时的实际参数。然而，当参数是数组变量等引用类型时，在方法中对形式参数作出的修改，将导致实际参数也发生相应的变化。例如下面的例 5-4。

【例 5-4】 数组作为方法的参数示例。

```
public class Example5_4 {
     public static void main(String[] args) {
          int x = 1;
          int[] y = new int[10];
          m(x,y);
          System.out.println("x is " + x);
          System.out.println("y[0] is " + y[0]);
     }

     public static void m(int number, int[] numbers) {
          number = 1001;
          numbers[0] = 5555;
     }
}
```

程序的运行结果如下：

```
x is 1
y[0]is 5555
```

在例 5-4 的程序中，方法 m()修改了两个形式参数 number 和 numbers 的值。在 main()中调用了 m()方法，并传入了实际参数 x 和 y。m()方法执行后，x 的值并没有改变，而数组变量 y 由于是引用类型，在调用 m()方法后，其数组元素的值也相应发生了变化。

5.3.3 数组作为方法的返回值

数组不但可以是方法的参数，也可以作为一个方法的返回值。例如例 5-5 程序中的方法 reverse()，它将参数传进来的数组变量 list 中的数据顺序倒置，并将倒置后的新数组 result 返回给调用者。

【例 5-5】 数组作为方法的返回值。

```
public class Example5_5{
     public static int[ ] reverse(int[ ] list) {
          int[ ] result = new int[list.length];
          for (int i = 0, j = result.length - 1; i < list.length; i++, j -- ) {
               result[j] = list[i];
          }
          return result;
     }

     public static void main(String[ ] args) {
          int[ ] x = {1,2,3,4,5,6,7,8};
          x = reverse(x);
          System.out.println("返回后的 x 数组中元素:");
          for (int t:x)
               System.out.print(t + "\t");
     }
}
```

程序的运行结果如下：

```
返回后的 x 数组中元素:
8  7  6  5  4  3  2  1
```

5.3.4 一维数组编程举例

数组是一种有序存储的数据集合，排序是这种数据结构的常见操作，例 5-6 和例 5-7 为数组在排序方面的相关应用。

【例 5-6】 从键盘逐个输入学生的成绩，并存储到数组，然后按从小到大的顺序进行排列并输出。

```
import java.util.Scanner;
public class Example5_6{
    public static void main(String[ ] args) {
        int numberOfStudent;
        int scores[ ];
        Scanner sc = new Scanner(System.in);
        System.out.print("请输入学生人数: ");
        numberOfStudent = sc.nextInt();
        scores = new int[numberOfStudent];
        for (int i = 0; i < numberOfStudent; i++) {
            System.out.print("请输入第 " + (i + 1) + " 位学生的成绩:");
            scores[i] = sc.nextInt();
        }
```

```
        // 双重循环,进行冒泡排序
        for (int j = 0; j < scores.length; j++) {
            for (int k = 0; k < scores.length - j - 1; k++) {
                int l;
                if (scores[k] > scores[k + 1]) {
                    l = scores[k];
                    scores[k] = scores[k + 1];
                    scores[k + 1] = l;
                }
            }
        }
        System.out.println("学生成绩从小到大排列如下:");
        for (int i = 0; i < numberOfStudent; i++)
            System.out.print(scores[i] + "\t");
    }
}
```

程序的运行结果如下：

```
请输入学生人数: 5
请输入第 1 位学生的成绩: 78
请输入第 2 位学生的成绩: 98
请输入第 3 位学生的成绩: 79
请输入第 4 位学生的成绩: 66
请输入第 5 位学生的成绩: 93
学生成绩从小到大排列如下:
66  78  79  93  98
```

例 5-6 把学生的成绩数据保存在一个整数数组中，学生的个数也就是数组的长度，由键盘录入。程序首先用循环语句从键盘读入所有学生的成绩到数组中，然后采用了一种经典的排序算法——冒泡排序法来对数组中的元素按从小到大的顺序进行排序，最后再输出整个数组的数据。

【例 5-7】 在一个有序数组中插入一个数据并使该数组保持有序(默认数组为升序排列)。

```
import java.util.Scanner;
public class Example5_7 {
    public static void main(String[] args) {
        //在有序数组中插入一个元素,使得插入后所有的元素也保持有序,这里以升序为例
        int[] number = {1,5,7,9};
        //定义一个新数组,长度为老数组的长度 + 1
        int[] number1 = new int[number.length + 1];
        int temp;
        System.out.println("要插入元素的有序数组如下所示:");
        for (int i = 0; i < number.length; i++) {
            System.out.print(number[i] + "\t");
        }
        System.out.println();
        System.out.print("请输入插入的数:");
        Scanner scanner = new Scanner(System.in);
        int insert_number = scanner.nextInt();
        for (int i = 0; i < number.length; i++) {
```

```
            number1[i] = number[i];
        }
        number1[number1.length - 1] = insert_number;  //先将待插入数据放到数组末尾
        //通过比较,找到插入数据的位置
        for (int i = number1.length - 1;i > 0 ; i -- ) {
            if (number1[i]< number1[i - 1]){
               temp = number1[i];
               number1[i] = number1[i - 1];
               number1[i - 1] = temp;
            }else
                break;
        }
        System.out.print("插入数据后的结果: ");
        for (int i = 0; i < number1.length; i++) {
            System.out.print(number1[i] + "\t");
        }
    }
}
```

程序的运行结果如下：

```
要插入元素的有序数组如下所示:
1  5  7  9
请输入插入的数: 3
插入数据后的结果: 1  3  5  7  9
```

5.4 Arrays 类与应用

Java 语言的工具包 util 中提供了工具类 Arrays，该类定义了常见操作数组的静态方法，可以更便捷地进行数组操作(如排序和搜索等)。

(1) equals()用于比较两个数组是否相等，例如 Arrays. equeals(int[],int[])可以比较两个整数数组并返回一个 boolean 类型的结果值，true 为相等，false 为不相等。

```
int[] a = {1,2,3};
int[] b = {1,2,3};
boolean isSame = Arrays.equals(a,b);
```

在上述代码中，isSame 变量的最后值为 true。需要注意，Arrays. equals(a,b)是比较数组 a 和 b 中的数据内容是否相等，而 a. equals(b)这样的方法是比较两个数组的地址引用值，也即判断 a 和 b 两个数组是否是引用的同一个内存空间。

(2) fill()用于以某个数值填充整个数组或者指定范围的数组。

```
String[] a = new String[6];
Arrays.fill(a, "Hello");
Arrays.fill(a, 3, 5,"World");
```

最后数组 a 中的内容是{Hello，Hello，Hello，World，World，Hello}。

(3) sort()用于对数组排序，默认按照从小到大的顺序排列。

```
int [ ] arr = {12,21,13,24};
Arrays.sort(arr);
```

（4）copyOf()方法可以实现数组的复制功能，该方法有两个形式参数，第一个参数是原数组，第二个参数是复制长度，返回值是将原数组内容进行复制并返回。请注意，该方法返回的是一个新数组引用，即把数据复制到新的存储单元中，然后返回存储单元的引用。

```
int[ ] arr1 = new int[ ]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int[ ] arr2 = new int[5];
arr2 = Arrays.copyOf(arr1, 10);
```

以上方法均有重载定义，可以操作不同数据类型的数组，具体内容可以参考相关的JDK文档。例5-8中的程序代码使用Arrays类实现了数组的排序功能。

【例5-8】 使用Arrays类进行数组元素排序。

```
import java.util.Arrays;
public class Example5_8{
     public static void main(String[ ] args) {
          int k;
          int baka[ ] = new int[11];
          int a[ ] = { 19, 22, 15, 13, 1, 0, 10, 8, 2, 4, 36 };
          System.out.println("\t\t排序前a数组的各元素为:");
          for (k = 0; k < a.length; k++) {
              System.out.print(a[k] + "\t");
              baka[k] = a[k];
          }
          System.out.println();
          Arrays.sort(a);
          System.out.println("\t\t完全排序后a数组的各元素为:");
          for (k = 0; k < a.length; k++)
              System.out.print(a[k] + "\t");
          System.out.println();
          for (k = 0; k < baka.length; k++) {
              a[k] = baka[k];
          }
          Arrays.sort(a, 3, 8);
          System.out.println("\t\t部分(下标第3至第7元素)排序后a数组的各元素为:");
          for (k = 0; k < a.length; k++)
              System.out.print(a[k] + "\t");
          System.out.println();
     }
}
```

程序的运行结果如下：

```
        排序前a数组的各元素为:
19  22  15  13  1  0  10  8  2  4  36
        完全排序后a数组的各元素为:
0  1  2  4  8  10  13  15  19  22  36
        部分(下标第3至第7元素)排序后a数组的各元素为:
19  22  15  0  1  8  10  13  2  4  36
```

仔细观察可以发现，和例 5-6 不同，这段程序没有使用经典的冒泡排序算法来对数组元素进行排序，而是采用 Arrays 类自带的排序功能，Array. sort()不仅可以对整个数组排序，还可以对数组中的部分元素排序。从这里可以看出，用 Java 语言编程时可以选择不同的方式来达到相同的目的，这增加了 Java 语言编程开发的灵活性。

【例 5-9】 彩票号码抽彩程序。

编写一个程序，可以生成一个抽彩游戏中的中奖号码的数字组合。例如从 1～n 的数字中随机抽取 k 个号码，则这 k 个号码构成了中奖号码。

```
import java.util.Arrays;
import java.util.Scanner;
public class Example5_9{
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("本次抽彩将抽取多少个数字?");
        int k = in.nextInt();
        System.out.print("抽取的数字最大值是多少? ");
        int n = in.nextInt();
        in.close();
        //初始化抽彩数字数组中的元素为 1,2,3…,n
        int[] numbers = new int[n];
        for (int i = 0; i < numbers.length; i++)
            numbers[i] = i + 1;
        //抽取 k 个数字,并把它们放到另外一个数组 result 中
        int[] result = new int[k];
        for (int i = 0; i < result.length; i++) {
            //生成一个 0～n - 1 的随机整数
            int r = (int) (Math.random() * n);
            //从数组 numbers 中随机抽取一个数字放到 result 数组中
            result[i] = numbers[r];
            //将 numbers 数组的最后一个数字放到已被抽走的数字所在的位置
            numbers[r] = numbers[n - 1];
            n--;
        }
        //输出排序后的抽彩数字组合
        Arrays.sort(result);
        System.out.println("抽取下列数字组合,你将获得大奖!");
        for (int r : result)
            System.out.println(r);
    }
}
```

程序的一次运行结果如下：

```
本次抽彩将抽取多少个数字?5
抽取的数字最大值是多少?34
抽取下列数字组合,你将获得大奖!
8
11
28
30
34
```

本程序首先构建了一个数组 numbers，用来存放所有供抽彩的数字，然后通过循环进行抽彩，每一次循环则随机从 numbers 中选取一个数字作为中奖号码，并把该数字存放在一个新数组 result 中。程序最后用 Arrays 类的静态方法 sort()对 result 数组按从小到大进行排序，并用 for-each 循环方式来输出 result 数组中所有中奖的号码。

需要注意的是，这段程序代码采用了 Math 类的静态方法 random()来生成随机数；此外，为了确保每次抽取的数字不会和以前抽取的数字重复，程序每抽取一个数，就将 numbers 数组的最后一个数字复制到被抽取的位置，以替换掉被抽取的数字，同时把 numbers 数组的长度值 n 减 1，这样就可以保证 numbers 数组中永远只保留有尚未被抽中的数字。

观看视频

5.5 二维数组

前面介绍的一维数组可以存储各种线性数据的集合。假设有如表 5-2 所示的一个学生成绩信息表，如果直接用一个整数类型的一维数组来存储则比较困难。对于这样的二维表格或者网格数据，则可以考虑使用二维数组进行存储和处理。

表 5-2 学生成绩信息表格 单位：分

语　文	数　学	英　语
85	89	91
78	83	80
92	85	89
81	90	88

5.5.1 二维数组的声明、创建和使用

如果 Java 语言一维数组的每个元素又都是一维数组，这个数组就称为二维数组。多维 Java 数组的定义实际上是一个递归的过程，三维数组可以看成是由二维数组组成的一维数组；以此类推，n 维数组可以看成每个元素都为 n－1 维数组的一维数组。

1. 二维数组的声明

声明二维数组有以下两种格式。

格式 1：

```
数组元素类型 数组名[][];
```

格式 2：

```
数组元素类型[][] 数组名;
```

例如定义一个存储学生成绩的二维数组：

```
double scores [][];
```

或

```
double [][] scores;
```

和一维数组类似，在 Java 语言中通常采用格式 2。

2. 二维数组的创建与初始化

与一维数组的创建方法一样，二维数组也是使用 new 关键字来申请内存存储空间的。例如下面的语句先声明了一个二维数组 scores，然后使用 new double[4][3]创建一个 double 类型的二维数组。

```
double[][] scores;
scores = new double[4][3];
```

该二维数组包含两个维度的数据，可以把第一个维度的取值 4 看作二维数据表中的行，把第二个维度的取值 3 看作二维数据表中的列，这样整个数组 scores 就是一个 4 行 3 列的数据表，可以非常方便地用来存储表 5-2 中的学生信息表。通常情况下，二维表格数据都可以采用 Java 语言的二维数组来进行数据处理。

Java 语言允许把声明和创建二维数组合并在一条语句中，例如上面的 scores 数组的创建可以简化成下面的代码：

```
double[][] scores = new double[4][3];
```

与一维数组的初始化一样，如果在创建数组时没有给出初始值，数组中各元素按不同数据类型的默认值来取值。如果在创建数组时有初始值，可以把声明、创建数组与初始化操作合并在一起完成，其格式如下：

```
类型[][] 数组名 = {{初值表 11},{初值表 12},…, {初值表 1n},{初值表 21},{初值表 22},…, {初值表 2n},…};
```

其中，从{初值表 1}到{初值表 n}分别代表每一行数据的初始值，例如采用表 5-2 中的数据对 scores 数组进行声明和初始化的语句如下：

```
double[][] scores = {
            {85 , 89 ,91},
            {78 , 83 ,80},
            {92 , 85 ,89},
            {81 , 90 ,88}
            };
```

3. 二维数组的长度

和一维数组类似，二维数组的长度也可以通过 length 属性来获取。但是二维数组有两个维度，可以通过 length 分别获取一个二维数组的行数以及每一行的长度。获得行数的方法如下：

```
数组名.length
```

获得指定行所包含列数的方法如下：

```
数组名[行号].length
```

假设定义并初始化一个二维数组如下：

```
int[][] matrix = new int[10][10];
```

下面的代码将通过一个嵌套的双重循环访问该二维数组，打印输出该二维数组的所有元素的内容：

```
for (int row = 0; row < matrix.length ; row++) {
    for (int column = 0; column < matrix[row].length ; column++) {
        System.out.print(matrix[row][column] + " ");
    }
    System.out.println();
}
```

4. 二维数组的访问示例

【例 5-10】 输出具有初始值的数组中的所有数据。

```
public class Example5_10 {
public static void main(String[] args) {
         double[][] scores ={{56.8,42.5,96.8},
                             {100,78},
                             {99,63,78,45}};
         int i,j;
         System.out.println("输出 scores 数组中所有的值");
         for(i = 0;i < scores.length;i++) {
             for (j = 0; j < scores[i].length; j++)
                 System.out.print(scores[i][j] + "\t");
             System.out.println();
         }
     }
}
```

程序的运行结果如下：

```
输出 score 数组中所有的值
56.8    42.5    96.8
100.0   78.0
99.0    63.0    78.0    45.0
```

从例 5-10 可以看出，Java 语言二维数组中每一行的数组元素可以不同，scores 二维数组的第一行有 3 个数，第二行有 2 个数，而第三行却有 4 个数。

其实，Java 语言二维数组的每一行数据就是一个一维数组，本质上二维数组就是由若干一维数组构成的，且每个一维数组的长度可以各不相同。在例 5-10 中，二维数组 scores 可以被看作具有 3 个元素的普通一维数组，且每个元素也是一维数组，即 3 个一维数组，它们分别是 scores[0]、scores[1]和 scores[2]，长度分别为 3、2、4，而 scores 数组的长度则为 3。关于 scores 二维数组和一维数组之间的关系，及其在内存中的存储方式可参见图 5-6。

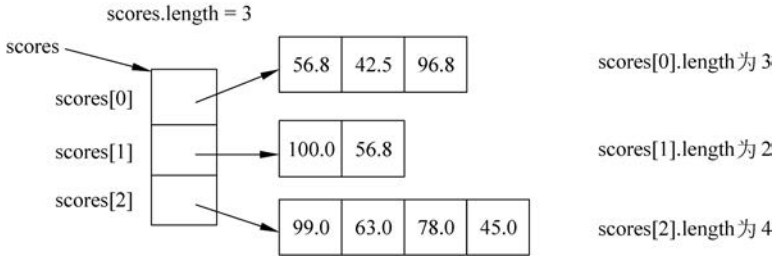

图 5-6 二维数组 scores 的存储示意图

二维数组的这种存储特点使得数组的初始化可以更加灵活，例如可以在初始化二维数组时不指定数组的全部长度，仅仅给出第一个维度的大小：

```
int[][] data = new int[3][];
data[0] = new int[2];
data[1] = new int[3];
```

上面的语句中在 data 数组初始化时只给出了第一维的长度大小，也就是行数，但第二维的长度却省略掉了。程序可以在后续处理中，根据实际情况确定 data 数组每一行的数据长度大小。根据图 5-6 容易看出，先初始化数组的行数，再初始化列数是完全可行的；但反过来先初始化列数，再初始化行数却不可以。

5.5.2 二维数组的应用案例

二维数组的应用场景有很多，比较常见的有二维数据的分析和处理、矩阵运算、科学计算等，其特点是数据具有比较明显的二维结构，程序通常采用多重循环处理数组中的数据。下面给出两个完整的程序，分别介绍二维数组在二维数据存储分析和矩阵运算方面的应用。

【例 5-11】 有一门考试的试卷有 10 道选择题，共有 8 名学生参加考试，表 5-3 中存储了 8 名学生的试卷答题结果，其中每一行表示一名学生的 10 道题目的答题结果。假设 10 道选择题每题 1 分，满分 10 分。10 道题的标准答案为 D B D C C D A E A D。

编写一个程序，用二维数组来存储学生的答题结果，并批改试卷输出每名学生的得分情况。

表 5-3 学生答题数据表

	题 号									
学号	0	1	2	3	4	5	6	7	8	9
0	A	B	A	C	C	D	E	E	A	D
1	D	B	A	B	C	A	E	E	A	D
2	E	D	D	A	C	B	E	E	A	D
3	C	B	A	E	D	C	E	E	A	D
4	A	B	D	C	C	D	E	E	A	D
5	B	B	E	C	C	D	E	E	A	D
6	B	B	A	C	C	D	E	E	A	D
7	E	B	E	C	C	D	E	E	A	D

```
public class Example5_11 {
    public static void main(String[] args) {
         char[][] answers = { { 'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D' },
                  { 'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D' },
                  { 'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D' },
                  { 'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D' },
                  { 'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D' },
                  { 'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D' },
                  { 'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D' },
                  { 'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D' }, };
        char[] keys = { 'D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D' };
        for (int i = 0; i < answers.length; i++) {
             int correctCount = 0;
             for (int j = 0; j < answers[i].length; j++) {
                if (answers[i][j] == keys[j])
                      correctCount++;
            }
            System.out.println("学生 " + i + " 的得分是: " + correctCount);
        }
    }
}
```

上面的程序首先初始化了一个二维数组用来存储学生答题结果，然后用双重循环进行评分汇总统计，内循环用来累加一名学生每道题目的得分，外循环完成批改并输出每一名学生的总得分。程序的运行结果如下：

```
学生 0 的得分是: 7
学生 1 的得分是: 7
学生 2 的得分是: 7
学生 3 的得分是: 8
学生 4 的得分是: 4
学生 5 的得分是: 5
学生 6 的得分是: 6
学生 7 的得分是: 7
```

【例 5-12】 通过键盘输入两个矩阵的数据，程序打印输出两个矩阵相乘的最后结果数据。

首先对矩阵相乘的算法进行分析，可以得出以下 3 个规律。

（1）当矩阵 **A** 的列数(column)等于矩阵 **B** 的行数(row)时，A 与 B 可以相乘。

（2）乘积矩阵 **C** 的行数等于矩阵 **A** 的行数，乘积矩阵 **C** 的列数等于矩阵 **B** 的列数。

（3）乘积矩阵 **C** 的第 m 行、第 n 列的元素等于矩阵 **A** 的第 m 行的元素与矩阵 **B** 的第 n 列对应元素的乘积之和。

根据以上规律，两个矩阵在进行乘法操作之前需要进行验证，只有满足一定的条件才可以相乘。

矩阵是一种典型的二维结构，因此采用二维数组来存储和计算矩阵是一种常见的手段。本例共声明了 3 个二维数组，分别用来保存两个相乘的矩阵和最后计算结果矩阵。

```
import java.util.Scanner;
public class Example5_12 {
    private static int[][] input() {
        Scanner s = new Scanner(System.in);
        System.out.println("请输入矩阵的行数和列数:");
        int x = s.nextInt();
        int y = s.nextInt();
        int[][] array = new int[x][y];              //初始化数组
        for (int i = 0; i < x; i++){
            System.out.println("请输入矩阵的第" + (i + 1) + "行数据:");
            for (int j = 0; j < y; j++)
                array[i][j] = s.nextInt();
        }
        System.out.println("你输入的矩阵为" + x + "行" + y + "列:");
        for (int i = 0; i < x; i++){
            for (int j = 0; j < y; j++)
                System.out.print(array[i][j] + "\t");
            System.out.println();
        }
        return array;
    }
    private static void multiplicationMatrix (int[][] a,int[][] b){
        int[][] result = new int[a.length][b[0].length];;
        //根据规律用 3 个 for 循环实现矩阵相乘
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                for (int k = 0; k < b[j].length; k++) {
                    result[i][k] += a[i][j] * b[j][k];
                }
            }
        }
        //将结果输出到控制台
        System.out.println("矩阵相乘的结果为:");
        for (int[] row : result){
            for (int cloumn : row) {
                System.out.print(cloumn + "\t");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        int[][] matrix1,matrix2;
        System.out.println("输入第一个矩阵的数据:");
        matrix1 = input();
        System.out.println("输入第二个矩阵的数据:");
        matrix2 = input();
        if (matrix1[0].length!= matrix2.length){
            System.out.println("这两个矩阵不能相乘!");
        }
        else{
            multiplicationMatrix(matrix1,matrix2);
        }
    }
}
```

5.6 多维数组

前面介绍了如何使用二维数组来表示一个二维数据结构，二维数组通常可以用来存储矩阵或者表格数据。为了处理更多维度的数据，Java 语言允许创建一个 n 维数组，这里的 n 可以是任意一个整数值，也就是所谓的多维数组。

在例 5-10 的程序中声明的二维数组 double[][] scores 可以看作由 3 个一维的数组 scores[0]、scores[1]和 scores[2]构成，类似地，一个 n 维数组也可以理解为由多个 n－1 维的数组组成。

因此，一个三维数组实际上可以看作由多个二维数组组成的一维数组，该数组中的每个元素都是一个二维数组。于是，可以沿用声明和创建一个二维数组的方法来声明和创建一个多维数组。例如下面的语法声明并初始化了一个三维数组：

```
double[][][] data = new double[2][3][5];
```

data 数组有 2 个元素 data[0]和 data[1]，它们分别都是一个 3 行 5 列的二维数组。而 data[0][0]、data[0][1]、data[1][0]、data[1][1]都分别是含有 5 个数据元素的一维数组。data.length 是 2，data[0].length 和 data[1].length 是 3，data[0][0].length 是 5。

和二维数组一样，三维数组在初始化时也可以只给出高维的长度，低维长度可以在后续程序创建和初始化时再来确定，例如下面的语句初始化了一个三维数组，数组含有 2 个元素，每个元素是一个二维数组，二维数组的大小暂时还未确定：

```
double[][][] data = new double[2][][];
```

以此类推，按照三维数组的这种定义方式，可以定义与创建更多维的数组。在实际应用中，多维数组往往被用来完成一些科学计算或者复杂数据的统计分析任务，程序的时间复杂度一般比较高。

习题

1. 下面的代码片段的输出是什么？

```
int[][] array = new int[5][6];
int[] x = {1, 2};
array[0] = x;
System.out.println("array[0][1] is " + array[0][1]);
```

2. 下面的数组声明语句都是合法的吗？如果不合法请说明原因。

```
int i = new int(30);
double d[] = new double[30];
```

```
int i[] = (3, 4, 3, 2);
int[][] r = new int[2];
int[][] y = new int[3][];
```

3. 下面的两个代码片段能否正常运行，如果能请写出运行结果，如果不能请说明为什么。

程序 1：

```
int[] a = {1, 2, 3};
for (int i = 0; i < a.length; i++) {
     a[i]++;
}
System.out.println(a[0]);
```

程序 2：

```
int[] a = {1, 2, 3};
int[] b = new int[3];
b = a;
b[0] = 0;
System.out.println(a[0]);
```

4. 声明一个数组，保存一名学生的数学、语文、英语、物理、化学等课程的成绩，编写一个程序，计算 5 门课程的平均成绩，精确到 0.1 分，成绩值从键盘录入。

5. 编程实现统计 50 名学生的百分制成绩中各分数段的学生人数，即分别统计出 100 分、90～99 分、80～89 分、70～79 分、60～69 分、不及格的学生人数。

6. 编写程序，可以从键盘录入若干阿拉伯数字，其个数不确定。键盘输入完毕后按 Enter 键，程序将打印显示出所有不重复的数字，下面是一个可能的程序运行结果。

```
请输入若干阿拉伯数字：1 2 3 4 1 6 3 4 5 2
所有不同的数字：1 2 3 6 4 5
```

7. 声明一个字符串数组，将 "apple"、"banana"、"cherry"、"date"和"elderberry"添加到数组中，并使用循环计算每个字符串中的元音字母数。

8. 编写一个 eliminateDuplicates()方法，该方法接收一个整型数组，返回去掉重复元素的数组。写个主方法调用该方法并验证方法的正确性。eliminateDuplicates()方法的格式如下所示：

```
public static int[] eliminateDuplicates(int[] numbers)
```

9. 编写一个实现两个二维数据相加的方法，该方法接收两个相同大小的二维数组，返回数组相加的结果。

10. 编写一个方法，该方法接收一个二维数组，返回转置后的二维数组。并编写入口方法，测试其是否可以正常工作。

11. 编程打印输出如下所示的杨辉三角形。

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  78  56  28  8   1
  1   9   36  84  126 126 84  36  9   1
```