第5章

数　组

数组是几乎所有程序设计语言都提供的一种数据存储结构。Java 语言的数组是一种引用数据类型，即数组是对象。数组在程序设计中具有广泛应用，它通常用来存储和操作一组类型相同的数据。本章首先介绍数组的创建和使用，然后介绍数组的有关应用、Arrays 类的使用和二维数组。

本章内容要点

- 创建和使用数组。
- 可变参数方法。
- 数组的排序和查找。
- java.util.Arrays 类。
- 二维数组及应用。

5.1　创建和使用数组

数组(array)是一种数据结构，它用来存储一组类型相同的数据，它是一个引用类型变量，通过下标来访问数组中的元素。下面介绍如何声明、初始化和使用数组。

5.1.1　声明和创建数组

使用数组一般需要两个步骤：①声明数组，即声明数组名称和元素的数据类型；②创建数组，即为数组元素分配存储空间。

1. 声明数组

使用数组之前需要声明。声明数组就是告诉编译器数组名和数组元素类型。数组声明可以使用下面两种等价形式。

```
elementType [ ]arrayName;
elementType arrayName[ ];
```

这里，elementType 为数组元素类型，它可以是基本数据类型(如 boolean 或 char 型)，也可以是引用数据类型(如 String 或 Employee 型等)。arrayName 为数组名，它是一个引用变量。方括号指明变量为数组变量，它既可以放在变量前面也可以放在变量后面。推荐放在变量前面，这样更直观。

例如，下面声明了 marks 和 staffs 两个数组。

```
double [ ]marks;
Employee [ ]staffs;
```

数组声明不能指定数组元素的个数。下面的代码发生编译错误。

```
double [5]marks;
```

上面声明的数组，它们的元素类型分别为 double 和 Employee 型。在 Java 语言中，**数组是引用数据类型**。也就是说，数组是一个对象，数组名就是对象名（或引用名）。数组声明实际上是声明一个引用变量。如果数组元素为引用类型，则该数组称为**对象数组**，如上面的 staffs 就是对象数组。所有数组都继承了 Object 类，因此可以调用 Object 类的所有方法。

2. 创建数组

数组声明仅仅声明一个数组对象引用，而**创建数组是为数组的每个元素分配存储空间**。创建数组使用 new 语句，一般格式为：

```
arrayName = new elementType[size];
```

该语句的功能是分配 size 个 elementType 型的存储空间，并通过 arrayName 来引用。例如：

```
marks = new double[5];                    // 数组包含 5 个 double 型元素
staffs = new Employee[3];                 // 数组包含 3 个 Employee 型元素
```

Java 数组的大小可以在运行时动态指定。比如，下面的代码是合法的。

```
int n = 5;
double [ ]marks = new double[n];          // 也是创建包含 5 个 double 型元素的数组
```

数组的声明与创建可以写在一条语句中。例如：

```
double [ ]marks = new double[5];
Employee [ ]staffs = new Employee[3];
```

当用 new 运算符创建一个数组时，系统就为数组元素分配了存储空间，这时系统根据指定的长度创建若干存储空间并为数组的每个元素指定默认值。数值型数组元素默认值是 0，字符型元素的默认值是'\u0000'，布尔型元素的默认值是 false。如果数组元素是引用类型，其默认值是 null。

上面两条语句分别分配了 5 个 double 型和 3 个 Employee 型的空间，并且每个元素使用默认值初始化。上面两条语句执行后的结果如图 5-1 所示。数组 marks 的每个元素都被初始化为 0.0，而数组 staffs 的每个元素被初始化为 null。

对于引用类型数组（对象数组），还要为每个数组元素分配引用空间，也就是创建每个元素对象。对于 staffs 数组，创建元素的代码如下。

```
staffs[0] = new Employee("张三", 28, 8000);
staffs[1] = new Employee("王五", 30, 8500);
staffs[2] = new Employee("李四", 26, 6000);
```

上面的语句执行后 staffs 数组的结果如图 5-2 所示。

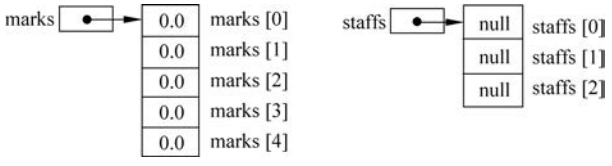

图 5-1　marks 数组和 staffs 数组

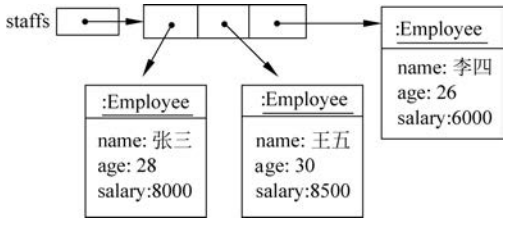

图 5-2　staffs 数组创建后的结果

5.1.2　访问数组元素

声明了一个数组，并使用 new 运算符为数组元素分配内存空间后，就可以使用数组中的

每一个元素。数组元素的使用方式如下：

```
arrayName [index]
```

其中，index 为数组元素的下标或索引，**下标从 0 开始，到数组的长度减 1**。例如，上面定义的 staffs 数组定义了 3 个元素，所以只能使用 staffs[0]、staffs[1]和 staffs[2]这三个元素。数组一经创建，大小不能改变。

数组作为对象提供了一个 length 成员变量，它表示数组元素个数，访问该成员变量的方法为 arrayName. length。

程序 5.1 演示了数组的使用和 length 成员的使用。

程序 5.1　ArrayDemo. java

```
package com.boda.xy;
public class ArrayDemo {
    public static void main(String[ ] args) {
        var marks = new double[5];
        marks[0] = 79;
        marks[1] = 84.5;
        marks[2] = 63;
        marks[3] = 90;
        marks[4] = 98;
        System.out.println(marks[2]);
        System.out.println(marks.length);
        for (var i = 0; i < marks.length; i++) {
            System.out.print(marks[i] + " " );
        }
    }
}
```

用 for 循环访问数组元素

程序运行结果如下：

```
63.0
5
79.0 84.5 63.0 90.0 98.0
```

为了保证安全性，Java 运行时系统要对数组元素的范围进行越界检查，若数组元素下标超出范围，将抛出 ArrayIndexOutOfBoundsException 运行时异常。例如，下面的代码抛出异常。

```
System.out.println(marks[5]);
```

5.1.3　数组初始化器

声明数组同时可以使用初始化器(initializer)对数组元素初始化，它是在一对花括号中给出数组的每个元素值。这种方式适合数组元素较少的情况，这种初始化也称为静态初始化。

```
var marks = new double[ ]{79, 84.5, 63,90, 98};
var staffs = new Employee[ ]{
                    new Employee("张三", 28, 800),
                    new Employee("王五", 30, 8500),
                    new Employee("李四", 26, 6000),
};
```

用这种方法创建数组不能指定大小，系统根据元素个数确定数组大小。另外，可以在最后一个元素后面加一个逗号，以方便扩充。

上面两个语句还可以写成如下形式：

```
double [ ]marks = {79, 84.5, 63,90, 98};
Employee [ ]staffs = { new Employee("张三", 28, 800),
                       new Employee("王五", 30, 8500),
                       new Employee("李四", 26, 6000),
                     };
```

5.1.4 增强的 for 循环

如果程序只需顺序访问数组中的每个元素，可以使用**增强的 for 循环**。增强的 for 循环可以用来迭代数组和对象集合的每个元素，也叫 **for-each 循环**。它的一般格式为：

```
for(type identifier:expression) {
    // 循环体
}
```

该循环的含义为：将 expression(数组或集合)中的每个元素依次存入 identifier 变量，然后执行一次循环体中的语句。这里，type 为数组或集合中的元素类型。expression 必须是数组或集合对象。

下面使用增强的 for 循环实现求 marks 数组中各元素的和，代码如下：

```
double sum = 0;
for(var score :marks){
      sum = sum + score;
}
System.out.println("总成绩 = " + sum);
```

扫一扫

视频讲解

5.2 数组的应用

本节介绍几个数组的应用，包括数组元素的复制、数组参数和返回值、数组的排序，以及可变参数方法等。

5.2.1 数组元素的复制

经常需要将一个数组的元素复制到另一个数组中。首先会想到将数组元素一个一个复制到目标数组中。设有一个数组 source，其中有 4 个元素，现在定义一个数组 target，与原来数组类型相同，元素个数相同。使用下面的方法将源数组的每个元素复制到目标数组中。

```
int[ ] source = {10,30,20,40};              // 源数组
int[ ] target = new int[source.length];     // 目标数组
for(var i = 0; i < source.length; i++){
    target[i] = source[i];
}
```

除上述方法外，还可以使用 System 类的 arraycopy()方法，格式如下：

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos,int length)
```

其中，参数 src 为源数组名，srcPos 为源数组的起始下标，dest 为目的数组名，destPos 为目的数组下标，length 为复制的数组元素个数。下面的代码实现将 source 中的每个元素复制到数组 target 中。

```
int[ ] source = {10,30,20,40};
int[ ] target = new int[source.length];
System.arraycopy(source, 0, target, 0, 4);
```

使用 arraycopy()方法可以将源数组的一部分元素复制到目标数组中。但要注意，如果目标数组不足以容纳源数组元素，会抛出异常。

程序 5.2 ArrayCopyDemo.java

```
package com.boda.xy;
public class ArrayCopyDemo{
  public static void main(String[ ] args){
    int[ ] a = {1,2,3,4};
    int[ ] b ={8,7,6,5,4,3,2,1};
    int[ ] c = {10,20};
    try{
        System.arraycopy(a, 0, b, 0, a.length);
        // 下面的语句发生异常，目标数组 c 容纳不下原数组 a 的元素
      System.arraycopy(a, 0, c, 0, a.length);
    }catch(ArrayIndexOutOfBoundsException e){
      System.out.println(e);
    }
    for(var elem: b){
      System.out.print(elem + " ");
    }
    System.out.println();
    for(var elem: c){
      System.out.print(elem + " ");
    }
    System.out.println("\n");
    }
}
```

运行程序，输出结果如下：

```
java.lang.ArrayIndexOutOfBoundsException
1  2  3  4  4  3  2  1
10  20
```

注意 不能使用下列方法试图将数组 source 中的每个元素复制到 target 数组中。

```
int[ ] source = {10,30,20,40};
int[ ] target = source;                    // 这是引用赋值
```

上述两条语句实现对象的引用赋值，两个数组引用指向同一个数组对象，如图 5-3 所示。

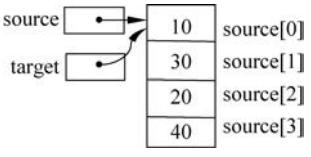

图 5-3 将 source 赋值给 target 的结果

5.2.2 数组参数与返回值

数组对象可以作为参数传递给方法，例如下面的代码定义了一个求数组元素和的方法。

```
public static double sumArray(double array[ ]){
    var sum = 0.0;
    for(var i = 0; i < array.length; i++){
        sum = sum + array[i];
    }
    return sum;
}
```

注意 由于数组是对象，因此将其传递给方法是按引用传递。当方法返回时，数组对象不变。但要注意，如果在方法体中修改了数组元素的值，则该修改反映到返回的数组对象。

一个方法也可以返回一个数组对象。例如，下面的方法返回参数数组的元素反转后的一

个数组。

```
public static int[ ] reverse(int[ ] list){
    var result = new int[list.length];     // 创建一个与参数数组大小相同的数组
    for(int i = 0, j = result.length - 1; i < list.length; i++, j--){
        result[j] = list[i];               // 实现元素反转
    }
    return result;                         // 返回数组
}
```

有了上述方法，可以使用如下语句实现数组反转。

```
int[ ] list = {6, 7, 8, 9, 10};
int[ ] list2 = reverse(list);
```

5.2.3 可变参数的方法

Java 语言**允许定义方法（包括构造方法）带数量可变的参数**，这种方法称为**可变参数**（variable argument）方法。具体做法是在方法参数列表的最后一个参数的类型名之后、参数名之前使用省略号（…）表示。例如：

```
public static double average(double … values){
    // 方法体
}
```

数量可变参数

这里，参数 values 被声明为一个 double 型值的序列。其中参数的类型可以是基本类型，也可以是引用类型。对可变参数的方法，调用时可以为其传递任意数量的指定类型的实际参数。在方法体中，编译器将为可变参数创建一个数组，并将传递来的实际参数值作为数组元素的值，这就相当于为方法传递一个指定类型的数组。请看程序 5.3。

程序 5.3　VarargsDemo.java

```
package com.boda.xy;
public class VarargsDemo{
    public static double average(double … values){
        var sum = 0.0;
        for(var value:values){
          sum = sum + value;                          // 求数组元素之和
        }
        double average = sum / values.length;
        return average;
    }
    public static void main(String[ ] args){
        System.out.println(average(60,70,86));        // 输出:72.0
    }
}
```

程序定义了带可变参数的方法 average()，它的功能是返回传递给该方法的多个 double 型数的平均值。该程序调用了 average()方法并为其传递三个参数，输出结果为 72.0。

在可变参数的方法中还可以有一般的参数，但是可变参数必须是方法的最后一个参数，例如，下面定义的方法也是合法的：

```
public static double average(String name,double … values){
     // 方法体
}
```

注意　在调用带可变参数的方法时，可变参数是可选的。如果没有为可变参数传递

一个值，那么编译器将生成一个长度为 0 的数组。如果传递一个 null 值，将产生一个运行时 NullPointerException 异常。

通常，对带一个数组参数或最后一个参数是数组的方法都可以使用可变参数。例如，Java 应用程序的 main()方法就可以写成下面的形式。

```
public static void main(String … args){}
```

5.2.4 数组的查找

在程序设计中，查找是计算机程序设计中常见的功能。**查找**(searching)是在数组中寻找特定元素的过程。例如，一个数组可能存放学生某科考试成绩，现要求在其中查找是否有某个成绩(如 99 分)。有很多算法用于查找，本节讨论两种常用的算法：**线性查找**(linear searching)和**二分查找**(binary searching)。

1. 线性查找法

线性查找法是将要查找的关键字 key 与数组中的元素逐个进行比较，直到在数组中找到与关键字相等的元素，或者查完所有元素也没有找到。如果查找成功，线性查找法返回与关键字相等的元素在数组中的下标，如果不成功，则返回－1。下面的 linearSearch()方法在数组 array 中查找关键字 key。

```
public static int linearSearch(int[ ] array, int key){
    for (var i = 0; i < array.length ; i++){
      if(array[i] == key)
        return i;
    }
    return -1;
}
```

线性查找法用关键字与数组中的每一个元素进行比较。数组中的元素可以按任意顺序排列。在平均情况下，这种算法需要比较数组中一半的元素。由于线性查找法的执行时间随数组元素个数的增加而线性增加，所以对于大的数组来说其效率不高。

2. 二分查找法

二分查找法是另一种常见的查找法。使用二分查找法的前提条件是数组元素必须已经排序。不失一般性，假设数组按升序排序。二分查找法首先将关键字与数组的中间元素比较，有下面 3 种情况：

(1) 如果关键字比中间元素小，则只需在前一半数组元素中查找。

(2) 如果关键字和中间元素相等，则查找成功，查找结束。

(3) 如果关键字比中间元素大，则只需在后一半数组元素中查找。

显然，二分查找法每比较一次就排除数组中一半的元素。假设用 low 和 high 分别记录当前查找的数组的第一个和最后一个下标。初始条件下，low 为 0，high 为 array.length－1。mid 表示数组中间元素的下标，这样 mid 就是(low＋high)/2。

```
public static int binarySearch(int[ ] array, int key){
  int low = 0;
  int high = array.length - 1;
  while (high >= low){
      int mid = (low + high) / 2;        ← 计算数组中间元素的下标
      if(key < array[mid])
        high = mid - 1;
      else if(key == array[mid])
        return mid;
```

```
            else
                low = mid + 1;
        }
        return - low - 1;
    }
```

首先，关键字 key 与中间元素 array[mid]比较，如果小于中间元素，将 high 设为 mid－1；如果等于中间元素，则匹配成功，返回 mid；如果大于中间元组，将 low 设为 mid ＋ 1。继续这样的查找，直到 low > high 或查找成功。如果 low > high，则返回－low－1，low 就是插入点。

扫一扫

视频讲解

5.3 案例学习——数组起泡排序

1. 问题描述

在程序设计中，**排序**(sorting)是一种常见的操作。例如，一个数组可能存放某个学生的各科成绩，现要求将成绩按从低到高的顺序输出，这就需要为数组排序。排序有多种方法，如起泡排序、选择排序、插入排序、快速排序等。本案例采用**起泡排序法**(bubble sorting)对一个数组按升序排序。

2. 设计思路

假设给定下面的整型数组：

```
int[ ]a = {75, 53, 32, 12, 46, 199, 17, 54};
```

根据问题要求，采用起泡排序法对该数组升序排序，设计思路如下：

从数组的第一个元素开始，与它后面的元素比较，如果逆序(即 a[j] > a[j＋1])，则交换两个元素的位置，如果正序，则不交换。经过第一趟排序后，结果如下：

```
53, 32, 12, 46, 75, 17, 54, 199
```

可以看到，经过第一趟排序，找到了数组中的最大值，而经过比较后，小的元素交换到前面，就像水泡向上冒，起泡排序由此得名。第二趟排序时，只需比较到倒数第二个元素为止，即可找到次大的元素，因此这里使用二重循环结构。

3. 代码实现

根据上面的设计思路，可以将排序过程定义为一个方法，如 bubbleSort(int[] a)，参数是要排序的数组。在主方法中定义要排序的数组，然后调用排序方法即可。实现代码如程序 5.4 所示。

程序 5.4　BubbleSort.java

```
package com.boda.xy;
public class BubbleSort {
    public static void bubbleSort(int[ ] a){
        for(var i = 0; i < a.length; i++) {
            for(var j = 0; j < a.length - i - 1; j ++) {
                if(a[j] > a[j + 1]) {
                    int t = a[j];
                    a [j] = a[j + 1];
                    a[j + 1] = t;
                }
            }
            System.out.print("第 " + (i + 1) + "趟结果:");
            for(var n:a) {
                System.out.print(" " + n);
            }
            System.out.println();
        }
```

若逆序，则交换元素

输出每趟排序后的结果

```
    }

    public static void main(String[ ] args) {
        int[ ]nums = {75, 53, 32, 12, 46, 199, 17, 54};
        //int[ ]nums = {199,75, 54,53, 46,32, 17, 12};
        System.out.print("初始元素:");
        for(var n:nums) {
            System.out.print(" " + n);
        }
        System.out.println();
        bubbleSort(nums);          ← 调用bubbleSort方法对nums数组排序
    }
}
```

4. 运行结果

程序运行结果如图 5-4 所示，这里输出了每趟排序的结果。

使用下节讨论的 java.util.Arrays 类的 sort()方法可以对任何类型元素的数组进行排序，还可以对数组执行查找和其他操作。

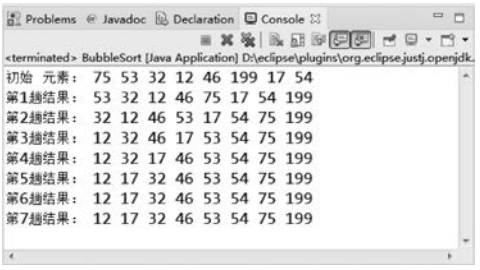

图 5-4 程序运行结果

5.4 java.util.Arrays 类

扫一扫 视频讲解

java.util.Arrays 类定义了若干静态方法对数组操作，包括对数组排序，在已排序的数组中查找指定元素，数组元素的复制，比较两个数组是否相等，将一个值填充到数组的每个元素中。上述操作都有多个重载的方法，可用于所有的基本数据类型和 Object 类型。

5.4.1 数组的复制

使用 Arrays 类的 copyOf()方法和 copyOfRange()方法将一个数组中的全部或部分元素复制到另一个数组中。有 10 个重载的 copyOf()方法，其中 8 个为各基本类型的，2 个为对象类型的。下面给出几个方法的格式。

- static boolean[] copyOf(boolean[] original,int newLength)
- static double[] copyOf(double[] original,int newLength)
- static < T > T[] copyOf(T[] original,int newLength)

这些方法的 original 参数是原数组，newLength 参数是新数组的长度。如果 newLength 小于原数组的长度，则将原数组的前面若干元素复制到目标数组。如果 newLength 大于原数组的长度，则将原数组的所有元素复制到目标数组，目标数组的长度为 newLength。

下面的代码创建了一个包含 4 个元素的数组，将 numbers 的内容复制到它的前三个元素中。

```
int[ ] numbers = {3, 7, 9};
int[ ] newArray = Arrays.copyOf(numbers, 4);
```

当然，也可以将新数组重新赋给原来的变量：

```
numbers = Arrays.copyOf(numbers, 4);
```

与 copyOf()类似的另一个方法是 copyOfRange()，它可以将原数组中指定位置开始的若干元素复制到目标数组中。下面是几个方法的格式。

- static boolean[] copyOfRange (boolean[] original,int from,int to)
- static double[] copyOfRange (double[] original,int from,int to)
- static < T > T[] copyOfRange (T[] original,int from,int to)

上述方法中,from 参数指定复制的元素在原数组中的起始下标,to 参数是结束下标(不包含),将这些元素复制到目标数组中。

```
char[ ] letter = {'a','b', 'c','d','e','f','g'};
letter = Arrays.copyOfRange(letter, 1, 5);
```

上述代码执行后,letter 数组的长度变为 4,其中包含'b'、'c'、'd'和'e'等 4 个元素。

5.4.2 数组的排序

使用 Arrays 的 sort()方法可以对数组元素排序。使用该方法的排序是稳定的(stable),即相等的元素在排序结果中不会改变顺序。对于基本数据类型,按数据的升序排序。对于对象数组的排序要求数组元素的类必须实现 Comparable 接口,若要改变排序顺序,还可以指定一个比较器对象。整型数组和对象数组的 sort()方法格式如下。

- static void sort(int[] a):对数组 a 按自然顺序排序。
- static void sort(int[] a,int fromIndex,int toIndex):对数组 a 中的元素从起始下标 fromIndex 到终止下标 toIndex 的元素排序。
- static void sort(Object[] a):对数组 a 按自然顺序排序。
- static void sort(Object[] a,int fromIndex,int toIndex):对数组 a 中的元素从起始下标 fromIndex 到终止下标 toIndex 的元素排序。
- static < T > void sort(T[] a,Comparator <? super T > c):使用比较器对象 c 对数组 a 排序。

注意 不能对布尔型数组排序。

由于在程序设计中经常使用排序,所以在 java.util.Arrays 类中提供几个重载的 sort()方法,它们用于对 int 型、double 型、char 型、float 型数组的排序。例如,下面的代码为 double 型数组和 char 型数组排序。

```
double[ ] numbers = {5.0, 4.4, 1.9, 2.9, 3.8, 3.5};
java.util.Arrays.sort(numbers);
char[ ] chars = {'a', 'A', '4', 'F', 'D','&'};
java.util.Arrays.sort(chars);
```

下面的代码演示了对一个字符串数组的排序,对字符串排序是按字符的 Unicode 码排序的。

```
String[ ] ss = {"中国", "英国","法国","美国","俄罗斯"};
for(var i = 0;i < ss.length;i++){
    System.out.print(ss[i] + " ");
}
System.out.println();
Arrays.sort(ss);                                    // 对数组 ss 排序
for(var s : ss){
    System.out.print(s + " ");
}
```

代码输出结果为:

```
中国  英国  法国  美国  俄罗斯
中国  俄罗斯  法国  美国  英国
```

5.4.3 元素的查找

对排序后的数组可以使用 binarySearch()方法从中快速查找指定元素，该方法也有多个重载的方法，下面是对整型数组和对象数组的查找方法。

- static int binarySearch(int[] a,int key)
- static int binarySearch(Object[] a,Object key)

查找方法根据给定的键值，查找该值在数组中的位置，如果找到指定的值，则返回该值的下标值。如果查找的值不包含在数组中，方法的返回值为(－插入点－1)。插入点为指定的值在数组中应该插入的位置。

例如，下面的代码输出结果为－3。

```
var a = new int[ ]{1,5,7,3};
Arrays.sort(a);                                    // 返回 a 的元素是:1 3 5 7
var i = Arrays.binarySearch(a,4);
System.out.println(i);                             // 输出:-3
```

在已排序的数组 a 中查找 4，返回的结果－3(－2－1)，它的含义要把 4 插入到数组 a 中，它的下标应该是 2。

注意 使用 binarySearch()方法前，数组必须已经排序。

5.4.4 数组的比较

使用 Arrays 的 equals()方法可以比较两个数组，被比较的两个数组要求数据类型相同且元素个数相同，比较的是对应元素是否相同。对于引用类型的数据，如果两个对象 e1 和 e2 的值都为 null 或者 e1.equals(e2)，则认为 e1 与 e2 相等。

下面是布尔型数组和对象数组 equals()方法的格式。

- static boolean equals(boolean[] a,boolean[] b)：比较布尔型数组 a 与 b 是否相等。
- static boolean equals(Object[] a,Object[] b)：比较对象数组 a 与 b 是否相等。

程序 5.5 给出了 equals()方法的示例。

程序 5.5 EqualsTest.java

```
package com.boda.xy;
import java.util.*;
public class EqualsTest {
  public static void main(String[ ] args) {
      int[ ] a1 = new int[10];
      int[ ] a2 = new int[10];
      Arrays.fill(a1, 47);
      Arrays.fill(a2, 47);
      System.out.println(a1.equals(a2));           // 输出:false
      System.out.println(Arrays.equals(a1, a2));   // 输出:true
      a2[3] = 11;
      System.out.println(Arrays.equals(a1, a2));   // 输出:false
      String[ ] s1 = new String[5];
      Arrays.fill(s1, "Hi");
      String[ ] s2 = {"Hi", "Hi", "Hi", "Hi", "Hi"};
      System.out.println(Arrays.equals(s1, s2));   // 输出:true
    }
}
```

使用数组对象的 equals()方法比较两个引用是否相同。使用 Arrays 类的 equals()方法

比较两个数组对应元素是否相同。

5.4.5 填充数组元素

调用 Arrays 类的 fill()方法可以将一个值填充到数组的每个元素中，也可将一个值填充到数组的几个连续元素中。下面是向整型数组和对象数组中填充元素的方法。

- static void fill(int[] a,int val)：用指定的 val 值填充数组 a 中的每个元素。
- static void fill(int[] a,int fromIndex,int toIndex,int val)：用指定的 val 值填充数组中的下标从 fromIndex 开始到 toIndex 为止的每个元素。
- static void fill(Object[] a,Object val)：用指定的 val 值填充对象数组中的每个元素。
- static void fill(Object[] a,int fromIndex,int toIndex,Object val)：用指定的 val 值填充对象数组中的下标从 fromIndex 开始到 toIndex 为止的每个元素。

下面的代码创建一个整型数组，然后使用 fill()方法为其每个元素填充一个两位随机整数。

```
var intArray = new int[10];
for(var i = 0;i < intArray.length;i++){
    var num = (int)(Math.random() * 90) + 10;
    Arrays.fill(intArray, i, i + 1, num);          ← num 填到下标是 i 的元素
}
for(var i : intArray)
    System.out.print(i + " ");
}
```

下面是该段代码某次输出的结果：

```
58  73  92  34  56  32  13  67  30  98
```

提示 除上述讨论的方法外，Arrays 类中还提供了许多其他对数组操作的方法，请读者参考 Java API 文档。

扫一扫 视频讲解

5.5 案例学习——桥牌随机发牌

1. 问题描述

桥牌是一种文明、高雅、竞技性很强的智力游戏，由 4 个人分两组玩。桥牌使用普通扑克牌去掉大小王后的 52 张牌，分为梅花(C)、方块(D)、红心(H)和黑桃(S)四种花色，每种花色有 13 张牌，从大到小的顺序为：A(最大)、K、Q、J、10、9、8、7、6、5、4、3、2(最小)。

打桥牌首先需发牌。本案例就是通过编程实现随机发牌。最后，按顺序显示输出 4 个玩家得到的牌。

2. 设计思路

根据问题描述，本案例的设计思路如下：

(1) 可以使用一个有 52 个元素的数组存储 52 张牌。为了区分 52 张牌，使用不同的元素值即可，为了方便这里使用 0～51。设元素值从 0～12 为黑桃，13～25 为红桃，26～38 为方块，39～51 为梅花。在创建数组后为每个元素赋值，如下所示。

```
int [ ] deck = new int[52];
for(var i = 0; i < deck.length; i++)                    // 填充每个元素
    deck[i] = i;
```

(2) 为实现随机发牌，需要打乱数组元素的值，这里对每个元素随机生成一个整数下标，将当前元素与产生的下标的元素交换，循环结束后，数组中的元素值被打乱。

```
for(var i = 0; i < deck.length;i++){
    // 随机产生一个元素下标 0～51
    int index = (int)(Math.random() * deck.length);
    int temp = deck[i];                          // 将当前元素与产生的下标的元素交换
    deck[i] = deck[index];
    deck[index] = temp;
}
```

(3) 牌的顺序打乱后,4 个玩家依次从 52 张牌中取出 13 张牌。第 1 个 13 张牌属于第 1 个玩家,第 2 个 13 张牌属于第 2 个玩家,第 3 个 13 张牌属于第 3 个玩家,第 4 个 13 张牌属于第 4 个玩家。最后要求每个玩家的牌按顺序输出,因此需要对每个玩家的牌排序,这里使用 Arrays 类的 sort()方法,它可以对数组部分元素排序,比如对第 3 个玩家的牌排序,使用下面的代码。

```
Arrays.sort(deck,26,39);
```

(4) 最后根据每张牌的数值转换为牌的名称(如♥K)。为此,定义两个 String 数组,如下所示。

```
String[ ] suits = {"♠","♥","◆","♣"};
String[ ] ranks = {"A","K","Q","J","10","9","8","7",
                   "6","5","4","3","2"};
```

根据下面的代码确定每张牌的花色和次序,然后输出。

```
String suit = suits[deck[i]/13];                 // 确定花色
String rank = ranks[deck[i] % 13];               // 确定次序
System.out.printf("%s%-3s",suit,rank);
```

3. 代码实现

该案例的完整实现代码如程序 5.6 所示。

程序 5.6　BridgeCards.java

```
package com.boda.xy;
import java.util.Arrays;
public class BridgeCards {
    public static void main(String[ ]args){
        int[ ]deck = new int[52];
        String[ ] suits = {"♠","♥","◆","♣"};
        String[ ] ranks = {"A","K","Q","J","10","9","8","7",
                           "6","5","4","3","2"};
        // 初始化每一张牌
        for(var i = 0; i < deck.length;i++)
            deck[i] = i;
        // 打乱牌的次序
        for(var i = 0; i < deck.length;i++){
            // 随机产生一个元素下标 0～51
            int index = (int)(Math.random() * deck.length);
            int temp = deck[i];                  // 将当前元素与产生的下标元素交换
            deck[i] = deck[index];
            deck[index] = temp;
        }

        Arrays.sort(deck,0,13);
        Arrays.sort(deck,13,26);
        Arrays.sort(deck,26,39);
        Arrays.sort(deck,39,52);
        // 显示所有 52 张牌
```

对指定范围的数组元素排序

```
            for(var i = 0; i < 52; i++){
                switch(i % 13) {
                    case 0 -> System.out.print("玩家" + (i/13 + 1) + ":");
                }
                String suit = suits[deck[i]/13];      // 确定花色
                String rank = ranks[deck[i] % 13];    // 确定次序
                System.out.printf(" % s % -3s",suit,rank);
                if((i + 1) % 13 == 0) {
                    System.out.println();
                }
            }
        }
    }
```

4. 运行结果

案例运行结果如图 5-5 所示。

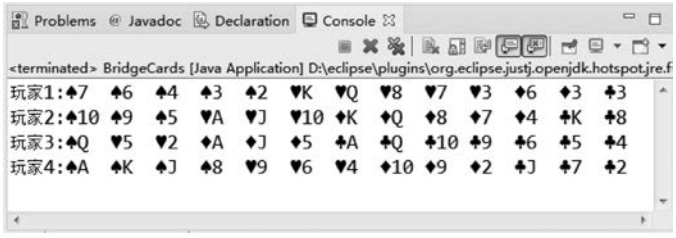

图 5-5 案例运行结果

从输出结果可以看到，每个玩家得到 13 张牌，并且每个玩家的牌按花色和大小进行了排序。

扫一扫 视频讲解

5.6 二维数组

Java 语言中的数组元素还可以是一个数组，这样的数组称为二维数组或数组的数组（array of array）。

5.6.1 二维数组的定义

二维数组的使用也分为声明和创建数组两个步骤。

1. 二维数组声明

二维数组有下面三种等价的声明格式。

```
elementType[ ][ ] arrayName;
elementType arrayName[ ][ ];
elementType[ ] arrayName[ ];
```

其中，elementType 为数组元素的类型，arrayName 为数组名。推荐使用第一种方法声明二维数组。下面的语句声明了一个整型二维数组 matrix 和一个 String 型二维数组 cities。

```
int [ ][ ] matrix;
String [ ][ ] cities;
```

2. 创建二维数组

创建二维数组就是为二维数组的每个元素分配存储空间。系统先为高维分配引用空间，然后再顺次为低维分配空间。二维数组的创建也使用 new 运算符，分配空间有两种方法，下面是直接为每一维分配空间。

```
int [ ][ ]matrix = new int[2][3];                    // 直接为每一维分配空间
```

这种方法适用于数组的低维具有相同个数的数组元素。在 Java 中，二维数组是数组的数组，即数组元素也是一个数组。上述语句执行后创建的数组如图 5-6 所示，二维数组 matrix 有两个元素，matrix[0]和 matrix[1]，它们又都是数组，各有三个元素。图 5-6 中共有三个对象：matrix、matrix[0]和 matrix[1]。

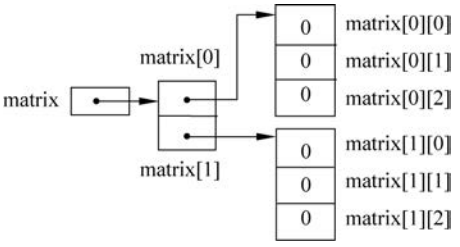

图 5-6 matrix 数组元素空间的分配

创建了二维数组后，它的每个元素被指定为默认值。上述语句执行后，数组 matrix 的 6 个元素值都被初始化为 0。

在创建二维数组时，也可以先为第一维分配空间，然后再为第二维分配空间。

```
int[ ][ ]matrix = new int[2][ ];                    // 先为第一维分配空间
matrix[0] = new int[3];                             // 再为第二维分配空间
matrix[1] = new int[3];
```

5.6.2 数组元素的使用

访问二维数组的元素，使用下面的形式：

```
arrayName[index1][index2]
```

其中，index1 和 index2 为数组元素下标，可以是整型常数或表达式。同样，每一维的下标也是从 0 到该维的长度减 1。

下面的代码给 matrix 数组的每个元素赋值：

```
matrix[0][0] = 80;
matrix[0][1] = 75;
matrix[0][2] = 78;
matrix[1][0] = 67;
matrix[1][1] = 87;
matrix[1][2] = 98;
```

下面的代码输出 matrix[1][2]元素值：

```
System.out.println(matrix[1][2]);
```

与访问一维数组一样，访问二维数组元素时，下标也不能超出范围，否则抛出异常。可以用 matrix.length 得到数组 matrix 的长度，结果为 2，用 matrix[0].length 得到 matrix[0]数组的长度，结果为 3。

对二维数组的第一维通常称为行，第二维称为列。要访问二维数组的所有元素，应该使用嵌套的 for 循环。例如，下面的代码输出 matrix 数组中所有元素。

```
for(var i = 0; i < matrix.length; i ++){
    for(var j = 0; j < matrix[0].length; j ++){
        System.out.print(matrix[i][j] + " ");
    }
    System.out.println();                           // 换行
}
```

同样，在访问二维数组元素的同时，可以对元素进行处理，比如计算行的和或者列的和等。

5.6.3 数组初始化器

对于二维数组也可以使用初始化器在声明数组的同时为数组元素初始化。例如：

```
int[ ][ ]matrix = {{15,56,20,-2},
                   {10,80,-9,31},
                   {76,-3,99,21},};
```

matrix 数组是 3 行 4 列的数组。多维数组每一维也都有一个 length 成员表示数组的长度。matrix.length 的值是 3，matrix[0].length 的值是 4。

5.6.4 实例——矩阵乘法

使用二维数组可以计算两个矩阵的乘积。如果矩阵 A 乘以矩阵 B 得到矩阵 C，则必须满足如下要求：

(1) 矩阵 A 的列数与矩阵 B 的行数相等。

(2) 矩阵 C 的行数等于矩阵 A 的行数，列数等于矩阵 B 的列数。

例如，下面的例子说明了两个矩阵是如何相乘的。

$$\begin{pmatrix} 1 & 2 & 1 \\ -2 & 4 & 1 \end{pmatrix} \times \begin{pmatrix} 4 & 3 & 0 & -1 \\ 2 & 3 & 5 & 2 \\ 1 & 0 & 6 & 3 \end{pmatrix} = \begin{pmatrix} 9 & 9 & 16 & 6 \\ 1 & 6 & 26 & 13 \end{pmatrix}$$

在结果矩阵中，第 1 行第 1 列的元素是 9，它通过下列计算得来：

```
1 × 4 + 2 × 2 + 1 × 1 = 9
```

即若矩阵 $A_{mn} \times B_{nl} = C_{ml}$，则

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

其中，A_{mn} 表示 m×n 矩阵，c_{ij} 是矩阵 C 的第 i 行 j 列元素。

程序 5.7　MatrixMultiply.java

```
package com.boda.xy;
public class MatrixMultiply {
    public static void main(String[ ]args){
        int a[ ][ ] = {{1,2,1},
                       {-2,4,1}};
        int b[ ][ ] = { {4,3,0,-1},
                        {2,3,5,2},
                        {1,0,6,3}};
        int c[ ][ ] = new int[3][4];
        // 用一个三重循环计算矩阵乘法
        for(var i = 0; i < 2; i++){
            for(var j = 0; j < 4; j++)
                for(var k = 0; k < 3; k++)
                    c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
        // 输出矩阵结果
        for(var i = 0; i < 2; i++){
            for(var j = 0; j < 4; j++)
              System.out.print(c[i][j] + " ");
            System.out.println();                    // 换行
        }
    }
}
```

程序运行结果为：

```
9  9  16  6
1  6  26  13
```

5.6.5 不规则二维数组

Java 的二维数组是数组的数组，对二维数组声明时可以只指定第一维的长度，第二维的每个元素可以指定不同的长度。例如：

```
String [ ][ ]cities = new String[2][ ];              // cities 数组有 2 个元素
cities[0] = new String[3];                           // cities[0]数组有 3 个元素
cities[1] = new String[2];                           // cities[1]数组有 2 个元素
```

这种方法适用于低维数组元素个数不同的情况，即每个数组的元素个数可以不同。对于引用类型的数组，除了为数组分配空间外，还要为每个数组元素的对象分配空间。

```
cities[0][0] = new String("北京");
cities[0][1] = new String("上海");
cities[0][2] = new String("广州");
cities[1][0] = new String("伦敦");
cities[1][1] = new String("纽约");
```

杨辉三角形，又称帕斯卡三角形，是二项式系数在三角形中的一种几何排列。程序 5.8 打印输出前 10 行杨辉三角形。

程序 5.8　Triangle.java

```
package com.boda.xy;
public class Triangle{
    public static void main(String[ ] args){
      int i, j;
      var level = 10;
      var triangle = new int[level][ ];
      for(i = 0;i < triangle.length;i++){
         triangle[i] = new int[i + 1];
      }
      // 为 triangle 数组的每个元素赋值
      triangle [0][0] = 1;
      for(i = 1;i < triangle.length;i++){
         triangle[i][0] = 1;
         for(j = 1; j < triangle[i].length - 1; j++){
            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
         }
         triangle[i][triangle[i].length - 1] = 1;
      }
      // 打印输出 triangle 数组的每个元素
      for(i = 0;i < triangle.length; i++){
        for(j = 0;j < triangle[i].length;j++)
            System.out.printf("%5d",triangle[i][j]);
         System.out.println();                       // 换行
      }
    }
}
```

程序运行结果如下：

```
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  70  56  28   8  1
1  9  36  84 126 126  84  36  9  1
```

提示　Java 也支持多维数组。例如，double [][][] sales = new double[3][3][4]；语句声明并创建一个三维数组，共有 36 个元素。

扫一扫 视频讲解

5.7　案例学习——打印输出魔方数

1. 问题描述

在中国传统文化中，有一种图表叫九宫图，它是在 9 个格子中放置数字 1～9，使得每行、每列和每条对角线上的 3 个数字之和相等，如图 5-7 所示。对于该图，古人还给出了图中数字的记忆方法，“戴九履一，左三右七，二四为肩，六八为足，五居中央”。

8	1	6
3	5	7
4	9	2

图 5-7　九宫格

数学上已经证明，对于任何 n×n 个自然数(1，2，…，n^2，n 为奇数)都有这样的排列，使得每行、每列和每条对角线上数的和都相等。这些数称为魔方数。编写程序，要求用户从键盘输入要打印的魔方数(比如 5)，程序计算并输出魔方数。

2. 设计思路

对于 n 为奇数的魔方数，可以按下列方法将 1～n^2 的数填充到二维数组的元素中。首先把 1 放在第 0 行的中间，剩下的数 2，3，…，n^2 依次向上移动一行，并向右移动一列。当可能越过数组边界时需要“绕回”到数组的另一端。例如，如果需要把下一个数放到－1 行，就将其存储到 n－1 行(最后一行)；如果需要把下一个数放到第 n 列，就将其存储到第 0 列。如果某个特定的数组元素已被占用，就把该数存储在前一个数的正下方。

根据上述规则，可按下列步骤设计实现本案例。

(1) 首先从键盘输入一个正奇数 size，然后定义一个 size×size 的二维数组，代码如下：

```
int [ ][ ] magic = new int[size][size];
```

(2) 将 1 填充到数组的第一行中间元素中，代码如下：

```
int row = 0, col = size/2;
magic[row][col] = 1;                                    // 将 1 填到第一行中间元素
```

(3) 将 2 到 size×size 填充到数组其他元素中，这里重点是确定下一个元素填充的位置。通过定义临时变量存放元素的行和列，如果下一个位置超出数组元素范围，将改变其行或列的值，最终确定正确位置。

(4) 输出填充了数值的二维数组的每个元素。

3. 代码实现

该案例的完整实现代码如程序 5.9 所示。

程序 5.9　MagicSquare.java

```
package com.boda.xy;
import java.util.Scanner;
public class MagicSquare {
    public static void main(String[ ] args) {
        var input = new Scanner(System.in);
        int size;
        System.out.print("请输入魔方矩阵的大小(1－99):");
        size = input.nextInt();
        if(size < 0 && size % 2 == 0){
            System.out.println("你输入的数不是奇数,不能计算魔方数!");
            System.exit(0);
        }
```

```
        var magic = new int[size][size];
        // 用 1 到 size×size 值填充每个元素
        int row = 0, col = size/2;
        magic[row][col] = 1;                    // 将 1 填充到第一行中间元素
        for(var n = 2; n <= size * size; n++){
            int tempx,tempy;
            tempx = row;
            tempy = col;
            tempx = tempx - 1;                  ← 找下一个元素位置，行减 1、列加 1
            tempy = tempy + 1;
            if(tempx < 0)                       ← 行超出范围，变到最后一行
              tempx = size - 1;
            if(tempy == size)                   ← 列超出范围，变到第一列
              tempy = 0;

            if(magic[tempx][tempy]!= 0){
              magic[row + 1][col] = n;          ← 当前位置被占，数存放到当前位置下方
              row++;
            }else if(magic[tempx][tempy] == 0) {
              magic[tempx][tempy] = n;          ← 数存放到当前元素
              row = tempx;
              col = tempy;
            }
          }
          // 输出数组元素
          for(var i = 0; i < magic.length; i ++){
            for(var j = 0;j < magic.length; j ++){
              System.out.printf(" % 4d",magic[i][j]);
            }
            System.out.println();
        }
    }
}
```

4. 运行结果

运行程序，当输入 5 时的运行结果如图 5-8 所示。

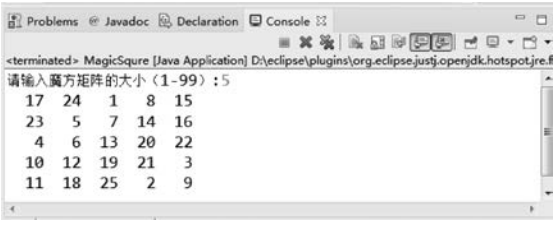

图 5-8 案例输出结果

关于魔方矩阵，本案例只计算输出奇数的矩阵。实际上，对于偶数的矩阵，也有一些满足这个特性，但有些不满足。读者可自己到网上查找资料，了解它的特性。

5.8 小结

本章详细介绍了 Java 语言的数组及其应用，主要知识点包括数组的创建和使用，数组元素的访问、复制等，数组的查找和排序，java. util. Arrays 类的使用，二维数组的创建和使用等。

下一章将介绍 Java 语言的字符串类，内容包括 String 类的基本操作，文本块，命令行参

数，格式化输出以及 StringBuilder 类的操作。

编程练习

1. 给定数组 a 的声明如下，编写程序，计算所有元素的和、最大值、最小值及平均值。

```
double [ ]a = {75, 53, 32, 12, 46, 199, 17, 54};
```

2. 编写一个方法，求出一个 double 型数组中的最小元素，方法的声明格式如下：

```
public static double min(double[ ] array)
```

编写测试程序，提示用户从键盘输入 5 个 double 型数，并存放到一个数组中，然后调用这个方法返回最小值。

3. 编程打印输出 Fibonacci 数列的前 20 个数。Fibonacci 数列是第一个和第二个数都是 1，以后每个数是前两个数之和，用公式表示为：$f_1=f_2=1$，$f_n=f_{n-1}+f_{n-2}(n\geqslant 3)$。要求使用数组存储 Fibonacci 数。

4. 编写程序，定义一个有 10 个元素的整型数组，然后将其前 5 个元素与后 5 个元素对换，即：第 1 个元素与第 10 个元素互换，第 2 个元素与第 9 个元素互换，……，第 5 个元素与第 6 个元素互换。分别输出数组原来各元素的值和互换后各元素的值。

5. 编写程序，定义一个有 8 个元素的整型数组，然后使用选择排序法对该数组按升序排序。选择排序法先找到数列中最小的数，然后将它和第一个元素交换。接下来，在剩下的数中找到最小数，将它和第二个元素交换，以此类推，直到数列中仅剩一个数为止。

6. 编写一个方法，计算给定的两个数组之和，格式如下：

```
public static int[ ] sumArray(int[ ] a, int[ ] b)
```

要求返回的数组元素是两个参数数组对应元素之和，不对应的元素直接赋给相应的位置，如{1,2,4}+{2,4,6,8}={3,6,10,8}。

7. 编写一个方法，合并给定的两个数组，并以升序返回合并后的数组，格式如下：

```
public static int[ ] arrayMerge(int[ ] a, int[ ] b)
```

例如，一个数组是{16,13,15,18}，另一个数组是{29,36,100,9}，返回的数组应该是{9,13,15,16,18,29,36,100}。

8. 编写程序，使用下面的方法头编写一个解一元二次方程式的方法：

```
public static int solveQuadratic(double [ ] eqn, double[ ] roots)
```

一元二次方程式 $ax^2+bx+c=0$ 的系数都传给数组 eqn，然后将两个非复数的根存在 roots 中。方法返回根的个数。

9. 编写程序，使用筛选法求出 2～100 中的所有素数。筛选法是在 2～100 的数中先去掉 2 的倍数，再去掉 3 的倍数，以此类推，最后剩下的数就是素数。注意：2 是最小的素数，不能去掉。

10. 如果两个数组 list1 和 list2 的长度相同，而且对于每个 i，list1[i]都等于 list2[i]，那么认为 list1 和 list2 是完全相同的。使用下面的方法头编写一个方法，如果 list1 和 list2 完全相同，那么这个方法返回 true。

```
public static boolean equals(int [ ] list1, int [ ] list2)
```

11. 编程求解约瑟夫(Josephus)问题：有 12 个人排成一圈，从 1 号开始报数，凡是数到 5

的人就离开，然后继续报数。试问：最后剩下的一人是谁？

12. 编写程序，从一副 52 张的牌中选出 4 张，然后计算它们的和，程序应该显示得到和为 24 的选牌次数。A、J、Q 和 K 分别表示 1、11、12 和 13。

13. 有下面两个矩阵 A 和 B：

$$A=\begin{pmatrix} 1 & 3 & 5 \\ -3 & 6 & 0 \\ 13 & -5 & 7 \\ -2 & 19 & 25 \end{pmatrix} \qquad B=\begin{pmatrix} 0 & -1 & -2 \\ 7 & -1 & 6 \\ -6 & 13 & 2 \\ 12 & -8 & -13 \end{pmatrix}$$

编写程序，计算：(1)A+B；(2)A−B；(3)矩阵 A 的转置。

14. 编写下面的方法，返回二维数组中最大元素的位置。

```
public static int[ ] locateLargest(double [ ][ ] a)
```

返回值是包含两个元素的一维数组。这两个元素表示二维数组中最大元素的行下标和列下标。编写一个测试程序，提示用户输入一个二维数组，然后显示这个数组中最大元素的位置。

```
请输入数组的行数和列数：3 4
请输入每行元素的值：
23.5  35    2   10
 4.5   3   45  3.5
  35  44  5.5  9.6
最大元素的位置是(1,2).
```

15. 编写程序，从键盘输入矩阵阶数 n，给 n 阶矩阵的元素按行序从 1～n×n 的顺序赋值，然后将矩阵按顺时针旋转 90°，输出旋转后的矩阵。运行结果如下：

```
请输入矩阵的阶数:4
13   9  5  1
14  10  6  2
15  11  7  3
16  12  8  4
```