

Chp5 数组（增加冒泡排序）

数组是一个语言中的基本要素，它能够用来保存和管理多个变量。例如，如果要统计三个学生的成绩，可以手动的定义三个变量 `a`、`b`、`c`，如果要输出这三个变量的值，也可以写三个输出语句。但是，如果要管理一个年级所有学生的成绩，此时，可能有上百个学生。如果为每个学生都手动定义一个变量的话，则程序中会有上百个变量。并且，如果要输出所有学生的成绩，就会有上百个输出语句。很显然，这样的代码是非常复杂和繁琐的。

因此，我们就需要有一个办法，能够比较方便的管理多个数据。在这种情况下，我们就应该用到数组。

1 数组的基本操作

数组是用来管理多个同类型的变量的。一个数组，是一段连续的内存空间，这段内存空间中，能够保存多个同类型的值。例如，一个长度为 3 的 `int` 数组，是说这个数组能够保存 3 个 `int` 类型的值。

下面，我们就来介绍一下数组的一些基本操作。首先是如何创建数组。

1.1 创建数组

如果要定义一个 `int` 类型的数组变量 `a`，可以写成如下的形式：

```
int[] a 或者 int a[]
```

上面的两种形式都可以用来定义 `int` 类型的数组变量 `a`。需要注意的是，定义完了数组变量之后，并没有分配连续的内存空间。怎么来理解这个问题呢？

数组是用来管理多个变量的，在这里，“多个”的概念并不明确。例如，在生活中，一本书、一支笔，这些都是比较精确的“一个”物品。而“一堆书”、“一捆笔”，这种描述多个物品的情况，则具体的数量就显得很不明确。数组变量也是如此，一个数组变量能够管理多个数据，但是，究竟这个数组的空间多大，还需要另外明确指定。

为了分配空间，也为了明确指定数组空间的大小，必须要使用关键字 `new`。用法如下：

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

上面的语句才真正分配了一个长度为 3 个 `int` 的内存空间。也可以在定义数组变量的时候立刻进行内存的分配：

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

1.2 下标， `ArrayIndexOutOfBoundsException`

为数组分配完空间之后，就可以使用数组了。使用数组的时候，应当用下标来表示数组元素。例如，上面分配了长度为 3 个 `int` 的内存空间，这 3 个 `int` 分别可以用：`a[0]`、`a[1]`、`a[2]`来表示。

需要注意的是，数组的下标从 0 开始，也就是说，如果分配了一个长度为 `n` 的数组，则数组的下标范围为 `0~n-1`。

有了下标之后，就可以操作数组元素。此时，数组的每个元素和普通的 `int` 变量没有区别。例如：

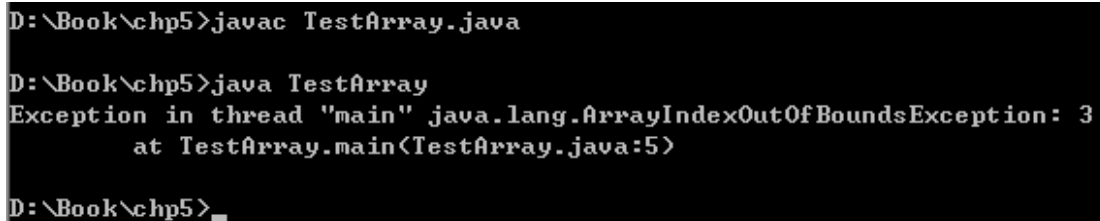
```
a[0] = 10; //对数组元素进行赋值
```

```
a[1] = 20;
a[2] = 30;
System.out.println(a[2]); //输出数组元素
```

而如果对数组进行下标操作时，超出了数组下标正常的范围，则 Java 会产生一个异常：**ArrayIndexOutOfBoundsException**。例如下面的代码：

```
public class TestArray {
    public static void main(String[] args) {
        int[] a = new int[3];
        a[0] = 10;
        a[3] = 20;
    }
}
```

上面的代码在编译时不会出错，因为从语法上说没有任何问题。但是，由于 a 数组长度为 3，因此起下标范围是 0~2，使用 a[3] 变量会在运行时出错。运行时得到的错误信息如下：



```
D:\Book\chp5>javac TestArray.java
D:\Book\chp5>java TestArray
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
    at TestArray.main<TestArray.java:5>
D:\Book\chp5>_
```

我们分析一下这个错误信息。首先，这个错误的类型是：**java.lang.ArrayIndexOutOfBoundsException**。这种类型的错误，产生的原因一定是数组下标越界。在第一行的末尾，还有一个数字 3，这表明数组下标越界时，越界的数字为 3。在第二行，有一个数字 5，这表明数组下标越界的代码是在源文件（**TestArray.java**）的第 5 行。

1.3 遍历数组

所谓遍历数组，指的是把数组的所有元素都进行某一个操作，这是数组中最常用的操作之一。最典型的遍历数组，就是把数组中的所有元素都打印一遍。例如，我们写一个打印数组 a 的程序：

```
public class TestArray {
    public static void main(String[] args) {
        int[] a = new int[3];
        a[0] = 10;
        a[1] = 20;
        a[2] = 30;
        for(int i = 0; i<3; i++){
            System.out.print(a[i] + "\t");
        }
        System.out.println();
    }
}
```

上面的程序就能完成打印 a 数组的功能，利用一个 for 循环，就能够完成数组的遍历。需要强调的是，for 循环中，循环条件是 $i < 3$ ，这表明 i 变量的变化是 0~2，正好在数组的

下标范围之内。可以看到，3正好是数组的长度。

由于打印数组这个功能非常常用，我们可以再提炼一下，把打印数组的代码写到一个函数中，这个函数叫做 `printArray`，接受一个 `int` 数组作为参数，签名可以写成如下形式：

```
public static void printArray(int[] n)
```

注意，这个函数接受数组类型作为函数形参，此处的写法与定义一个数组变量相同。

下面我们完成这个函数。从上面的代码中，我们可以看到，写遍历数组的循环，往往是这样的形式：

```
for(int i = 0; i<XXX; i++)
```

其中，`i<XXX` 这个循环条件，一定是 `i` 小于数组的长度。但是，由于 `printArray` 函数可以接受任何一个 `int` 数组作为参数，因此打印数组时，数组的长度是不固定的，必须根据传入的数组参数的不同，动态的获得数组的长度。那数组的长度我们应当怎么获得呢？

在 `Java` 中，我们可以使用“数组名.length”的方式获得数组的长度。举例来说，如果想要获得 `n` 数组的长度，则可以使用 `n.length` 这个变量来获得长度。

例如下面的代码：

```
int[] a = new int[3];  
System.out.println(a.length);
```

上面的代码，会输出 `a` 数组的长度：3。

需要注意的是，`n.length` 这种写法只能读取数组的长度，而不能修改。也就是说，不能为 `n.length` 这种变量进行赋值，否则会产生一个编译时错误。

例如，下面的代码就会产生错误：

```
int[] a = new int[3];  
System.out.println(a.length);  
a.length = 5; //想要改变数组的长度，出错！
```

有了 `length` 这个变量，我们就能动态的获得数组的长度，因此 `printArray` 方法就可以完成了。代码如下：

```
public static void printArray(int[] a){  
    for(int i = 0; i<a.length; i++){  
        System.out.print(a[i] + "\t");  
    }  
    System.out.println();  
}
```

1.4 数组的初始化

下面我们可以调用一下 `printArray()` 方法。定义一个数组 `a`：

```
int[] a = new int[3];  
printArray(a);
```

注意，在分配完三个 `int` 变量之后，并没有为这三个变量赋值。也就是说，此时，这三个变量并没有被赋初始值，而直接被 `printArray` 方法打印。

运行结果如下：

```
D:\Book\chp5>javac TestArray.java

D:\Book\chp5>java TestArray
0      0      0

D:\Book\chp5>
```

编译顺利通过，打印的结果是：三个 0。因此，我们可以得出结论：数组元素和局部变量不同，数组元素可以在没有赋初始值的情况下就使用。此时，这些数组元素也有特定的值，这就是元素的“默认值”。在为数组分配空间的时候，数组的元素会被 JVM 赋一个默认值。

那么元素的默认值都是什么呢？这需要根据不同类型来看。对于基本类型来说，默认值为各种各样的 0。怎么来理解呢？byte、short、int、long 这四种整数类型，默认值为 0；float 和 double 这两种小数类型，默认值为 0.0，boolean 默认值为 false，char 默认值也为 0。注意，char 类型的 0 不是指的字符 ‘0’，而是指的编码为 0。

对于对象类型来说，默认值为 null 值。

有了默认值这个概念，考虑下面的代码：

```
int[] a = new int[3]; a[0] = 10; a[1] = 20; a[2] = 30;
printArray(a);
```

在这段代码中，每一个数组元素都被赋值了两次：一次是默认值，另一次是初始值 10、20、30。那有没有什么办法能够避免那一次不必要的默认值呢？

我们可以使用一个叫做“数组的显式初始化”的语法。这种语法有两种形式。第一种形式如下：

```
int[] a = {10, 20, 30};
```

这种语法的特点是，只能在定义数组变量的同时使用。如果代码如下：

```
int[] a;
//! a = {10, 20, 30};
```

上面的代码在数组定义之后没有直接进行显式初始化，而是换了一个语句又一次赋值。这样会产生一个编译时错误。而且，这个错误的出错信息相当吓人：

```

D:\Book\chp5>javac TestArray.java
TestArray.java:4: illegal start of expression
        a = {1,2,3};
            ^
TestArray.java:4: not a statement
        a = {1,2,3};
            ^
TestArray.java:4: ';' expected
        a = {1,2,3};
            ^
TestArray.java:5: invalid method declaration; return type required
        printArray(a);
        ^
TestArray.java:5: <identifier> expected
        printArray(a);
                ^
TestArray.java:7: class, interface, or enum expected
        public static void printArray(int[] a){
                ^
TestArray.java:8: class, interface, or enum expected
        for(int i = 0; i<a.length; i++){
                ^
TestArray.java:8: class, interface, or enum expected
        for(int i = 0; i<a.length; i++){
                ^
TestArray.java:10: class, interface, or enum expected
        }
        ^
TestArray.java:12: class, interface, or enum expected
    }
    ^
10 errors

```

这些错误，其实都只是因为数组显示初始化的语法不对，才产生的。显示初始化的语法一旦产生错误，就会引起一系列的连锁反应。

第二种语法形式如下：

```
int[] a = new int[]{10, 20, 30};
```

注意，这种语法下，`new` 关键字后面的方括号中没有数字，也就是说，显式初始化时不能规定数组长度，数组长度由后面的元素个数决定。

要注意的是，这种语法形式能够进行赋值。也就是说，下面的代码可以编译通过：

```
int[] a;
a = new int[]{10, 20, 30};
```

2 数组在内存中的表示

下面我们分析一下，Java 数组在内存中的表示情况。看下面两行代码

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

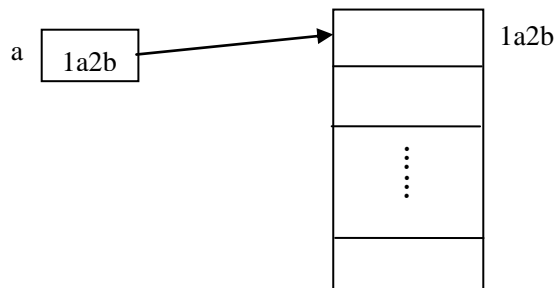
我们结合代码，分析一下数组在内存中的表示。

第一行，定义了一个数组变量 `a`，此时没有分配连续的内存空间。

第二行，首先执行了 `new int[3]`，这句代码分配了一个段连续的内存空间，总共能够放

入 3 个 `int`，因此是 12 个字节。这 12 个字节每个字节都有自己的一个内存地址，其中，12 个字节中的第一个字节，它的地址被称之为这块内存的“首地址”。假设首地址为 `1a2b`。

第三步，执行完了 `new int[3]` 之后，进行了赋值。究竟是把什么赋值给了变量 `a` 呢？注意，赋值赋的是内存的首地址。也就是说，数组变量保存的是数组中的首地址 `1a2b`。如下图所示



3 二维数组和多维数组

3.1 二维数组的基本操作

上面我们介绍的都是一维数组，接下来要介绍的是 `Java` 中的二维数组以及多维数组。

在 `Java` 中，所谓的多维数组指的是：数组的数组。怎么来理解这一点呢？比如说，我们日常生活中的抽屉，我们可以认为抽屉就是用来存放多个物品的，因此抽屉就是一个物品的数组。而一个柜子里，可以存放多个抽屉，因此我们可以理解为，柜子就是抽屉组成的数组。因此，柜子就可以理解为是“数组的数组”，也就是：柜子的元素是抽屉，而抽屉本身又是一个数组。

那在代码方面怎么表示呢？我们首先来看二维数组的创建和遍历。

二维数组在创建时，也需要一个数组变量。定义数组变量，可以使用下面的代码：

```
int[][] a; 或者 int[] a[]; 或者 int a[][];
```

可以看出，定义数组变量在 `Java` 中格式是非常自由的。需要注意的是，在定义二维数组变量的时候，同样没有分配数组空间。

如果要为二维数组分配内存空间，则可以使用下面的代码：

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

这表示，分配一个三行四列的二维数组。怎么来理解“行”和“列”呢？我们可以这么来看：我们分配的这个二维数组就相当于一个柜子，这个柜子有三层，每层放一个抽屉。这个抽屉里面分成了四个格子，每个格子又能放一个元素。由于二维数组是“数组的数组”，因此，二维数组的“行”，指的是这个二维数组中，包含几个元素。由于二维数组的元素是一维数组，因此，“行”也就是二维数组中包含几个一维数组。而列，则指的是，二维数组中的每一个一维数组，各自都包含几个元素。

如果要获得第 0 行第 2 列的元素，则可以使用双下标：`a[0][2]` 来获得。

下面我们来介绍遍历二维数组。遍历二维数组时，要获得行和列两个数值。首先，二维数组同样有 `a.length` 这样的变量，而使用 `a.length` 获得的长度，是二维数组元素的个数，也就是是行的数目，也可以理解成：柜子里抽屉的个数。那如果要获得列呢？列，就相当于于是每一个一维数组的长度，也可以理解为，是每一个抽屉的大小。对于第 `i` 个抽屉，我

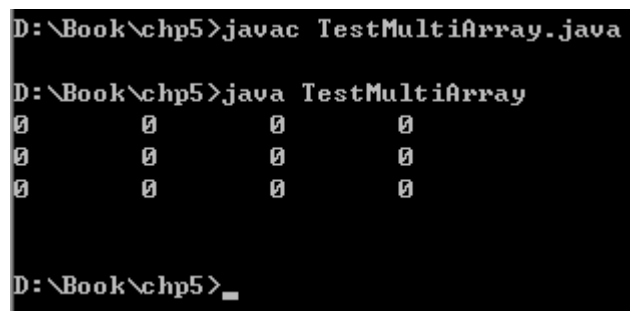
们可以使用 `a[i].length` 来获得它的长度。遍历二维数组的代码如下：

```
public static void printMultiArray(int[][] a){
    for(int i = 0; i<a.length; i++){
        for(int j = 0; j<a[i].length; j++){
            System.out.print(a[i][j] + "\t");
        }
        System.out.println();
    }
    System.out.println();
}
```

完整的二维数组的代码如下：

```
public class TestMultiArray{
    public static void main(String args[]){
        int[][] a = new int[3][4];
        printMultiArray(a);
    }
    public static void printMultiArray(int[][] a){
        for(int i = 0; i<a.length; i++){
            for(int j = 0; j<a[i].length; j++){
                System.out.print(a[i][j] + "\t");
            }
            System.out.println();
        }
        System.out.println();
    }
}
```

运行结果如下：



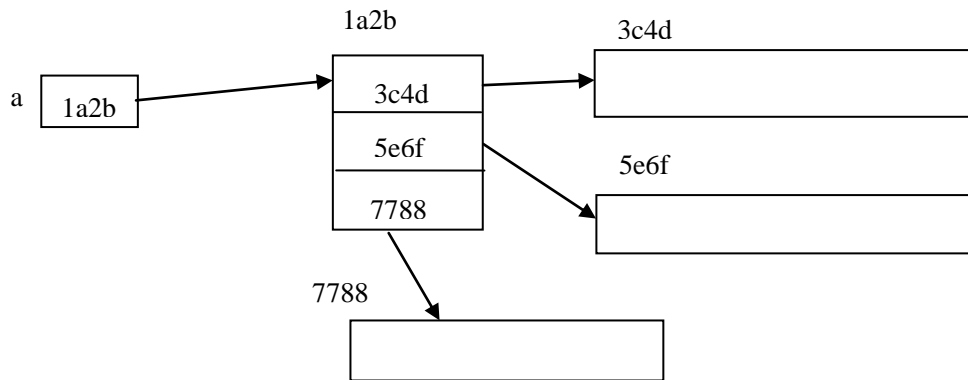
```
D:\Book\chp5>javac TestMultiArray.java

D:\Book\chp5>java TestMultiArray
0      0      0      0
0      0      0      0
0      0      0      0

D:\Book\chp5>_
```

3.2 二维数组的内存表示

二维数组在内存中的表示如下：



可以看出，在一维数组的情况下，第一个维度是一个长度为 3 的数组，这个数组保存的还是某一块内存的地址。我们可以把这个数组当做是 a 的 3 个元素，因此，非常明显，a 的这三个元素，是三个一维数组。

从这个例子我们可以推而广之到任意维度的数组。一个二维数组，就是数组的数组，一个 3 维数组就是数组的数组的数组，一个 4 维数组就是数组的数组的数组的数组，一个……

但是，对于平时的编程实践而言，使用比较多的还是一维数组和二维数组，更高维度的数组使用起来比较复杂，相对而言用的较少。

3.3 不规则数组

除了普通的二维数组之外，Java 还支持不规则数组。举例来说，如果一个柜子有三个抽屉，这个三个抽屉中并不一定每个抽屉都具有一样的大小，完全有可能第三个抽屉更大，元素更多，而第一个抽屉相对就比较小。在 Java 中，可以用下面的代码创建不规则数组：

```

int[][] a; //定义数组变量
a = new int[3][]; //先确定第一个维度，表明柜子里有三个抽屉
a[0] = new int[3]; //顶上的抽屉有三个元素
a[1] = new int[4]; //下一层有四个元素
a[2] = new int[5]; //最底层有五个元素
  
```

完整代码如下：

```

public class TestMultiArray{
    public static void main(String args[]){
        int[][] a; //定义数组变量
        a = new int[3][]; //先确定第一个维度，表明柜子里有三个抽屉
        a[0] = new int[3]; //顶上的抽屉有三个元素
        a[1] = new int[4]; //下一层有四个元素
        a[2] = new int[5]; //最底层有五个元素

        printMultiArray(a);
    }
    public static void printMultiArray(int[][] a){
        for(int i = 0; i<a.length; i++){
  
```

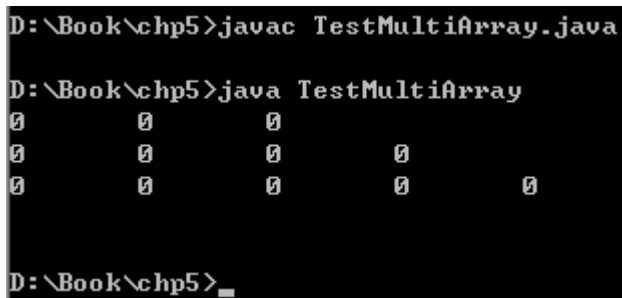


```

        for(int j = 0; j<a[i].length; j++){
            System.out.print(a[i][j] + "\t");
        }
        System.out.println();
    }
    System.out.println();
}
}

```

运行结果如下：



```

D:\Book\chp5>javac TestMultiArray.java

D:\Book\chp5>java TestMultiArray
0      0      0
0      0      0      0
0      0      0      0      0

D:\Book\chp5>_

```

3.4 数组的显式初始化

与一维数组相似，二维数组也有显式初始化的语法。

在一维数组中，一对花括号，就表示一个数组。而二维数组，是数组的数组，所以，二维数组显式初始化的基本语法如下：

```

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

```

可以看到，这个二维数组 a，本身有三个元素。这三个元素都是数组，因此这三个元素都要加上花括号。元素和元素之间用逗号隔开，因此，三个花括号之间，用逗号隔开。

每一个花括号表示一个数组，在这三个元素中，分别表示三个数组：

第一个数组：{1, 2, 3}

第二个数组：{4, 5}

第三个数组：{6, 7, 8}

与一维数组类似，二维数组也有两种显式初始化的语法。除了我们上面介绍的初始化语法之外，二维数组还能用下面的方式显式初始化：

```

int[][] a;
a = new int[][]{
    {1,2,3},
    {4,5,6}
};

```

与一维数组一样，第一种显式初始化的语法只能用在初始化上；而第二种显式初始化的语法能够用来赋值。

4 数组的常见算法

接下来这一部分，我们将为大家介绍两个数组常见的算法。

4.1 数组的扩容

我们之前介绍过，数组有一个 `length` 变量，这个变量只能够读，不能够修改。那么，如果我们需要让数组的长度增大怎么办？例如，原有数组的长度为 3，而我们希望能够得到长度更大的数组，应该怎么做呢？

首先，数组空间一旦分配完成之后，长度就不能改变了。因此，我们不能够直接在原有数组的空间后面增加新的内存空间。我们采用一个曲线救国的方式，来增加数组的长度：

- 1、分配一个新的数组，新数组的长度比原有数组要大（比如长度是原有数组的两倍）
- 2、把原有数组中的数据，拷贝到新数组中。

我们可以把数组的扩容，理解成：人想要住大房子。如果一个家庭，觉得目前的房子太小，面积不够了，应当怎么做呢？显然，不可能在原有的房子上增加新的面积，不可能在原来的墙外面再造一个新房间（这属于违章搭建）。如果想要住大房子的话，只能够重新找一间屋子，这件屋子比原来的面积要大。这就相当于分配了一块新的内存空间。

但是，分配完空间之后，一家子人不会马上就住进新房，还需要把老房子当中的东西，都搬到新房中。这样，原有的家具、电器等，都不会丢失。这就相当于在分配完空间之后，要把数组中原有的数据，拷贝到新数组中。

为此，我们写一个函数，用来完成数组扩容。这个函数接受一个 `int[]` 数组类型作为参数，把这个数组扩容一倍，然后再把扩容以后的数组返回。函数定义如下：

```
public static int[] expand(int[] a)
```

注意，如果返回一个 `int` 数组类型的话，返回值的写法就是 `int[]`。

这个方法的实现如下：

```
public static int[] expand(int[] a){
    int[] newArray = new int[a.length * 2];
    for(int i = 0; i < a.length; i++){
        newArray[i] = a[i];
    }
    return newArray;
}
```

完整代码如下：

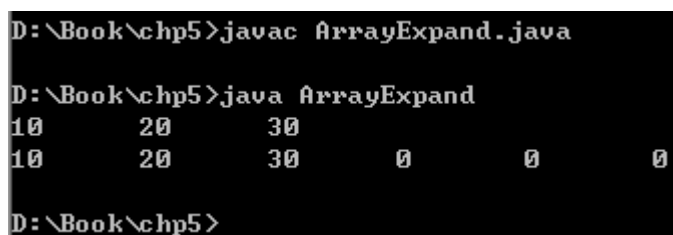
```
public class ArrayExpand{
    public static void main(String args[]){
        int[] a = {10, 20, 30};
        printArray(a);
        a = expand(a);
        printArray(a);
    }
}
```

```

public static int[] expand(int[] a){
    int[] newArray = new int[a.length * 2];
    for(int i = 0; i<a.length; i++){
        newArray[i] = a[i];
    }
    return newArray;
}
public static void printArray(int[] a){
    for(int i = 0; i<a.length; i++){
        System.out.print(a[i] + "\t");
    }
    System.out.println();
}
}

```

运行结果如下：



```

D:\Book\chp5>javac ArrayExpand.java
D:\Book\chp5>java ArrayExpand
10      20      30
10      20      30      0      0      0
D:\Book\chp5>

```

4.2 冒泡排序

排序，是数组操作中一个非常重要的部分。排序要完成的工作，就是把数组中的所有元素，进行位置上的调整，最终使得所有元素的排列都符合一定的顺序。例如，有如下数组：

```
int[] a = {5,7,1,3,2};
```

经过排序之后，希望 a 数组的值变为：

```
{1, 2, 3, 5, 7}
```

计算机科学领域中，排序的算法有很多种。在本章，我们为大家介绍最常用，也是最基本的一种排序方法：冒泡排序。

首先，冒泡排序的核心是这样的一个过程：

对数组相邻的元素两两进行比较，如果左边的元素值大于右边的元素值，则交换两个元素的位置。

我们演示一下这个过程，以上面的 a 数组为例：一开始，数组值为：

5	7	1	3	2
---	---	---	---	---

之后，把 a[0]和 a[1]进行比较。由于 5 比 7 小，因此，两个位置不交换。

然后，是 a[1]和 a[2]进行比较。由于 7 比 1 大，因此，交换两个元素的位置。得到下面的结果：

5	1	7	3	2
---	---	---	---	---

之后，进行 a[2]和 a[3]的比较。由于 7 比 3 大，因此，交换两个元素的位置，得到下

面的结果：

5	1	3	7	2
---	---	---	---	---

最后，进行 $a[3]$ 和 $a[4]$ 的比较。由于 7 比 2 大，因此，交换两个元素的位置，结果如下：

5	1	3	2	7
---	---	---	---	---

这样的一轮操作，把最大的数字 7 排到了整个数组的末尾。因此，这一轮操作就把最大的元素排好了。因此，第二轮操作的时候，就应该不用再考虑最后一个元素了。

下面我们开始第二轮操作。第二轮操作的过程不再用文字详细描述，过程如下：

5	1	3	2	7
1	5	3	2	7
1	3	5	2	7
1	3	2	5	7

可以看到，进过第二轮操作，把倒数第二个位置也排好了。第三轮，则只需要看前三个元素。第三轮执行的流程如下：

1	3	2	5	7
1	2	3	5	7

之后，第四轮还是会比较一下前两个元素。最后，排序完成。

我们可以看到，进行了四轮操作，第一轮排好了最大的元素，第二轮排好了第二大的元素，第三轮排好了第三大的元素……以此类推。在排序的过程中，相邻元素不停进行比较和交换。在交换的过程中，大的元素沉向数组的末尾，小的元素走向数组的开头；这就好像在水里面：重的东西往下沉，而轻的东西往上浮起来。正因为这中排序方式很像水里的气泡往上浮的过程，因此，这种排序方式被称为冒泡排序。

接下来，我们来写冒泡排序的代码。如果有五个元素，则需要进行 4 次循环，也就是说，如果数组的长度是 $a.length$ 的话，则需要进行 $a.length-1$ 次循环。因此，外层循环如下：

```
for(int i = 0; i<a.length-1; i++){  
    ...  
}
```

内层循环稍有点复杂。我们让内层循环的循环变量为 j ，则每次进行比较的时候，比较的都是 $a[j]$ 和 $a[j+1]$ 这两个元素。那么 j 的循环条件怎么写呢？

第 1 次循环， i 的值为 0，因为要排到最后一个，因此 $j+1$ 最大值为 $a.length$ ， j 的最大值为 $a.length-1$ ；

第 2 次循环， i 的值为 1， $j+1$ 的最大值为 $a.length-1$ ， j 的最大值为 $a.length-2$ ；

第 3 次循环， i 的值为 2， $j+1$ 的最大值为 $a.length-2$ ， j 的最大值为 $a.length-3$ ；

.....

由上面，我们可知，每次循环 $j+1$ 的最大值，都是 $a.length-i$ ；而 j 的最大值，就是 $a.length-i-1$ 。

因此，内层循环条件如下：

```
for(int i = 0; i<a.length-1; i++){  
    for(int j = 0; j<a.length-i-1; j++){  
        比较  $a[j]$  和  $a[j+1]$ ，  
        如果  $a[j]$  比  $a[j+1]$  大，则交换两个元素的值
```

```

    }
}

```

进一步细化，代码为

```

for(int i = 0; i<a.length-1; i++){
    for(int j = 0; j<a.length-i-1; j++){
        if(a[j] > a[j+1]){
            则交换 a[j] 和 a[j+1] 的值
        }
    }
}

```

如何交换两个变量的值呢？假设有两个变量 `a = 5`；`b=4`；要交换两个 `a` 和 `b` 的值，应该怎么做呢？

如果直接执行 `a = b` 的话，则 `a` 的值 5 就会被 `b` 的值覆盖。这样，`a` 有了 `b` 的值，但是 `b` 却无法获得 `a` 变量原来的值了。因此，为了交换两个变量的值，需要第三个变量参与。

首先，定义一个新变量 `t`；

然后，把 `a` 的值赋值给 `t`：`t = a`；

接下来，把 `b` 的值赋值给 `a`：`a=b`。这样会覆盖 `a` 原有的值，但是 `a` 原有的值已经被保存在 `t` 变量中了。

再接下来，把在 `t` 变量中保存的原有的 `a` 变量的值，赋值给 `b`。

完整的代码如下：

```

int a = 5, b=4;
int t;
t = a;
a = b;
b = t;

```

因此，冒泡排序的交换部分，也可以完成了：

```

for(int i = 0; i<a.length-1; i++){
    for(int j = 0; j<a.length-i-1; j++){
        if(a[j] > a[j+1]){
            //交换 a[j] 和 a[j+1] 的值
            int t = a[j];
            a[j] = a[j+1];
            a[j+1] = t;
        }
    }
}

```

完整代码如下：

```

public class TestBubbleSort{
    public static void main(String args[]){
        int[] a = {7,5,1,3,2};
        System.out.println("排序前");
    }
}

```

```

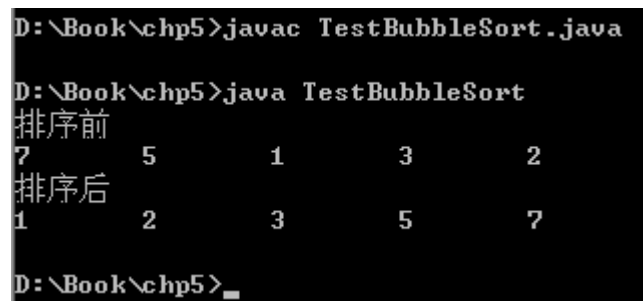
        printArray(a);

        for(int i = 0; i<a.length-1; i++){
            for(int 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.println("排序后");
        printArray(a);
    }

    public static void printArray(int[] a){
        for(int i = 0; i<a.length; i++){
            System.out.print(a[i] + "\t");
        }
        System.out.println();
    }
}

```

运行结果如下：



```

D:\Book\chp5>javac TestBubbleSort.java

D:\Book\chp5>java TestBubbleSort
排序前
7      5      1      3      2
排序后
1      2      3      5      7

D:\Book\chp5>_

```