



主讲教师 张 智  
计算机学院软件工程系  
课程群: 421694618

## 3 数组

### 3.1 一维数组

### 3.2 二维数组

### 3.3 数组编程

【附录1】 几种排序方法

【附录2】 Arrays类用法

## 3.1 一维数组

### ■ 一维数组的定义

`int a[5];` ✗ ← Java数组定义时一定不要指定长度

`int a[ ];` ✓ // 过时写法 (中括号放在数组名后面)

`int[ ] a;` ✓ // 推荐形式 (中括号放在类型后面)

`int[ ] s1, s2;` // s1, s2都是整型数组

`int s1[ ], s2;` // s1是整型数组, s2是整型变量

`String[ ] names;` // 字符串数组

`Point[ ] p;` // 对象数组

- Java数组要求所有的元素具有相同的数据类型
- Java数组是引用类型，例如：`int` 是一个基本类型，但 `int[]` 是一种引用类型
- Java数组既可以存储基本类型的数据，也可以存储引用类型的数据，只要所有的数组元素具有相同的类型即可

## 一维数组的初始化

### ■ 用new语句来初始化数组：

```
int[ ] a = new int[5];    //推荐写法
```

注：数组new之后元素默认值均为0

```
a[0] = 1; a[1] = 2; ... a[4] = 5; ← 特别注意：数组下标越界会报错(安全保证)
```

//其他方式

```
int[ ] b;
```

```
b = new int[5];    //分开写也ok
```

```
int n = 5;
```

```
int[ ] c = new int [n]; // 分配空间时可使用变量
```

```
c = new int[10];    // 指向新的引用 ✓
```

数组名只是一个引用，可用来引用其他相同类型的数组

## 快速初始化数组

`int[ ] a = { 1, 2, 3, 4, 5 };` //在定义时就初始化, 最简单, 推荐

注意: `int[ ] a; a={ 1,2,3,4,5 };` 分开写 ×

`int[ ] a = new int[ ] { 1,2,3,4,5 };` //这种形式也可以, 长度自动

不能指定元素个数

注意: `int[ ] a=new int[5]{ 1,2,3,4,5 };` 指定长度 ×

`String[ ] names = { "Tom", "Jerry", "Simon" };` //字符串数组

`Point[ ] p = { new Point(0,1), new Point(1,2) };` //对象数组

## 一维数组遍历：使用length属性

```
int[ ] arr = { 1, 2, 3, 4, 5 };  
for ( int i = 0; i < arr.length; i++) {  
    arr[i] += 100;  
}  
for ( int i = arr.length-1; i >= 0 ; i--) {  
    System.out.println( "arr[" + i + "]= " + arr[i] );  
}  
System.out.println( arr ); // 输出的是数组的地址值  
System.out.println( Arrays.toString(arr) ); // 一次性输出
```

length属性获得数组元素个数

运行结果

```
arr[4]=105  
arr[3]=104  
arr[2]=103  
arr[1]=102  
arr[0]=101  
[I@7291c18f  
[101, 102, 103, 104, 105]
```

需要导入包：import java.util.Arrays;

Arrays.toString(数组)：将数组转化为字符串

## 一维数组遍历的新方法 ★

### ■ for-each循环基本结构

```
for ( 类型 变量 : 数组 ) {  
    // 变量将依次遍历数组所有元素  
    // 变量类型一般与数组类型保持一致  
    // 注意遍历变量是只读的  
}
```

## 示例：for-each循环遍历数组

```
int [] arr = { 1, 2, 3, 4, 5 };  
  
for( int a : arr ) {  
    System.out.println(a);  
}
```



## 程序阅读

Main.java


```
public class Main {  
    public static void main(String[] args) {  
        int [ ] arr = { 1, 2, 3, 4, 5 };  
        for( int a : arr ) {  
            a += 100; //不会影响数组  
        }  
        System.out.println( Arrays.toString(arr) );  
    }  
}
```

运行结果：数组没有被修改

[1, 2, 3, 4, 5]

## 关于数组传参问题

- 在Java中，基本类型总是按值传递
- 对象类型为传引用 (如数组，包装类，集合类等)



注意：传引用仍然视为值传递!!! 对于对象来说，是将对象的引用也就是副本传递给了方法参数，在方法中只有对对象进行修改才能影响该对象的值，操作对象的引用是无法影响对象的。

```
public class Main {
```

基本类型是按值传递

```
    public static void swap1(int a, int b) {
```

```
        int c=a;
```

```
        a=b;
```

```
        b=c;
```

```
    }
```

该函数无法实现外部交换

数组作为形参传递的是引用

```
    public static void swap2( int[ ] a ) {
```

```
        a = new int[] { 99, 10 };
```

```
    }
```

操作对象引用无法影响外部对象

```
    public static void swap3( int[ ] a ) {
```

```
        int c = a[0];
```

```
        a[0] = a[1];
```

```
        a[1] = c;
```

```
    }
```

该函数能实现数组元素交换

//主程序

```
public static void main(String[] args) {
```

```
    int a=2, b=3;
```

```
    System.out.println("a="+a+",b="+b);
```

```
    swap1(a,b);    //不能实现交换
```

```
    System.out.println("a="+a+",b="+b);
```

```
    int[] arr = { 10,99 };
```

```
    System.out.println(Arrays.toString(arr));
```

```
    swap2(arr);    //不能实现交换
```

```
    System.out.println(Arrays.toString(arr));
```

```
    swap3(arr);    //实现交换
```

```
    System.out.println(Arrays.toString(arr));
```

```
}
```

```
}
```

运行结果

a=2, b=3

a=2, b=3

[10, 99]

[10, 99]

[99, 10]

## 程序阅读

```
public class Main {  
    public static void foo1( int[ ] a ){  
        a = new int[ ] {1, 2, 3};    // a指向了新的数组单元, 但外部aa并没有变化  
    }  
  
    public static void foo2( int[ ] a ){  
        if( a != null && a.length > 0 )  
            a[0]++;    //修改a的数据, 相当于修改外部aa的数据  
    }  
  
    public static void main(String[] args) {  
        int[ ] aa = { 3, 2, 1 };  
        System.out.println( Arrays.toString(aa) );  
        foo1(aa);  
        System.out.println( Arrays.toString(aa) );  
        foo2(aa);  
        System.out.println( Arrays.toString(aa) );  
    }  
}
```

运行结果

[3, 2, 1]
[3, 2, 1]
[4, 2, 1]

## 可变参数

- 可变参数用 ... 定义，例如：int ...x;
- 适用于参数个数不确定，类型确定的情况
- 可变参数当做数组来处理

## 可变参数示例

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println( add(2, 3) );    // 输出5  
        System.out.println( add(2, 3, 5) ); // 输出10  
    }  
}
```

可变参数：只能出现在参数列表最后

```
public static int add( int x, int... args ) {  
    int sum = x;  
    for ( int i = 0; i < args.length; i++ ) {  
        sum += args[i];  
    }  
    return sum;  
}
```

可变参数当做数组来处理

[【返回】](#)

## 3.2 二维数组

### ■ 二维数组创建:

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

或者:

```
int[ ][ ] a = new int [3][ ]; // 先指定第一维
```

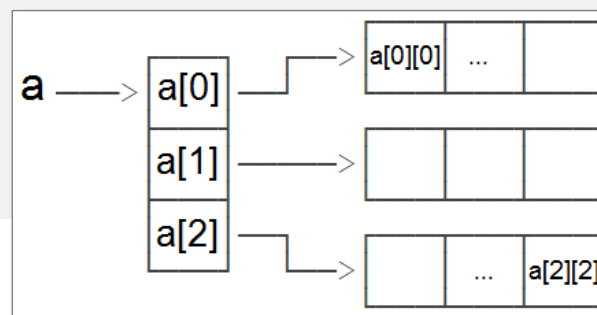
错误用法: `int a[ ][ ] = new int [ ][3]` ✗

```
a[0] = new int[3]; // 再第二维分配
```

```
a[1] = new int[3];
```

```
a[2] = new int[3];
```

视为  
一维  
数组



注意：二维数组的第二维大小可不相等

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

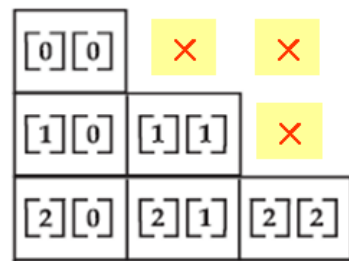
```
a[0] = new int[1];
```

```
a[1] = new int[2];
```

```
a[2] = new int[3];
```

第二维可以不相等（不规则数组）

没有分配的空间是不能访问的，如a[0][2]报错





## 二维数组初始化

### ■ 方式1：普通赋值

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

```
a[0][0] = 1;   a[0][1] = 2; ... a[2][2] = 9;
```

### ■ 方式2：一行行赋值（每行视为一维数组）

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

不要指定长度



```
a[0] = new int[ ] {1};   a[1] = new int[ ] {4,5};   a[2] = new int[ ] {7,8,9};
```

### ■ 方式3：简单用法

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

```
int[ ][ ] b= { {1}, {4,5} , {7,8,9} };
```

## 二维数组遍历：使用length属性

```
int[ ][ ] a = { {1}, {4,5} , {7,8,9} };  
for( int i=0; i < a.length ; i++) {           // 总行数  
    for(int j=0; j < a[i].length ; j++)        // 每行长度(列数)  
        System.out.print(String.format("%3d", a[i][j] ) );  
    System.out.println();  
}
```

## 二维数组遍历：使用for-each循环

```
int[ ][ ] a= { {1}, {4,5} , {7,8,9} };  
for ( int[ ] x : a ) {      // 遍历二维数组每一行  
    for ( int e : x ) {      // 遍历每一行元素  
        System.out.print(String.format("%3d", e));  
    }  
    System.out.println();  
}
```

思考：如何按列输出不规则二维数组？即上例输出为：1 4 7 5 8 9

[【返回】](#)

## 3.3 数组示例

1. [数组最大/小值](#)
2. [数组排序](#)：冒泡排序
3. [获取命令行参数](#)：输入多个整数求和
4. [for-each循环](#)：改写第二章2.5节例1（字符统计）
5. [Fibonacci数列](#)：前20项，普通和递归2种方法实现
6. [杨辉三角形](#)：10行，普通和递归2种方法实现

[【返回】](#)

## 1. 数组最大/小值

```
public class Ex_1 {  
    private static int max,min;  
  
    //求最大/最小值  
    public static void maxmin( int[ ] q ) {  
        max = min = q[0];  
        for(int i=1;i<q.length;i++) {  
            if( max<q[i] ) max=q[i];  
            if( min>q[i] ) min=q[i];  
        }  
    }  
}
```

```
public static void main(String[] args) {  
    int[ ] arr = { 9,3,18,5,6,1,10,15 };  
    System.out.println(Arrays.toString(arr));  
    maxmin( arr );  
    System.out.println("最大值:"+max+  
        "\n最小值:"+min);  
}
```

```
[9, 3, 18, 5, 6, 1, 10, 15]  
最大值:18  
最小值:1
```

[【返回】](#)

## 2. 数组排序

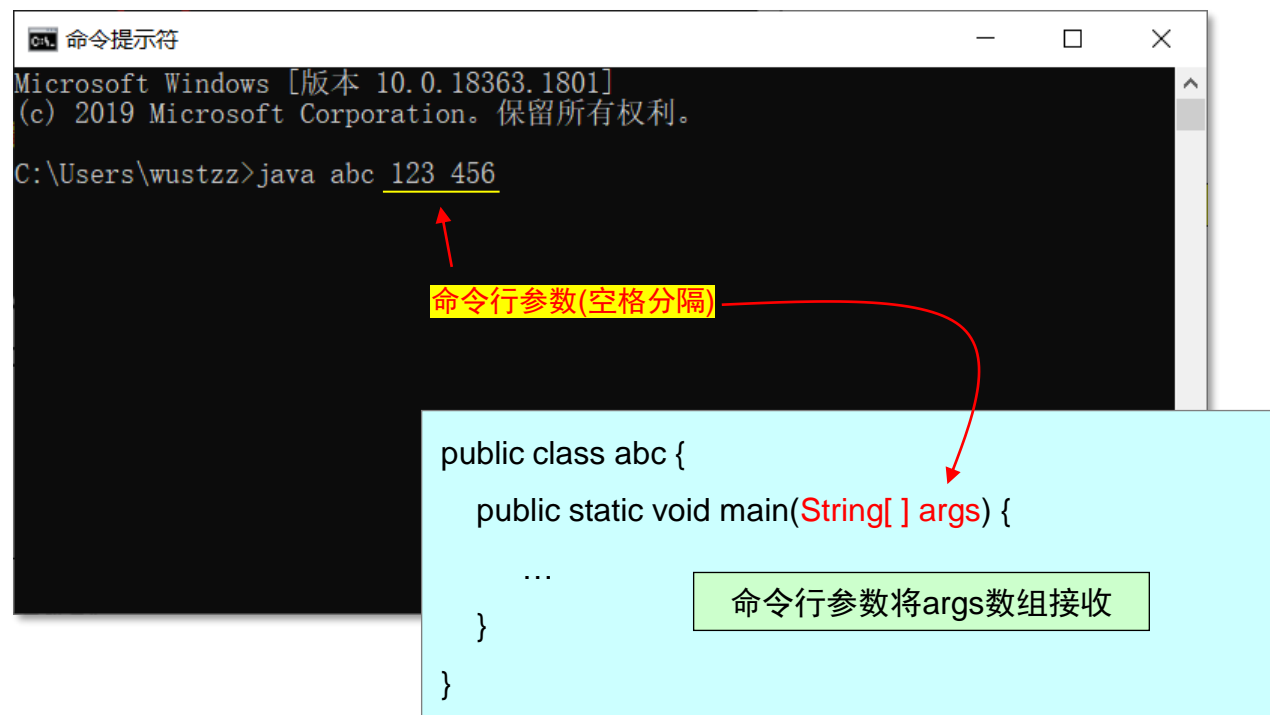
```
public class Ex_2 {  
    //冒泡排序  
    public static void bubble_sort( int[ ] a ) {  
        for (int i = 0; i < a.length - 1; i++) {  
            for (int j = 0; j < a.length - i - 1; j++) {  
                if (a[j] > a[j+1]) {  
                    // 交换 (大数放到后面)  
                    int tmp = a[j];  
                    a[j] = a[j+1];  
                    a[j+1] = tmp;  
                }  
            }  
        }  
    }  
    public static void main(String[] args) {  
        int[ ] arr = { 28, 12, 89, 73, 65, 18, 96, 50, 8, 36 };  
        System.out.println( Arrays.toString(arr) );  
        bubble_sort ( arr );  
        System.out.println( Arrays.toString(arr) );  
    }  
}
```

```
[28, 12, 89, 73, 65, 18, 96, 50, 8, 36]  
[8, 12, 18, 28, 36, 50, 65, 73, 89, 96]
```

[【返回】](#)

### 3. 获取命令行参数:

- 命令行参数: 在运行Java程序时, 可以通过命令行参数传值给main函数



## 示例：对命令行输入的多个整数求和

```
public class Ex_3 {  
    public static void main(String[] args) {  
        int sum=0;  
  
        for( int i=0; i < args.length; i++ ) {  
            sum += Integer.parseInt( args[i] );  
        }  
  
        System.out.println( Arrays.toString(args) );  
  
        System.out.println("和="+sum);  
    }  
}
```

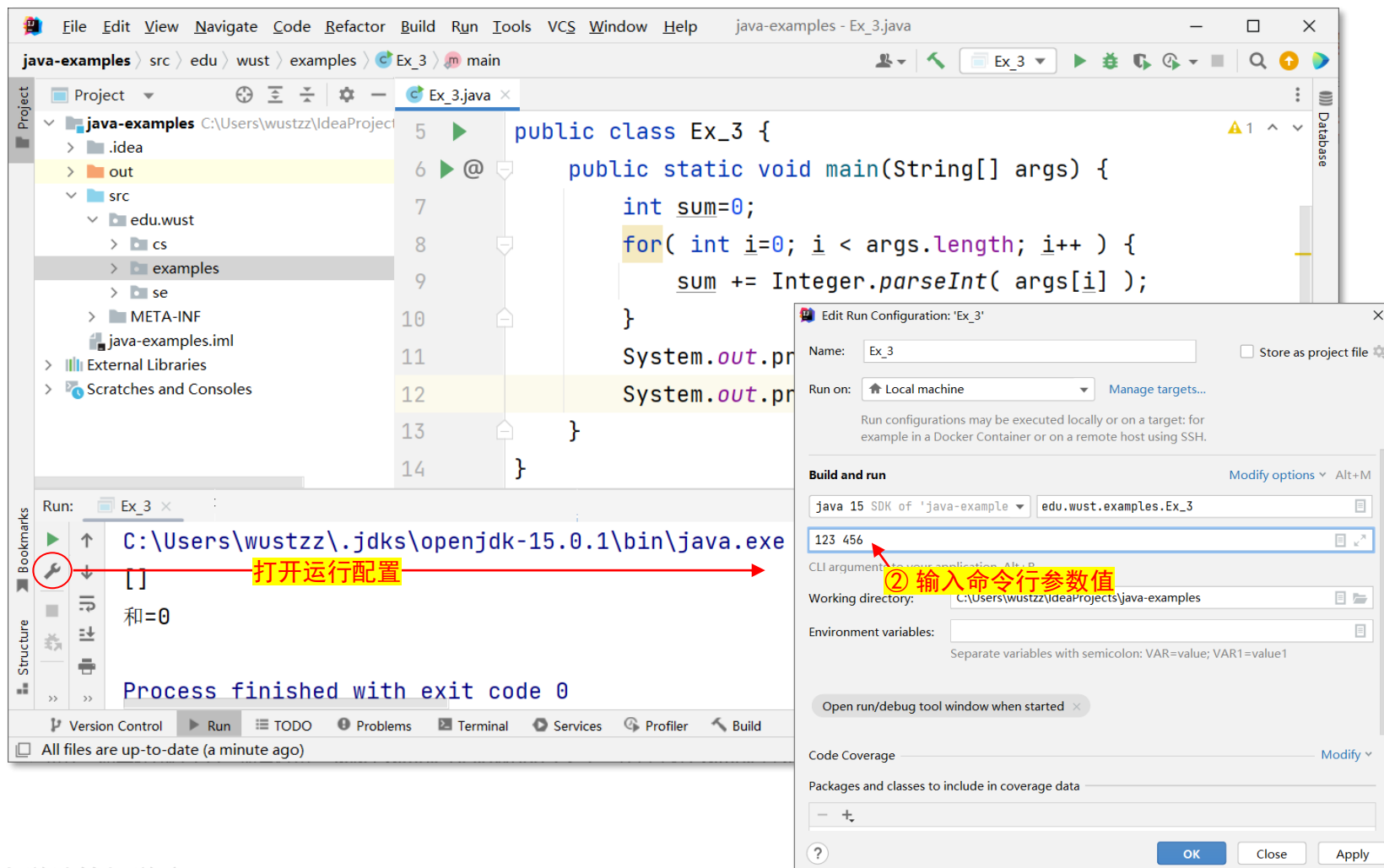
接收命令行参数

补充：main参数也可写成可变参数：  
public static void main(String... args)

将字符串转为int类型



## 如何输入(配置)命令行参数



③ 运行结果

[123, 456]  
和=579

【返回】

## 4. for-each循环：修改2.6章节例1

[原例代码](#)

```
public static void count(String s) {  
    for ( int i = 0; i < s.length(); i++ ) {  
        char ch = s.charAt(i);  
        if ( Character.isLowerCase(ch) )  
            lower++;  
        else if ( Character.isUpperCase(ch) )  
            upper++;  
        else if ( Character.isDigit(ch) )  
            number++;  
        else if ( Character.isWhitespace(ch) )  
            space++;  
        else  
            other++;  
    }  
}
```

修改如下：

```
public static void count(String s) {  
    char[ ] chArr = s.toCharArray(); ←  
    for ( char ch : chArr ) {  
        if ( Character.isLowerCase(ch) )  
            lower++;  
        else if ( Character.isUpperCase(ch) )  
            upper++;  
        else if ( Character.isDigit(ch) )  
            number++;  
        else if ( Character.isWhitespace(ch) )  
            space++;  
        else  
            other++;  
    }  
}
```

先将字符串转换为字符数组

[【返回】](#)

## 5. Fibonacci数列：非递归实现

非递归实现

```
public class Ex_5 {  
    public static void main(String[] args) {  
        int n = 20;  
        int arr[ ] = new int[n];  
        fib(arr);  
        for ( int x:arr ) {  
            System.out.print(String.format("%d ", x));  
        }  
    }  
}
```

new分配空间时可使用变量



//Fibonacci数列

```
public static void fib(int[] a) {  
    a[0] = a[1] = 1;  
    for ( int i = 2; i < a.length; i++ )  
        a[i] = a[i - 1] + a[i - 2];  
}
```

1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765
---	---	---	---	---	---	----	----	----	----	----	-----	-----	-----	-----	-----	------	------	------	------

## Fibonacci数列：递归实现

递归实现

```
public class Ex_5 {  
    public static void main(String[] args) {  
        int n = 20;  
        int arr[] = new int[n];  
        for (int i = 0; i < n; i++) {  
            arr[i] = fab(i);  
        }  
        for (int x:arr) {  
            System.out.print(String.format("%d ", x));  
        }  
    }  
}
```

//Fibonacci数列

```
public static int fab(int n) {  
    if (n == 0 || n == 1)  
        return 1;  
    else  
        return fab(n - 1) + fab(n - 2);  
}  
}
```

递归公式

[【返回】](#)

## 6. 杨辉三角型：输出10行

```
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
```

非递归实现

```
public class Ex_6 {  
  
    //杨辉三角形  
  
    public static void yang(int[][] a) {  
        for (int i = 0; i < a.length; i++) {  
            a[i] = new int[i + 1];    //每行元素个数=i+1  
            a[i][0] = 1;  
            a[i][i] = 1;    //每行首尾数据都为1  
        }  
        for (int i = 2; i < a.length; i++)  
            for (int j = 1; j < a[i].length-1; j++)  
                a[i][j] = a[i - 1][j - 1] + a[i - 1][j];  
    }  
}
```

```
public static void main(String[] args) {  
    int n = 10;  
    int a[][] = new int[n][];  
    yang(a);  
    for (int i = 0; i < a.length; i++) {  
        for (int j = 0; j < a[i].length; j++)  
            System.out.print(String.format("%-4d", a[i][j]));  
        System.out.println();  
    }  
}
```

## 递归实现

```
public class Ex_6 {  
  
    //杨辉三角形  
  
    public static int yang(int i, int j) {  
        if (j == 0 || i == j) return 1;  
        return yang(i - 1, j - 1) + yang(i - 1, j);  
    }  
  
    public static void main(String[] args) {  
        int n = 10;  
        int a[][] = new int[n][];  
        for (int i = 0; i < n; i++)  
            a[i] = new int[i + 1];    //每行元素个数=i+1
```

```
        for (int i = 0; i < a.length; i++)  
            for (int j = 0; j < a[i].length; j++)  
                a[i][j] = yang(i, j);    //调用递归方法  
  
        for (int i = 0; i < a.length; i++) {  
            for (int j = 0; j < a[i].length; j++)  
                System.out.print(String.format("%-4d", a[i][j]));  
            System.out.println();  
        }  
    }  
}
```

[【返回】](#)



## 【附录1】几种排序方法

### 选择排序

```
public static void select_sort (int[] arr) {  
    for (int i = 0; i < arr.length - 1; i++) {  
        int minIndex = i;    // 用来记录最小值的索引位置，默认值为i  
        for (int j = i + 1; j < arr.length; j++) {  
            if (arr[j] < arr[minIndex]) {  
                minIndex = j;    // 遍历 i+1~length 的值，找到其中最小值的位置  
            }  
        }  
        // 交换当前索引 i 和最小值索引 minIndex 两处的值  
        if (i != minIndex) {  
            int temp = arr[i];  
            arr[i] = arr[minIndex];  
            arr[minIndex] = temp;  
        }  
        // 执行完一次循环，当前索引 i 处的值为最小值，直到循环结束即可完成排序  
    }  
}
```

选择排序算法思想：遍历元素找到一个最小（或最大）的元素，把它放在第一个位置，然后再在剩余元素中找到最小（或最大）的元素，把它放在第二个位置，依次下去，完成排序。

## 插入排序

```
public static void insert_sort(int[] arr) {  
    int j;    // 已排序列表下标  
    int t;    // 待排序元素  
    for (int i = 1; i < arr.length; i++) {  
        if (arr[i] < arr[i - 1]) {  
            t = arr[i];    // 赋值给待排序元素  
            for (j = i - 1; j >= 0 && arr[j] > t; j--) {  
                arr[j + 1] = arr[j];    // 从后往前遍历已排序列表，逐个和待排序元素比较，如果已排序元素较大，则将它后移  
            }  
            arr[j + 1] = t;    // 将待排序元素插入到正确的位置  
        }  
    }  
}
```

插入排序算法思想：

- ① 从第一个元素开始，该元素可以认为已经排好序。
- ② 取出下一个新元素，然后把这个新元素在已经排好序的元素序列中从后往前扫描进行比较。
- ③ 如果该元素(已排序)大于新元素，则将这个已排序的元素移到下一个索引位置。
- ④ 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置。
- ⑤ 将新元素插入到该已排序的元素的索引位置后面。
- ⑥ 重复步骤②-⑤

```
public static int[] quick_sort(int[] arr, int left, int right) {  
    if (left < right) {  
        int partitionIndex = partition(arr, left, right);  
        quick_sort(arr, left, partitionIndex - 1);  
        quick_sort(arr, partitionIndex + 1, right);  
    }  
    return arr;  
}
```

```
public static int partition(int[] arr, int left, int right) {  
    // 设定基准值 (pivot)  
    int pivot = left;  
    int index = pivot + 1;  
    for (int i = index; i <= right; i++) {  
        if (arr[i] < arr[pivot]) {  
            swap(arr, i, index);  
            index++;  
        }  
    }  
    swap(arr, pivot, index - 1);  
    return index - 1;  
}
```

```
public static void swap(int[] arr, int i, int j) {  
    int temp = arr[i];  
    arr[i] = arr[j];  
    arr[j] = temp;  
}
```

快速排序算法思想：

- ① 从数列中挑出一个元素，称为 “基准” (pivot)
- ② 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区 (partition) 操作；
- ③ 递归地 (recursive) 把小于基准值元素的子数列和大于基准值元素的子数列排序。

[【返回】](#)

## 【附录2】Arrays类用法

```
import java.util.Arrays;
```

- Arrays是针对数组的工具类，提供了**排序，查找，复制填充，二分查找**等功能，大大提高了开发效率

Arrays常用方法	说明
<code>sort(array)</code>	对指定的基本数据类型数组array按升序排列
<code>binarySearch(array,val)</code>	对基本数据类型数组array进行二分查找val
<code>equals(array1,array2)</code>	如果两个指定的基本数据类型数组长度和内容都相等返回true
<code>fill(array,val)</code>	将指定的基本数据类型值数组array的所有元素都赋值为val (填充)
<code>copyof(array,length)</code>	把基本数据类型数组array复制成一个长度为length的新数组
<code>toString(array)</code>	把基本数据类型数组array内容转换为字符串
<code>asList(T... a)</code>	把数组a转换成List集合（注：不可以使用集合的增删方法）

```
int[] a = new int[] { 3, 4, 5, 6 };
int[] a2 = new int[] { 3, 4, 5, 6 };
// a数组和a2数组的长度相等，每个元素依次相等，将输出true
System.out.println("a数组和a2数组是否相等: " + Arrays.equals(a, a2));
// 通过复制a数组，生成一个新的b数组
int[] b = Arrays.copyOf(a, 6);
System.out.println("a数组和b数组是否相等: " + Arrays.equals(a, b));
// 输出b数组的元素，将输出[3, 4, 5, 6, 0, 0]
System.out.println("b数组的元素为: " + Arrays.toString(b));
// 将b数组的第3个元素（包括）到第5个元素（不包括）赋值为1
Arrays.fill(b, 2, 4, 1);
// 输出b数组的元素，将输出[3, 4, 1, 1, 0, 0]
System.out.println("b数组的元素为: " + Arrays.toString(b));
// 对b数组进行排序
Arrays.sort(b);
// 输出b数组的元素.将输出[0,0,1,1,3,4]
System.out.println("b数组的元素为: " + Arrays.toString(b));
// b数组(已排序)查找1和2，找到返回索引值(第一个)
// 没找到返回"-插入点"。插入点指搜索键将要插入数组的位置值(=索引值+1)
System.out.println("查找1位置为: " + Arrays.binarySearch(b,1));
System.out.println("查找2位置为: " + Arrays.binarySearch(b,2));
```

```
a数组和a2数组是否相等: true
a数组和b数组是否相等: false
b数组的元素为: [3, 4, 5, 6, 0, 0]
b数组的元素为: [3, 4, 1, 1, 0, 0]
b数组的元素为: [0, 0, 1, 1, 3, 4]
查找1位置为: 2
查找2位置为: -5
```

【完】