复习：

1. 循环结构：while，for ---循环的嵌套
2. break和continue：break可以用于循环和选择结构continue只能用于循环结构

## 数组

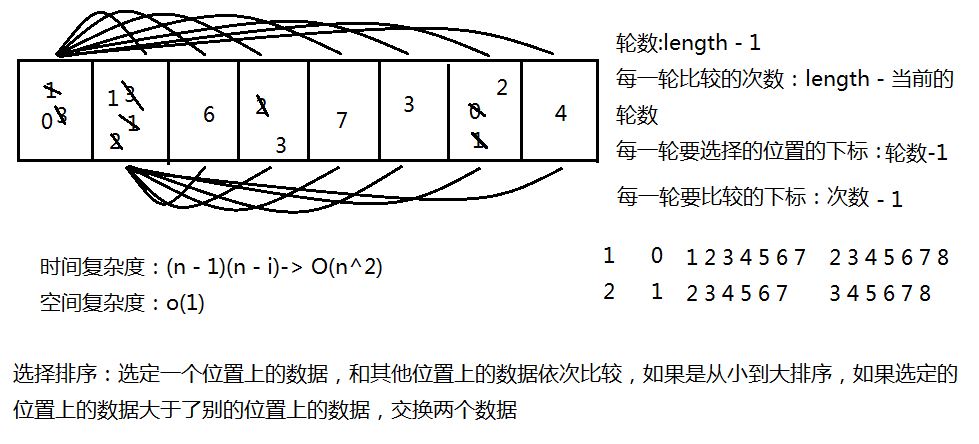
定义格式---3种

数组的内存---Java将内存分为了5块：栈内存、堆内存、方法区、本地方法栈、寄存器---数组是存在堆内存中。会自动的赋予默认值

### 数组的应用

1. 获取数组的元素
2. 数组的长度
3. 遍历数组
4. 获取最值
5. 数组的排序

冒泡排序：时间复杂度---O（n2）,空间复杂度---o(1)



// 选择排序

for(int i = 1; i < arr.length; i++){ // 控制轮数和选择的下标

for(int j = 1 + i; j <= arr.length ; j++){ // 控制次数和比较的下标

if(arr[i - 1] > arr[j - 1]){

int temp = arr[i - 1];

arr[i - 1] = arr[j - 1];

arr[j - 1] = temp;

}

}

}

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

Arrays.sort(数组);---只能升序排列，时间复杂度是O(nlogn)

1. 数组的反转

// 首尾互换

for(int start = 0, end = arr.length - 1; start < end; start++, end--){

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

}

1. 数组元素的查找

int[] arr = {1,3,5,9,14,25,53,72};

int number = 58;

// 折半查找

// 时间复杂度是O(logn)

// 空间复杂度是o(1)

int min = 0;

int max = arr.length - 1;

int mid = (min + max) / 2;

while(arr[mid] != number){

if(arr[mid] > number){

max = mid - 1;

} else {

min = mid + 1;

}

if(min > max){

mid = -1;

break;

}

mid = (min + max) / 2;

}

System.out.println(mid);

1. 数组的复制

System.arraycopy(要复制的数组,要复制的起始下标,要存放的数组,要存放的起始下标,要复制的元素的个数);

数组的扩容/缩容---本质上就是数组的复制，所以扩容之后地址必然发生了改变

数组 = Arrays.copyOf(要扩容的数组, 扩容之后的长度);

### 二维数组

存储数组的数组---二维数组---二维数组中的每一个元素实际上是一个一维数组

#### 定义格式

数据类型[][] 数组名 = new 数据类型[二维数组的长度][包含的一维数组的长度];

int[][] arr = new int[3][5]; --- 定义包含3个一维的整型数组，每个一维数组能存储5个整数。

数据类型[][] 数组名 = new 数据类型[二维数组的长度][];

int[][] arr = new int[3][]; --- 定义了包含3个一维的整型数组，没有规定每一个一维数组的大小

arr[0] = new int[3];

arr[1] = new int[7];

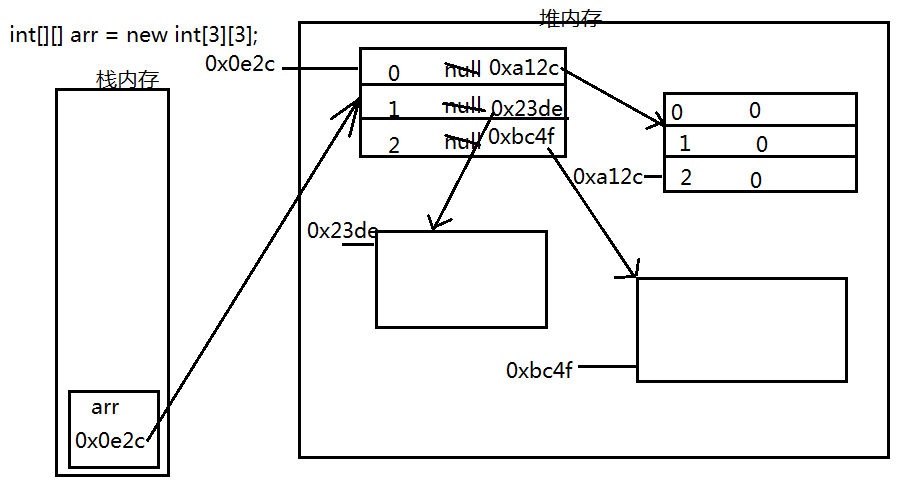
arr[2] = new int[4];

数据类型[][] 数组名 = {{一维数组1},{一维数组2},{一维数组3}, ... ,{一维数组n}};

int[][] arr = {{2,4}, {1,2,3,5,7}, {8}, {8,4,3,0}}; --包含了4个一维数组

System.out.println(arr[1][2]); --- 3

System.out.println(arr[1]);---打印对应的一维数组的地址



如果没有给二维数组中对应的一维数组定义大小，那么针对这个一维数组操作会出现空指针异常---NullPointerException

面试题：对于int[] x[], y均以初始化，下列各项中正确的是：---BC

1. x[0][0] = y;
2. x[0] = y;
3. x[0][0] = y[0];
4. x[0] = y[0];
5. x = y;

练习：杨辉三角

1

1. 1
2. 2 1
3. 3 3 1
4. 4 6 4 1
5. 5 10 10 5 1

arr[i][j] = arr[i - 1][j] + arr[i - 1][j - 1];

输入行数，然后输出对应的前n行

## 方法

将一段代码/逻辑提取出来进行包装，这种包装形式就称之为方法/函数。---为了方便重复利用这段代码

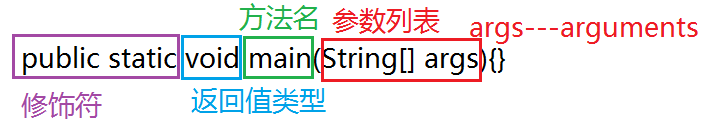
#### 定义格式

修饰符 返回值类型 方法名(参数列表){

代码块;

return 返回值;

}



// 求2个整数的和

// 明确方法的返回值类型---求2个整数的和---结果一定是一个int类型---返回值类型是int

// 明确方法是否需要未知量---这2个整数对于当前方法而言是2个未知量，需要以参数的形式体现

// 形式参数---形参

public static int add(int i, int j){

int sum = i + j;

// 将结果返回给用方法的人

return sum;

}

// 打印m行n列的\*组成的矩形

// 明确返回值类型 --- 打印矩形 --- 没有结果 --- void

// 明确是否有未知量---m行n列，m和n是两个未知量，需要以参数形式来体现

public static void printStar(int m, int n){

for(int i = 0; i < m; i++){

for(int j = 0; j < n; j++){

System.out.print("\*");

}

System.out.println();

}

}

练习：

1. 哥德巴赫猜想：任意一个大于等于6的偶数，都能分解成两个质数之和

18 = 7 + 11

18 = 5 + 13

14 = 3 + 11

14 = 7 + 7

输入一个大于等于6的偶数，将其进行分解

2. 亲密数：如果A的所有因子（包含1而不包含本身）之和等于B，并且B的所有因子（包含1而不包含本身）之和等于A，A和B是一对亲密数---打印3000以内的所有的亲密数

12 : 1 2 3 4 6 12

1+2+3+4+6=16

16：1 2 4 8 16

1+2+4+8=15

### 方法的递归

public static void main(String[] args){

// StackOverFlowError ---栈溢出错误

// 方法在栈中执行

// 当方法执行的时候需要在栈中占用一部分空间，当递归的次数比较多的时候，会耗费大量的内存。如果栈内存不够用了就会出现栈溢出错误

// 循环的效率要高于递归---所以能使用循环的而时候减少使用递归

System.out.println(sum(10500));

}

public static int sum(int n){

if(n == 1)

return 1;

// 在方法中调用了自己本身 --- 方法的递归

return n + sum(n - 1);

}

练习：求n的阶乘 ---> 5! = 5 \* 4 \* 3 \* 2 \* 1

智能开发工具

JCreator--现在不用了

Eclipse --- 日食 --- 绿色的、开源的、免费的、基于插件 -- 要求必须安装了JDK

Luna --- Mars(jdk1.7) --- Neon.2(jdk1.8) Neon.3 ---- Oxygen(jdk1.8)

MyEclipse --- 适合于Web开发 --- 收费

IntelliJ ---IDEA ---适合于Web，Maven管理和代码重构性比MyEclipse要好很多 -- 收费

Alt+/ 提示键

Ctrl+D 删除一行

Ctrl+Shift+o 自动导包

方法的传值和方法的重载