# 方法 Method：

作用：为了减少重复的代码

语法： 三个要素 返回值 方法名 参数列表（变化的部分）

重点

static 返回值 方法名(参数1, 参数2, ... 参数n) {

// 方法内容(重复的代码）

}

调用方法：

重点

方法名(参数值1, 参数值2, ... 参数值n)

定义一个方法, 计算两个整数的和

static int add(int a , int b) {

int c = a + b;

return c;

}

如果方法需要返回一个结果，使用return关键字返回，方法的返回值类型定义需要与实际的结果类型一致，return是方法执行时最后一条语句

注意1. static方法可以调用其他static方法

static 方法不能调用其他非static 方法

非static 方法可以调用static 方法

注意2： 如果方法有返回值，方法内有分支条件，每个分支条件都必须考虑返回值

## 方法的重载：

方法的名字如果一样，但参数的个数和类型如果不一样，称之为方法被重载

比如 System.out.println(字符串);

System.out.println(int);

System.out.println(double);

static int add(int a , int b) {

int c = a + b;

return c;

}

static double add(double a, double b) {

double c = a + b;

return c;

}

// 不能个数和类型都一样：定义会冲突

static int add(int c, int d ) {

return 10;

}

对于重载后的方法，调用时会根据实参的类型，找到与之最匹配的方法进行调用

例如：

add(5,6) 会调用第一个add方法

add(5.0, 6.0) 会调用第二个add方法

## 方法执行时的内存分配

每个方法在执行时，占用的内存是该方法私有的，别的方法不能干预

方法参数，方法内的局部变量，作用范围在方法之内

方法调用时，参数的值其实是进行了拷贝操作，不会影响原来的值

## 方法的递归调用（难点）

求阶乘

int n = 5;

// 5\*4\*3\*2\*1

int result = 1;

for(int i = 1; i <= 5; i++) {

result = result \* i

}

用递归方法求阶乘

static int jiecheng(int n) { n=1

if(n==1) {

reutrn 1;

}

}

static int jiecheng(int n) { n = 2

return n\*jiecheng(n-1); // 2 \* 1

}

static int jiecheng(int n) { n = 3

return n\*jiecheng(n-1); // 3\* jiecheng(2)

}

static int jiecheng(int n) { n = 4

return n\*jiecheng(n-1)

}

# 数组：

定义：一组类型相同的变量

语法：

int[] a;

double[] b;

String[] c;

分配空间: 确定数组的大小

a = new int[10]; // 为整型数组分配空间，最多存10个

b = new double[5]; // 为double数组分配空间，最多存5个

分配空间会为数组元素赋一个默认值

byte[] short[] int[] long[] 默认值为0

float[] double[] 默认值为0.0

char[] 默认值 '\u0000' = 0

boolean[] 默认值 false

其他类型:例如 String[] 默认值 null

通过下标可以获取或修改数组元素

int[] a;

a = new int[10];

a[下标] 下标从0开始

a[0] a[1] .... a[9]

重点

// 1) 定义数组 + 分配空间

// // 定义变量

// int[] a;

// // 分配空间

// a = new int[10];

// 2) 定义数组 + 分配空间

// int[] a = new int[10];

// 3) 定义数组 + 分配空间 + 赋初始值 (不用加大小)

int[] a ;

a = new int[]{1,2,3,4,5,6,7,8,9,10};

// 4) 定义数组 + 分配空间 + 赋初始值　(对 方式3的简化，不能分开)

// int[] a = {1,2,3,4,5,6,7,8,9,10};

// 5) 定义数组的两种写法

int[] b;

int c[];

## 遍历数组：

重点

// 方法1： i代表数组下标

for(int i = 0; i < 数组.length; i++){

元素 = 数组[i]

}

// 方法2：

for(元素类型 元素的变量 : 数组变量 ){

}

// 方法3：

java.util.Arrays.toString(数组);

## 数组变量和基本类型变量之间的差别(重点)考试

重点

public static void main(String[] args) {

int a = 10;

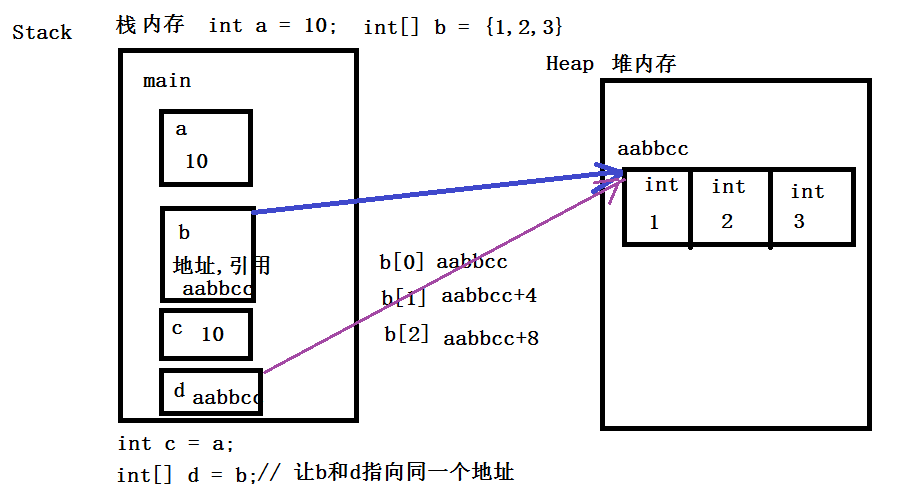
int[] b = {1,2,3};

int c = a;

int d = b;

}

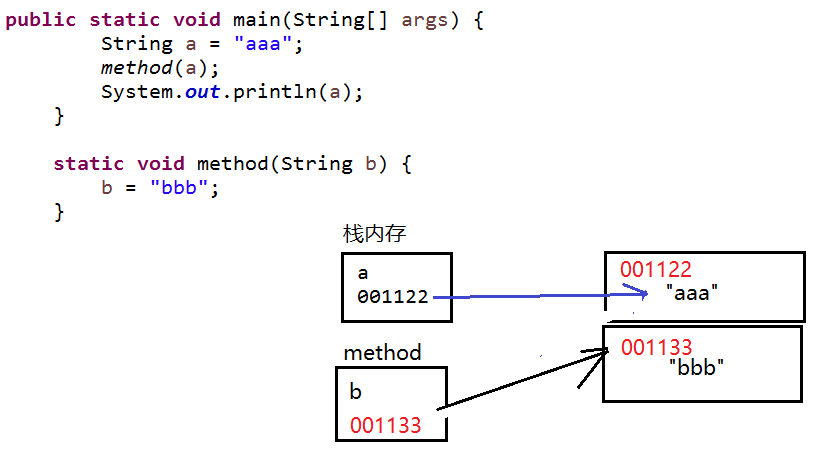
方法内的定义的基本类型变量（局部变量），使用的内存称为 栈内存



普通类型的变量，(int c=a) 赋值是值进行了拷贝

数组类型的变量, (int[] d = b) 赋值时把地址进行了拷贝

除了8种基本数据类型以外，剩下所有类型都存储是一个地址(引用)



## 二维数组(了解)

int[] array = new int[]{1,2,3}; // 一维数组

int[][] array = new int[3][2]; // 二维数组

|  |  |
| --- | --- |
| a[0][0] | a[0][1] |
| a[1][0] | a[1][1] |
| a[2][0] | a[2][1] |

## 数组的应用：

1. 倒置元素(重点)

{1,2,3,4,5} => {5,4,3,2,1}

**public** **class** TestArrayReverse {

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

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

// 关键，交换数组 元素的位置

// int i = 0; i < array.length/2; i++

// 0 与 array.length-(0+1)

// 1与 array.length-(1+1)

// 2 与 array.length-(2+1) ...

// array.length /2

// int tmp = array[0];

// array[0] = array[4];

// array[4] = tmp;

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

**for**(**int** i = 0; i < array.length/2; i++) {

**int** tmp = array[i];

array[i] = array[array.length - (i+1)] ;

array[array.length - (i+1)] = tmp;

}

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

}

}

1. 排序

冒泡排序 (重点)

// 测试冒泡排序

**public** **class** TestBubble {

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

**int**[] array = **new** **int**[] {5,4,3,2,1};

// 模拟第一轮比较，两两比较，将最大的元素交换到数组的末尾

**for**(**int** i=0; i < array.length-1; i++) {

**for**(**int** j=0; j< array.length-i-1; j++) {// 4 3 2 1

// 相邻的元素比较,如果array[j]>array[j+1] 交换次序

**if**(array[j] > array[j+1]) {

**int** tmp = array[j];

array[j] = array[j+1];

array[j+1] = tmp;

}

}

}

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

// for(int j=0; j< array.length-1-1; j++) {// 4 3 2 1

// // 相邻的元素比较,如果array[j]>array[j+1] 交换次序

// if(array[j] > array[j+1]) {

// int tmp = array[j];

// array[j] = array[j+1];

// array[j+1] = tmp;

// }

// }

// System.out.println(Arrays.toString(array));

}

选择排序、

快速排序、归并排序 ...

作业：

1. 写一个方法，接受一个整数，返回这个整数是几位数

|  |
| --- |
| **public static int** test(**int** n){  **int** count=0;  **while**(n>0){  n=n/10;  count++;  }  **return** count; } |

1. 写一个方法，判断是否是一个质数，如果是质数返回true，否则返回false

|  |
| --- |
| **public static boolean** test(**int** n){  **for**(**int** i=2;i<Math.*sqrt*(n);i++){  **if**(n%i==0){  **return false**;  }  }  **return true**; } |

1. 用递归和非递归分别实现一个方法求阶乘

|  |
| --- |
| **public static int** test(**int** n) {  **if** (n == 1) {  **return** 1;  } **else** {  **return** n \* *test*(n - 1);  } } |

1. 验证歌德巴赫猜想：任何一个大于6 的偶数，都能分解成两个质数的和。 要求输入一个整数，输出这个数能被分解成哪两个质数的和。

eg :

14

14=3+11

1*4=7+7*

|  |
| --- |
| **public static void** test(**int** n) {  **for**(**int** i=2;i<n;i++){  **if**(*iszhi*(i)&&*iszhi*(n-i)){  System.***out***.println(n+**"="**+i+**"+"**+(n-i));  }  } } **public static boolean** iszhi(**int** n){  **for**(**int** i=2;i<n;i++){  **if**(n%i==0){  **return false**;  }  }  **return true**; } |

1. （扩展）用递归的方法解决汉诺塔问题 汉诺塔是源自印度神话。上帝创造世界的时候做了三根金刚石柱子，在一根 柱子上从下往上安大小顺序摞着n 片黄金圆盘。上帝命令婆罗门把圆盘从下面开 始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘， 在三根柱子之间一次只能移动一个圆盘。

要求：读入 n，在方法内输出移动的顺序

a b c

a -> b

a -> c

1. 设计一个static void add(int i)方法，为一个static整型数组安全添加元素，要求：

新的元素加入到数组的尾部，一旦超过数组容量，数组自动扩容为原来大小的2倍

例如： static int[] array = {1,2,3,4,5};

当调用 add(6)后，数组的内容变为 {1,2,3,4,5,6,0,0,0,0}

1. 给定一个数组，把这个数组中所有元素顺序进行颠倒
2. 输出杨辉三角 杨辉三角如下：

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

……

杨辉三角的特点：

1. 第i 行有i 个元素
2. 每一行的第一个元素和最后一个元素都为1
3. 除了1 之外，每个元素的值，都等于上一行同位置的元素以及前一个元 素的和

要求：读入一个整数n，输出杨辉三角的前n 行

1. 参考课堂的冒泡排序的例子，实现选择排序算法

选择排序算法：

首先在未排序序列中找到最小元素，存放到排序序列的起始位置

然后，再从剩余未排序元素中继续寻找最小元素，然后放到已排序序列的末尾。 以此类推，直到所有元素均排序完毕。

第一轮

a[0]

a[0] a[1]

a[0] a[2]

a[0] a[3]

a[0] a[4]

第二轮

a[1]

a[1] a[2]

a[1] a[3]

a[1] a[4]

...

|  |
| --- |
| **public static int**[] test(**int**[] arr) {  **for**(**int** i=0;i<arr.**length**;i++){  **for**(**int** j=i+1;j<arr.**length**;j++){  **if**(arr[i]<arr[j]){  **int** temp=arr[i];  arr[i]=arr[j];  arr[j]=temp;  }  }  }  **return** arr; } |

1. (选做)自己研究快速排序和归并排序算法，并尝试使用java语言实现

需要用递归

1. 十五个猴子围成一圈选大王，依次1-7 循环报数，报到7 的猴子被淘汰， 直到最后一只猴子成为大王。问，哪只猴子最后能成为大王？

|  |
| --- |
| **public static void** main(String[] args){  **boolean**[] houzi=**new boolean**[15];  **int** i=0;  **int** baoshu=1;  **int** taotai=0;  **while**(taotai!=14){  **if**(houzi[i]==**false**){  baoshu++;  **if**(baoshu==7){  houzi[i]=**true**;  taotai++;  baoshu=1;  }  }  i++;  **if**(i==15){  i=0;  }  **for** (**int** j = 0; j <houzi.**length** ; j++) {  System.***out***.print(houzi[j]+**" "**);  }  System.***out***.println();  } } |