# 数组定义格式

## 1.1数组的定义格式

* 格式1： 数据类型[ ] 变量名
* 例如：int [] arr
* 定义了一个int类型的数组，数组名为arr
* 格式2：数据类型 变量名[ ]
* 例如：int arr [ ]
* 定义了一个int类型的变量，变量名为arr数组

# 数组初始化

## 2.1动态初始化：初始化时只指定数组长度，由系统为数组分配初始值

* 格式：数据类型 [ ] 变量名 = new 数据类型[数组长度];
* 例如：int [ ] arr = new arr[3];

int[ ] arr = new int[3];

/\*

\* 左边：

\* int:说明数组中的元素类型是int类型

\* []:数组

\* arr:数组名称

\* 右边：

\* new:为数组申请内存空间

\* int:说明数组中的元素类型是int型

\* 3:数组长度，其实就是数组中的元素个数

\* \*/

## 2.2静态初始化：初始化时指定每个数组元素的初始值，由系统决定数组长度

* 格式：数据类型 [ ] 变量名 = new 数据类型[ ]{数据1，数据2，数据3，……}
* 例如：int [] arr = new int []{1,2,3….}
* 简化格式：数据类型[] 变量名 = {数据1，数据2，数据3，……}
* 例如： int [] arr = {1,2,3,….}

## 2.3两个常见小问题

* 索引越界：访问了数组中不存在的索引对应的元素，造成索引越界
* 空指针异常：访问的数组已经不再指向堆内存的数据，造成空指针异常
* null：空值，引用数据类型的默认值，表示不指向任何有效对象

# 3.数组元素访问

## 3.1数组元素访问

* 数组变量访问方式：数组名
* 数组内部保存的数据的访问方式：数组名[索引]
* 索引是数组中数据的编号方式
* 作用：索引用于访问数组中的数据使用，数组名[索引]等于变量名，是一种特殊的变量名
* 特征：索引从0开始，索引是连续的，逐一增加，每次加1

# 4.内存分配

## 4.1Java中内存分配

* 栈内存：存储局部变量，定义在方法中的变量，例如：arr，使用完毕，立即消失
* 堆内存：存储new出来的内容（实体，对象）数组在初始化时，会为存储空间添加默认值。整数：0 浮点数：0.0 布尔：false 字符：空字符 引用数据类型：null。每一个new出来的东西都有一个地址值，使用完毕，会在垃圾回收空闲时被回收。

001 new int[3]

Int [ ] arr 001(数组名)

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |

堆内存

栈内存

# 5.数组常见操作

## 5.1遍历

## 5.2获取数组元素数量

* 格式：数组名.length
* 例如：arr.length

## 5.3获取最值

思路：

1：定义一个变量，用于保存最大值或最小值

2：取数组中第一个数据作为变量的初始值

3：与数组中剩余的数据逐个对比，每次比最大值保存到变量中

4：循环结束后打印变量值

# 6方法概述

## 6.1什么是方法

* 方法：是将具有独立功能的代码块组织成为一个整体，使其具有特殊功能的代码集
* 方法必须先创建才能使用，该过程称为方法定义
* 方法创建后并不是直接运行，需手动使用后执行，该过程称为方法调用

## 6.2方法的定义和调用

* 格式：public static void 方法名（）{

//方法体

}

* 范例：public static void isEvenNumber(){

//方法体

}

* 方法调用 格式：方法名（）； 例如：isEvenNumber();
* 方法必须先定义再调用

# 7带参数方法的定义和调用

## 7..1.带参数方法定义

* 格式： public static void 方法名(参数){。。。。。}
* 格式（单个参数） public static void 方法名(数据类型 变量名){……}
* 格式（多个参数） public static void 方法名(数据类型 变量名1，数据类型 变量名2，……){……}
* 注意：方法定义时，参数中的数据类型与变量名都不能缺少，缺少任意一个程序将会报错，多个参数之间使用逗号隔开。

## 7.2 带参数方法调用

* 格式：方法名(参数)；
* 格式（单个参数）方法名(变量名/常量值)；
* 格式（多个参数）方法名(变量名1/常量值1，变量名2/常量值2)；
* 注意：方法定义时，参数中的数量与类型必须与方法定义中的设置相匹配，否则程序将会报错

## 6.3.3 形参和实参

* 形参：方法定义中的参数，等同于变量定义格式，例如int number
* 实参：方法调用中的函数，等同于使用变量或常量，例如：isEven Number(10);

# 8带返回值方法的定义和调用

## 8.1.带返回值方法定义

* 格式： public static 数据类型 方法名(参数){

return 数据

}

* 范例：public static boolean isEvenNumber(int number){

return true;

}

* 注意：方法定义时，return后面的返回值与方法定义上的数据类型要匹配，否则程序将会报错

## 8.2.带返回值方法调用

* 格式1：方法名(参数)；
* 范例：isEvenNumber(5)；
* 格式2：数据类型 变量名 = 方法名(参数)；
* 范例：boolean flag = isEvenNumber(5)；
* 注意：方法的返回值通常会使用变量接收，否则该返回值将无意义

# 9.方法的注意事项

* 方法不能嵌套定义
* Void表示无返回值，可以省略return，也可单独书写return，后面不加数据。
* 方法通用格式：

public static 返回值类型 方法名（参数）{

方法体；

return 数据；

}

* 定义方法时，要做到两个明确：

明确返回函数值：主要是明确方法操作完毕之后是否有数据返回，如果没有，写void；如果有，写对应数据类型。

明确参数：主要是明确参数的类型和数量。

* 调用方法时

void类型的方法，直接调用即可

非void类型的方法，推荐用变量接收调用

# 10.方法重载

## 10.1方法重载概述：指同一个类中定义的多个方法之间的关系，满足下列条件的多个方法相互构成重载

* 多个方法在同一个类中
* 多个方法具有相同的方法名
* 多个方法的参数不相同，类型不同或者数量不同

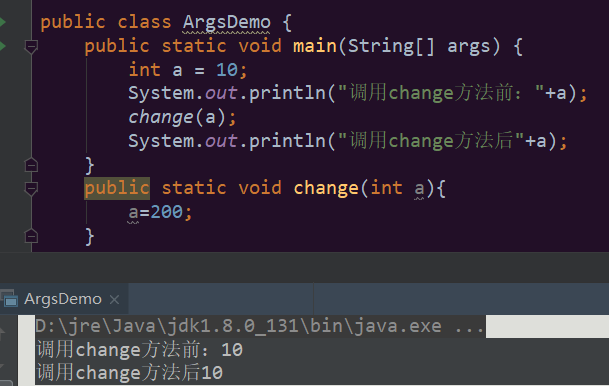
## 10.2方法重载特点

* 重载仅对应方法的定义，与方法的调用无关，调用方式参照标准格式
* 重载仅针对同一个类中方法的名称与参数进行识别，与返回值无关，换句话说不能通过返回值来判断两个方法是否相互构成重载。

# 11.方法的参数传递

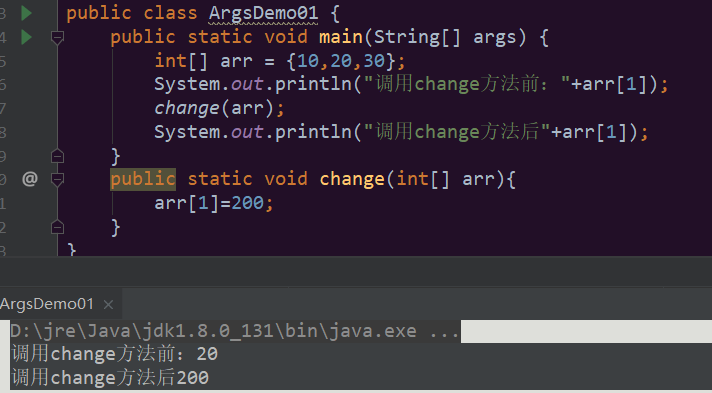
## 11.1方法参数传递（基本类型）

* 对于基本数据类型的参数，形式参数的改变，不影响实际参数的值。



## 11.2方法参数传递（引用类型）

* 对于引用数据类型的参数，形式参数的改变，影响实际参数的值



# 12.Debug

## 12.1Debug操作流程

* Debug：是供程序员使用的程序调试工具，它可以用于查看程序的执行流程，也可用于追踪程序执行过程来调试程序。

Debug调试：又称断点调试，断点其实是一个标记，告诉我们从哪里开始查看。

1. 如何加断点：选择要设置断点的代码行，在行号的区域的后面单击鼠标左键。
2. 如何运行加了断点的程序：在代码区域右键Debug执行。
3. 看哪里：看Debugger窗口和Console窗口。
4. 点哪里：点step into(F7)这个箭头。点stop结束。
5. 如何删除断点：选择要删除的断点，单击鼠标左键即可。

* 注意事项：如果数据来自键盘输入，一定要记得输入数据，不然就不能继续往下查看了。

类和对象

# 类和对象

## 1.1什么是对象

万物皆对象，客观存在的事务皆为对象。

## 1.2什么是类

* 类是对现实生活中一类具有共同属性和行为的事务的抽象。
* 类的特点：

1. 类是对象的数据类型
2. 类是具有相同属性和行为的一组对象的集合

## 1.3什么是对象的属性

* 属性：对象具有的各种特征，每个对象的每个属性都有特定的值。

## 1.4什么是对象的行为

* 行为：对象能够执行的行为。

## 1.5类和对象的关系

* 类是对现实生活中一类具有共同属性和行为的事务的抽象。
* 对象是能够看得到摸得着的真是存在的实体
* 类是对象的抽象。对象是类的实体。

## 1.6类的定义

* 类的重要性：是Java程序的基本组成单位。
* 类的组成：属性和行为

属性：在类中通过成员变量来体现（类中方法外的变量）

行为：在类中通过成员方法来体现（和前面的方法相比去掉static关键字即可）

* 类的定义步骤：

1. 定义类
2. 编写类的成员变量
3. 编写类的成员方法
4. public class 类名{

//成员变量

变量的数据类型 变量；

……

//成员方法

方法1；

方法2；

}

## 1.7对象的使用

* 创建对象：类名 对象名 = new 类名();
* 使用对象：

1. 使用成员变量：对象名.变量名 例如：p.brand
2. 使用成员方法：对象模.方法名 例如：p.call()

# 2.成员变量和局部变量

|  |  |  |
| --- | --- | --- |
| 区别 | 成员变量 | 局部变量 |
| 类中位置不同 | 类中方法外 | 方法内或方法声明上 |
| 内存中的位置不同 | 堆内存 | 栈内存 |
| 生命周期不同 | 随着对象的存在而存在，随着对象的消失而消失 | 随着方法的调用而存在，随着方法的调用完毕而消失 |
| 初始化值不同 | 有默认初始值 | 没有默认初始值，必须先定义，赋值才能使用 |

# 封装

## 3.1 private 关键字

* 是一个权限修饰符，可以修饰成员（成员变量和成员方法）
* 作用是保护成员不被别的类使用，被private修饰的成员只在本类中才能访问
* 针对private修饰的成员变量，如果需要被其它类使用，提供相应的操作
* 提供“get变量名（）“方法，用于获取成员变量的值，方法用public修饰
* 提供“set变量名（）“方法，用于设置成员变量的值，方法用public修饰

## 3.2 private 关键字的使用

一个标准类的编写：

* 把成员变量用private修饰
* 提供对应的getXxxx()/setXxx()方法

## 3.3 this关键字的使用

1. this修饰的变量用于指代成员变量

* 方法的形参如果与成员变量同名，不带this修饰的变量指的是形参，而不是成员变量
* 方法的形参如果没有与成员变量同名，不带this修饰的变量指的是成员变量

1. 什么时候用this？解决局部变量隐藏成员变量时
2. this：代表所在类的对象引用

* 方法被那个对象调用，this就代表哪个对象

## 3.4 封装

1. 封装概述

* 是面向对象三大特征之一（封装，继承，多态）
* 是面向对象编程语言对客观世界的模拟，客观世界里成员变量都是隐藏在对象内部的，外界是无法接触的。