文档类别：JAVA进阶课程文档

**大数据课程**

**JAVA进阶**

**任课教师：许兆磊**

**联系方式：15201392594**

**QQ： 1004929800**

**微信：15201392594**

目录

[文档类别：JAVA进阶课程文档 1](#_Toc5257179)

[目录 2](#_Toc5257180)

[一、 JAVA基础知识回顾 4](#_Toc5257181)

[1 JAVA基础知识 4](#_Toc5257182)

[1.1 关键字 4](#_Toc5257183)

[1.1.1 具体关键字及其含义 4](#_Toc5257184)

[1.1.2 按照分类分组 5](#_Toc5257185)

[1.2 标识符>驼峰命名法 12](#_Toc5257186)

[1.2.1 驼峰定义 12](#_Toc5257187)

[1.2.2 类名定义采用大驼峰命名法 12](#_Toc5257188)

[1.2.3 变量名定义采用小驼峰命名法 13](#_Toc5257189)

[1.2.4 优秀攻城师的习惯： 14](#_Toc5257190)

[1.3 注释信息 14](#_Toc5257191)

[1.3.1 单行注释 14](#_Toc5257192)

[1.3.2 多行注释 14](#_Toc5257193)

[1.3.3 文档注释(java特有的，通javadoc 可以提取) 14](#_Toc5257194)

[1.4 变量和常量 15](#_Toc5257195)

[1.4.1 变量定义和使用规则： 15](#_Toc5257196)

[1.4.2 JAVA常量定义和使用规则 15](#_Toc5257197)

[1.5 JAVA数据类型 16](#_Toc5257198)

[1.5.1 数据类型分类 16](#_Toc5257199)

[1.5.2 常用的基本数据类型以及字节长度： 16](#_Toc5257200)

[1.5.3 数值大小限制(超过则报错-1)： 17](#_Toc5257201)

[1.6 类型转换 17](#_Toc5257202)

[1.6.1 隐士转换>自动类型转换 17](#_Toc5257203)

[1.6.2 显示转换>强制类型转换 17](#_Toc5257204)

[1.7 运算符 18](#_Toc5257205)

[1.7.1 算数运算符 18](#_Toc5257206)

[1.7.2 关系运算符 19](#_Toc5257207)

[1.7.3 逻辑运算符 20](#_Toc5257208)

[1.7.4 赋值运算符 22](#_Toc5257209)

[1.7.5 位运算符 22](#_Toc5257210)

[1.7.6 条件（三目）运算符 24](#_Toc5257211)

[1.7.7 运算符的优先级 25](#_Toc5257212)

[1.8 语句 26](#_Toc5257213)

[1.8.1 if语句 26](#_Toc5257214)

[1.8.2 switch语句 28](#_Toc5257215)

[1.8.3 while语句 29](#_Toc5257216)

[1.8.4 do while 语句 30](#_Toc5257217)

[1.8.5 for循环 30](#_Toc5257218)

[1.8.6 break-continue循环跳转语句 32](#_Toc5257219)

[1.8.7 多重循环 33](#_Toc5257220)

[1.9 函数 34](#_Toc5257221)

[1.9.1 方法（函数）的格式 34](#_Toc5257222)

[1.9.2 可以根据有无返回值有无形式参数分为四种 34](#_Toc5257223)

[1.9.3 值传递和引用传递 35](#_Toc5257224)

[1.10 数组 38](#_Toc5257225)

[1.10.1 Java数组的定义和使用 38](#_Toc5257226)

[1.11 集合 46](#_Toc5257227)

[1.11.1 集合对象的由来： 46](#_Toc5257228)

[1.11.2 集合的数据结构： 46](#_Toc5257229)

[1.11.3 Collection的常见方法： 46](#_Toc5257230)

[1.11.4 56](#_Toc5257231)

[1.12 面向对象 56](#_Toc5257232)

[1.12.1 了解面向对象的三个特征 56](#_Toc5257233)

[1.12.2 掌握对象的创建和使用方式 70](#_Toc5257234)

[1.12.3 掌握类的封装方式 75](#_Toc5257235)

[1.12.4 掌握方法的使用方式 75](#_Toc5257236)

[1.12.5 掌握this和static关键字的使用 77](#_Toc5257237)

[二、 JAVA进阶 77](#_Toc5257238)

[2.1 基于文档注释的JAVADOC接口文档 77](#_Toc5257239)

[2.1.1 Java的文档注释 77](#_Toc5257240)

[2.1.2 Javadoc文档的使用 77](#_Toc5257241)

[2.2 Jar与可执行文件的 77](#_Toc5257242)

[2.3 JXM管理在框架 77](#_Toc5257243)

[2.4 SVN版本控制 77](#_Toc5257244)

[三、 JAVAWEB编程 77](#_Toc5257245)

[3.1 Servlet的具体用法 77](#_Toc5257246)

[3.1.1 定义 77](#_Toc5257247)

[3.1.2 Demo 77](#_Toc5257248)

1. JAVA基础知识回顾
2. JAVA基础知识
   1. 关键字
      1. 具体关键字及其含义

|  |  |
| --- | --- |
| **关键字** | **含义** |
| abstract | 表明类或者成员方法具有抽象属性 |
| assert | 断言，用来进行程序调试 |
| case | 用在[switch](https://www.baidu.com/s?wd=switch&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)语句之中，表示其中的一个分支 |
| catch | 用在异常处理中，用来捕捉异常 |
| char | 基本数据类型之一，字符类型 |
| class | 声明一个类 |
| const | 保留关键字，没有具体含义 |
| continue | 回到一个块的开始处 |
| default | 默认，例如，用在switch语句中，表明一个默认的分支 |
| do | 用在do-while循环结构中 |
| double | 基本数据类型之一，双精度浮点数类型 |
| else | 用在条件语句中，表明当条件不成立时的分支 |
| enum | 枚举 |
| extends | 表明一个类型是另一个类的子类型，这里常见的类型有类和接口 |
| final | 用来说明最终属性，表明一个类不能派生出子类，或者成员方法不能被覆盖，或者成员域的值不能被改变，用来定义常量 |
| finally | 用于处理异常情况，用来声明一个基本肯定会被执行到的语句块 |
| float | 基本数据类型之一，单精度浮点数类型 |
| for | 一种循环结构的引导词 |
| goto | 保留关键字，没有具体含义 |
| if | 条件语句的引导词 |
| implements | 表明一个类实现了给定的接口 |
| import | 表明要访问指定的类或包 |
| package | 包 |
| private | 一种访问控制方式：私用模式 |
| protected | 一种访问控制方式：保护模式 |
| public | 一种访问控制方式：共用模式 |
| return | 从成员方法中返回数据 |
| short | 基本数据类型之一,短整数类型 |
| static | 表明具有静态属性 |
| strictfp | 用来声明FP\_strict（单精度或双精度浮点数）表达式遵循[IEEE 754](https://baike.baidu.com/item/IEEE%20754)算术规范 [1] |
| super | 表明当前对象的父类型的引用或者父类型的构造方法 |
| switch | 分支语句结构的引导词 |
| synchronized | 表明一段代码需要同步执行 |
| this | 指向当前实例对象的引用 |
| throw | 抛出一个异常 |
| throws | 声明在当前定义的成员方法中所有需要抛出的异常 |
| transient | 声明不用序列化的成员域 |
| try | 尝试一个可能抛出异常的程序块 |
| void | 声明当前成员方法没有返回值 |
| volatile | 表明两个或者多个变量必须同步地发生变化 |
| while | 用在循环结构中 |

* + 1. 按照分类分组
       1. 数据类型相关(10)

|  |  |  |  |
| --- | --- | --- | --- |
| 关键字 | 类型 | 占字节数 | 备注 |
| boolean | 布尔数据类型 | 1 | 有true和false两个值 |
| int | 整型数据类型 | 4 |  |
| long | 长整型数据类型 | 8 |  |
| short | 短整型数据类型 | 2 |  |
| byte | 字节数据类型 | 1 | 大小范围为-27—27-1 |
| float | 浮点型（实数型） | 4 |  |
| double | 双精度型数据类型 | 8 |  |

* + - 1. 流程控制相关(13)

|  |
| --- |
| **if：** 表示条件判断，一般用法if（关系表达式），后跟else或{……} |
|  |
| else： 条件转折，如if （关系表达式）{语句块1}else{语句块2}，如果关系表达式的值为true，则执行语句块1，否则执行语句块2. |
| do……while……：do和while一般一起使用，用于表示循环语句。do{……}while（关系表达式）……；当关系表达式的值为true是继续循环。 |
| **for：**用于表示循环，for循环是最常使用的循环，格式for（表达式a; 表达式b; 表达式c）括号里面的书通常用于控制循环的次数，一般会用一个int类型的变量类计数，如（int i=0; i<10; i++）表达式a用于流程控制的开始值，表达式b表示循环终止条件，表达式c用于计数。 |
| **switch（**条件a）case……：switch和case合起来用于表示条件分支流程。如：  while(int c) {  case 1: {语句块1}  case 2: {语句块2}  ……?????? ……  case n: {语句块n}  default：exit(0);  }  如果c为1，则执行语句块1；如果c为2，则执行语句块2；以此类推，如果c为n，则执行语句块n。default表示除case以外出现的情况。 |
| **default：**在switch……case……分支语句可知，default是在所有case条件下都不成立时使用。用于包表示s“rc”文件夹下的当前包；如果用于类，表示只可被本文件内的其它类访问。 |
| **break：**用于结束本层循环，或跳出某层循环。 |
| **continue：**用于跳出本次循环，而break跳出本层循环。Break和continue可以实现类似于C\C++中goto语句的用法：  label0:  {  　for (int k = 0; k < 10; k++) {  　..........  　label1:  　for (int j = 0; j < 10; j++) {  　　................  　　break label0;　　//跳转至label0  　}  　...........  　label2:  　for (int kk = 0; kk < 10; kk++) {  　　..............  　　break label0;　　//跳至label2  　}  } |
| **return:**返回一个值，通常用于函数中，返回一个具有特定类型的值。如:  public int fuction() { 　　int a;  　　…… 　　return a;//返回int型的值 } |
| **try……catch……finally……：**用于异常处理，使用形式如：  try{  　……  }catch(类型b){  　……  }finally{  　……  }  try{……}中放置可能会发生异常的的语句块，catch(){……}用于抓住异常，{}定义当出现异常时的处理方法。finally{……}表示不管异常是否发生，都得进行finally{}中的处理 |

* + - 1. 修饰符相关(12)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 范围 | 本类 | 同软件包 | 不同包中子类 | 不同包且无继承 | 作用(含义) |
| **public** | 可以 | 可以 | 可以 | 可以 | 公有的 |
| **protect** | 可以 | 可以 | 可以 |  | 受保护的 |
| **default** | 可以 | 可以 |  |  | 当前的 |
| **private** | 可以 |  |  |  | 私有的 |

如：

|  |
| --- |
| 一个".java"后缀名的文件里面只能一个public型的class类。如c13.swing包下有HelloWorld.java 文件，该文件下有如下代码：  package c13.swing;  public class HelloWorld {  public void show() {  System.out.println("HelloWorld!");  }  } |
| 在default包下有如下代码： import c13.swing.\*;//引入所需要的包里的所有类 public class Test { public static void main(String args[]) { HelloWorld h = new HelloWorld(); h.show(); } } 此时运行正常，结果：HelloWorld! 如果把public class HelloWorld改成privateclass HelloWorld则会运行错误，因为private是私有的不能被其他包内的对象所引用。 |
| **final：(**类似于C++中的const)，用法如下： a，final的变量的值不能被改变；1），final的成员变量； 2）final的局部变量； b，final的方法不能被重写； c，final的类不能被继承。 |
| **void：**空类型，用于函数前表示空返回值。  **static：**可用于定义静态成员变量或方法 |
| **Strictfp（了解内容）**：意思是FP-strict，也就是说精确浮点的意思。在Java虚拟机进行浮点运算时，如果没有指定strictfp关键字时，Java的编译器以及运行环境在对浮点运算的表达式是采取一种近似于我行我素的行为来完成这些操作，以致于得到的结果往往无法令你满意。而一旦使用了strictfp来声明一个类、接口或者方法时，那么所声明的范围内Java的编译器以及运行环境会完全依照浮点规范IEEE-754来执行。因此如果你想让你的浮点运算更加精确，而且不会因为不同的硬件平台所执行的结果不一致的话，那就请用关键字strictfp。  你可以将一个类、接口以及方法声明为strictfp，但是不允许对接口中的方法以及构造函数声明strictfp关键字，例如下面的代码：  1. 合法的使用关键字strictfp  strictfp interface A {}  public strictfp class FpDemo1 {  strictfp void f() {}  }  2. 错误的使用方法  interface A {  strictfp void f();  }  public class FpDemo2 {  strictfp FpDemo2() {}  }  一旦使用了关键字strictfp来声明某个类、接口或者方法时，那么在这个关键字所声明的范围内所有浮点运算都是精确的，符合IEEE-754规范的。例如一个类被声明为strictfp，那么该类中所有的方法都是strictfp的。 |
| **abstract：抽象的意思，**可以用定义抽象类，抽象函数。抽象类的用法规则如下：  a，用abstract关键字来修饰一个类时，这个类叫做抽象类；用abstract来修饰一个方法时，这个方法叫做抽象方法；  b，含有抽象方法的类必须被声明为抽象类，抽象类必须被继承，抽象方法必须被重写；  c，抽象类不能被实例化；  d，抽象方法只需声明，而不需实现。 |
| **Transient（了解）:** Java语言的关键字，用来表示一个域不是该对象串行化的一部分。当一个对象被串行化的时候，transient型变量的值不包括在串行化的表示中，然而非transient型的变量是被包括进去的。  使用对象：字段  介绍：字段不是对象持久的一部分，不应该字段和对象一起串起。 |
| **Synchronized**(很重要)：锁定当前对象，在执行当前这个对象时不应许其他线程打断插入。  **悲观锁：**  **乐观锁：** |
| **volatile**(很重要)**：volatile变量一个时间只有一个线程可以访问。 保证线程安全。**  **可以被写入 volatile 变量的这些有效值独立于任何程序的状态，包括变量的当前状态。**  **所以，Volatile 变量是一种非常简单但同时又非常脆弱的同步机制，它在某些情况下将提供优于锁的性能和伸缩性** |
| **native：**是用作java 和其他语言（如c++）进行协作时用的也就是native 后的函数的实现不是用java写的， native的意思就是通知操作系统，这个函数你必须给我实现，因为我要使用。所以native关键字的函数都是操作系统实现的， java只能调用。 |

* + - 1. 动作相关（10）

|  |
| --- |
| **package**：打包，把java程序写在一个包中。 |
| **import**：引入报名，用法：  a，写全包名：jsxt.java140.Cat c = new com.bjsxt.java140.Cat  a，引入一个包中的具体某个对象：xt.java140.Cat;  b，引入一个包中的所有象：import com.bjsxt.java140.Cat.\*; |
| **throw：**抛出用户自己定义的异常 |
| **throws：**抛出系统异常 |
| **extends**：继承（类继承类，或接口继承接口） |
| **implements**：实现，通过类来实现接口 |
| **this**：是指向该对象自身的标示符 |
| **super：**调用基类的构造函数 |
| **instanceof：**判断某个对象是否为自己所要的对象 |
| **new：**用于新建一个对象，即类的实例化 |

* + - 1. 其他（5）

|  |
| --- |
| **true**：表示boolean类型的真值 |
| **false**：表示boolean类型的假值，即不成立的情况 |
| **null**：表示空值，如果是对象，则表示此对象为空，如果是数据类型，在表示相应数据类型的默认值，如int的默认值为0。 |
| **goto**：C\C++中实现无条件转向语句，为了结构化程序设计java中现在一般不用。 |
| **const**：和const一样，在Java中,const是作为保留字以备扩充。可以用final替换const，一般C++中用const，java中用final |

* 1. 标识符>驼峰命名法
     1. 驼峰定义

骆驼式命名法就是当变量名或函式名是由一个或多个单词连结在一起，而构成的唯一识别字时，第一个单词以小写字母开始；第二个单词的首字母大写或每一个单词的首字母都采用大写字母，例如：myFirstName、myLastName，这样的变量名看上去就像骆驼峰一样此起彼伏，故得名。

骆驼式命名法（Camel-Case）一词来自 Perl 语言中普遍使用的大小写混合格式，而 Larry Wall等人所著的畅销书《Programming Perl》（O'Reilly 出版）的封面图片正是一匹骆驼。

骆驼式命名法的命名规则可视为一种惯例，并无绝对与强制，为的是增加识别和可读性。

* + - 1. 小驼峰法

变量一般用小驼峰法标识。驼峰法的意思是：除第一个单词之外，其他单词首字母大写。譬如

int myStudentCount;

变量myStudentCount第一个单词是全部小写，后面的单词首字母大写。

* + - 1. 大驼峰法

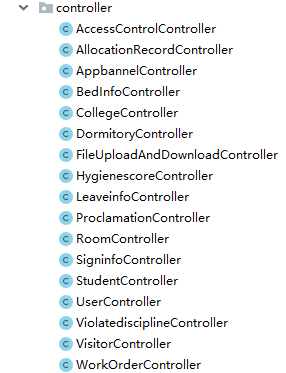
相比小驼峰法，大驼峰法（即帕斯卡命名法）把第一个单词的首字母也大写了。常用于[类名](http://baike.baidu.com/view/738143.htm)，函数名，属性，命名空间。譬如

public class DatabaseUser;

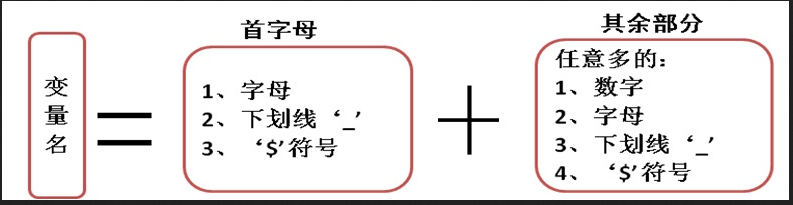
* + 1. 类名定义采用大驼峰命名法

比如：

public class DataBaseUser;

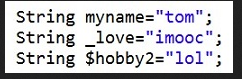


* + 1. 变量名定义采用小驼峰命名法



举例说明：

正确定义变量



错误的定义变量



* + 1. 优秀攻城师的习惯：

1. 变量名由多单词组成时，第一个单词的首字母小写，其后单词的首字母大写，俗称骆驼式命名法（也称驼峰命名法），如 myAge
2. 类名有多个单子组成时，所有字母的首字母都大写（约定俗成）(也称为大驼峰)
3. 变量和类命名时，尽量简短且能清楚的表达变量的作用，做到见名知意。如：定义变量名 stuName 保存“学生姓名”信息 。Dept.class
   1. 注释信息
      1. 单行注释

|  |
| --- |
| //打印要输出的内容  System.out.println("Hello World!~"); |

* + 1. 多行注释

|  |
| --- |
| /\*  \* System.out.println("Hello World!~");  \*/ |

* + 1. 文档注释(java特有的，通javadoc 可以提取)

|  |
| --- |
| /\*\*  \* 需求： 在控制台中打印"HelloWorld"内容。  \*/ |

* 1. 变量和常量
     1. 变量定义和使用规则：

1、定义：

在JAVA中我们通过三个元素来描述变量：变量类型，变量名以及变量值。

**S**tring love="imooc";

变量类型 变量名 值（其中String具有不可变性，重新赋值后会生成新的String对象，love变量名这实际是指向对象地址的引用，"imooc"为具体的值）。

love="I love imooc"; 变量重新赋值，重新指向了一个新的对象，对象值为"i love imooc“。

2、使用规则：

1、Java 中的变量需要先声明后使用  
2、变量使用时，可以声明变量的同时进行初始化,也可以先声明后赋值  
String love="imooc";或者String love;love="I love imooc";  
3、变量中每次只能赋一个值，但可以修改多次  
4、main 方法中定义的变量必须先赋值，然后才能输出  
5、虽然语法中没有提示错误，但在实际开发中，变量名不建议使用中文，容易产生安全隐患，譬如后期跨平台操作时出现乱码等等

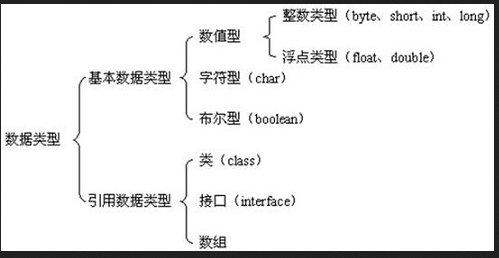
* + 1. JAVA常量定义和使用规则

所谓常量可以理解成一种特殊的变量，它的值被设定后，在程序运行过程中不允许被改变。  
final 常量名=值;  
final double PI=3.14;

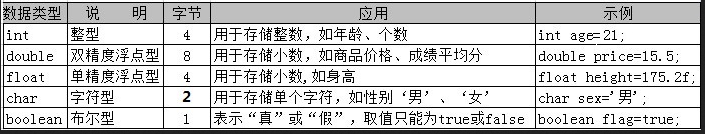
final String LOVE="imooc";  
常量名一般使用大写字符。  
程序中使用常量可以提高代码的可维护性。例如，在项目开发时，我们需要指定用户的性别，此时可以定义一个常量 SEX，赋值为 "男"，在需要指定用户性别的地方直接调用此常量即可，避免了由于用户的不规范赋值导致程序出错的情况。

* 1. JAVA数据类型
     1. 数据类型分类

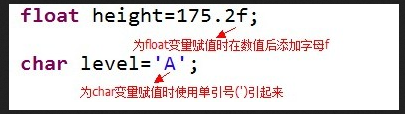
Java 语言是一种强类型语言。通俗点说，在 Java 中存储的数据都是有类型的，而且必须在编译时就确定其类型。 Java 中有两类数据类型：在 Java 的领域里，基本数据类型变量存的是数据本身，而引用类型变量存的是保存数据的空间地址。说白了，基本数据类型变量里存储的是直接放在抽屉里的东西，而引用数据类型变量里存储的是这个抽屉的钥匙，钥匙和抽屉一一对应。



* + 1. 常用的基本数据类型以及字节长度：



注意：



String 是一种常见的引用数据类型，用来表示字符串。在程序开发中，很多操作都要使用字符串来完成，例如系统中的用户名、密码、电子邮箱等。

* + 1. 数值大小限制(超过则报错-1)：

byte 127(2^7-1) -128(-2^7)  
short 32767(2^15-1) -32768(-2^15)  
int 2147483647(2^31-1) -2147483648(-2^31)  
long 9223372036854775807(2^63-1) -9223372036854775808(2^63)

默认值0L long a=100000L;  
float 默认值是0.0f float f1=253.4f;  
double 默认值是0.0d double d1=123.4;  
boolean 默认值是false;

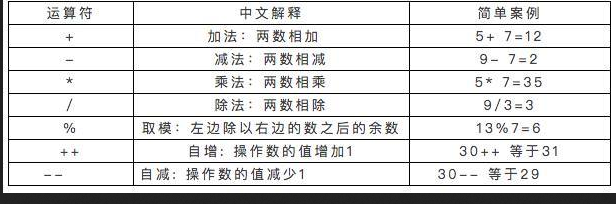
* 1. 类型转换
     1. 隐士转换>自动类型转换

int转换double类型时可以直接转换，这种叫做自动类型转换，当然自动类型转换是需要满足特定条件的：

1. 目标类型能与源类型兼容，如 double 型兼容 int 型；
2. 目标类型大于源类型，如 double 类型长度为 8 字节， int 类型为 4 字节，因此 double 类型的变量里直接可以存放 int 类型的数据，但反过来就不可以了。
   * 1. 显示转换>强制类型转换

强制类型转换容易造成数据丢失，所以在转换时需要多加注意  
强制类型转换的语法是 （数据类型）数值  
double avg1=75.8;  
int avg2=(int)avg1;

* 1. 运算符
     1. 算数运算符
        1. 算数运算符包括的种类：



* + - 1. 个别运算符补充介绍

**自增运算符(++)和自减运算符(--),其写在变量前和变量后有不同的效果：**

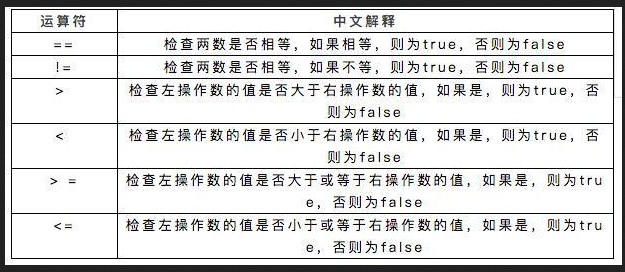
**1）如果写在变量前表示在使用这个变量之前加1或减1 ++a**

**2）如果写在变量后表示这个变量使用完之后再加1或减1 a++**

**测试用例如下：**

|  |
| --- |
| **package com.hbdfxy;**  **/\*\***  **\* 实现步骤：**  **\* 1、定义两个变量，并分赋初始值**  **\* 2、变量分别进行 a++ 和 ++a 运算 并测试最终输出的结果**  **\***  **\*/**  **public class ConvertAndMath {**  **public static void main(String[] args) {**    **int a=10 ,b=10;**    **int temp\_a=a++;**  **int temp\_b=++b;**    **System.out.println(a); //11**  **System.out.println(temp\_a); //10**  **System.out.println(b); //11**  **System.out.println(temp\_b); //11**      **}**  **}** |

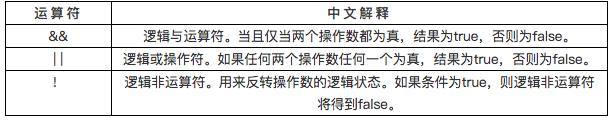
* + 1. 关系运算符
       1. 运算符及其种类



* + - 1. 测试用例如下：

|  |
| --- |
| package com.hbdfxy.lesson;  /\*\*  \* 实现步骤：  \* 1、定义两个变量，并分赋初始值  \* 2、利用逻辑运算判断两个值的大小  \*  \*/  public class Arithmetic {  public static void main(String[] args) {    int a=9 ,b=10;  System.out.println(a>b);//false  System.out.println(a>=b);//false  System.out.println(a==b);//false  System.out.println(a<b);//true  System.out.println(a<=b);//true  System.out.println(a!=b);//true    }  } |

* + 1. 逻辑运算符
       1. 逻辑运算符包含的种类



补充“短路逻辑”介绍

1）对于”&&”，当第一个操作数为false时，将不会判断第二个操作数，因为此时无论第二个操作数为何，最后的运算结果一定是false；

例如：（4>6）&&（7==7）；//输出结果为false，且第二个括号内的操作不会被判断

2）对于”||”，当第一个操作数为true时，将不会判断第二个操作数，因为此时无论第二个操作数为何，最后的运算结果一定是true。

例如：（8>6）||（6==7）；//输出结果为true，且第二个括号内的操作不会被判断

* + - 1. 测试案例：

|  |
| --- |
| package com.hbdfxy.lesson;  /\*\*  \* 实现步骤：  \* 1、定义两个变量，并分赋初始值  \* 2、利用逻辑运算判断两个值的大小  \*  \*/  public class Logic {  public static void main(String[] args) {  int a=9 ,b=10;  System.out.println(a>b&&b<a);//false  System.out.println(a<b&&b>a);//true  System.out.println(a>b||b>a);//true  System.out.println(!(a>b||b>a));//true  }  } |

* + 1. 赋值运算符
       1. 逻辑运算符简介：

1）“=”被称为赋值运算符，用于对变量赋值。关于赋值运算符，除了将右边的表达式计算出来复制给左边。

例如：C = A+B；等价于，将A+B的和赋值给C。

（2）在赋值运算符=前加上其他运算符，即为扩展赋值运算符。

例如：加号与赋值运算符的结合A+=B；等价于A= A+B

（3）同理，减号与赋值运算符，乘号与赋值运算符等。

* + 1. 位运算符
       1. 位运算适用的类型包括：

int，long，char，byte等类型

* + - 1. 位运算符的计算过程：

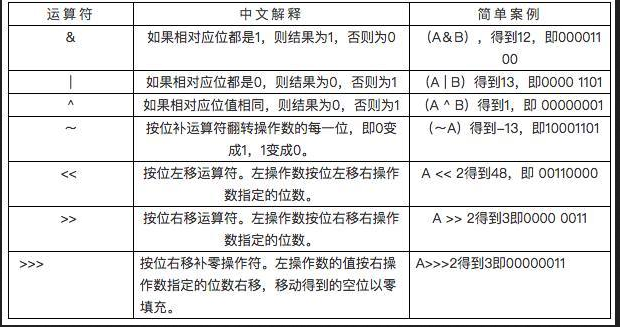
第一步：将数字转换为二进制

第二步：按照不同位运算符的要求，进行计算（见下边表格）

用例：

|  |
| --- |
| 例子：A = 12，B = 13；  A = 00001100  B = 00001101  A&B = 1100 = 12 |

* + - 1. 位运算符包括的种类：



测试用例如下：

|  |
| --- |
| package com.hbdfxy.lesson;  /\*\*  \* 实现步骤：  \* 1、定义两个变量，并分赋初始值  \* 2、利用逻辑运算判断两个值的大小  \*  \*/  public class BitOperate {  public static void main(String[] args) {    int A=12 ,B=13;  System.out.println(A&B);//12  System.out.println(A | B);//13  System.out.println(A^B);//1  System.out.println(~A);//-13  System.out.println(A<<2);//48  System.out.println(A>>2);//3  System.out.println(A>>>2);//3  }  } |

* + 1. 条件（三目）运算符
       1. 条件运算符又称三目运算符

boolean表达式？表达式1：表达式2

* + - 1. 计算过程：

先计算boolean表达式；

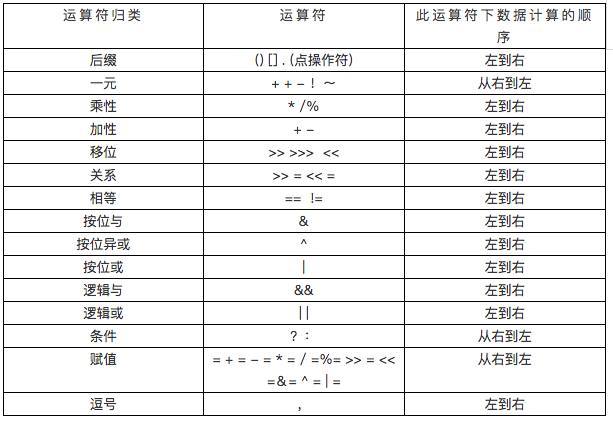
如果boolean表达式的值为true，整个表达式的值为表达式1的值；

如果为false，整个表达式的值为表达式2的值。

|  |
| --- |
| package com.hbdfxy.lesson;  /\*\*  \* 实现步骤：  \* 1、定义两个变量，并分赋初始值  \* 2、利用三目运算符,根据不同的结果分别取汁  \*  \*/  public class ThirdLogic {  public static void main(String[] args) {    int a = 5;  int b = 6;  int score;  score = a>b? 15:16;  System.out.println(score);  }  } |

* + 1. 运算符的优先级
       1. 运算符等级含义

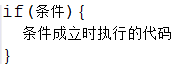
多个运算符出现在一个运算表达式中时，需要按照优先级进行计算，现按照优先级从高到底罗列如下：



number = 5+ 4\*6；按照优先级是，先进行乘法，然后再进行加法； 24+5 = 29 而不是9\*6 = 54

* 1. 语句
     1. if语句
        1. if条件语句

语法：

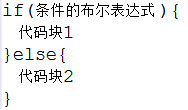


注意哦：如果 if 条件成立时的执行语句只有一条，是可以省略大括号滴！但如果执行语句有多条，那么大括号就是不可或缺的喽~~

* + - 1. if...else语句

if...else 语句的操作比 if 语句多了一步: 当条件成立时，则执行 if 部分的代码块； 条件不成立时，则进入 else 部分。

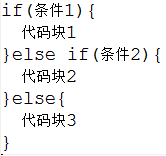
语法：



* + - 1. 多重 if 语句

在条件 1 不满足的情况下，才会进行条件 2 的判断；当前面的条件均不成立时，才会执行 else 块内的代码。

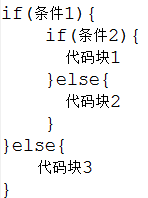
语法：



* + - 1. 嵌套if语句

只有当外层 if 的条件成立时，才会判断内层 if 的条件。例如，活动计划的安排，如果今天是工作日，则去上班，如果今天是周末，则外出游玩；同时，如果周末天气晴朗，则去室外游乐场游玩，否则去室内游乐场游玩。

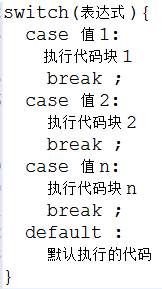
语法：



* + 1. switch语句
       1. 具体的用法

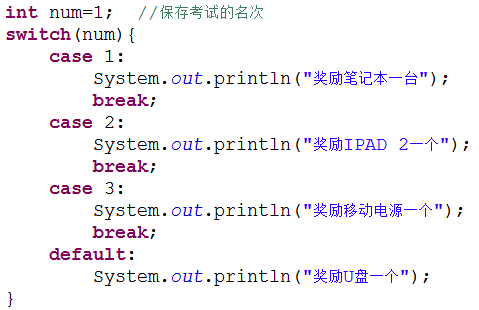
当需要对选项进行等值判断时，使用 switch 语句更加简洁明了。例如：根据考试的名次，给予前 4 名不同的奖品。第一名，奖励笔记本一台；第二名，奖励 IPAD 2 一个；第三名，奖励移动电源一个；最后一名奖励 U 盘一个。

语法：



执行过程：当 switch 后表达式的值和 case 语句后的值相同时，从该位置开始向下执行，直到遇到 break 语句或者 switch 语句块结束；如果没有匹配的 case 语句则执行 default 块的代码。

如：



* + - 1. 使用注意点：

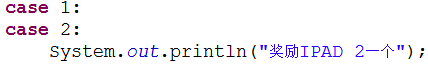
特别要注意的地方：

1、 switch 后面小括号中表达式的值必须是整型或字符型

2、 case 后面的值可以是常量数值，如 1、2；也可以是一个常量表达式，如 2+2 ；但不能是变量或带有变量的表达式，如 a \* 2

3、 case 匹配后，执行匹配块里的程序代码，如果没有遇见 break 会继续执行下一个的 case 块的内容，直到遇到 break 语句或者 switch 语句块结束

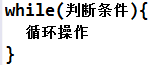
4、 可以把功能相同的 case 语句合并起来，如



5、 default 块可以出现在任意位置，也可以省略

* + 1. while语句
       1. 语法

语法：



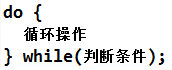
执行过程：

< 1 >、 判断 while 后面的条件是否成立( true / false )

< 2 >、 当条件成立时，执行循环内的操作代码 ，然后重复执行< 1 >、< 2 >， 直到循环条件不成立为止

特点：先判断，后执行

* + 1. do while 语句
       1. 语法：



执行过程：

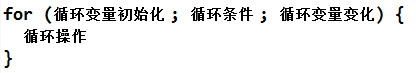
<1>、 先执行一遍循环操作，然后判断循环条件是否成立

<2>、 如果条件成立，继续执行< 1 > 、< 2 >，直到循环条件不成立为止

特点： 先执行，后判断

由此可见，do...while 语句保证循环至少被执行一次！

* + 1. for循环
       1. 语法



执行过程：

<1>、 执行循环变量初始化部分，设置循环的初始状态，此部分在整个循环中只执行一次

<2>、 进行循环条件的判断，如果条件为 true ，则执行循环体内代码；如果为 false ，则直接退出循环

<3>、 执行循环变量变化部分，改变循环变量的值，以便进行下一次条件判断

<4>、 依次重新执行< 2 >、< 3 >、< 4 >，直到退出循环

特点：相比 while 和 do...while 语句结构更加简洁易读

需要留心的几个小细节：

1、 for 关键字后面括号中的三个表达式必须用 “;” 隔开，三个表达式都可以省略，但 “;” 不能省略。

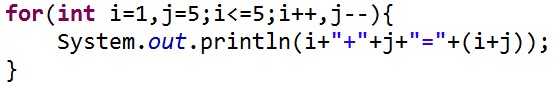
    a. 省略“循环变量初始化”，可以在 for 语句之前由赋值语句进行变量初始化操作，

    b. 省略“循环条件”，可能会造成循环将一直执行下去，也就是我们常说的“死循环”现象，

在编程过程中要避免“死循环”的出现，因此，对于上面的代码可以在循环体中使用 break 强制跳出循环（关于 break 的用法会在后面介绍）。

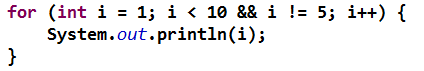
    c. 省略“循环变量变化”，可以在循环体中进行循环变量的变化

2、 for 循环变量初始化和循环变量变化部分，可以是使用 “,” 同时初始化或改变多个循环变量的值，如：



代码中，初始化变量部分同时对两个变量 i 和 j 赋初值，循环变量变化部分也同时对两个变量进行变化

3、 循环条件部分可以使用逻辑运算符组合的表达式，表示复杂判断条件，但一定注意运算的优先级，如：



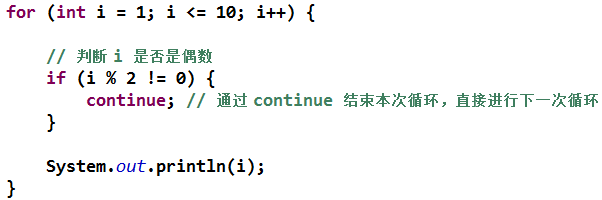
代码中，必须同时满足变量 i 小于 10 ，并且 i 不等于 5 时才会进行循环，输出变量 i 的值。

* + 1. break-continue循环跳转语句
       1. 概述

在 Java 中，我们可以使用 break 语句退出指定的循环，直接执行循环后面的代码。

continue 的作用是跳过循环体中剩余的语句执行下一次循环。

例如，打印 1--10 之间所有的偶数，使用 continue 语句实现代码为：



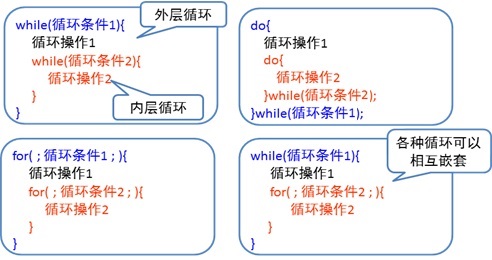
运行结果：

[http://img.mukewang.com/536b50770001d42a00350109.jpg](http://img.mukewang.com/536b50770001d42a00350109.jpg)

* + 1. 多重循环
       1. 概述

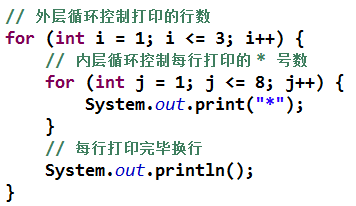
循环体中包含循环语句的结构称为多重循环。三种循环语句可以自身嵌套，也可以相互嵌套，最常见的就是二重循环。在二重循环中，外层循环每执行一次，内层循环要执行一圈。

如下所示：



例如：使用 \* 打印长方形：

http://img.mukewang.com/536c5253000181a900810058.jpg



执行流程：当 i = 1 时，外层循环条件成立，进入内层循环，开始打印第一行内容。此时， j 从 1 开始，循环 8 次，内层循环结束后换行，实现第一行 8 个 \* 的输出。接下来返回外层循环 i 变为 2 ，准备打印下一行，依此类推，直到完成长方形的打印。

* 1. 函数
     1. 方法（函数）的格式

函数的组成是：

|  |
| --- |
| 访问修饰符 返回值 函数名(形式参数)  {  函数内容;  } |

调用函数：

函数名(需要传递给‘形式参数’的数值或者变量)

传递的数据的类型要与形式参数的数据类型一致

* + 1. 可以根据有无返回值有无形式参数分为四种
       1. 无返回值 无参

|  |
| --- |
| void A()  {} |

* + - 1. 有返回值 无参

|  |
| --- |
| int B()//返回值为int类型  {  return 0;  } |

* + - 1. 无返回值 有参

|  |
| --- |
| void C(int m)//接受int类型的参数 并把参数的值赋值给m  {    } |

* + - 1. 有返回值 有参

|  |
| --- |
| int D(int n)  {  } |

* + 1. 值传递和引用传递
       1. 基本类型和引用类型在内存中传递

Java中数据类型分为两大类，基本类型和对象类型。相应的，变量也有两种类型：基本类型和引用类型。

基本类型的变量保存原始值，即它代表的值就是数值本身；

而引用类型的变量保存引用值，"引用值"指向内存空间的地址，代表了某个对象的引用，而不是对象本身，

对象本身存放在这个引用值所表示的地址的位置。

基本类型包括：byte,short,int,long,char,float,double,Boolean,returnAddress，

引用类型包括：类类型，接口类型和数组。

相应的，变量也有两种类型：基本类型和引用类型。

* + - 1. 变量的基本类型和引用类型的区别

基本数据类型在声明时系统就给它分配空间:

 引用则不同，它声明时只给变量分配了引用空间，而不分配数据空间:



看一下下面的初始化过程，注意"引用"也是占用空间的，一个空Object对象的引用大小大概是4byte：



* + - 1. 引用传递和值传递

这里要用实际参数和形式参数的概念来帮助理解，

值传递：

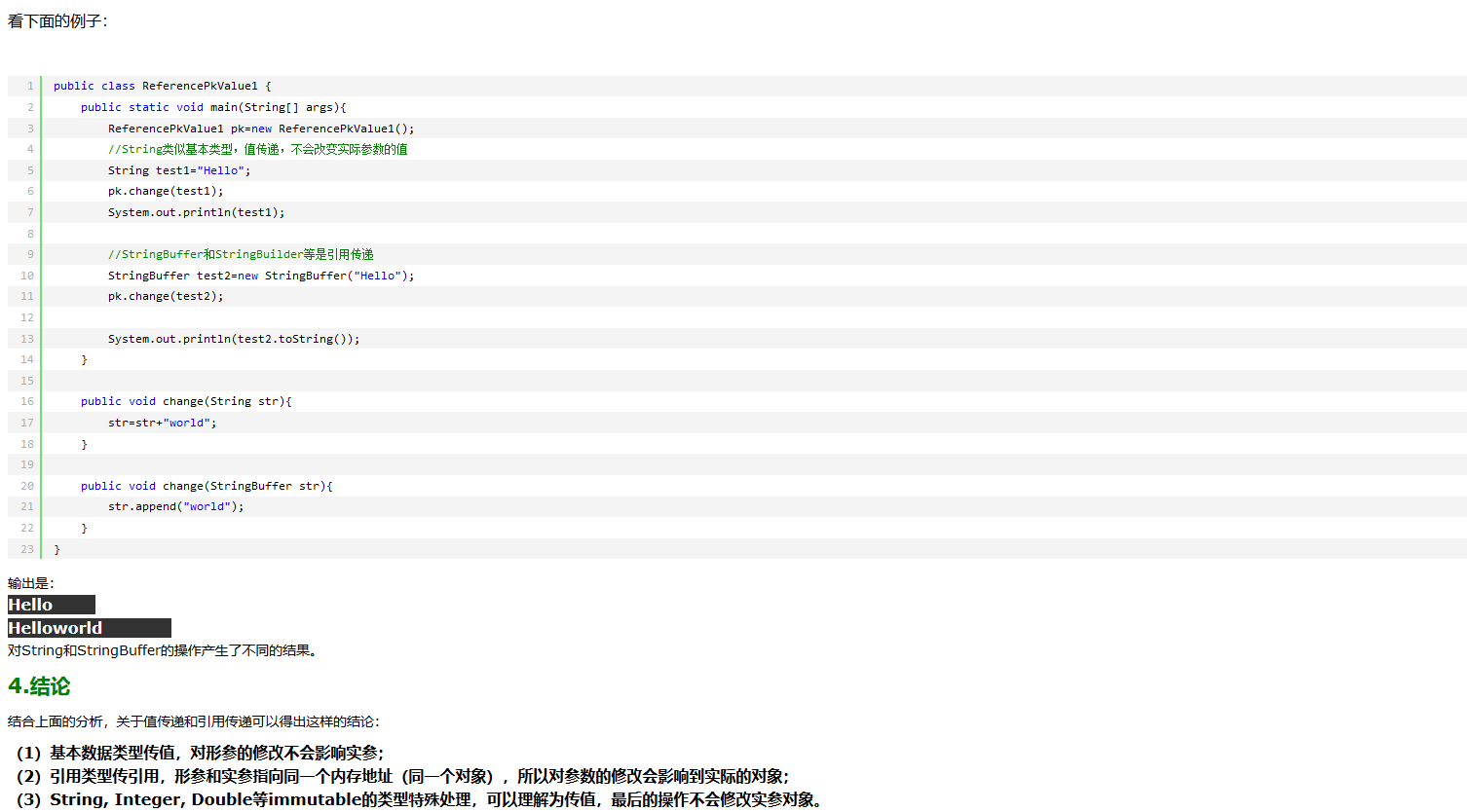
方法调用时，实际参数把它的值传递给对应的形式参数，函数接收的是原始值的一个copy，此时内存中存在两个相等的基本类型，即实际参数和形式参数，后面方法中的操作都是对形参这个值的修改，不影响实际参数的值。

引用传递：

也称为传地址。方法调用时，实际参数的引用(地址，而不是参数的值)被传递给方法中相对应的形式参数，函数接收的是原始值的内存地址；

在方法执行中，形参和实参内容相同，指向同一块内存地址，方法执行中对引用的操作将会影响到实际对象。





* 1. 数组
     1. Java数组的定义和使用
        1. 数组的概念

数组的概念：同一种类型数据的集合，其中数组就是一个容器。

* + - 1. 数组的好处

可以自动的给数组中的元素从0开始编号， 方便操作这些元素

* + - 1. 数组定义的格式

格式1：元素类型[] 数组名=new 元素类型[元素个数或者元素长度]；

int [] arr=new int[10];

格式2：元素类型[] 数组名=new 元素类型[]{元素，元素，元素…}

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

int[] arr={1,2,3,4,5}

|  |
| --- |
| /\*\*  \* 数组的定义和使用  \* **@author** Administrator  \*  \*/  **public** **class** Demo {  **public** **static** **void** main(String[] args) {  /\*\*  \* 第一种方式：元素类型[] 数组名=new 元素类型[元素个数或者元素长度]；  \*/  **int** [] arr=**new** **int**[7];  **for**(**int** i=0;i<arr.length;i++ )  {  arr[i]=i;  }  System.***out***.println(arr[5]);    /\*\*  \* 第二种方式：元素类型[] 数组名=new 元素类型[]{元素，元素，元素…}  \*  \* int[] arr=new int[]{1,2,3,4,5,6}  \* int[] arr={1,2,3,4,5}  \*/  **int**[] arr1=**new** **int**[] {1,2,3,4,5,6,7,8,9};    **for**(**int** i=0;i<arr1.length;i++) {  System.***out***.println(arr1[i]);  }  }  } |

* + - 1. 内存的划分：
  1. **寄存器** CPU来处理，目前涉及不到
  2. **本地方法区**
  3. **方法区**
  4. **栈内存**

**存储的时都是局部变量（局部代码块的作用和用途：）**

**而且变量所属的作用域，一旦结束，该变量自动释放。**

* 1. **对内存**

**堆内存存储的是数组和对象(数组本身就是对象)，凡是new建立的都是在堆中.**

**特点：**

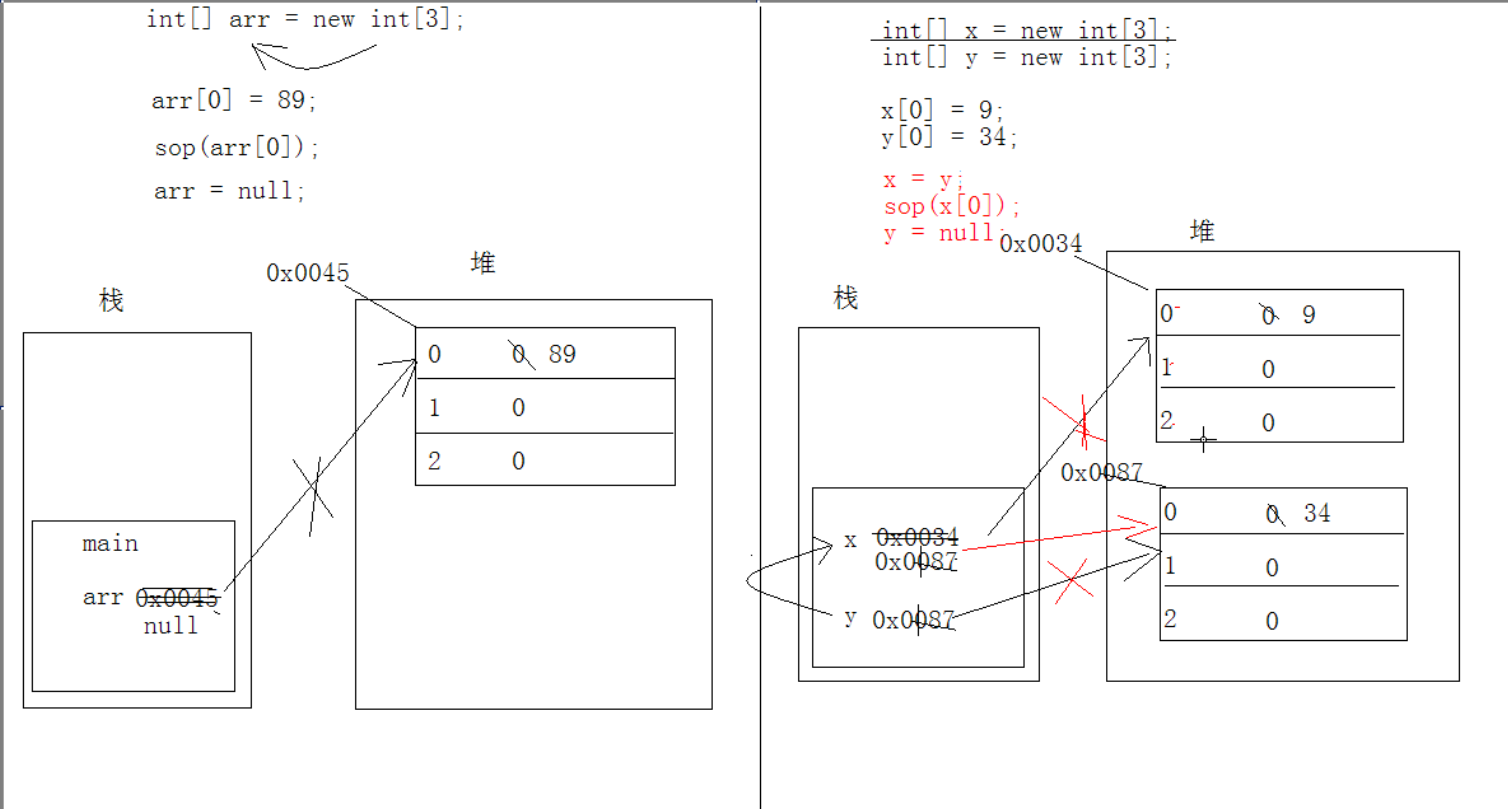
**a)堆内存中的每一个实体都有首地址**

**b)堆内存中的数据默认都会进行初始化，根据类型的而赋值不同类型**

**c)垃圾回收机制**

方法执行的过程：

**程序执行的时候先进栈内存，再进堆内存 ：如何体现，变量都存储在方法中，方法都在栈内存中。方法都没有加载进来呢，如何加载堆内存中**



**数组应用过程中常见的问题：**

1. **数组下标越界。 取值不存在的坐标是就会出现数组虾苗越界**
2. **数组未初始化，从变量中取值是就会发生 空指针异常**
   * + 1. 常用的算法：
          1. 字符串转数组计算和

|  |
| --- |
| /\*\*  \* 测试数组的定义和使用  \* 元素类型[] 数组名=new 元素类型[元素个数或者元素长度]  \* 计算值：789+456的和  \*/  public class ArryDemo {  public static void main(String[] args) {  String str="123,456,789,543,321";  String[] s= str.split(",");  for(int i=0;i<s.length;i++) {  System.out.println(s[i]);  }  int a=Integer.parseInt(s[2]);  int b=Integer.parseInt(s[1]);  System.out.println(a+b);  }  } |

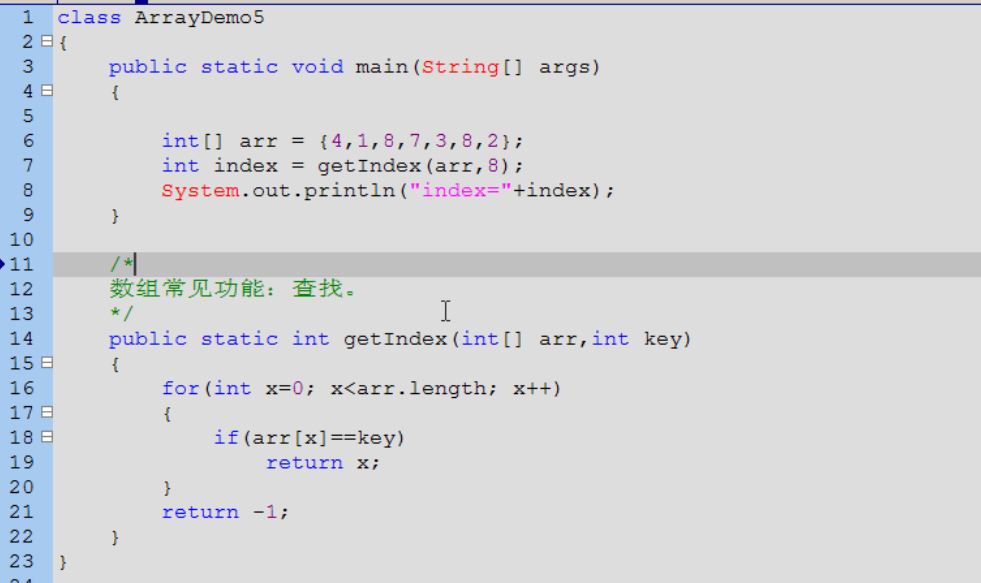
* + - * 1. 冒泡排序

|  |
| --- |
| /\*\*  \* 取字符串中"1,9,2,8,3,7,4,6,5"  \* 按照从小到大的顺序排列  \* 按照从大到校的顺序排序  \*/  public class OrderByDescAndAsc {  public static void main(String[] args) {  //从小到大排序  String str="9,1,2,8,3,7,4,6,5";  String[] s=str.split(",");  int[] a=new int[s.length];    for(int i=0;i<s.length;i++) {  a[i]=Integer.parseInt(s[i]);  }  int temp=0;  for(int i=0;i<a.length;i++) {  for(int j=i+1;j<a.length;j++) {  if(a[i]>a[j])  {  temp=a[i];  a[i]=a[j];  a[j]=temp;  }  }  }  for(int i=0;i<s.length;i++) {  System.out.print(a[i]+" ");;  }    }  } |

* + - * 1. 最大值最小值

|  |
| --- |
| /\*\*  \* 取字符串中"1,9,2,8,3,7,4,6,5"  \* 最大值 ----  \* 最小值 ----  \*/  public class MinAndMax {  public static void main(String[] args) {  String str="9,1,2,8,3,7,4,6,5";  String[] s=str.split(",");    int temp=0;//计算最小值  int temp\_max=0;  for(int i=0;i<(s.length-1);i++) {  if(i==0) {  temp=Integer.parseInt(s[i]);  temp\_max=Integer.parseInt(s[i]);  }  if(temp>Integer.parseInt(s[i+1])) {  temp=Integer.parseInt(s[i+1]);  }    if(temp\_max<Integer.parseInt(s[i+1])) {  temp\_max=Integer.parseInt(s[i+1]);  }  }    System.out.println("最小值"+temp);  System.out.println("最大值"+temp\_max);      }  } |

* + - * 1. 查找法 ，给出数字返回数字的坐标



* + - * 1. 二分查找法
        2. 课堂作业练习

1. 求出100以内的所有质数
2. 利用循环中找出数字最小数和最大数，并求出它的公约数。

21、24、6、27、9、12、15、18、30

1. 求出数字序列中第二小的数，并计算这个数的三次方

1、9、2、8、3、7、4、6、5

1. 数字排列：把数字1、9、2、8、3、7、4、6、5按照以下格式排序

1、9、2、8、3、7、4、6、5、5、6、4、7、3、8、2、9、1

1. 现在有如下一个数组去除数组中的0值：

String str=“1,3,4,5,0,0,6,6,0,5,4,7,6,7,0,5”

要求将以上数组中的0项去掉，将不为0的值存入一个新的数组，生成新的数组为

int newArr[]={1,3,4,5,6,6,5,4,7,6,7,5};

1. 去除数组中的重复元素：String str=“1,3,4,5,0,0,6,6,0,5,4,7,6,7,0,5”

并计算出重去除元素的总和。

* 1. 集合
     1. 集合对象的由来：

对象用于封装特有数据，对象多了需要存储，如果对象的个数不确定就需要用集合容器来存储。

集合特点：

1. 用于存储对象的容器。
2. 集合的长度是可变的
3. 集合中不可以存储基本数据类型。
4. 集合中的元素内容允许重复
   * 1. 集合的数据结构：

集合容器因为内部的数据结构不同，有多种具体容器。不断的向上抽取，就形成了集合框架。

框架的顶层Collection接口：

* + 1. Collection的常见方法：
       1. 添加。

boolean add(Object obj):

|  |
| --- |
| **public** **static** **void** show(Collection coll) {  //1、添加元素 会改变集合的长度  coll.add("abc1");  coll.add("abc2");  coll.add("abc3");  System.***out***.println(coll+" "+coll.size());  } |

boolean addAll(Collection coll):

|  |
| --- |
| **public** **static** **void** show(Collection c1,Collection c2) {    //在c1添加元素 会改变集合的长度  c1.add("abc1");  c1.add("abc2");  c1.add("abc3");  System.***out***.println(" c1:"+c1);  //在c2添加元素 会改变集合的长度  c2.add("abc2");  c2.add("abc6");  c2.add("abc7");  System.***out***.println(" c2:"+c2);    c1.addAll(c2); //把集合放在另外一个结合中    System.***out***.println(" c1:"+c1);  } |

* + - 1. 删除。

boolean remove(object obj):

boolean removeAll(Collection coll);

void clear();

* + - 1. 判断：

boolean contains(object obj):

|  |
| --- |
| **public** **static** **void** show(Collection coll) {  //1、添加元素 会改变集合的长度  coll.add("abc1");  coll.add("abc2");  coll.add("abc3");  System.***out***.println(coll+" "+coll.size());    //2、删除元素  coll.remove("abc2");//会改变集合的长度的    System.***out***.println(coll+" "+coll.size());  System.***out***.println(coll.contains("abc4"));//判断 给定的元素是否包含    //2.1清空元素  coll.clear();  System.***out***.println(coll+" "+coll.size());    } |

boolean containsAll(Colllection coll);

|  |
| --- |
| **public** **static** **void** show(Collection c1,Collection c2) {    //在c1添加元素 会改变集合的长度  c1.add("abc1");  c1.add("abc2");  c1.add("abc3");  System.***out***.println(" c1:"+c1);  //在c2添加元素 会改变集合的长度  c2.add("abc2");  c2.add("abc6");  c2.add("abc7");  System.***out***.println(" c2:"+c2);    c1.addAll(c2); //把集合放在另外一个结合中    System.***out***.println(" c1:"+c1);  //c2.add("abc8");  System.***out***.println(c1.containsAll(c2));//包含整个集合中的元素  } |

boolean isEmpty():判断集合中是否有元素。

|  |
| --- |
| **public** **static** **void** isEmpty(Collection c1,Collection c2) {    //在c1添加元素 会改变集合的长度  c1.add("abc1");  c1.add("abc2");  c1.add("abc3");  System.***out***.println(" c1:"+c1.isEmpty());  //在c2添加元素 会改变集合的长度  c2.add("abc2");  c2.add("abc6");  c2.add("abc7");  System.***out***.println(" c2:"+c2.isEmpty());  c2.clear();  System.***out***.println(" c2:"+c2.isEmpty());  } |

retainAll：去除元素的交际

|  |
| --- |
| **public** **static** **void** show(Collection c1,Collection c2) {    //在c1添加元素 会改变集合的长度  c1.add("abc1");  c1.add("abc2");  c1.add("abc3");  System.***out***.println(" c1:"+c1);  //在c2添加元素 会改变集合的长度  c2.add("abc2");  c2.add("abc6");  c2.add("abc7");  System.***out***.println(" c2:"+c2);    c1.retainAll(c2);//取交际  System.***out***.println(" c1:"+c1);    } |

* + - 1. 读取：

int size():

Iterator iterator():取出元素的方式：迭代器。

该对象必须依赖于具体容器，因为每一个容器的数据结构都不同。

所以该迭代器对象是在容器中进行内部实现的。

对于使用容器者而言，具体的实现不重要，只要通过容器获取到该实现的迭代器的对象即可，

也就是iterator方法。

Iterator接口就是对所有的Collection容器进行元素取出的公共接口。

其实就是抓娃娃游戏机中的夹子！

|  |
| --- |
| **public** **static** **void** show(Collection coll) {  //1、添加元素 会改变集合的长度  coll.add("abc1");  coll.add("abc2");  coll.add("abc3");  System.***out***.println(coll+" "+coll.size());    Iterator it=coll.iterator();//使用collection中的iterator方法。调用集合中的迭代器方法，是为了获取集合中的迭代器对象  **while**(it.hasNext()) {  System.***out***.println(it.next());  }  System.***out***.println(); //企业开发中推荐使用下面这种： 为什么  **for**(Iterator it1=coll.iterator();it1.hasNext();) {  System.***out***.println(it1.next());  }    } |

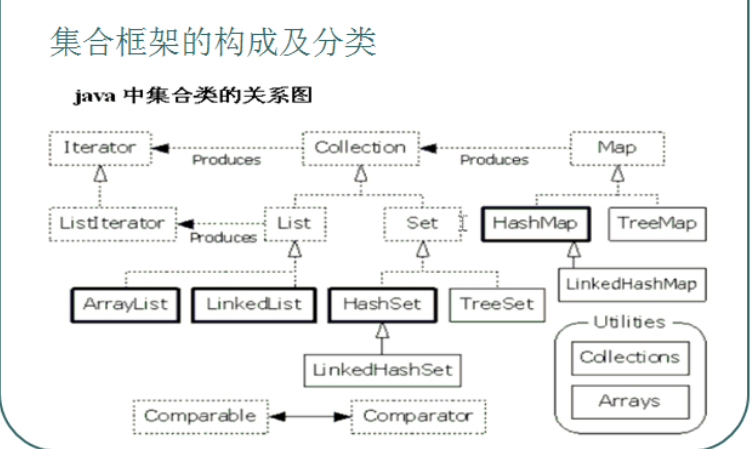
* + - 1. 其他：

boolean retainAll(Collection coll);取交集。

Object[] toArray():将集合转成数组。

* + - 1. Collection的子集

集合框架的结构图



* + - * 1. List

**|--List：**有序(存入和取出的顺序一致),元素都有索引(角标)，元素可以重复。

|--Set：元素不能重复,无序。

List:特有的常见方法：有一个共性特点就是都可以操作角标。

1，添加

void add(index,element);

void add(index,collection);

2，删除；

Object remove(index):

3，修改：

Object set(index,element);

4，获取：

Object get(index);

int indexOf(object);

int lastIndexOf(object);

List subList(from,to);

list集合是可以完成对元素的增删改查。

List:

|--Vector:内部是数组数据结构，是同步的。增删，查询都很慢！

|--ArrayList:内部是数组数据结构，是不同步的。替代了Vector。查询的速度快。

|--LinkedList:内部是链表数据结构，是不同步的。增删元素的速度很快。

* + - * 1. Set

作业：

1，自己去查文档演示Vector中的elements()方法。

2，LinkedList中的，addFirst addLast getFirst，getLast removeFirst removeLast。

3，既然集合是存储对象的，请定义ArryaList集合，并存储Person对象。如new Person("lisi",20);

并取出。将姓名和年龄打印出来。

* 1. 面向对象
     1. 了解面向对象的三个特征

面向对象的特点：

1、面向对象就是一种常见的思想，符合人们的思考习惯

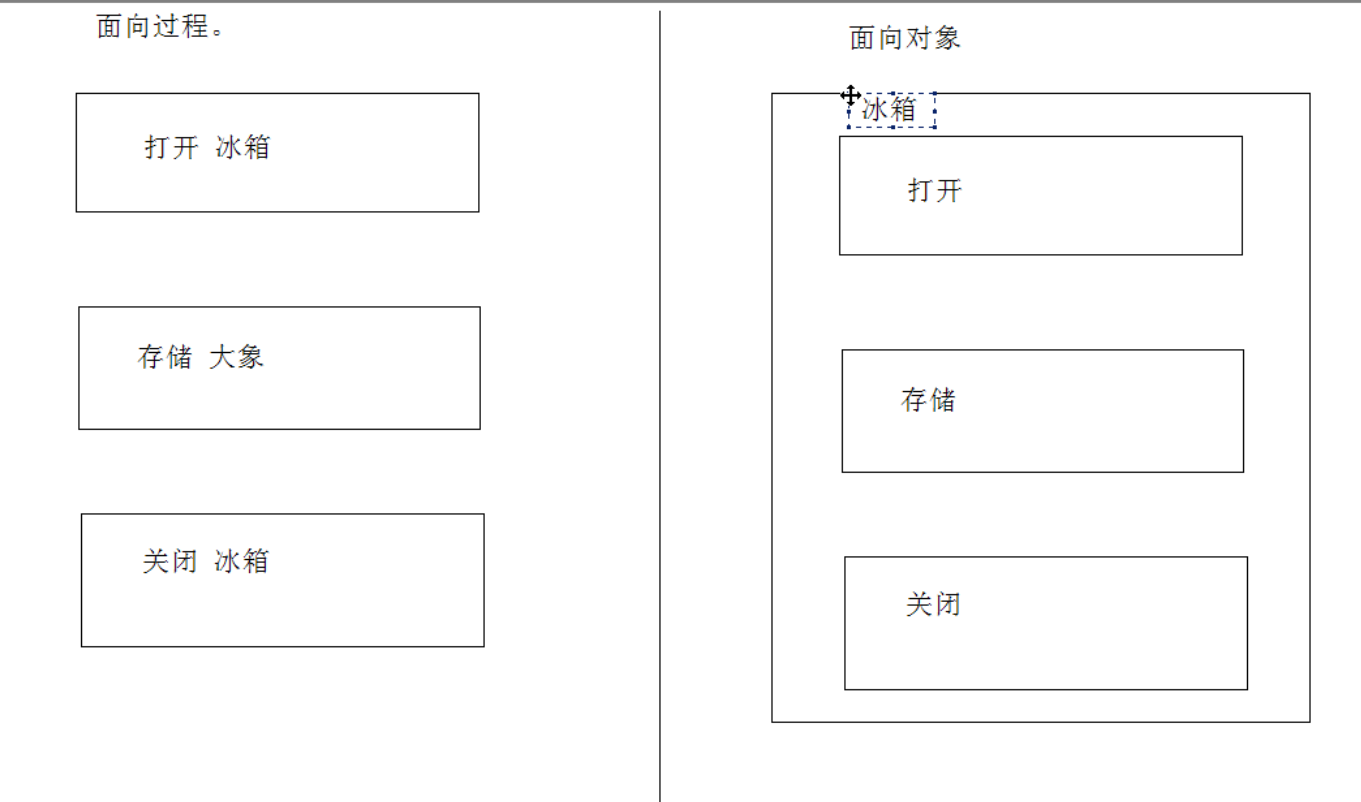
2、面向对象的出现将复杂问题简单话化。

3、面向对象的出现，让曾经在过程中的执行者变成了面向对象的指挥着。

面向过程和面向对象的区别：

：举个例子：把大象放在冰箱中的步骤：

* 1. 面向过程：打开冰箱、存储大象、关闭冰箱
  2. 面向对象: 冰箱打开、冰箱存储、冰箱关闭



* + - 1. 封装





* + - 1. 继承









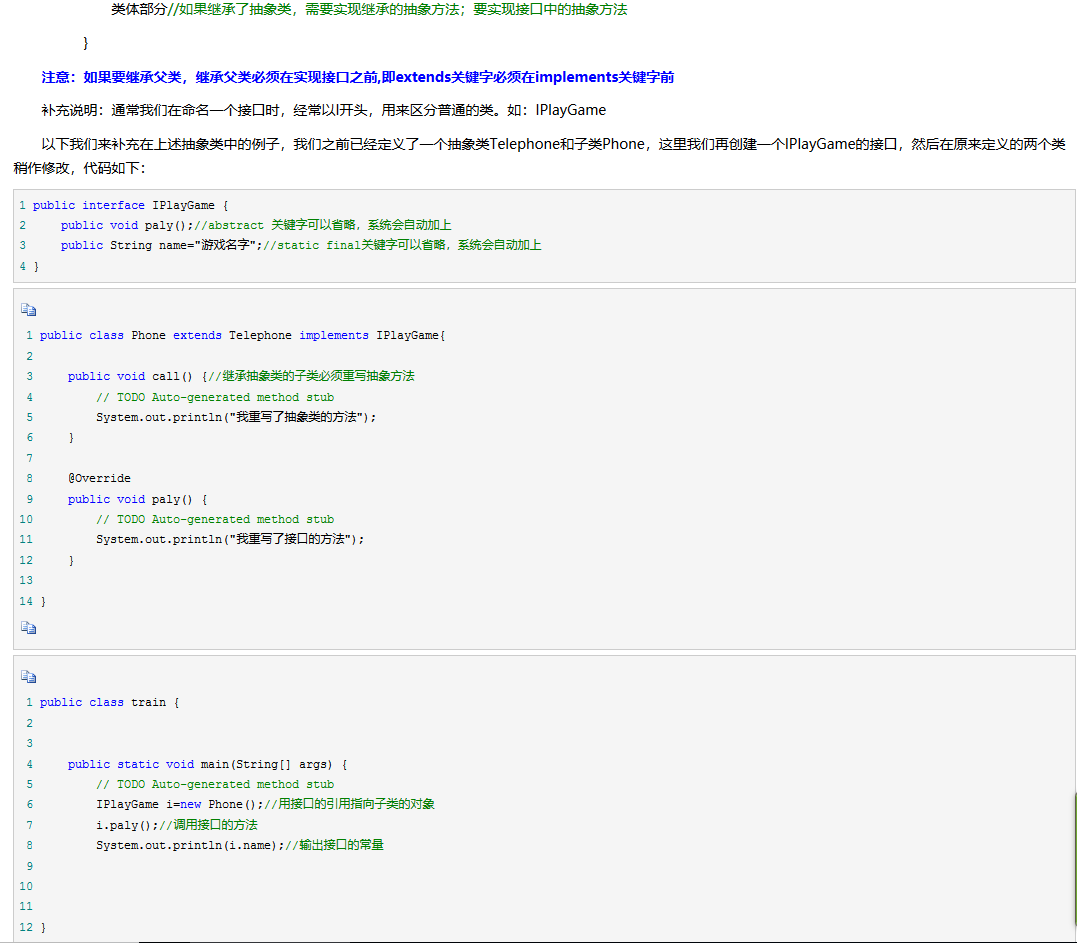
* + - 1. 多态

















* + 1. 掌握对象的创建和使用方式



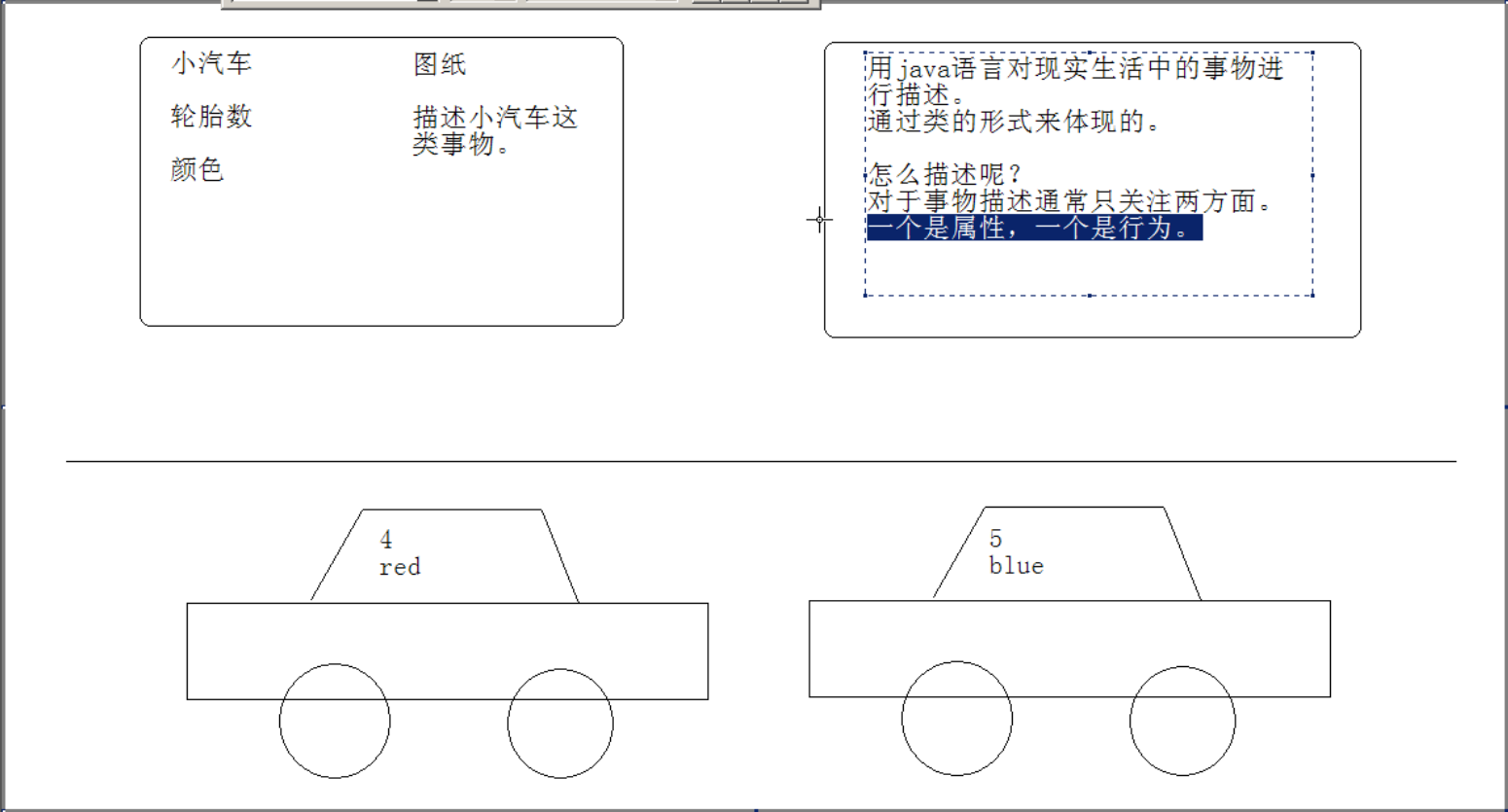




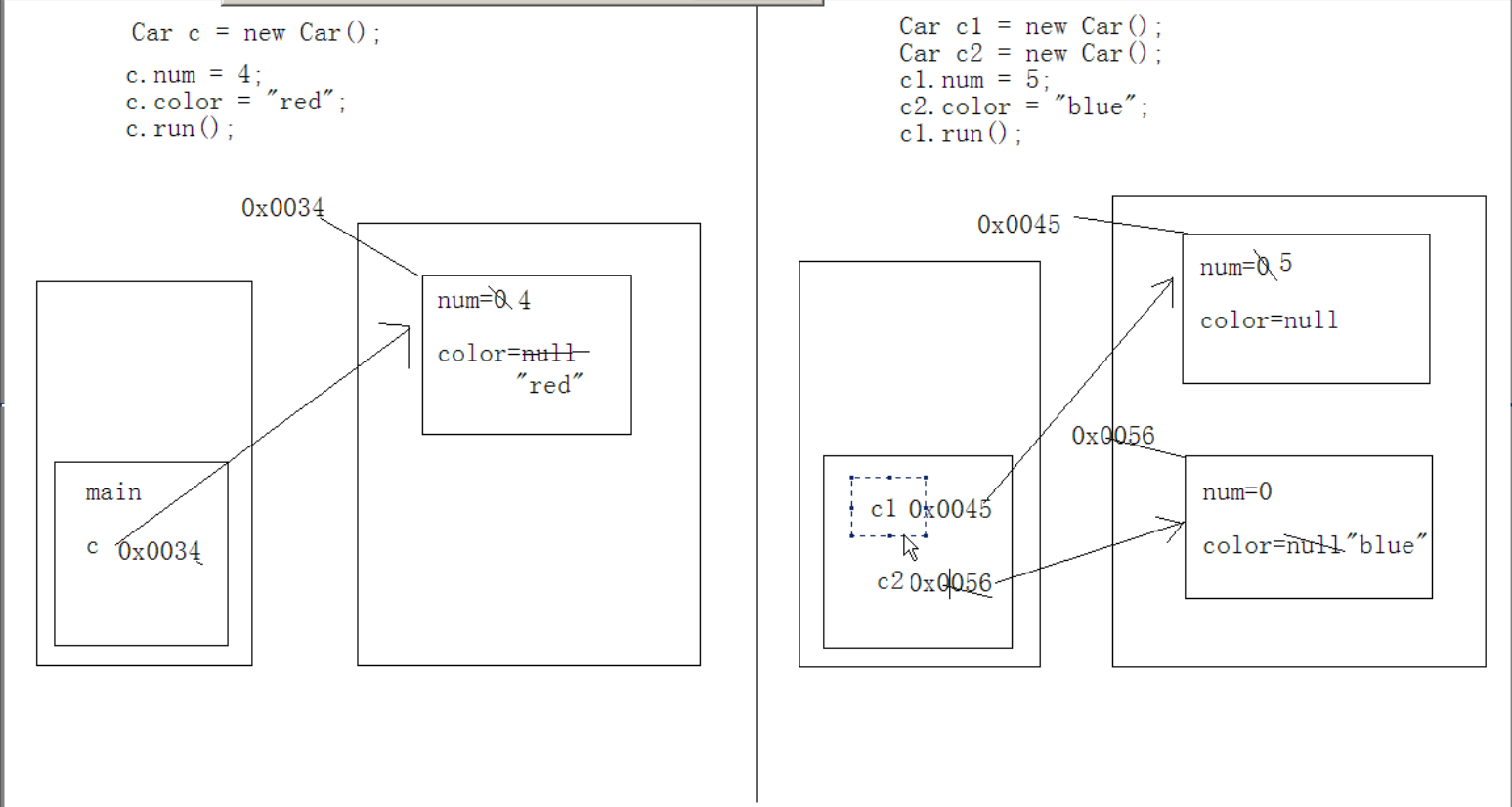




* + 1. 掌握类的封装方式



* + 1. 掌握方法的使用方式



成员变量和局部变量的区别

1. 变量的定义和作用域
   1. 成员变量定义在类中整个类中都可以访问。
   2. 局部变量定义在方法中只在语句、代码块中，只在所属的区域中有效
2. 内存中的位置
   1. 成员变量存储在堆内存的对象中
   2. 局部变量存储在栈内存的方法中
3. 变量的作用域
   1. 成员变量随着对象的创建而存在，随着对象消失的而消失
   2. 局部变量随着所属区域的执行而存在，随着所属区域的结束而释放
   3. 局部变量起作用的地方成员变量不起作用（除非通过类的对象名或者this关键字调用：this关键字指向的是对象本身（本类））
4. 初始化值
   1. 成员变量都有默认值
   2. 局部变量没有默认值
5. JAVA 的一个特性是多态

多态有两种体现：

第一种是子类和父类之间的 叫做 重写、覆盖

第二种是本类之间的，方法名完全一样，参数和返回值可以不同 重载

* + 1. 掌握this和static关键字的使用

1. JAVA进阶
2. 基于文档注释的JAVADOC接口文档
3. Java的文档注释
4. Javadoc文档的使用
5. Jar与可执行文件的
6. JXM管理在框架
7. SVN版本控制
8. JAVAWEB编程
9. Servlet的具体用法
10. 定义
11. Demo