传智播客

------[ XZQ ]------

2017.8.1 – 2017.8.

**传智播客--JAVA学习笔记**

讲师：毕向东

目录

[Day01 3](#_Toc496562311)

[第一章【JAVA概述】 3](#_Toc496562312)

[一、基础常识 3](#_Toc496562313)

[二、JAVA语言概述 3](#_Toc496562314)

[三、JAVA环境配置 4](#_Toc496562315)

[四、JAVA程序开发（Hello world） 5](#_Toc496562316)

[Day02-05 5](#_Toc496562317)

[第二章【JAVA语言基础组成】 5](#_Toc496562318)

[一、关键字 5](#_Toc496562319)

[二、标识符 6](#_Toc496562320)

[三、注释 6](#_Toc496562321)

[四、常量和变量 7](#_Toc496562322)

[五、运算符 9](#_Toc496562323)

[六、语句 14](#_Toc496562324)

[七、函数 15](#_Toc496562325)

[八、数组 16](#_Toc496562326)

[Day06-07 20](#_Toc496562327)

[第三章【面向对象】 20](#_Toc496562328)

[一、面向对象概念 20](#_Toc496562329)

[二、类与对象的关系 21](#_Toc496562330)

[三、基本/引用数据类型的参数传递（图解） 22](#_Toc496562331)

[四、封装（Encapsulate） 22](#_Toc496562332)

[五、构造函数 23](#_Toc496562333)

[六、this关键字 24](#_Toc496562334)

[七、static关键字 24](#_Toc496562335)

[八、单例设计模式 27](#_Toc496562336)

[Day08 27](#_Toc496562337)

[第四章【继承】 27](#_Toc496562338)

[一、继承的概述 27](#_Toc496562339)

[二、继承的特点 28](#_Toc496562340)

[三、super关键字 28](#_Toc496562341)

[四、函数覆盖 29](#_Toc496562342)

[五、子类的实例化过程 29](#_Toc496562343)

[六、final关键字 31](#_Toc496562344)

[七、抽象类 31](#_Toc496562345)

[八、接口（抽象类的特殊形式） 32](#_Toc496562346)

[九、多态 33](#_Toc496562347)

[十、内部类 35](#_Toc496562348)

[十一、匿名内部类 36](#_Toc496562349)

[十二、Object类 36](#_Toc496562350)

[十三、异常 37](#_Toc496562351)

[十四、包（package） 39](#_Toc496562352)

[Day09 40](#_Toc496562353)

[第五章【多线程】 40](#_Toc496562354)

[一、进程 40](#_Toc496562355)

[二、线程 40](#_Toc496562356)

[三、好处与弊端 40](#_Toc496562357)

[四、JVM中的多线程解析 40](#_Toc496562358)

[五、如何创建一个线程呢？ 41](#_Toc496562359)

[六、多线程运行图解-栈 42](#_Toc496562360)

[七、多线程的状态 42](#_Toc496562361)

[八、线程安全问题产生的原因（引出同步） 43](#_Toc496562362)

[九、同步（synchronized） 43](#_Toc496562363)

[十、多线程间通信 45](#_Toc496562364)

[一、进程 45](#_Toc496562365)

# Day01

## 第一章【JAVA概述】

### 一、基础常识

1. 软件开发
   1. 什么是软件：
      1. 软件：一系列安装特定顺序组织的计算机数据和指令的集合。
      2. 常见的软件：
         * 系统软件

Eg：DOS、windows、Linux等。

* + - * 应用软件

Eg：QQ、迅雷、浏览器

* 1. 什么是开发？
     1. 软件制作

1. 人机交互方式
   1. 软件的出现实现人与计算机之间的更好交互、
   2. 交互方式：
      1. 图形界面： 这种方式简单、直观，使用者易于接受，容易上手。
      2. 命令行方式： 需要一个控制台，输入特定的指令，让计算机完成一些操作，比较麻烦
   3. 什么是计算机语言？
      1. 语言： 人与人之间用于沟通的一种方式。

Eg：中文、英文、韩文

* + 1. 计算机语言：人与计算机交流的方式。

Eg：C、 C++、 JAVA

1. 常用的DOS命令

Help + 命令 可以查询命令的基本使用方法。

dos命令行，课程中常见的命令。

• dir : 列出当前目录下的文件以及文件夹

• md : 创建目录

• rd : 删除目录

• cd : 进入指定目录

• cd.. : 退回到上一级目录

• cd/ : 退回到根目录

• del : 删除文件

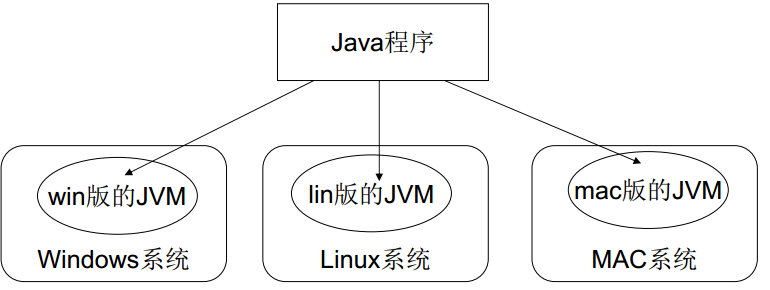
• exit : 推出dos命令行

### 二、JAVA语言概述

1. Java是一种面向对象的高级编程语言。
2. JAVA的三大架构
   1. JAVA EE 企业级开发，主要针对于Web应用程序开发
   2. JAVA SE 标准版 ，完成一些桌面应用程序的开发
   3. JAVA ME（被android取代了）
3. JAVA语言的跨平台性特点
   1. 什么是跨平台性

通过java语言编写的应用程序在不同的操作系统平台上都可以运行。

* 1. 其原理是什么?
     1. Java程序在JVM虚拟机中运行。
     2. 而虚拟机不是跨平台性。
     3. 图解如下：



因此JAVA具有良好的可移植性。

### 三、JAVA环境配置

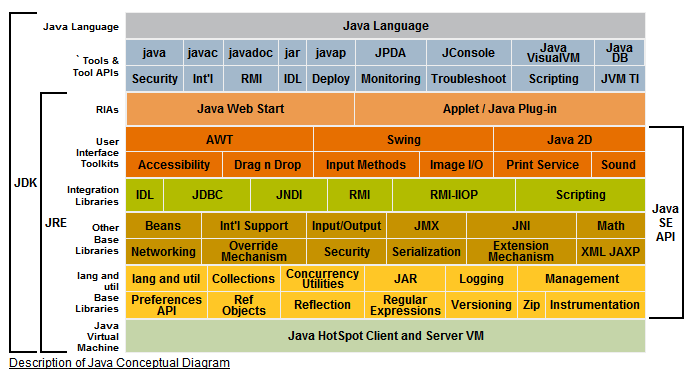
1. 什么是JRE、JDK？
   1. JRE (Java Runtime Environment Java运行环境)

虚拟机JVM + JAVA核心类库

* 1. JDK (Java Development Kit Java开发工具包)

JRE + JAVA开发工具

Java类库图如下：



1. 下载JDK、配置环境
   1. 下载：到甲骨文官网下载即可。
   2. 配置java环境（告诉系统JDK在哪里）

如果没有配置环境只能在其java本目录运行，有局限性。当我配置环境后可以在任意目录运行。

1. 变量名： JAVA\_HOME

变量值：C:\Program Files\Java\jdk1.5.0\_17

1. 变量名: CLASSPATH

变量值: .;%JAVA\_HOME%\lib\tools.jar;%JAVA\_HOME%\lib\dt.jar;

1. 变量名: PATH

变量值: %JAVA\_HOME%\bin

1. 临时配置方式：通过dos命令中set命令完成

• set ：用于查看本机的所有环境变量的信息。

• set 变量名 ：查看具体一个环境变量的值。

• set 变量名=：清空一个环境变量的值。

• set 变量名=具体值 ：给指定变量定义具体值。

1. 想要在原有环境变量值基础上添加新值呢？

• 首先，通过%变量名%操作符获取到原有环境变量的值。

• 然后加上新值后在定义给该变量名即可

• 举例：给path环境变量加入新值

• set path=新值;%path%

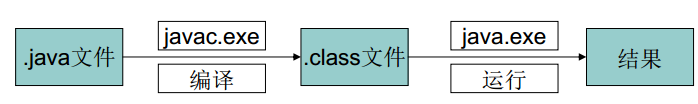
注意：这种配置方式只在当前dos窗口有效。窗口关闭，配置消失。

### 四、JAVA程序开发（Hello world）

1. 将Java代码编写到扩展名为.java的文件中。
   1. 建议Java文件名要与文件内类相同。（public 使用是必须相同）
   2. 编译生成的class 文件名就是其类名
   3. 单个Java内必须要有一个main函数
2. 通过javac命令对该java文件进行编译。

编译后在文件目录生成 “.class” 文件

1. 通过java命令对生成的class文件进行运行。

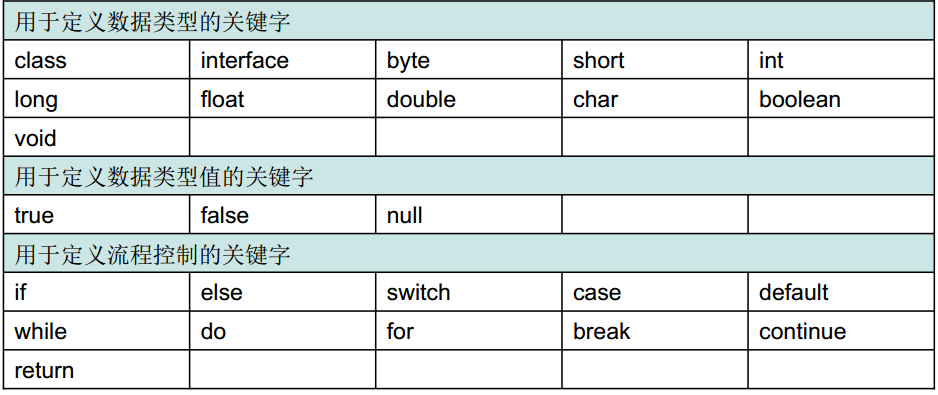


# Day02-05

## 第二章【JAVA语言基础组成】

### 一、关键字

1. 定义：被java语言赋予了特殊含义的单词
2. 特点：关键字中的所有字母都是小写
3. 关键字表如下：



### 二、标识符

1. 定义：在程序中自定义的一些名称。
2. 由26个英文字母大小写，数字： 0-9 符号： \_ $ 组成
3. 定义合法标识符规则：
   1. 数字不可以开头。
   2. 不可以使用关键字。

 Java中严格区分大小写。

注意：在起名字的时，为了提高阅读性，要尽量有意义。

1. Java中的名称规范：
   1. 包名：多单词组成时所有字母都小写。

• xxxyyyzzz

* 1. 类名、接口名：多单词组成时，所有单词的首字母大写。

• XxxYyyZzz

* 1. 变量名、函数名：多单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写。

• xxxYyyZzz（驼峰式）

* 1. 常量名：所有字母都大写。多单词时每个单词用下划线连接。

• XXX\_YYY\_ZZZ

### 三、注释

1. 定义：用于注解说明解释程序的文字。
2. 提高了代码的阅读性。
3. Java中的注释格式：

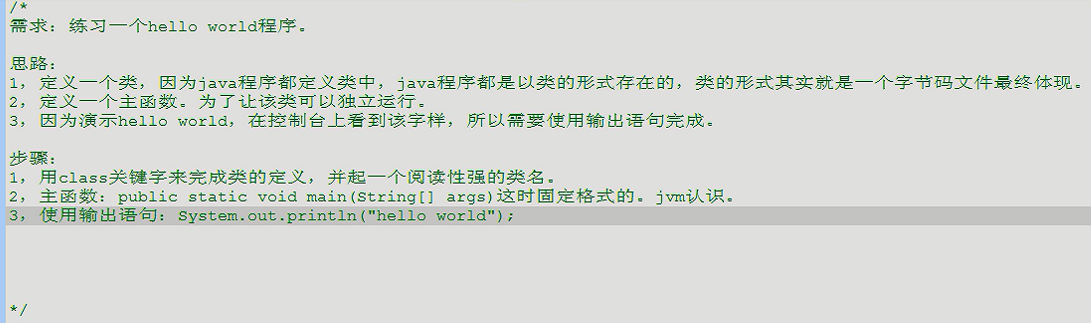
• 单行注释， 格式： //注释文字‘

• 多行注释， 格式： /\* 注释文字 \*/

• 文档注释， 格式： /\*\* 注释文字 \*/ （只有java有的注释格式，利用javadoc.exe工具）

* 1. 对于单行和多行注释，被注释的文字，不会被JVM（java虚拟机）解释执行。
  2. 对于文档注释，是java特有的注释，其中注释内容可以被JDK提供的工具 javadoc 所解析，生成一套以网页文件形式体现的该程序的说明文档。

1. 注释是一个程序员必须要具有的良好编程习惯。
   1. 初学者编写程序可以养成习惯：先写注释再写代码。①需求 ②思路 ③步骤
   2. 将自己的思想通过注释先整理出来，在用代码去体现。因为代码仅仅是思想的一种体现形式而已。



### 四、常量和变量

**常量**

1. 常量表示不能改变的数值。
2. Java中常量的分类：
   1. 整数常量。所有整数
   2. 小数常量。所有小数
   3. 布尔型( boolean )常量。较为特有，只有两个数值。 true和 false。
   4. 字符常量。将一个数字字母或者符号用单引号( ' ' )标识。
   5. 字符串常量。将一个或者多个字符用双引号（“ ”）标识。
   6. null常量。只有一个数值就是:null.
   7. 对于整数： java有三种表现形式：

• 二进制： 0-1 ，满2进1.

• 十进制： 0-9 ，满10进1.

• 八进制： 0-7 ，满8进1. 用0开头表示。

• 十六进制： 0-9， A-F（10-15），满16进1. 用0x开头表示。

1. 进制的基本转换
   1. 十进制--二进制 互转

• 十进制转成二进制 除以2取余数

• 二进制转成十进制 乘以2的幂数

* 1. 十进制--八进制 互转
  2. 十进制--十六进制 互转
  3. 负数的二进制表现形式

• 对应的正数二进制取反加1

**变量**

1. 变量的概念：

•内存中的一个存储区域

•该区域有自己的名称（变量名）和类型（数据类型）

•该区域的数据可以在同一类型范围内不断变化

1. 为什么要定义变量：

•用来不断的存放同一类型的常量，并可以重复使用

1. 使用变量注意：

•变量的作用范围（一对{}之间有效）

•初始化值



1. 定义变量的格式：

•数据类型 变量名 = 初始化值；

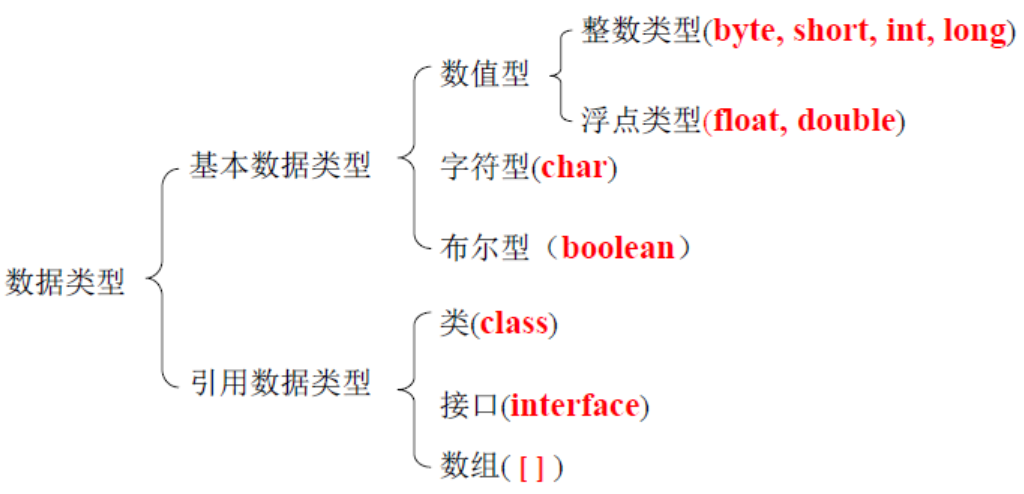
•格式是固定的，记住格式，以不变应万变。



理解：变量就如同数学中的未知数。

1. Java中数据类型如下

Java语言是强类型语言，对于每一种数据都定义了明确的具体数据类型，在内存总分配了不同大小的内存空间



整数默认：int 小数默认：double

1. 类型转换
   1. 自动类型转换（也叫隐式类型转换）
   2. 强制类型转换（也叫显式类型转换）
   3. 类型转换的原理

什么时候要用强制类型转换?

 表达式的数据类型自动提升

• 所有的byte型、 short型和char的值将被提升到int型。

• 如果一个操作数是long型，计算结果就是long型；

• 如果一个操作数是float型，计算结果就是float型；

• 如果一个操作数是double型，计算结果就是double型。分析

• System.out.println(‘a’)与System.out.println(’a’+1) 的区别。

1. 自动类型提升

byte b = 3;

int x = 4;

x = x + b;//b会自动提升为int类型进行运算。

1. 强制类型转换

byte b = 3;

b = b + 4;//报错

b = (byte)b+4;//强制类型转换，强制将b+4的结果转换为byte类型，再赋值给b。

1. 思考：

byte b1=3,b2=4,b;

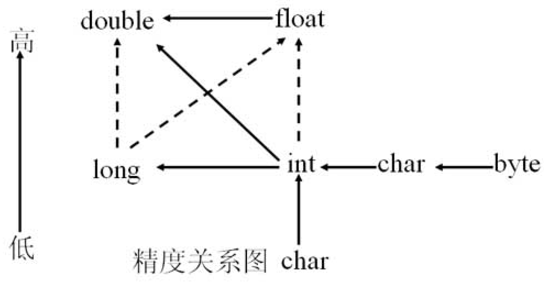
b=b1+b2;

b=3+4;

哪句是编译失败的呢？为什么呢？

第二句，凡是整数的都是int类型，赋值号右边都是整数参量运算如果没有超过byte范围既不会报错，如果是赋值号右边都是变量就会报错，也就意味着变量有可能超过byte范围。

1. 下图中展示了运算过程中，数据自动向上造型的原则:



注：1、实线箭头表示没有信息丢失的转换，也就是安全性的转换，虚线的箭头表示有精度损失的转化，也就是不安全的。

2、当两个操作数类型不相同时，操作数在运算前会子松向上造型成相同的类型，再进行运算。

### 五、运算符

1. 运算符种类：
   1. 算术运算符
   2. 赋值运算符
   3. 比较运算符
   4. 逻辑运算符
   5. 位运算符
   6. 三元运算符
2. 运算符详细如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 优先级 | 运算符分类 | 结合顺序 | 运算符 |
| 由  高  到  低 | 分隔符 | 左结合 | .    []     ( )     ;      , |
| 一元运算符 | 右结合 | !  ++     --     -   ~ |
| 算术运算符  移位运算符 | 左结合 | \*     /      %    +     -      <<   >>   >>> |
| 关系运算符 | 左结合 | <     >     <=   >=   instanceof(Java 特有)   = =  != |
| 逻辑运算符 | 左结合 | ! &&  ||  ~  &  |  ^ |
| 三目运算符 | 右结合 | 布尔表达式?表达式1:表达式2 |
| 赋值运算符 | 右结合 | =  \*=     /=  %=   +=   -=    <<= >>= >>>=  &=  \*=  |= |

* 1. 一元运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 含义 | 例子 |
| - | 改变数值的符号，取反 | -x（-1\*x） |
| ~ | 逐位取反，属于位运算符 | ~x |
| ++ | 自加1 | x++ |
| -- | 自减1 | x-- |

++x 因为++在前，所以先加后用。

x++ 因为++在后，所以先用后加。

注意：a+ ++b和a+++b是不一样的（因为有一个空格）。

* 1. 算术运算符

所谓算术运算符，就是数学中的加、减、乘、除等运算。因算术运算符是运算两个操作符，故又称为二元运算符。

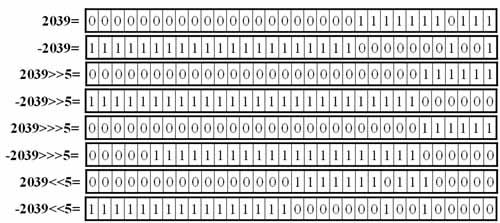
|  |  |  |
| --- | --- | --- |
| 运算符 | 含义 | 例子 |
| + | 加法运算 | x+y |
| - | 减法运算 | x-y |
| \* | 乘法运算 | x\*y |
| / | 除法运算 | x/y |
| % | 取模运算（求余运算） | x%y |

* 1. 移位运算符

移位运算符操作的对象就是二进制的位，可以单独用移位运算符来处理int型整数。

|  |  |  |
| --- | --- | --- |
| 运算符 | 含义 | 例子 |
| << | 左移运算符，将运算符左边的对象向左移动运算符右边指定的位数（在低位补0） | x<<3 |
| >> | "有符号"右移运算 符，将运算符左边的对象向右移动运算符右边指定的位数。使用符号扩展机制，也就是说，如果值为正，则在高位补0，如果值为负，则在高位补1. | x>>3 |
| >>> | "无符号"右移运算 符，将运算符左边的对象向右移动运算符右边指定的位数。采用0扩展机制，也就是说，无论值的正负，都在高位补0. | x>>>3 |

Eg: 一个值为2039。

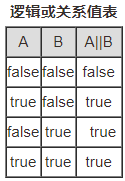
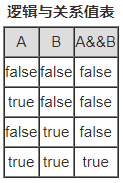
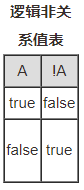


* 1. 关系运算符

Java具有完备的关系运算符，这些关系运算符同数学中的关系运算符是一致的。具体说明如下：

|  |  |  |
| --- | --- | --- |
| 运算符 | 含义 | 例子 |
| < | 小于 | x<y |
| > | 大于 | x>y |
| <= | 小于等于 | x<=y |
| >= | 大于等于 | x>=y |
| == | 等于 | x==y |
| != | 不等于 | x!=y |

* 1. 逻辑运算符



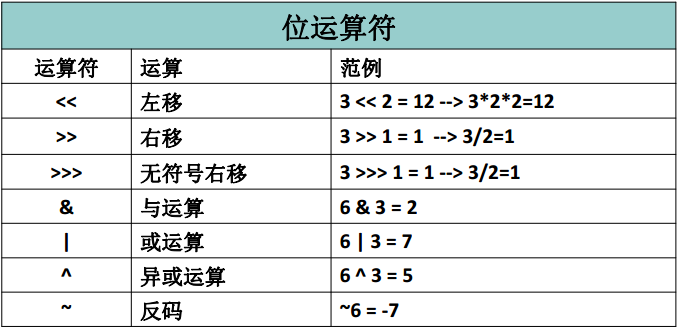
* + - * 在运用逻辑运算符进行相关的操作，就不得不说“短路”现象。代码如下：

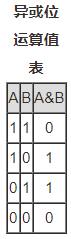
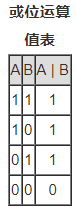
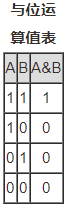
if(1==1 && 1==2 && 1==3){ }

代码从左至右执行，执行第一个逻辑表达式后：true && 1==2 && 1==3

执行第二个逻辑表达式后：true && false && 1==3

* + - * 因为其中有一个表达式的值是false，可以判定整个表达式的值是false，就没有必要执行第三个表达式了，所以java虚拟机不执行1==3代码，就好像被短路掉了。
      * 逻辑或也存在“短路”现象，当执行到有一个表达式的值为true时，整个表达式的值就为true，后面的代码就不执行了。
  1. 位运算符号





* 与运算符

与运算符用符号“ & ”表示，其使用规律如下：

两个操作数中位都为1，结果才为1，否则结果为0。

* 或运算符

或运算符用符号“ | ”表示，其运算规律如下：

两个位只要有一个为1，那么结果就是1，否则就为0

* 非运算符

非运算符用符号“ ~ ”表示，其运算规律如下：

如果位为0，结果是1，如果位为1，结果是0。

* 异或运算符

异或运算符是用符号“ ^ ”表示的，其运算规律是：

两个操作数的位中，相同则结果为0，不同则结果为1。

注意：左移动几位其实就是该数据乘以2的几次方

右移动几位其实就是该数据除以2的几次方

位运算详细请看《粤嵌c语言学习笔记》

* 两个数换位典例





* 1. 三目运算符

三目运算符是一个特殊的运算符，它的语法形式如下：

布尔表达式？表达式1：表达式2

运算过程：如果布尔表达式的值为true，就返回表达式1的值，否则返回表达式2的值，

* 1. 赋值运算符

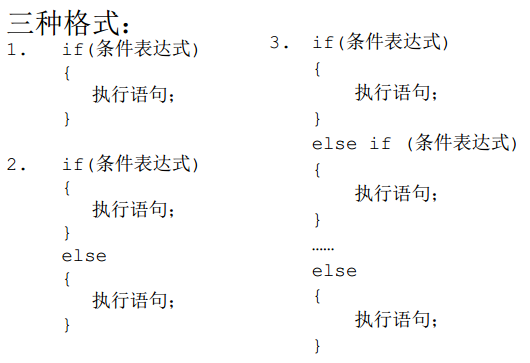
赋值运算符是程序中最常用的运算符了，示例如下：

|  |  |  |
| --- | --- | --- |
| 运算符 | 例子 | 含义 |
| += | x+=y | x=x+y |
| -= | x-=y | x=x-y |
| \*= | x\*=y | x=x\*y |
| /= | x/=y | x=x/y |
| %= | x%=y | x=x%y |
| >>= | x>>=y | x=x>>y |
| >>>= | a>>>=y | x=x>>>y |
| <<= | a<<=y | x=x<<y |
| &= | x&=y | x=x&y |
| |= | x|=y | x=x|y |
| ^= | x^=y | x=x^y |

这类运算符中有一个特性，就是计算机会根据左边变量的类型，把计算的结果强制成左边变量的类型再把值赋给左边。

### 六、语句

1. 判断结构



if语句特点：

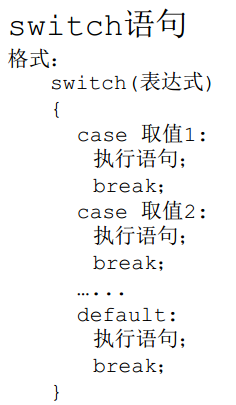
a,每一种格式都是单条语句。

b,第二种格式与三元运算符的区别：三元运算符运算完要有值出现。

好处是：可以写在其他表达式中。

c,条件表达式无论写成什么样子，只看最终的结构是否是true 或者 false;

1. 选择结构



switch语句特点：

a, switch语句选择的类型只有四种： byte， short， int ， char。

b, case之间与default没有顺序。先执行第一个case，没有匹配的case执行default。

c, 结束switch语句的两种情况：遇到break，执行到switch语句结束。

d, 如果匹配的case或者default没有对应的break，那么程序会继续向下执行，运行可以执行的语

句，直到遇到break或者switch结尾结束。

1. 循环结构
2. 局部代码块

语法：

{

程序语句；

}

### 七、函数

1. 函数的定义
2. 什么是函数？

• 函数就是定义在类中的具有特定功能的一段独立小程序。

• 函数也称为方法

1. 函数的格式：

修饰符 返回值类型 函数名(参数类型 形式参数1，参数类型 形式参数2， )

{

执行语句;

return 返回值;

}

返回值类型：函数运行后的结果的数据类型。

参数类型：是形式参数的数据类型。

形式参数：是一个变量，用于存储调用函数时传递给函数的实际参数。

实际参数：传递给形式参数的具体数值。

return：用于结束函数。特殊情况没有返回值时可以直接写return；

返回值：该值会返回给调用者。

1. 函数的特点
   1. 定义函数可以将功能代码进行封装
   2. 便于对该功能进行复用
   3. 函数只有被调用才会被执行
   4. 函数的出现提高了代码的复用性
   5. 对于函数没有具体返回值的情况，返回值类型用关键字void表示，那么该函数中的return语句如果在最后一行可以省略不写。

注意：

• 函数中只能调用函数，不可以在函数内部定义函数。

• 定义函数时，函数的结果应该返回给调用者，交由调用者处理。

1. 函数的应用
2. 两个明确

• 明确要定义的功能最后的结果是什么？

• 明确在定义该功能的过程中，是否需要未知内容参与运算

1. 示例：

• 需求：定义一个功能，可以实现两个整数的加法运算。

• 分析：

• 该功能的运算结果是什么？两个数的和，也是一个整数(int)

• 在实现该功能的过程中是否有未知内容参与运算？加数和被加数是不确定的。

(两个参数int， int)

• 代码：

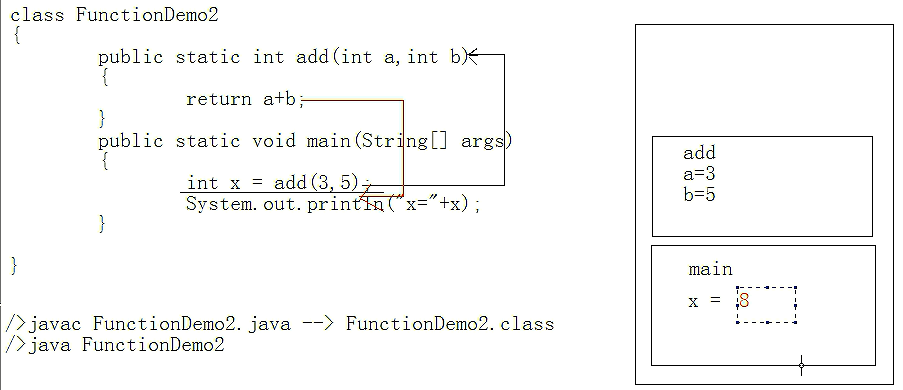
int getSum(int x,int y)

{

return x+y;

}

1. 函数内存加载过程



1. 函数的重载
   1. 概念：在同一个类中，允许存在一个以上的同名函数， 只要它们的参数个数或者参数类型不同即可。
   2. 特点：与其返回类型无关，只看其参数列表。
   3. 重载的好处：方便于阅读，优化了程序设计
   4. Eg

//返回两个整数的和

int add(int x,int y){return x+y;}

//返回三个整数的和

int add(int x,int y,int z){return x+y+z;}

//返回两个小数的和

double add(double x,double y){return x+y;}

### 八、数组

1. 数组的定义
   1. 概念：同一种类型数据的集合，其实数组就是容器
   2. 数组的好处：可以自动给数组中的元素从0开始编号，方便操作这些元素。
   3. 格式：
      1. 元素类型 [ ] 数组名 = new 元素类型 [ 元素个数或者数组长度 ] ；]

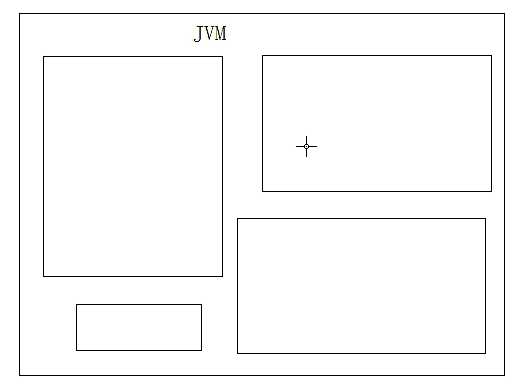
Eg: int [ ] arr = new int [50];

* + 1. 元素类型 [ ] 数组名 = < new 元素类型 [ ] > { 元素1 ， 元素2 ，··````` }

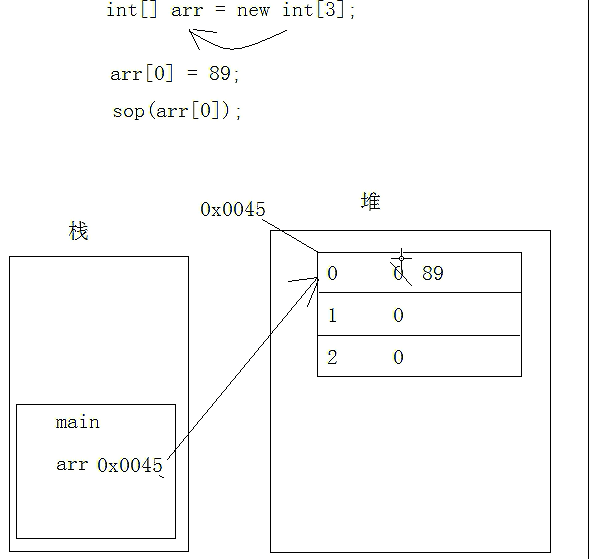
Eg: int [ ] a = new int[ ] { 1, 2, 3, 4, 5}

Int [ ] b = { 1 ,2, 3, 4, 5}

1. 数组的内存分配及特点
   1. 物理机内存的分配



* 1. 内存的划分
     1. 寄存器
     2. 本地方法区
     3. 方法区
     4. 栈内存
     5. 堆内存
  2. 栈内存
     1. 基本数据类型的变量（int、short、long、byte、float、double、boolean、char等）以及对象的引用变量，其内存分配在栈上。存储的都是局部变量，凡是在方法内部的都是局部变量，作用域一旦结束就自动释放。
     2. 特点："后进先出"
  3. 堆内存
     1. 存储是数组和对象（其实数组就是对象），凡是new建立在堆中。
     2. 特点：
        1. 每一个实体都有其首地址值
        2. 堆内存中的每一个变量都有默认初始化值，根据类型的不同而不同，整数是0，小数0.0。
        3. 垃圾回收机制



1. 数组操作常见问题
   1. 数组下标越界

数组脚标越界异常(ArrayIndexOutOfBoundsException)

* 1. 当引用类型变量没有任何实体指向时，还在用其操作实体，就会发生该异常。

空指针异常(NullPointerException)

1. 数组常见操作
   1. 获取最值(最大值，最小值)
   2. 排序（选择排序，冒泡排序）
   3. 折半查找(二分查找)
2. 数组中的数组（二维数组）
   1. 格式：
      1. 数据类型[ ][ ] 数组名 = new 数据类型[m][n];

m:表示这个二维数组有多少个一维数组。

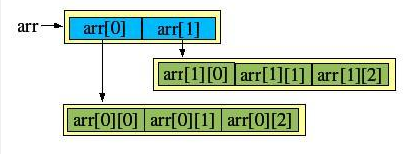
n:表示每一个一维数组的元素有多少个。

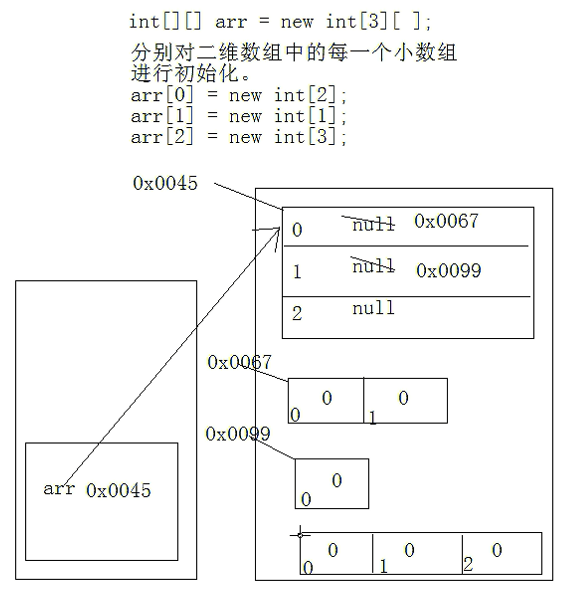
* + 1. 数据类型[ ][ ] 数组名 = new 数据类型[m][ ];

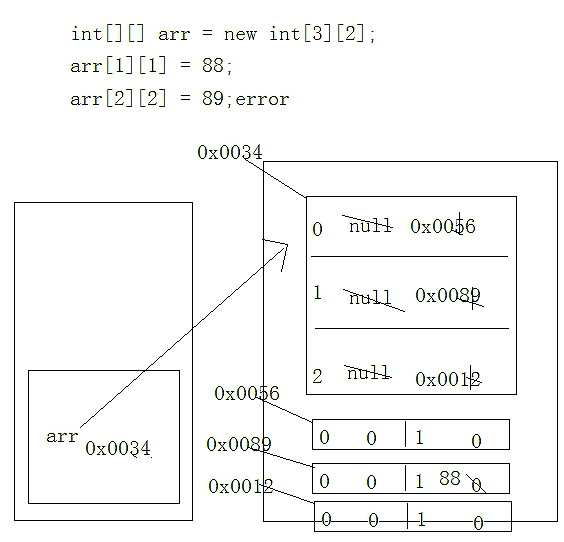
m:表示这个二维数组有多少个一维数组。

列数没有给出，可以动态的给。这一次是一个变化的列数。

* + 1. 数据类型[ ][ ] 数组名 = {{元素1,元素2...},{元素1,元素2...},{元素1,元素2...}};
  1. 内存结构







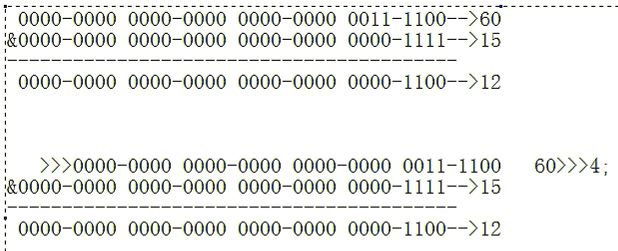
1. 二维数组的长度

Eg： int a[ ][ ] = new int[5][6]

1. a . length 就是二维数组中一维数组的个数。
2. a [ 0 ] . length j就是第a[ 0 ] 中的一维数组的元素个数

所以二维数组的个数就等于**count = a.length \* a[0].length**

1. 数组常见应用
   1. 进制转换
      1. 利用位运算0的来实现进制转换，思想如下图



数组在其中做为参照表。

Char a[] = {1, 2, 3 , 4, 5, 6, 7 , 8, 9, A, B , C , D, E, F};

# Day06-07

## 第三章【面向对象】

### 一、面向对象概念

例子： 把大象放进冰箱。

1. 理解面向对象
   1. 面向对象是相对面向过程而言
   2. 面向对象和面向过程都是一种思想
   3. 面向过程：强调的是功能行为（动作）
   4. 面向对象：将功能封装进对象，强调具备了功能的对象。
   5. 面向对象是基于面向过程的。
2. 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，将复杂的问题简单化。
   3. 面向对象的出现，让曾经在过程中的执行者，变成了对象中的指挥者。
   4. 完成需求时：

• 先要去找具有所需的功能的对象来用。

• 如果该对象不存在，那么创建一个具有所需功能的对象。

• 这样简化开发并提高复用。

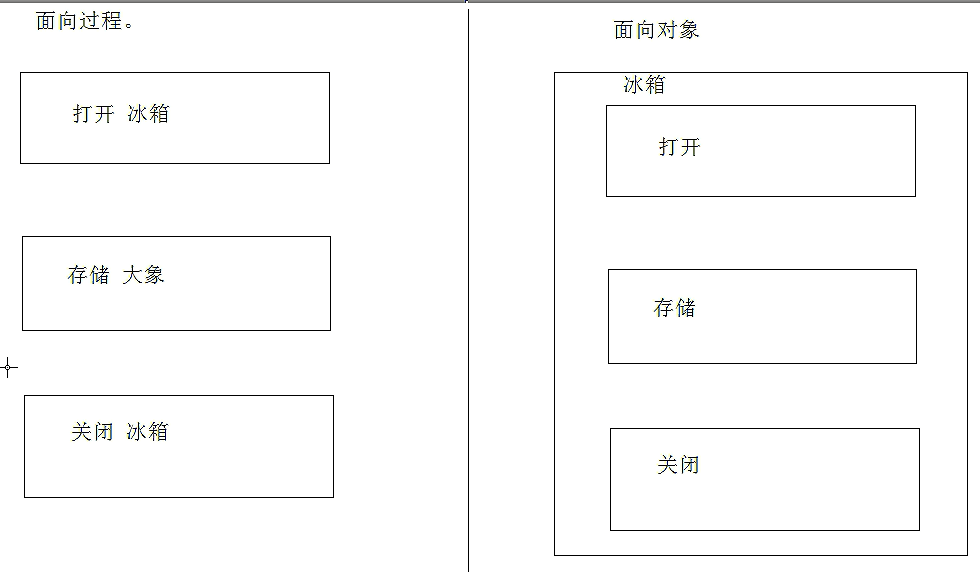
* 1. 面向对象的特征：

• 封装(encapsulation)

• 继承(inheritance)

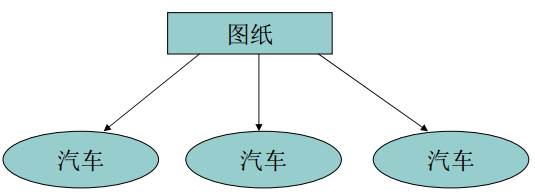
• 多态(polymorphism)

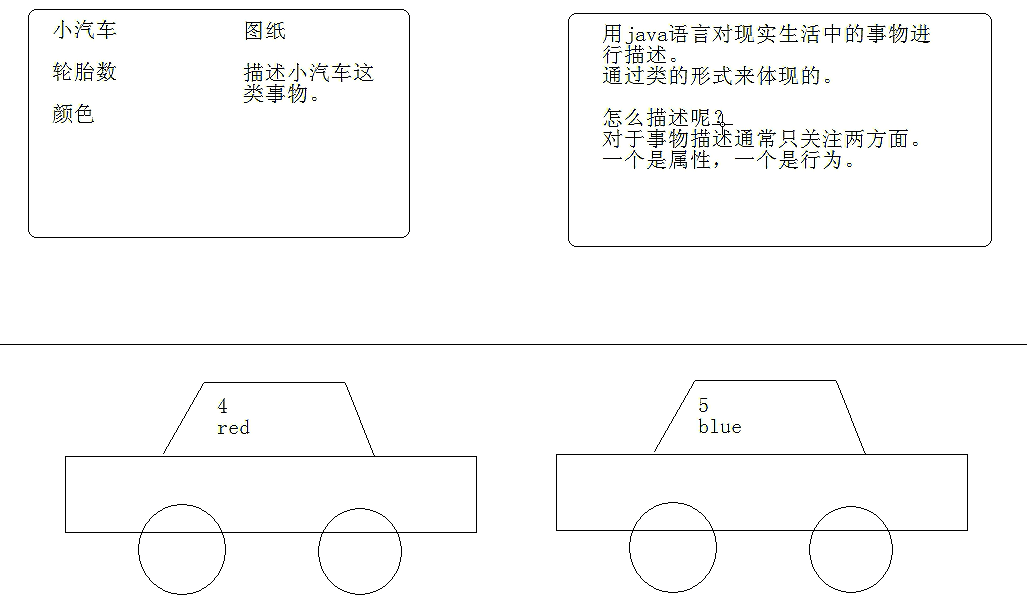
1. 例子：



### 二、类与对象的关系

1. 类与对象(图例)
   1. 类与对象的关系如图：



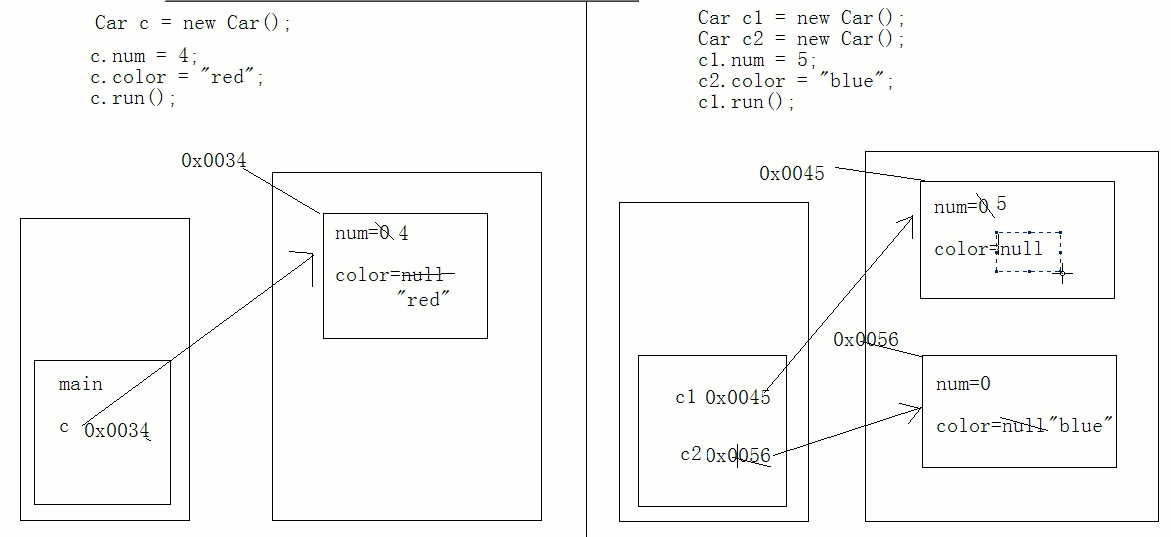


1. 类与对象的定义
   1. 类：生活中描述事物无非就是描述事物的属性和行为。
   2. 对象：该类实物实实存在的个体。（万物皆为对象）

定义类其实就是在定义类中的成员。

成员：成员变量<—> 属性， 成员函数就是<—>行为

1. 对象的内存体现



对象是储存在堆 内存的。

1. 成员变量和局部变量的区别
   1. 成员变量：（实例变量）

• 成员变量定义在类中，在整个类中都可以被访问。

• 成员变量随着对象的建立而建立，存在于对象所在的堆内存中。

• 成员变量有默认初始化值。

* 1. 局部变量：

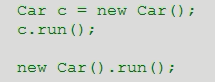
• 局部变量只定义在局部范围内，如：函数内，语句内等。

• 局部变量存在于栈内存中。

• 作用的范围结束，变量空间会自动释放。

• 局部变量没有默认初始化值。

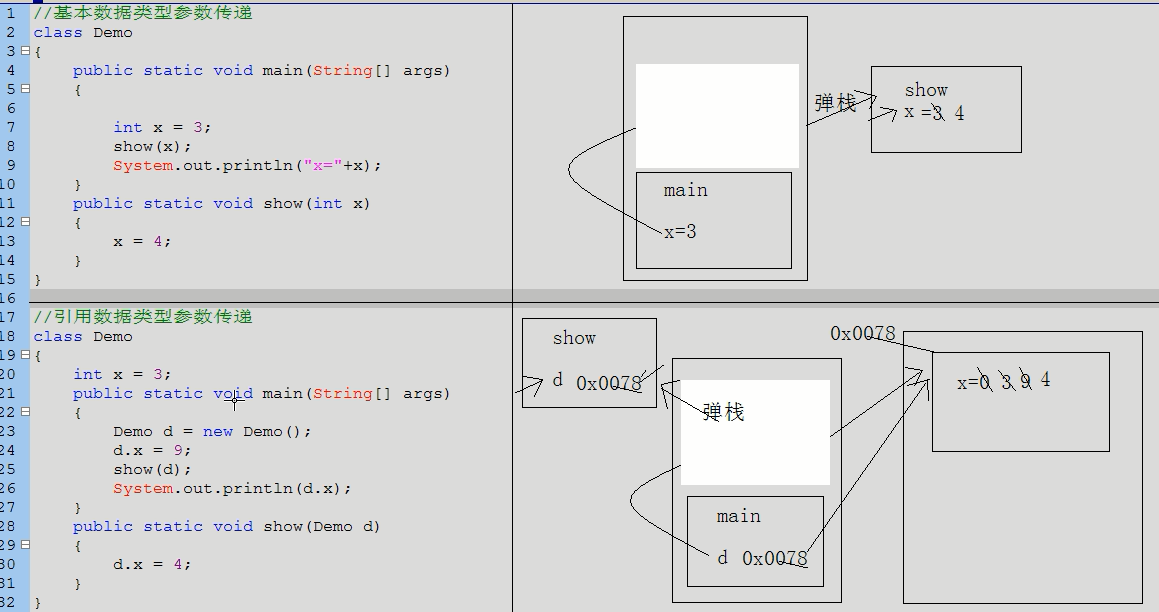
注意：在操作对象的时候，不是在操作类。

1. 匿名对象
   1. 匿名对象是定义对象的简化形式（没有名字的对象）
   2. 匿名对象两种使用情况

• 当对对象方法仅进行一次调用的时

• 匿名对象可以作为实际参数进行传递

### 三、基本/引用数 据类型的参数传递（图解）



弹栈： 执行结束后弹出栈

### 四、封装（Encapsulate）

1. 封装：是指隐藏对象的属性和实现细节，仅对外提供公共访问方式。
2. 好处：

• 将变化隔离。

• 便于使用。

• 提高重用性。

• 提高安全性

1. 封装原则：

• 将不需要对外提供的内容都隐藏起来。

• 把属性都隐藏，提供公共方法对其访问。

1. private(私有)关键字
   1. private关键字

• 是一个权限修饰符。

• 用于修饰成员(成员变量和成员函数)

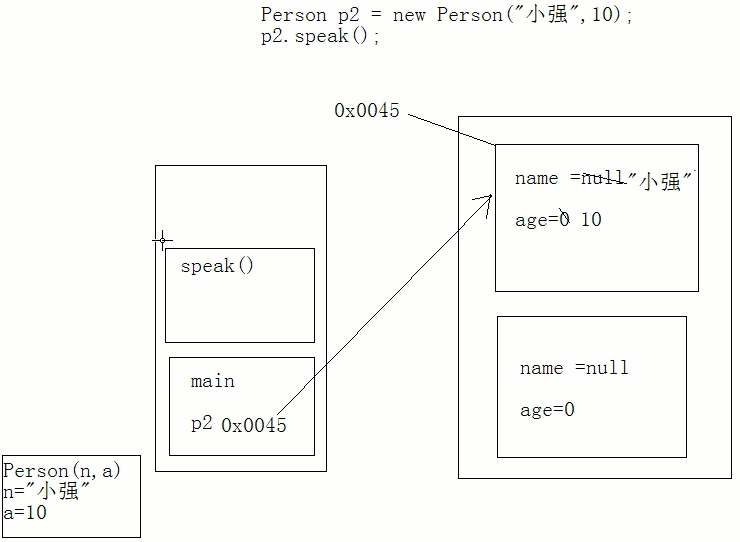
• 被私有化的成员只在本类中有效

* 1. 常用于：

讲成员变量私有化，对外提供set，get方法对其进行访问。提高对数据访问的安全性。

### 五、构造函数

1. 构造函数：构造对象时调用的函数（创建对象都必须要给其初始化）
2. 特点：
   * 函数名与类名相同
   * 不用定义返回值类型
   * 不可以写return语句
3. 作用：给对象进行初始化。
4. 一般函数与构造函数区别：
   1. 构造函数：对象创建时，就会调用与之对应的构造函数，对对象进行初始化（只能一次调用）
   2. 一般函数：对象创建后，需要函数功能是才调用（可多次调用）
5. 构造函数重载
   1. 多个构造函数是以重载的形式存在的
6. 注意：
   1. 默认构造函数的特点。
   2. 多个构造函数是以重载的形式存在的
7. 构造函数图解：



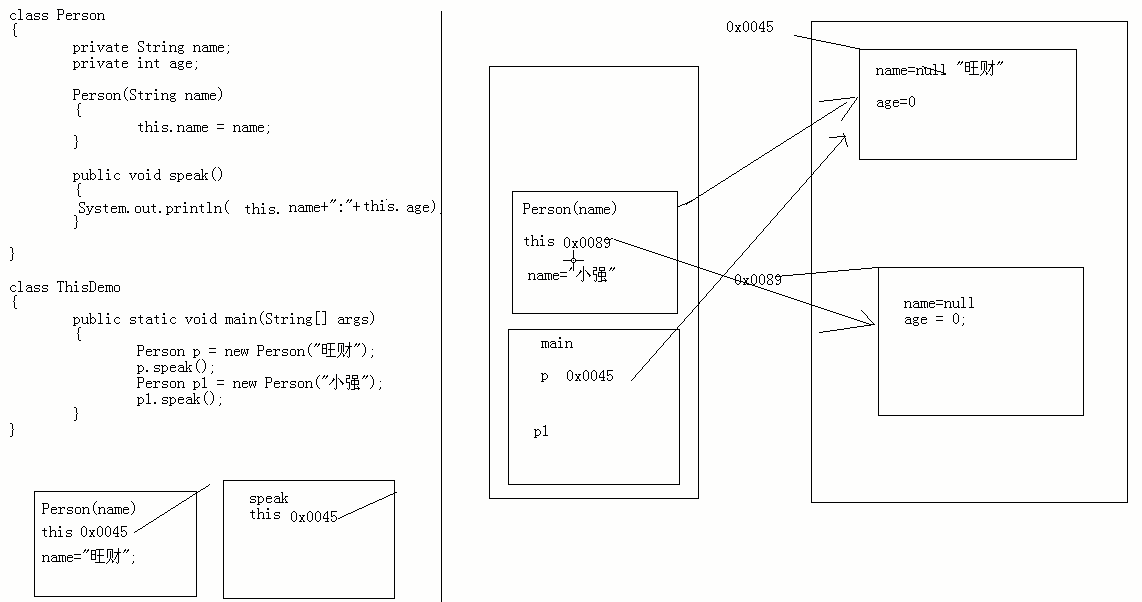
### 六、this关键字

1. 特点： this代表其所在函数所属对象的引用。

换言之： this代本类对象的引用。

简单说：哪个对象调用了this所在的函数。This就代表哪个对象。

1. 作用：当成员变量和局部变量重名，可以用this来区分。将其函数使用的同名对象指向堆内存的对象。
2. 什么时候使用this关键字呢？
   1. 当在函数内需要用到调用该函数的对象时，就用this。
   2. 用于在构造函数中调用其他函数。调用其他构造函数必须在该构造函数语句中的第一行。构造函数初始化动作后执行，函数先执行。



1. This还可以这样使用
   1. 本类对象的方法，和变量。不是静态方法的时候需要调用方法时需要加个this
   2. 哪个对象调用了该方法，则this代表了该对象

### 七、static关键字

1. static关键字：用于修饰成员（成员变量和成员函数）
2. 被修饰后的成员具备以下特点：
   1. 随着类的加载而加载
   2. 优先于对象存在（加载就存在了）
   3. 被所有对象所共享，对象中存储的是特有数据（才用）
   4. 可以直接被类名调用 Eg： 类名.静态成员
3. 成员变量和静态变量的区别：
   1. 成员变量
      1. 成员变量随着对象的创建而存在，随着对象的回收被释放
      2. 成员变量只能被对象调用。
      3. 成员变量数据存储在堆内存的对象中，所以也叫对象的特有数据
      4. 成员变量也称为： 实例变量
   2. 静态变量
      1. 静态变量随着类的加载而存在，随着类的结束而消失。
      2. 静态变量可以被对象调用，还可以被类名调用。
      3. 静态变量数据存储在方法区（共享数据区）中的静态区，所以也叫对象的共享数据
      4. 静态变量称为：类变量
4. Static使用的注意事项
   1. 静态方法只能访问静态成员（非静态即可方法静态、也可以方法非静态）
   2. 静态方法中不可以使用this或者super关键字（因为没有对象）
   3. 主函数是静态的。
5. mian主函数的特殊之处
   1. 格式是固定的
   2. 被JVM所识别和调用。

Public ： 因为权限必须是最大的

Static ： 不需要对象的

Void ： 主函数没有具体的返回值

Main ： 函数名，不是关键字，只是JVM识别的固定名称

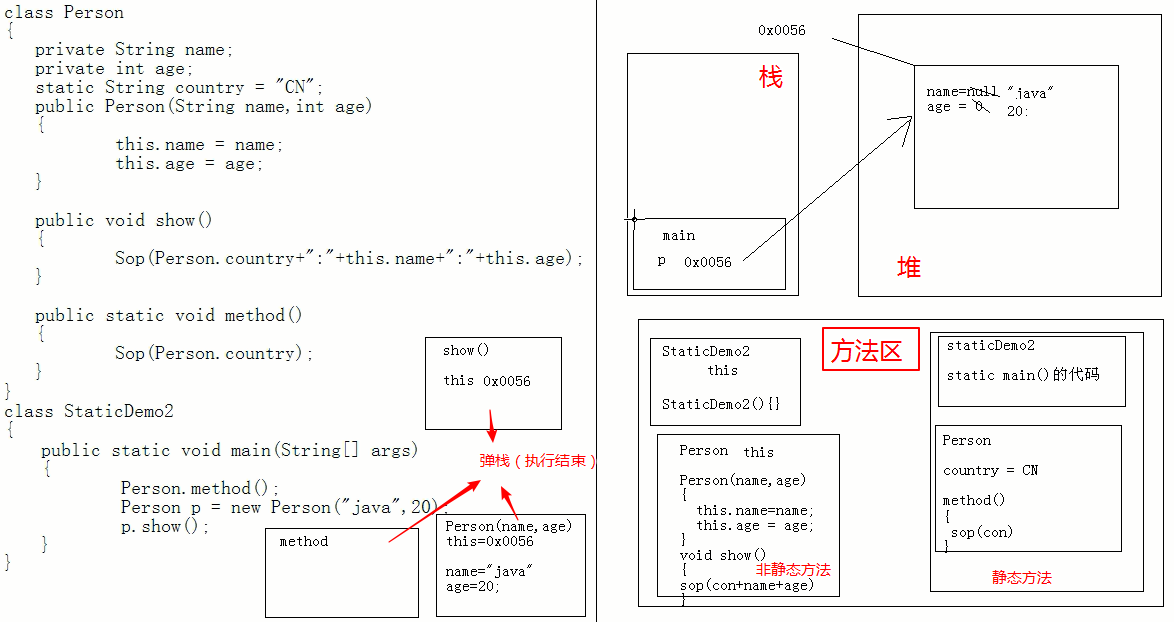
String[] args ： 这是主函数的参数列表，是一个数组类型的参数，而且元素都是字符串类型。





1. Static关键字的内存图解

Main中所有的代码都存放在方法区



1. 什么时候使用Static
   1. 静态变量
      1. 当分析对象中具备的成员变量的值都是相同的。这时成员就可以被静态修饰。

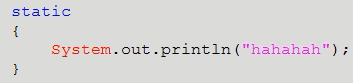
只要数据在对象中都是不相同的，就是对象的特有数据，必须存储在对象中，是非静态的。

如果数据在对象中都是相同的，对象不惜要修改，只需要使用即可，不需要存储在对象中，定义为静态变量即可。

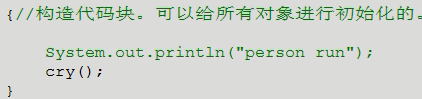
* 1. 静态函数（方法）
     1. 函数是否使用静态修饰，就参考一点，该函数功能是否有访问到对象中的特有数据。

简单来说，从源代码看，该功能是否需要访问非静态的成员变量，如果需要就，该功能就是非静态的，如果不需要，就可以定义为静态函数。当然也可以定义成非静态的，但是这样耗费内存空间（无意义）。

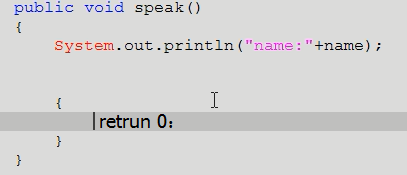
1. 静态代码块
   1. 没有名字的代码块
   2. 随着类的加载而执行。但仅执行一次。
   3. 作用：可以给类的静态变量 进行初始化。（类似构造函数）



1. 构造代码块（类中的无名代码块）
   1. 构造对象的代码块
   2. 随着对象的创建而执行
   3. 作用：是给对应的对象进行针对性初始化（共性代码）



1. 局部代码块
   1. 在函数内部的代码块
   2. 作用是验证函数的生存周期（可用做测试）



1. 三大代码块的优先执行顺序

静态代码块 🡪 构造代码块（存在创建对象）🡪 构造函数 🡪 < 局部代码块 >

1. 举例：
   1. 数组工具类

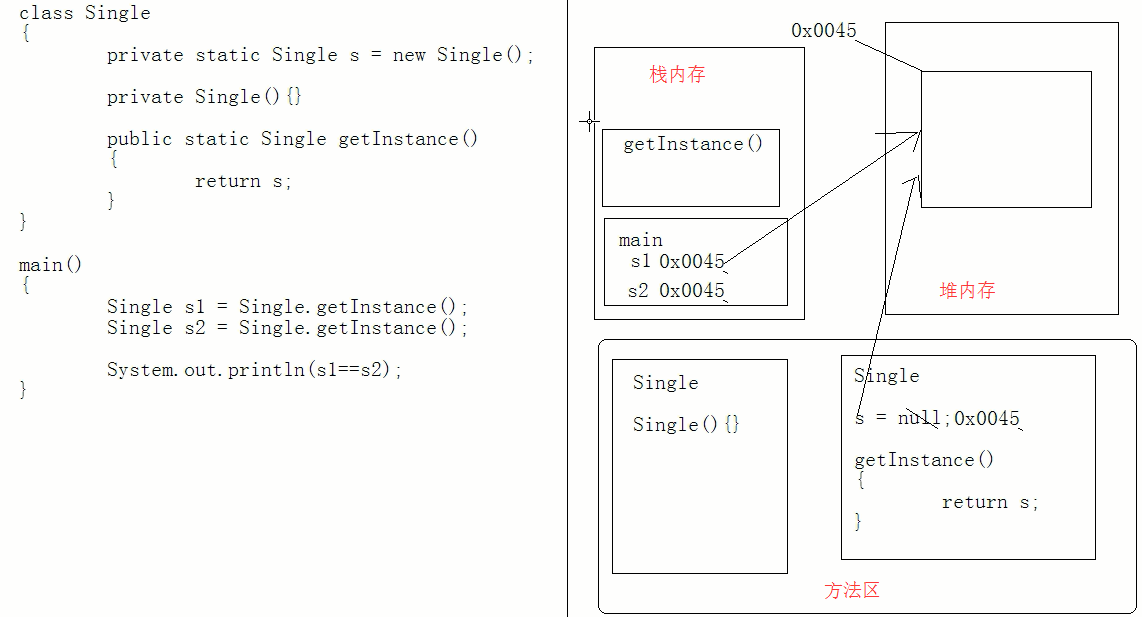
工具类的方法都是静态的，所以该类是不需要创建对象的。为了保证不然其他类对象创建工具类对象，只要将其构造函数私有化即可。

### 八、单例设计模式

1. 设计模式：对问题行之有效的解决方式，其实它是一种思想
2. 单例设计模式：
   1. 解决的问题：就是可以保证一个类在内存中的对象唯一性。
      1. 必须对于多个成语使用同一个配置信息对象时，就是需要保证该对象的唯一性
      2. 如何保证对象的唯一性：
         1. 不允许其他程序使用new创建该类对象。
         2. 通过new在本来创建一个本类对象。
         3. 定义一个公有的方法，将创建的对象返回。
      3. 饿汉式与懒汉式
         1. 饿汉式：类一加载就创建对象。
         2. 懒汉式：只有调用getInstance方法才去创建对象。

懒汉式存在的问题：多线程并发访问时不能保证其对象的唯一性。

* 1. 内存图



# Day08

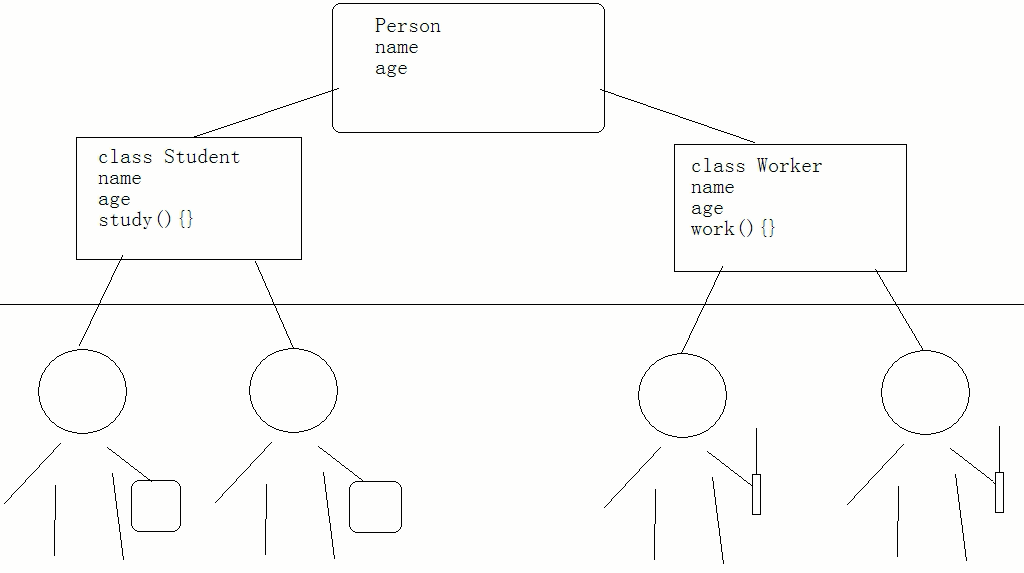
## 第四章【继承】

### 一、继承的概述

1. 多个类中相同属性和行为时，将这些内容抽取到单独一个类中，那么多个类无需再定义这些属性和行为，只要继承单独的那个类即可。
2. 多个类可以称为子类，单独这个类称为父类或者超类。
3. 子类可以直接访问父类中的非私有的属性和行为。
4. 通过 extends 关键字让类与类之间产生继承关系

Eg： class SubDemo extends Demo{}

1. 继承的好处
   1. 继承的出现提高了代码的复用性。
   2. 继承的出现让类与类之间产生了关系，提供了多态的前提。



### 二、继承的特点

1. Java只支持单继承，不支持多继承。

• 一个类只能有一个父类，不可以有多个父类。

• 单继承：一个子类有只有一个直接父类。 多继承：一个子类可以有多个直接父类

• class SubDemo extends Demo{} //ok

• class SubDemo extends Demo1,Demo2...//error

1. Java支持多层继承( 多重继承体系 )
   1. 要使用一个继承体系时
      1. 查看体系中的顶层类，了解该体系的基本功能。
      2. 创建体系中的最子类对象，完成功能的使用。

• class A{ }

• class B extends A{ }

• class C extends B{ }

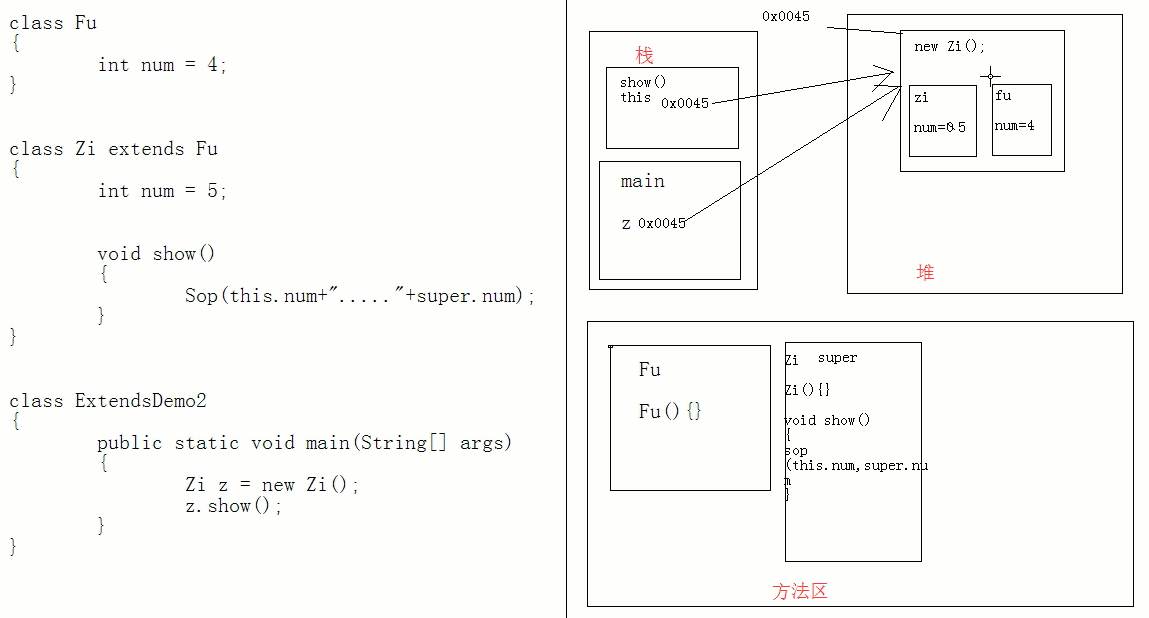
1. 什么时候定义继承呢？
   1. 类与类之间存在着所属( " is a " )关系的时候，就定义继承，xxx是yyy中的一种。xxx extends yyy
   2. 不要仅为了获取其他类中某个功能而去继承

### 三、super关键字

当本类成员和局部变量同名this区分。

当子父类中的成员变量同名使用super区分父类。

1. this ： 代表一个本类对象的引用
2. super ： 带便一个父类空间（引用）
3. 注意：子类不能直接访问父类中私有的内容，但能继承。
4. 子父类中的成员变量图解



### 四、函数覆盖

1. 子类中出现与父类一模一样的方法时，会出现覆盖操作，也称为重写或者复写。（不是重载）
2. 父类中的私有方法不可以被覆盖。
3. 在子类覆盖方法中，继续使用被覆盖的方法可以通过super.函数名获取
4. 函数的两个特性：
   1. 重载 ： 在同一个类中。Overload
   2. 覆盖 ： 子类中，覆盖也称为重写。复写
5. 覆盖注意事项：

• 覆盖时，子类方法权限一定要大于等于父类方法权限（才是覆盖）

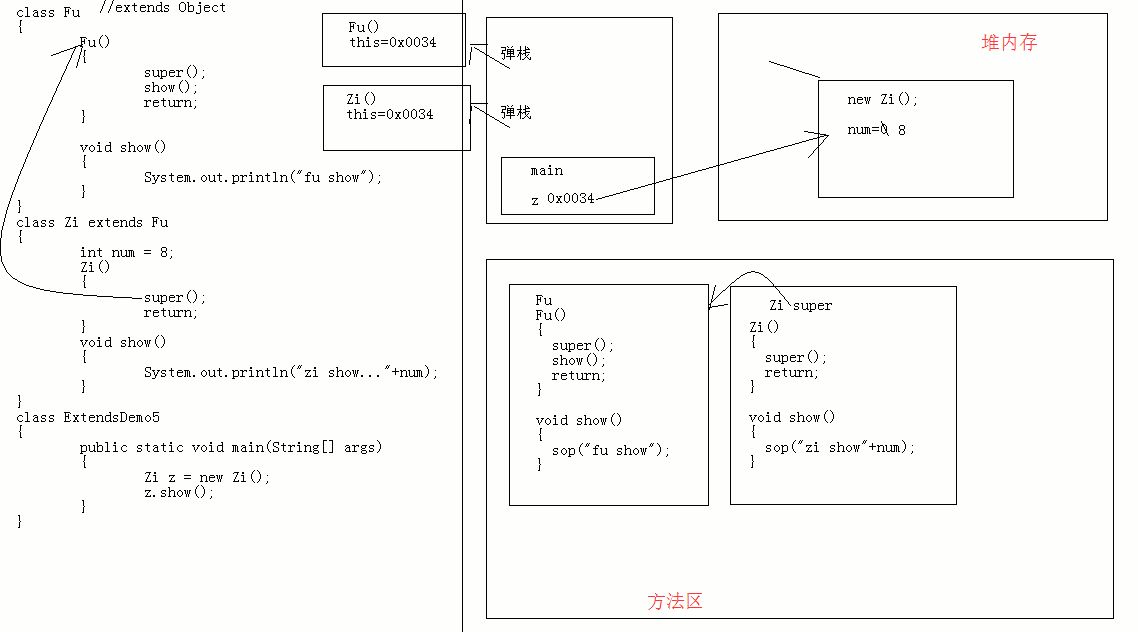
• 静态只能覆盖静态。

1. 覆盖的应用：

• 当子类需要父类的功能，而功能主体子类有自己特有内容时，可以复写父类中的方法，这样，即沿袭了父类的功能，又定义了子类特有的内容。

### 五、子类的实例化过程

1. 子父类中的构造函数的特点：
   1. 在子类构造对象时，发现，访问子类构造函数时，父类也运行了。
   2. 原因是：在子类的构造函数中第一行有个默认的隐式语句。调用了父类的构造函数super（）;
2. 子类的实例化过程：
   1. 子类中的所有构造函数默认都会访问父类中你的空参数构造函数
   2. 为什么子类实例化要访问父类中你的构造函数呢？
      1. 子类继承父类，获取到了父类中的内容（属性），所以在使用父类内容之前要看父类是如何对自己的内容进行初始化。所以子类在构造对象时，必须访问父类的构造函数。
   3. 为什么完成这个必须的动作，就在子类的构造函数加入super（）语句
      1. 如果父类中没有定义空参数构造函数，那么子类的构造函数必须使用super明确 调用。
      2. 父类中哪个构造函数，同时子类构造函数中如果使用this调用了本累构造函数是，那么super就没有了，因为super和this都只能存在第一句。所以只有一个。但是可以保证的是，子类中肯定会有其他的构造函数调用了父类的构造函数。
      3. 注意：super语句必须要定义在子类构造函数的第一句。因为父类初始化动作要比子类先完成。
3. 补充：创建一个类的时候默认继承Object类（省略不写）。隐式语句有super（）；和return；
4. 子类父类初始化有个问题需要理解：
   1. 通过super初始化父类内容是，子类的成员变量并未显示初始化，等super初始化完成后才进行子类的成员变量初始化！ 所以一开始其成员成员变量为0；
   2. 图解如下：



结果如下：

1. 一个对象实例过程：
   1. JVM读取指定路径下的Person.class文件，并加载进内存。并会先加载Person的父类（有直接父类的时候）
   2. 在堆内存中开辟空间，分配地址。
   3. 并在对象空间中，对象中的属性进行默认初始化
   4. 调用对应的构造函数进行初始化。
   5. 在构造函数中，第一行会先调用父类中构造函数进行父类初始化。
   6. 父类初始化完毕后，再对子类的属性进行显示初始化。
   7. 在进行子类构造函数的特定初始化。
   8. 初始化完毕后，将地址值赋值给引用变量。

### 六、final关键字

1. Final是一个修饰符，可以修饰类，方法，变量。
2. final修饰的类不可以被继承。
3. final修饰的方法不可以被覆盖。
4. final修饰的变量是一个常量。只能被赋值一次。定义时赋值；
   1. 注意：final固定的是显式值的初始化，而不是隐式/默认初始化值。
   2. public static final double d；这是个全局变量。
5. 内部类只能访问被final修饰的局部变量。
6. 为什么要用final修饰变量。其实在程序中如果一个数据是固定的那么就可以使用这个数据就可以了，但是这样阅读性差，所以它该数据起个名称。而且这个变量名称的值不能变化，所以加上final固定。
   1. 写法规范：常量所有字母都是大写，多个单词用“\_”隔开。

### 七、抽象类

1. 抽象类概述
   1. 抽象就是笼统、模糊，看不懂，不具体。
   2. 抽象定义：

• 抽象就是从多个事物中将共性的，本质的内容抽取出来。

• 例如：狼和狗共性都是犬科，犬科就是抽象出来的概念。

* 1. 抽象类：

• Java中可以定义没有方法体的方法，该方法的具体实现由子类完成，该方法称为抽象方法，包含抽象方法的类就是抽象类

* 1. 抽象方法的由来

• 多个对象都具备相同的功能，但是功能具体内容有所不同，那么在抽取过程中，只抽取了功能定义，并未抽取功能主体，那么只有功能声明，没有功能主体的方法称为抽象方法。

• 例如：狼和狗都有吼叫的方法，可是吼叫内容是不一样的。所以抽象出来的犬科虽然有吼叫功能，但是并不明确吼叫的细节。

1. 抽象类的特点
   1. 方法只有声明没有实现时，该方法就是抽象方法，需要被abstract修饰。抽象方法必须定义在抽象类中。该类必须也被abstract修饰。
   2. 抽象方法只有方法声明，没有方法体。

• 格式： 修饰符 abstract 返回值类型 函数名(参数列表) ;

* 1. 抽象类不可以实例化，也就是不可以用new创建对象。

• 抽象类是具体事物抽取出来的，本身是不具体的，没有对应的实例。例如：犬科是一个抽象的概念，真正存在的是狼和狗。

• 抽象方法没有方法体。而且抽象类即使创建了对象，调用抽象方法也没有意义。

* 1. 抽象类必须有其之类覆盖了所有抽象方法后，该子类才可以实例化。否则这个子类也是抽象类。

1. 抽象类举例代码讲解
2. 抽象类与一般类的区别;
   1. 相同点：
      1. 抽象类和一般类都是用来描述事务的，都在内部定了成员。
   2. 不同点
      1. 一般类都有足够的信息描述事物。

抽象类描述事物的信息不全

* + 1. 一般类中不能定义抽象方法，只能定义非抽象方法。

抽象类中可定义抽象方法，同时也可以定义非抽象方法

* + 1. 一般类可以被实例化，抽象类不可以被实例化。

抽象类不可以实例化（关键）

1. 抽象类相关问题
   1. 抽象类中有构造函数吗？

有，用于给子类对象进行初始化

* 1. 抽象类可以不定义抽象方法？

可以的，但是很少见。目的是就是不让该类创建对象。AWT适配器对象就是这类

* 1. 抽象关键字不可和哪些关键字共存？

不需要创建对象，和继承的都不行。

|  |  |  |
| --- | --- | --- |
| Private | Static | Final |
| 不行 | 不行 | 不行 |

1. 抽象类一定是个父类吗?

是的，因为需要子类覆盖其方法后才可以使用

### 八、接口（抽象类的特殊形式）

1. 定义：当一个抽象类中的方法都是抽象的时候，这是就可以将该抽象类用另一种形式定义和表示，就是 接口 interface。
2. 格式： interface { }

定义接口使用的关键字是interface而不是class

1. 接口中的成员修饰符是固定的吧。

•成员常量： public static final （全局常量）

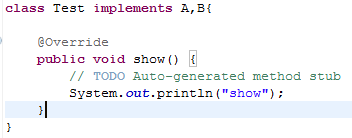
•成员函数： public abstract （抽象方法）

1. 接口中的成员都是公共权限（public）。
2. 接口是不可以实例化。

只能由实现了接口的子类并覆盖其接口中所有抽象方法后，该子类才可以实例化，否则这个子类就是一个抽象类。

1. 接口多实现

在java中不能直接支持多继承，因为出现调用的不确定性。所以java将多继承机制进行改良，在java中变成了多实现。



1. 接口细节
   1. 一个类在继承另一类的同时，还可以实现多个接口。
   2. 接口的出现避免了单继承的局限性
2. 接口的特点
   1. 接口是对外暴露的规则。
   2. 接口是程序的功能扩展。
   3. 接口的出现降低耦合性。
   4. 接口可以用来多实现。
   5. 类与接口之间是实现关系，而且类可以继承一个类的同时实现多个接口。
   6. 接口与接口之间可以有继承关系。
3. 补充

类与类之间是继承关系，

类与接口之间是实现关系（实现简写Impl）

接口与接口之间是继承关系。（接口可以多继承）

1. 接口和抽象类的区别：

相同点：都是不断向上抽取而来的

不同点如下表：

|  |  |
| --- | --- |
| 接口 | 抽象类 |
| 接口需要被实现，而且可以多实现 | 抽象类需要被继承，而且只能单继承 |
| 接口只能定义抽象方法，必须有子类去实现 | 抽象类中可以定义抽象方法，子类可使用非抽象方法 |
| 接口的实现是 like a 关系，在定义体系额外功能 | 抽象类的继承，是is a 关系，在定义该体系的基本共性内容。 |

1. 接口的应用
   1. 接口类型的引用，用于接收（指向）接口的子类的对象

### 九、多态

1. 定义：某一个类事物的多种存在形态。
   1. Eg： 动物类中的猫和狗

猫这类事务即具备猫的形态，又具备着动物的形态。这就是对象的多态性

简单说：就是一个对象对应着不同类型。

• 动物 y = new 猫();

• 动物是猫和狗具体事物中抽取出来的父类型。

• 父类型引用指向了子类对象。

1. 多态在代码中的体现：

父类或接口的引用指向其子类的对象。

1. 多态的好处：

提高代码的扩展性，前期定义的代码可以使用后期的内容

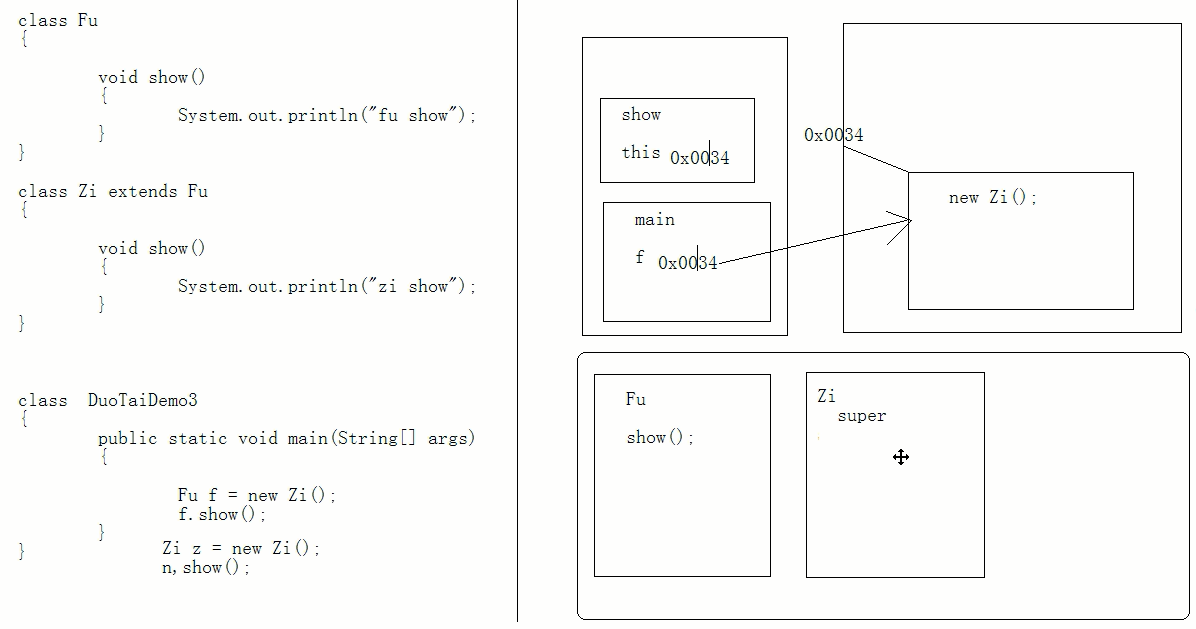
1. 多态的弊端：

前期定义的内容不能使用（）后期子类特有的内容。（使用本类的内容）

1. 多态的前提：
   1. 必须有关系，继承，实现
   2. 要有覆盖（方法的覆盖）
2. 多态的转型
   1. 向上转型
      1. 自动类型提升，猫对象提升为动物类型，但是特有功能无法访问
      2. 其作用是：限制对特有功能的访问。
      3. 即是将子类对象转为父类对象。此处父类对象可以是接口。
   2. 向下转型
      1. ，将动物类强制转为猫类型，使用猫的特有功能（前提是有向上转型）
      2. 作用：使用子类中的特有方法
      3. 与向上转型相反，即是把父类对象转为子类对象。
   3. 注意：对于转型，至始至终都是子类对象在做类型变化。父类对象不行。
3. 类型的判断
   1. Instanceof ：用于判断对象的具体类型，只能用于引用数据类型的判断
   2. 用法： 对象 instanceof 类型

Eg： 

1. 多态时，成员的特点：
   1. 成员变量
      1. 编译时：参考引用型变量的所属的类中是否有调用的成员变量，有，编译通过，反之失败
      2. 运行时：参考引用型变量的所属的类中是否有调用的成员变量，并运行该所属类的成员变量
      3. 简单理解：编译和运行都参考等号的左边。
   2. 成员函数
      1. 编译时：参考引用型变量的所属的类中是否有调用的函数，有，编译通过，反之失败
      2. 运行时：参考的是对象所属的类中是否有调用函数。
      3. 简单理解：编译看左边，运行看右边



* 1. 静态函数
     1. 个人认为，因为多态性是对象特有的，而静态方法是不需要对象存在的所以静态不涉及到对象。
     2. 编译时：参考引用型变量的所属的类中是否有调用的静态方法。
     3. 运行时：参考引用型变量的所属的类中是否有调用的静态方法
     4. 简单说：编译运行都看左边。

其实对于静态方法，是不需要对象的，直接用类名调用。

### 十、内部类

1. 将一个类定义在另一个类里面，对里面那个类就称为内部类（内置类，嵌套类）、
2. 访问特点：

• 内部类可以直接访问外部类中的成员，包括私有成员。

• 而外部类要访问内部类中的成员必须要建立内部类的对象。

1. 用途：

分析事物是，发现该事物描述中还有事物，而且这个事物还在访问被描述事物的内容。这时就是还有的事物定义在内部类来描述。

1. 内部类修饰
   1. 内部类定义在成员位置上

• 可以被private static成员修饰符修饰。

• 被static修饰的内部类只能访问外部类中的静态成员。

* 1. 内部类定义在局部位置上

• 也可以直接访问外部类中的成员。

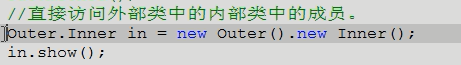
• 同时可以访问所在局部中的局部变量，但必须是被final修饰的。

* 1. 如果内部类是静态的，相当于一个外部类。
  2. 如果内部类是静态的，成员也是静态的。

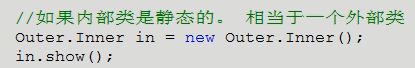
1. 内部类直接访问外部类中成员

内部类持有外部类的引用。 格式： 外部类名.this（默认的）你

1. 直接访问内部类中成员
   1. 内部类是非静态类，成员是非静态



* 1. 内部类是静态类，成员是非静态



* 1. 内部类是静态类，成员是静态



注意：当内部类的成员是静态的，则内部类一定要被静态修饰。

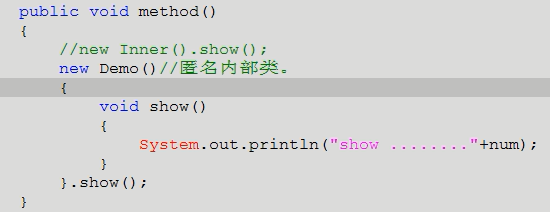
1. 内部类可以放在局部变量的位置上

内部类在局部位置上只能访问局部中被final修饰的局部变量

### 十一、匿名内部类

1. 内部类的简写格式
   1. 必须有前提：内部类必须继承或者实现一个外部类或者接口
   2. 匿名内部类其实就是子类对象

格式：new 父类 or 接口 （）{ 子类内容 }



* 1. 简单理解： 就是建立一个带内容的外部类或者接口的子类 匿名对象

1. 匿名内部类缺点：不能减少代码的复用性（每次写一个匿名内部类只能使用一次！））
2. 使用场景：
   1. 当函数参数是接口类型时，而且接口中的方法不超过三个。可以使用匿名内部类作为实际参数进行传递。

### 十二、Object类

1. Object类是所有类的根类，是不断抽取而来的，具备所有对象都具备的共性内容
2. equals（）方法
   1. 一般都会覆盖此方法，根据对象的特有内容吗，建立判断对象是否相同的依据
3. 查询object方法用api文档’

### 十三、异常

1. 异常：是在运行时期发生的不正常情况。
2. 在java中用类的形式对不正常情况进行了描述和封装对象。

**描述不正常情况的类，称为：异常类**

以前正常流程代码和问题处理代码的结合

现在将正常的流程代码和问题分离，提高代码的阅读性

（其实异常就是java通过面向对象思想将问题封装成了对象，用异常类来对其描述）

1. 不同的问题用不同的类进行描述。比如：脚标越界，空指针等等。

问题很多，意味着描述的类也很多，将其共性向上抽取，形成了异常体系。

1. 最终问题（不正常情况）就分层了两大类。

**throwable: ----可抛出类**

该体系的特点throwable及其所有的子类都具有可抛性。

可抛性是什么？怎么体现可抛性。

通过两个关键字可以体现出来：throws、throw，凡是被这两个关键字所操作的类和对象都是可抛  
 • thorws用在函数上，后面跟异常类名。

• throw用在函数内，后面跟异常对象

* 1. **一般不可以处理的（error）**

特点：是由JVM抛出的严重性问题，这种问题发生一般不针对性处理，直接修改程序

* 1. **可以处理的（Exception）**

该体系的特点：

子类的后缀名都是用其父类名作为后缀，阅读性很好

1. 异常处理原理
   1. 判断异常区间是否出现异常
   2. 出现异常则new一个对应的异常类
   3. 将异常类抛出

补充：异常类提示可以自定义。

1. （**自定义异常**）问题引出解决方案：
   1. 对于角标都是整数不存在，可以用角标越界表示，对于负数角标的情况，准备用负数角标异常来表示。
   2. 负数角标这种异常类在java中并没有定义过。所以安装java异常的创建思想，面向对象，将负数角标镜像自定义描述。并封装成对象。
   3. 这种解决方案就是自定义异常。

注意：如果让一个类称为异常类，必须要继承异常类体系，因为只有称为异常体系的子类才有资格具备有可抛性。才可以被两个关键字所操作 throw throws

1. 自定义异常注意事项
   1. 定义异常类后使用其异常类的时候要用throw 异常类名 加载函数后



1. 异常的分类：
   1. **编译时被检测异常**：只要是Exception和其子类都是，出了特殊子类RuntimeException体系
      1. 这种问题一旦出现，就希望在编译时进行检测，让这个问题有对应的处理方法，

这样的问题都可以针对性处理。

* 1. **编译时不检测异常**（正常运行时异常）：就是Exception中的RuntimeExceptin和其子类
     1. 这种问题的发生，无法让功能继续，运算无法进行，更多是因为调用者的原因导致的而火者引发了内部状态的改变的。

那么这种问题一般不处理，直接编译通过，在运行是让调用者调用时的程序强制停止。让调用者对代码进行修正。

1. throw 和 throws的区别
   1. throws是使用在函数上

throw是使用在函数内

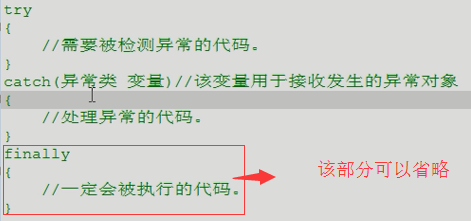
* 1. throws抛出的是异常类，可以抛出多个，用逗号隔开

throw是抛出异常对象

1. 异常处理的捕捉

这是可以对异常进行针对性处理的方式

* 1. 捕捉格式：



* 1. 何时使用异常捕捉
     1. 当问题能够解决的时候我们就用try
     2. 当问题不能解决的时候我们就抛出该父级处理
  2. 捕捉和抛出区别
     1. 捕捉：处理完成后程序继续运行
     2. 抛出：整个程序在异常出直接终止运行
  3. 多异常-多catch

1. 异常处理原则
   1. 函数内容如果抛出需要进程的异常，那么函数闪必须要声明。否则必须在函数内使用try catch，不然编译失败
   2. 如果调用到了声明异常函数，那么try catch要么throw，否则编译失败。
   3. 什么时候catch，什么时候throw呢？
      1. 功能内容可以解决使用catch
      2. 解决不了的使用throw告诉调用者，有调用者处理
   4. 一个功能如果抛出了多个异常，那么调用时，必须有多个catch进行针对性处理内部有几个需要检测的异常，就抛出几个异常，抛出几个就catch几个
2. finally执行语句

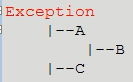
该代码块是发生异常的时候一定会执行的

常用于关闭系统资源所用

1. try-catch-finally组合
   1. try-catch-finally 存在资源需要释放
   2. try-catch 没有资源需要释放的时候不需要定义finally
   3. try-finally 发生异常不处理，但需要关闭资源
2. 异常应用

老师上课，电脑出问题。

1. 异常的注意实事项
   1. 子类覆盖父类方法时，如果父类的方法抛出了异常，那么子类的方法只能抛出父类的异常或者异常的子类



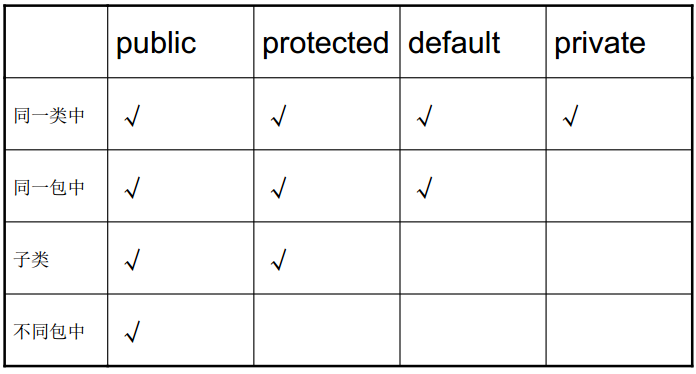
* 1. 如果父类抛出多个异常，那么子类只能抛出父类异常的子集

简单说来：子类覆盖父类只能抛出父类的异常或者子类或者子集

注意：如果父类的方法没有抛出异常，那么子类覆盖时绝对不能抛，只能try

### 十四、包（package）

1. 包
   1. 对类文件进行分类管理。
   2. 给类提供多层命名空间。
   3. 写在程序文件的第一行。
   4. 类名的全称的是 包名.类名
   5. 包也是一种封装形式。
2. 包之间的访问
   1. 被访问的包中的类权限必须是public的。
   2. 类中的成员权限： public或者protected
3. 访问权限



1. 导包的原则：用到哪个类，就导入哪个类
2. Import
   1. 一个程序文件中只有一个package，可以有多个import。
   2. 用来导包中的类，不导入包中的包。
   3. 通常写import mypack.Demo;
3. Jar：Java的压缩包
4. Jar操作

通过jar.exe工具对jar的操作。

* 1. 创建jar包

• jar -cvf mypack.jar packa packb

* 1. 查看jar包

• jar -tvf mypack.jar [>定向文件]

* 1. 解压缩

• jar -xvf mypack.jar

* 1. 自定义jar包的清单文件

• jar –cvfm mypack.jar mf.txt packa packb

# Day09

## 第五章【多线程】

### 一、进程

1. 进程：正在进行中的程序（直意 ）
2. 功能：为线程划分执行空间

### 二、线程

1. 线程：就是进程中一个负责程序的执行的控制单元（执行路径）。
2. 特点
   1. 一个进程中可以有多个执行路径/执行的控制单元
   2. 一个进程中至少要有一个线程
   3. 开启多个线程就是为了同时运行多部分代码。
   4. 每一个线程都有自己的运行的内容，这个内容可以称为线程要执行的任务

### 三、好处与弊端

1. 多线程好处：解决了多部分同时运行的问题
2. 多线程弊端：线程太多会导致效率降低

其实应用程序的执行都是CPU在做着快速切换完成的，这个切换是随机的

### 四、JVM中的多线程解析

JVM启动时候启动了多个线程，至少有两个线程可以分析出来。

1. 执行main函数的线程、
   1. 该线程的任务代码都定义在main函数中。
2. 负责垃圾回收的线程
   1. 该线程的任务代码在垃圾回收机制函数中。
   2. 每个对象都有垃圾回收器调用finalize()方法，进行垃圾回收。

### 五、如何创建一个线程呢？

创建线程的目的就是为了开启一条路径，去运行指定的代码和其他的代码实现同时运行。而运行的指定代码就是这个执行路径（控制单元）的任务。JVM创建的主线程的任务都定义在主函数中。而自定义的线程它的任务在哪里呢？对就是在run（）方法中。

Thread类用于描述线程，线程是需要任务的，所以Thread类也对任务的描述。这个任务就是通过Thread类中的run方法来体现，也就是说，run方法就是封装自定义线程运行任务。开启线程是为了运行指定代码，所以只有继承Thread类，并复写run方法。才能运行写run方法里面的自定义代码。

* 创建线程方式一：继承Thread类

步骤：

1. 定义一个类继承Thread类。
2. 覆盖Thread类的run方法。
3. 直接创建Thread子类对象创建现场
4. 调用start方法开启线程并调用线程的任务run（）方法

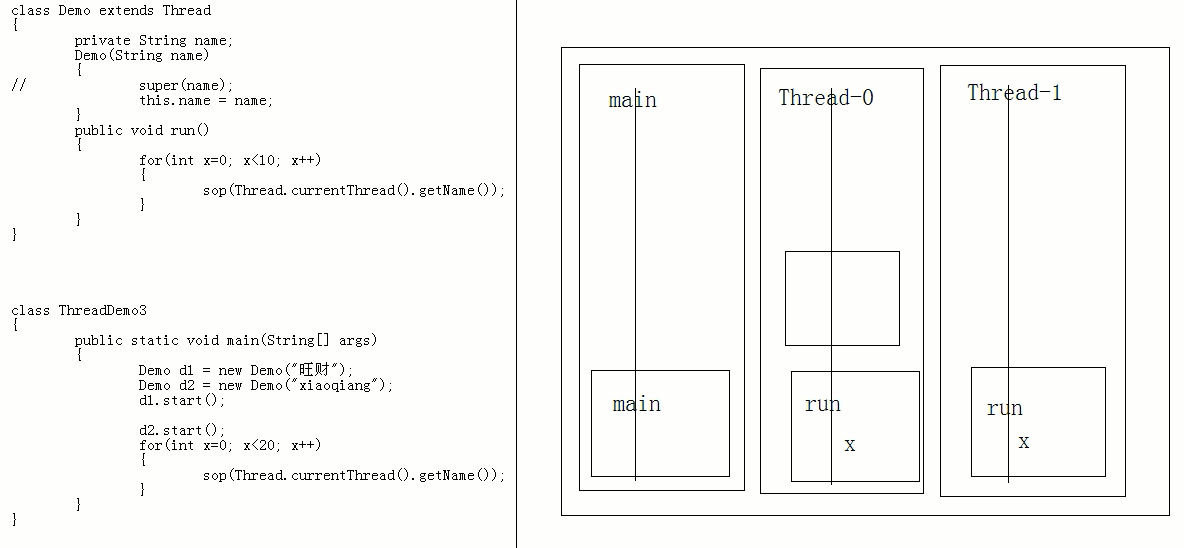
* 创建线程方式二：实现Runnable接口

步骤：

1. 定义类实现Runnable接口
2. 覆盖接口中的run方法，将线程的任务代码封装到run方法中。
3. 通过Thread类创建线程对象，并将Runnable接口的子类对象作为构造函数的参数进行传递。
4. 调用线程对象的start方法开启线程。
5. 它的出现仅仅是键线程的任务进行了对象封装。
6. 好处：
   1. 将线程的任务从线程的子类中分离出来，进行了单独的封装。按照面向对象的思想将任务封装成对象。
   2. 避免了java单继承的局限性。

注意：创建线程的第二种方式比较常见

### 六、多线程运行图解-栈

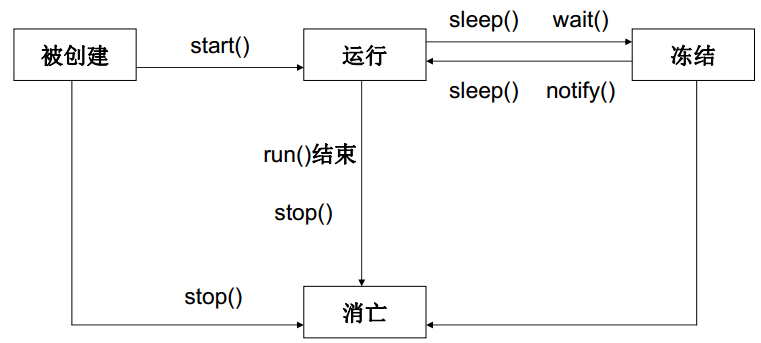


注意：

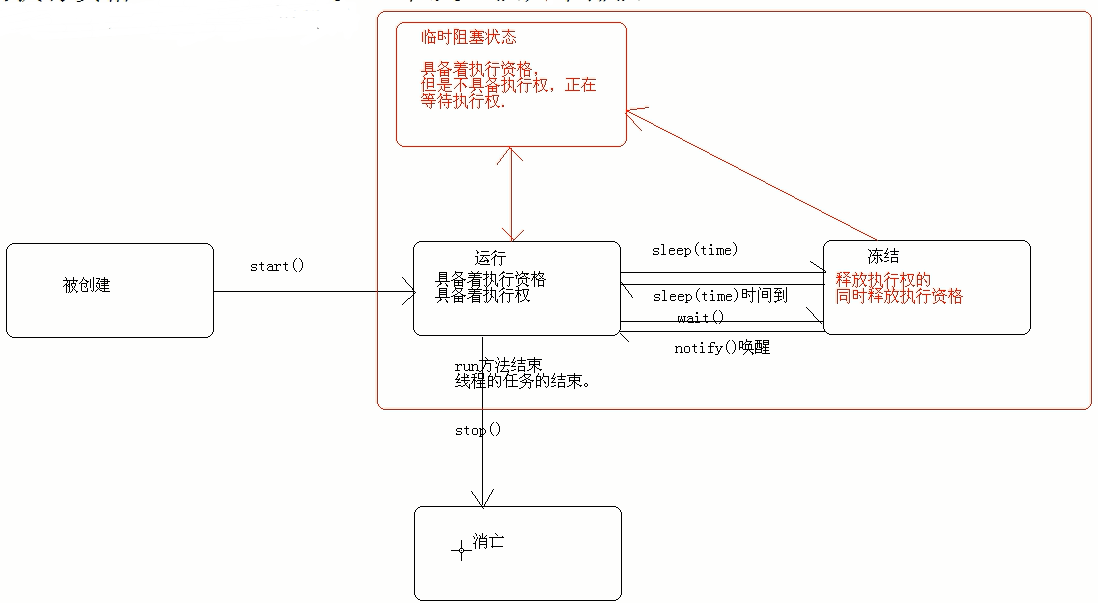
1. 每一个线程都有自己独立的栈空间
2. 除了main栈意外自定义的线程的run方法执行结束后栈空间被释放

### 七、多线程的状态

官方图：



老师画的：



CPU的执行资格：可以被CPU的处理，在处理队列中排队。

CPU的执行权：正在被CPU的处理

### 八、线程安全问题产生的原因（引出同步）

线程出现安全问题前提：

1. 多个线程再操作共享的数据
2. 操作共享数据的线程代码有多条

当以个线程在执行操作共享数据的多条代码过程中，其他线程参与运算。就会导致线程安全问题的产生

解决思路：

1. 就是将多条操作共享数据的线程代码进行封装起来，当有线程再执行这些代码块的时候，其他线程是不可以参与运算的。必须等待正在执行该代码块的线程执行完才能让其他线程才能参与运算
2. 在java中可以使用同步代码块，synchronized
   1. 格式：

Synchronize( 对象 ) 一般所以用obj类对象（对象锁）

{

需要被同步的代码

}

### 九、同步（synchronized）

1. 格式：

synchronized（对象）

{

需要同步的代码；

}

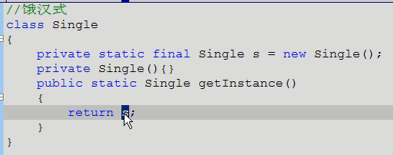
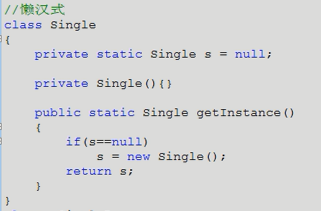
1. 同步可以解决安全问题的根本原因就在那个对象上。该对象如同锁的功能。
2. 同步的前提：
   1. 同步需要两个或者两个以上的线程。
   2. 多个线程使用的是同一个锁。

未满足这两个条件，不能称其为同步。

1. 同步的弊端：
   1. 当线程相当多时，因为每个线程都会去判断同步上的锁，这是很耗费资源的，无形中会降低程序的运行效率
2. 同步的好处：
   1. 解决了线程的安全问题
3. 同步使用方案
   1. 同步代码块
   2. 同步函数
4. 同步代码块和同步函数的区别
   1. 同步函数的锁是固定的：this
   2. 同步代码块的锁是任意对象。（开发经常使用，因为是同步函数的简写）

建议使用同步代码块。

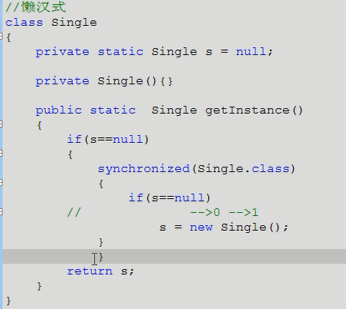
1. 多线程单例设计模式
   1. 懒汉式
   2. 饿汉式



当单例在多线程时哪些会引起隐患？

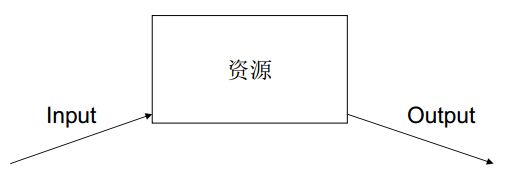
懒汉式会引起多线程问题，引发single new多个对象、解决该方法如下：

双重判断加锁（面试会面试到）



1. 同步的死锁
   1. 同步嵌套导致。

### 十、多线程间通信

示例：

多个线程在处理同一个资源，但是两个任务(input、ouput)却不同

1. 等待唤醒机制
   1. wait() 让线程处于冻结状态，被wait的线程会倍存储到线程池中。
   2. notify() 唤醒线程池中一个线程（任意）
   3. notifyAll() 唤醒线程池中你的所有线程

这些方法必须都定义在同步中，因为这些方法是用于操作线程状态的方法，必须要明确到底操作哪个锁上的线程。

1. 为什么操作线程的方法定义在了Object类中？

因为这些方法是监视器的方法，监视器其实就是锁。锁可以是任意对象，所以定义在object是最合适的。

### 一、进程