一

Jdk：开发安装包，包含JRR

JRE:java运行环境，包含JVM

配置环境变量是为了去执行bin下面的javac.exe等指令，只有配置了路径才能找到指令，否则无法执行指令，除非切换到指令所在的目录

如：在 c> java.exe 报错 （因为没配置环境变量）

Cd 到 c：\dev\jdk1.8\bin>java.exe 可以执行

配置了环境变量是为了可以在任何目录下面都去执行javac.exe等指令

Path：windows执行命令时需要搜寻的路径

1. // 表示单行注释
2. /\*

表示多行注释 不可以嵌套使用

\*/

3.

/\*\*

文档注释 可以被javadoc解析

\*/

关键字,保留字

关键字：都是小写 class，if，private，int等

保留字：现在的java版本没使用，以后可能会用比如： goto,const

标识符：对变量，方法，类等命名的字符序列

1. 由26个英文大小写，0-9，\_或$组成
2. 数字不可以开头
3. 不能使用关键字和保留字
4. 区分大小写 如static（关键字）与Static（标识符）不同
5. 不含空格
6. 包名都小写
7. 类名，接口的首字母大写
8. 变量，方法第二个单词大写 常量全部大写

变量：包含变量类型，变量值和存储的值

数据类型：基本数据类型（八种），引用数据类型（类，接口，数组）

Byte为一个字节，为八位，8bit，每一位都有0,1二种表示，所以组合起来为2的八次方，所以范围是0~256 为了区分正负，所以范围是-128 ~ 127

Byte（字节）：-128 ~ 127 byte a1 = -128，byte b2 = 12 ，byte b3 = 128（超出范围）

Char（字符）: (1字符=2字节) char a =’a’ char=‘张’ char c = ‘bm’(“错误”) char里面只能有一个字符

Char c5 = ‘\n’ 换行

字符集：

内存或则硬盘都是存储的01010101等二进制

相当于对应法则 如：1. 中（A） --- > 字符集（65） --🡪变成电脑能懂的二进制（00000001）

电脑能懂的二进制--🡪字符集---🡪变成汉子或则数字符号等

代码编写java文件有中文 sout（“你好中国”），我们设置字符集为UTF-8，保存硬盘进去假设为1234

命令行 读取java文件采用GBK，找不到1234，就会出现乱码

自动类型提升：低范围数据类型可以自动转为高范围数据类型

Byte：一个字节（8位），char：二个字节（16位）

Byte,short,char(这三个变量做运算都只能用int接收)--🡪int --🡪long--🡪float-->double

Java做运算的时候，如果操作数据范围在int范围内，那么一律按int的空间内运算

整型常量默认时int类型 如 byte a = 1；int b = a + 3；（3 为int类型，所以需要用int接收）

浮点型默认为double 如：byte a = 1；double b = a + 3.2 (……)

String不能强转为基本数据类型

进制见的转换

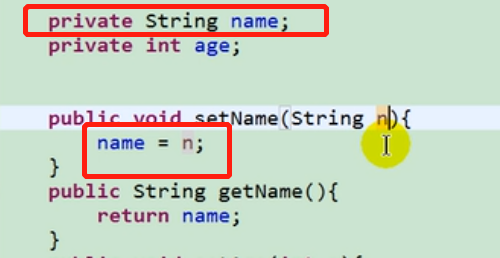
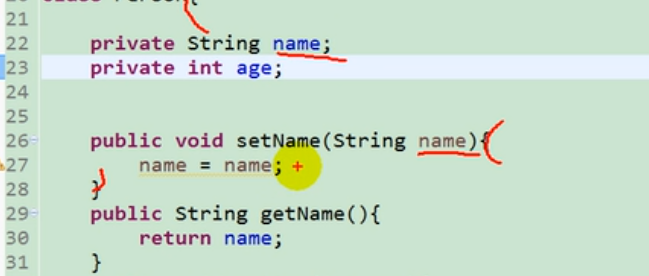
二进制：0,1 以0b或0B开头（计算机采用）

十进制：0-9，（平时工作计算中采用）

八进制：0-7以数字0开头

十六进制：0-9及A-F 以0x或0X开头

关键字：



1. 对于图2 由于name一个是全局变量，一个是在方面里面的，级别不一样，所以不报错
2. 对于图1 由于name从方法里面找不到，向上有全局变量可以找到，因此尽管不加this，name就是代表全局变量name，对于图2，由于name可以在方法里面找到，所以不加this就代表的是本方法里面的name，无法指向全局变量name（方法也是这样，因为方法名字一般来说不重载不相同，重载的情况下参数不一样，所以在类里面调用方法不需要加this）

This表示当前类的对象（A a = new A（） this相当于a）

Return：在方法中使用 1.结束方法 2.针对有返回值的方法，reutrn 数据 3.return后面不能有执行语句

Break ：结束循环

Super：调用父类的全局属性，方法，构造器 。子父类声明了相同的属性值，如果想在子类中调用父类中的属性需要用super

Instance of： a instanceof A：判断对象a是否是类A的实例，如果是 返回true（用在向下转型的时候做判断，看对象是否属于类）

Final：1.修饰类string，System，stringBuffer（线程安全的）等 2.修饰方法不能被重写

3.修饰变量，此时的变量变成了常量，不可以再进行改变（变量分为属性和局部变量）

异常：

Error：如OOM，jvm无法解决的严重问题

Exception：性能问题1.空指针2.数组下标越界3.网络连接中断

Exception：编译时异常（受检异常）与运行时异常(非受检异常)

处理方式：

Try—catch—finally

集合数组：

数组里面的算法：

JVM：

方法区：常量池，静态域（静态变量等）,以及类加载信息

堆：1.new出来的结构以及数组 A a = new A() A中的属性存在于堆中，即A中全局变量存在堆中，但A中方法以压栈的方式成为栈帧保存在栈中 2.数组

栈：局部变量

类变量（static）和实例变量内存分析：

Static：存在方法区中的静态域中，随类的加载而加载

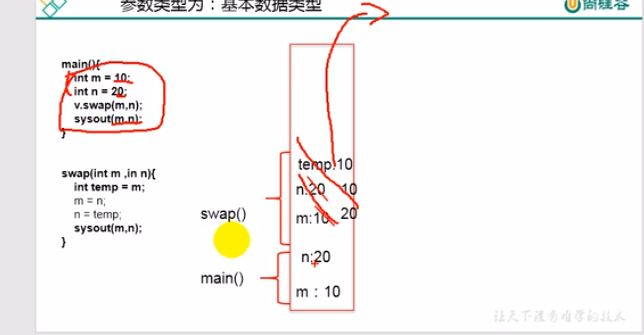
类加载器加载.class文件--🡪类信息加载到内存区域

方法区和堆一个进程一份，多个线程共享（使得线程间通信更简便,高效。也带来了数据安全问题）

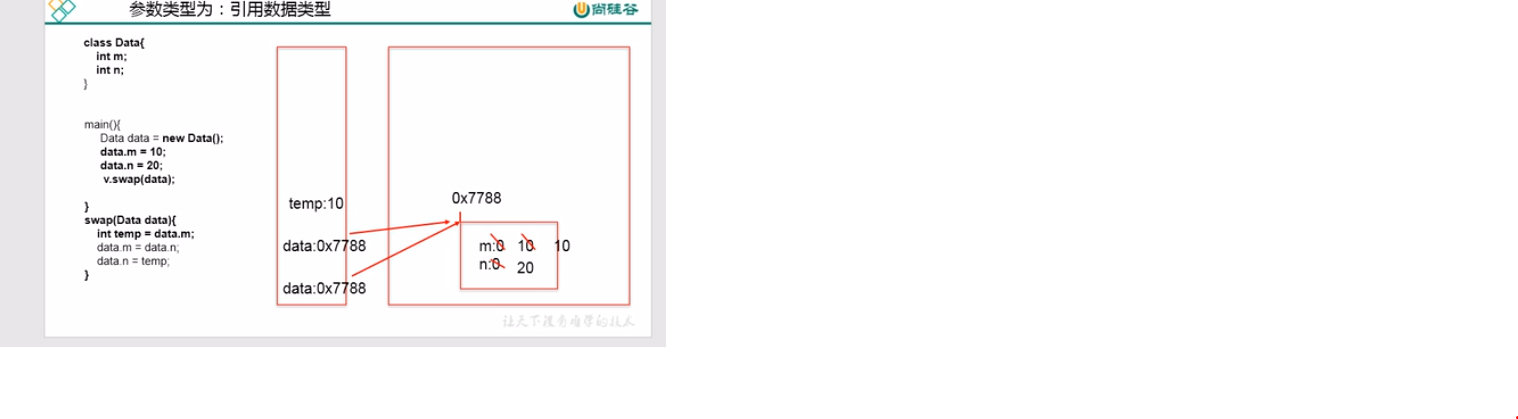
虚拟机栈，本地方法栈，程序计数器一个线程一份

值传递

<https://www.bilibili.com/video/av50420483/?p=195>



1. 基本数据类型的传递，第一个syso输出的是main方法里面的变量，第二个syso输出的是swap方法里面的变量

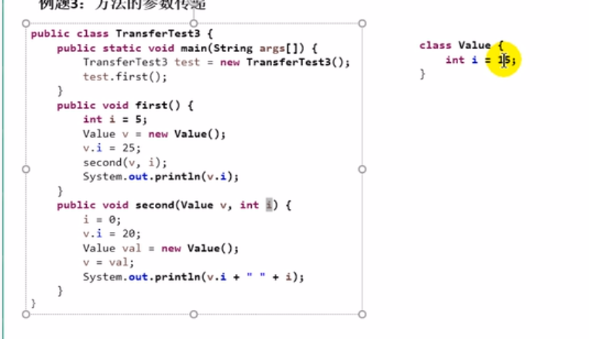
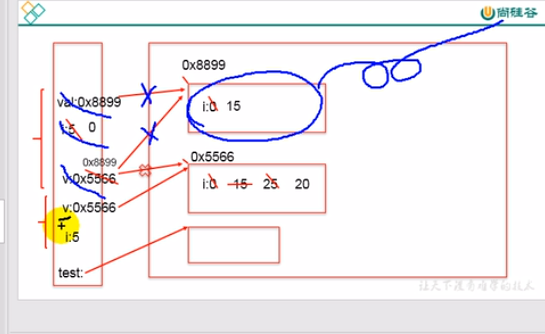


1. 引用数据类型，指向的是堆，所以swap改变了

如果参数是基本数据类型，实参赋给形参的是实参真实存储的数据值

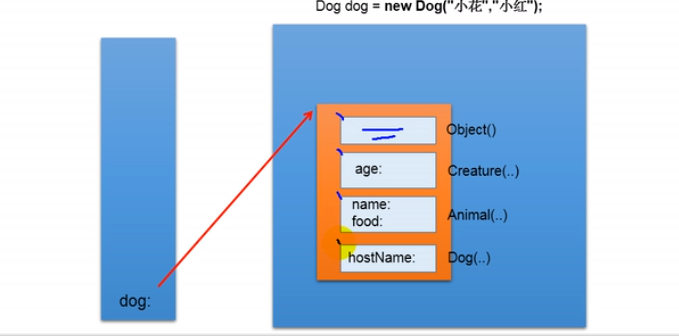
如果参数是引用数据类型，实参赋给形参的是实参存储的地址值

案列一：

继承，封装，多态

子类对象实例化的过程：创建子类对象的时虽然调用了父类构造器，但是只创建了一个对象，即为new出来的子类对象，但是获取父类的属性和方法，如下图所示，之创建了一个Dog对象



包装类的使用

<https://www.bilibili.com/video/av50420483/?p=288>

1. 基本数据类型--------🡪包装类

Int a = 3； Integer mm = new Integer(a);…..其他包装类相同

1. 包装类--------🡪基本数据类型

Integer in1 = new Integer(12); int i2 = in1.intValue()

Float f1 = new Float(12.3) ; float f2 = f1.floatValue()….其他相同

自动装箱与拆箱（jdk5.0新特性）：

3.1 int num1 = 10； Integer num2 = num1；自动装箱

3.2 Integer num3 = 50； int num5 = num3； 自动拆箱

3.基本数据类型转string 一：string snum = string.valueof( 基本数据类型)

二：string smun2 = 基本数据类型+“”；

Object obj = true？new Interger（1.0）：new Double（2.0） ====》 obj = 1.0

因为编译的时候，三元运算符会做类型的提升int转为double

Integer：中有内部类 IntegerCache-----🡪有一个数组存了-128~127这些范围的数，

即这些范围数字不需要造（用的时候已经造好），直接从缓存拿（提升了造数据的时间），不需要再去new

<https://www.bilibili.com/video/av50420483/?p=294>

类结构：

代码块：1.用来初始化类和对象 2.代码块如果有修饰只能用static（静态代码块，没加修饰是非静态代码块）3.静态代码块随着类的加载而加载，并且会去执行里面的逻辑 4. 非静态代码块随着对象的创建而加载，并且会去执行里面的逻辑 4.创建多个对象非静态代码块就执行多次，而静态代码块只执行一次

异常（抓，抛模型）

处理方式 （抓）1. Try catch finally 2.throws（作用在方法声明处）处理异常

1. 多个catch存在，如果catch没有子父类关系，无所谓先后顺序，如果满足子父类，则子类catch需在父类catch之前
2. Catch处理完异常，程序继续向下执行
3. Finally中如果有return，那么执行完finally中的return之后，将结束程序的执行
4. 释放资源需要放在finally中，确保被释放
5. 子类重写父类的异常方法throws抛出的异常比父类要小

处理方式（抛）throw，手动产生异常

1. throw new runtimeException 运行时异常不需要处理
2. throw new Exception 异常，需要用throws来处理

线程相关

程序：某种语言编写的一组指令的集合，可以看到静态的代码组成的app或则文件或则应用

进行：程序的一次执行过程（程序是静态的，进程是动态的）

线程：进程某一个执行的操作/路径， 线程作为调度和执行的单位

单核多核CPU衍生出来并发与并行：并行-🡪多个cpu同时执行多个任务

并发：一个CPU同时执行多个任务（采用时间片）

Start作用：1.启动线程 2.调用当前线程的run方法（子类重写的ruun方法）

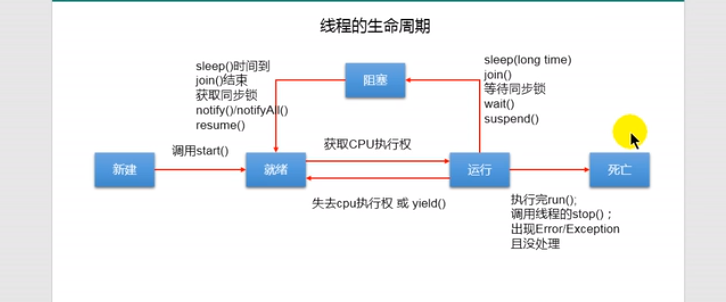
3.start（）方法，一个线程只能调用一次，调用二次回报异常

关于CountDownLatch的用法： 1.类似于join 2.需要在run方法内部调用其countDown方法，让其count减1

线程的调度：调度策略（时间片）

Runnable：1.继承runnable接口2.调用Thread的带参数构造方法3.调用thread的start方法

线程的生命周期：新建--🡪就绪-🡪运行/阻塞--🡪死亡



线程安全问题：主要是因为共享数据

同步代码块：Synchronize（同步监视器）{需要被同步的代码}

需要被同步的代码：操作共享数据的代码

共享数据：多个线程操作的变量

同步监视器（🔒鎖）：任何一个类的对象都可以充当锁

要求：多个线程必须要公用一把锁，即对象只能new一个

同步方法：方法返回值前面加上synchronize

死锁：不同线程分别占用对方需要的共享资源不放弃，都在等待对方放弃自己需要的同步资源