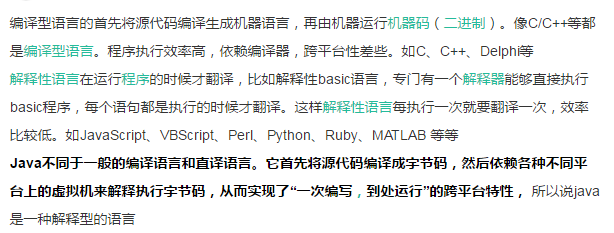
# SE

* 子类默认调用父类的构造方法，也就是说super（）是默认调用的，显示调用必须放在构造方法第一行！
* 重载：同一个类中，方法名相同，参数列表不同（参数类型，参数个数，参数顺序一个不同即可），与方法返回值无关。
* 复写：父子类中，子类中方法名相同，参数列表相同，返回值相同，子类的权限修饰符>=父类方法。
* package语句必须作为源文件的第一条非注释性语句，一个源文件只能指定一个包，只能包含一条package语句
* 标识符可以包括这4种字符：字母、下划线、$、数字；开头不能是数字；不能是关键字
* 
* Java源码编译和执行过程

@编译型语言：c/c++语言。将源码编译成机器语言，再由机器运行[机器码](http://baike.baidu.com/view/52.htm)（[二进制](http://baike.baidu.com/view/18536.htm)）

@解释型语言：。javascript知道执行这段代码的时候再翻译这段代码。执行一次翻译一次。效率低。



<http://blog.csdn.net/mccand1234/article/details/52013918>

* i++和++i的区别

@i++ : 先赋值，在自增1；

@++i ：自增1，再赋值。

* HashMap中是用哪些方法来解决哈希冲突的

<http://blog.csdn.net/abcd1430/article/details/52745155>

<https://zhuanlan.zhihu.com/p/21673805> （java8中的hashMap）

<http://www.cnblogs.com/highriver/archive/2011/08/15/2139600.html> 位运算符

hashMap:初始化容量（16），负载因子loadFactor(0.75),theadhold(阈值)

增加负载因子，减少hashMap占用的内存空间，但是会降低查询时的效率，同时也会增加hash冲突发生的可能性，即产生单链表。

减少负载因子，提高了查询效率，但是增加了占用的内存空间。

0.75是经过试验论证得出的结论。

**链表法**：

**单链表**：当key的hashcode一样，但是key不一样的时候，就会形成单链表。

**HashMap和hashTable的区别**：

1.HashMap的工作原理、ArrayList与Vector的比较以及这个问题是有关Java 集合框架的最经典的问题。

HashMap和hashTable的联系和区别

1.HashMap:非同步;不安全；;快；可接受null值。

2.HashTable:同步；安全 ；;慢；不可接受null值。【java5:ConcurrentHashMap可替代】

StringBuffer和Stringbuilder联系和区别

1.StringBuffer :安全；慢。

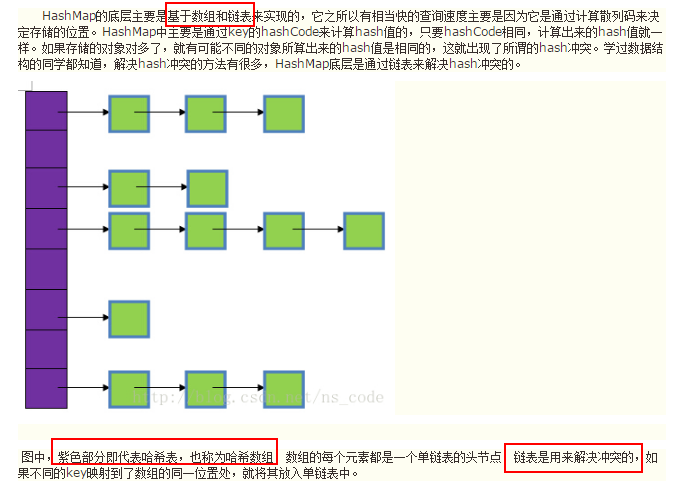
2.StringBuilder:不安全；快。

解决hash冲突

单链表

哈希表

**HashMap的底层主要是基于数组和链表来实现的**



难道hashMap扩容之后，hash值一样的就不会形成一个链表了吗？

Java 位运算符

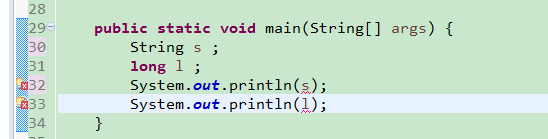
&:与运算符；同时为1才为1，其他为0.【有0则0】

|:或运算符；有1则1.【有1则1】

~：0变1,1变0

^：两边的位不同时，结果为1，否则为0.如1100^1010=0110

* 类中成员变量会有默认初始化值，局部变量没有默认值，不初始化无法编译通过。



* 注意事项：

Ant和Maven都是基于Java的构建(build)工具。

理论上来说，有些类似于（Unix）C中的make ，但没有make的缺陷。Ant是软件构建工具，Maven的定位是软件项目管理和理解工具。

Ant特点 ›

没有一个约定的目录结构 ›必须明确让ant做什么，什么时候做，然后编译，打包 ›没有生命周期，必须定义目标及其实现的任务序列 ›没有集成依赖管理

Maven特点

›拥有约定，知道你的代码在哪里，放到哪里去 ›拥有一个生命周期，例如执行 mvn install 就可以自动执行编译，测试，打包等构建过程 ›只需要定义一个pom.xml,然后把源码放到默认的目录，Maven帮你处理其他事情 ›拥有依赖管理，仓库管理

* Final

@ 类;不可被继承

@方法：不可被子类覆盖

@变量：

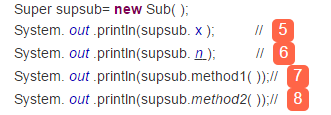
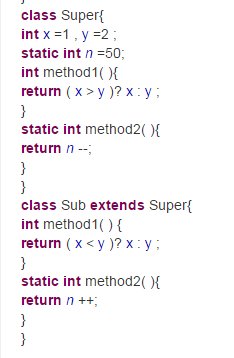
* 引用数据类型：变量指向的地址值不可以修改，但是对象的内容可以变化。
* 基本数据类型：基础类型的不可变化。
* Jiji
* Equals和“==”的区别

== ：判断地址值

Euqals:判断内容是否一致。

* 如果某个方法是静态的，它的行为就不具有多态性. 静态方法可以被继承，只是不能被重写.

Eg.父类的引用指向子类的对象，且父类和子类有一个同名的static方法



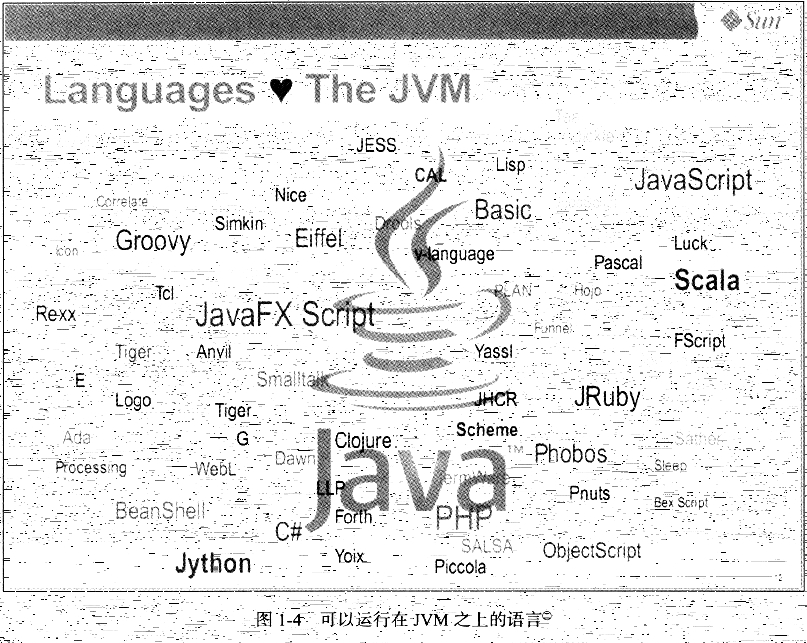
如图，第8行指向的是父类的method2的方法，不是子类的。但是第七行指向的是子类的方法

结果：1；50；1；50

* Swing 是在AWT的基础上构建的一套新的图形界面系统，它提供了AWT 所能够提供的所有功能，并且用纯粹的Java代码对AWT 的功能进行了大幅度的扩充。AWT 是基于本地方法的C/C++程序，其运行速度比较快；Swing是基于AWT 的Java程序，其运行速度比较慢。
* 进入DEAD的线程，它还可以恢复，GC不会回收
* 子类的构造函数，如果不明确指明，调用的是父类的无参构造。
* Java源文件中最多只能有一个public类，其它类的个数不限。

**Java一个源程序只能有一个public类存在，且类名与文件名相同。**Java程序是从main方法开始执行的，public为类加载器提供入口，然后找到public类中的main方法开始执行。如果存在多个public类，程序将不知道该从哪里执行。  
注意，内部类可以是public的，因为内部类是作为外部类的成员存在的。

* 可以运行在jvm中的语言：



# SQL

## Distinct-去重

select DISTINCT(memberSid) from coupons where type = 4

select DISTINCT memberSid, sid from coupons where type = 4

## And Or

select \* from coupons where type >2 and type < 4

select \* from coupons where type BETWEEN 3 and 5 （3<=x<=5）

## in

select \* from coupons where type IN (3,4,5)

## order by

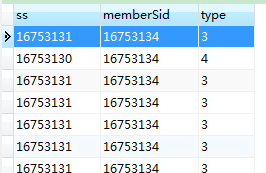
select type,memberSid from coupons ORDER BY 2 DESC

## 组函数（sum,avg,max,min,count）

* **AVG** (平均)
* **COUNT** (计数)
* **MAX** (最大值)
* **MIN** (最小值)
* **SUM** (总合)

select SUM(memberSid)/COUNT(memberSid) from coupons

select memberSid-type as ss,memberSid,type from coupons



例如文字相连 ([**concatenation**](http://www.1keydata.com/cn/sql/sql-concatenate.php))，文字修整 ([**trim**](http://www.1keydata.com/cn/sql/sql-trim.php))，以及子字串 ([**substring**](http://www.1keydata.com/cn/sql/sql-substring.php))。

练习：

SELECT

count(DISTINCT memberSid)

FROM

coupons

SELECT

count(memberSid)

FROM

coupons where memberSid != '' and memberSid is not NULL

## 一旦有栏位使用组函数，那么其他没有使用函数的栏位必须写在group by之后

select MAX(changetime) ,memberSid from coupons GROUP BY memberSid

## Where后不可使用组函数，having后可以使用组函数

一个含有 **HAVING** 子句的 SQL 并不一定要包含 [**GROUP BY**](http://www.1keydata.com/cn/sql/sql-groupby.php) 子句

select type, MIN(memberSid) from coupons GROUP BY type HAVING MAX(memberSid) > 1790234

练习：根据pid和supplySid获取最小价格，门店。

SELECT

product\_sid,

supply\_sid,

shop\_sid,

promotion\_price

FROM

ssd\_price

WHERE

product\_sid = 6479811

AND supply\_sid = 100282

ORDER BY

promotion\_price

LIMIT 0,1

//标红处：**完美解决门店取值不对的问题！！**

* 使用count（列名）当某列出现null值的时候，count（\*）仍然会计算，但是count(列名)不会。

## 执行顺序

书写顺序：

**Select**

average(score)

**From**

student

**Where**

score>=60 and s\_name !=’jr’

**Group** **by**

s\_name

**Having**

average(score)>70

**Order** **by**

average(score) desc;

执行顺序:

From -🡪where--🡪group by -🡪having---🡪select--🡪order by

From:表格自右向左解析，所以讲数据量较小的表格写在最右边（小表关联大表）

Where:自左向右解析，所以将筛选出小量数据的条件写在左边（小表关联大表）

**例子**：

完成一个复杂的查询语句，需求如下：

按由高到低的顺序显示个人平均分在70分以上的学生姓名和平均分，为了尽可能地提高平均分，在计算平均分前不包括分数在60分以下的成绩，并且也不计算贱人（jr）的成绩。 分析：

1．要求显示学生姓名和平均分

因此确定第1步select s\_name,avg(score) from student

2．计算平均分前不包括分数在60分以下的成绩，并且也不计算贱人（jr）的成绩

因此确定第2步 where score>=60 and s\_name!=’jr’ 3．显示个人平均分

相同名字的学生（同一个学生）考了多门科目 因此按姓名分组 确定第3步 group by s\_name 4．显示个人平均分在70分以上

因此确定第4步 having avg(s\_score)>=70 5．按由高到低的顺序

因此确定第5步 order by avg(s\_score) desc

## Inner join

**SELECT A1.Region\_Name REGION, SUM(A2.Sales) SALES   
FROM Geography A1, Store\_Information A2   
WHERE A1.Store\_Name = A2.Store\_Name   
GROUP BY A1.Region\_Name;**

## Union(并集and去重)

**UNION** 的一个限制是两个 SQL 语句所产生的栏位需要是同样的资料种类。另外，当我们用 **UNION** 这个指令时，我们只会看到不同的资料值 (类似 **SELECT**[**DISTINCT**](http://www.1keydata.com/cn/sql/sql-distinct.php))。

SELECT

member\_sid

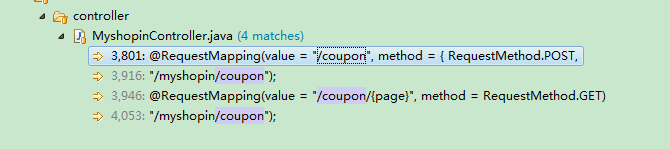
FROM

invitation i

UNION

(SELECT member\_sid FROM praise)

两个Select先查询结果，然后对两个结果集的并集进行去重。（注意：除了把两个结果集的结果去重，如果一个结果集中存在重合记录，也会被去重掉。所以union的结果个数是可以小于两个结果集的任何一个的。）



## Union All(并集and不去重)

两个sql的栏位必须是一样的。

select member\_sid from praise WHERE `status` = 1

union ALL

select member\_sid from invitation where `status` = 1

## Intersect（mysql不支持）

<http://blog.sina.com.cn/s/blog_4586764e0100h514.html>

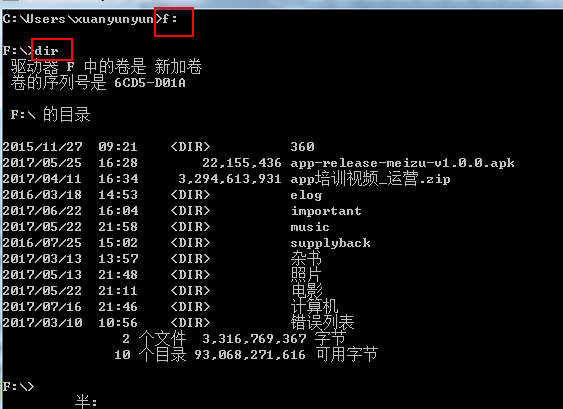
# Java核心技术I

1. 数字签名（微软的active X）
2. 问题：Jvm和一般虚拟机的概念混淆：

答案：Java虚拟机，简称JVM，是支持[Java语言](https://www.baidu.com/s?wd=Java%E8%AF%AD%E8%A8%80&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPjD1uHTznWNhuyc3mHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1f4rHcLP1fL)运行的环境，Java号称跨平台么，因此它上面一般运行Java程序，而下面需要一个虚拟机来支持，这个虚拟机对于不同[操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPjD1uHTznWNhuyc3mHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1f4rHcLP1fL)是不一样的，所以又JVM for win, JVM for linux等等。  
VMWare Workstation是[VMWare公司](https://www.baidu.com/s?wd=VMWare%E5%85%AC%E5%8F%B8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPjD1uHTznWNhuyc3mHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1f4rHcLP1fL)出的虚拟机，它的功能是在程序里面模拟[操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPjD1uHTznWNhuyc3mHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1f4rHcLP1fL)，比如linux, windows等等，运行时候一个窗口里面就是一个[操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPjD1uHTznWNhuyc3mHK-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1f4rHcLP1fL)。  
简单的说来：**Java虚拟机上运行Java程序，VMWare Workstation上运行操作系统。这就是最大的区别。**

1. Dos命令行：

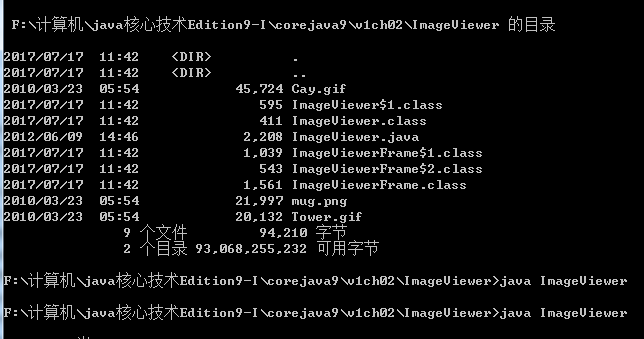
dir:查看当前目录信息，同时查看该目录下的文件

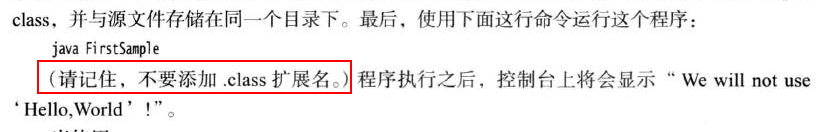


1. Java区分大小写。
2. 为什么要设置classpath??
3. Javac 和Java

Javac:编译java源文件。

Java:运行字节码文件（注意java 空格 后跟的是class文件，但是不要要上.class后缀）



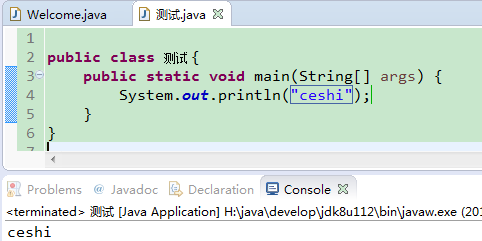


## 第三章

访问修饰符；关键字；大小写敏感；类名（骆驼命名法）【数字字母，字母开头，首字母大写，中文不行？】

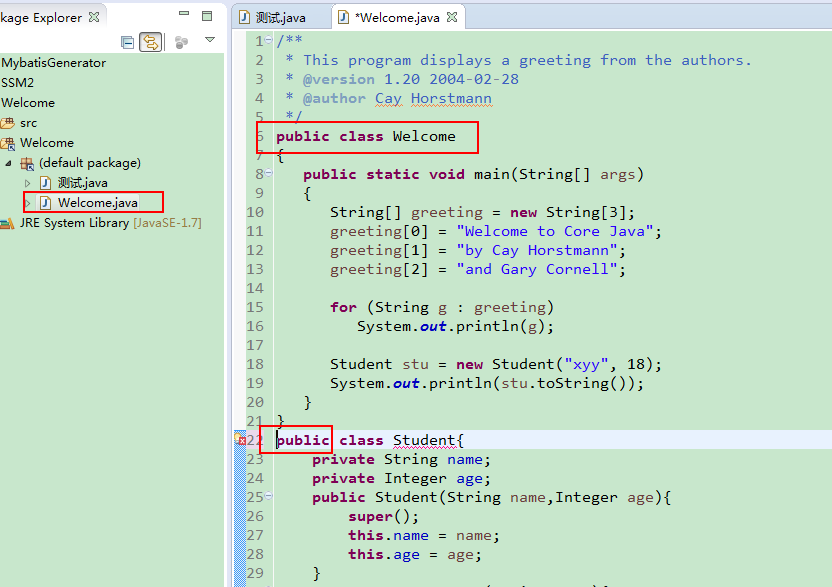


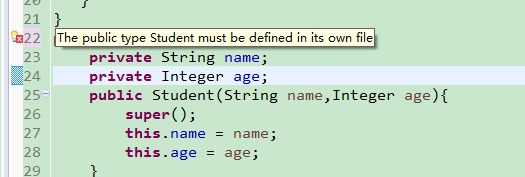
中文是可以作为类名的。



一个类文件中只可以有一个公共类（被public修改的类），同时类文件名字必须和公共类的类名一直，区分大小写。





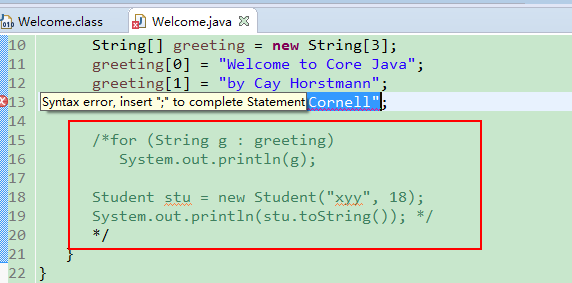


每个句子需要使用分号（；）结束。回车不是语句结束的标志，如果需要可以将一个语句分成多行。

### 注释

注释：单行注释，多行注释，文档注释（/\*\* \*/）





### 四类八种（primitive type-基本类型）

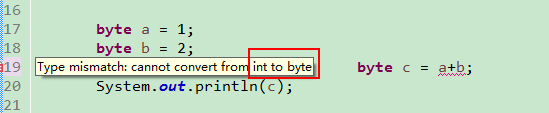
byte-short-int-long

float-double

char

boolean

int < long < float < double



两个数字相加减：

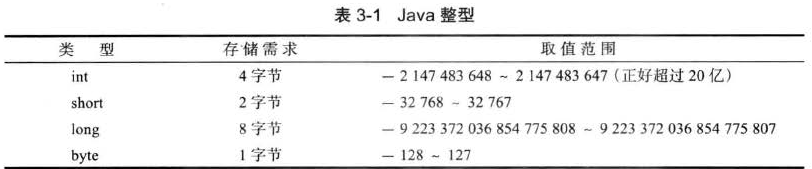
有一个操作数是double,则另外一个会强转为double;

有一个操作数是float,则另外一个会强转为float;

有一个操作数是long,则另外一个会强转为long;

否则，两个数默认是int.

#### 整型





#### 浮点型



Double>float>long>int>short>char

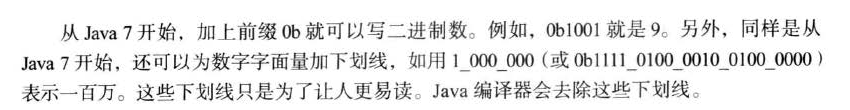
1字节：8位。

八进制：0开头。

十六进制：0x开头

二进制：0b开头。

Java7特性：



浮点型数据：

正无穷:double.POSITIVE\_INFINITY

负无穷:DOUBLE.NEGATIVE\_INFINITY

NAN:NOT A NUMBER

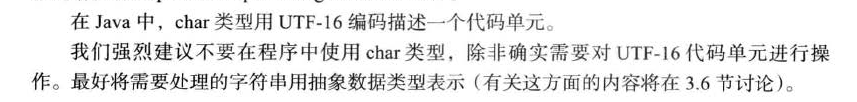
isNaN（..）

任何一个非数值的值之前互不相等的。



#### char





1. 范围： \u0000~\uffff

65-A；97-a

1. 字符常量：a-z;A-Z;0-9;各种汉字；各种符号。
2. 转义序列符：\u;\\;\’;\”;\r;\n;\b\t;
3. 两个字节-🡪16位--🡪二进制（16位）；16进制（4位）；

可表示2的16次方（65536个数字）



#### Boolean

整数型不可和boolean进行转换。

### 强类型？弱类型？解释型？编译型？

### 变量

1．变量名称范围：数字，字母（a-zA-Z）,\_(下划线)，$(美元符号)

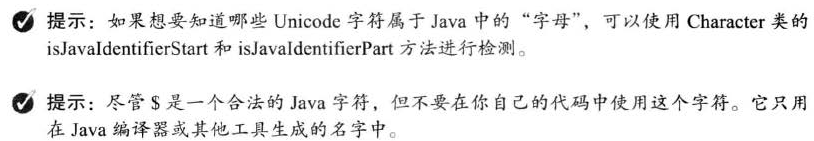
2. 不可以以数字开头，其他都可以。

3. 不可有空格。

4. 大小写敏感。（name,NAME是不同的。）

5. 不限制长度。

6. 不可使用保留字。



#### 变量初始化

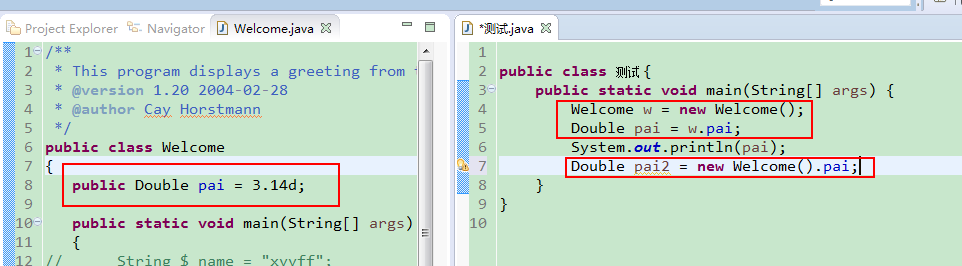
变量的声明和初始化。

Int number;//声明

number = 4;//初始化

int num = 5;//声明and初始化

在java中，变量的声明尽可能的靠近变量第一次使用的地方，这是一种良好的程序编码风格。



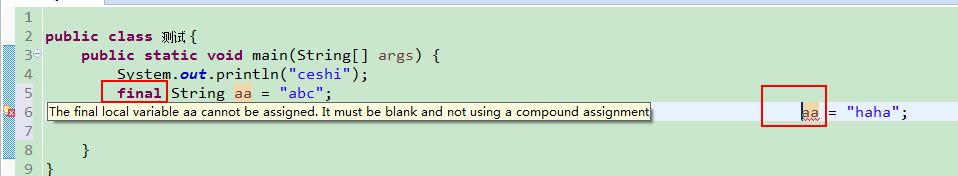
变量如果使用public修饰，那么其他类可以通过对象.属性名直接就可以拿到属性值了，不安全啊。。

#### 常量（final）

2. 大写

3. 类常量，成员变量。

4. 被final修饰的变量只可以赋值一次。但是如果final修饰的是一个对象，修改对象的内容是可以的，只要不修改变量指向的地址值即可。

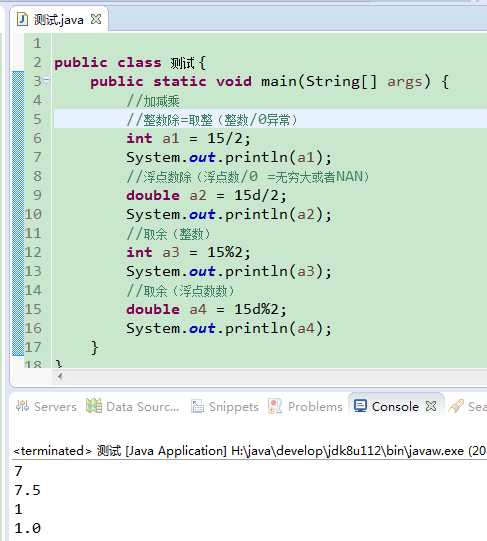


Java中的常量的分类

1. 整数常量，所有整数
2. 小数常量，所有小数
3. 布尔型常量，较为特有，只有两个数值，true false
4. 字符常量，将一个数字字母或符号用单引号（‘’）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

遗留问题：常量池

### 运算符



**public** **class** 测试 {

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

//加减乘

//整数除=取整（整数/0异常）

**int** a1 = 15/2;

System.***out***.println(a1);

//浮点数除（浮点数/0 =无穷大或者NAN）

**double** a2 = 15d/2;

System.***out***.println(a2);

//取余（整数）

**int** a3 = 15%2;

System.***out***.println(a3);

//取余（浮点数数）

**double** a4 = 15d%2;

System.***out***.println(a4);

}

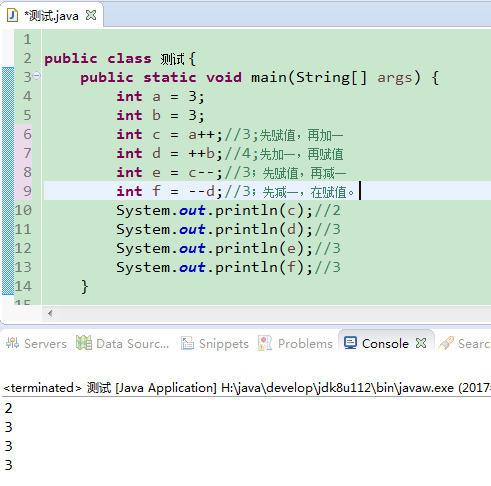
}

关键字Strictfp:strict float point 精确浮点

说明：<http://blog.csdn.net/summerhust/article/details/37994225>

#### 自增自减

1. 操作数不可以是数值。Eg. 5++ ,非法语句。



**public** **class** 测试 {

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

**int** a = 3;

**int** b = 3;

**int** c = a++;//3;先赋值，再加一

**int** d = ++b;//4;先加一，再赋值

**int** e = c--;//3；先赋值，再减一

**int** f = --d;//3；先减一，在赋值。

System.***out***.println(c);//2

System.***out***.println(d);//3

System.***out***.println(e);//3

System.***out***.println(f);//3

}

}

#### 关系运算符与boolean运算符

1. >;<;==;>=;<=;!=
2. &;|;!;&&;||;
3. 三元运算符 exp?x:y;

#### 位运算符(没仔细看，有时间，细看)

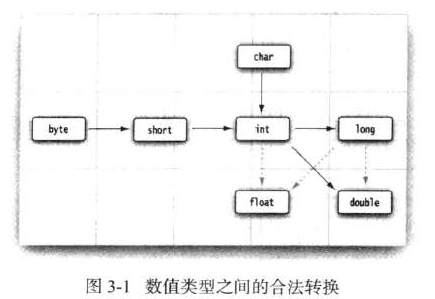
#### 数学函数和常量

Math: PI E

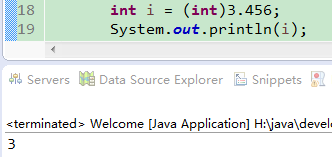
静态导入： import static java.lang.Math.\*; (这样可以直接Math内的方法，和常量，不需要在前面加上Math)

StrictMath：得到一个可以预测的结果比运行速度更加重要，可以确保在所有平台得到的结果都一样。

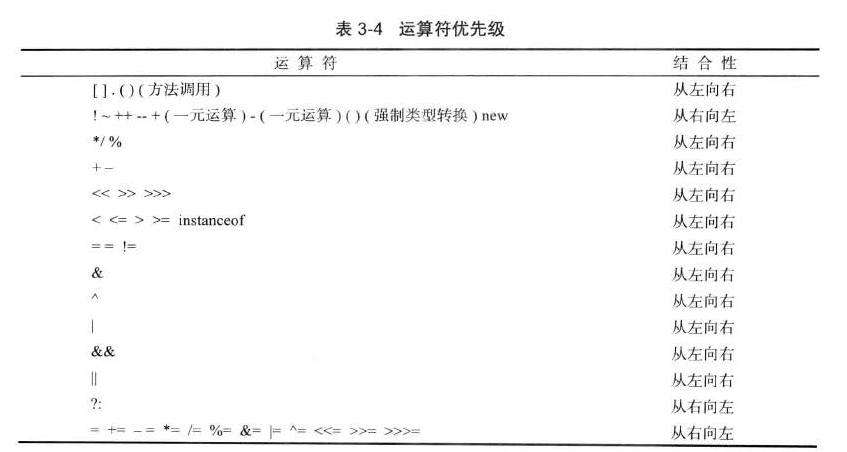
#### 强制类型转换



1. 将浮点数（double,float）强制转换成整数类型（int）,其实就是将小数部分删除掉。



#### 运算符的优先级



### 字符串

1. 双引号引着的。
2. 任何对象和字符串继续拼接，都会调用toString（）被转化成字符串。

Syso(“98huhuh”+ new Person());

1. 字符串的不可变性。表面修改字符串，实际上是让字符串变量指向了另外一个字符串常量而已，修改了地址值。
2. == 和equals

答案：== 比较地址值。

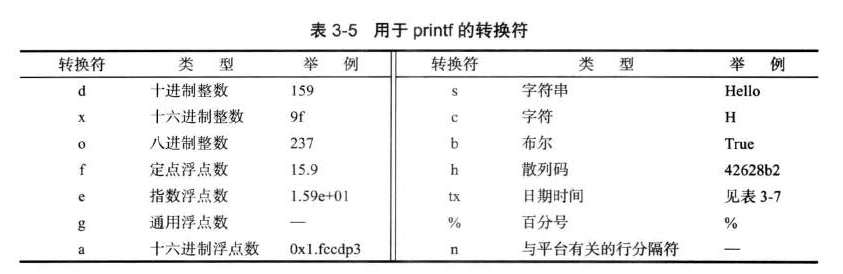
Equals比较内容。

1. 代码点and 代码单元？？
2. 当需要不断的去对字符串进行拼接的时候，请使用StringBuilder,stringbuffer.

Stringbuilder-高效，线程不安全；

Stringbuffer-抵消，线程安全；

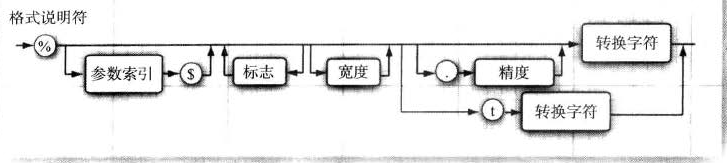
### 输入输出

1. scanner对象
2. 格式化输出 【System.out.printf(“%f”,19.00)】
3. 
4. 
5. 



一般以%开头；

一般每个%，都是修改对应索引上的内容，从1开始计数；符号<表示修饰前一个索引的内容；所以上面除了第一个和最后一个%修饰其他，从第二个%到第四个%都是修饰new Date();

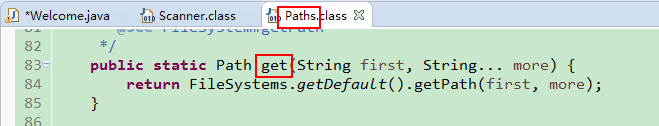


1. 

Scanner 是对输入流的装饰流，用于读取数据。

读取file,读取给定的字符串（不是文件名）



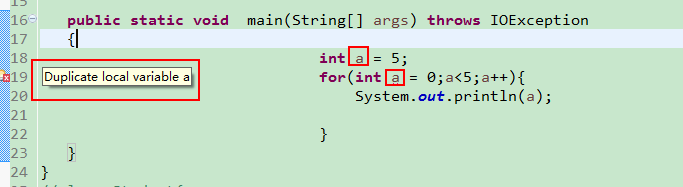


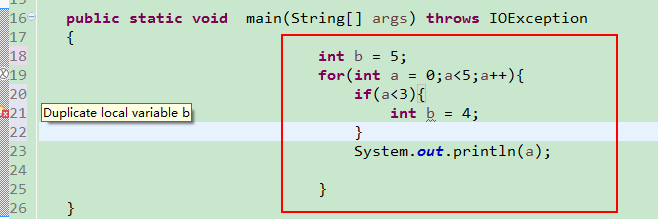
### 控制流程

* 条件语句：if-else;if-elseIf;
* 循环语句
* 块：一对花括号围起来的范围

块：一对花括号围起来的范围，即块。（复合语句）。【变量的作用域】

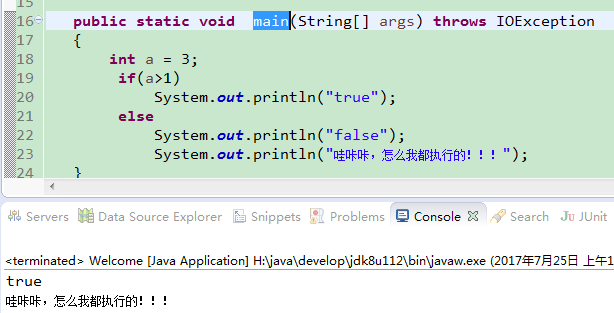
块可以嵌套，但是不能再声明的两个块中声明同名变量。



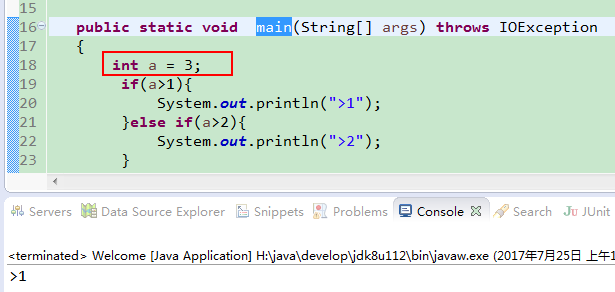


#### 条件语句（if-else;else if）

If-else:如果只有一条语句，可以不需要花括号，没有花括号的情况下，if-else只对第一个语句负责的。



If-elseIf：只要满足一个条件，执行完对应的语句后，就不会再判断下面的条件，直接跳出条件语句了。



#### 循环语句

* while-先判断，再执行。
* do-while-至少执行一次，先执行，再判断。

试一试：while-先判断，再执行。

**public** **static** **void** main(String[] args) **throws** IOException

{

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("目标存款：");

**double** goat = sc.nextDouble();

System.***out***.println("每年存款：");

**double** averageMoney = sc.nextDouble();

**double** nowMoney=0;

**int** year = 0;

**while**(goat>nowMoney){

nowMoney +=averageMoney;

// nowMoney = nowMoney\*1.01;

BigDecimal nowMoneybD = **new** BigDecimal(nowMoney).multiply(**new** BigDecimal(1.01d));

nowMoney = nowMoneybD.doubleValue();

year++;

}

System.***out***.println("需要"+year+"年才够存款");

}

试一试：do-while-至少执行一次，先执行，再判断。

**public** **static** **void** main(String[] args) **throws** IOException

{

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("每年存款：");

**double** averageMoney = sc.nextDouble();

**double** nowMoney=0;

**int** year = 0;

String str="";

**do**{

nowMoney +=averageMoney;

BigDecimal nowMoneybD = **new** BigDecimal(nowMoney).multiply(**new** BigDecimal(1.01d));

nowMoney = nowMoneybD.doubleValue();

System.***out***.println("您的存款即将="+nowMoney+",准备退休了吗");

str = sc.next();

year++;

}**while**(str.equals("N"));

System.***out***.println("需要"+year+"年才够存款");

}

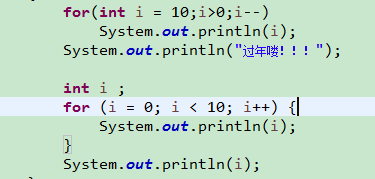
Scanner类中nextLine和其他next方法连用的问题：【看绿色注释重点】

原理：<http://blog.sina.com.cn/s/blog_81547cad01018mnd.html>



#### for循环

注意下面的两个for循环，因为不是块嵌套的关系，所以同名变量没有问题。



#### switch语句

1. switch语句，进入到某一个case后，执行语句，知道遇到break;所以，每一个case下都要有break；

否则一旦满足条件，执行完某一个case以后，继续执行下面的case，知道遇到break;或者执行完毕。

1. switch(参数类型)：byte,short,int,枚举，String

#### break（for,switch,while）和continue的用法（while,for）

对于使用

跳出for循环

**public** **static** **void** main(String[] args) **throws** IOException

{

first:

**for**(**int** i = 0;i<5;i++){

inner:

**for**(**int** j = 0;j<5;j++){

System.***out***.println("--"+i+"--"+j);

**if**(i>3){

**break** first;//跳出外循环

}

}

}

}

Continue:跳出本次循环，继续外层的下次循环。

**public** **static** **void** main(String[] args) **throws** IOException

{

first:

**for**(**int** i = 0;i<5;i++){

inner:

**for**(**int** j = 0;j<5;j++){

System.***out***.println("--"+i+"--"+j);

**if**(i==2){

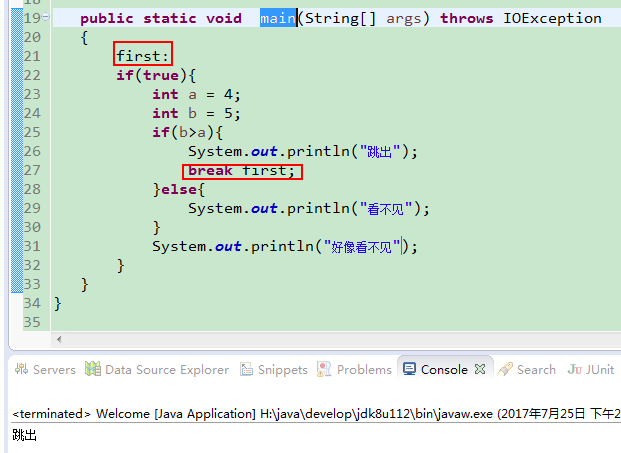
**continue** first;//继续外循环的下次循环

}

}

}

}

 If语句也可以使用标签，实现跳出if语句：（不建议使用）

### 大数值- BigDecimal-BigInteger



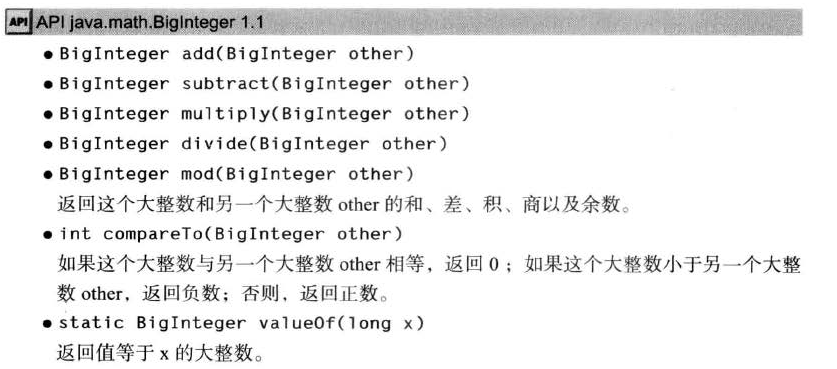
Add

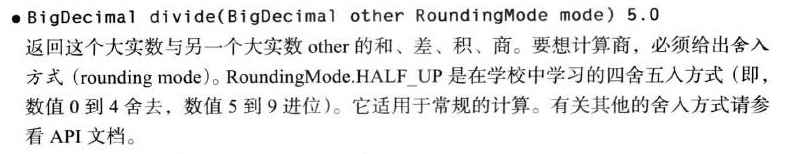
Substract

Multiply

Divide

mod







### 数组



声明定义数组的几种方式：



数组元素的默认初始化值：

Byte,short,int,long,float,double : 0

对象：null

Booean:false

Char:空字符串（ascii为0）

数组列表？？

#### 增强for(for each)

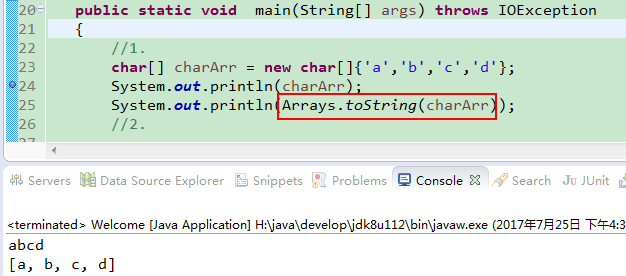
1.使用范围：数组，实现了Iterable接口的对象。

2.数组长度可以为0；new char[0]。 长度为0和null不同。

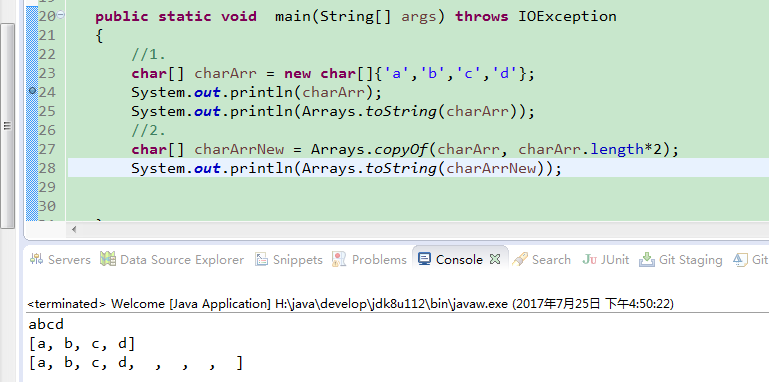
#### Arrays-操作数组的类

* 获取数组内容：tostring()

Arrays.toString();可以遍历数组



* 数组拷贝：copyOf（）



* 排序：sort(快速排序，二分法，冒泡排序java几种排序方法)

试一试：从100个数字中选择9不相同的数字

**public** **static** **void** main(String[] args) **throws** IOException

{

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("一共有几个数字呢？");

**int** total = sc.nextInt();

System.***out***.println("需要抽取几个数字呢？");

**int** num = sc.nextInt();

String s = *getMessage*(num,total);

System.***out***.println(s);

}

**private** **static** String getMessage(**int** num, **int** total) {

//构造所有数字

**int**[] numArr = **new** **int**[total];

**for**(**int** i = 0;i<numArr.length;i++){

numArr[i] = i+1;

}

**int**[] newArr = **new** **int**[num];//新数组

**for**(**int** j = 0;j<num;j++){

//随机抽取一个数字

**int** indexRandom = (**int**)(Math.*random*()\*total);//注意这个括号，要先获取数字，再强转成int

//复制新数组

newArr[j] = numArr[indexRandom];

//抽取的数字不可以再被抽取到

numArr[indexRandom] = numArr[total-1];

total -=1;

}

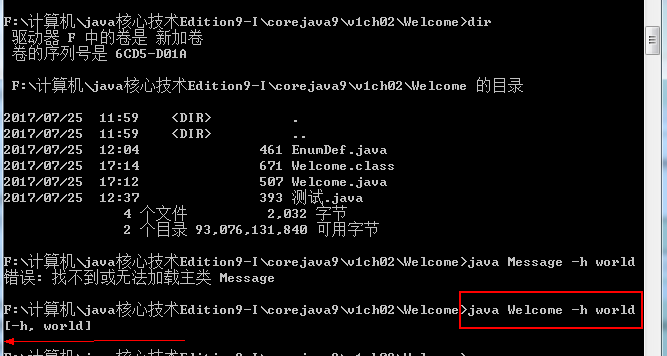
Arrays.*sort*(newArr);

**return** Arrays.*toString*(newArr);

}

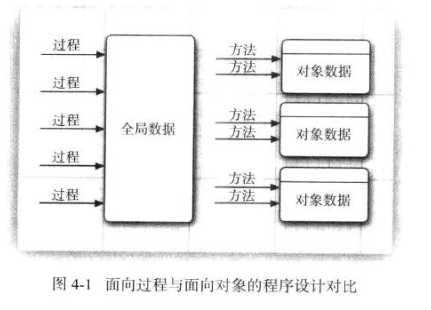
* Java中数组在内存中如何存储的呢？？
* Main方法中的args存储着命令行参数，但是不包含程序名





## 第四章 对象和类

程序=操作数据（操作-算法；数据-数据结构）



类的实例化即创建对象的过程。【类实例-🡪对象】

封装（encapsulation:capsule[n.胶囊]---🡪encapsule【v.装入胶囊】---🡪encapsulation[n。封装]）

状态：（实例域构成），对象的数据。

方法：操作数据的过程。

getter,setter..



依赖【use-a】：一个类（A）的方法中操作另外一个类（B）的对象，即A依赖B.

聚合【has-a】：类A中包含类B.(B是A的成员变量)

继承【is-a】（inherit[v.继承]--🡪 inheritance[n.继承，遗传]）

### 概述

* 对象变量在使用之前，一定要初始化，否则无法通过编译



* New操作符，返回的是一个对象的引用。

Date d = null;（显示设置为null,表示这个变量d现在没有指向任何一个地址值）

Date d1;

不赋值就不会分配空间,赋值null会分配0大小的初始空间.

* 成员变量和局部变量

局部变量：必须进行显示初始化(new或者null); 作用域是所在代码块{}；生命周期和类一致；栈

成员变量：默认初始化，数字为0，char为‘’,对象为null;作用域是整个类；生命周期；堆

试一试：使用Calendar展示如下图日历（没做）



**public** **static** **void** main(String[] args) **throws** IOException, ParseException

{

Locale.*setDefault*(Locale.***CHINA***);

//星期文字显示

String[] shortWeekdays = **new** DateFormatSymbols().getShortWeekdays();

**for** (**int** i = 0; i < shortWeekdays.length; i++)

System.***out***.print(" "+shortWeekdays[i]);

System.***out***.println();

Calendar cal = Calendar.*getInstance*();

cal.set(Calendar.***DAY\_OF\_MONTH***, 1);

**int** weekday = cal.get(Calendar.***DAY\_OF\_WEEK***);

**for**(**int** i = 0;i<weekday;i++){

System.***out***.print(" ");

}

System.***out***.println(cal.get(Calendar.***DAY\_OF\_MONTH***));

**if**(weekday==7){

System.***out***.println();

}

**for**(**int** j = 0;j<31;j++){

cal.add(Calendar.***DAY\_OF\_MONTH***, 1);

**int** dayOfWeek = cal.get(Calendar.***DAY\_OF\_WEEK***);

System.***out***.print(" "+cal.get(Calendar.***DAY\_OF\_MONTH***));

**if**(dayOfWeek==7){

System.***out***.println();

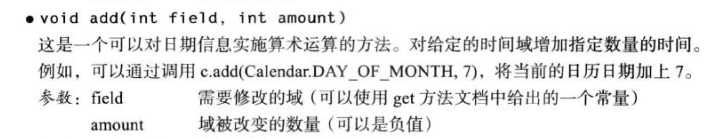
}

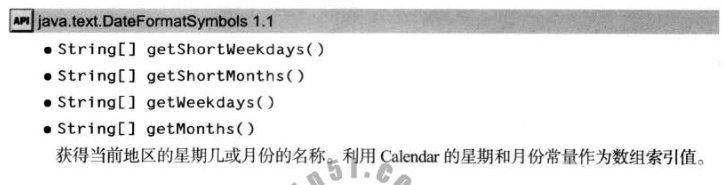
}

}

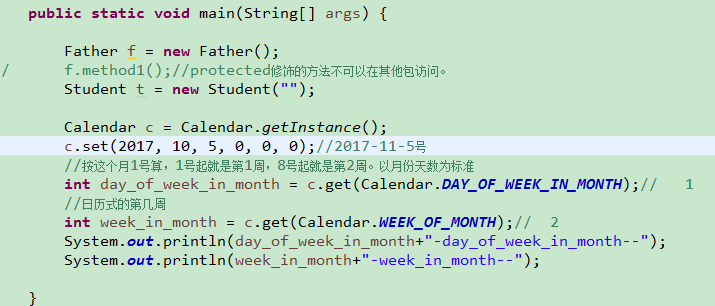
Calendar.get（filed）field可选字段。







Calendar中的**DAY\_OF\_WEEK\_IN\_MONTH**和**WEEK\_OF\_MONTH**的区别



### \*自定义类and final

* 一个源文件中可以有多个class,但是只可以有一个被public修改的公共类，并且源文件的名字必须和公共类类名一致。

**public** **class** EmployeeTest {

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

Employee[] eeArr = **new** Employee[3];

eeArr[0] = **new** Employee("xyy",500d,2007,7,5);

eeArr[1] = **new** Employee("xyy1",1000d,2007,7,5);

eeArr[2] = **new** Employee("xyy2",2000d,2007,7,5);

**for** (Employee employee : eeArr)

employee.raiseSalary(0.5d);

System.***out***.println(Arrays.*toString*(eeArr));

}

}

**class** Employee{

**private** **double** salary;

**private** String name;

**private** Date fireDate;

**public** Employee(String name,**double** salary,**int** year,**int** month,**int** date){

Calendar cal = Calendar.*getInstance*();

cal.set(year, month, date);

Date d = cal.getTime();

**this**.fireDate = d;

**this**.name = name;

**this**.salary = salary;

}

**public** String getName(){

**return** **this**.name;

}

**public** **void** setName(String name){

**this**.name = name;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

**public** Date getFireDate() {

**return** fireDate;

}

**public** **void** setFireDate(Date fireDate) {

**this**.fireDate = fireDate;

}

**public** **void** raiseSalary(**double** rate){

BigDecimal upSalary = **new** BigDecimal(rate).multiply(**new** BigDecimal(salary));

**this**.salary = salary+upSalary.doubleValue();

}

@Override

**public** String toString() {

**return** "Employee [salary=" + salary + ", name=" + name + ", fireDate=" + fireDate + "]";

}

}

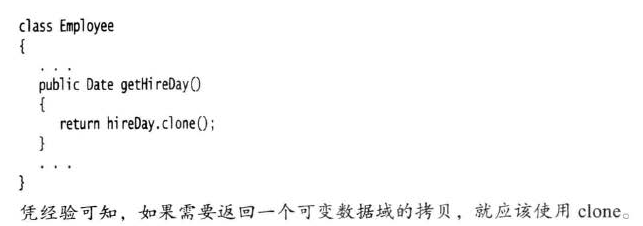
* 构造器相关

1. 构造器与类名同名
2. 构造器参数可以是0,1，多个。
3. 构造器没有返回数据
4. 构造器总是伴随new 一起被调用的。

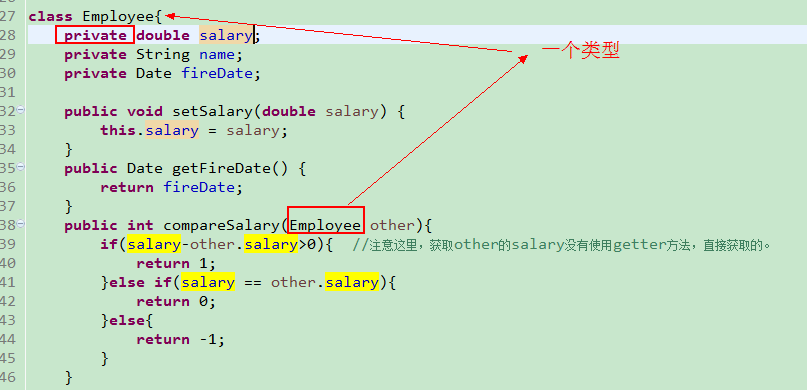
* 显式参数&隐式参数 + 形式参数&实际参数

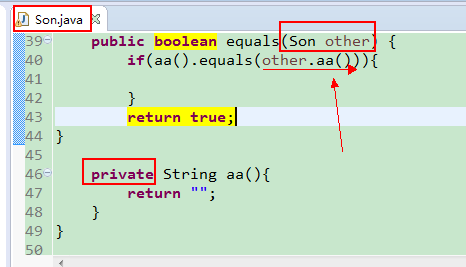
1. 显式参数：方法形参；
2. 隐式参数：this
3. 形式参数：方法声明中的参数（方法中的局部变量）
4. 实际参数：调用方法是，实际传递给方法的参数。

* Getter方法，如果成员变量是一个引用类型的数据，那么最好不要吧成员变量返回，而是clone一个完全相同的对象返回即可。



* 类（A）的方法可以访问属于类A的所有对象的私有特性。【一个方法可以访问所属类的所有对象的私有数据】



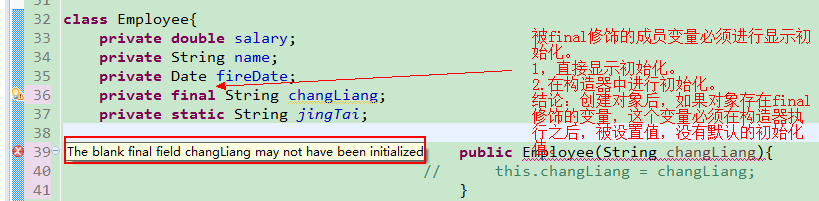


云云补充：将某些场景下，1.处理逻辑太复杂，将方法分解成几个方法，2.内部调用严格的调用顺序；这样的方法一般不对外公开，设置为**private**.一旦某天发现这样的方法不再适合业务场景，那么可以放心大胆的删除，或者屏蔽，因为这些方法不可能被其他系统所调用，删除不会对未知的其他系统产生影响。

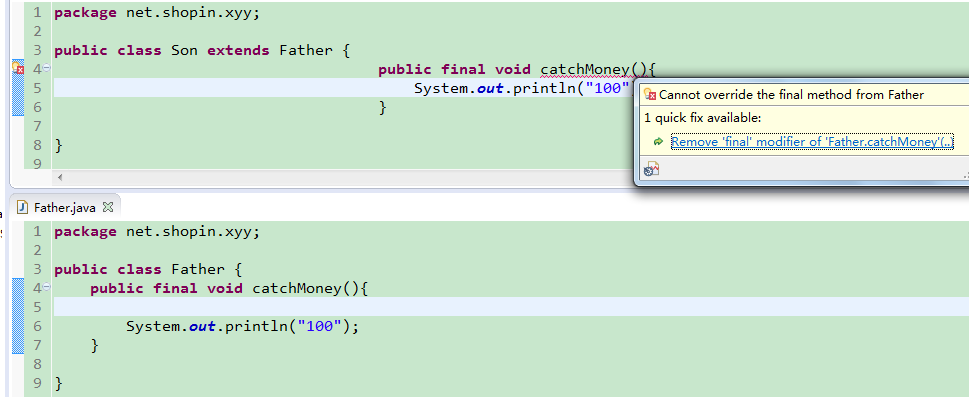
* Final修饰符

1. 修饰类：表示这个类是不可变类，不可以被继承
2. 修饰成员变量：表示这个类是常量，初始化对象时这个变量必须被赋值。【必须显示初始化，没有默认值】

但是static修饰的变量，有默认初始化值。



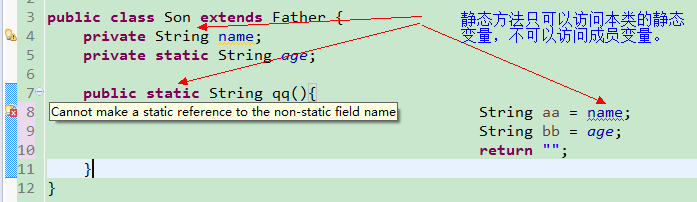
1. 修饰方法：不可以被复写



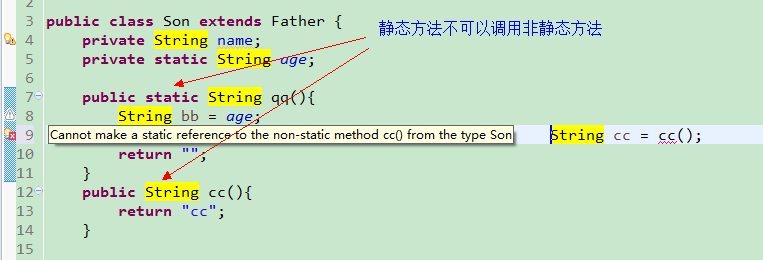
1. Final修饰一些引用类型的变量其实没有什么意义，因为final只是限制变量指向的引用不可以变化，但是地址值指向的对象内容发生变化是可以的，通过setter或者其他方法都是可以修改对象实际内容的。【所以final一般都是修饰基础数据类型或者是不可变类】
2. 被final修饰的变量因为不可以修改，所以，不需要提供setter方法。

### \*Static

1. 修饰变量：
   1. 所有类的对象共享这个属性，属于类。【类变量】
   2. 直接通过类型获取即可。Math.PI;
2. 静态常量（static final 共享并且不可修改）--🡪Math.PI
3. 修饰方法：
   1. 可以通过类直接调用。
   2. 本类的静态方法是不可以操作本类的对象【即不可以使用this】
   3. 静态方法只可以访问类变量，不可以访问成员变量。【如果是新建一个本类对象，没有问题，通过getter即可获取成员变量】(因为成员变量是基于对象的，但是静态方法加载的时候，对象不存在，所以不允许直接使用成员变量。)

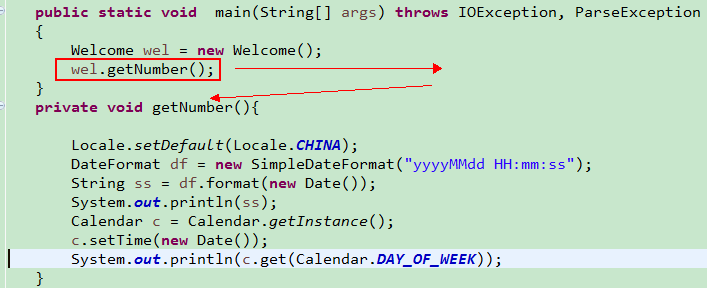


* 1. 静态方法只可以访问静态方法。

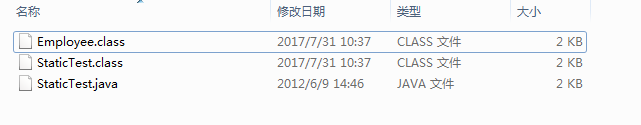


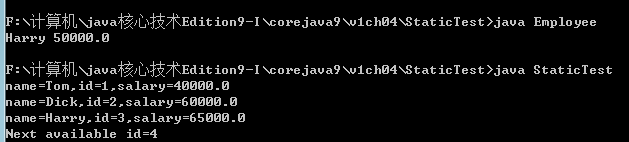
* 1. 什么场景使用static
     1. 方法内部没有访问此类的成员状态，所操作的都是形参。【静态方法不可以操作此类的成员变量，因为没有this】
     2. 方法内部只操作了类的静态状态。

**重点**：注意静态方法中只可以调用静态方法，这种限制只作用于直接调用本类的方法。如果先创建一个任意类的对象，再去调用这个对象的方法，那么不管这个方法是静态还是非静态，都是可以的。



重点：





通过java命令去执行某一个class文件，就会执行这个类的main方法，而其他的main方法不会执行。

### \*方法参数

1. 参数传值方式： 按值调用（java总是采取这种方式：常量值或者地址值）； 引用调用。
2. Java方法是无法修改实参变量的指向,只能够修改形参的指向。（**重点**）
   1. 地址值传递：只可以修改地址值对应的对象内容，【实参也指向这个对象，所以可以感知变化。】【对象】
   2. 值传递：无法修改值，修改即让形参指向另外一个值而已。【实参依然指向原来的值，所以不感知变化。】【基本数据类型】

例外：String虽然是对象，但是采用的是值传递。

1. 结论：
2. 方法参数是基本类型，修改只对形参有效，实参是感知不到的。
3. 方法参数是引用类型，因为形参和实参指向同一个对象，所以修改内容时可以感知的。
4. 方法修改了形参的指向，实参是感知不到的，不论是基本类型，还是引用类型。

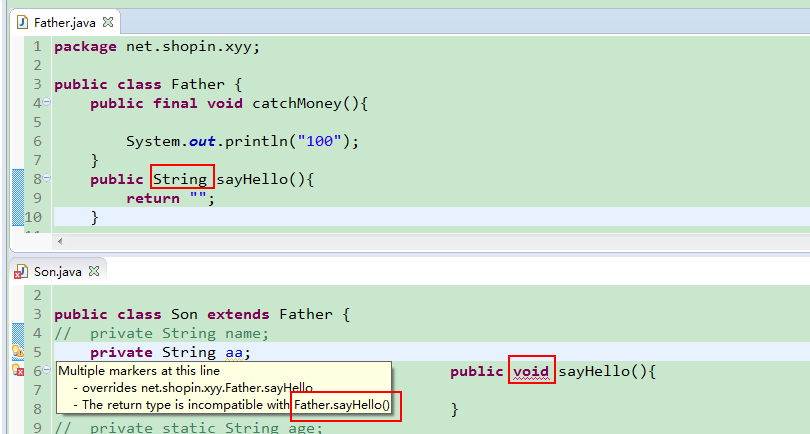
### \*构造器&初始化&父子类部分知识点

* 重载

方法重载（overload）：方法参数列表不同（类型，数目，顺序），方法名字相同，和返回值无关。【同一个类中】

方法复写：方法参数列表相同，名字相同，返回值必须一致。（见下截图）【父子类】

方法重写：同【复写】



1. 方法声明=方法签名=方法名称+参数列表（类型，数目，顺序）【不包含方法返回值】
2. 一个类中不可以存在两个方法：方法名字and参数列表相同，返回值不同。

* 成员变量（域）默认初始化

成员变量和局部变量

1. 成员变量：默认初始化，【0，false,空字符，null】
2. 局部变量：没有默认初始化，必须显示初始化

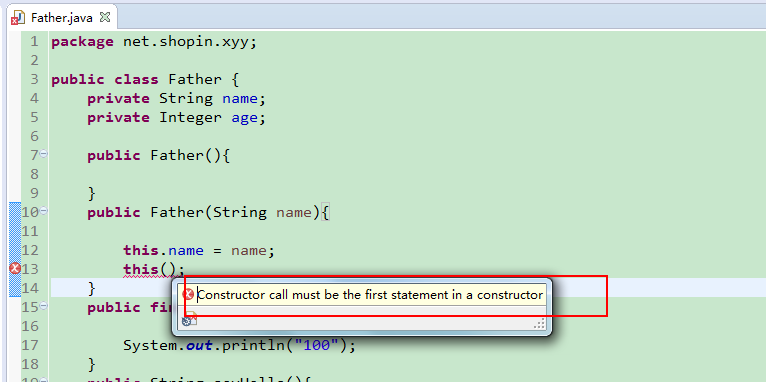
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 默认初始化 | 作用域 | 类别 |  |
| 成员变量 | 有 | 类 | 对象内成员变量 |  |
| 局部变量 | 无，必须显示初始化 | 代码块 | 方法参数，内部变量 |  |

* 无参构造

1. 类中如果没有提供构造器，系统默认添加无参构造。
2. 一旦类中指明了一个构造器，那么系统不会添加无参构造，如果需要，必须自己创建无参构造。

* 构造器引用。

1. this( );使用this引用【注意没有点”.”，不同于方法调用】
2. 构造器如果使用this引用其他构造器，那么这个语句必须位于第一行。（？？为什么）

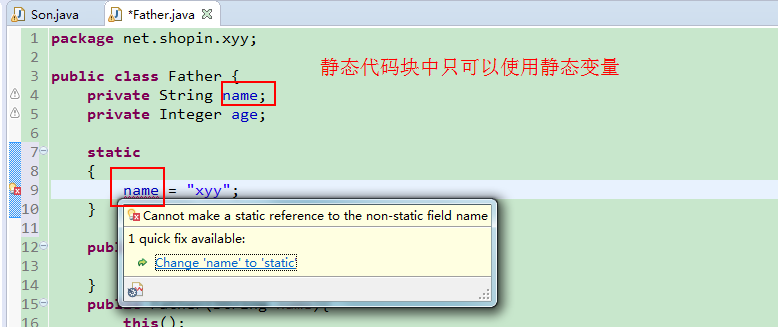


* 初始化块

1. 初始化数据方式：
2. 默认初始化；声明时候显示初始化；
3. 构造器中初始化；
4. 代码块中初始化【特殊：静态代码块】
5. 初始化代码块最好放在域的后面，代码块中不要读取后面初始化的域。
6. 先执行代码块，在执行构造器。（**重点**）

* 静态代码块

1. 静态代码块中只可以使用静态变量。



1. 只有加载类的时候执行一次。

* 类的初始化顺序

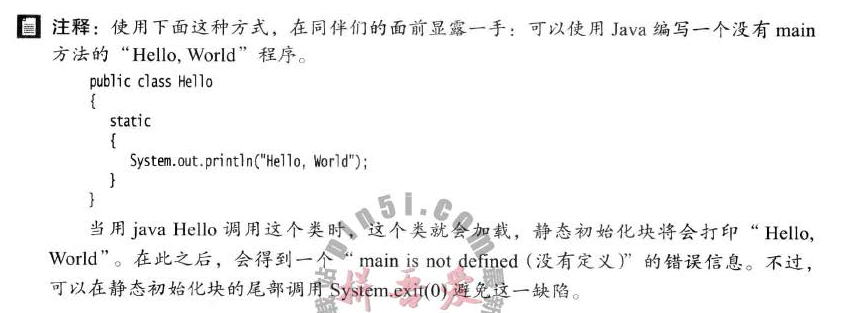
1. 静态域的默认初始化，
2. 静态域的显示初始化；
3. 静态代码块

【1-3只有在类第一次加载的时候才会执行。4-7创建一次对象就执行一次】

1. 域的默认初始化
2. 域的显示初始化语句
3. 一般代码块【非静态代码块】
4. 构造器，如果调用了其他构造器，先执行其他构造器

什么会触发类的第一次加载呢？（使用类的时候）

答：新建对象，调用类的静态变量，调用类的静态方法。



（上图实验失败！！！）

* 类加载顺序

0-父静态初始化

1-父静态代码块

2-子静态初始化

3-子静态代码块

4-无意义（忽略）

5-父代码块

6-父构造器

7-子代码块

8-子构造器

**public** **class** Father {

**private** **static** String *name* = "0-父静态初始化";

**private** Integer age;

**static**

{

System.***out***.println(*name*);

*name* = "1-父静态代码块";

System.***out***.println(*name*);

}

{

System.***out***.println("4-无意义");

System.***out***.println("5-父代码块");

}

**public** Father(Integer age) {

**super**();

**this**.age = age;

System.***out***.println("6-父构造器");

}

}

**public** **class** Son **extends** Father {

**private** **static** String *name*="2-子静态初始化";

**private** Integer age;

**static**

{

System.***out***.println(*name*);

*name* = "3-子静态代码块";

System.***out***.println(*name*);

}

{

System.***out***.println("7-子代码块");

}

**public** Son(Integer age) {

**super**(age);

System.***out***.println("8-子构造器");

}

}

* 实验，父子类关系时候，同名变量和方法的覆盖关系如何

**public** **class** Son **extends** Father {

**private** String name="2-子";

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**public** **class** Father {

**private** String name = "0-父";

**private** Integer age;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**public** **class** FatherAndSon {

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

Son son = **new** Son();

System.***out***.println("son name = "+son.getName());//2-子

Father father = **new** Father();

System.***out***.println("father name = "+father.getName());//0-父

Father f = **new** Son();

System.***out***.println("多态 name = "+f.getName());//2-子

}

}

返回结果：

son name = 2-子

father name = 0-父

多态 name = 2-子

结论：

1. 父子类关系时候，同门变量是不存在覆盖的，父类获取到的是父类的变量，子类获取的是子类的变量；
2. 当“父类的引用指向子类的变量”时，如果父子类都有这个方法，调用子类的方法。【原理？？】

* 如果将上述实验的name属性设置为static,同时又有setter和getter方法，如何？

**public** **class** Father {

**private** **static** String *name* = "0-父";

**private** Integer age;

**public** **static** String getName() {

**return** *name*;

}

**public** **static** **void** setName(String name) {

Father.*name* = name;

}

}

**public** **class** Son **extends** Father {

**private** **static** String *name*="2-子";

**public** **static** String getName() {

**return** *name*;

}

**public** **static** **void** setName(String name) {

Son.*name* = name;

}

}

**public** **class** FatherAndSon {

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

Son son = **new** Son();

System.***out***.println("son name = "+son.*getName*());//2-子

Father father = **new** Father();

System.***out***.println("father name = "+father.*getName*());//0-父

Father f = **new** Son();

System.***out***.println("多态 name = "+f.*getName*());//2-子

}

}

返回结果：

son name = 2-子

father name = 0-父

多态 name = 0-父

结论：

重要的结论：

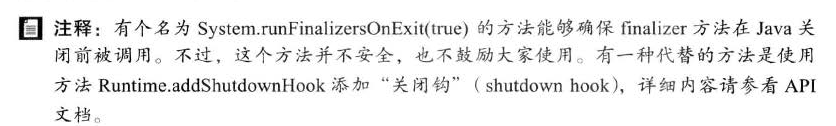
当多态时，恰好调用的是同名静态方法，调用的是父类的方法，而不是子类的静态方法。【即静态方法不存在复写。】

重点：

1. 子类不会继承父类的私有方法。
2. 如果父子类存在同名方法，但是父类方法是private,那么这两个方法就不是复写关系，因为父类私有方法子类感知不到，那么多态时，调用这个方法，实际调用的是父类的方法。。【即私有方法是不存在复写的。】（而且这个调用还必须是在父类方法中调用的，其他类中父类的对象是无法调用自己的私有方法的。）

* Finalize（）

1. 析构器：清理不需要使用的对象
2. Java有垃圾回收机制，没有析构器；
3. Finalize方法：在垃圾回收器处理之前，调用此方法。



### 包&权限修饰符

1. 什么类可以导入本类：
2. 本包中的类。
3. 其他包中的public类
4. 静态导入

**package** net.shopin.xyy;

**import** **static** java.lang.System.\*;

**import** **static** java.lang.Math.\*;

**public** **class** Son **extends** Father {

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

// System.out.println("xyy");

***out***.println("xyy");

*random*();//直接调用Math的静态方法。，不需要类名了。

}

}

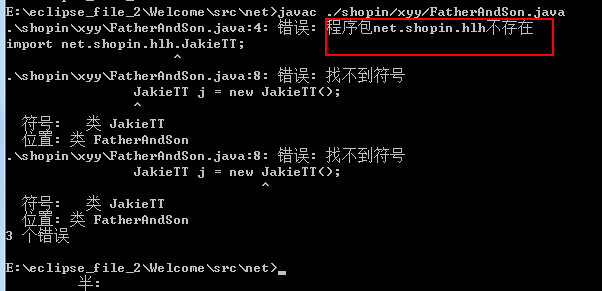
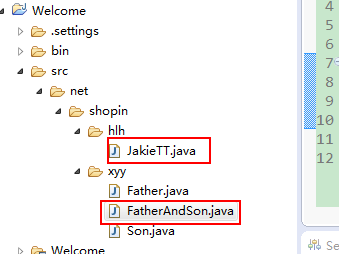
1. Javac和java命令(**重点**)

Javac:作用于【文件分隔符/和以.java结尾的文件】（编译器）

Java:用于加载类文件【.分隔符】（解释器）

注意事项：javac命令，如果主类中调用了其他包中的类，那么javac命令后，必须跟着项目的根目录，如下图目录结构:

FatherAndSon.java是主类，内部调用JakieTT，那么应该这么写：javac net.shopin.xyy.FatherAndSon.java (必须从根目录开始写,否则如下右图报错。)



1. 包的作用域

详情见markdown文档名为：**0-3权限修饰符**

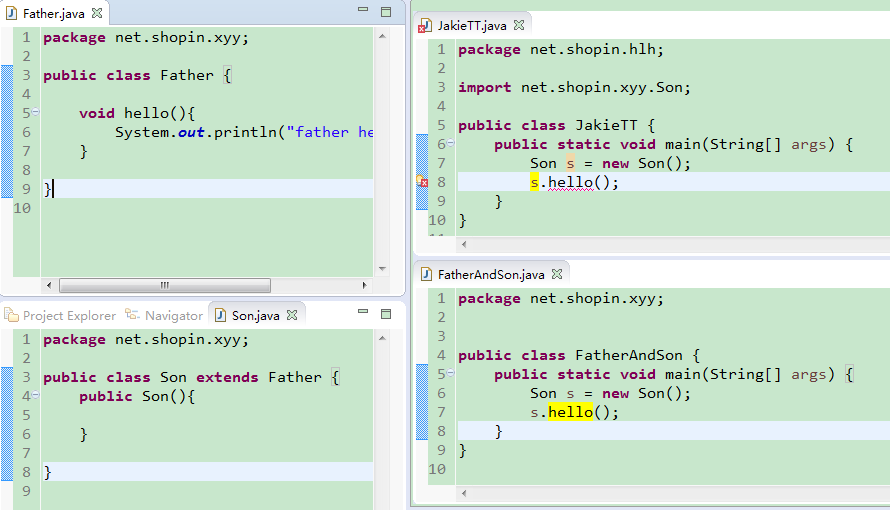
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Public | protected | default | private |
| 作用对象： | 类，方法，变量 | | | |
| 使用范围： | 所有类 | 子类以及同包类 | 同包类的所有方法 | 所在类 |
| 特征： |  |  |  | 方法：不可以被子类继承。 变量：不可以被子类继承。（setter，getter获取修改） |

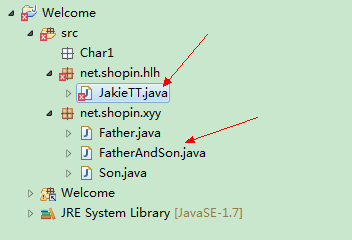
* 包封闭（第十章讲解）

问题描述：理解默认的方法只有同包下的类可以访问。 (**重点**)

答案：

1. 类A有一个默认权限的方法b();在c包下。【那么b()只可以在c包下访问。】
2. B()方法是类A的，自然是A的对象去调用方法b();
3. 但是A的对象不是在什么地方都可以调用方法b(),只有在C包的情况下，类A的对象才可以访问方法b();

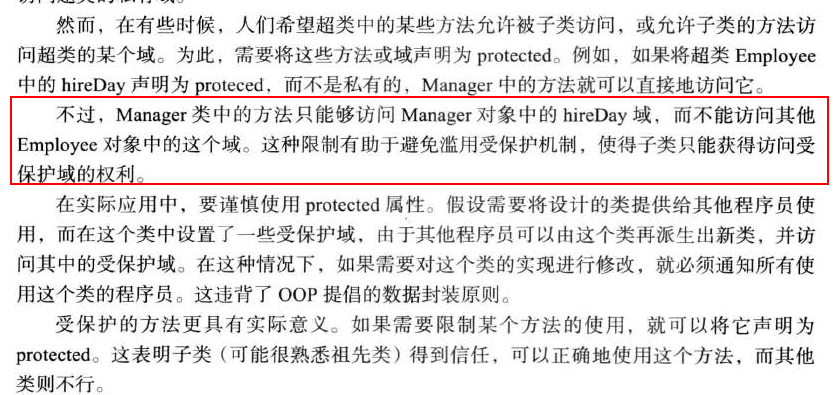




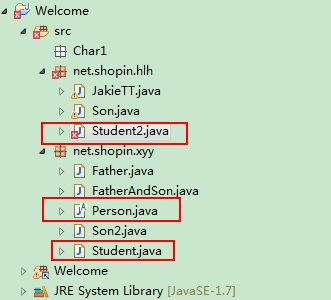
问题描述：理解protected (**重点**)

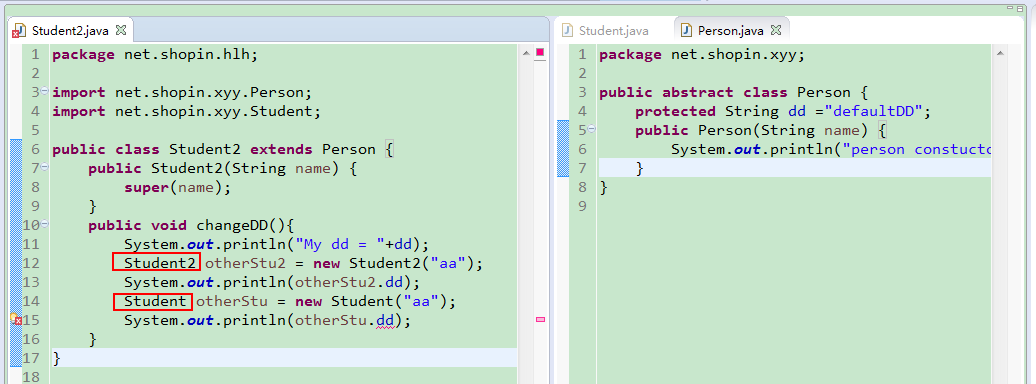
答案：被protected修改的变量，可以被子类（直接使用变量）或者同包类访问（对象调用）。但是注意一点：子类调用时，只可以是

在子类可以直接获取protected变量，如果是通过子类对象去调用父类protected变量是，那么A子类值可以获取A子类对象的protected变量，如果是其他子类，是无法通过对象.protected变量获取值的。



目录结构：

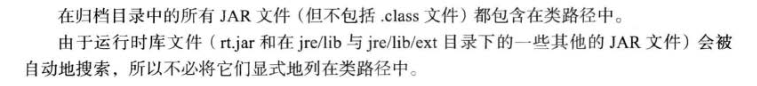


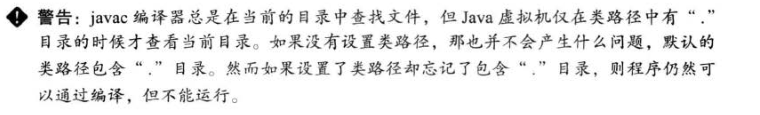


### ？类路径（最好自己上网详细了解）

1. 什么叫类路径：
2. 类路径的分隔符：

|  |  |  |  |
| --- | --- | --- | --- |
| 系统 | Windows | unix |  |
|  | 以点（.）表示当前目录 | |  |
| 类路径分隔符 | ；（分号） | ：(冒号) |  |





（javac编译默认是会在当前目录查找文件的，但是java命令需要查看类路径的配置，只有包含了点（.）,才会在当前目录中查找，但是类路径默认是包含点.的。）

1. 类文件的查找顺序

答：@jdk中包含的class文件@查看classpath配置的类路径，按照顺序查找

1. 编译器的任务：

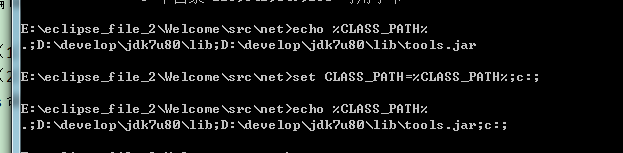
（1）

（2）查看源文件是否比类文件新，如果新，重新编译。

5. dos命令行查看和编辑classpath

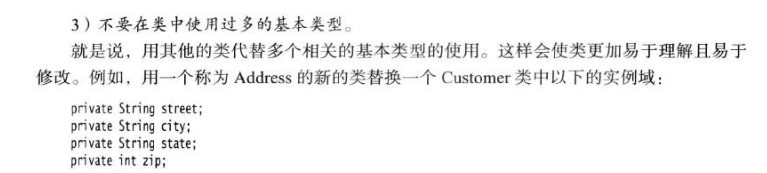
查看：echo %CLASS\_PATH%

临时编辑：set CLASS\_PATH=% CLASS\_PATH %其他【%%表示引用原来的设置】



### 文档注释（没看）

### 类的设计技巧

1. 数据：私有【成员变量】
2. 数据：最好初始化。【成员变量最好显示初始化，不要依赖系统默认的初始化。局部变量必须初始化】
3. 
4. 
5. 
6. 

## 第五章 继承（inheritance）

### 类&超类&子类

#### 概述

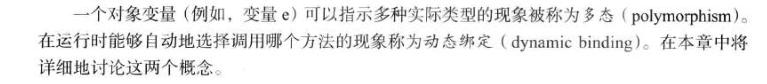
1. 继承了什么：方法和类。【公有继承：public】，在子类中是无法直接获取父类的私有变量的，需要通过getter来获取数据。
2. 方法：public,protected,默认修饰都可以继承到【private,static方法获取不到。】
3. 域：public【private获取不到。】
4. 反射(reflection)：在程序运行期间发现更多的类及其属性的能力。
5. 覆盖：（复写）
6. Super（**重点**）
7. 如果父子类存在同名方法，使用super.方法（）调用父类的方法。

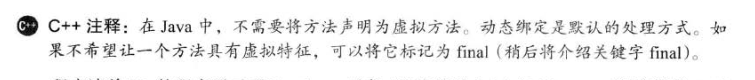


1. 构造方法中使用**super**关键字调用父类的构造器， 必须位于构造器的第一行。通过父类的构造器对父类的私有与进行初始化。
2. 子类的构造器默认去调用父类的无参构造，如果父类没有无参构造，子类又没有显示调用父类的其他构造，那么编译失败。
3. 调用其他构造器的语句，也必须位于构造方法的第一行。

This:@调用隐式参数（调用本类其他方法）@调用其他构造

Super:@调用父类的方法@调用父类构造

1. 父类的引用指向子类的对象
2. 这个对象只可以调用父类中有的方法，如果需要调用子类特有方法，需要强制转换。
3. 如果父子类存在同名方法，那么调用子类的方法。（private,static，final方法无法复写的，依然调用的是父类的）
4. 
5. 虚拟特征：



1. 多态&动态绑定
   1. 多态：一个对象变量可以指示多种实际类型的现象。(polymorphism)
   2. 动态绑定：在运行时能够自动的选择调用哪个方法的现象称为动态绑定（dynamic binding）

#### 继承层次

1. 名词：
2. 继承层次(inheritance hierarchy)：由一个超类派生出来的类的集合
3. 继承链(inheritance chain)：继承层次中，有一个类寻找其祖先的路径被称为该类的继承链。

#### 多态

注意以下示例：子类的数组，赋值给了父类类型的数组，然后通过父类类型的数组添加父类类型的对象，遍历子类数组时候，调用子类特有方法，报错。

**public** **class** FatherAndSon {

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

Son[] ss = **new** Son[2];

ss[0]= **new** Son();

ss[1]= **new** Son();

Father[] fs = **null**;

fs = ss;

fs[0] = **new** Father();

**for**(**int** i = 0;i<ss.length;i++){

Son son = ss[i];

son.sonTT();//子类特有方法sonTT();

}

}

}

**public** **class** Son **extends** Father {

**public** **void** sonTT(){

System.***out***.println("son...");

}

}

报错

father hello

Exception in thread "main" java.lang.ArrayStoreException: net.shopin.xyy.Father

at net.shopin.xyy.FatherAndSon.main(FatherAndSon.java:19)

#### \*动态绑定(重点)

1. 子类复写父类的方法：

@权限修饰要>=父类原有的。(即如果父类A方法权限修饰符为public,那么子类的权限修饰符必须>=public,所以只能是public)

比较大小：Public>protected>default>private

@返回类型<=父类的原有的。【即子类返回类型必须是父类的返回类型或者是父类返回类型的子类。】

1. 静态绑定：编译器准确的知道调用哪个方法，这就是静态绑定。

动态绑定：调用哪个类的方法依赖于隐式参数的实际类型，并且只有在运行时实现动态绑定。

Eg .像private,static,final修饰的方法，就属于静态绑定。

原因:

@private方法对本类外不可见，所以必定是由本类调用的。

@static 方法不允许子类复写的，所以多态方式调用的时候，就不存在调用子类的意思了。因为没有复写，所以是调用父类的方法了。【静态方法子类可以继承但是不可以复写。】所以，一旦变量声明是什么类型，那么调用的时候必然调用这个类型的这个静态方法，所以也是静态绑定。

@final方法和static方法一样，也是只能继承不能复写的。理由和static一致。

1. 采用动态绑定的方法，会去调用最合适的方法。

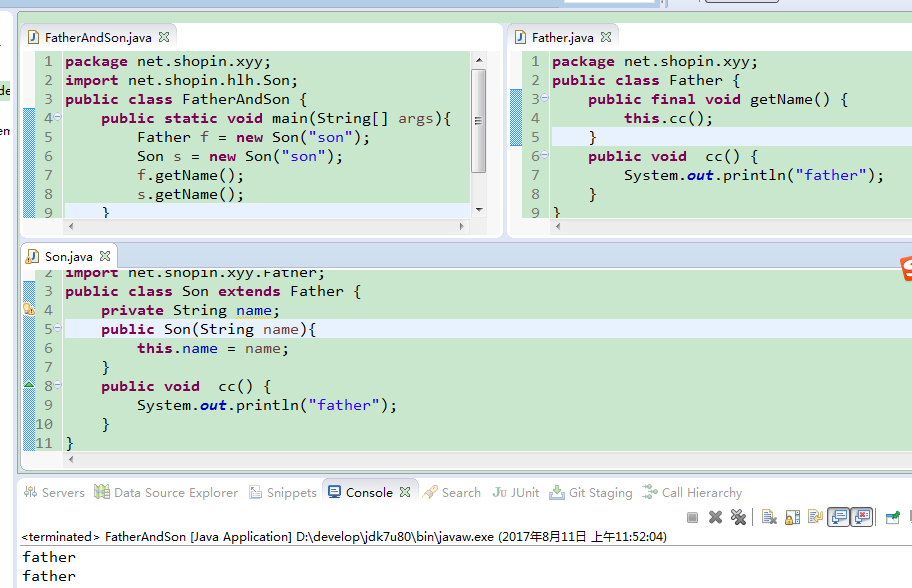
解释说明：当多态调用的时候，先去子类中寻找，如果子类复写方法，就调用子类的，子类没有复写，再调用父类方法。

1. 多态的方式调用，父类的引用只可以调用父类中声明的方法，子类特有的方法，必须进行**强制转换**（见3.3.6）

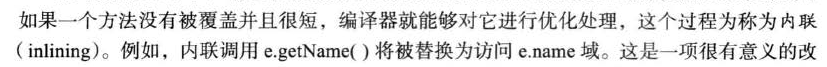
#### Final

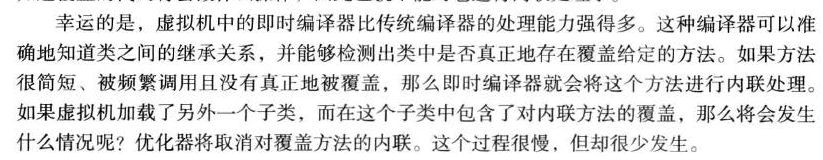
1. 修饰类：不可被继承【final类中的方法自动添加final，但是域不会。】
2. 修饰变量：常量
3. 修饰方法：不可被复写。
4. Final不存在多态性，不管是类还是方法。

@方法：因为final不可以复写，所以多态时调用的就是声明类型的方法，那么方法内部如果操作成员变量和方法，也必然是声明类型的成员变量和方法（不会是子类的）。【如下图所示】



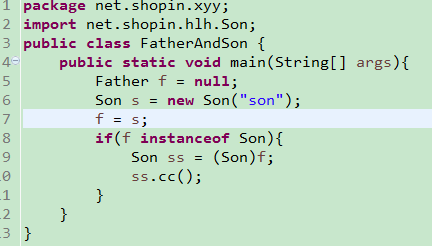


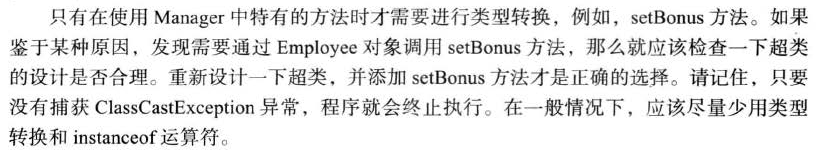




#### 强制类型转换

1. 进行强制转换之前，使用instanceof进行类型校验可以避免一些异常。

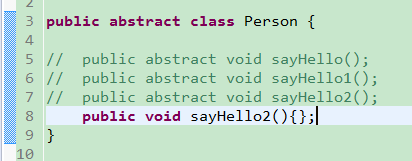


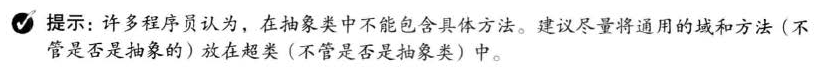


1. 强制类型：只可以在继承层次内进行装换。
2. 强制类型装换：多态情况下，需要使用子类特有方法，才会把对象强制转换成子类型。

#### 抽象类

1. 包含一个或者多个抽象方法的类必须声明为abstract.
2. 抽象类内的方法不一定都是抽象方法。【抽象类可以没有抽象方法】



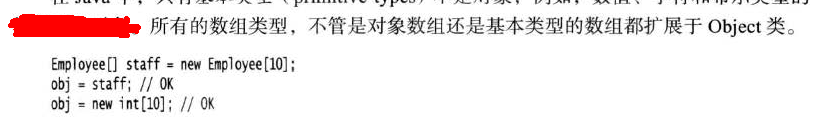


1. 继承抽象类的子类

@复写所有抽象方法，子类可以不声明为abstract

@不复写或者复写部分抽象方法，子类必须声明为abstract

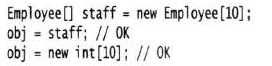
1. 抽象类不可以被实例化。但是可以创建一个抽象类的引用，指向非抽象子类的对象。
2. 多态情况下，如果父类是抽象类，那么引用指向的实际类型必然是这个抽象类的非抽象子类对象，因为抽象类是不可实例化的，那么调用的实际方法，也是子类 的方法实现。
3. 抽象类也需要构造方法，用于子类的实例化。【子类的构造方法至少有一个要去调用super(…),父类的构造方法】，初始化那些位于父类的域。



### Object

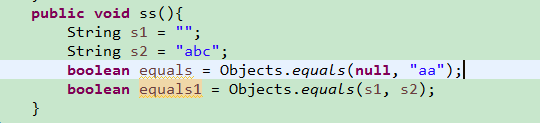


知识点：所有数组类型，不管是对象数组还是基本类型数组都是Object类的子孙。

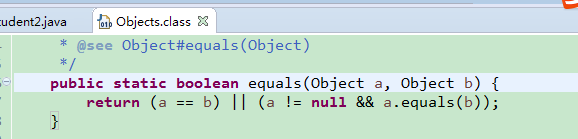


#### Equals

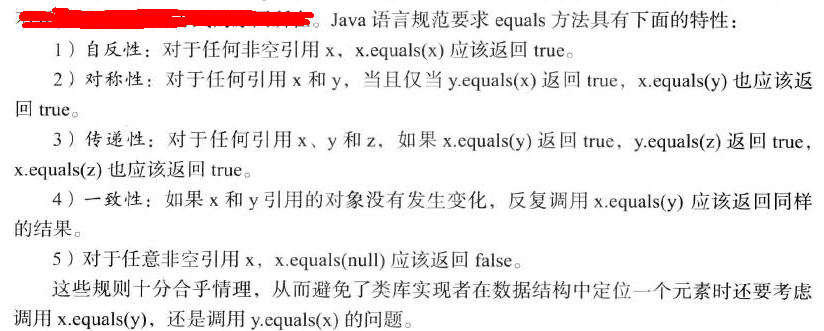
1. 从java1.7开始，可以使用如下方法代替equals

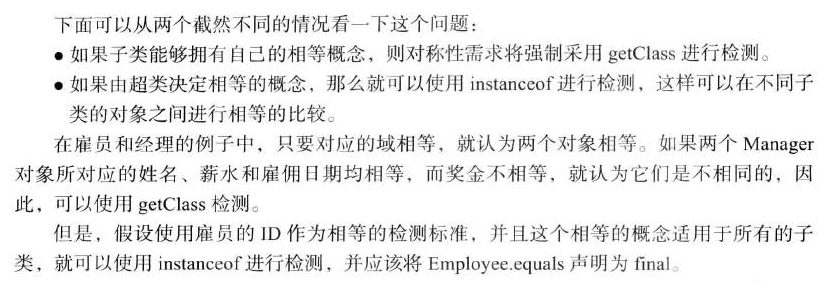


源码如下：

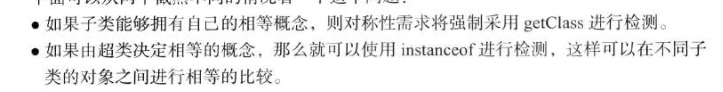


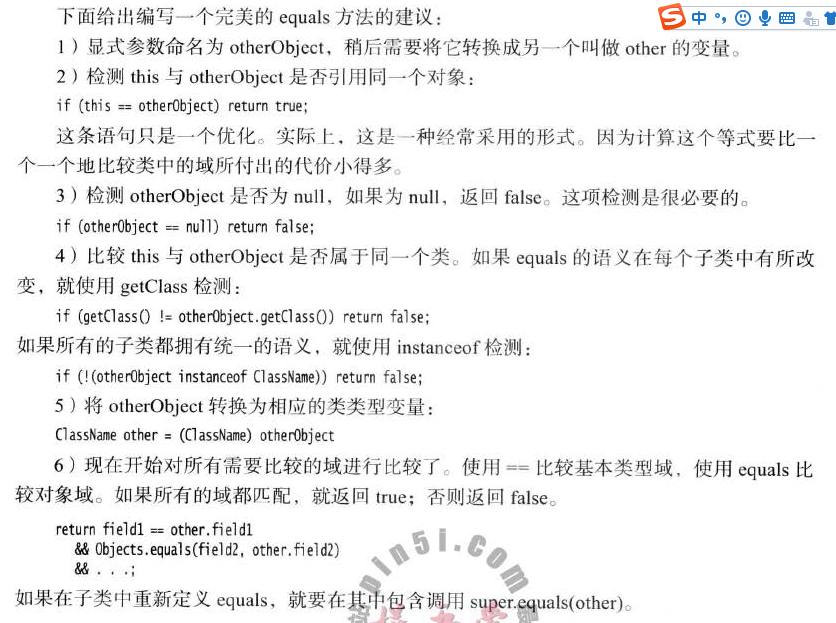
1. Equals规范





重要结论：如下





父类的equals:

@Override

**public** **boolean** equals(Object otherObj) {

// 地址值判断

**if**(otherObj == **this**) **return** **true**;

// 非null判断

**if**(otherObj == **null**) **return** **false**;

// 类型校验

**if**(! (otherObj **instanceof** Father)) **return** **false**;

// 强转

Father other = (Father) otherObj;

**if** (age == **null**) {

**if** (other.age != **null**)

**return** **false**;

} **else** **if** (!age.equals(other.age))

**return** **false**;

**return** **true**;

}

子类的equals:

@Override

**public** **boolean** equals(Object otherObj) {

// 调用父类

**if** (!**super**.equals(otherObj))

**return** **false**;

// 地址值判断

**if**(otherObj == **this**) **return** **true**;

// 非null判断

**if**(otherObj == **null**) **return** **false**;

// 类型校验

**if**(!**this**.getClass().equals(otherObj.getClass())) **return** **false**;

// 强转

Son other = (Son)otherObj;

// field校验

**return** Objects.*equals*(**this**.name, other.getName());

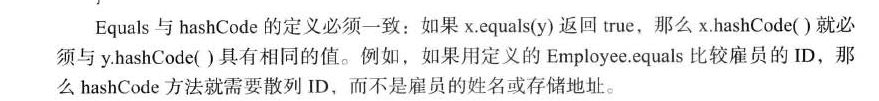
}

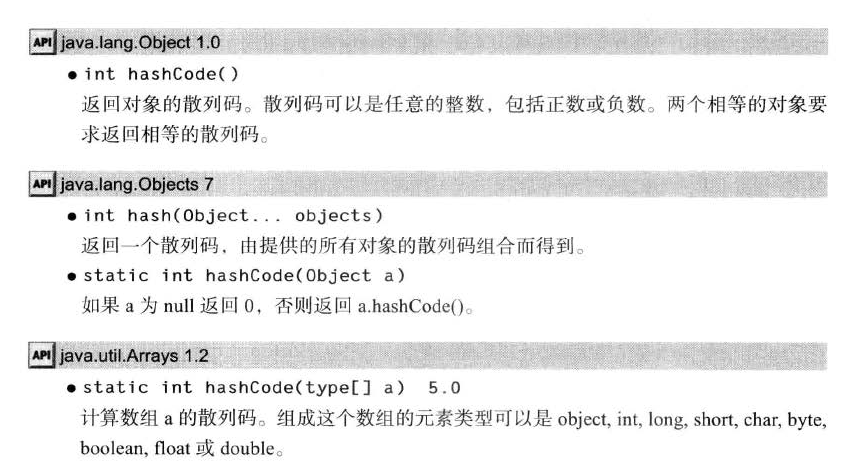
* Arrays.equals()比较对应位置的数据是否一致，一致，否则返回true;

#### Hashcode

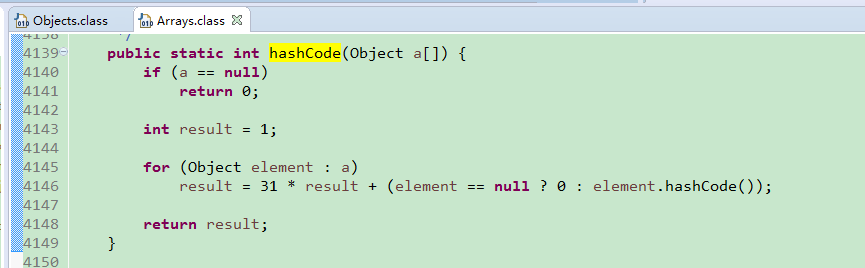
下面截图为什么？？关联set唯一性比较的判断，先判断hashcode(),如果一致，再去判断equals()是否一致。



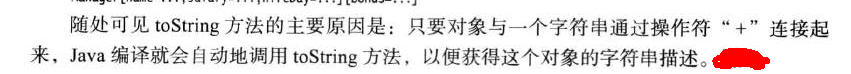




Arrays,Collections,Objects---工具类



#### toString



* 将对象和任意字符串通过“+”联结，java编译就会默认调用对象的toString()方法。

# 其他问题

## Eclipse enter键失效，总是弹出窗口

