笔记

[笔记 1](#_Toc26385313)

[JVM的划分 11](#_Toc26385314)

[Return后只能写一个值 11](#_Toc26385315)

[Void: 无返回值 11](#_Toc26385316)

[抽象方法没有方法主体（大括号） 11](#_Toc26385317)

[导包可以使用通配符把该目录下的类全部导入 11](#_Toc26385318)

[实参与形参 11](#_Toc26385319)

[方法参数是基本数据类型 11](#_Toc26385320)

[方法参数是引用数据类型 12](#_Toc26385321)

[方法的格式 13](#_Toc26385322)

[修饰符 返回值类型 方法名（参数列表 /\*逗号隔开\*/）{} 13](#_Toc26385323)

[重载（overload） 13](#_Toc26385324)

[方法重载注意事项 13](#_Toc26385325)

[a: 参数列表必须不同 13](#_Toc26385326)

[b: 重载和参数变量名无关 13](#_Toc26385327)

[c: 重载和返回值类型无关 13](#_Toc26385328)

[d: 重载和修饰符无关 13](#_Toc26385329)

[构造器也有重载 13](#_Toc26385330)

[重载的几种测试 13](#_Toc26385331)

[引用数据类型（类，数组，接口） 16](#_Toc26385332)

[创建引用数据类型的格式 数据类型 变量名 = new 数据类型（）； 17](#_Toc26385333)

[New 实例化对象 17](#_Toc26385334)

[参数列表为引用数据类型时 17](#_Toc26385335)

[形参内往往为接口，配合下面的格式，实现向上转型 17](#_Toc26385336)

[形参里的引用类型 引用名 = 实参（）；（一般为向上转型） 17](#_Toc26385337)

[实参为 new 引用类型 形参里的引用类型 引用名 = new 引用类型 向上转型 17](#_Toc26385338)

[实参为之前已实例化的对象 形参里的引用类型 引用名 = new 之前实例化对象的类型 17](#_Toc26385339)

[注意;实参必须是可以实例化对象的类型且是形参的实现类，原因在向上转型部分 17](#_Toc26385340)

[变量名保存的是一个内存地址 17](#_Toc26385341)

[自定义类 17](#_Toc26385342)

[定义类中属性时修饰符可以不写 17](#_Toc26385343)

[Cmd中的运行谁编译谁 17](#_Toc26385344)

[一个引用变量的内存图 18](#_Toc26385345)

[两个引用类型变量内存图 18](#_Toc26385346)

[ArrayList 19](#_Toc26385347)

[格式 19](#_Toc26385348)

[ArrayList中的方法 19](#_Toc26385349)

[注意事项 20](#_Toc26385350)

[Add存入的是什么类型的数据，get取出的就是什么类型的数据 20](#_Toc26385351)

[Add方法存入自定义类类型数据，直接取出打印会显示地址，应·用该自定义类的变量接收，再打印该变量的属性。 20](#_Toc26385352)

[百位十位个位的数如何求 20](#_Toc26385353)

[百位 i/100 20](#_Toc26385354)

[十位 i/10%10 20](#_Toc26385355)

[个位 i%10 20](#_Toc26385356)

[Char 与 int 20](#_Toc26385357)

[Char两字节 int四字节， 20](#_Toc26385358)

[两者计算 char自动转换为int型（根据ascii表） 20](#_Toc26385359)

[想要把数字转为编码表对应的char字符 需要强制类型准备转换；char c= (char)i 20](#_Toc26385360)

[Char中保存汉字 查询unicode （万国表） 20](#_Toc26385361)

[Char转为int类型取值0-65535 20](#_Toc26385362)

[Char和short均为两字节取值不同的原因 20](#_Toc26385363)

[冒泡排序问题汇总 21](#_Toc26385364)

[1.注意存在j+1，注意溢出问题 21](#_Toc26385365)

[2.冒泡方法中的i小于数组长度-1，所以直接输出数组【i】时会少一个元素 22](#_Toc26385366)

[面向对象的特性 继承 封装 多态 22](#_Toc26385367)

[成员变量和局部变量区别 22](#_Toc26385368)

[定义的位置不同 22](#_Toc26385369)

[在内存中的位置不同 22](#_Toc26385370)

[生命周期不同 22](#_Toc26385371)

[内存位置不同 23](#_Toc26385372)

[默认值不同 23](#_Toc26385373)

[局部变量生命周期 23](#_Toc26385374)

[Private 只修饰成员变量，不修饰局部变量； 23](#_Toc26385375)

[继承时 子类中有则优先自己 没有则使用父类的变量或方法 23](#_Toc26385376)

[方法“就近” 方法内有不使用则本类成员变量 23](#_Toc26385377)

[Super调用父类 23](#_Toc26385378)

[This 调用自己本类成员 23](#_Toc26385379)

[Get和set方法 24](#_Toc26385380)

[Get返回值 为成员变量数据类型 24](#_Toc26385381)

[Set返回值 为void 24](#_Toc26385382)

[重写override 24](#_Toc26385383)

[子类方法声明应该与父类相同 24](#_Toc26385384)

[重写父类方法 子类权限应该不小于父类 24](#_Toc26385385)

[权限由大到小 Public protected default private 24](#_Toc26385386)

[父类私有方法(private)子类不能重写 24](#_Toc26385387)

[抽象类 25](#_Toc26385388)

[抽象方法 25](#_Toc26385389)

[方法只有声明部分 25](#_Toc26385390)

[没有方法体（大括号） 25](#_Toc26385391)

[用abstract修饰 25](#_Toc26385392)

[抽象类： 包含抽象方法的类，一定是抽象类，使用 abstract 修饰的类 25](#_Toc26385393)

[抽象方法所在类一定是抽象类： 25](#_Toc26385394)

[抽象类与抽象方法都必须使用 abstract来修饰 25](#_Toc26385395)

[Abstract不能和 private， final， static一起使用 25](#_Toc26385396)

[抽象类不能直接实例化对象（不能 new 抽象类） 25](#_Toc26385397)

[抽象类中可以有抽象方法，也可以不写抽象方法 25](#_Toc26385398)

[若子类只重写了父类部分抽象方法，子类仍为抽象类 25](#_Toc26385399)

[继承（extends） 25](#_Toc26385400)

[获得父类所有的成员：成员变量与方法 25](#_Toc26385401)

[弊端：耦合性高 25](#_Toc26385402)

[耦合性：一个地方修改，得修改很多地方 25](#_Toc26385403)

[接口 26](#_Toc26385404)

[接口内只有抽象方法 26](#_Toc26385405)

[方法格式：Public abstract 返回值类型 方法名（参数列表）; 26](#_Toc26385406)

[接口定义中Public abstract可以不写或选择性书写，默认为全有，实现时必须写出该有的； 26](#_Toc26385407)

[接口中的方法均为public，接口内不写权限也默认为public 26](#_Toc26385408)

[接口内成员变量必须定义为常量，只有常量没有变量 26](#_Toc26385409)

[格式：public static final 数据类型 变量名 = 值； 26](#_Toc26385410)

[Static：可以直接被 类名. 的形式调用（其他为 对象.） 26](#_Toc26385411)

[Final ：变量值不可变，不可重新赋值，即使第二次赋值与第一次一样 26](#_Toc26385412)

[接口定义中public static final可以不写或选择性书写，默认为全有 26](#_Toc26385413)

[实现 implements 27](#_Toc26385414)

[实现类可以为抽象类 27](#_Toc26385415)

[接口中多实现（public void c implements a,b） 27](#_Toc26385416)

[可以多实现的原因 27](#_Toc26385417)

[多继承的安全隐患 27](#_Toc26385418)

[多实现中两接口内可以有重名方法，但若是两方法返回值不同，子类实现方法会报错（重复定义），两种返回值都实现或只实现一种全都报错 27](#_Toc26385419)

[继承的同时多实现 （extends b implements c，d（）{}） 29](#_Toc26385420)

[接口的多继承 29](#_Toc26385421)

[接口之间有多继承，类没有 29](#_Toc26385422)

[接口的好处 29](#_Toc26385423)

[1.接口的出现扩展了功能。 29](#_Toc26385424)

[2.接口其实就是暴漏出来的规则。 29](#_Toc26385425)

[3.接口的出现降低了耦合性，即设备与设备之间实现了解耦。 29](#_Toc26385426)

[接口和抽象类的区别： 29](#_Toc26385427)

[相同点: 29](#_Toc26385428)

[区别: 29](#_Toc26385429)

[二者的选用: 29](#_Toc26385430)

[多态 30](#_Toc26385431)

[需要有继承或实现关系 30](#_Toc26385432)

[父类 变量名 = new 子类（） 30](#_Toc26385433)

[成员变量 编译运行全看父类 30](#_Toc26385434)

[成员方法 编译看父类，运行看子类 30](#_Toc26385435)

[Instanceof ：比较对象是否是某引用类型 30](#_Toc26385436)

[返回类型：boolean 30](#_Toc26385437)

[格式：Boolean b = 变量名 instanceof 引用类型； 30](#_Toc26385438)

[注意：需要变量所属类需要和instanceof后的引用类型存在继承或实现关系，否则报错 30](#_Toc26385439)

[多态中的转型 30](#_Toc26385440)

[向上转型 父类 变量名 = new 子类（） 30](#_Toc26385441)

[优点：可以调用父类子类公共方法 30](#_Toc26385442)

[缺点：无法调用子类特有方法（调用父类没有的方法报错） 30](#_Toc26385443)

[向下转型 将向上转型的对象强制类型转换成子类 30](#_Toc26385444)

[子类 新变量名 = （子类类型）原变量名 30](#_Toc26385445)

[优点：可以调用子类特有方法 30](#_Toc26385446)

[注意：应转回原本子类，转成其他类型，编译无错，运行报类型转换错误； 30](#_Toc26385447)

[注意：可使用if与instanceof避免出错 30](#_Toc26385448)

[构造方法（构造器） 31](#_Toc26385449)

[格式：权限修饰符 方法名（参数列表）{} 31](#_Toc26385450)

[方法名必须与类名一模一样 31](#_Toc26385451)

[无返回值，void也无，可以写return （无意义） 31](#_Toc26385452)

[有方法主体 31](#_Toc26385453)

[每一次new执行一次，也只执行一次 31](#_Toc26385454)

[Class中必有构造方法，不写也有 31](#_Toc26385455)

[写了有参构造方法，new 数据类型（）要加上参数，否则报错 32](#_Toc26385456)

[构造方法的内存图（在new对象赋默认值之后，把内存地址传给变量名之前调用） 32](#_Toc26385457)

[构造器的重载 32](#_Toc26385458)

[构造方法可以修改权限修饰符，但权限为private无法被其他类调用 32](#_Toc26385459)

[构造方法如果赋值了成员变量，也有必要写get和set方法 32](#_Toc26385460)

[This（）与super（） 33](#_Toc26385461)

[小括号中为空或参数列表 33](#_Toc26385462)

[This（）：用于调用本类的构造方法（根据参数列表） 33](#_Toc26385463)

[super（）：用于调用父类构造方法（根据参数列表） 33](#_Toc26385464)

[This（）与super（）都必须在构造方法第一条 33](#_Toc26385465)

[This（）与super（）两者冲突 33](#_Toc26385466)

[子类this()与super（）都不写 默认为super()； 33](#_Toc26385467)

[继承关系中先在父类写有参构造方法后子类报错 因为子类构造方法第一句默认的super（）为无参 33](#_Toc26385468)

[子类构造器无论重载多少个 都要注意默认super（）为无参的问题； 33](#_Toc26385469)

[默认为super（）是因为子类构造方法只能选其一 但必须要保证可以调用父类的构造方法，允许间接调用 33](#_Toc26385470)

[子类构造方法可以间接调用父类构造方法 图 33](#_Toc26385471)

[Super（）用于了解父类成员变量的赋值。 34](#_Toc26385472)

[父类构造方法中第一行没必要super 因为父类的父类是 object（object类的构造方法为空参）没必要写 34](#_Toc26385473)

[子类调用父类构建方法的内存图 34](#_Toc26385474)

[在方法区中父类class文件先进去，子类后进入并根据super()获取父类地址 34](#_Toc26385475)

[创建子类对象 对象空间分为两个区域，一个留给父类，一个留个自己，故父类成员变量跟随子类进入堆内存 34](#_Toc26385476)

[父类有多个构建方法 子类可以任选其一调用 调用不同的构建方法也可以 34](#_Toc26385477)

[Final 用来修饰类，类的成员，以及局部变量 35](#_Toc26385478)

[final修饰类不可以被继承，但是可以继承其他类 35](#_Toc26385479)

[final修饰的方法不可以被覆盖,但父类中没有被final修饰方法，子类覆盖后可以加final 35](#_Toc26385480)

[final修饰的变量称为常量，这些变量只能赋值一次。 35](#_Toc26385481)

[final修饰的变量，所记录的地址值不能改变，可以更改对象中的属性值 35](#_Toc26385482)

[修饰成员变量，需要在创建对象前赋值，否则报错。(当没有显式赋值时，多个构造方法的均需要为其赋值。) 35](#_Toc26385483)

[可以在创建对象时所调用的构造方法中，为变量n赋值 35](#_Toc26385484)

[static 35](#_Toc26385485)

[被static修饰的成员变量属于类，不属于这个类的某个对象。（多对象共用） 35](#_Toc26385486)

[被static修饰的成员可以并且建议通过类名直接访问 35](#_Toc26385487)

[静态内容是优先于对象存在，只能访问静态，不能使用this/super。静态修饰的内容存于静态区 36](#_Toc26385488)

[同一个类中，静态成员只能访问静态成员 36](#_Toc26385489)

[main方法为静态方法仅仅为程序执行入口，它不属于任何一个对象，可以定义在任意类中。 36](#_Toc26385490)

[静态常量 36](#_Toc26385491)

[public static final修饰 36](#_Toc26385492)

[变量名用全部大写，多个单词使用下划线连接 36](#_Toc26385493)

[接口中的每个成员变量都默认使用public static final修饰。 36](#_Toc26385494)

[所有接口中的成员变量已是静态常量，由于接口没有构造方法，所以必须显示赋值。可以直接用接口名访问。 36](#_Toc26385495)

[匿名对象 36](#_Toc26385496)

[匿名对象在没有指定其引用变量时，只能使用一次 36](#_Toc26385497)

[匿名对象可以作为方法接收的参数、方法返回值使用 36](#_Toc26385498)

[内部类 37](#_Toc26385499)

[定义 37](#_Toc26385500)

[内部类的分类 37](#_Toc26385501)

[成员内部类 37](#_Toc26385502)

[局部内部类。 37](#_Toc26385503)

[特点 37](#_Toc26385504)

[内部类可以直接访问外部类的成员，包含私有的成员 37](#_Toc26385505)

[成员内部类 37](#_Toc26385506)

[定义格式 37](#_Toc26385507)

[访问方式 37](#_Toc26385508)

[外部类名.内部类名 变量名 = new 外部类名().new 内部类名(); 37](#_Toc26385509)

[局部内部类 37](#_Toc26385510)

[定义格式 38](#_Toc26385511)

[访问方式 38](#_Toc26385512)

[匿名内部类 局部内部类的一种 38](#_Toc26385513)

[访问修饰符 38](#_Toc26385514)

[要想仅能在本类中访问使用private修饰； 39](#_Toc26385515)

[要想本包中的类都可以访问不加修饰符即可； 39](#_Toc26385516)

[要想本包中的类与其他包中的子类可以访问使用protected修饰 39](#_Toc26385517)

[要想所有包中的所有类都可以访问使用public修饰。 39](#_Toc26385518)

[注意：如果类用public修饰，则类名必须与文件名相同。一个文件中只能有一个public修饰的类 39](#_Toc26385519)

[修饰符使用细节 39](#_Toc26385520)

[常用来修饰类、方法、变量的修饰符 39](#_Toc26385521)

[ public 权限修饰符，公共访问, 类,方法,成员变量 39](#_Toc26385522)

[ protected 权限修饰符，受保护访问, 方法,成员变量 39](#_Toc26385523)

[ 默认什么也不写 也是一种权限修饰符，默认访问, 类,方法,成员变量 39](#_Toc26385524)

[ private 权限修饰符，私有访问, 方法,成员变量 39](#_Toc26385525)

[ static 静态修饰符 方法,成员变量 39](#_Toc26385526)

[ final 最终修饰符 类,方法,成员变量,局部变量 39](#_Toc26385527)

[ abstract 抽象修饰符 类 ,方法 39](#_Toc26385528)

[修饰类能够使用的修饰符： 39](#_Toc26385529)

[Public 39](#_Toc26385530)

[默认的 39](#_Toc26385531)

[final 39](#_Toc26385532)

[abstract关键字 39](#_Toc26385533)

[修饰成员变量能够使用的修饰符 40](#_Toc26385534)

[public : 公共的 40](#_Toc26385535)

[protected : 受保护的 40](#_Toc26385536)

[：默认的 40](#_Toc26385537)

[private ：私有的 40](#_Toc26385538)

[final : 最终的 40](#_Toc26385539)

[static : 静态的 40](#_Toc26385540)

[修饰构造方法能够使用的修饰符： 40](#_Toc26385541)

[public : 公共的 40](#_Toc26385542)

[protected : 受保护的 40](#_Toc26385543)

[: 默认的 40](#_Toc26385544)

[private ：私有的 40](#_Toc26385545)

[修饰成员方法能够使用的修饰符： 40](#_Toc26385546)

[public : 公共的 40](#_Toc26385547)

[protected : 受保护的 40](#_Toc26385548)

[: 默认的 40](#_Toc26385549)

[private ：私有的 40](#_Toc26385550)

[final : 最终的 40](#_Toc26385551)

[static : 静态的 40](#_Toc26385552)

[abstract : 抽象的 40](#_Toc26385553)

[代码块 41](#_Toc26385554)

[局部代码块 41](#_Toc26385555)

[构造代码块 41](#_Toc26385556)

[特点： 41](#_Toc26385557)

[优先于构造方法执行，构造代码块用于执行所有对象均需要的初始化动作 41](#_Toc26385558)

[每创建一个对象均会执行一次构造代码块。 41](#_Toc26385559)

[静态代码块 41](#_Toc26385560)

[特点： 42](#_Toc26385561)

[它优先于主方法执行、优先于构造代码块执行，当以任意形式第一次使用到该类时执行。 42](#_Toc26385562)

[该类不管创建多少对象，静态代码块只执行一次。 42](#_Toc26385563)

[可用于给静态变量赋值，用来给类进行初始化。 42](#_Toc26385564)

[总结 42](#_Toc26385565)

[局部代码块：定义在方法中的，用来限制变量的作用范围 42](#_Toc26385566)

[构造代码块：定义在类中方法外，用来给对象中的成员初始化赋值 42](#_Toc26385567)

[静态代码块：定义在类中方法外，用来给类的静态成员初始化赋值 42](#_Toc26385568)

[自定义数据类型的使用 42](#_Toc26385569)

[类作为方法参数时，说明要向方法中传入该类的对象 42](#_Toc26385570)

[类作为方法返回值时，说明该方法要返回一个该类的对象。 42](#_Toc26385571)

[抽象类作为方法参数时，说明要传入一个实现抽象类所有抽象方法的子类对象。 42](#_Toc26385572)

[抽象类作为方法返回值时，说明需要返回一个实现抽象类所有抽象方法的子类对象。 42](#_Toc26385573)

[接口作为方法参数时，说明该方法要传入一个接口实现类对象。 42](#_Toc26385574)

[接口作为方法返回值时，说明该方法需要返回一个接口实现类对象。 42](#_Toc26385575)

# JVM的划分

# Return后只能写一个值

# Void: 无返回值

调用方法的时候,返回值是void, 不能写在输出语句中。

错误: 此处不允许使用 '空' 类型

# 抽象方法没有方法主体（大括号）

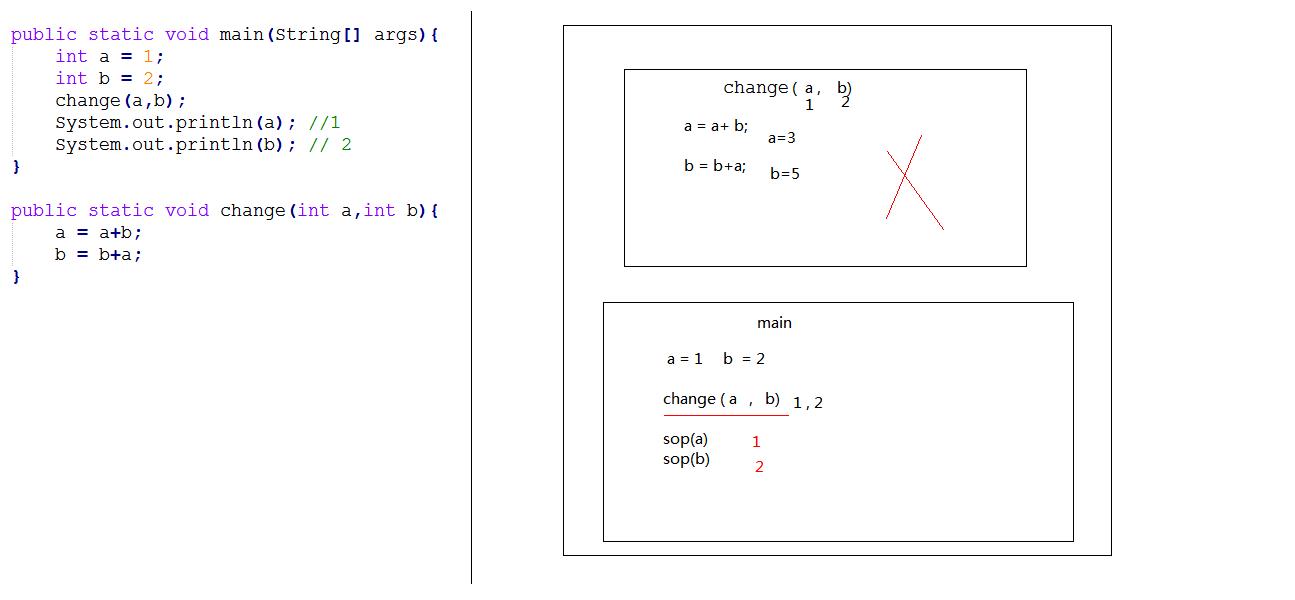
# 导包可以使用通配符把该目录下的类全部导入

如java.util.\*

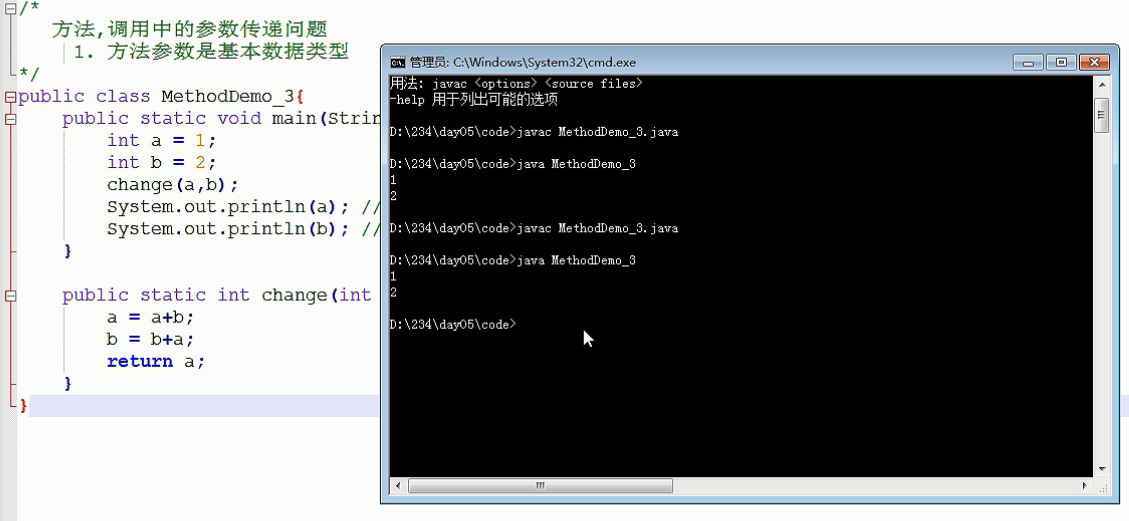
# 实参与形参

## 方法参数是基本数据类型

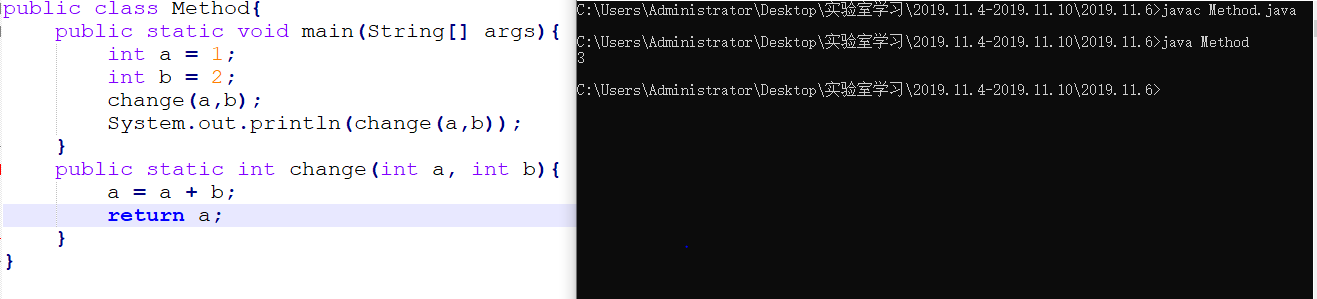
方法参数是基本类型时，传递的是值。/\*返回后不改变主方法变量的值，即使变量名一样。除非将方法返回值 赋值给主方法变量 且下图也无返回值 \*/



Tip 只传递给change方法数值，方法内的变量ab运算后随着方法从内存一起释放，故完全不影响main方法ab变量



Tip 虽然有返回a 但是返回的也是一个数值，不是变量。即change（a，b）== 3，输出change（a，b），值为3. 如果将返回值赋值给变量，即写成a = change（a，b），输出a为3



## 方法参数是引用数据类型

方法参数是引用类型时，传递的是内存地址值。/\*数组名为地址，调用的方法凭借地址对数组进行更改 （合租）\*/

# 方法的格式

## 修饰符 返回值类型 方法名（参数列表 /\*逗号隔开\*/）{}

方法名首字母小写其后首字母大写

未调用时 方法的形参没有具体值

# 重载（overload）

## 方法重载注意事项

### a: 参数列表必须不同

### b: 重载和参数变量名无关

### c: 重载和返回值类型无关

### d: 重载和修饰符无关

在同一个类中，允许方法名相同，只需参数列表不同（包括参数个数，数据类型，顺序）只看参数列表

## 构造器也有重载

## 重载的几种测试

结果摘要

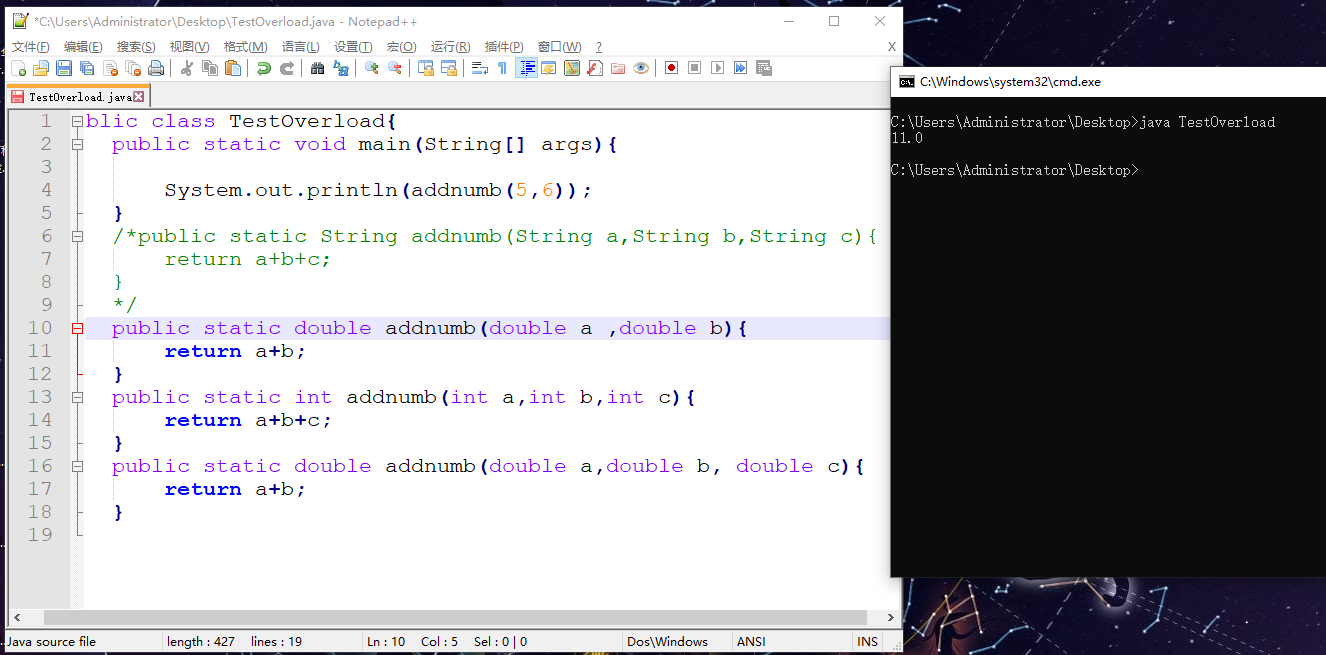
1.优先以参数个数为基准匹配方法

2.整型默认为int， int型实参可以自动转为double传给形参

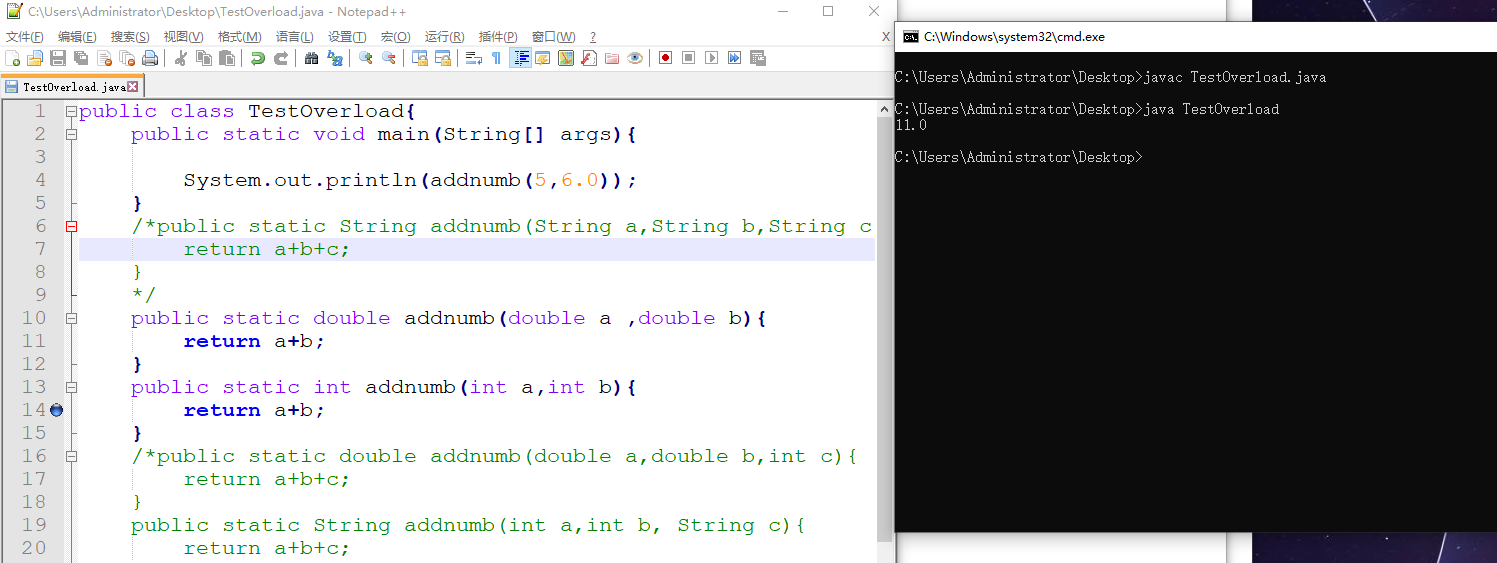
3. int型无法转为String，不兼容

4. char型可以取表值看作int计算

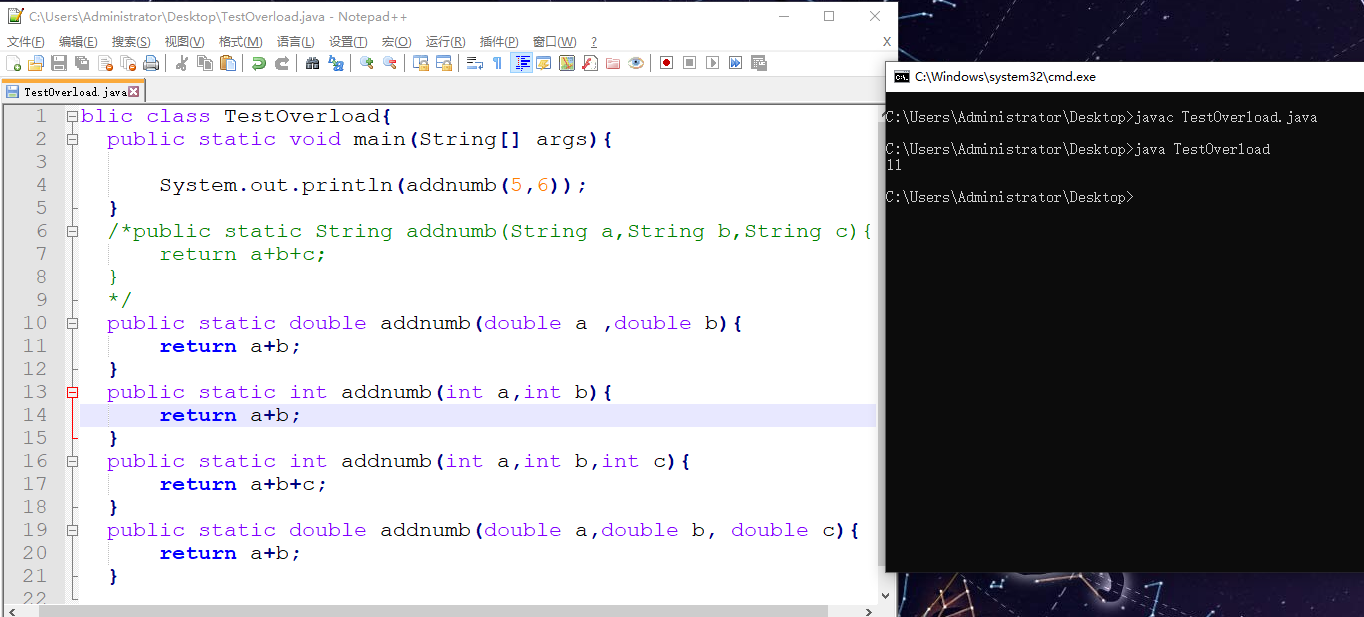
1.以参数个数为基准



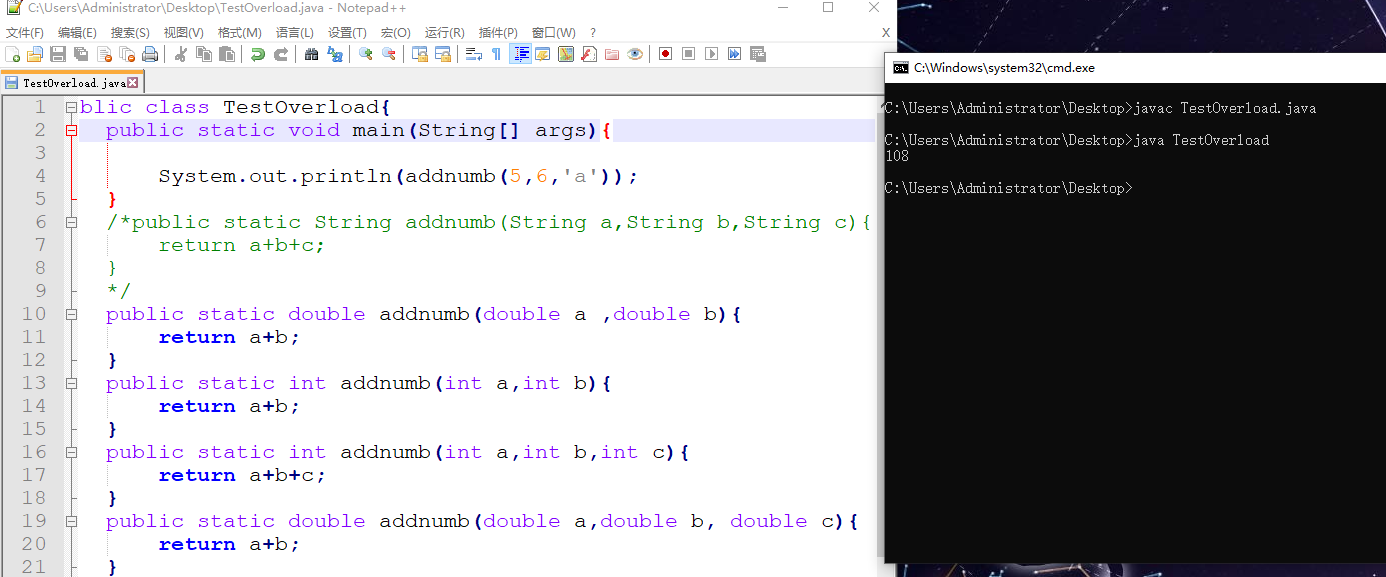
2.int型实参可以自动转为double传给形参



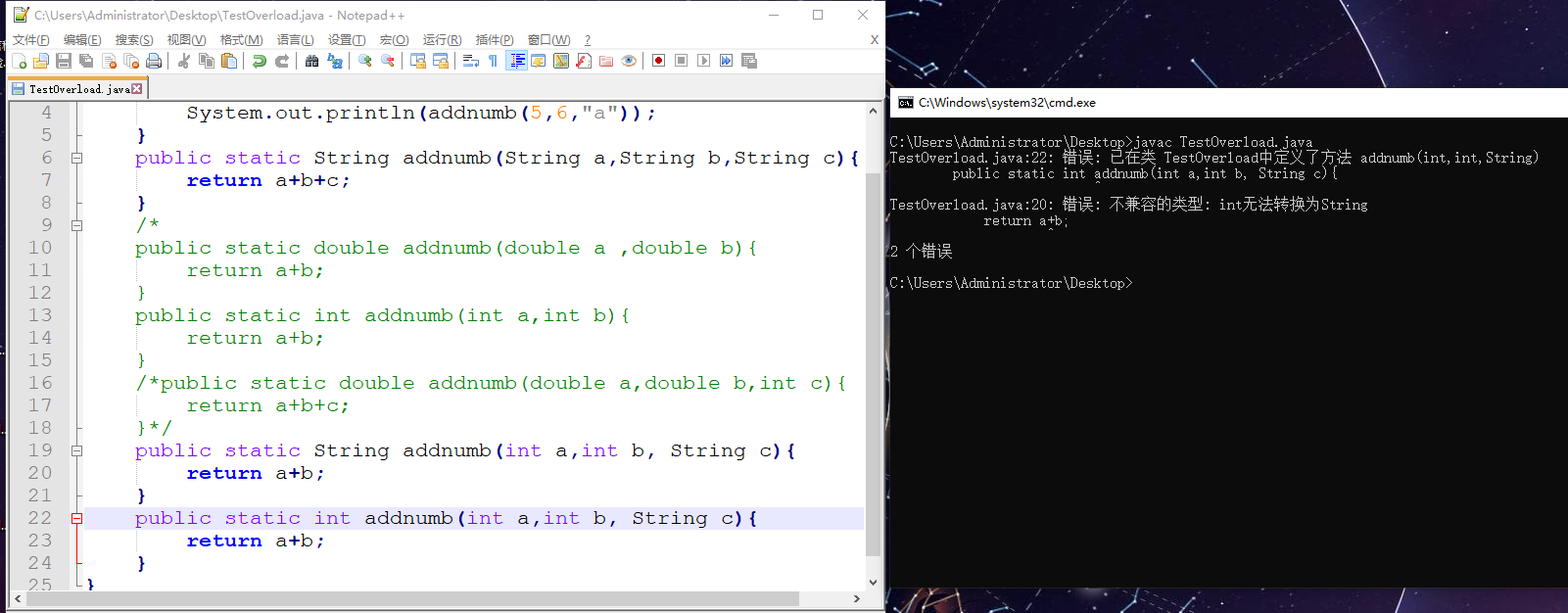
3.实参为整型时，默认为int



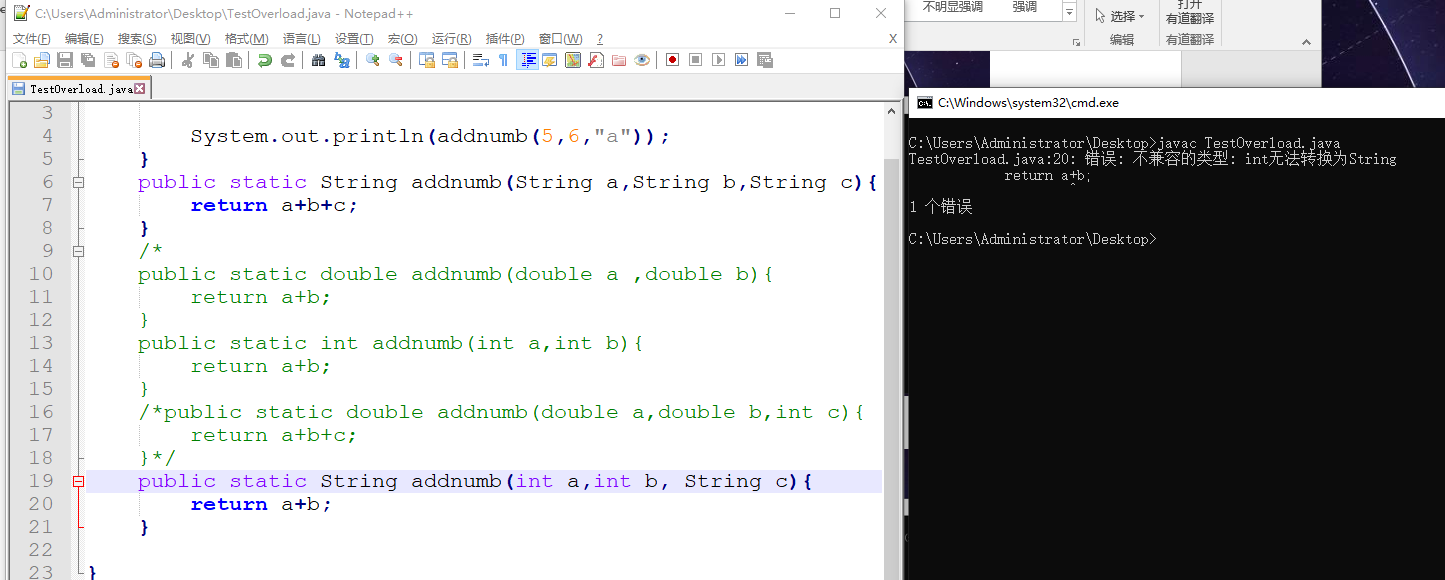
4.char型可以取表值看作int计算



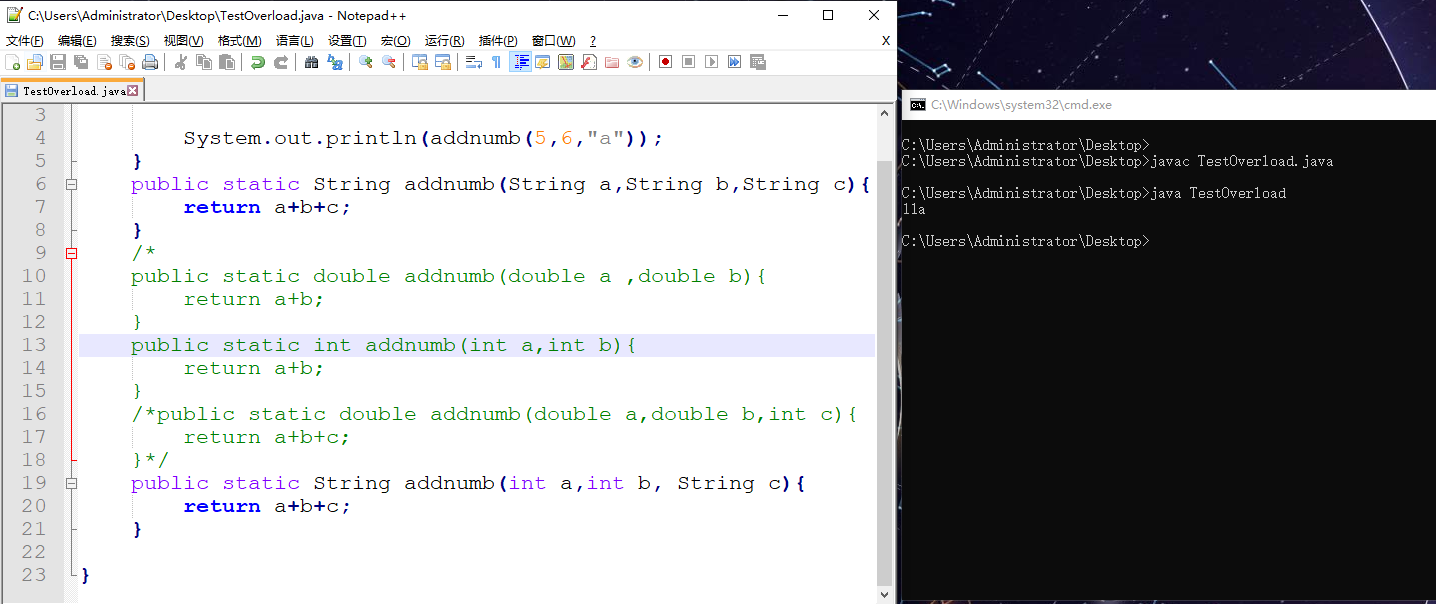
5.重载只和参数列表有关，与返回值类型无关



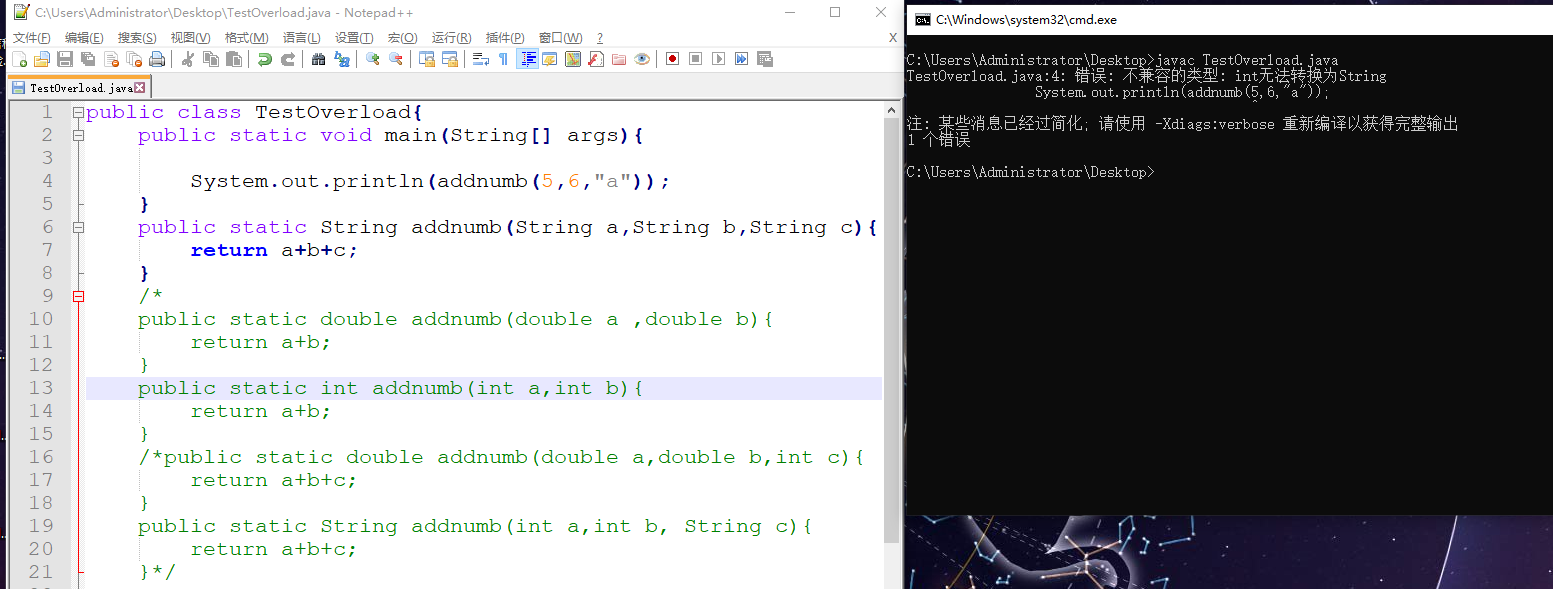
6.int型无法转为String，不兼容



7.返回值为String的情况



8.实参为int不能传给为String的形参



# 引用数据类型（类，数组，接口）

默认值为内存地址，即创建后直接输出显示内存地址。

## 创建引用数据类型的格式 数据类型 变量名 = new 数据类型（）；

# New 实例化对象

# 参数列表为引用数据类型时

## 形参内往往为接口，配合下面的格式，实现向上转型

## 形参里的引用类型 引用名 = 实参（）；（一般为向上转型）

### 实参为 new 引用类型 形参里的引用类型 引用名 = new 引用类型 向上转型

### 实参为之前已实例化的对象 形参里的引用类型 引用名 = new 之前实例化对象的类型

### 注意;实参必须是可以实例化对象的类型且是形参的实现类，原因在向上转型部分

# 变量名保存的是一个内存地址

# 自定义类

类中的属性是变量——>变量可以赋值取值

属性随类进入堆内存，赋值为默认值

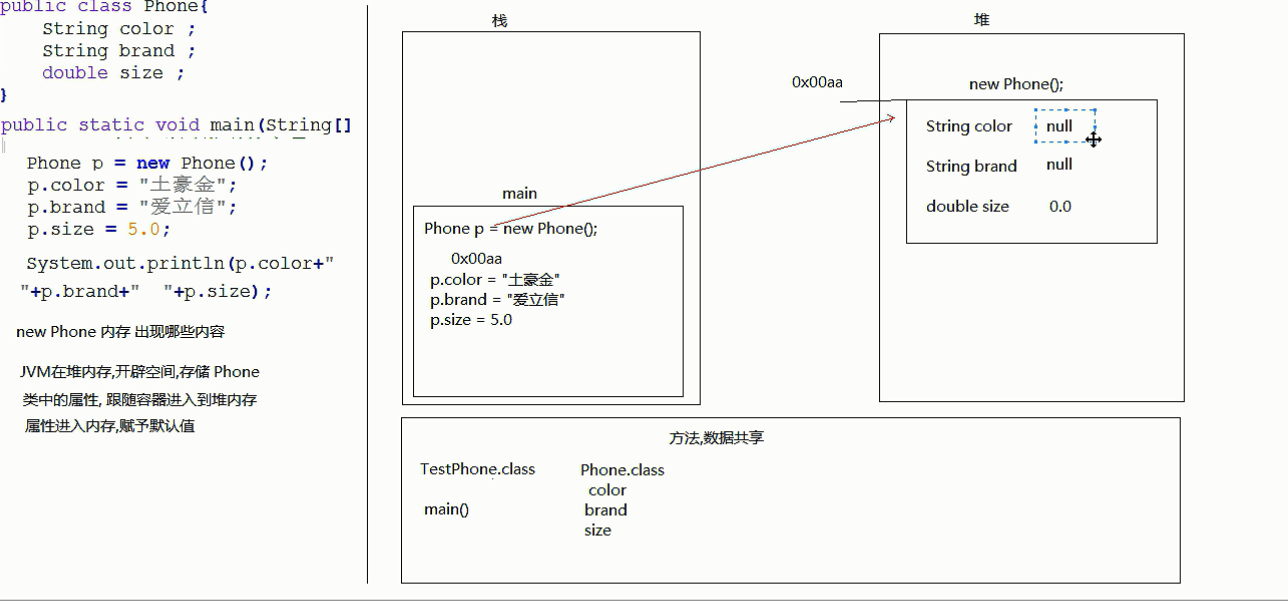
## **定义类中属性时修饰符可以不写**

取决于想要给予的权限，

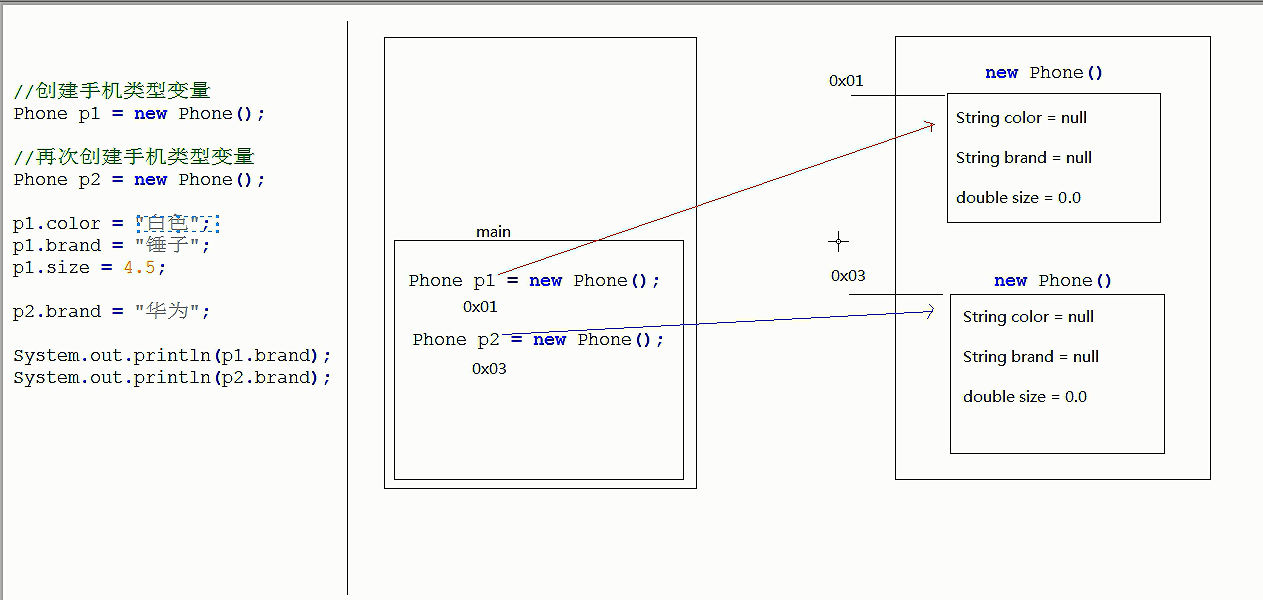
# Cmd中的运行谁编译谁

Cmd中运行谁编译谁即可，因为被调用的类也会被自动编译。

## 一个引用变量的内存图

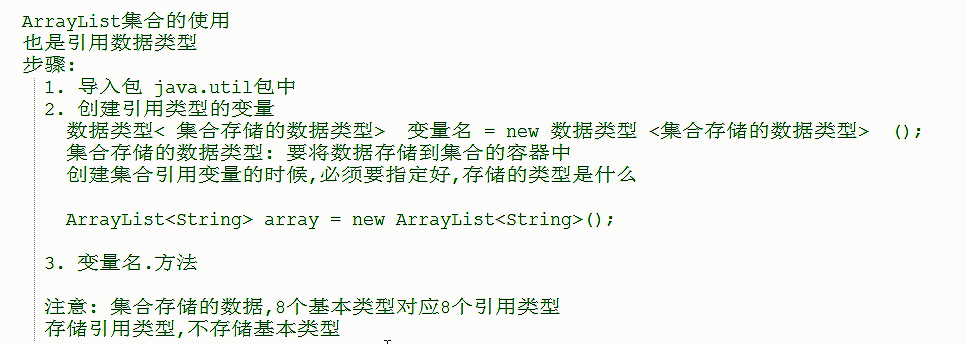


## 两个引用类型变量内存图



# ArrayList

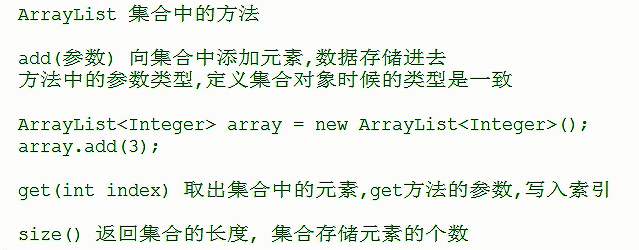
## 格式





ArrayList<> 变量名 = new ArrayList<>中的<>放引用类型均可，不只以上八种，包括自定义类。

## ArrayList中的方法



## 注意事项

### Add存入的是什么类型的数据，get取出的就是什么类型的数据

### Add方法存入自定义类类型数据，直接取出打印会显示地址，应·用该自定义类的变量接收，再打印该变量的属性。

# 百位十位个位的数如何求

## 百位 i/100

## 十位 i/10%10

## 个位 i%10

# Char 与 int

## Char两字节 int四字节，

## 两者计算 char自动转换为int型（根据ascii表）

## 想要把数字转为编码表对应的char字符 需要强制类型准备转换；char c= (char)i

## Char中保存汉字 查询unicode （万国表）

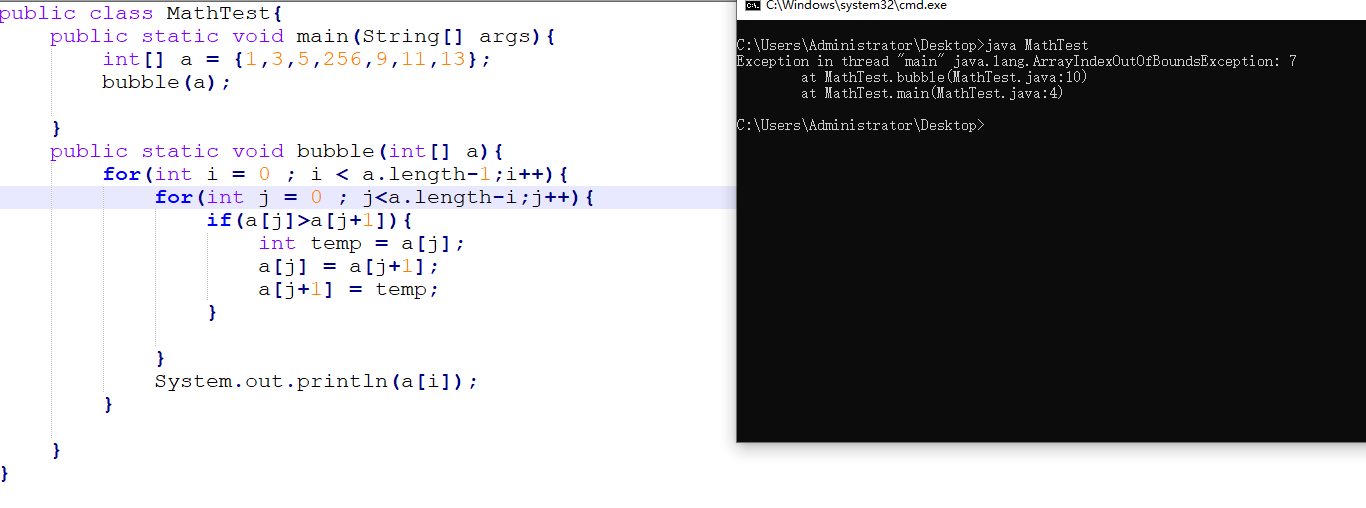
## Char转为int类型取值0-65535

## Char和short均为两字节取值不同的原因

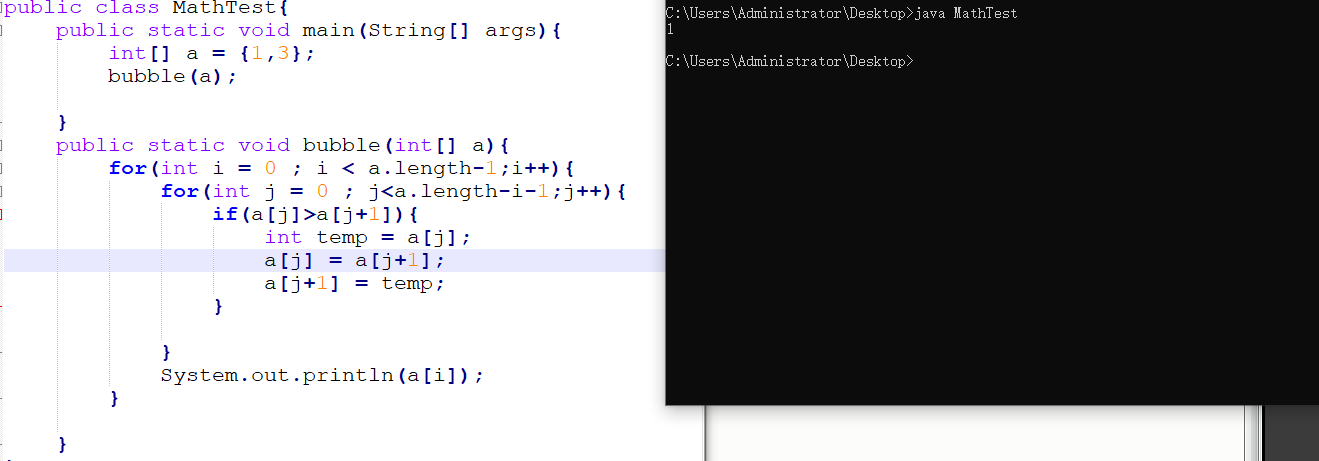


# 冒泡排序问题汇总

## 1.注意存在j+1，注意溢出问题



## 2.冒泡方法中的i小于数组长度-1，所以直接输出数组【i】时会少一个元素



# 面向对象的特性 继承 封装 多态

# 成员变量和局部变量区别

## 定义的位置不同

\* 定义在类中的变量是成员变量

\* 定义在方法中或者{}语句里面的变量是局部变量

## 在内存中的位置不同

\* 成员变量存储在对内存的对象中

\* 局部变量存储在栈内存的方法中

## 生命周期不同

\* 成员变量随着对象的出现而出现在堆中，随着对象的消失而从堆中消失

\* 局部变量随着方法的运行而出现在栈中，随着方法的弹栈而消失

## 内存位置不同

\* 成员变量因为在堆内存中

\* 局部变量没有默认的初始化值，。

## 默认值不同

\*成员变量有默认的初始化值

\*局部变量无默认值，不赋值不能使用，必须手动的给其赋值才可以使用

# 局部变量生命周期

只在定义时所在的方法或代码段内；

如for（int i= 0 ；；） i的生命周期只在for中

# **Private 只修饰成员变量，不修饰局部变量**；

# 继承时 子类中有则优先自己 没有则使用父类的变量或方法

# 方法“就近” 方法内有不使用则本类成员变量

方法内的局部变量并非真正的“覆盖“成员变量 因为一个在栈中 一个进入堆

# Super调用父类

# This 调用自己本类成员

带this.的是成员变量

# Get和set方法

在父类成员变量权限为private时提供调用修改成员变量的方法

## Get返回值 为成员变量数据类型

## Set返回值 为void

# 重写override

## 子类方法声明应该与父类相同

## 重写父类方法 子类权限应该不小于父类

### 权限由大到小 Public protected default private

其中default是默认权限，不写即为default 写default报错。

## 父类私有方法(private)子类不能重写

# 抽象类

## 抽象方法

### 方法只有声明部分

### 没有方法体（大括号）

### 用abstract修饰

## 抽象类： 包含抽象方法的类，一定是抽象类，使用 abstract 修饰的类

## 抽象方法所在类一定是抽象类：

## 抽象类与抽象方法都必须使用 abstract来修饰

## Abstract不能和 private， final， static一起使用

## 抽象类不能直接实例化对象（不能 new 抽象类）

## 抽象类中可以有抽象方法，也可以不写抽象方法

## 若子类只重写了父类部分抽象方法，子类仍为抽象类

# 继承（extends）

## 获得父类所有的成员：成员变量与方法

## 弊端：耦合性高

## 耦合性：一个地方修改，得修改很多地方

# 接口

## 接口内只有抽象方法

## 方法格式：Public abstract 返回值类型 方法名（参数列表）;

## 接口定义中Public abstract可以不写或选择性书写，默认为全有，实现时必须写出该有的；

## 接口中的方法均为public，接口内不写权限也默认为public

## 接口内成员变量必须定义为常量，只有常量没有变量

## 格式：public static final 数据类型 变量名 = 值；

## Static：可以直接被 类名. 的形式调用（其他为 对象.）

## Final ：变量值不可变，不可重新赋值，即使第二次赋值与第一次一样

## 接口定义中public static final可以不写或选择性书写，默认为全有

# 实现 implements

## 实现类可以为抽象类

## 接口中多实现（public void c implements a,b）

类不可以多继承，因为存在安全隐患，接口可以多实现，无安全隐患

### 可以多实现的原因

接口中全为抽象方法，无方法主体，方法的实现依赖实现类，重写才可执行，接口中的功能都没有方法体，由子类来明确，所以无安全隐患

### 多继承的安全隐患

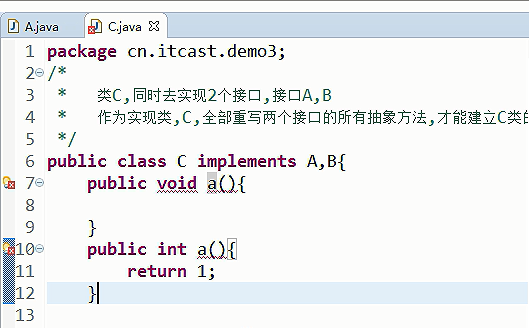
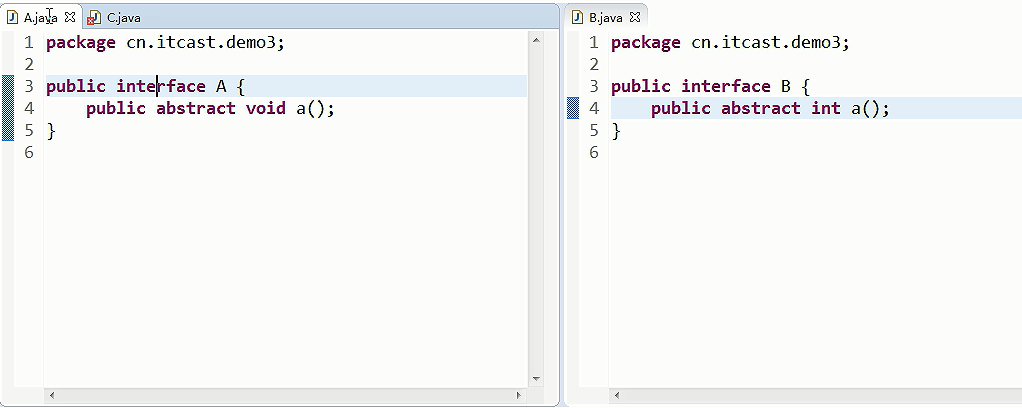
多继承时，当多个父类中有相同功能时，子类调用会产生不确定性。

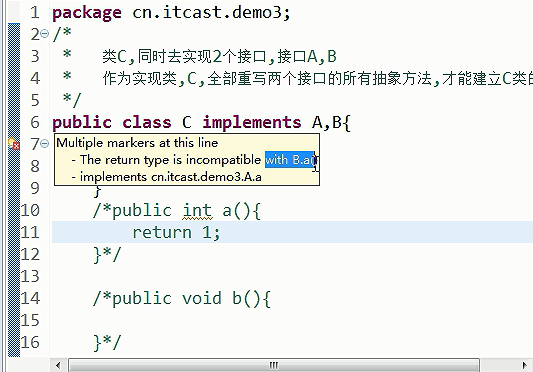
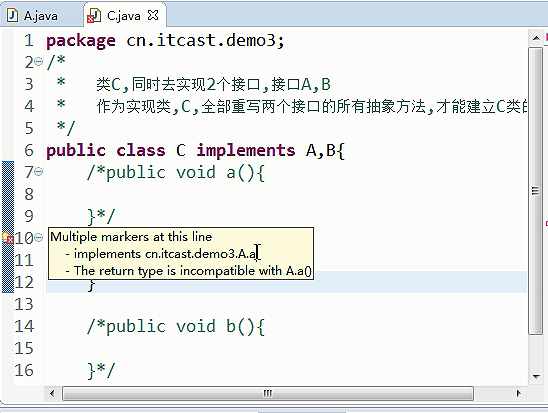
核心原因就是在于多继承父类中功能有主体，而导致调用运行时，不确定运行哪个主体内容。

为什么多实现能解决了呢？

因为接口中的功能都没有方法体，由子类来明确。

### 多实现中两接口内可以有重名方法，但若是两方法返回值不同，子类实现方法会报错（重复定义），两种返回值都实现或只实现一种全都报错



## 继承的同时多实现 （extends b implements c，d（）{}）

## 接口的多继承

### 接口之间有多继承，类没有

## 接口的好处

### **1.接口的出现扩展了功能**。

### **2.接口其实就是暴漏出来的规则**。

### 3.接口的出现降低了耦合性，即设备与设备之间实现了解耦。

# **接口和抽象类的区别**：

## 相同点:

都位于继承的顶端,用于被其他类实现或继承;

都不能直接实例化对象;

都包含抽象方法,其子类都必须覆写这些抽象方法;

## 区别:

抽象类为部分方法提供实现,避免子类重复实现这些方法,提高代码重用性;接口只能包含抽象方法;

一个类只能继承一个直接父类(可能是抽象类),却可以实现多个接口;(接口弥补了Java的单继承)

抽象类是这个事物中应该具备的你内容, 继承体系是一种 is..a关系

接口是这个事物中的额外内容,继承体系是一种 like..a关系

## 二者的选用:

优先选用接口,尽量少用抽象类;

需要定义子类的行为,又要为子类提供共性功能时才选用抽象类;

# 多态

## 需要有继承或实现关系

### 父类 变量名 = new 子类（）

### 成员变量 编译运行全看父类

调用父类的，父类没有该变量会报错

### 成员方法 编译看父类，运行看子类

调用子类实现或重写的方法,父类没有该方法会报错

## Instanceof ：比较对象是否是某引用类型

### 返回类型：boolean

### 格式：Boolean b = 变量名 instanceof 引用类型；

### 注意：需要变量所属类需要和instanceof后的引用类型存在继承或实现关系，否则报错

## 多态中的转型

### 向上转型 父类 变量名 = new 子类（）

#### 优点：可以调用父类子类公共方法

#### 缺点：无法调用子类特有方法（调用父类没有的方法报错）

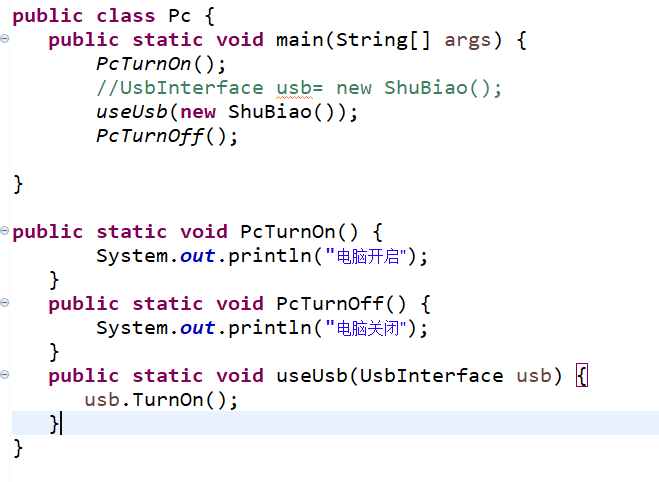
### 向下转型 将向上转型的对象强制类型转换成子类

#### 子类 新变量名 = （子类类型）原变量名

#### 优点：可以调用子类特有方法

#### 注意：应转回原本子类，转成其他类型，编译无错，运行报类型转换错误；

#### 注意：可使用if与instanceof避免出错



# 构造方法（构造器）

## 格式：权限修饰符 方法名（参数列表）{}

### 方法名必须与类名一模一样

### 无返回值，void也无，可以写return （无意义）

### 有方法主体

## 每一次new执行一次，也只执行一次

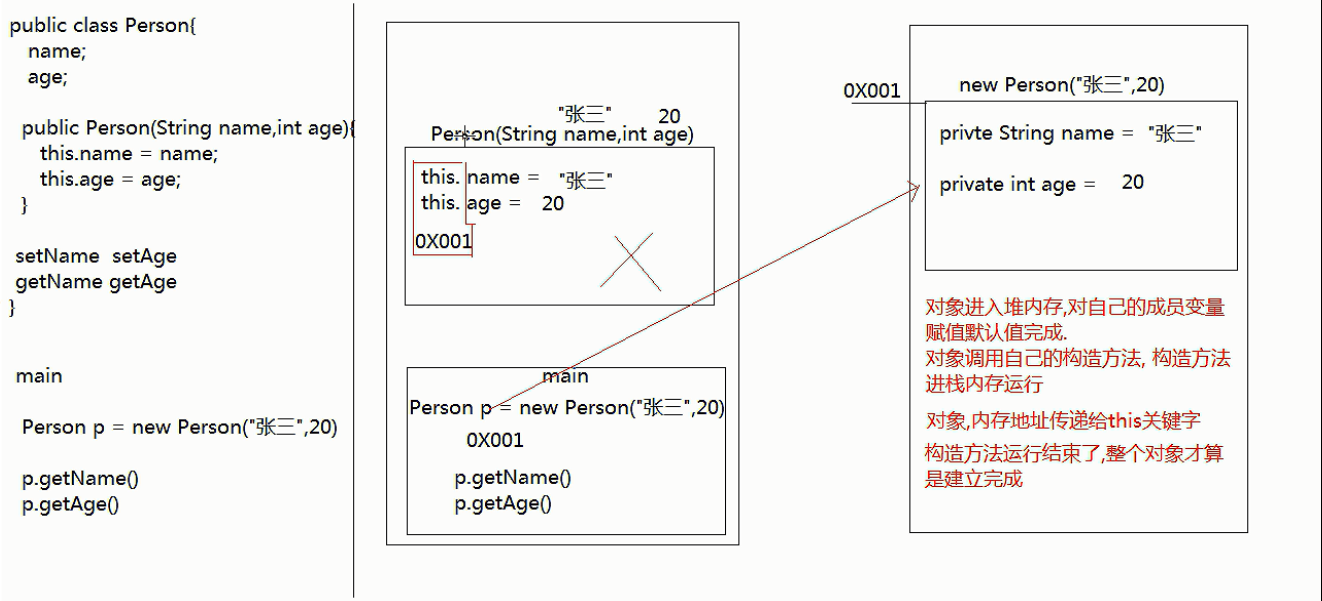
## Class中必有构造方法，不写也有

编译时，javac自动检查有无构造方法

如果没写构造方法，则自动添加

## 写了有参构造方法，new 数据类型（）要加上参数，否则报错

## 构造方法的内存图（在new对象赋默认值之后，把内存地址传给变量名之前调用）



## 构造器的重载

根据参数列表决定调用哪个

## 构造方法可以修改权限修饰符，但权限为private无法被其他类调用

## 构造方法如果赋值了成员变量，也有必要写get和set方法

## This（）与super（）

### 小括号中为空或参数列表

### This（）：用于调用本类的构造方法（根据参数列表）

### super（）：用于调用父类构造方法（根据参数列表）

### This（）与super（）都必须在构造方法第一条

### This（）与super（）两者冲突

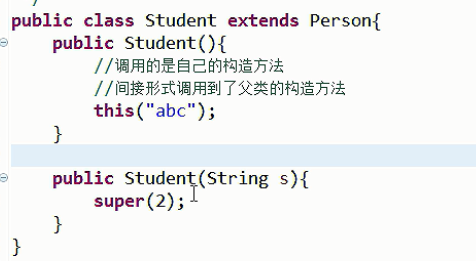
### 子类this()与super（）都不写 默认为super()；

#### 继承关系中先在父类写有参构造方法后子类报错 因为子类构造方法第一句默认的super（）为无参

#### 子类构造器无论重载多少个 都要注意默认super（）为无参的问题；

### 默认为super（）是因为子类构造方法只能选其一 但必须要保证可以调用父类的构造方法，允许间接调用

### 子类构造方法可以间接调用父类构造方法 图



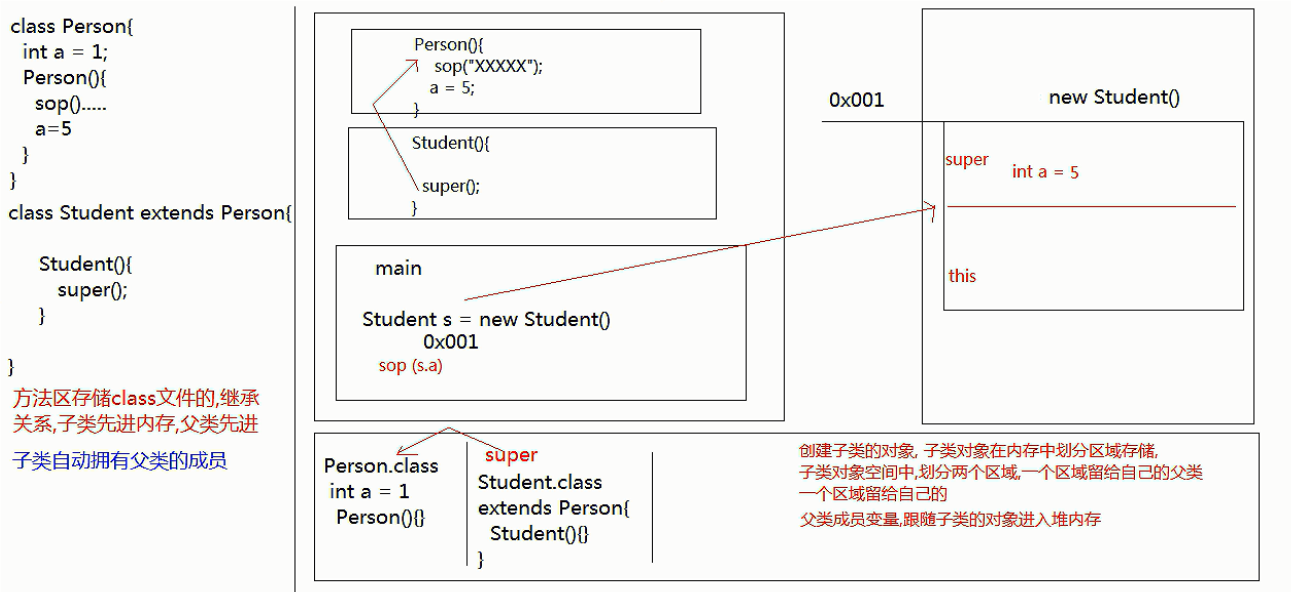
### Super（）用于了解父类成员变量的赋值。

### 父类构造方法中第一行没必要super 因为父类的父类是 object（object类的构造方法为空参）没必要写

### 子类调用父类构建方法的内存图

#### 在方法区中父类class文件先进去，子类后进入并根据super()获取父类地址

#### 创建子类对象 对象空间分为两个区域，一个留给父类，一个留个自己，故父类成员变量跟随子类进入堆内存



### 父类有多个构建方法 子类可以任选其一调用 调用不同的构建方法也可以

# Final 用来修饰类，类的成员，以及局部变量

## final修饰类不可以被继承，但是可以继承其他类

## final修饰的方法不可以被覆盖,但父类中没有被final修饰方法，子类覆盖后可以加final

## final修饰的变量称为常量，这些变量只能赋值一次。

## final修饰的变量，所记录的地址值不能改变，可以更改对象中的属性值

## 修饰成员变量，需要在创建对象前赋值，否则报错。(当没有显式赋值时，多个构造方法的均需要为其赋值。)

### 可以在创建对象时所调用的构造方法中，为变量n赋值

# static

## **被static修饰的成员变量属于类，不属于这个类的某个对象。（多对象共用）**

也就是说，多个对象在访问或修改static修饰的成员变量时，其中一个对象将static成员变量值进行了修改，其他对象中的static成员变量值跟着改变，即多个对象共享同一个static成员变量

## 被static修饰的成员可以并且建议通过类名直接访问

类名.静态成员变量名

类名.静态成员方法名(参数)

对象名.静态成员变量名 ------不建议使用该方式，会出现警告

对象名.静态成员方法名(参数) ------不建议使用该方式，会出现警告

## 静态内容是优先于对象存在，只能访问静态，不能使用this/super。静态修饰的内容存于静态区

## 同一个类中，静态成员只能访问静态成员

## main方法为静态方法仅仅为程序执行入口，它不属于任何一个对象，可以定义在任意类中。

## 静态常量

### public static final修饰

### 变量名用全部大写，多个单词使用下划线连接

开发中，我们想在类中定义一个静态常量，通常使用public static final修饰的变量来完成定义。此时变量名用全部大写，多个单词使用下划线连接

## 接口中的每个成员变量都默认使用public static final修饰。

## 所有接口中的成员变量已是静态常量，由于接口没有构造方法，所以必须显示赋值。可以直接用接口名访问。

# 匿名对象

## 匿名对象在没有指定其引用变量时，只能使用一次

## 匿名对象可以作为方法接收的参数、方法返回值使用

# 内部类

## 定义

什么是内部类

将类写在其他类的内部，可以写在其他类的成员位置和局部位置，这时写在其他类内部的类就称为内部类。其他类也称为外部类

## 内部类的分类

### 成员内部类

### 局部内部类。

## 特点

### 内部类可以直接访问外部类的成员，包含私有的成员

## 成员内部类

成员内部类，定义在外部类中的成员位置。与类中的成员变量相似，可通过外部类对象进行访问

### 定义格式

class 外部类 {

修饰符 class 内部类 {

//其他代码

}

}

### 访问方式

#### 外部类名.内部类名 变量名 = new 外部类名().new 内部类名();

## 局部内部类

局部内部类，定义在外部类方法中的局部位置。与访问方法中的局部变量相似，可通过调用方法进行访问

### **定义格式**

class 外部类 {

修饰符 返回值类型 方法名(参数) {

class 内部类 {

//其他代码

}

}

}

### **访问方式**

在外部类方法中，创建内部类对象，进行访问

### 匿名内部类 局部内部类的一种

定义的匿名内部类有两个含义

 临时定义某一指定类型的子类

 定义后即刻创建刚刚定义的这个子类的对象。

# 访问修饰符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | public | protected | default | private |
| 同一类中 | √ | √ | √ | √ |
| 同一包中(子类与无关类) | √ | √ | √ |  |
| 不同包的子类 | √ | √ |  |  |
| 不同包中的无关类 | √ |  |  |  |

## 要想仅能在本类中访问使用private修饰；

## 要想本包中的类都可以访问不加修饰符即可；

## 要想本包中的类与其他包中的子类可以访问使用protected修饰

## 要想所有包中的所有类都可以访问使用public修饰。

## 注意：如果类用public修饰，则类名必须与文件名相同。一个文件中只能有一个public修饰的类

## 修饰符使用细节

### 常用来修饰类、方法、变量的修饰符

#### public 权限修饰符，公共访问, 类,方法,成员变量

#### protected 权限修饰符，受保护访问, 方法,成员变量

#### 默认什么也不写 也是一种权限修饰符，默认访问, 类,方法,成员变量

#### private 权限修饰符，私有访问, 方法,成员变量

#### static 静态修饰符 方法,成员变量

#### final 最终修饰符 类,方法,成员变量,局部变量

#### abstract 抽象修饰符 类 ,方法

### **修饰类能够使用的修饰符：**

#### Public

#### 默认的

#### final

#### abstract关键字

### **修饰成员变量能够使用的修饰符**

#### public : 公共的

#### protected : 受保护的

#### ：默认的

#### private ：私有的

#### final : 最终的

#### tatic : 静态的

### **修饰构造方法能够使用的修饰符：**

#### public : 公共的

#### protected : 受保护的

#### : 默认的

#### private ：私有的

### 修饰成员方法能够使用的修饰符：

#### public : 公共的

#### protected : 受保护的

#### : 默认的

#### private ：私有的

#### final : 最终的

#### static : 静态的

#### abstract : 抽象的

# 代码块

## 局部代码块

局部代码块是定义在方法或语句中

## 构造代码块

构造代码块是定义在类中成员位置的代码块

### 特点：

#### 优先于构造方法执行，构造代码块用于执行所有对象均需要的初始化动作

#### 每创建一个对象均会执行一次构造代码块。

## 静态代码块

静态代码块是定义在成员位置，使用static修饰的代码块。

### 特点：

#### 它优先于主方法执行、优先于构造代码块执行，当以任意形式第一次使用到该类时执行。

#### 该类不管创建多少对象，静态代码块只执行一次。

#### 可用于给静态变量赋值，用来给类进行初始化。

## 总结

### 局部代码块：定义在方法中的，用来限制变量的作用范围

### 构造代码块：定义在类中方法外，用来给对象中的成员初始化赋值

### 静态代码块：定义在类中方法外，用来给类的静态成员初始化赋值

# **自定义数据类型的使用**

## 类作为方法参数时，说明要向方法中传入该类的对象

## 类作为方法返回值时，说明该方法要返回一个该类的对象。

## 抽象类作为方法参数时，说明要传入一个实现抽象类所有抽象方法的子类对象。

## 抽象类作为方法返回值时，说明需要返回一个实现抽象类所有抽象方法的子类对象。

## 接口作为方法参数时，说明该方法要传入一个接口实现类对象。

## 接口作为方法返回值时，说明该方法需要返回一个接口实现类对象。