笔记

[笔记 1](#_Toc26545633)

[JVM的划分 17](#_Toc26545634)

[Return后只能写一个值 一个方法可以有多个return 17](#_Toc26545635)

[输出语句中写对象 默认调用tostring方法 17](#_Toc26545636)

[Void: 无返回值 17](#_Toc26545637)

[抽象方法没有方法主体（大括号） 17](#_Toc26545638)

[This 谁调用就指谁 17](#_Toc26545639)

[导包可以使用通配符把该目录下的类全部导入 17](#_Toc26545640)

[实参与形参 17](#_Toc26545641)

[方法参数是基本数据类型 17](#_Toc26545642)

[方法参数是引用数据类型 18](#_Toc26545643)

[方法的格式 19](#_Toc26545644)

[修饰符 返回值类型 方法名（参数列表 /\*逗号隔开\*/）{} 19](#_Toc26545645)

[重载（overload） 19](#_Toc26545646)

[方法重载注意事项 19](#_Toc26545647)

[a: 参数列表必须不同 19](#_Toc26545648)

[b: 重载和参数变量名无关 19](#_Toc26545649)

[c: 重载和返回值类型无关 19](#_Toc26545650)

[d: 重载和修饰符无关 19](#_Toc26545651)

[构造器也有重载 19](#_Toc26545652)

[重载的几种测试 19](#_Toc26545653)

[引用数据类型（类，数组，接口） 22](#_Toc26545654)

[创建引用数据类型的格式 数据类型 变量名 = new 数据类型（）； 23](#_Toc26545655)

[New 实例化对象 23](#_Toc26545656)

[参数列表为引用数据类型时 23](#_Toc26545657)

[形参内往往为接口，配合下面的格式，实现向上转型 23](#_Toc26545658)

[形参里的引用类型 引用名 = 实参（）；（一般为向上转型） 23](#_Toc26545659)

[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 23](#_Toc26545660)

[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 23](#_Toc26545661)

[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 23](#_Toc26545662)

[变量名保存的是一个内存地址 23](#_Toc26545663)

[自定义类 23](#_Toc26545664)

[定义类中属性时修饰符可以不写 24](#_Toc26545665)

[Cmd中的运行谁编译谁 24](#_Toc26545666)

[一个引用变量的内存图 24](#_Toc26545667)

[两个引用类型变量内存图 24](#_Toc26545668)

[ArrayList 25](#_Toc26545669)

[格式 25](#_Toc26545670)

[ArrayList中的方法 25](#_Toc26545671)

[注意事项 26](#_Toc26545672)

[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 26](#_Toc26545673)

[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。 26](#_Toc26545674)

[百位十位个位的数如何求 26](#_Toc26545675)

[百位 i/100 26](#_Toc26545676)

[十位 i/10%10 26](#_Toc26545677)

[个位 i%10 26](#_Toc26545678)

[Char 与 int 26](#_Toc26545679)

[Char两字节 int四字节， 26](#_Toc26545680)

[两者计算 char自动转换为int型（根据ascii表） 26](#_Toc26545681)

[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 26](#_Toc26545682)

[Char中保存汉字 查询unicode （万国表） 26](#_Toc26545683)

[Char转为int类型取值0-65535 26](#_Toc26545684)

[Char和short均为两字节取值不同的原因 26](#_Toc26545685)

[冒泡排序问题汇总 27](#_Toc26545686)

[1.注意存在j+1，注意溢出问题 27](#_Toc26545687)

[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 28](#_Toc26545688)

[面向对象的特性 继承 封装 多态 28](#_Toc26545689)

[成员变量和局部变量区别 28](#_Toc26545690)

[定义的位置不同 28](#_Toc26545691)

[在内存中的位置不同 28](#_Toc26545692)

[生命周期不同 28](#_Toc26545693)

[内存位置不同 29](#_Toc26545694)

[默认值不同 29](#_Toc26545695)

[局部变量生命周期 29](#_Toc26545696)

[Private 只修饰成员变量，不修饰局部变量； 29](#_Toc26545697)

[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 29](#_Toc26545698)

[方法“就近” 方法内有不使用则本类成员变量 29](#_Toc26545699)

[Super调用父类 29](#_Toc26545700)

[This 调用自己本类成员 谁调用就指谁 29](#_Toc26545701)

[Get和set方法 30](#_Toc26545702)

[Get返回值 为成员变量数据类型 30](#_Toc26545703)

[Set返回值 为void 30](#_Toc26545704)

[重写override 30](#_Toc26545705)

[子类方法声明应该与父类相同 30](#_Toc26545706)

[重写父类方法 子类权限应该不小于父类 30](#_Toc26545707)

[权限由大到小 Public protected default private 30](#_Toc26545708)

[父类私有方法(private)子类不能重写 30](#_Toc26545709)

[抽象类 31](#_Toc26545710)

[抽象方法 31](#_Toc26545711)

[方法只有声明部分 31](#_Toc26545712)

[没有方法体（大括号） 31](#_Toc26545713)

[用abstract修饰 31](#_Toc26545714)

[抽象类： 包含抽象方法的类，一定是抽象类，使用 abstract 修饰的类 31](#_Toc26545715)

[抽象方法所在类一定是抽象类： 31](#_Toc26545716)

[抽象类与抽象方法都必须使用 abstract来修饰 31](#_Toc26545717)

[Abstract不能和 private， final， static一起使用 31](#_Toc26545718)

[抽象类不能直接实例化对象（不能 new 抽象类） 31](#_Toc26545719)

[抽象类中可以有抽象方法，也可以不写抽象方法 31](#_Toc26545720)

[若子类只重写了父类部分抽象方法，子类仍为抽象类 31](#_Toc26545721)

[继承（extends） 31](#_Toc26545722)

[获得父类所有的成员：成员变量与方法 31](#_Toc26545723)

[弊端：耦合性高 32](#_Toc26545724)

[耦合性：一个地方修改，得修改很多地方 32](#_Toc26545725)

[接口 32](#_Toc26545726)

[接口内只有抽象方法 32](#_Toc26545727)

[方法格式：Public abstract 返回值类型 方法名（参数列表）; 32](#_Toc26545728)

[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； 32](#_Toc26545729)

[接口中的方法均为public，接口内不写权限也默认为public 32](#_Toc26545730)

[接口内成员变量必须定义为常量，只有常量没有变量 32](#_Toc26545731)

[格式：public static final 数据类型 变量名 = 值； 32](#_Toc26545732)

[Static：可以直接被 类名. 的形式调用（其他为 对象.） 32](#_Toc26545733)

[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 32](#_Toc26545734)

[接口定义中public static final可以不写或选择性书写，默认为全有 32](#_Toc26545735)

[实现 implements 33](#_Toc26545736)

[实现类可以为抽象类 33](#_Toc26545737)

[接口中多实现（public void c implements a,b） 33](#_Toc26545738)

[可以多实现的原因 33](#_Toc26545739)

[多继承的安全隐患 33](#_Toc26545740)

[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 33](#_Toc26545741)

[继承的同时多实现 （extends b implements c，d（）{}） 35](#_Toc26545742)

[接口的多继承 35](#_Toc26545743)

[接口之间有多继承，类没有 35](#_Toc26545744)

[接口的好处 35](#_Toc26545745)

[1.接口的出现扩展了功能。 35](#_Toc26545746)

[2.接口其实就是暴漏出来的规则。 35](#_Toc26545747)

[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。 35](#_Toc26545748)

[接口和抽象类的区别： 35](#_Toc26545749)

[相同点: 35](#_Toc26545750)

[区别: 35](#_Toc26545751)

[二者的选用: 36](#_Toc26545752)

[多态 37](#_Toc26545753)

[需要有继承或实现关系 37](#_Toc26545754)

[父类 变量名 = new 子类（） 37](#_Toc26545755)

[成员变量 编译运行全看父类 37](#_Toc26545756)

[调用父类的，父类没有该变量会报错 37](#_Toc26545757)

[成员方法 编译看父类，运行看子类 37](#_Toc26545758)

[调用子类实现或重写的方法,父类没有该方法会报错 37](#_Toc26545759)

[静态方法运行父类，非静态运行子类 37](#_Toc26545760)

[Instanceof ：比较对象是否是某引用类型 37](#_Toc26545761)

[返回类型：boolean 37](#_Toc26545762)

[格式：Boolean b = 变量名 instanceof 引用类型； 37](#_Toc26545763)

[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 37](#_Toc26545764)

[多态中的转型 37](#_Toc26545765)

[向上转型 父类 变量名 = new 子类（） 37](#_Toc26545766)

[优点：可以调用父类子类公共方法 37](#_Toc26545767)

[缺点：无法调用子类特有方法（调用父类没有的方法报错） 38](#_Toc26545768)

[向下转型 将向上转型的对象强制类型转换成子类 38](#_Toc26545769)

[子类 新变量名 = （子类类型）原变量名 38](#_Toc26545770)

[优点：可以调用子类特有方法 38](#_Toc26545771)

[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； 38](#_Toc26545772)

[注意：可使用if与instanceof避免出错 38](#_Toc26545773)

[构造方法（构造器） 39](#_Toc26545774)

[格式：权限修饰符 方法名（参数列表）{} 39](#_Toc26545775)

[方法名必须与类名一模一样 39](#_Toc26545776)

[无返回值，void也无，可以写return （无意义） 39](#_Toc26545777)

[有方法主体 39](#_Toc26545778)

[每一次new执行一次，也只执行一次 39](#_Toc26545779)

[Class中必有构造方法，不写也有 39](#_Toc26545780)

[写了有参构造方法，new 数据类型（）要加上参数，否则报错 39](#_Toc26545781)

[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） 39](#_Toc26545782)

[构造器的重载 40](#_Toc26545783)

[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 40](#_Toc26545784)

[构造方法如果赋值了成员变量，也有必要写get和set方法 40](#_Toc26545785)

[This（）与super（） 41](#_Toc26545786)

[小括号中为空或参数列表 41](#_Toc26545787)

[This（）：用于调用本类的构造方法（根据参数列表） 41](#_Toc26545788)

[super（）：用于调用父类构造方法（根据参数列表） 41](#_Toc26545789)

[This（）与super（）都必须在构造方法第一条 41](#_Toc26545790)

[This（）与super（）两者冲突 41](#_Toc26545791)

[子类this()与super（）都不写 默认为super()； 41](#_Toc26545792)

[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 41](#_Toc26545793)

[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； 41](#_Toc26545794)

[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 41](#_Toc26545795)

[子类构造方法可以间接调用父类构造方法 图 41](#_Toc26545796)

[Super（）用于了解父类成员变量的赋值。 43](#_Toc26545797)

[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 43](#_Toc26545798)

[子类调用父类构建方法的内存图 43](#_Toc26545799)

[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 43](#_Toc26545800)

[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 43](#_Toc26545801)

[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 44](#_Toc26545802)

[Final 用来修饰类，类的成员，以及局部变量 44](#_Toc26545803)

[final修饰类不可以被继承，可被调用和实例化，但是可以继承其他类， 44](#_Toc26545804)

[final不能修饰构造方法 44](#_Toc26545805)

[string类也是final类 44](#_Toc26545806)

[final修饰的方法不可以被覆盖重写,可被调用，但父类中没有被final修饰方法，子类覆盖后可以加final 44](#_Toc26545807)

[final修饰的变量称为常量，这些变量只能赋值一次。 44](#_Toc26545808)

[final修饰的引用类型变量，所记录的地址值不能改变，可以更改对象中的属性值 44](#_Toc26545809)

[即不能用该变量名保存新实例化对象，也不能赋值后再写 变量名 =null； 44](#_Toc26545810)

[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。) 44](#_Toc26545811)

[可以在创建对象时所调用的构造方法中，为变量n赋值 44](#_Toc26545812)

[不能用set方法赋值，因为set方法在创建对象之后 45](#_Toc26545813)

[final固定的不是默认值，须手动赋值 45](#_Toc26545814)

[static 45](#_Toc26545815)

[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） 45](#_Toc26545816)

[被static修饰的成员可以并且建议通过类名直接访问 46](#_Toc26545817)

[类名.静态成员变量名 46](#_Toc26545818)

[类名.静态成员方法名(参数) 46](#_Toc26545819)

[对象名.静态成员变量名 ------不建议使用该方式，会出现警告 46](#_Toc26545820)

[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 46](#_Toc26545821)

[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 46](#_Toc26545822)

[静态优先于非静态存在于内存，所以无法调用非静态 46](#_Toc26545823)

[Static内存图 46](#_Toc26545824)

[同一个类中，静态成员只能访问静态成员 47](#_Toc26545825)

[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。 47](#_Toc26545826)

[方法加不加静态看是否调用了静态成员 47](#_Toc26545827)

[静态常量 public static final修饰 47](#_Toc26545828)

[变量名用全部大写，多个单词使用下划线连接 47](#_Toc26545829)

[接口中的每个成员变量都默认使用public static final修饰。 47](#_Toc26545830)

[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。 47](#_Toc26545831)

[匿名对象 47](#_Toc26545832)

[匿名对象在没有指定其引用变量时，只能使用一次 47](#_Toc26545833)

[匿名对象可以作为方法接收的参数、方法返回值使用 47](#_Toc26545834)

[内部类 48](#_Toc26545835)

[定义 48](#_Toc26545836)

[内部类的分类 48](#_Toc26545837)

[成员内部类 48](#_Toc26545838)

[局部内部类。 48](#_Toc26545839)

[特点 48](#_Toc26545840)

[内部类可以直接访问外部类的成员，包含私有的成员 48](#_Toc26545841)

[外部类要使用内部类成员，必须建立内部类对象 48](#_Toc26545842)

[成员内部类 48](#_Toc26545843)

[可以使用成员修饰符 49](#_Toc26545844)

[可以继承，可以实现 49](#_Toc26545845)

[定义格式 49](#_Toc26545846)

[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(); 49](#_Toc26545847)

[成员内部类的同名调用 49](#_Toc26545848)

[输出I 值为3 （成员内部类的局部变量） 50](#_Toc26545849)

[输出this.i 值为2（成员内部类的成员变量） 50](#_Toc26545850)

[输出outer.this.i 值为1（外部类的成员变量） 50](#_Toc26545851)

[局部内部类 50](#_Toc26545852)

[定义格式 50](#_Toc26545853)

[访问方式 图 50](#_Toc26545854)

[匿名内部类 局部内部类的一种 50](#_Toc26545855)

[简化 定义实现类 重写方法 建立实现类对象 合并 51](#_Toc26545856)

[New 接口或父类（）{重写抽象方法}.方法； 分号结束 51](#_Toc26545857)

[以上代码相当于父类或接口的对象，可用多态形式的变量名保存 51](#_Toc26545858)

[以上代码不能同时调用两个方法，但是可保存到变量名后用变量名多个调用 51](#_Toc26545859)

[与实现类在调用方法等价 51](#_Toc26545860)

[包名全小写 51](#_Toc26545861)

[导包要导入到最低层文件名，通配符也一样 51](#_Toc26545862)

[访问修饰符 51](#_Toc26545863)

[要想仅能在本类中访问使用private修饰； 52](#_Toc26545864)

[要想本包中的类都可以访问不加修饰符即可； 52](#_Toc26545865)

[要想本包中的类与其他包中的子类可以访问使用protected修饰 52](#_Toc26545866)

[要想所有包中的所有类都可以访问使用public修饰。 52](#_Toc26545867)

[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 52](#_Toc26545868)

[修饰符使用细节 52](#_Toc26545869)

[常用来修饰类、方法、变量的修饰符 52](#_Toc26545870)

[public 权限修饰符，公共访问, 类,方法,成员变量 52](#_Toc26545871)

[protected 权限修饰符，受保护访问, 方法,成员变量 52](#_Toc26545872)

[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 52](#_Toc26545873)

[private 权限修饰符，私有访问, 方法,成员变量 52](#_Toc26545874)

[static 静态修饰符 方法,成员变量 52](#_Toc26545875)

[final 最终修饰符 类,方法,成员变量,局部变量 52](#_Toc26545876)

[abstract 抽象修饰符 类 ,方法 52](#_Toc26545877)

[修饰类能够使用的修饰符： 52](#_Toc26545878)

[Public 53](#_Toc26545879)

[默认的 53](#_Toc26545880)

[final 53](#_Toc26545881)

[abstract关键字 53](#_Toc26545882)

[修饰成员变量能够使用的修饰符 53](#_Toc26545883)

[public : 公共的 53](#_Toc26545884)

[protected : 受保护的 53](#_Toc26545885)

[：默认的 53](#_Toc26545886)

[private ：私有的 53](#_Toc26545887)

[final : 最终的 53](#_Toc26545888)

[tatic : 静态的 53](#_Toc26545889)

[修饰构造方法能够使用的修饰符： 53](#_Toc26545890)

[public : 公共的 53](#_Toc26545891)

[protected : 受保护的 53](#_Toc26545892)

[: 默认的 53](#_Toc26545893)

[private ：私有的 54](#_Toc26545894)

[修饰成员方法能够使用的修饰符： 54](#_Toc26545895)

[public : 公共的 54](#_Toc26545896)

[protected : 受保护的 54](#_Toc26545897)

[: 默认的 54](#_Toc26545898)

[private ：私有的 54](#_Toc26545899)

[final : 最终的 54](#_Toc26545900)

[static : 静态的 54](#_Toc26545901)

[abstract : 抽象的 54](#_Toc26545902)

[代码块 55](#_Toc26545903)

[局部代码块 55](#_Toc26545904)

[局部代码块是定义在方法或语句中 55](#_Toc26545905)

[构造代码块 55](#_Toc26545906)

[构造代码块是定义在类中成员位置的代码块 55](#_Toc26545907)

[特点： 55](#_Toc26545908)

[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，运行完构造代码块再运行构造方法 55](#_Toc26545909)

[每创建一个对象均会执行一次构造代码块。 55](#_Toc26545910)

[静态代码块 static{}； 55](#_Toc26545911)

[静态代码块是定义在成员位置，使用static修饰的代码块。 55](#_Toc26545912)

[特点： 55](#_Toc26545913)

[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。 55](#_Toc26545914)

[该类不管创建多少对象，静态代码块只执行一次。 55](#_Toc26545915)

[可用于给静态变量赋值，用来给类进行初始化。 55](#_Toc26545916)

[总结 56](#_Toc26545917)

[局部代码块：定义在方法中的，用来限制变量的作用范围 56](#_Toc26545918)

[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 56](#_Toc26545919)

[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 56](#_Toc26545920)

[静态代码块 >构造代码块>构造方法 56](#_Toc26545921)

[Eclipse导出文档注释 导出-java-javadoc 56](#_Toc26545922)

[Eclipse导出Jar包 导出-java-jar 56](#_Toc26545923)

[自定义数据类型的使用 57](#_Toc26545924)

[类作为方法参数时，说明要向方法中传入该类的对象 57](#_Toc26545925)

[类作为方法返回值时，说明该方法要返回一个该类的对象。 57](#_Toc26545926)

[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。（多态） 57](#_Toc26545927)

[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。（多态） 57](#_Toc26545928)

[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。（多态） 57](#_Toc26545929)

[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。（多态） 57](#_Toc26545930)

[调用方法接受返回值时，要注意方法的返回值类型，不能用子类类型接受父类，即使该父类向上 57](#_Toc26545931)

[Java的API 应用程序接口 jdk提供的类 57](#_Toc26545932)

[Application(应用) Programming(程序) Interface(接口) 57](#_Toc26545933)

[Object类概述 57](#_Toc26545934)

[接口不继承object类 58](#_Toc26545935)

[equals方法 58](#_Toc26545936)

[重写注意传入参数自动转为object形式 要向下转型 58](#_Toc26545937)

[重写案例 58](#_Toc26545938)

[toString方法 58](#_Toc26545939)

[重写案例 58](#_Toc26545940)

[String类 免new 59](#_Toc26545941)

[String创建 59](#_Toc26545942)

[双引号形式创建 59](#_Toc26545943)

[New形式创建 59](#_Toc26545944)

[区别 59](#_Toc26545945)

[双引号形式创建 59](#_Toc26545946)

[New形式创建 59](#_Toc26545947)

[String是一个常量 一旦创建不能改变 字符串““本身是一个对象 60](#_Toc26545948)

[String本质是一个一维char数组 60](#_Toc26545949)

[Str1==str2 与str1.equals(str2)结果不同原因 60](#_Toc26545950)

[String（byte【】 bytes）解码平台字符集的实质 61](#_Toc26545951)

[限定解码首位索引与解码个数 62](#_Toc26545952)

[String（char【】 value） 字符数组转为字符串 不查询编码表 63](#_Toc26545953)

[限定首位索引与个数 63](#_Toc26545954)

[Substring 返回新字符串 原字符串是常量不可以改变 故需要新字符串接受才可以正常显示 64](#_Toc26545955)

[substring索引包头不包尾 64](#_Toc26545956)

[starswith与endswith 与contains返回值布尔 64](#_Toc26545957)

[indexof（char ch）查字符第一次出现的索引 无返回-1； 64](#_Toc26545958)

[getBytes（）；按表得到一维byte数组 64](#_Toc26545959)

[toCharArray（）返回字符数组 64](#_Toc26545960)

[toUpperCase（），toLowerCase（）； 64](#_Toc26545961)

[方法 64](#_Toc26545962)

[StringBuffer 字符串缓冲区 66](#_Toc26545963)

[字符数组无final 66](#_Toc26545964)

[初始16字符 66](#_Toc26545965)

[可变长度的数组 66](#_Toc26545966)

[StringBuilder: 字符串缓冲区 66](#_Toc26545967)

[兼容StringBuffer 方法一样 66](#_Toc26545968)

[StringBuffer/StringBuilder:方法 66](#_Toc26545969)

[public StringBuffer append(String str) 在原有字符串缓冲区内容基础上，在末尾追加新数据 66](#_Toc26545970)

[正则表达式 67](#_Toc26545971)

[Regular Expression，在代码中常简写为regex 67](#_Toc26545972)

[字符串类中涉及正则表达式的常用方法 69](#_Toc26545973)

[public boolean matches(String regex) //判断字符串是否匹配给定的规则 70](#_Toc26545974)

[public String[] split(String regex) //根据给定正则表达式的匹配规则，拆分此字符串 70](#_Toc26545975)

[public String replaceAll(String regex,String replacement) //将符合规则的字符串内容，全部替换为新字符串 70](#_Toc26545976)

[匹配正确的数字 70](#_Toc26545977)

[匹配正整数：”\\d+” 70](#_Toc26545978)

[匹配正小数：”\\d+\\.\\d+” 70](#_Toc26545979)

[匹配负整数：”-\\d+” 70](#_Toc26545980)

[匹配负小数：”-\\d+\\.\\d+” 70](#_Toc26545981)

[匹配保留两位小数的正数：”\\d+\\.\\d{2}” 70](#_Toc26545982)

[匹配保留1-3位小数的正数：\\d+\\.\\d{1,3} 70](#_Toc26545983)

[匹配合法的邮箱 70](#_Toc26545984)

[Date 精确到毫秒 71](#_Toc26545985)

[DateFormat:是日期/时间格式化子类的抽象类, 使用其子类SimpleDateFormat 实例化构造方法： 72](#_Toc26545986)

[public SimpleDateFormat() 默认的格式化操作 72](#_Toc26545987)

[public SimpleDateFormat(String pattern) 按照指定的格式，进行日期格式化 72](#_Toc26545988)

[日期和时间模式 72](#_Toc26545989)

[y 年 72](#_Toc26545990)

[M 年中的月份 72](#_Toc26545991)

[d 月份中的天数 72](#_Toc26545992)

[H 一天中的小时数（0-23） 72](#_Toc26545993)

[m 小时中的分钟数 72](#_Toc26545994)

[s 分钟中的秒数 72](#_Toc26545995)

[S 毫秒数 72](#_Toc26545996)

[方法： 72](#_Toc26545997)

[public final String format(Date date) 把日期 格式化成字符串 72](#_Toc26545998)

[public Date parse(String source) 把日期字符串 转换成 日期对象 73](#_Toc26545999)

[Calendar:日历类，可获取日期中指定字段的值 73](#_Toc26546000)

[public static Calendar getInstance() //获取日期对象 73](#_Toc26546001)

[public int get(int field) //获取时间字段值 73](#_Toc26546002)

[public void add(int field,int amount) //指定字段增加某值 73](#_Toc26546003)

[public final void set(int field,int value)//设置指定字段的值 73](#_Toc26546004)

[public final Date getTime() //获取该日历对象转成的日期对象 73](#_Toc26546005)

[基本类型包装类 73](#_Toc26546006)

[8种基本类型对应的包装类 73](#_Toc26546007)

[基本类型 包装类 73](#_Toc26546008)

[byte Byte 73](#_Toc26546009)

[short Short 73](#_Toc26546010)

[int Integer 73](#_Toc26546011)

[log Long 73](#_Toc26546012)

[float Float 74](#_Toc26546013)

[double Double 74](#_Toc26546014)

[char Character 74](#_Toc26546015)

[boolean Boolean 74](#_Toc26546016)

[自动装箱、自动拆箱 74](#_Toc26546017)

[自动装箱：基本数值转成对象（int  Integer） 74](#_Toc26546018)

[自动拆箱：对象转成基本数值（Integer  int） 74](#_Toc26546019)

[常用方法 74](#_Toc26546020)

[public int parseInt(String str):把字符串转成基本类型int 74](#_Toc26546021)

[public static String toString(int x):把基本类型int转成字符串 74](#_Toc26546022)

[public static Integer valueOf(int x):把基本类型i字符串转成Integer对象 74](#_Toc26546023)

[public int intValue():以 int类型返回该包装类对象的值 74](#_Toc26546024)

[System类: 系统属性信息工具类 75](#_Toc26546025)

[public static long currentTimeMillis()：获取当前系统时间与1970年01月01日00:00点之间的毫秒差值 75](#_Toc26546026)

[public static void exit(int status)：用来结束正在运行的Java程序。参数传入一个数字即可。通常传入0记为正常状态，其他为异常状态 75](#_Toc26546027)

[public static void gc()：用来运行JVM中的垃圾回收器，完成内存中垃圾的清除。 75](#_Toc26546028)

[public static String getProperties()：用来获取指系统属性信息 75](#_Toc26546029)

[Arrays类：数组操作工具类 76](#_Toc26546030)

[public static void sort方法，用来对指定数组中的元素进行排序（元素值从小到大进行排序） 76](#_Toc26546031)

[public static String toString方法，用来返回指定数组元素内容的字符串形式 76](#_Toc26546032)

[public static void binarySearch方法，在指定数组中，查找给定元素值出现的位置。若没有查询到，返回位置为-插入点-1。要求该数组必须是个有序的数组 76](#_Toc26546033)

[Math类：数学运算工具类 77](#_Toc26546034)

[abs方法,结果都为正数 77](#_Toc26546035)

[ceil方法，结果为比参数值大的最小整数的double值 77](#_Toc26546036)

[floor方法，结果为比参数值小的最大整数的double值 77](#_Toc26546037)

[max方法，返回两个参数值中较大的值 77](#_Toc26546038)

[min方法，返回两个参数值中较小的值 77](#_Toc26546039)

[pow方法，返回第一个参数的第二个参数次幂的值 77](#_Toc26546040)

[round方法，返回参数值四舍五入的结果 77](#_Toc26546041)

[random方法，产生一个大于等于0.0且小于1.0的double小数 77](#_Toc26546042)

[throwable 所有异常错误的超类 77](#_Toc26546043)

[子类 77](#_Toc26546044)

[error错误 必须修改程序 编译期无 运行期出现 77](#_Toc26546045)

[exception异常 修改异常仍可运行 编译异常 运行异常 77](#_Toc26546046)

[jvm建立异常对象 抛出给调用者 main方法抛出会抛给虚拟机 77](#_Toc26546047)

[一旦异常被抛出 之后程序不再运行 77](#_Toc26546048)

[Throw 在方法内抛出异常 78](#_Toc26546049)

[Throw new 对象 （异常或其子类） 78](#_Toc26546050)

[Throws 在方法声明上 表明方法可能有问题 后加异常类名 78](#_Toc26546051)

[Try{} catch（异常类型 变量名）{处理异常代码} finally{必须执行} 78](#_Toc26546052)

[Finally代码块内代码必须执行 除非结束虚拟机 78](#_Toc26546053)

[多catch处理细节 78](#_Toc26546054)

[平级异常无顺序 有继承关系的异常类型 越高级越在下面 78](#_Toc26546055)

[原因 多态 高级写上面无论抛出哪种 catch里的高级异常都能接受 78](#_Toc26546056)

[运行异常 78](#_Toc26546057)

[声明中无throws 78](#_Toc26546058)

[因为一般不会发生 一旦发生应该修改源代码 78](#_Toc26546059)

[用于出现不合理的条件停止程序提示用户 78](#_Toc26546060)

[方法重写时的异常 78](#_Toc26546061)

[父类抛出异常 78](#_Toc26546062)

[子类可以抛出可以不抛出 78](#_Toc26546063)

[若抛出 父类异常应该大于子类异常 79](#_Toc26546064)

[父类不抛 子类不能抛出 79](#_Toc26546065)

[此时子类重写方法调用了异常方法 只能trycatch 79](#_Toc26546066)

[自定义异常需要继承exception或者runtimeexception 否则不可抛 79](#_Toc26546067)

# JVM的划分

# Return后只能写一个值 一个方法可以有多个return

# 输出语句中写对象名 默认调用tostring方法 显示内存地址

# 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）

# This 谁调用就指谁

# 导包可以使用通配符把该目录下的类全部导入

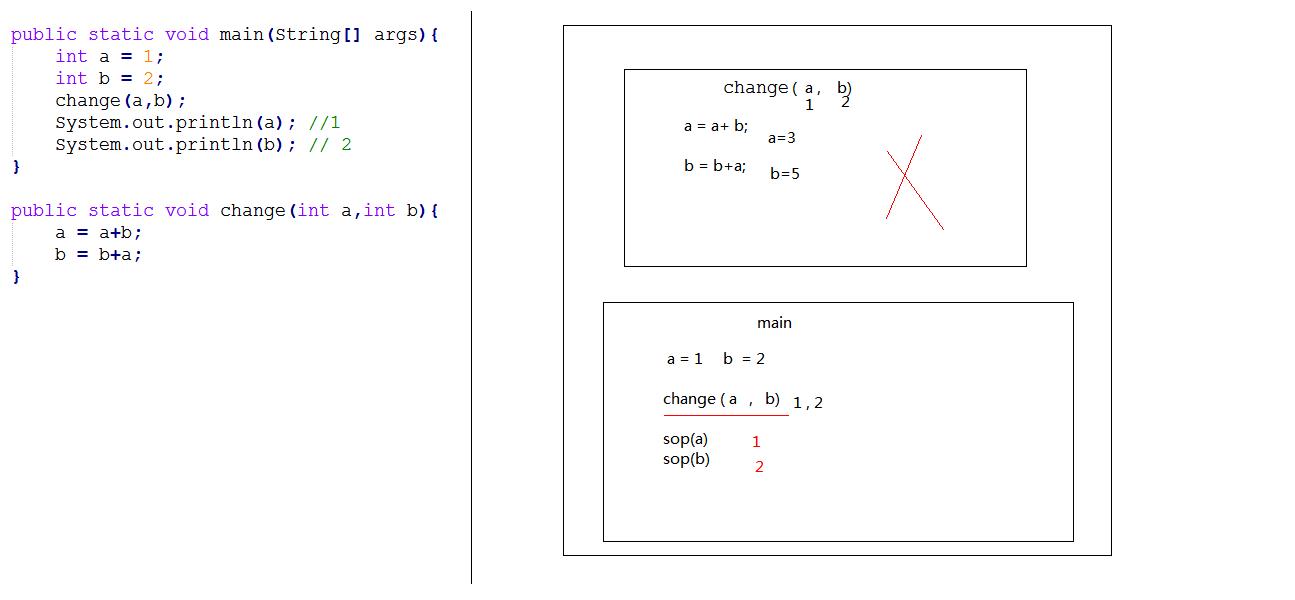
## 如java.util.\*

# 实参与形参

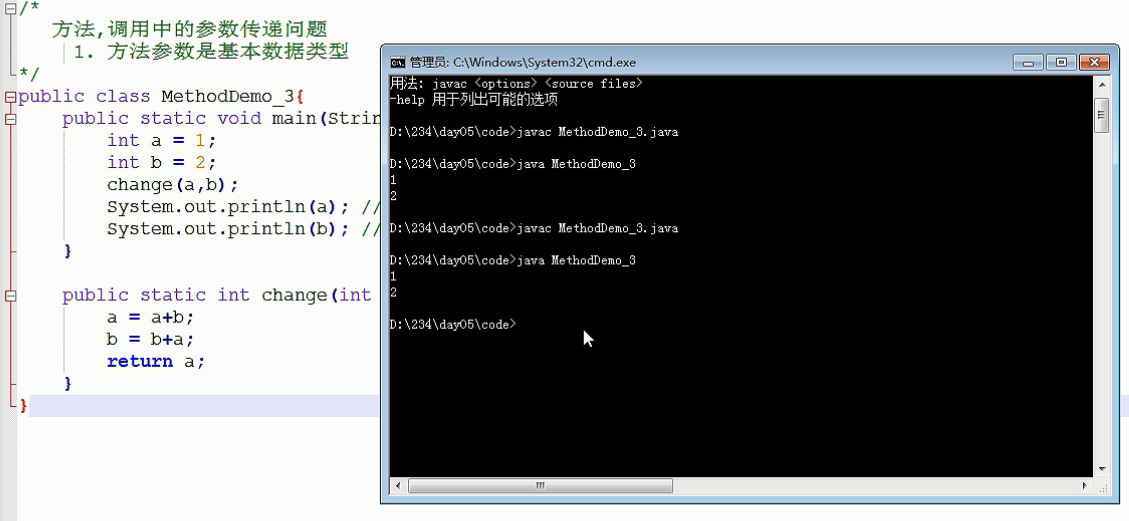
## 方法参数是基本数据类型

### 方法参数是基本类型时，传递的是值。

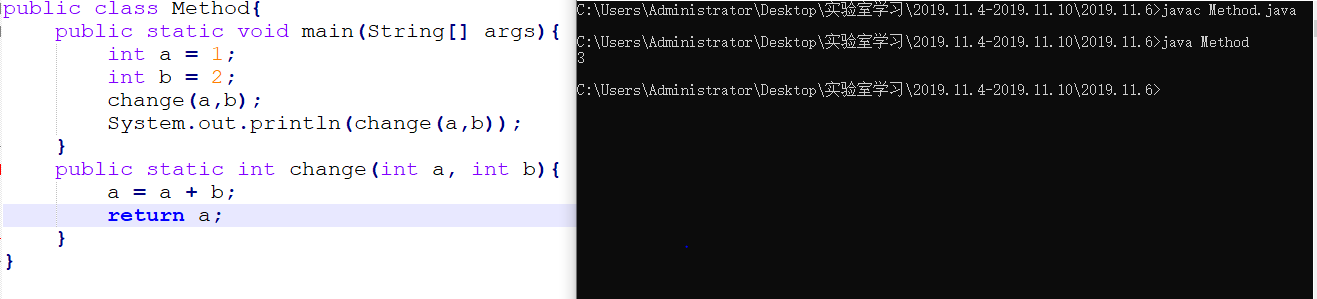
/\*返回后不改变主方法变量的值，即使变量名一样。除非将方法返回值 赋值给主方法变量 且下图也无返回值 \*/



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单词首字母大写

## 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

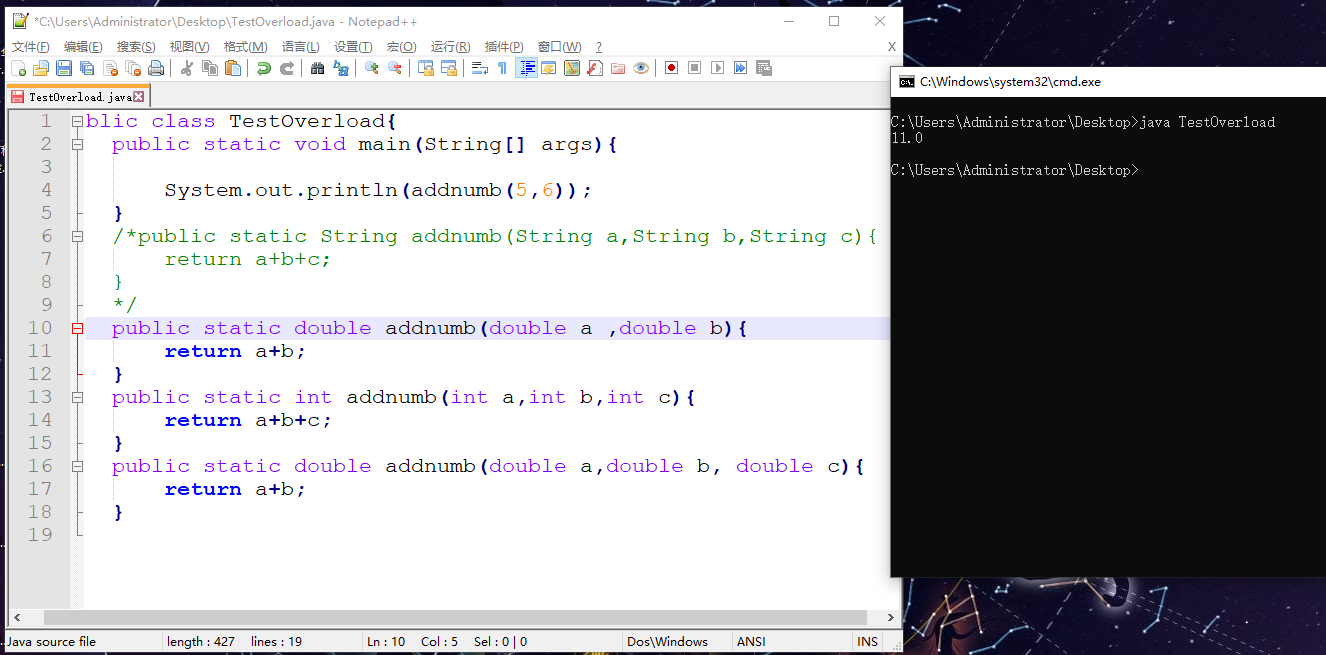
1.优先以参数个数为基准匹配方法

2.整型默认为int， int型实参可以自动转为double传给形参

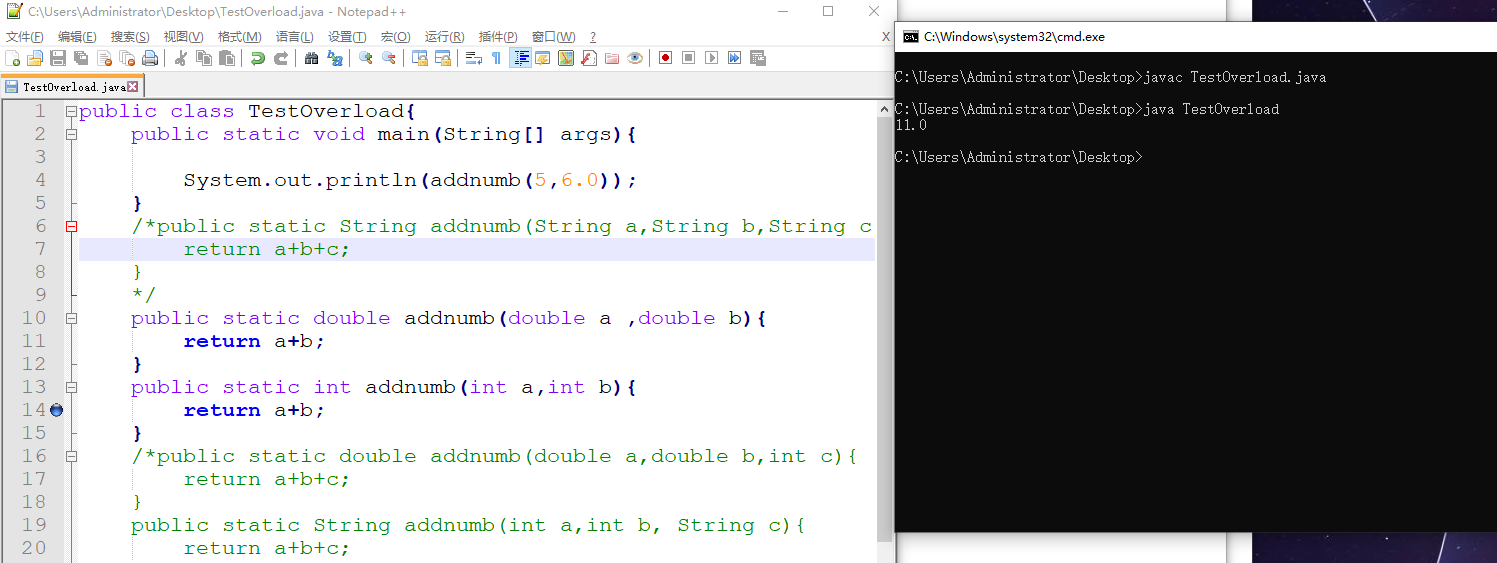
3. int型无法转为String，不兼容

4. char型可以取表值看作int计算

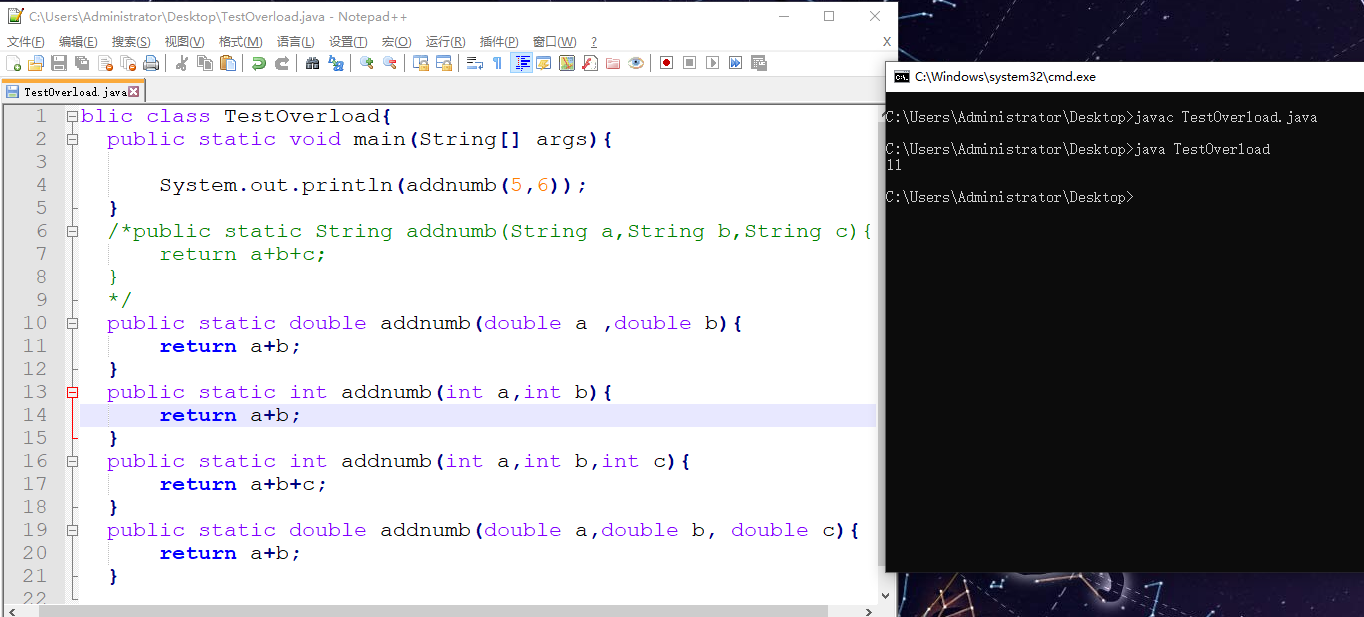
1.以参数个数为基准



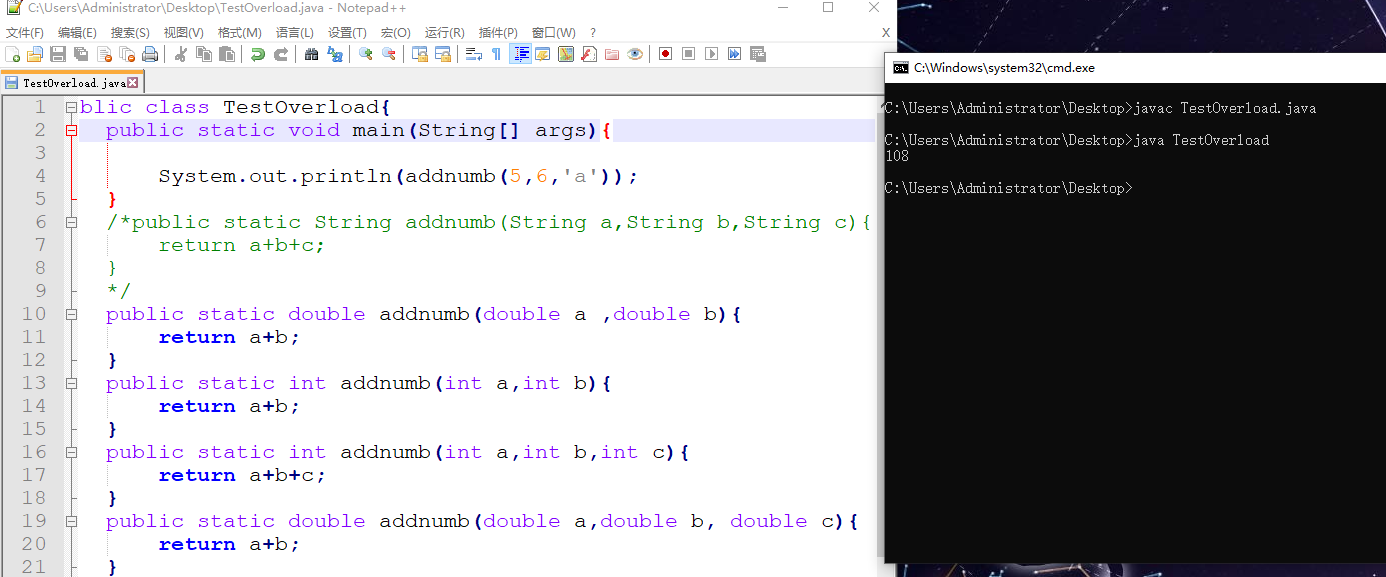
2.int型实参可以自动转为double传给形参



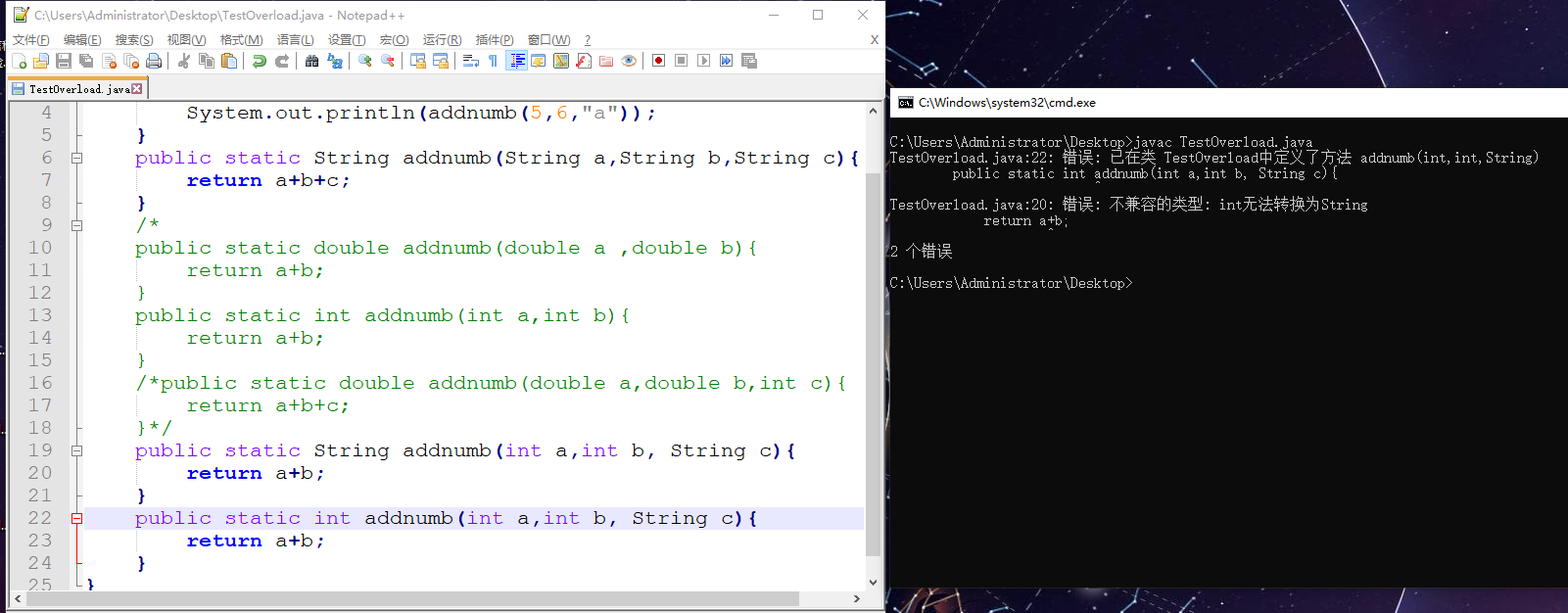
3.实参为整型时，默认为int



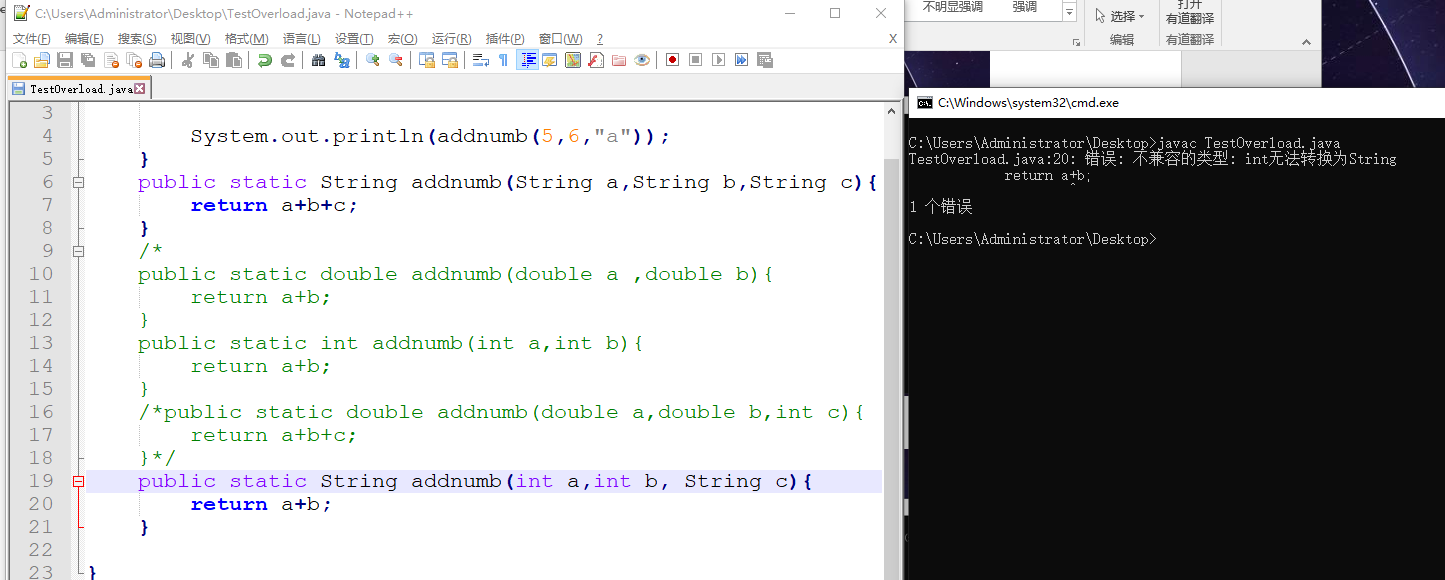
4.char型可以取表值看作int计算



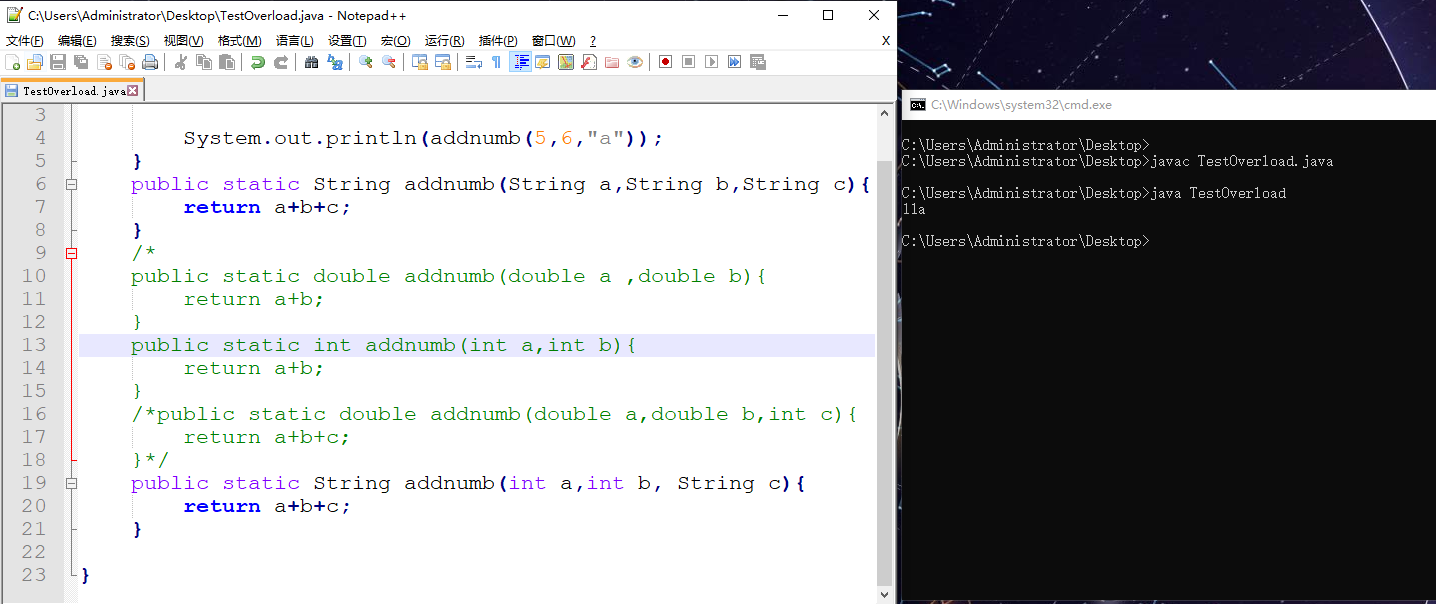
5.重载只和参数列表有关，与返回值类型无关



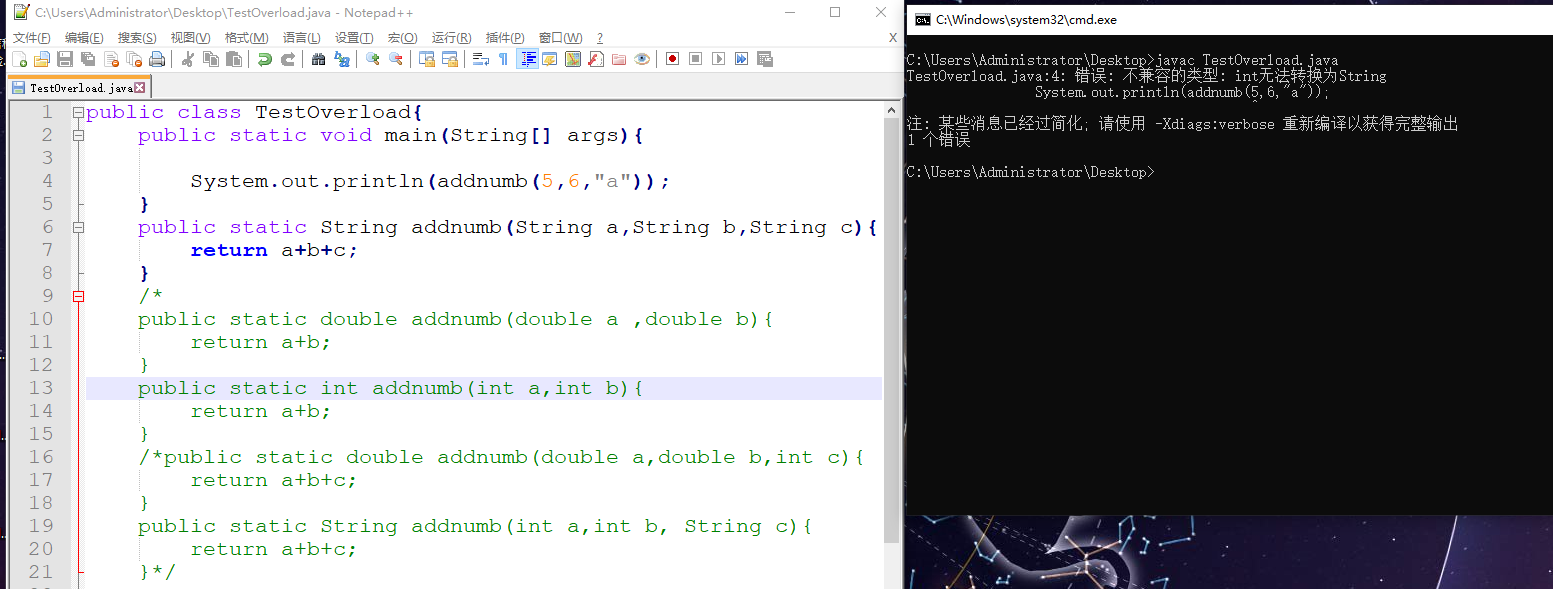
6.int型无法转为String，不兼容



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。

## 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

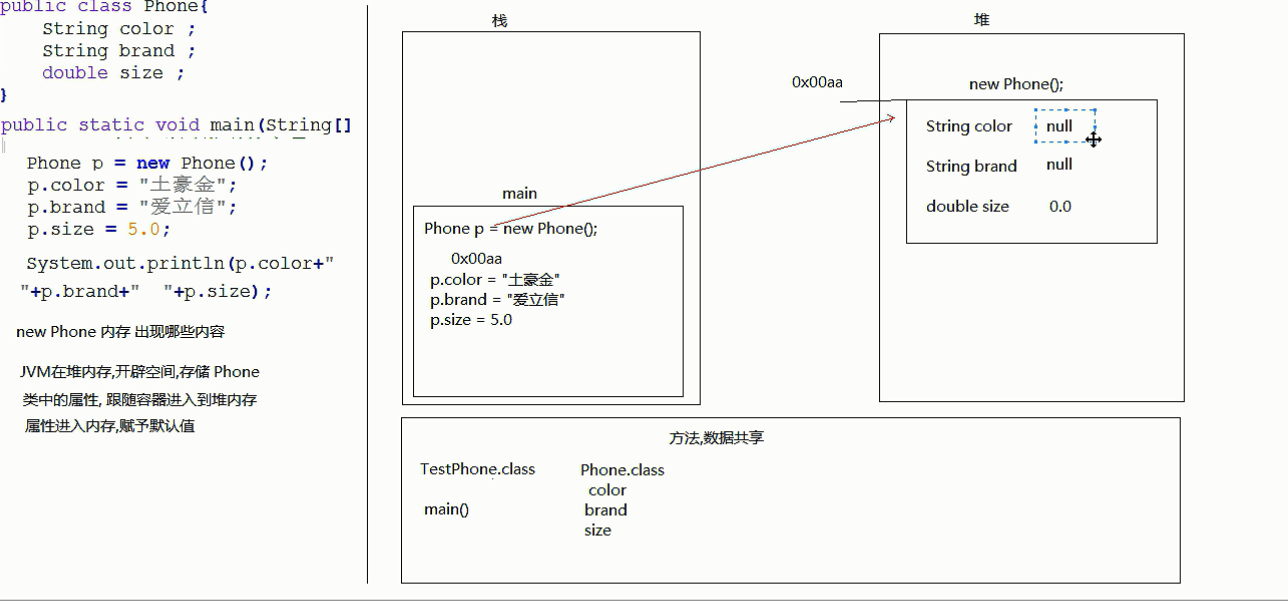
## **定义类中属性时修饰符可以不写**

取决于想要给予的权限，

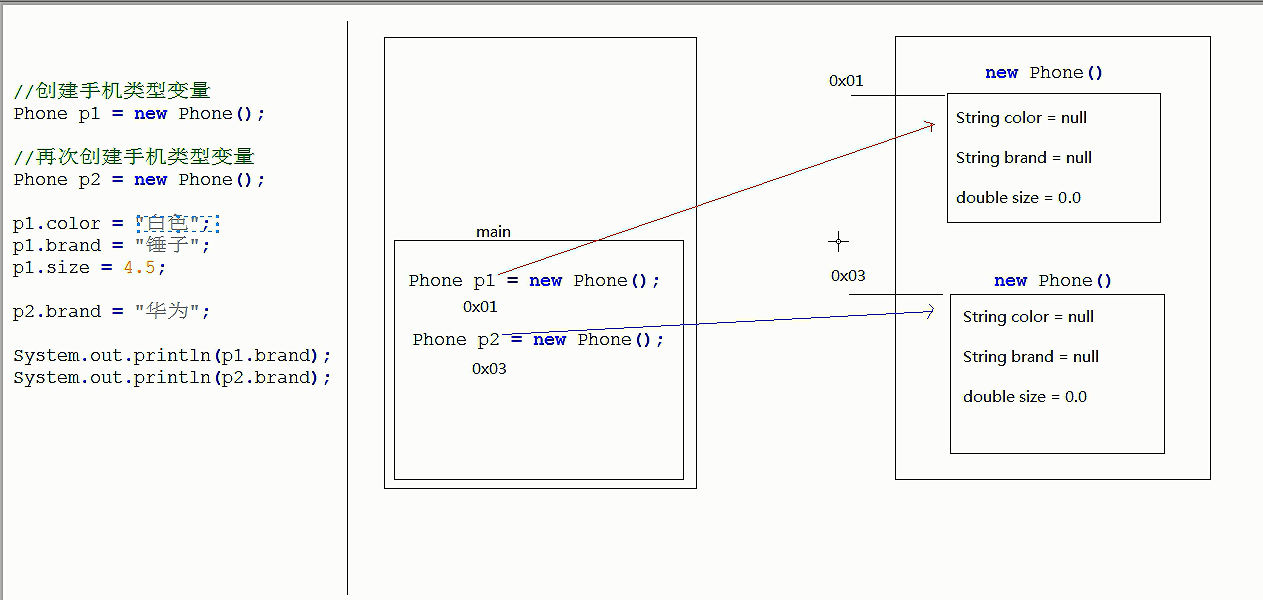
# Cmd中的运行谁编译谁

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

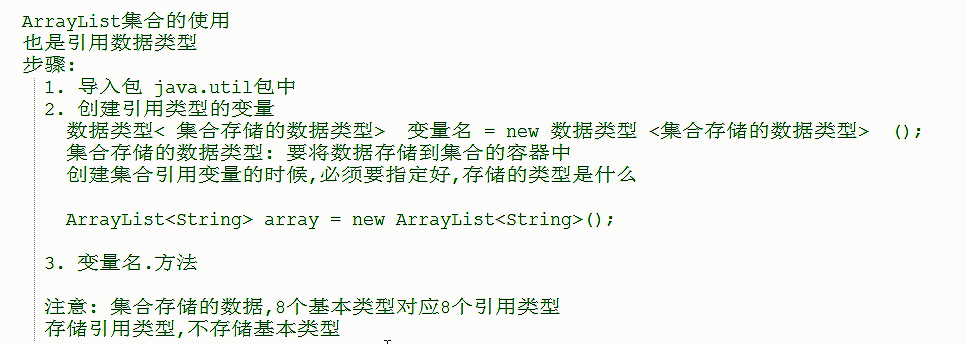


## 两个引用类型变量内存图



# ArrayList

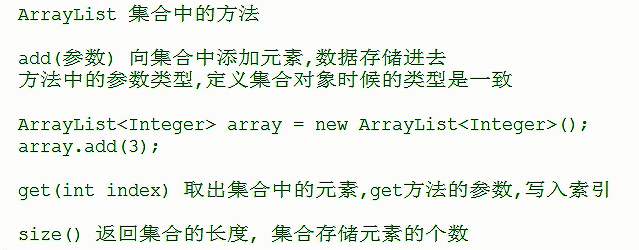
## 格式





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）

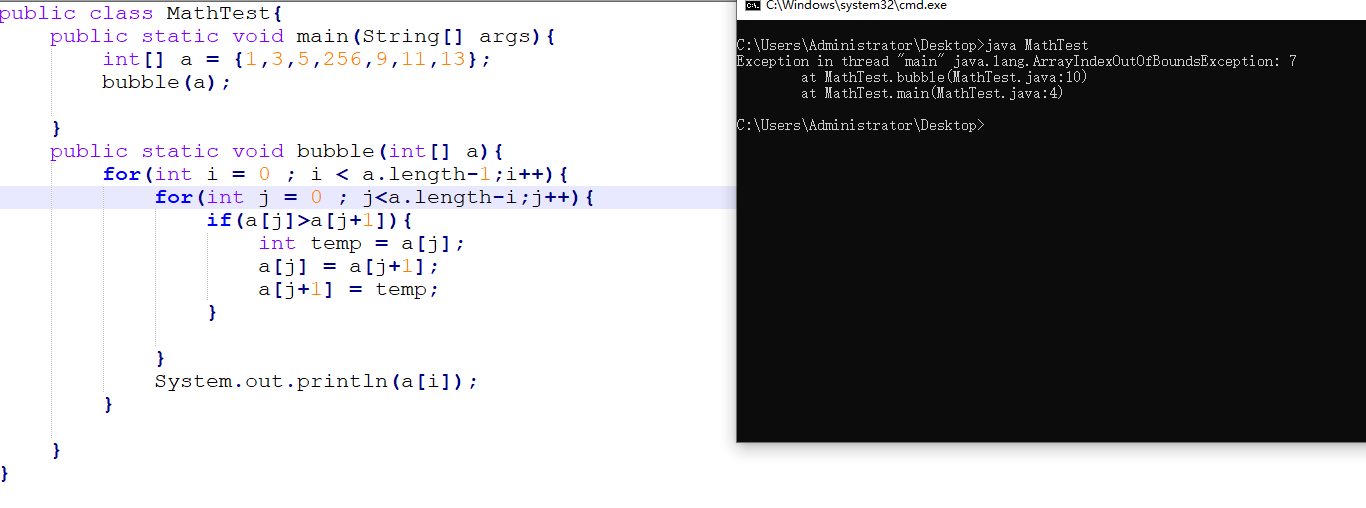
## Char转为int类型取值0-65535

## Char和short均为两字节取值不同的原因

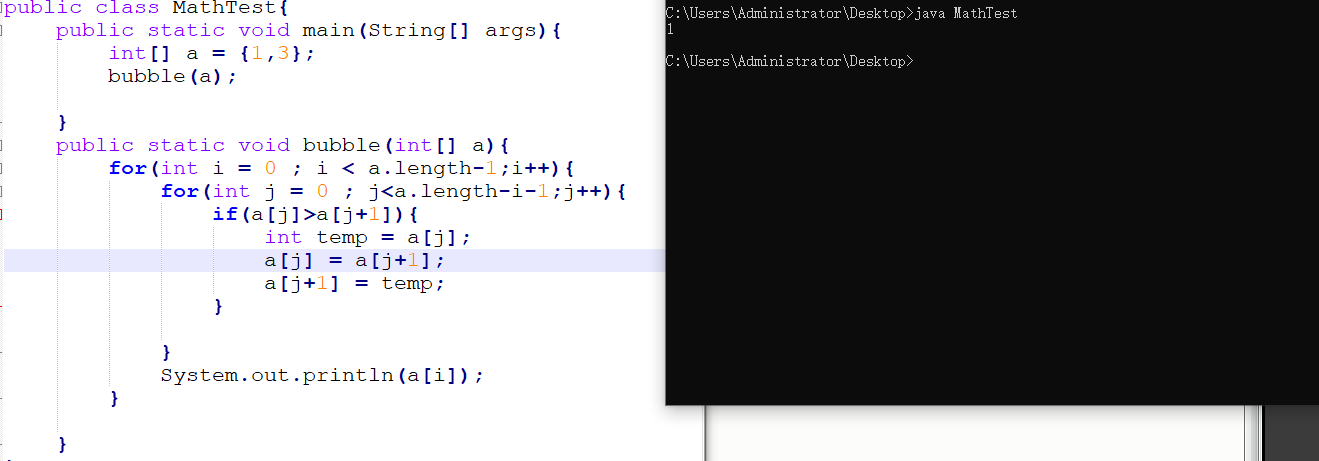


# 冒泡排序问题汇总

## 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

\* 成员变量存储在堆内存的对象中

\* 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 谁调用就指谁

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 interface

## 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）;无方法主体

### 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

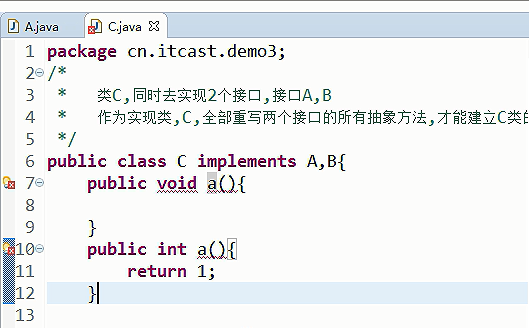
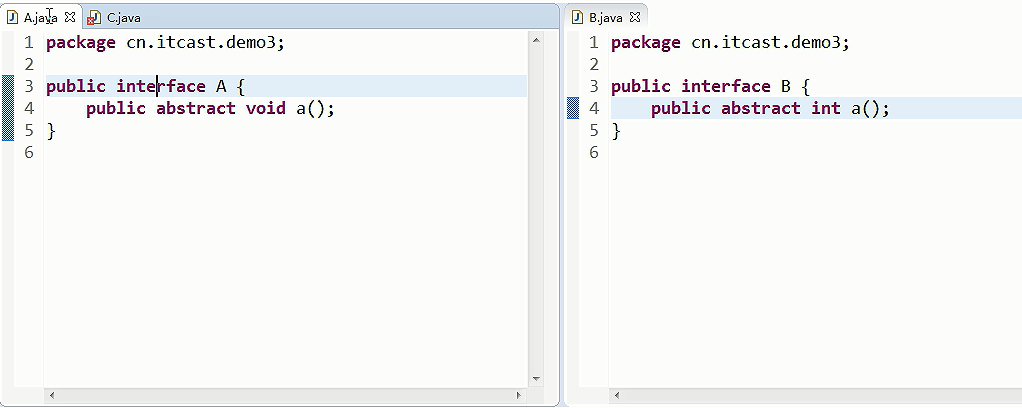
多继承时，当多个父类中有相同功能时，子类调用会产生不确定性。

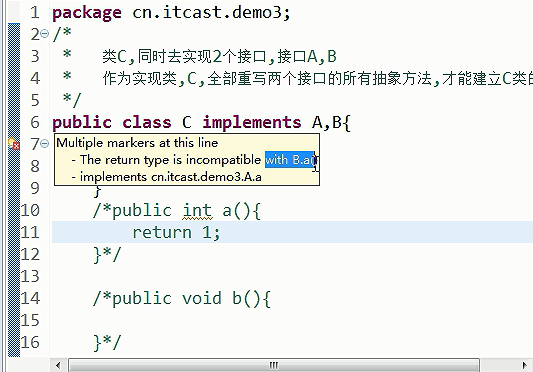
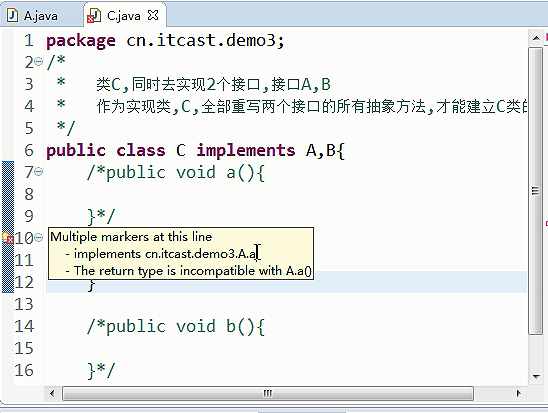
核心原因就是在于多继承父类中功能有主体，而导致调用运行时，不确定运行哪个主体内容。

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

#### 静态方法运行父类，非静态运行子类

## 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）

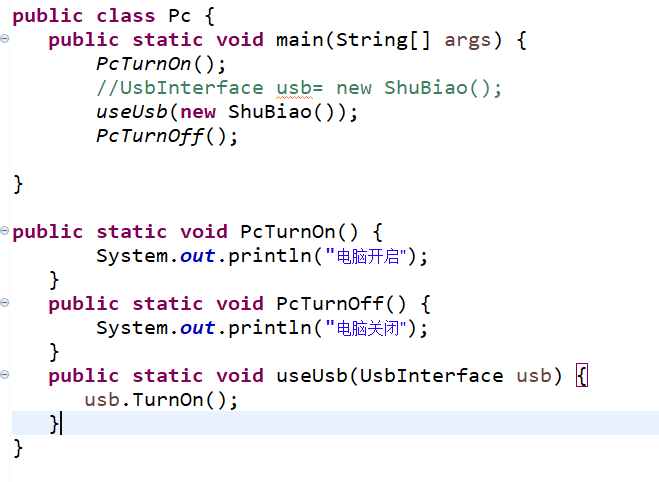
### 向下转型 将向上转型的对象强制类型转换成子类

#### 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

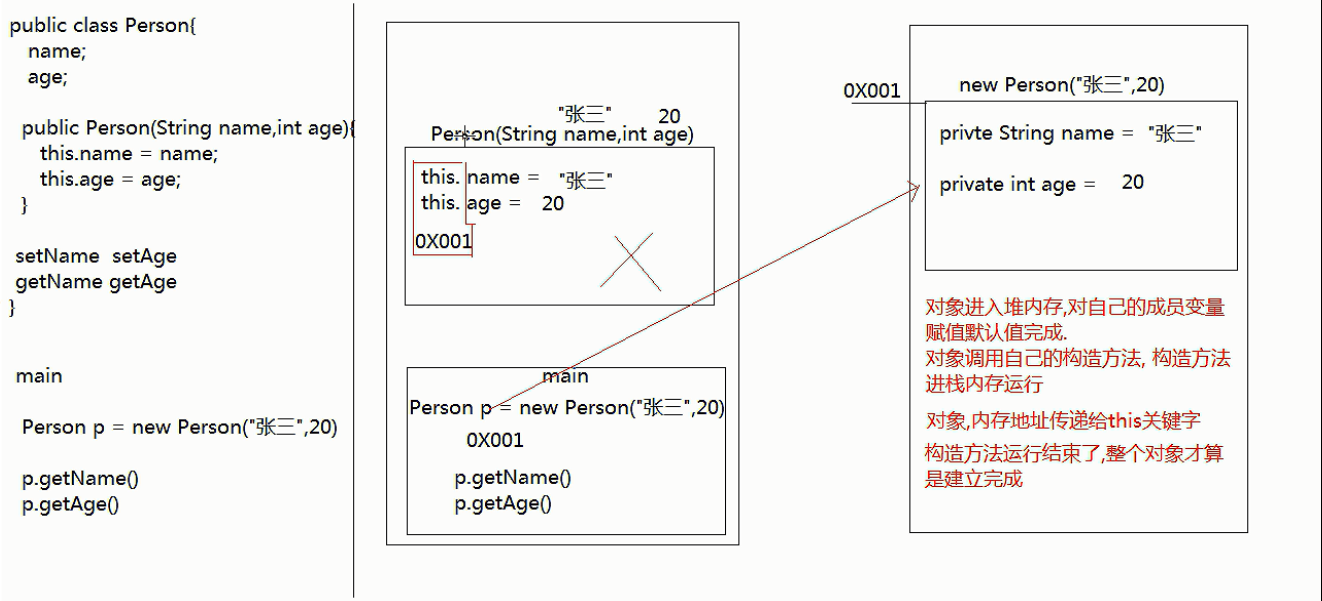
## Class中必有构造方法，不写也有

### 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（）小括号内要加上参数，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

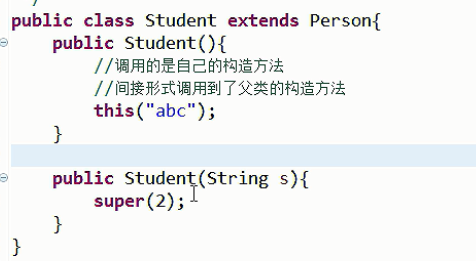
### 子类this()与super（）都不写 默认为super()；

#### 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



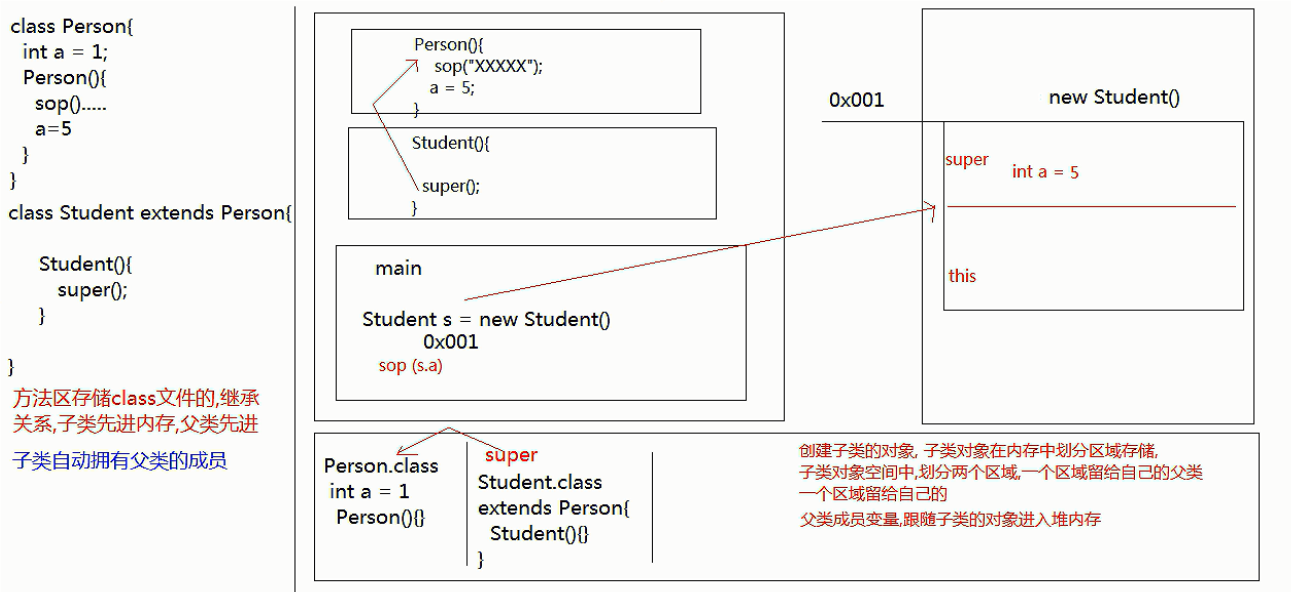
### Super（）用于了解父类成员变量的赋值。

### 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，可被调用和实例化，但是可以继承其他类，

## final不能修饰构造方法

## string类也是final类，一个final修饰的char[];

## final修饰的方法不可以被覆盖重写,可被调用，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修饰的引用类型变量，所记录的地址值不能改变，可以更改对象中的属性值

### 即不能用该变量名保存新实例化对象，也不能赋值后再写 变量名 =null；

## 修饰成员变量，需要在创建对象前赋值，否则报错。 (当没有显式赋值时每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

### 不能用set方法赋值，因为set方法在创建对象之后

## final固定的不是默认值，须手动赋值

# 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)

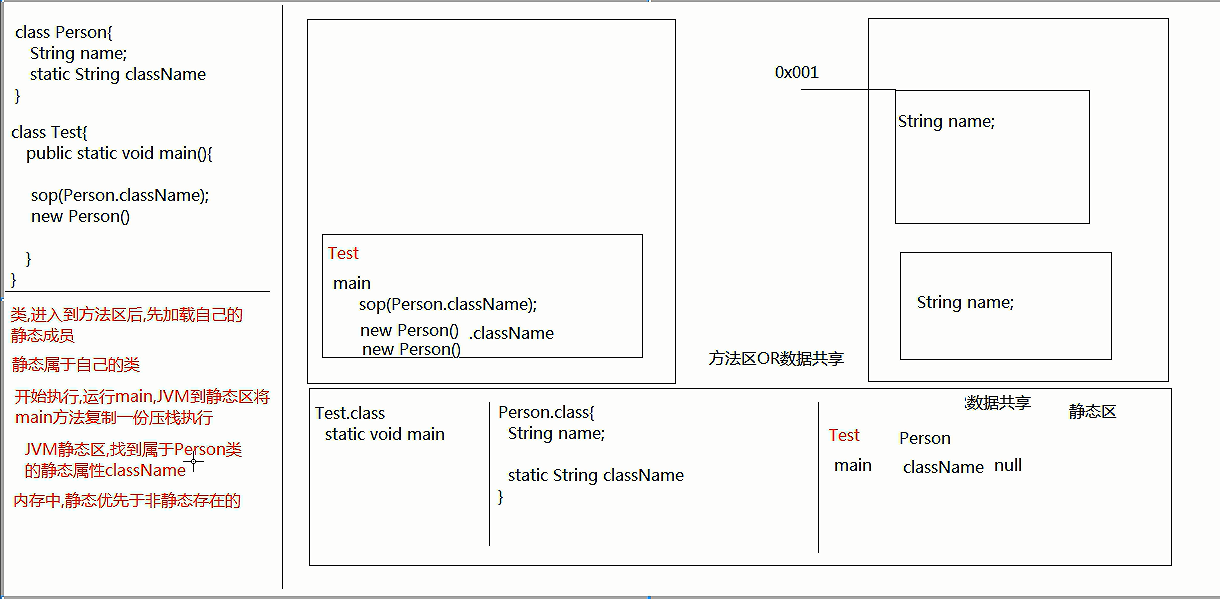
### 对象名.静态成员变量名 ------不建议使用该方式，会出现警告

### 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

### 静态优先于非静态存在于内存，所以无法调用非静态

## Static内存图



## 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。

## 方法加不加静态看是否调用了静态成员

## 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

### 外部类要使用内部类成员，必须建立内部类对象

## 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

### 可以使用成员修饰符

### 可以继承，可以实现

### 定义格式

class 外部类 {

修饰符 class 内部类 {

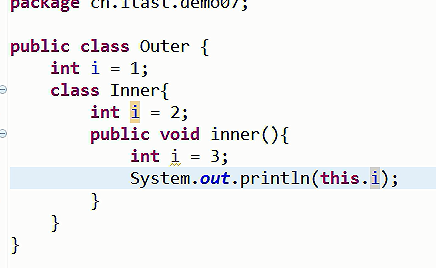
//其他代码

}

}

### 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();

### 成员内部类的同名调用



#### 输出I 值为3 （成员内部类的局部变量）

#### 输出this.i 值为2（成员内部类的成员变量）

#### 输出outer.this.i 值为1（外部类的成员变量）

## 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

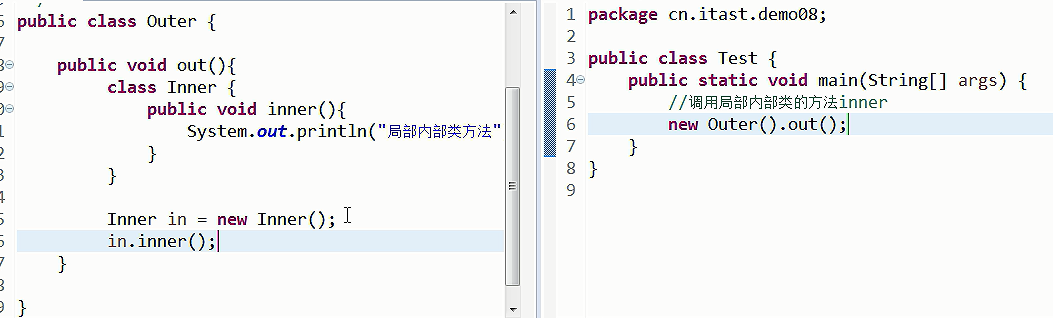
}

}

}

### **访问方式 图**

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。

#### 简化 定义实现类 重写方法 建立实现类对象 合并

#### New 接口或父类（）{重写抽象方法}.方法； 分号结束

#### 以上代码相当于父类或接口的对象，可用多态形式的变量名保存

#### 以上代码不能同时调用两个方法，但是可保存到变量名后用变量名多个调用

#### 与实现类在调用方法等价

# 包名全小写

## 导包要导入到最低层文件名，通配符也一样

# 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，运行完构造代码块再运行构造方法

#### 每创建一个对象均会执行一次构造代码块。

## 静态代码块 static{}；

### 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

### 静态代码块 >构造代码块>构造方法

# Eclipse导出文档注释 导出-java-javadoc

# Eclipse导出Jar包 导出-java-jar

# **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。（多态）

## 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。（多态）

## 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。（多态）

## 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。（多态）

## 调用方法接受返回值时，要注意方法的返回值类型，不能用子类类型接受父类，即使该父类向上

# Java的API 应用程序接口 jdk提供的类

## Application(应用) Programming(程序) Interface(接口)

# Object类概述

Object类是Java语言中的根类，即所有类的父类。它中描述的所有方法子类都可以使用。所有类在创建对象的时候，最终找的父类就是Object。

## 接口不继承object类

## equals方法

### 重写注意传入参数自动转为object形式 要向下转型

### 重写案例

class Person extends Object{

int age ;

//复写父类的equals方法，实现自己的比较方式

public boolean equals(Object obj) {

//判断当前调用equals方法的对象和传递进来的对象是否是同一个

if(this == obj){

return true;

}

//判断传递进来的对象是否是Person类型

if(!(obj instanceof Person)){

return false;

}

//将obj向下转型为Perosn引用，访问其属性

Person p = (Person)obj;

return this.age == p.age;

}

}

## toString方法

### 重写案例

class Person extends Object{

int age ;

//根据Person类的属性重写toString方法

public String toString() {

return "Person [age=" + age + "]";

}

}

# String类 免new

## String创建

### 双引号形式创建

### New形式创建

### 区别

#### 双引号形式创建

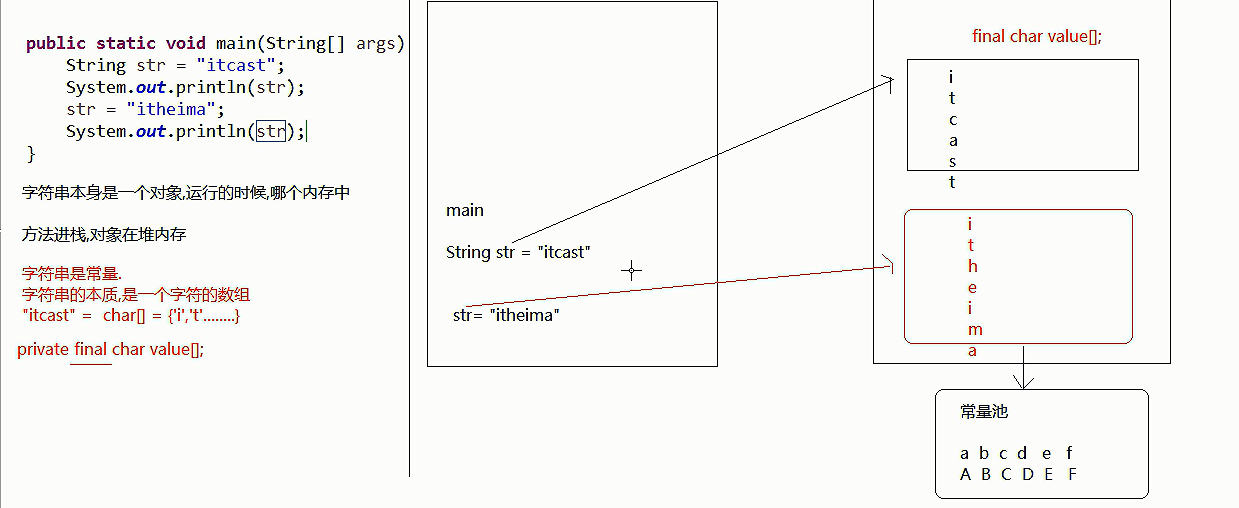
在内存中只有一个对象。这个对象在字符串常量池中

#### New形式创建

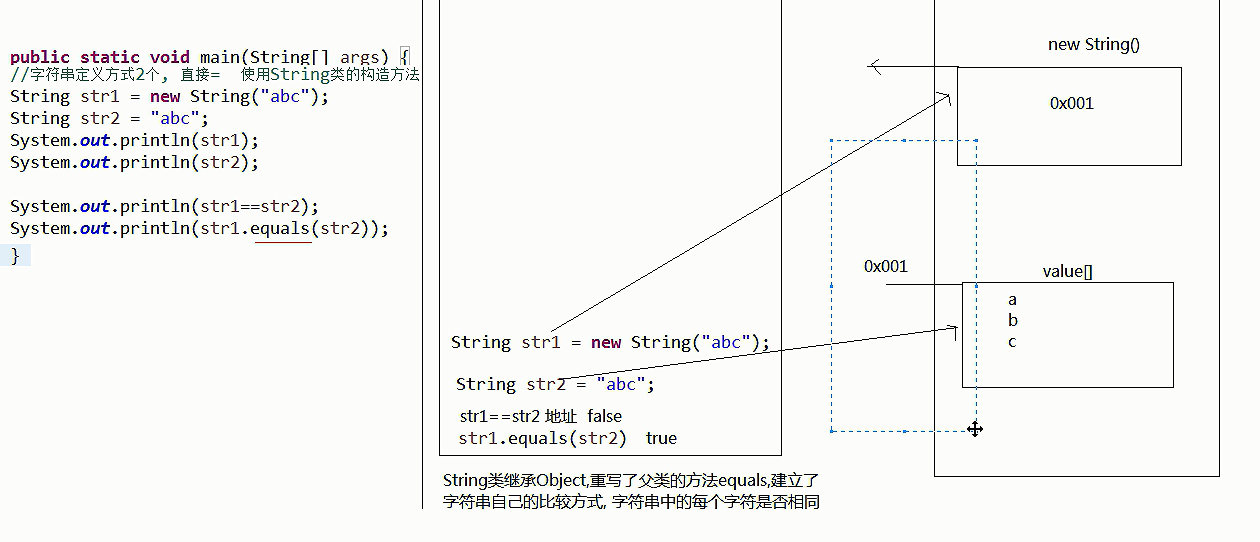
在内存中有两个对象。一个new的对象在堆中，一个字符串本身对象，在字符串常量池中

## String是一个常量 一旦创建不能改变 字符串““本身是一个对象

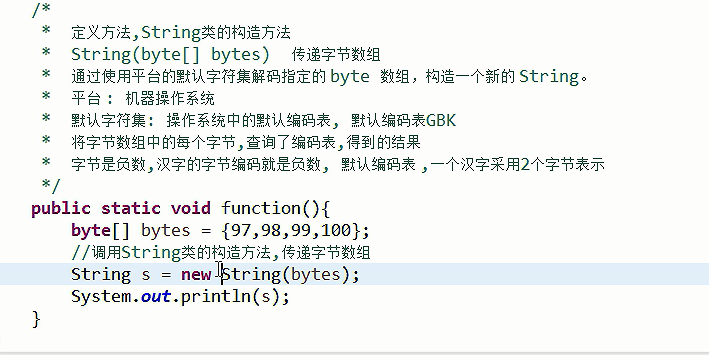
## String本质是一个一维char数组



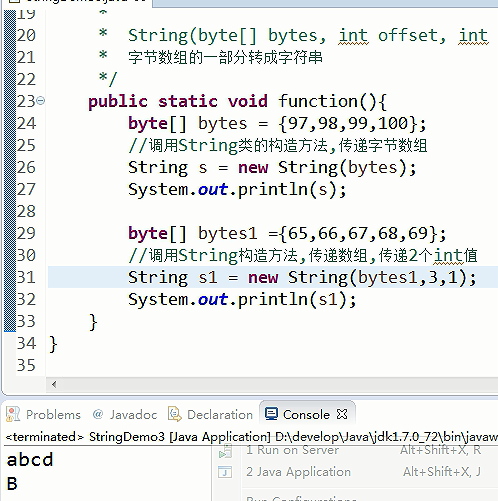
## Str1==str2 与str1.equals(str2)结果不同原因



## String（byte【】 bytes）解码平台字符集的实质

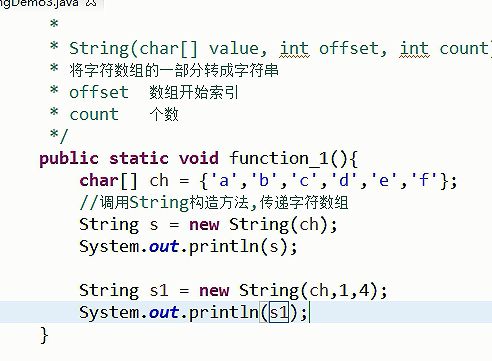


### 限定解码首位索引与解码个数



## String（char【】 value） 字符数组转为字符串 不查询编码表

### 限定首位索引与个数



## Substring 返回新字符串 原字符串是常量不可以改变 故需要新字符串接受才可以正常显示

### substring索引包头不包尾

## starswith与endswith 与contains返回值布尔

## indexof（char ch）查字符第一次出现的索引 无返回-1；

## getBytes（）；按表得到一维byte数组

## toCharArray（）返回字符数组

## toUpperCase（），toLowerCase（）；

## 方法

boolean equals(Object obj) 判断两个字符串中的内容是否相同

boolean equalsIgnoreCase(String str) 判断两个字符串中的内容是否相同, 忽略大小写

boolean contains(String str) 判断该字符串中 是否包含给定的字符串

boolean startsWith(String str) 判断该字符串 是否以给定的字符串开头

boolean endsWith(String str) 判断该字符串 是否以给定的字符串结尾

boolean isEmpty() 判断该字符串的内容是否为空的字符串 ""

int length() 获取该字符串的长度

char charAt(int index) 获取该字符串中指定位置上的字符

String substring(int start) 从指定位置开始，到末尾结束，截取该字符串，返回新字符串

String substring(int start,int end) 从指定位置开始，到指定位置结束，截取该字符串，返回新字符串

int indexOf(int ch ) 获取给定的字符，在该字符串中第一次出现的位置

int indexOf(String str) 获取给定的字符串，在该字符串中第一次出现的位置

int indexOf(int ch,int fromIndex) 从指定位置开始，获取给定的字符，在该字符

byte[] getBytes() 把该字符串 转换成 字节数组

char[] toCharArray() 把该字符串 转换成 字符数组

String replace(char old,char new) 在该字符串中，将给定的旧字符，用新字符替换

String replace(String old,String new) 在该字符串中， 将给定的旧字符串，用新字符串替换

String trim() 去除字符串两端空格，中间的不会去除，返回一个新字符串

String toLowerCase() 把该字符串转换成 小写字符串

String toUpperCase() 把该字符串转换成 大写字符串

int indexOf(String str,int fromIndex) 从指定位置开始，获取给定的字符串，在该字符串中第一次出现的位置

## StringBuffer 字符串缓冲区

### 字符数组无final

### 初始16字符

### 可变长度的数组

## StringBuilder: 字符串缓冲区

### 兼容StringBuffer 方法一样

## StringBuffer/StringBuilder:方法

### public StringBuffer append(String str) 在原有字符串缓冲区内容基础上，在末尾追加新数据

public StringBuffer insert(int offset,String str) 在原有字符串缓冲区内容基础上，在指定位置插入新数据

public StringBuffer deleteCharAt(int index) 在原有字符串缓冲区内容基础上，删除指定位置上的字符

public StringBuffer delete(int start,int end) 在原有字符串缓冲区内容基础上，删除指定范围内的多个字符

public StringBuffer replace(int start,int end,String str)在原有字符串缓冲区内容基础上，将指定范围内的多个字符 用给定的字符串替换

public StringBuffer reverse() 将字符串缓冲区的内容 反转 "abc"----"cba"

public String substring(int start) 从指定位置开始，到末尾结束，截取该字符串缓冲区，返回新字符串

public String substring(int start,int end) 从指定位置开始，到指定位置结束，截取该字符串缓冲区，返回新字符串

# 正则表达式

## Regular Expression，在代码中常简写为regex

**字符：x**

含义：代表的是字符x

例如：匹配规则为 **"a"**，那么需要匹配的字符串内容就是 ”a”

**字符：\\**

含义：代表的是反斜线字符'\'

例如：匹配规则为**"\\" ，**那么需要匹配的字符串内容就是 ”\”

**字符：\t**

含义：制表符

例如：匹配规则为**"\t**" ，那么对应的效果就是产生一个制表符的空间

**字符：\n**

含义：换行符

例如：匹配规则为**"\n"**，那么对应的效果就是换行,光标在原有位置的下一行

**字符：\r**

含义：回车符

例如：匹配规则为**"\r"** ，那么对应的效果就是回车后的效果,光标来到下一行行首

**字符类：[abc]**

含义：代表的是字符a、b 或 c

例如：匹配规则为**"[abc]"** ，那么需要匹配的内容就是字符a，或者字符b，或字符c的一个

**字符类：[^abc]**

含义：代表的是除了 a、b 或 c以外的任何字符

例如：匹配规则为**"[^abc]"**，那么需要匹配的内容就是不是字符a，或者不是字符b，或不是字符c的任意一个字符

**字符类：[a-zA-Z]**

含义：代表的是a 到 z 或 A 到 Z，两头的字母包括在内

例如：匹配规则为**"[a-zA-Z]"**，那么需要匹配的是一个大写或者小写字母

**字符类：[0-9]**

含义：代表的是 0到9数字，两头的数字包括在内

例如：匹配规则为**"[0-9]"**，那么需要匹配的是一个数字

**字符类：[a-zA-Z\_0-9]**

含义：代表的字母或者数字或者下划线(即单词字符)

例如：匹配规则为**" [a-zA-Z\_0-9] "**，那么需要匹配的是一个字母或者是一个数字或一个下滑线

**预定义字符类：.**

含义：代表的是任何字符

例如：匹配规则为**" . "**，那么需要匹配的是一个任意字符。如果，就想使用 . 的话，使用匹配规则"\\."来实现

**预定义字符类：\d**

含义：代表的是 0到9数字，两头的数字包括在内，相当于[0-9]

例如：匹配规则为**"\d "**，那么需要匹配的是一个数字

**预定义字符类：\w**

含义：代表的字母或者数字或者下划线(即单词字符)，相当于**[a-zA-Z\_0-9]**

例如：匹配规则为**"\w "**，，那么需要匹配的是一个字母或者是一个数字或一个下滑线

**边界匹配器：^**

含义：代表的是行的开头

例如：匹配规则为**^[abc][0-9]$** ，那么需要匹配的内容从[abc]这个位置开始, 相当于左双引号

**边界匹配器：$**

含义：代表的是行的结尾

例如：匹配规则为**^[abc][0-9]$** ，那么需要匹配的内容以[0-9]这个结束, 相当于右双引号

**边界匹配器：\b**

含义：代表的是单词边界

例如：匹配规则为**"\b[abc]\b"** ，那么代表的是字母a或b或c的左右两边需要的是非单词字符(**[a-zA-Z\_0-9]**)

**数量词：X?**

含义：代表的是X出现一次或一次也没有

例如：匹配规则为**"a?"**，那么需要匹配的内容是一个字符a，或者一个a都没有

**数量词：X\***

含义：代表的是X出现零次或多次

例如：匹配规则为**"a\*"** ，那么需要匹配的内容是多个字符a，或者一个a都没有

**数量词：X+**

含义：代表的是X出现一次或多次

例如：匹配规则为**"a+"**，那么需要匹配的内容是多个字符a，或者一个a

**数量词：X{n}**

含义：代表的是X出现恰好 n 次

例如：匹配规则为**"a{5}"**，那么需要匹配的内容是5个字符a

**数量词：X{n,}**

含义：代表的是X出现至少 n 次

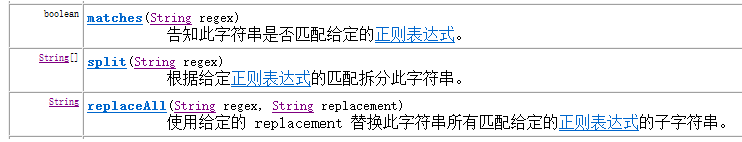
例如：匹配规则为**"a{5, }"**，那么需要匹配的内容是最少有5个字符a

**数量词：X{n,m}**

含义：代表的是X出现至少 n 次，但是不超过 m 次

例如：匹配规则为**"a{5,8}"**，那么需要匹配的内容是有5个字符a 到 8个字符a之间

## 字符串类中涉及正则表达式的常用方法



正则表达式的常用方法：

### public boolean matches(String regex) //判断字符串是否匹配给定的规则

### public String[] split(String regex) //根据给定正则表达式的匹配规则，拆分此字符串

### public String replaceAll(String regex,String replacement) //将符合规则的字符串内容，全部替换为新字符串

## **匹配正确的数字**

### 匹配正整数：”\\d+”

### 匹配正小数：”\\d+\\.\\d+”

### 匹配负整数：”-\\d+”

### 匹配负小数：”-\\d+\\.\\d+”

### 匹配保留两位小数的正数：”\\d+\\.\\d{2}”

### 匹配保留1-3位小数的正数：[\\d+\\.\\d{1,3}](\\\\d+\\\\.\\\\d{1,3})

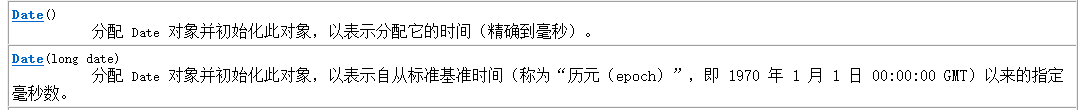
## 匹配合法的邮箱

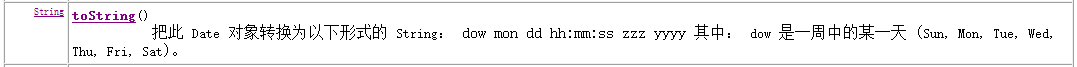
匹配规则：

”[a-zA-Z\_0-9]+@[a-zA-Z\_0-9]+(\\.[a-zA-Z\_0-9]+)+”

[\\w+@\\w+(\\.\\w+)+](file:///\\w+@\\w+(\\.\\w+)+)

# Date 精确到毫秒







构造方法：

public Date()// 系统当前日期时间

public Date(long date) 得到一个1970年1月1日 0点这个时间基础上，加上参数date 毫秒值对应的日期时间

方法：

public long getTime() 获取日期所对应的毫秒值

# DateFormat:是日期/时间格式化子类的抽象类, 使用其子类SimpleDateFormat 实例化构造方法：

## public SimpleDateFormat() 默认的格式化操作

## public SimpleDateFormat(String pattern) 按照指定的格式，进行日期格式化

## 日期和时间模式

### y 年

### M 年中的月份

### d 月份中的天数

### H 一天中的小时数（0-23）

### m 小时中的分钟数

### s 分钟中的秒数

### S 毫秒数

## 方法：

### public final String format(Date date) 把日期 格式化成字符串

### public Date parse(String source) 把日期字符串 转换成 日期对象

# Calendar:日历类，可获取日期中指定字段的值

## public static Calendar getInstance() //获取日期对象

## public int get(int field) //获取时间字段值

## public void add(int field,int amount) //指定字段增加某值

## public final void set(int field,int value)//设置指定字段的值

## public final Date getTime() //获取该日历对象转成的日期对象

# 基本类型包装类

## 8种基本类型对应的包装类

### 基本类型 包装类

### byte Byte

### short Short

### int Integer

### log Long

### float Float

### double Double

### char Character

### boolean Boolean

## 自动装箱、自动拆箱

### 自动装箱：基本数值转成对象（int  Integer）

### 自动拆箱：对象转成基本数值（Integer  int）

## 常用方法

### public int parseInt(String str):把字符串转成基本类型int

### public static String toString(int x):把基本类型int转成字符串

### public static Integer valueOf(int x):把基本类型i字符串转成Integer对象

### public int intValue():以 int类型返回该包装类对象的值

# **System类: 系统属性信息工具类**

## public static long currentTimeMillis()：获取当前系统时间与1970年01月01日00:00点之间的毫秒差值

## public static void exit(int status)：用来结束正在运行的Java程序。参数传入一个数字即可。通常传入0记为正常状态，其他为异常状态

## public static void gc()：用来运行JVM中的垃圾回收器，完成内存中垃圾的清除。

## public static String getProperties()：用来获取指系统属性信息

# Arrays类：数组操作工具类

## public static void sort方法，用来对指定数组中的元素进行排序（元素值从小到大进行排序）

## public static String toString方法，用来返回指定数组元素内容的字符串形式

## public static void binarySearch方法，在指定数组中，查找给定元素值出现的位置。若没有查询到，返回位置为-插入点-1。要求该数组必须是个有序的数组

# Math类：数学运算工具类

## abs方法,结果都为正数

## ceil方法，结果为比参数值大的最小整数的double值

## floor方法，结果为比参数值小的最大整数的double值

## max方法，返回两个参数值中较大的值

## min方法，返回两个参数值中较小的值

## pow方法，返回第一个参数的第二个参数次幂的值

## round方法，返回参数值四舍五入的结果

## random方法，产生一个大于等于0.0且小于1.0的double小数

# throwable 所有异常错误的超类

## 子类

### error错误 必须修改程序 编译期无 运行期出现

### exception异常 修改异常仍可运行 编译异常 运行异常

## jvm建立异常对象 抛出给调用者 main方法抛出会抛给虚拟机

## 一旦异常被抛出 之后程序不再运行

## Throw 在方法内抛出异常

### Throw new 对象 （异常或其子类）

### Throws 在方法声明上 表明方法可能有问题 后加异常类名

## Try{} catch（异常类型 变量名）{处理异常代码} finally{必须执行}

### Finally代码块内代码必须执行 除非结束虚拟机

## 多catch处理细节

### 平级异常无顺序 有继承关系的异常类型 越高级越在下面

### 原因 多态 高级写上面无论抛出哪种 catch里的高级异常都能接受

### 运行异常

#### 声明中无throws

#### 因为一般不会发生 一旦发生应该修改源代码

#### 用于出现不合理的条件停止程序提示用户

## 方法重写时的异常

### 父类抛出异常

### 子类可以抛出可以不抛出

### 若抛出 父类异常应该大于子类异常

### 父类不抛 子类不能抛出

### 此时子类重写方法调用了异常方法 只能trycatch

## 自定义异常需要继承exception或者runtimeexception 否则不可抛