# 以不变应万变：final关键词和不变性

## 什么是不变性？(Immutable)

### 如果对象在被创建后，状态就不能被修改，那么他就是不可变的。

这个含义啊，不仅仅是说他的对象指向，也就是引用不可变，还包括他里面的字段啊，里面的成员变量啊，这些都是不能变化的，那么我们就来举一个例子。

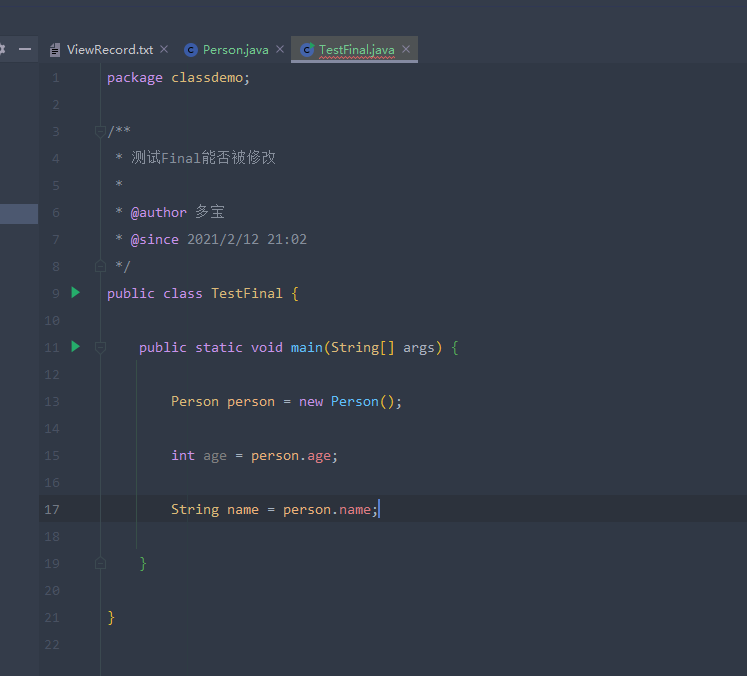
1. person对象，age和name都不能再变

比如说是我处了一个娘们，可能说是因为他脸盘子大，后来我们分手了，但是他的姓名、年龄都不能在改变了，因为后期我也去找其他的娘们去了，所以呢再想去设置他，比如说我们想修改它，用set、get方法之类啊，都不能去做到了，那么我们就用类去演示一下这样的的一个Person对象该如何去写？

对于我们不可变的对象而言，他有一个非常好的特点，他就是他一定是线程安全的

1. 具有不变性的对象一定是线程安全的，我们不需要对其采取任何额外的安全措施，也能保证线程安全

这是为什么呢？因为如我们刚才所示，你不能修改不可变性的类中的成员变量。但是你可以读取，你直接在其他类里面直接调用不可变性类的成员变量去将这个值赋予给其他的成员变量，我们这个属性不能改不代表不能读



现在我们直接读出来赋予给一个新的变量，你现在无论跑多少个线程同时去读取，并且你没有加任何的保护措施，这都是没问题的，因为你能做的只是并发读取，没有任何操作是可以修改他的值，所以呢他依然是并发安全的，不可变是我们并发安全版图中一个重要的模块。也是一个很重要的策略，我们在之前啊讲过各种手段，比如说是锁，或者是并发容器，他们这个对象是可变的，正因为可变呢，我们才需要采取很多的措施，来保障他的安全，可是到了这里只要他是不可变的，我们什么措施都不用采取。这就是不可变的一大优势。

## final的作用？

### 早期

1. 锁定
2. 效率：早期的Java实现版本中，会将final方法转为内嵌调用

内嵌的意思就是说，我们用一个方法去调用另外一个fianl方法，那么当编译器发现他是final的，他就会把那个方法里面的东西全都给挪过来，相当于我们只在同一个方法内就完成了整个的工作，而不是方法之间调来调去，因为我们知道啊，方法之间的调用它是有一定的性能损耗的，所以呢这样一来可以提高一定的效率。

不过在现在的这个版本中啊，已经是有所不同了。

### 现在

1. 类防止被继承 、方法防止被重写、变量防止被修改。
2. 天生是线程安全的，而不需要额外的同步开销。

其实现在用final的一大原因就是实现线程安全，如果我们可以用final把对象做到不可变，那就不在需要额外的同步开销，这是一个很划算的生意吗，并且第三点呢，就是之前的，我们刚才说在早期版本中用final带来的性能提高，在目前我们几乎是不需要再考虑了，因为啊我们目前的JVM他非常智能，她会把能优化的点都优化到，这样一来呢用不用final所带来的区别可以说是可以忽略不计的，而且也有人做过测试，目前呢从性能的角度考虑已经看不出他的优势了。目前我们使用它更多的还是基于设计的清晰，因为修饰之后呢，我们就知道了这个属性或者这个类或者这个方法他拥有了final语义，也就是我们不希望它被继承啊重写啊或者是被修改，这是目前使用final的原因，而不再是性能原因了。

## 3种用法：修饰变量、方法、类

### final修饰变量

#### 含义：被final修饰的变量，以为着值不能被修改。如果变量是对象，那么对象的引用不能变，但是对象自身的内容依然可以变化。

#### final修饰三种变量

1. 、 final instance variable (类中的final属性)
2. 、final static variable (类中的static final属性)
3. 、final local variable (方法中的final变量)

#### final修饰变量，赋值时机

我们刚才说，那三种变量他们最主要的区别呢就是在赋值时机上，一旦一个属性被声明成final之后啊，该变量则只能被赋值一次，且一旦被赋值，final的变量就不能在被改变，无论如何也不会变。

### 赋值时机

1. 、final instance variable(类中的final属性)
2. 第一种就是我声明变量的等号右边直接赋值。
3. 第二种就是在构造函数中赋值
4. 第三种就是在类的初始化代码块中赋值(不常用)
5. 如果不采用第一种赋值方法，那么就必须在第2、3种挑选一个来赋值，而不能不赋值，这是final语法所规定的。
6. 、final static variable(类中的static final属性)
7. 两种赋值时机：除了在声明变量的等号右边直接赋值外，static final变量还可以用static初始代码块来赋值，但是不能用普通的初始代码块赋值
8. 、final local variable(方法中的final变量)
9. 和前面两种不同，由于这里的变量是在方法里面的，所以没有构造函数，也不存在初始代码块。
10. 方法中的final变量不规定赋值时机，只要求在使用前必须赋值，这个方法中非final变量的要求也是一样的。

#### 为什么要规定赋值时机？

你这不同的变量还有不同的时机，你给我规定这么多，我哪里记得住？你有什么意义呢？

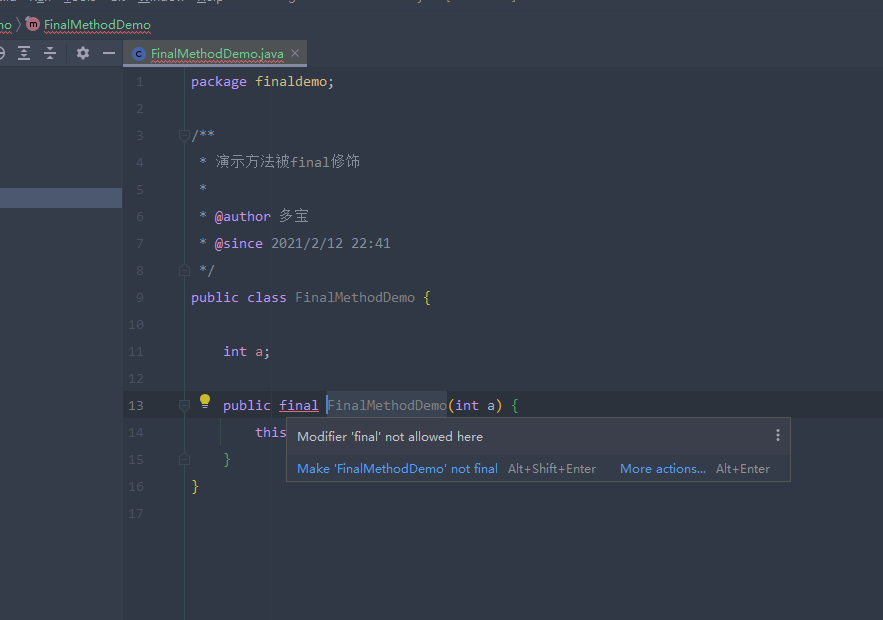
其实是很有意义的，我们之所以JAVA会设计成这样，我们站在他的角度去考虑一下，如果他不给我们这么严格的要求，那么会怎么样呢？

1. 、我们来思考一下为什么java要求这么严格？：如果初始化不赋值，后续赋值，就是从null变成你的赋值，这就违反final不变的原则了！

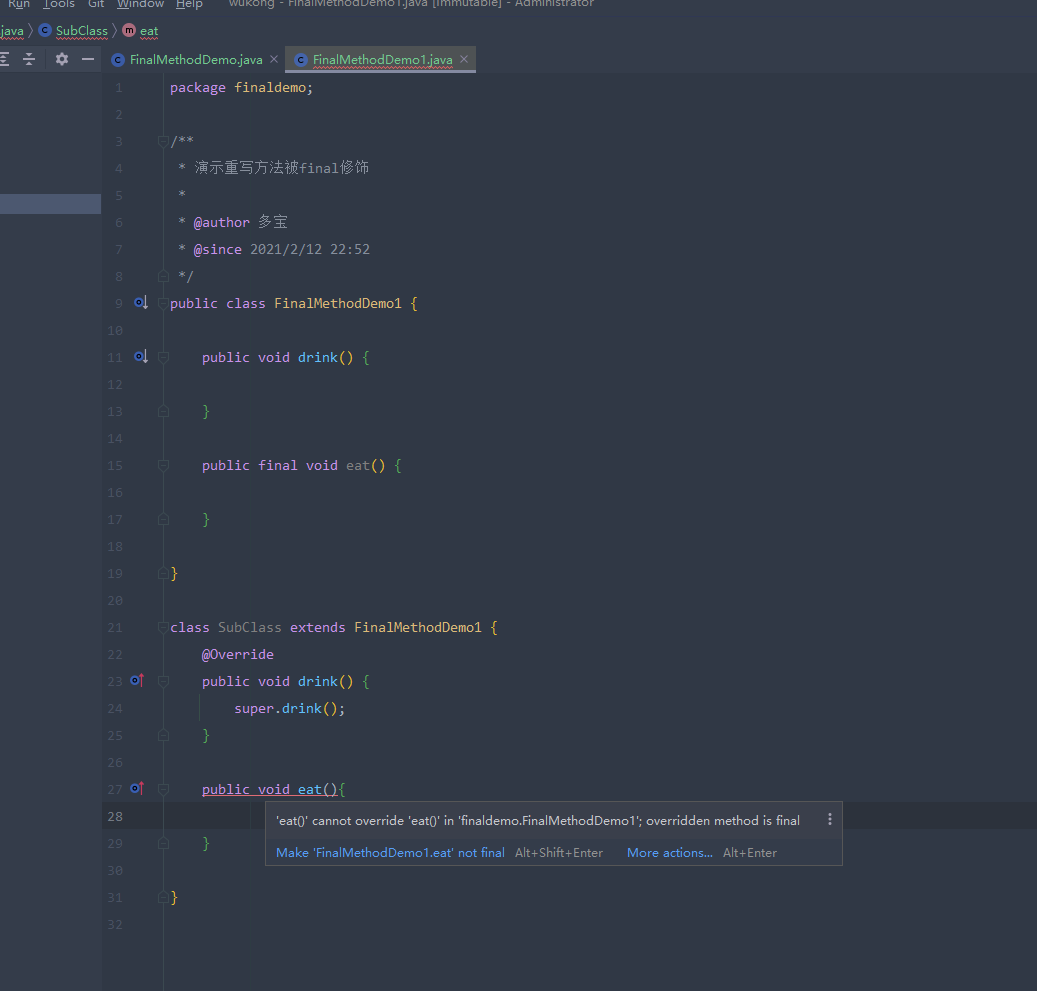
final的原则是连这个从null修改过来的过程都不允许有，只要你一诞生那么就再也变不了了，你可以诞生为null，那是Null变不了其他的值

### final修饰方法

#### 构造方法不允许final修饰



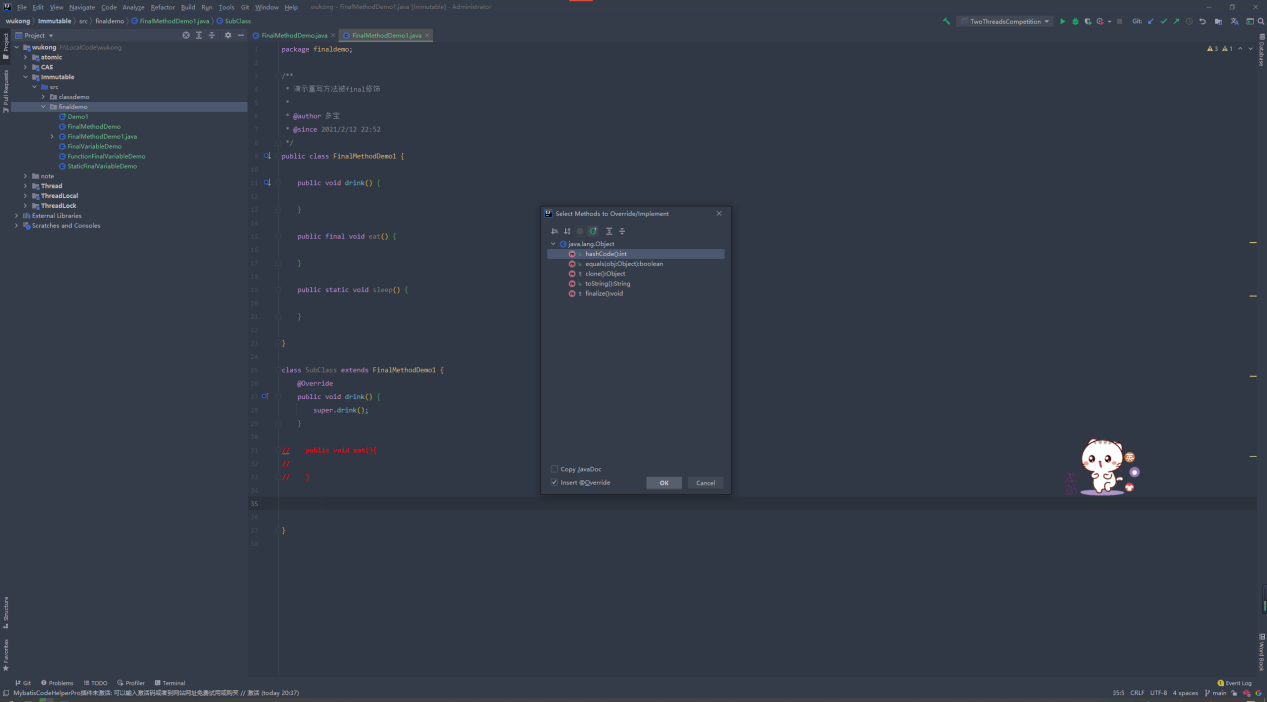
#### 不可被重写，也就是不能被override，即便是子类有同样名字的方法，那也不是override，这个和static方法是一个道理



编译器就提示我们不能重写一个来自父类中被final修饰的方法。

#### 引申：static方法不能被重写

在这里final和static在被重写的方面是很像的。



我们继承了父类之后，我们是不能重写父级的静态方法，但是我们可以在子类中创建和父类静态方法一样的方法，这一点是和final修饰的方法是不一样的。

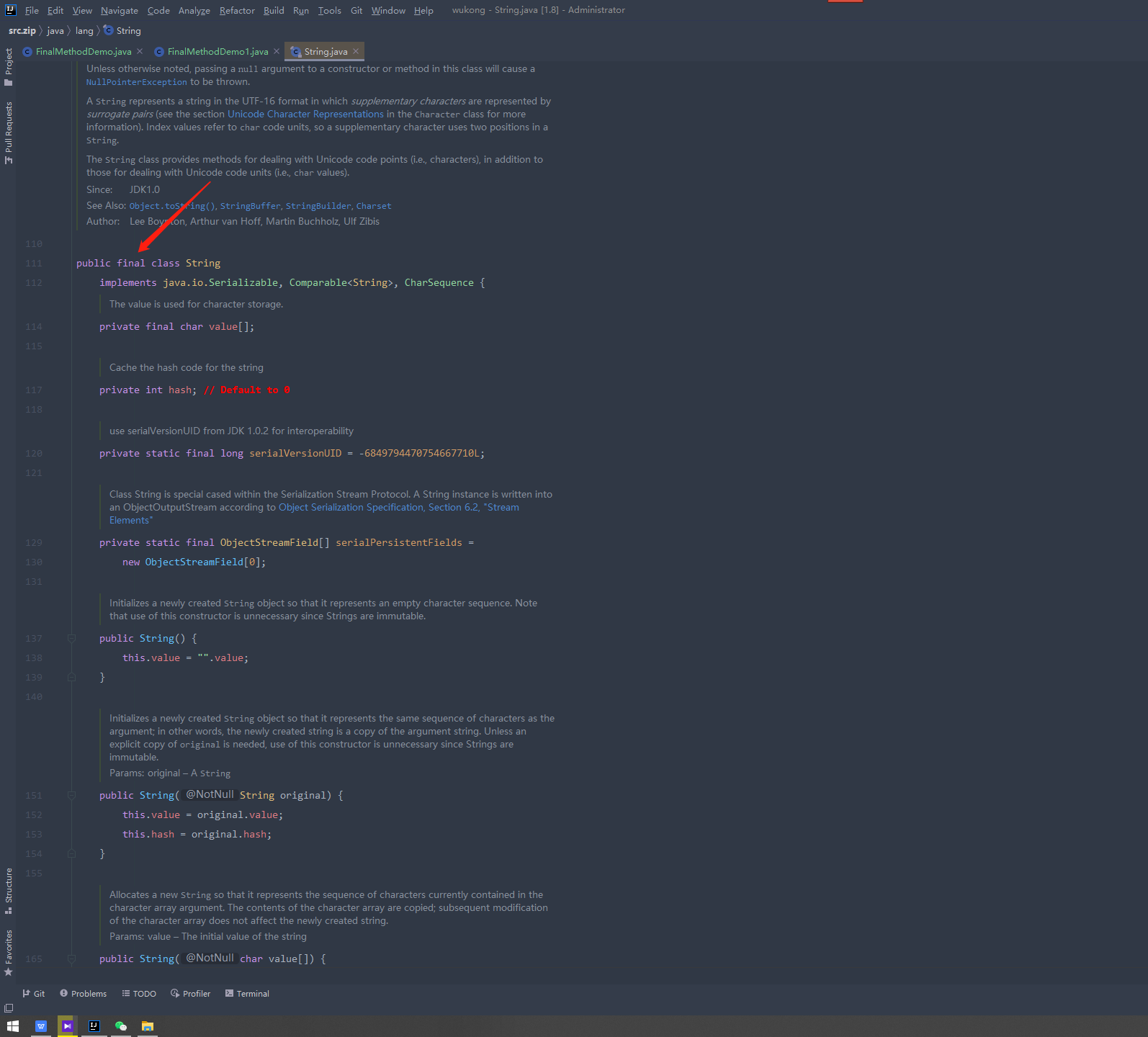
这是为什么呢？

这是因为啊，我们的static变量它是在创建的时候就已经绑定了这个方法，而不是后期动态绑定的，但是父级的static方法在子类中虽然看上去是重写的，其实并不是，子类中的static方法是只属于子类的，和子类继承的父类是没有任何关系的，也不是从写的父类，而父类中的static方法也只是和父类进行绑定的，所以虽然看上去他们长得很像，甚至长得一模一样，但是对于java而言，他们是属于两个不同的对象的两个静态方法，它们不存在override的关系。

### final修饰类

#### 不可被继承

例如典型的String类就是final的，我们从没见过哪个类是继承String类的。



我们的String，一个如此高频使用的类，他就被final修饰了，这一定是有含义的，所以修饰之后他的含义就是当前这个类不可被继承，所以在我们的印象中好像确实是没用过继承String类的类，因为它是final的。

## 注意点

### final修饰对象的时候，只是对象的引用不可变，而对象本身的属性是可以变化的。

实时上他本身的属性是完全可以变化的，这个final所针对的是这个对象的引用而不是这个对象内部所针对的数据

### final使用规则：良好的编程习惯

什么是良好的编程习惯呢？也就是说啊，如果我们明确的知道某一个对象它在生成之后不会被改变，而且这个意图通常是我们定义的，我们可能就希望某一个对象或者某一个值或者某一个属性它在创建好之后呢是不再变化的，那么这个时候我们最好就加上一个final。一方面呢，可以保证它的不变性，因为加了这个关键字之后谁都改变不了了，避免我们出错，那么小伙伴可能想，我不会出错的，我不会再去修改它的，我不加不行吗？所以在这里我们用了final之后的第二点很大的好处呢其实就是可以提醒其他的编程的小伙伴，当他们看到这个对象被final修饰之后，他都不用去找他是在哪里进行赋值的，因为赋值的地方不唯一嘛。也可能是在等号右边，也可能是在构造函数，也有可能是代码块，所以他看到final之后心里立刻就会明白了，OK，这个变量是有其他的人想让它不可变，所以对他去开发，他去理解我们编写的这个类也是有很大好处的，相当于是一种很明确的行为，告诉别人，它是被final修饰的，这样一种明确的提示，对于团队协作也是很有好处的。

## 不变性和final的关系

### 不变性并不意味着，简单的用final修饰就是不变性

这里你又说你被final修饰了他不能被修改，又说修饰了又不具备不可变性，这到底是什么意思呢？

#### 对于基本数据类型，确实被final修饰后就具有不变性

#### 但是对于对象类型，需要该对象保证自身被创建后，状态永远不会变才可以

这样才叫对象不可变，而不仅仅是被final修饰就不可变了，假设说对象里面的某个属性不是被final修饰的，那么即便我们在创建该对象的时候，给这个对象加了final，这个对象也不是不可变的，因为该对象内部的属性不满足要求，如果对象里面的全部属性都被final修饰，那么该对象就是一个不可变的了。

#### 如何利用final实现对象不可变？

1. 把所有属性都声明为final？

虽然是上面说到不可变性的对象是其内部所有属性用final修饰，但是呢这句话也是有问题的，因为该对象里面如果有一个属性是一个对象的话，那么即便你用final修饰，他只是对象的引用不可变，而对象本身还可变。

那么我是不是这个对象里面有个对象属性，那就是不是我这个对象不可能不可变了？因为我里面的对象他永远可变。

这句话也是不正确的，

1. 一个属性是对象类型的不可变对象的正确例子。

#### 4.总结(满足以下条件，对象才是不可变的)

1.对象创建后，其状态就不能修改

2.所有属性都是final修饰的

3.对象创建过程中没有发生溢出。

这是我们线程安全的一种情况，如果你发生溢出了，那这个对象就会被其他的线程拿到并且修改，那这是不满足不可变条件的。

### 2.把变量写在线程内部 -- 栈封闭

#### 1、在方法里新建的局部变量，实际上是存储在每个线程私有的栈空间，而每个栈的占空间是不能被其他线程所访问到的，所以不会有线程安全问题，这就是著名的”栈封闭”技术，是“线程封闭”技术的一种情况。

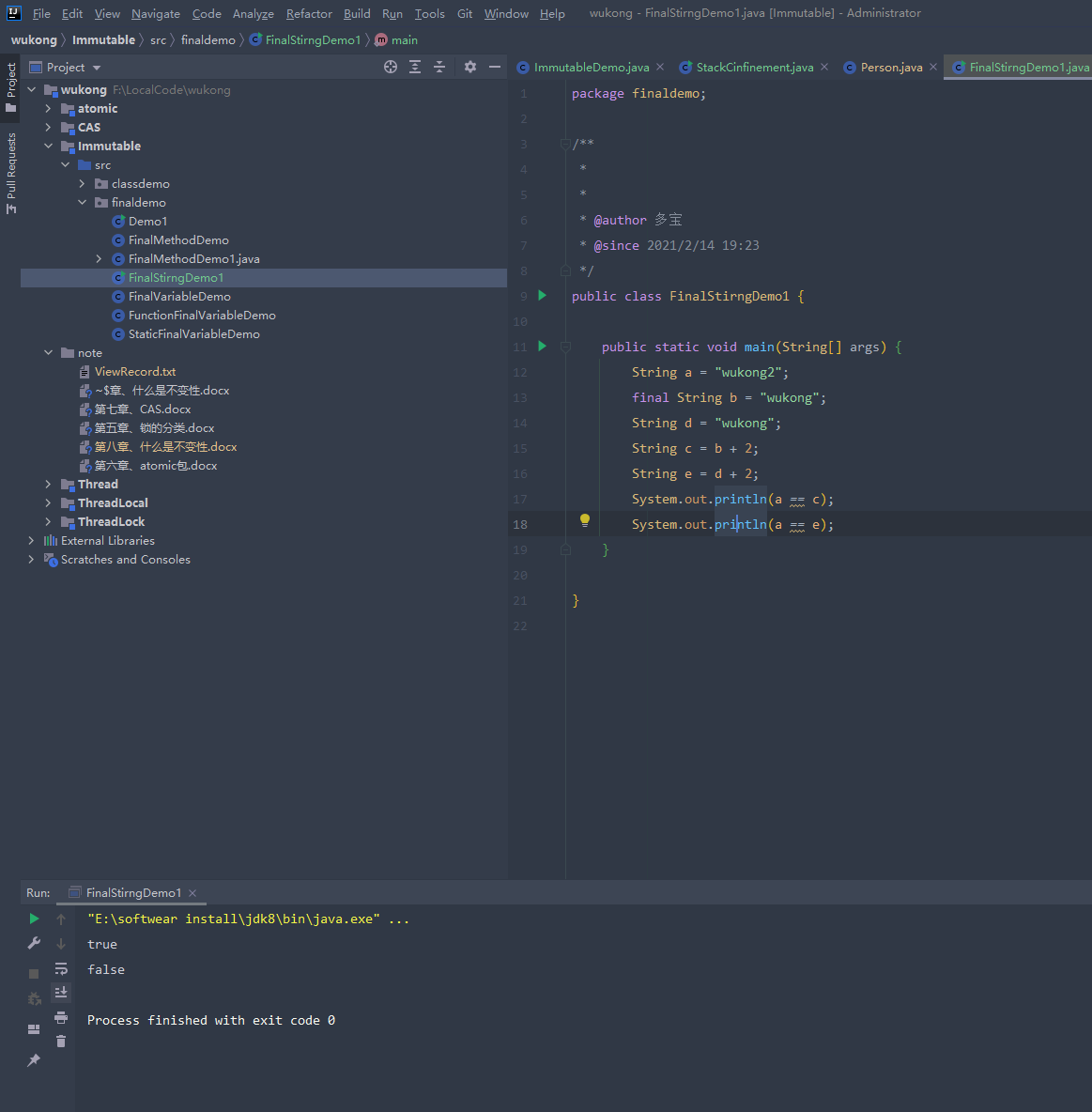
#### 2.代码演示

## 7.真假美猴王(面试题)

### 1.代码(实例1)

public static void main(String[] args) {  
 String a = "wukong2";  
 final String b = "wukong";  
 String d = "wukong";  
 String c = b + 2;  
 String e = d + 2;  
 System.out.println(a == c);  
 System.out.println(a == e);  
}

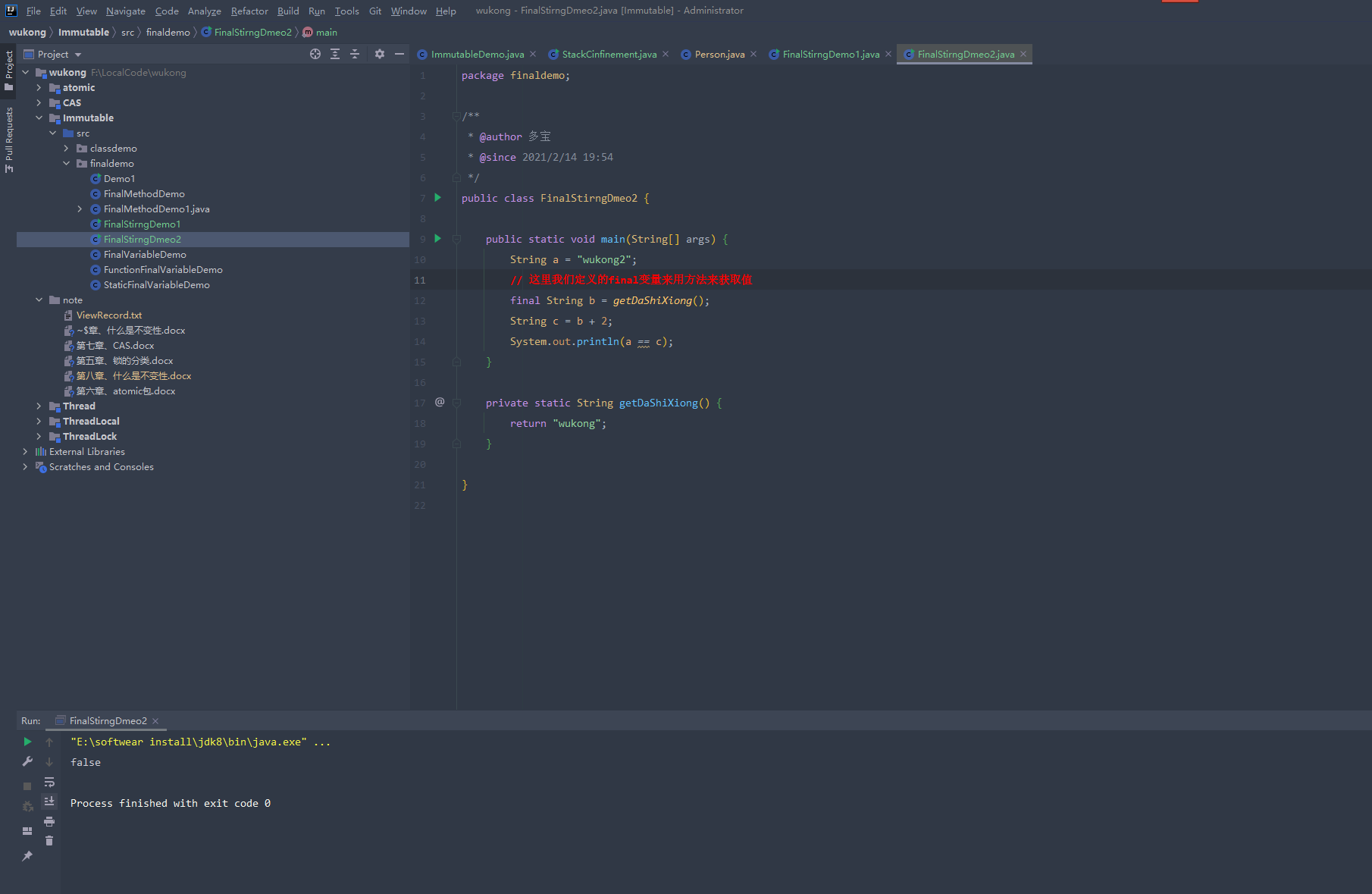
按理说打印的布尔值应该全部是true，但是我们来看下运行结果。



在这其中，我们一旦将我们的变量加了final修饰了之后，编译期间我们就知道他的准确值了，而且这个b是永远不会变化的，所以后期呢编译器会把它当做编译时期的常量来使用，用到这个final String b的时候，就直接访问这个常量，而不需要在编译的时间再确定，这和C语言中的宏替换有点像，那么像上面b被final修饰了，后面c用到b了，所以这个时候呢C就会发现b已经是wukong了，他会直接计算出那么c就是wukong2，所以呢在这个时候原来a已经是wukong2了，已经建立过wukong2了，那么对于c而言他没有必要再去新建一个对象了，那么这个时候呢他就会指向和a一样的地址，所以啊，他们是同一个常量，所以a和c之间等等号是true。

但是d的情况有所不同，d呢它最开始是指向常量池中的wukong，而不是wukong2，而且呢这个d没有被final修饰，所以编译器在使用d的时候，他也不会提前知道d具体的值，所以这个e的值啊，也要在运行的时候才能确定，所以呢，像这种运行时才能确定的e，他会在堆上生成wukong2，所以最终呢e指向的是堆上的，而我们a和c是指向的和常量池中的，所以呢a和e实际上就是不相等的。他们并不是同一个对象。

### 2.代码(示例2)



为什么和我们刚才的结论是不一样的呢？注意对比，我们刚才在声明final变量的时间是直接在 = 右边将值书写出来，这里我们是调用方法来给这个final变量进行赋值，因为这里面是不明确的，编译器在编译时间是不知道这个final定义的变量的初始值是多少的，只有在运行时间调用方法才能知道具体的值，所以这两个变量走到了运行时间对比，还是在堆上指向了两个不同的引用。

## 总结(以不变应万变：final关键字的不变性)

### 什么是不变性(Immutable)

### final的作用

对于final而言早起呢我们可以利用它来提升效率，但是近期我们是已经没有必要从效率的角度去使用它了，更多的是线程安全以及是提示其他的编程小伙伴这个对象是不可变得，从这个角度去考虑去使用final关键字。

### 3、3种用法：修饰变量、方法、类

我们尤其在使用final修饰变量的时候我们还分为三种变量，

1. final instance variable(类中的final属性)
2. Final static variable(类中的static final属性)
3. Final local variable(方法中的final变量)

他们各自都有不同的赋值时机，但是总结出来就是一旦被赋值，那么他就不可以再变化了。

final修饰方法

1. 构造方法不允许final修饰
2. 不可被重写，也就是不能被override，即便是子类有同样的名字的方法，那也不是override，这个和static方法是一个道理
3. 引申：static方法不能被重写

final修饰类

1. 不可被继承
2. 例如典型的String类就是final的，我们从没见过哪个类是继承String类的。

### 4、final的注意点

1.final修饰对象的时候，只是对象的引用不可变，而对象本身的属性是可以变化的

2.final使用原则，良好的编程习惯

### 5.不可变和final的关系

并不是简单地用final修饰了他就不可变，而是说要保证这个对象确实在完全构造之后永远内部都不会发生任何变化的，那么他才是不可变的，我们还举了一个hashSet的例子，虽然hashSet本身是一个对象，但是呢由于它在构造之后，不会再有任何的变化，并且外部的类也无法修改它，所以他整个对象呢都是不可变的，他满足这个要求。

### 把变量写在线程内部 -- 栈封闭

他的意思是把变量写在方法中，那由于不同的线程之间，对此部分的内容是不做共享的，所以他也是线程安全的

### 真假美猴王

对于final修饰的String而言。我们在编译的时候如果发现这个变量是final的，就会把它当做常量来使用，如果没有被final修饰呢，那么编译器也没有办法来确定他的值。