# 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。

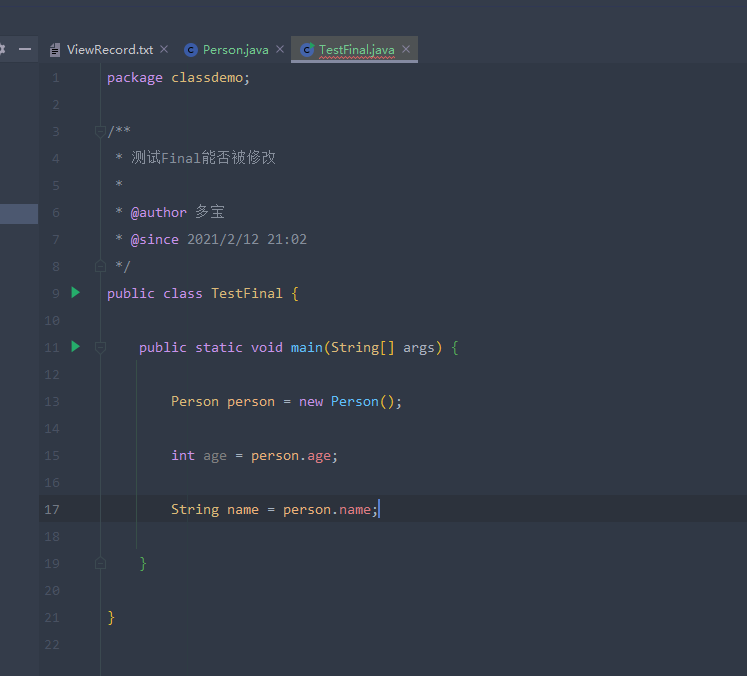
1. person对象，age和name都不能再变

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？

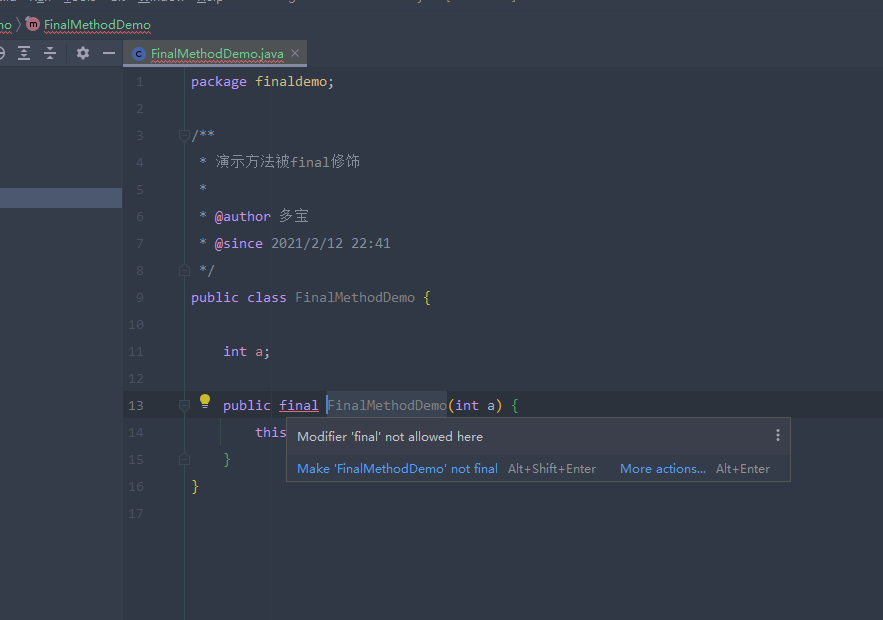
其实是很有意义的，我们之所以JAVA会设计成这样，我们站在他的角度去考虑一下，如果他不给我们这么严格的要求，那么会怎么样呢？

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！

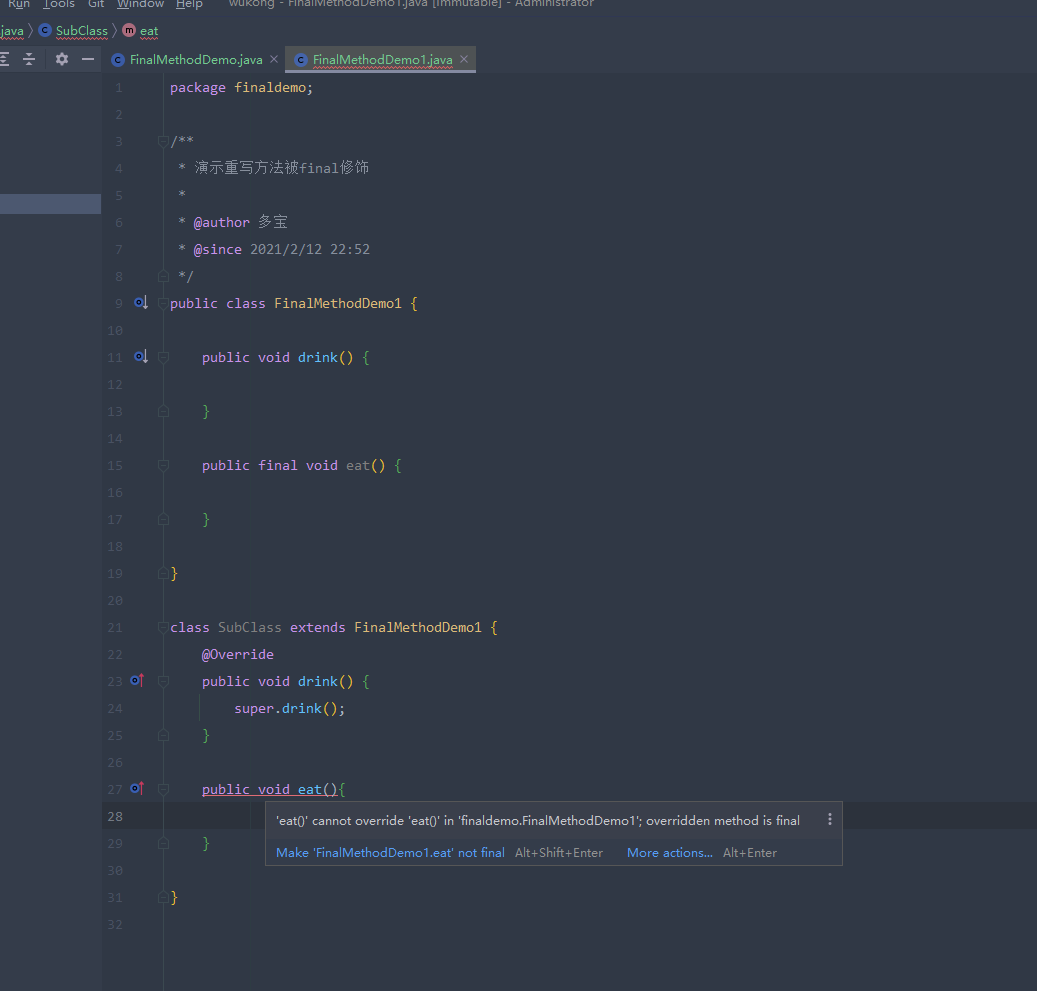
final的原则是连这个从null修改过来的过程都不允许有，只要你一诞生那么就再也变不了了，你可以诞生为null，那是Null变不了其他的值

### final修饰方法

#### 构造方法不允许final修饰



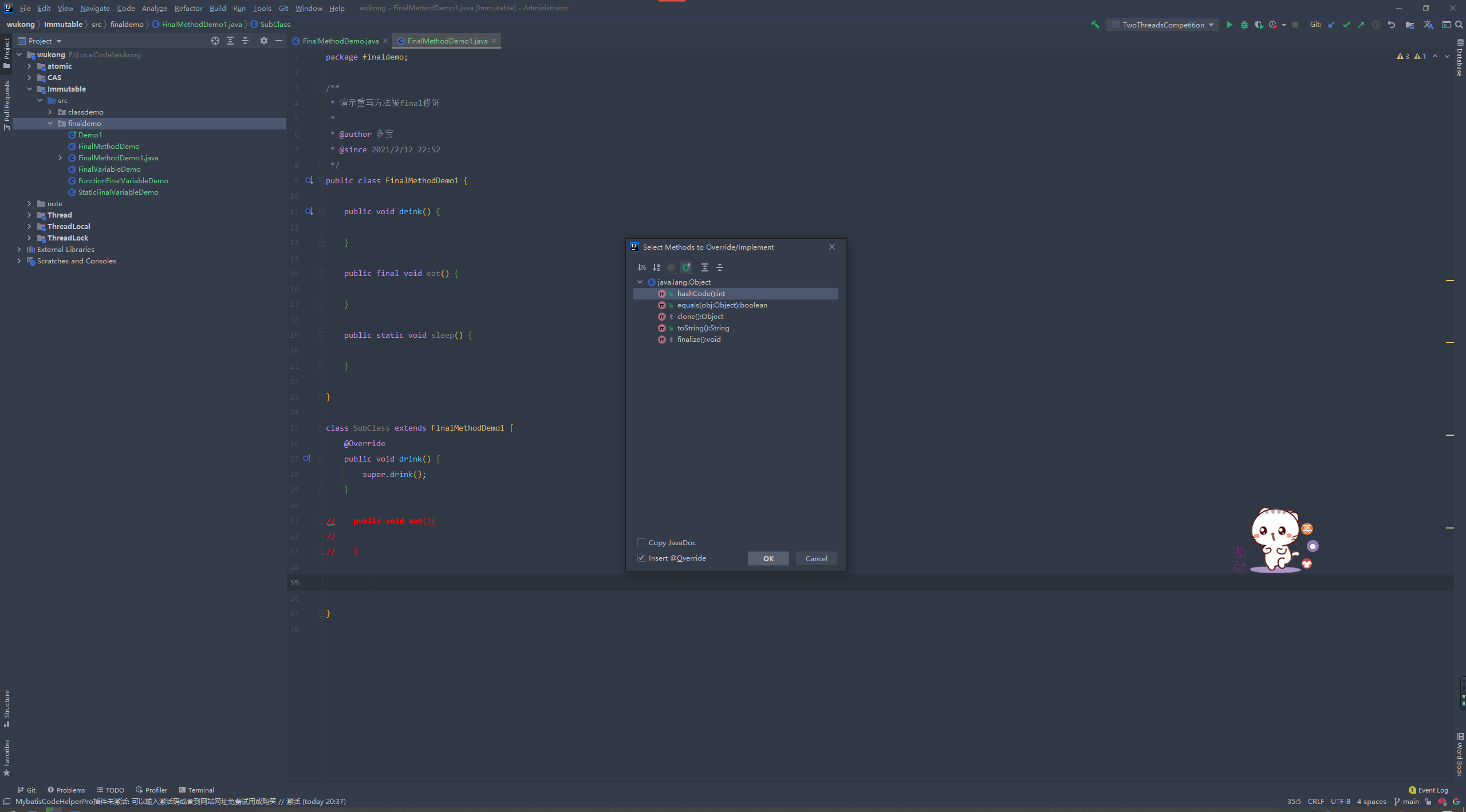
#### 不可被重写，也就是不能被override，即便是子类有同样名字的方法，那也不是override，这个和static方法是一个道理



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。

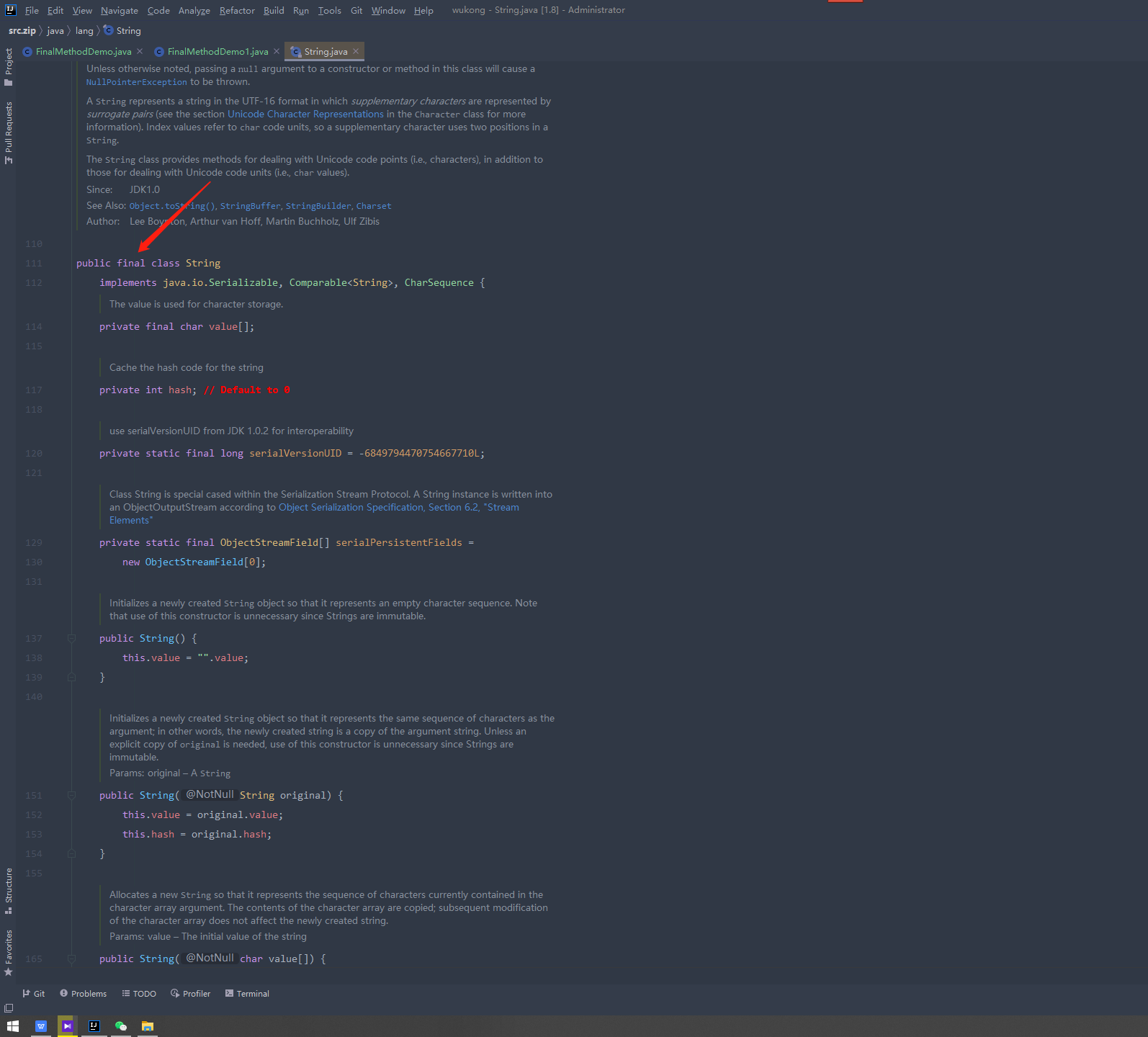
这是为什么呢？

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