# 1.设计模式

## 1.1 简单工厂模式

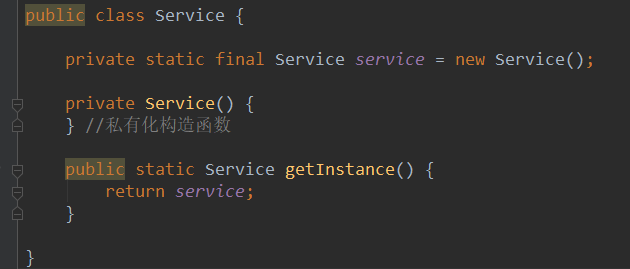
## 1.2 方法工厂模式

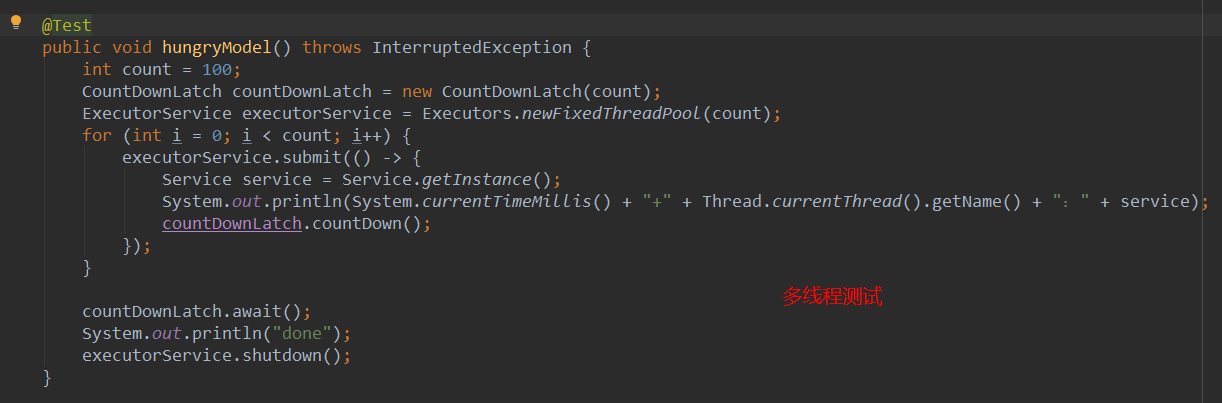
## 1.3 抽象工厂模式

## 1.4 单例模式

### 1.4.1 饿汉式

饿汉式的单例对象是线程安全的。

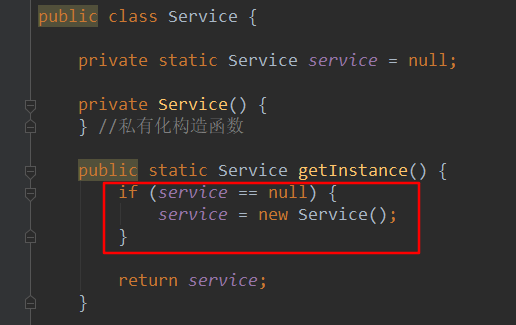


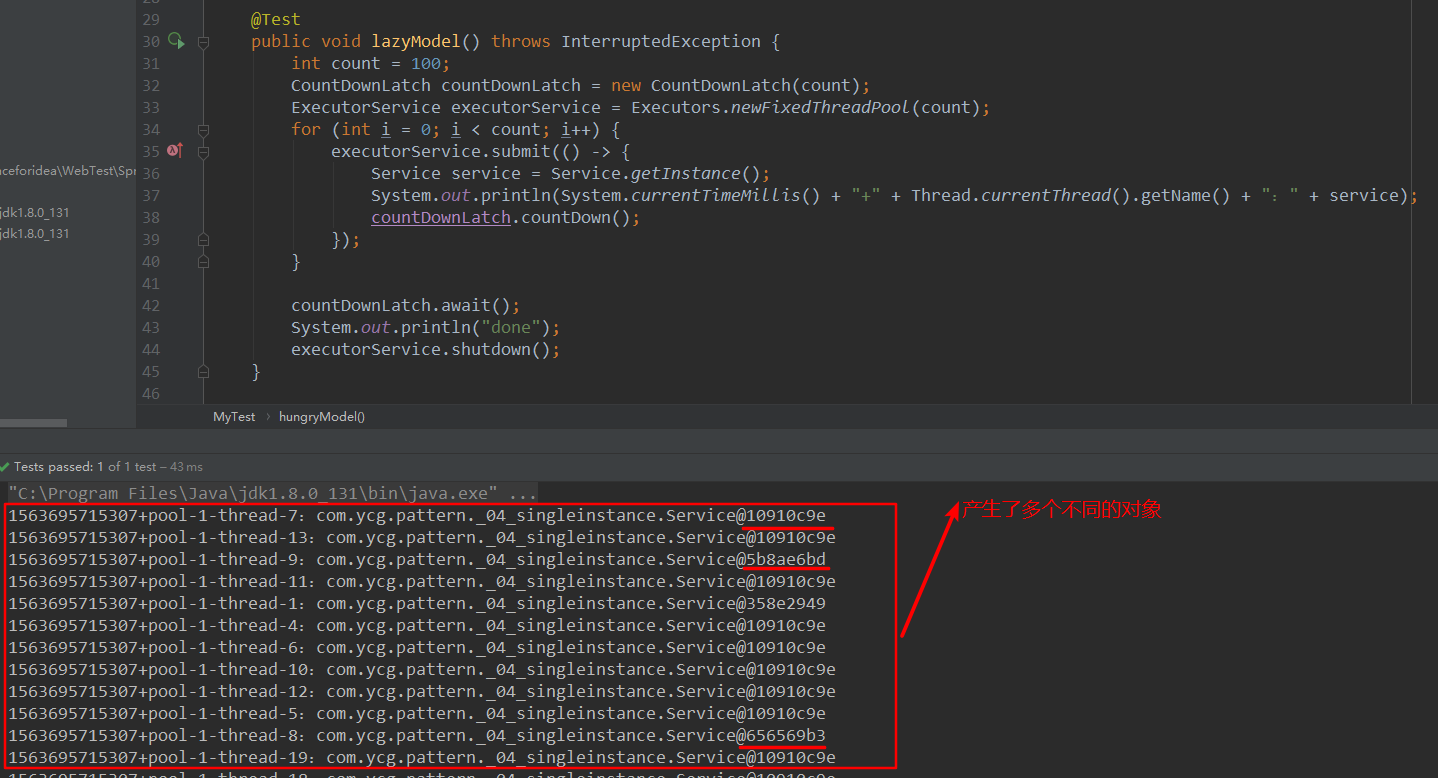


### 1.4.2 懒汉式

#### 1.4.2.1 懒汉式的错误示范

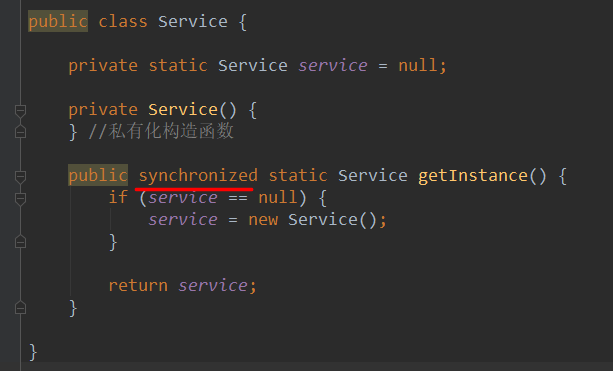
在默认情况下懒汉式是有线程安全隐患的，即违背单例模式只产生一个对象的原则，可能不同的线程会拿到不同的对象：



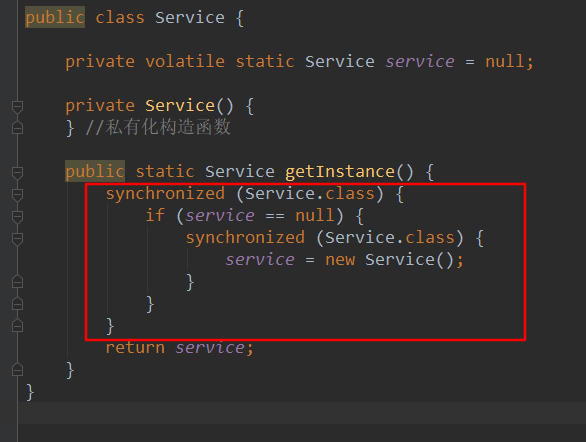


#### 1.4.2.2 懒汉式的正确示范

方式一：同步方法，性能最低：

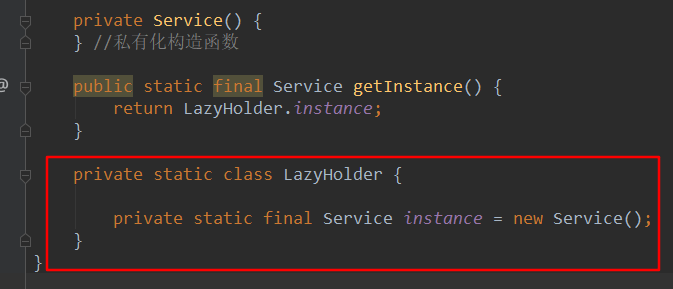


方式二：双重检查锁：

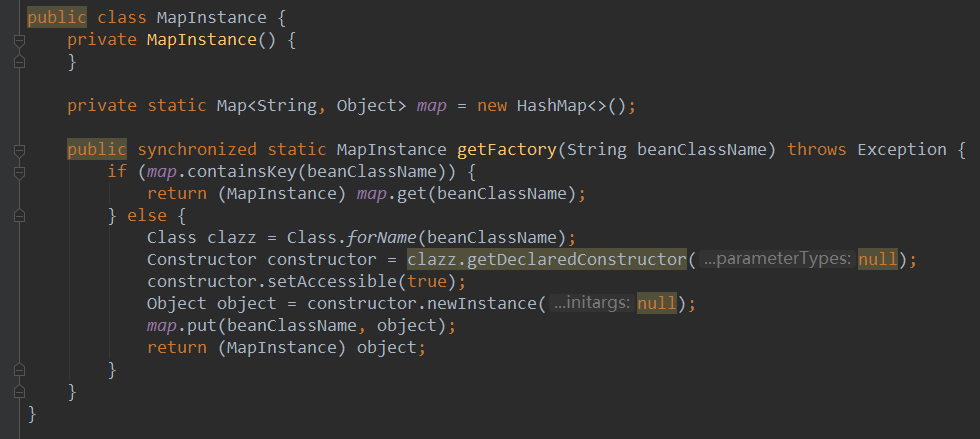


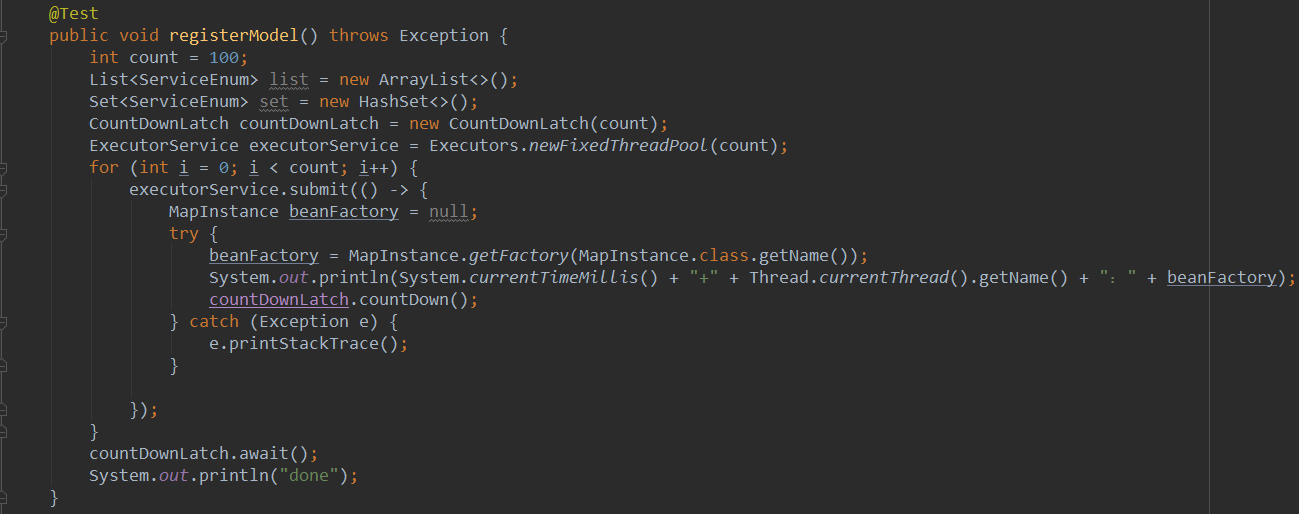
方法三：使用内部类模式：

因为内部类只有在被使用时才会加载，所以也算是懒汉式。



### 1.4.3 注册式单例



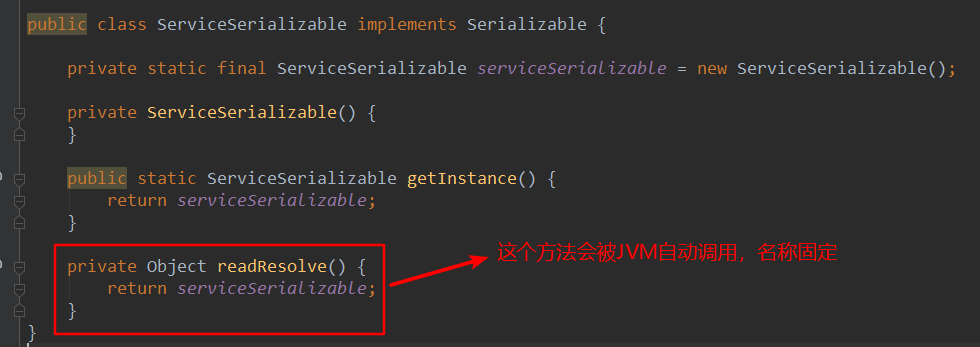


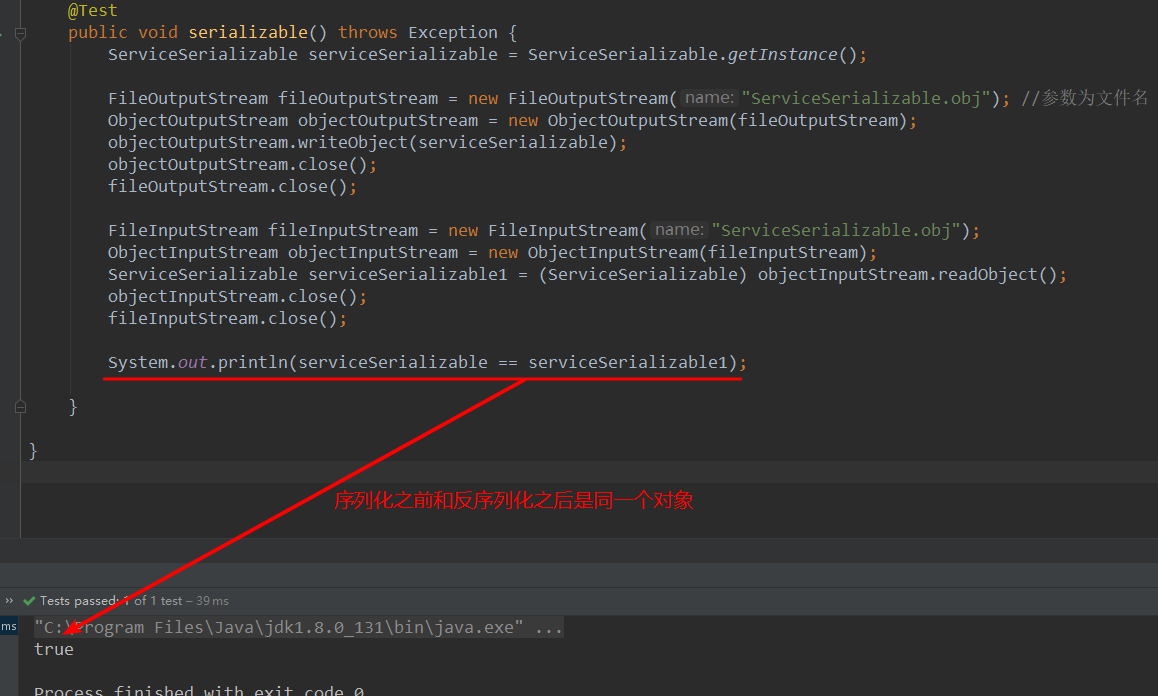
### 1.4.4 枚举式单例

枚举的每一个实例都是线程安全的。

### 1.4.5 序列化式单例

就是为了保证序列化之前的对象和反序列化之后的对象是同一个对象。

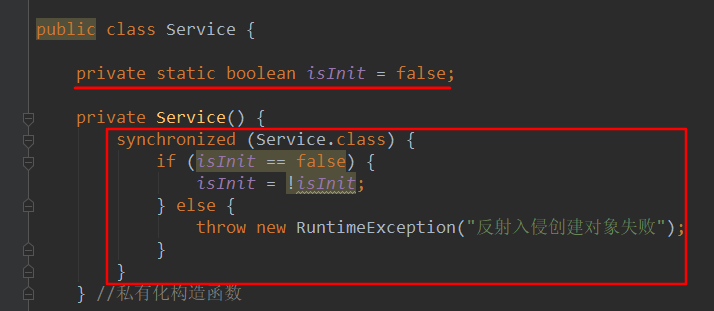


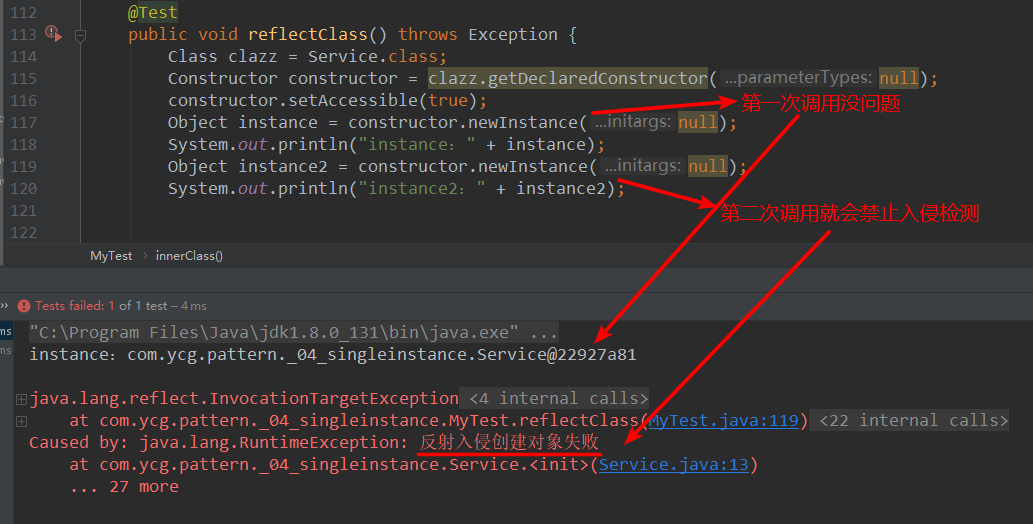


### 1.4.6 反射式单例

防止反射侵入

让反射也只能创建一个实例对象；

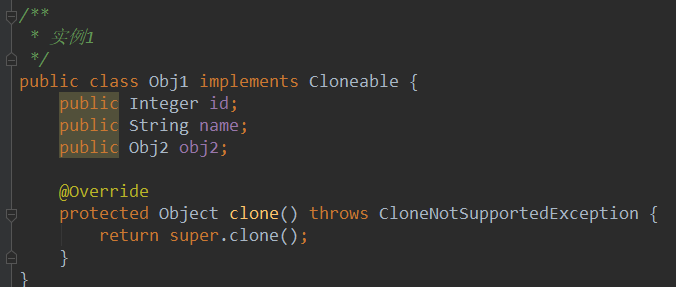


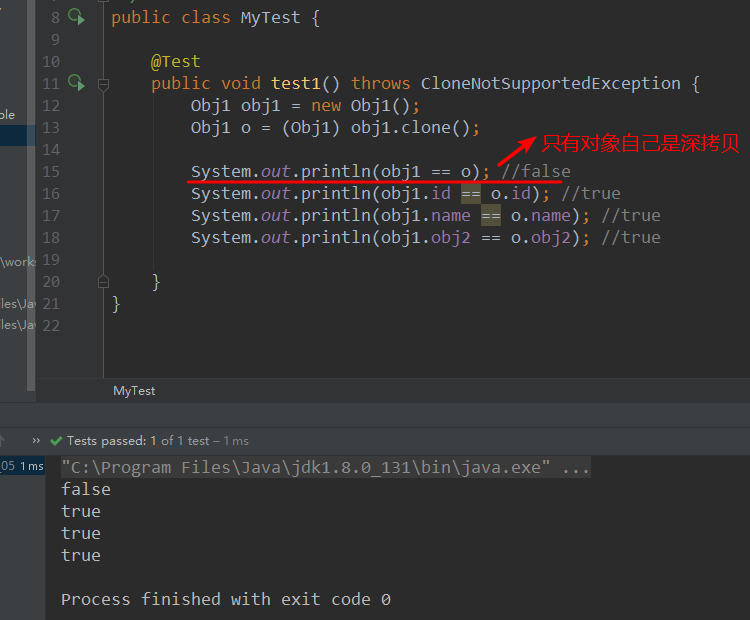


## 1.5 原型模式

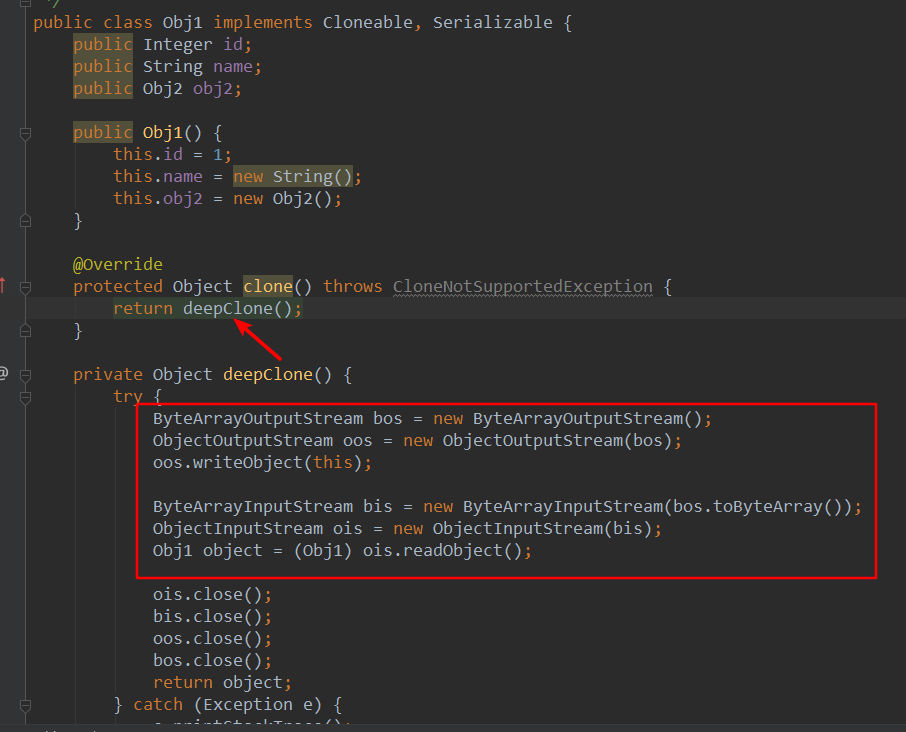
原型模式的目的就是要以已有的对象深拷贝出一个全新的对象。

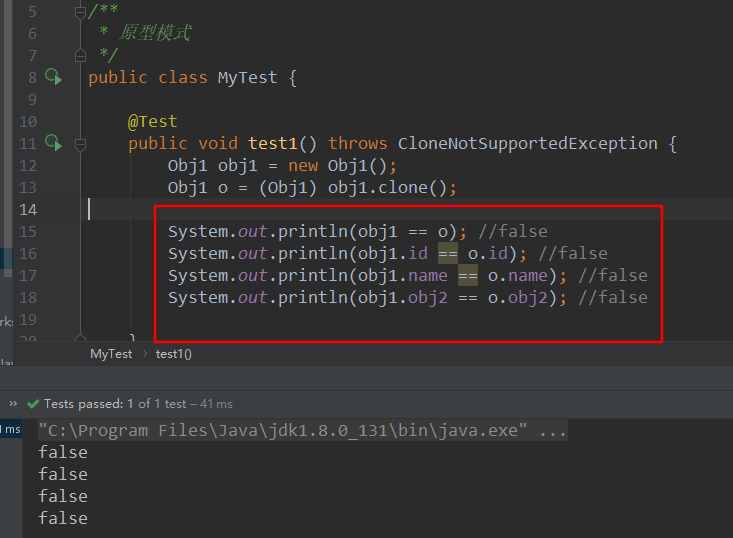
JDK提供的默认clone()只对对象本身是深拷贝，而对象里面的属性还是浅拷贝（值类型和引用类型都是浅拷贝，如下图所示）：



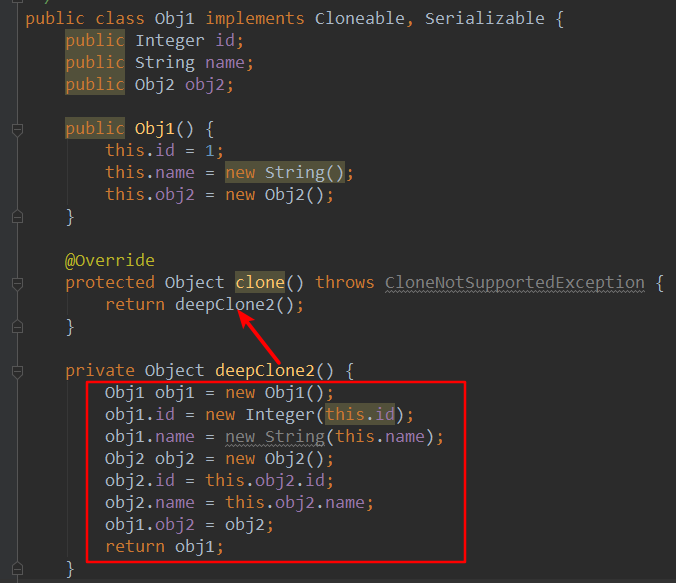


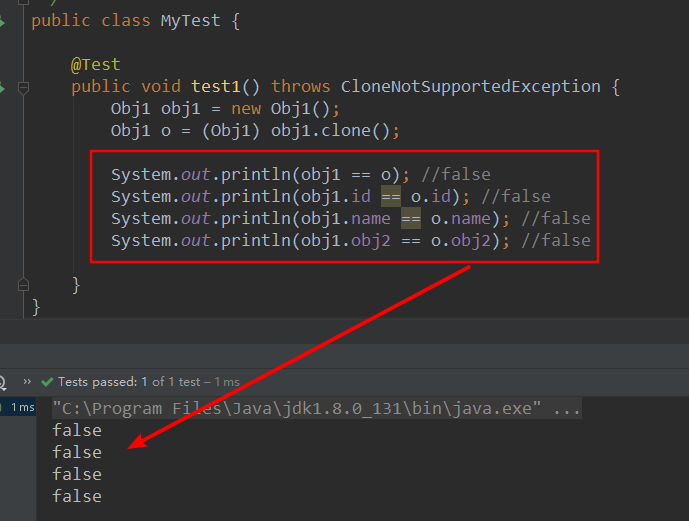
### 1.5.1 序列化方式实现深拷贝





### 1.5.2 循环属性赋值方式实现深拷贝





### 1.5.3 反射方式实现深拷贝

## 1.6 代理模式

### 1.6.1 静态代理

静态代理就是，在代理类中创建目标类的对象调用目标类的方法，在调用目标类方法的前后插入自己的方法（即代理类的方法），客户端请求时直接请求代理类。

在真正的开发框架中一般不用静态代理，因为使用静态代理就要为每个目标类创建一共对应的代理类。

静态代理的实现方式有接口聚合和父子继承两种方式，但是这两种方式都会产生大量的类造成类爆炸，父子继承方式又比接口聚合效果更差。

### 1.6.2 动态代理

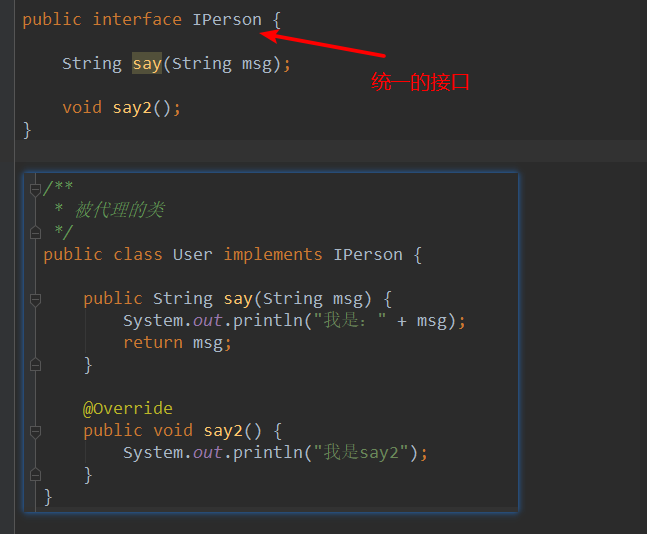
动态代理时AOP思想的一种具体的技术实现，底层采用反射实现字节码的重组。

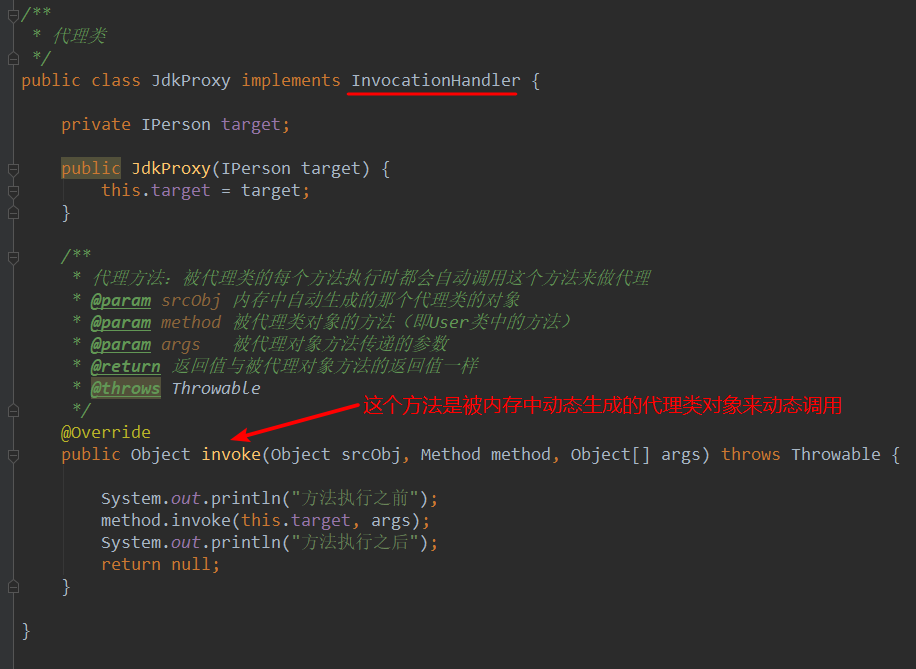
jkd动态代理和cglib动态代理都是运行期间自动在内存中自动生成一个代理类，把被代理对象的方法调用转化为内存中代理类的方法调用。

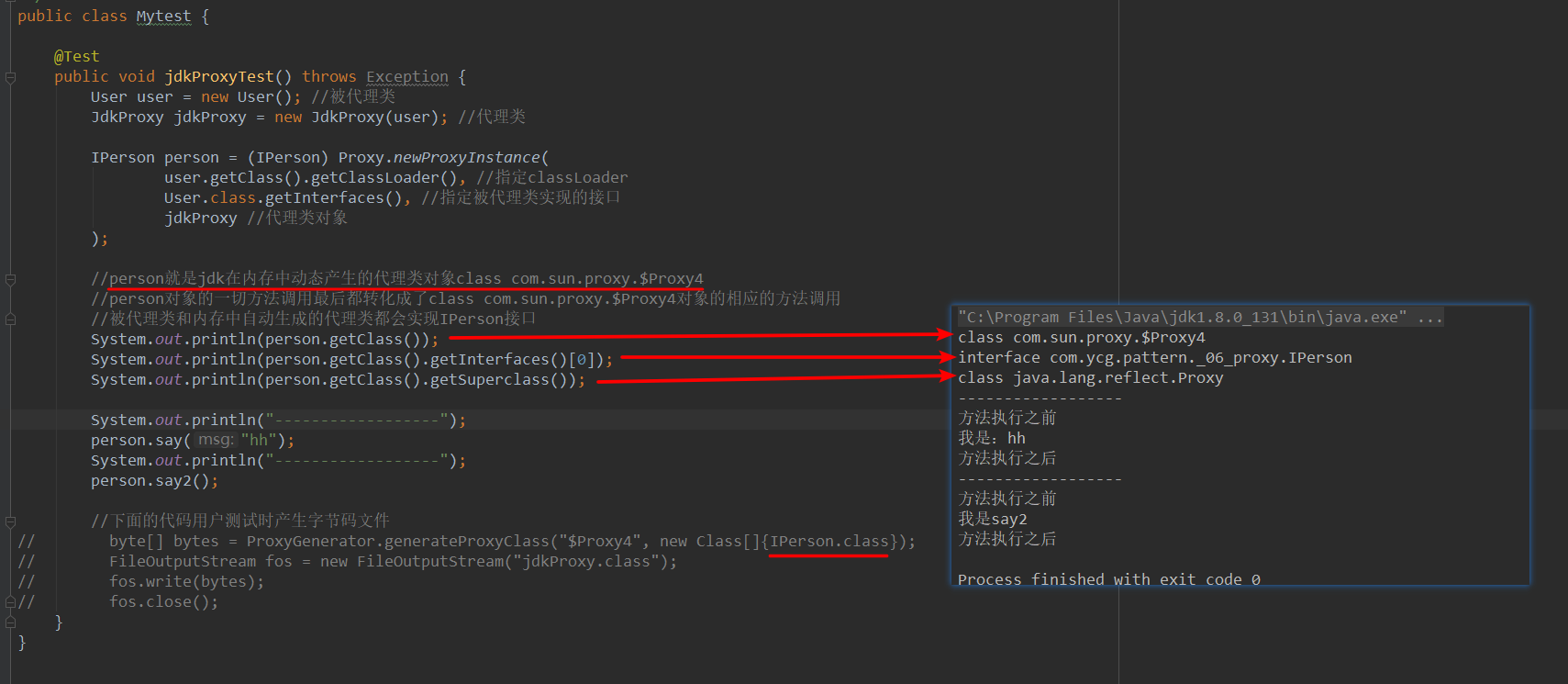
#### 16.2.1 JDK动态代理

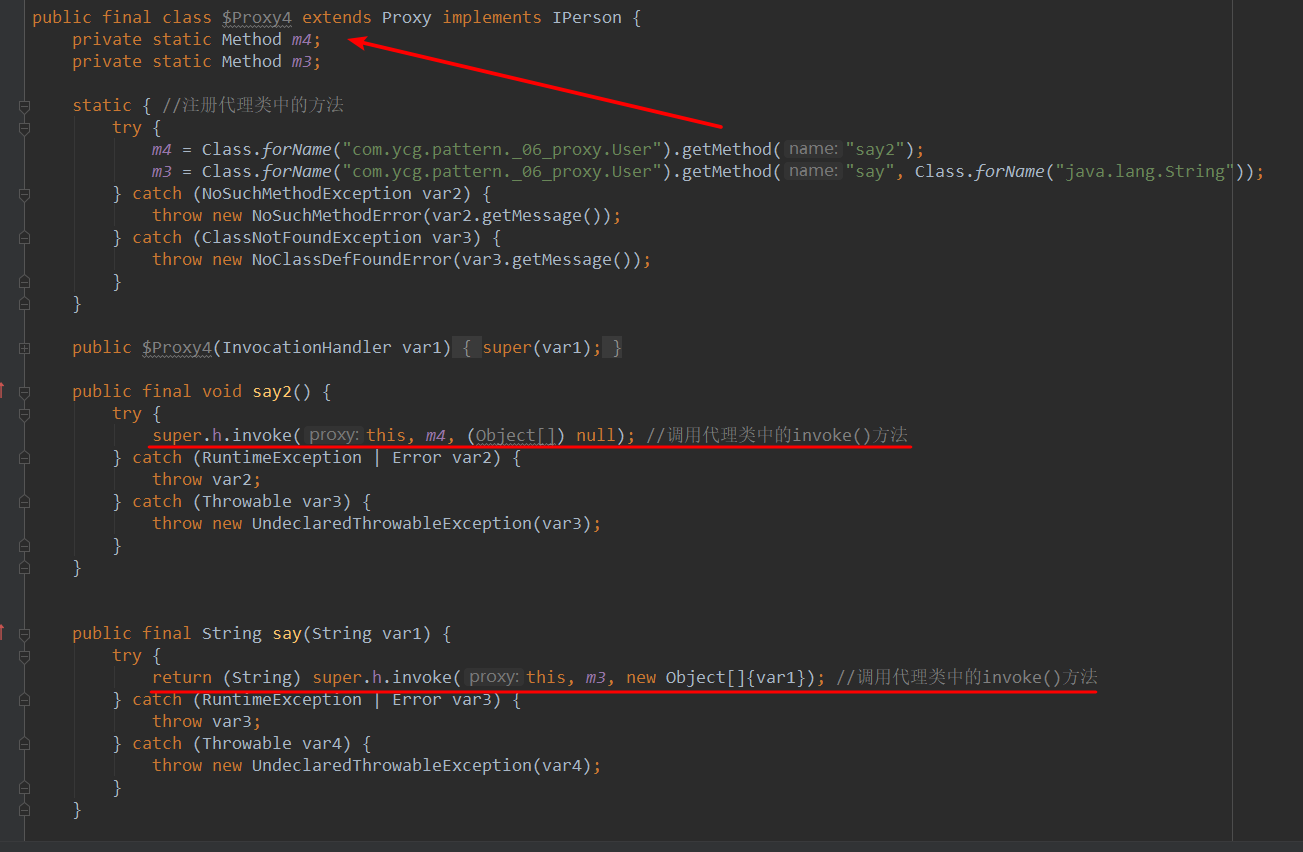
Java通过反射把拼接的代理类生成byte[]数组后，再调用native方法把byte[]转为Class对象。

需要定义一个接口，被代理类需要实现这个接口，而最终在内存中动态生成的代理类也会自动实现这个接口。







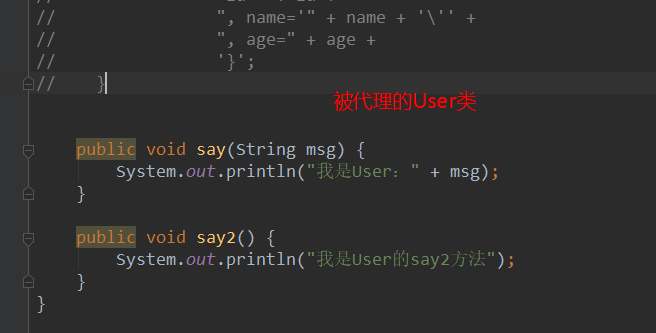


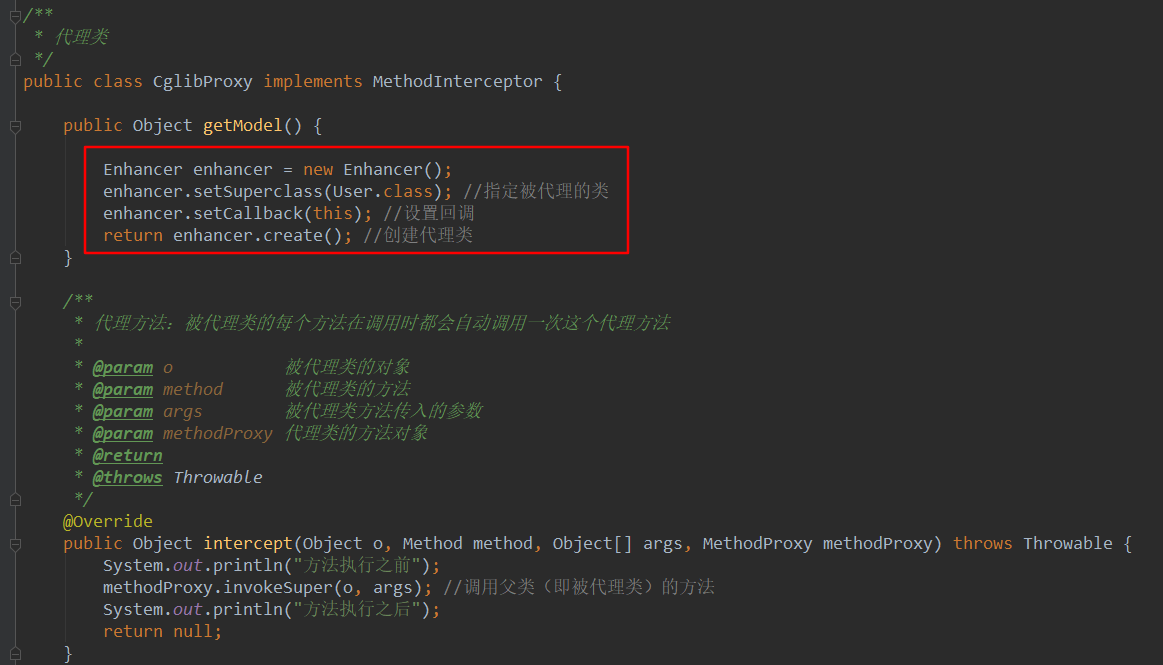
#### 1.6.2.2 cglib动态代理

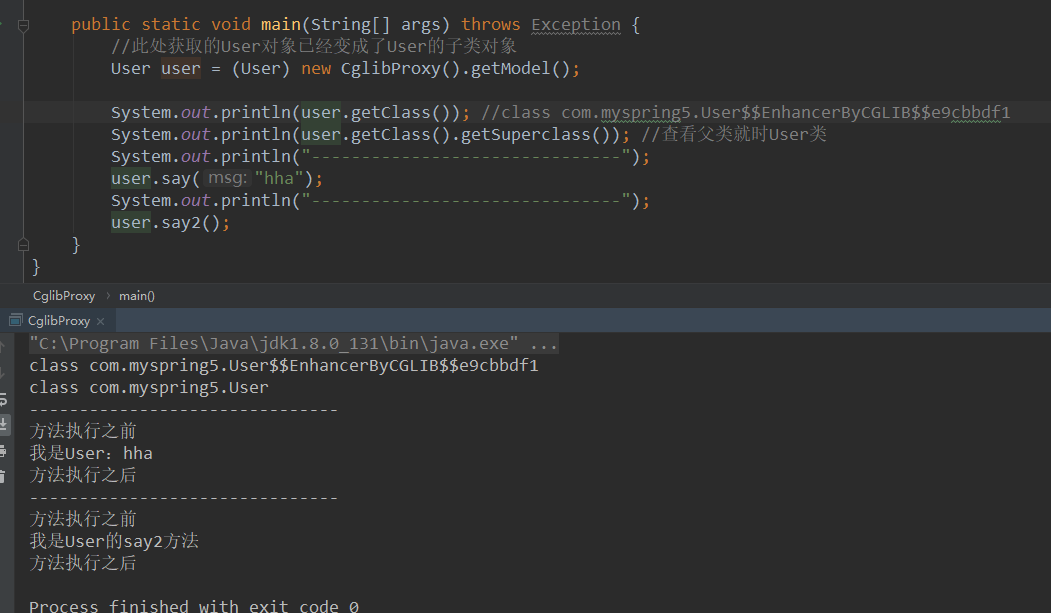
cglib动态代理采用的继承实现的方式，即动态创建被代理类的一个子类作为代理类来操作。

cglib通过ASM操作字节码创建代理类。

使用cglib代理需要引入cglib的jar包。

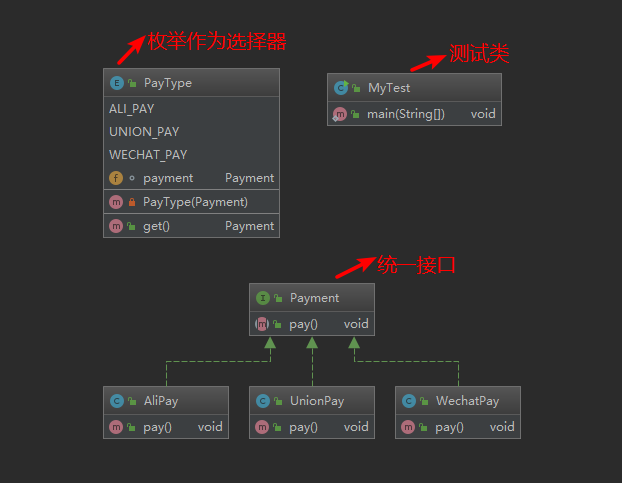


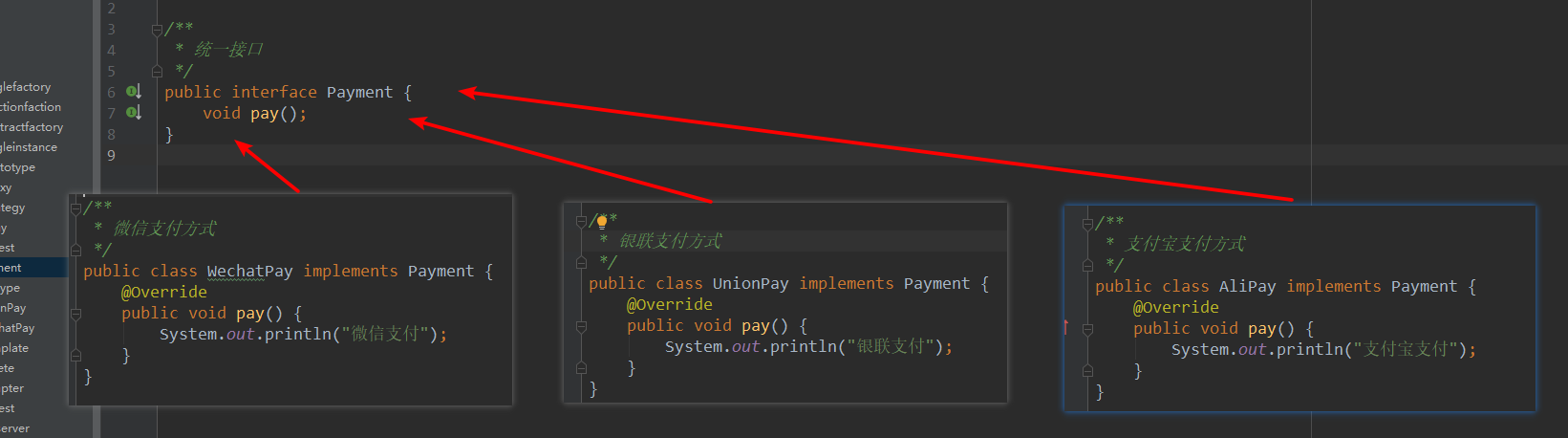


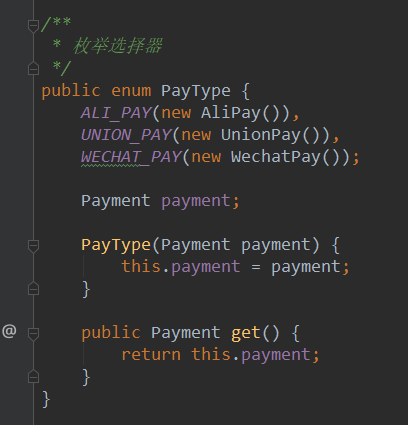


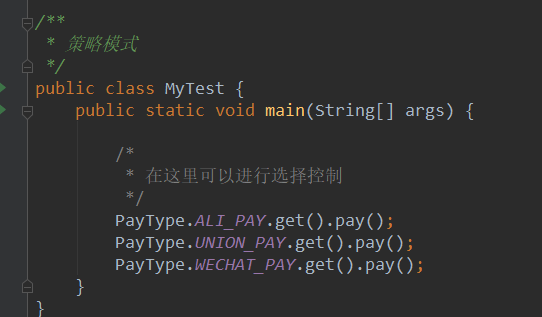
## 1.7 策略模式

典型场景：网上支付时支付方式的选择。









## 1.8 模板模式

## 1.9 委派模式/委托模式

### 1.9.1 委派模式和代理模式的区别

委派模式不属于23种经典模式，是静态代理模式和策略模式的特殊组合。

代理模式侧重的是过程，

策略模式侧重的是扩展；

委派模式侧重的是结果；

1. 委派模式：第一种属于全权代理，代理被代理对象的所有功能，；

第二种属于任务分发：Nginx、SpringMVC的dispatcher、Android的Handler消息分发、Android的进程间通信等；

1. 代理模式：代理对象只代理被代理对象的部分功能，最终还是由被代理对象做决定；

## 1.10 适配器模式

## 1.11 装饰器模式/包装器模式

典型实现：jdk的IO流类。

目的：包装类对被包装类做功能加强，包装类持有被包装类的引用。

一般Wrapper和Decorator结尾的类都属于装饰器模式设计。

## 1.12 观察者模式/发布-订阅/事件驱动

使用场景：事件监听器，日志记录，短信通知；

元素：事件源（发布者），事件，事件类型，观察者（订阅者），事件注册中心；

初始化时，观察者把自己要观察的事件类型以及自己的回调方法注册到注册中心，发布者在执行对应的方法时会自动触发事件注册中心注册的方法。

# 2.数据结构

# 3.算法设计