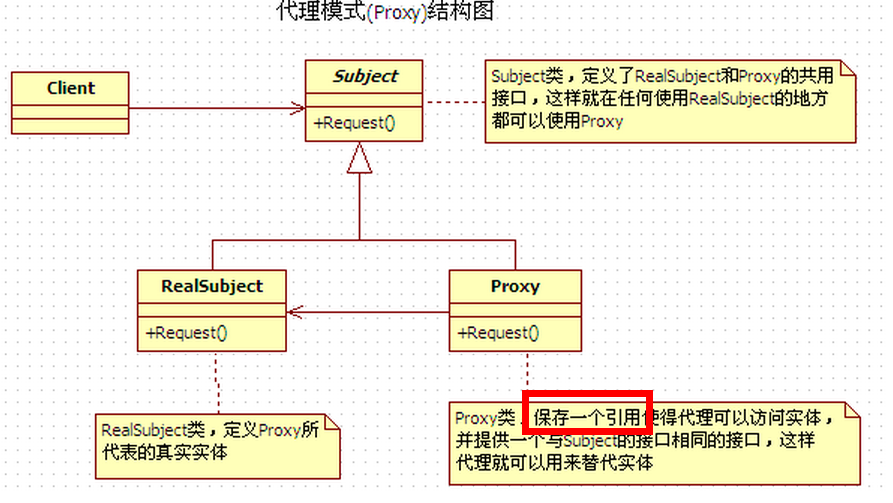
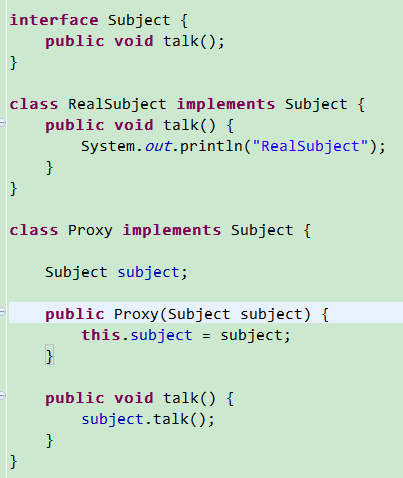
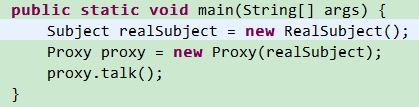
1. 代理模式：为其他对象提供一种代理以控制对这个对象的访问。为真实对象提供一个代理, 从而控制对真实对象的访问
2. 类图



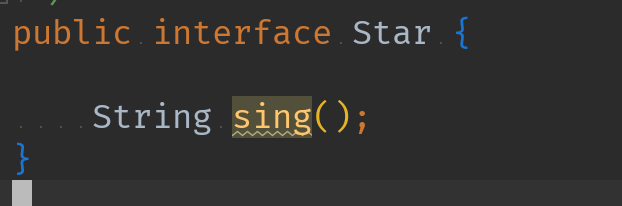
* 1. Subject类(抽象角色)：定义了RealSubject和Proxy的共用接口，这样就可以在任何使用RealSubject的地方都可以用Proxy。
  2. RealSubject类(真实角色)：定义Proxy所代表的真实实体。
  3. Proxy类(代理角色)：保存一个引用使得代理可以访问实体，并提供一个与Subject的接口相同的接口，这样代理就可以用来替代实体。

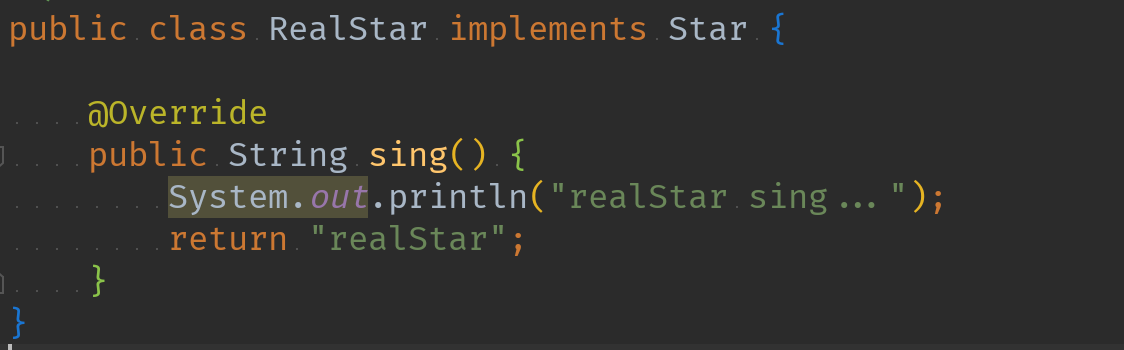
1. 代理模式应用场合(场景)
   1. 远程代理，也就是为一个对象在不同的地址空间提供局部代表，这样可以隐藏一个对象存在于不同地址空间的事实。
   2. 虚拟代理，是根据需要创建开销很大的对象，通过它来存放实例化需要很长时间的真实对象。
   3. 安全(保护)代理，用来控制真实对象访问时的权限。
   4. 智能引用，是指当调用真实对象时，代理处理另外一些事
2. 分类
   1. 静态代理(静态定义代理类)
   2. 动态代理(动态生成代理类), 可以更加灵活和统一处理多个方法(每个方法调用都需要经过代理)
      1. JDK自带动态代理
      2. Javaassist字节码操作库实现
      3. CGLIB
      4. ASM(底层使用指令, 维护性差)
3. 简单例子—静态代理



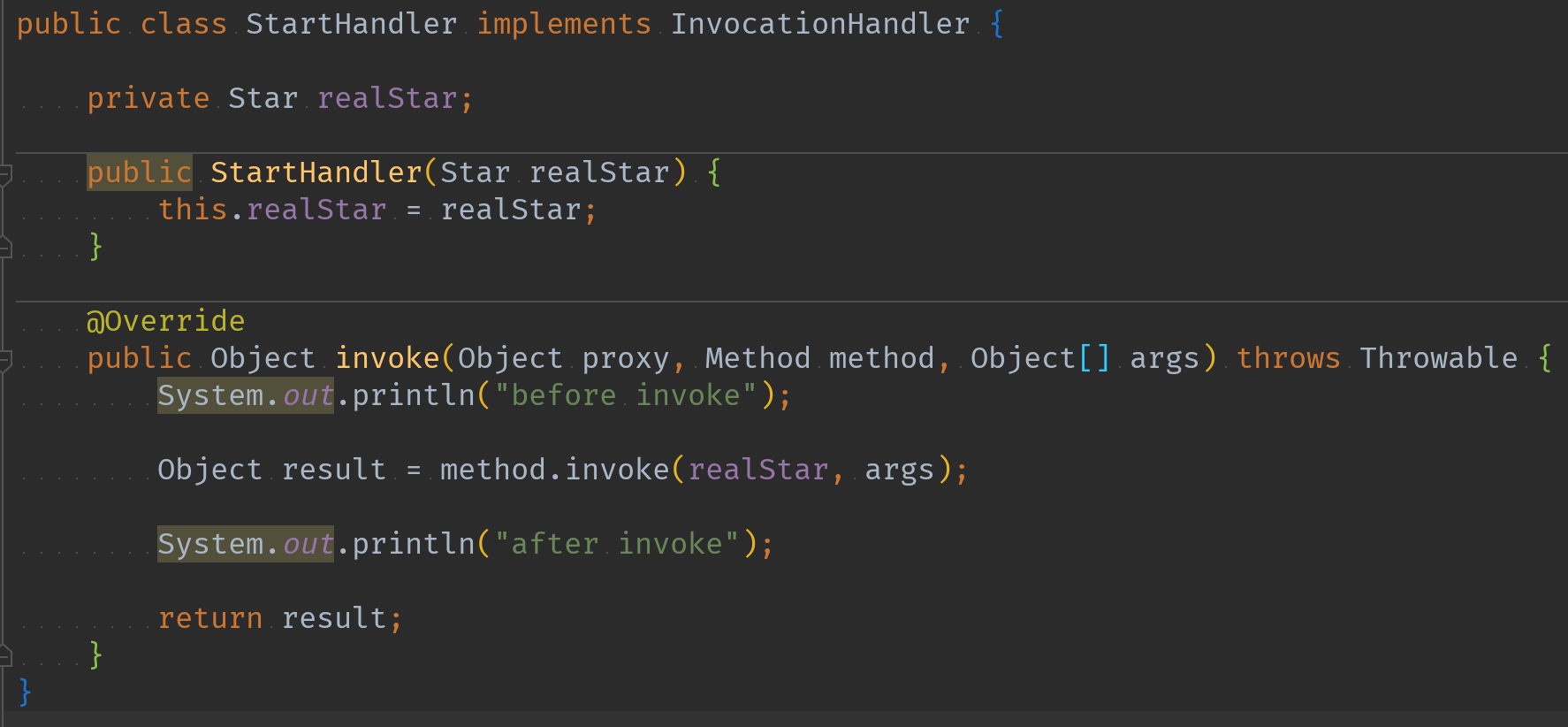
测试

1. 例子—动态代理(JDK自带)

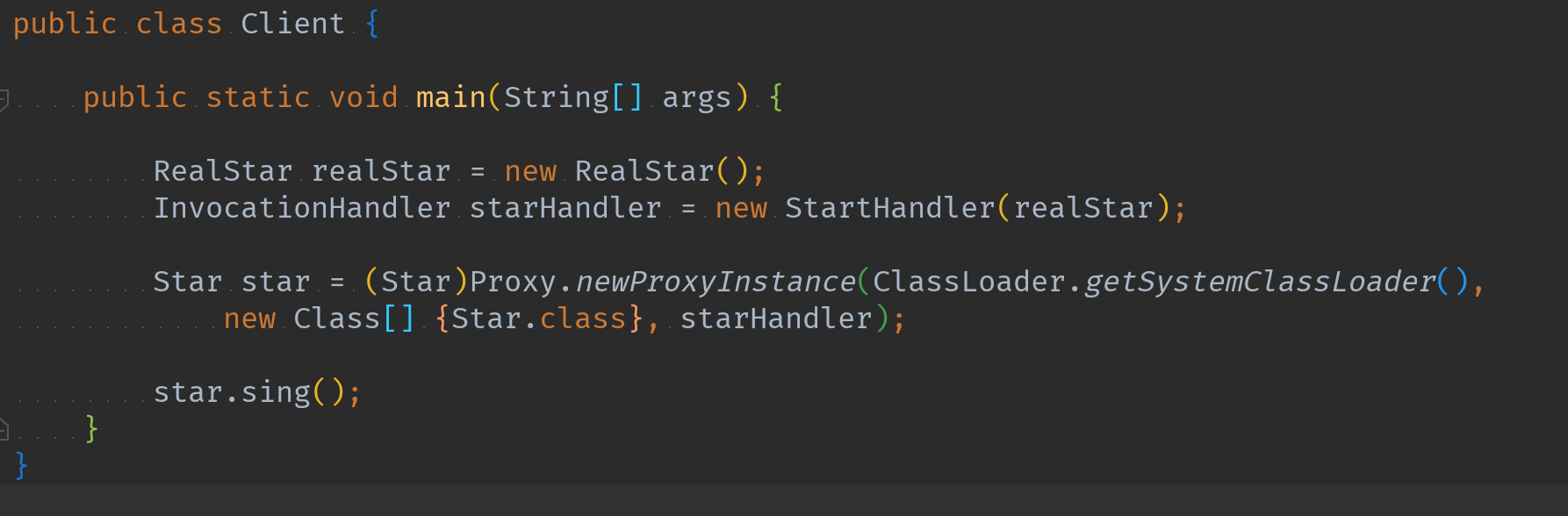




* 1. 处理器接口java.lang.reflect.InvocationHandler, 通过invoke方法实现对真实角色的代理访问



* 1. 动态生成代理类和对象java.lang.reflect.Proxy



1. 开发框架应用场景
   1. Struts2 拦截器实现
   2. DB连接池关闭处理
   3. MyBatis实现拦截器插件
   4. AspectJ的实现
   5. Spring中AOP的实现
      1. 日志拦截
      2. 声明式事务处理
   6. RMI远程方法调用