结构型模式

1. 类和对象按照某种布局组成更大的结构。

类结构型模式：采用继承机制。

对象结构型模式：采用对象的组合和聚合。

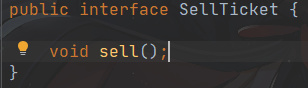
1. 代理模式

需要给某对象提供一个代理以控制对该对象的访问。

根据代理类生成时机不同：静态代理（编译期生成），动态代理。

组成：

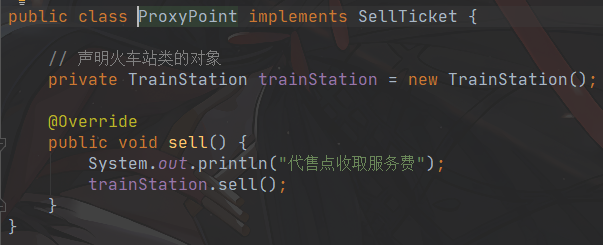
1. 抽象主题类：真实主题和代理对象实现的业务方法。



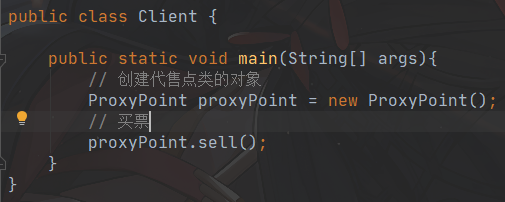
1. 真实主题类：具体业务。



1. 代理类：对真实主题的引用。可以访问、控制或扩展真实主题的功能（在这里是输出）。



1. 访问类：访问代理对象而非目标对象。



优点：

1. 在客户端和目标对象之间起中介和保护目标对象的作用；
2. 代理对象可以扩展目标对象的功能；
3. 将客户端和目标对象分离，降低耦合度。

缺点：增加了系统的复杂度。

适用场景：

1. 远程代理。
2. 防火墙代理（VPN）。
3. 保护代理，给不同的用户提供不同级别的使用权限。
4. 适配器模式

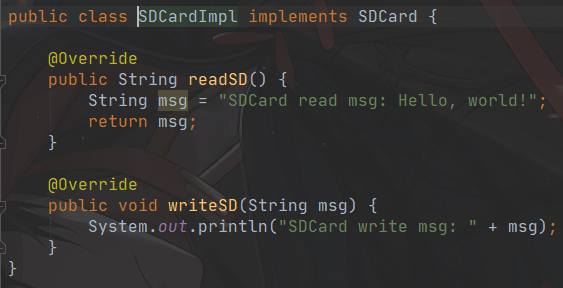
将一个类的接口转换成客户希望的另一个接口，使得原本因为接口不兼容不能一起工作的类能一起工作。

①适配器类模式组成：

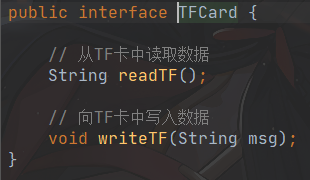
1. 目标接口：当前系统业务希望的接口；



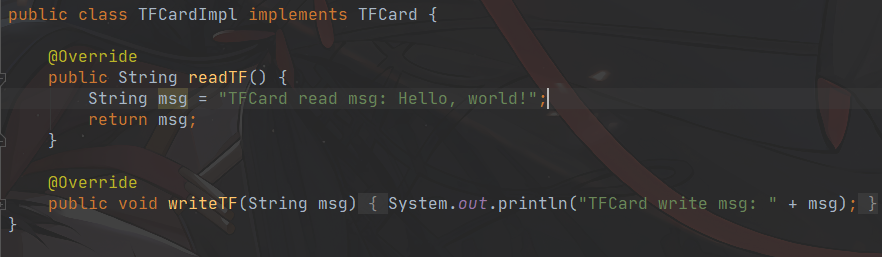
目标接口的具体实现：



1. 适配者接口：现存组件库中的组件接口；

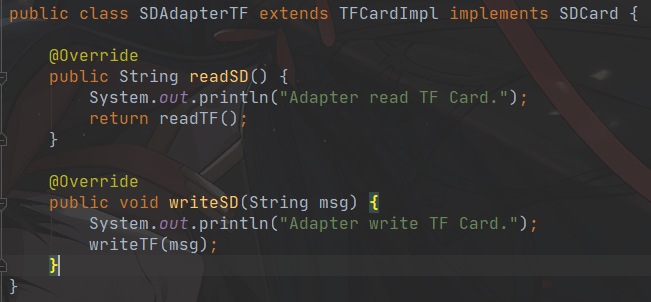


具体实现：

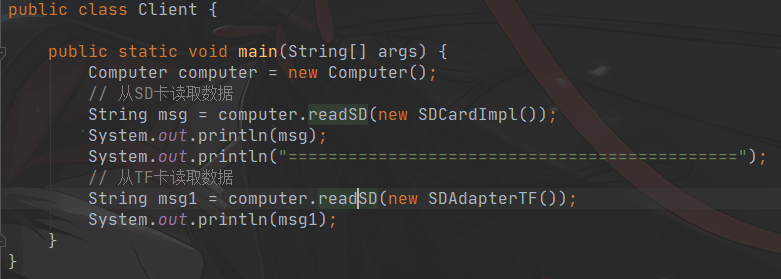


1. 适配器类：转换器。

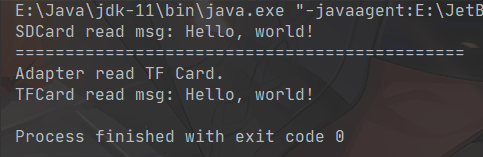
继承现有组件的接口，实现目标接口。



1. 客户类



输出结果：



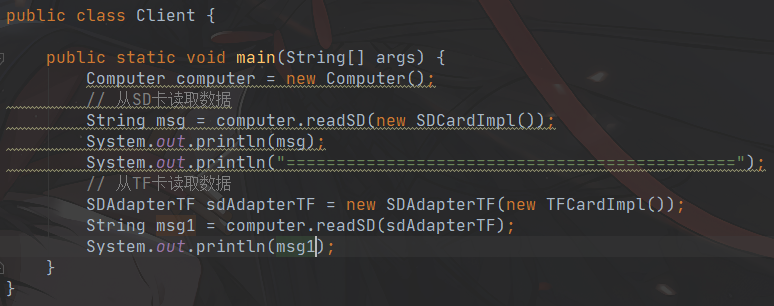
要求系统必须拥有一个目标接口，提供给适配器类实现。

不满足合成复用原则：尽可能使用对象组合而不是继承来达成复用的目的。

②对象适配器模式组成

适配器组合现有的组件而非继承。





满足合成复用原则。

当不存在目标接口时，也可以直接继承实现类。

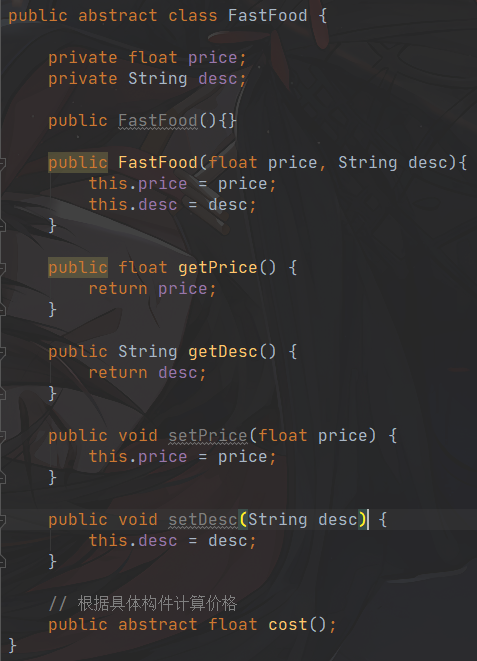
③应用场景

1. 原有系统的接口和新系统的接口不一致；
2. 使用第三方提供的组件，但是组件的接口定义和我们系统要求的接口不同。
3. 装饰者模式

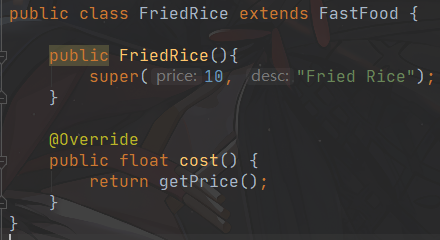
在不改变现有对象结构的情况下，动态地给该对象增加一些职责（功能）的模式。

①组成

1. 抽象构件：抽象接口



1. 具体构件：实现抽象构件，通过装饰角色增加一些职责



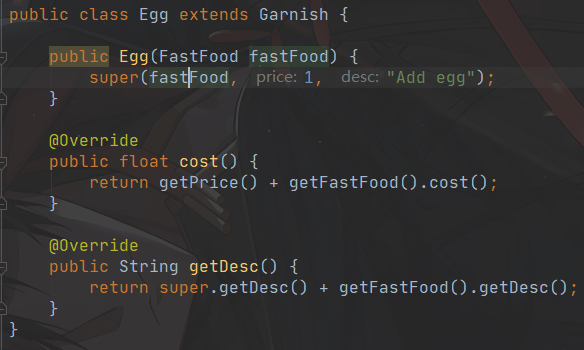
1. 抽象装饰：继承或者实现抽象构件角色，包含具体构件的实例，通过其子类扩展具体构件的功能。



继承抽象构件角色：在添加新的职责时，可以使用抽象构件的方法。

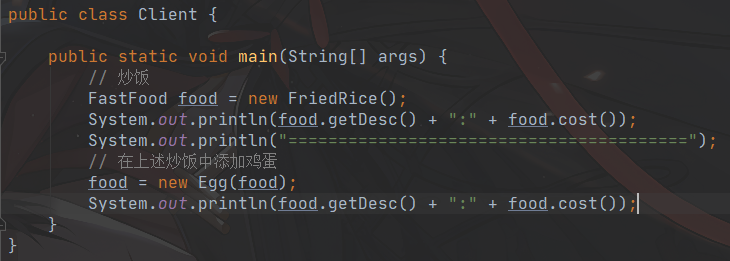
包含具体构件的实例：可以接受具体构件的对象作为参数，在其上叠加。

1. 具体装饰



这里，把鸡蛋作为一个FastFood类的新对象，因此，通过FastFood类的getPrice方法可以获取其价格（不能用cost因为在FastFood中cost没有实现）；传入的参数本身是一个FastFood类的对象（而且应该是FriedRice或FriedNoddles中的一种），通过父类的getFastFood方法可以获取，再用cost方法求其价格。

1. 客户类



直接传递food作为参数，在其上叠加鸡蛋或培根。