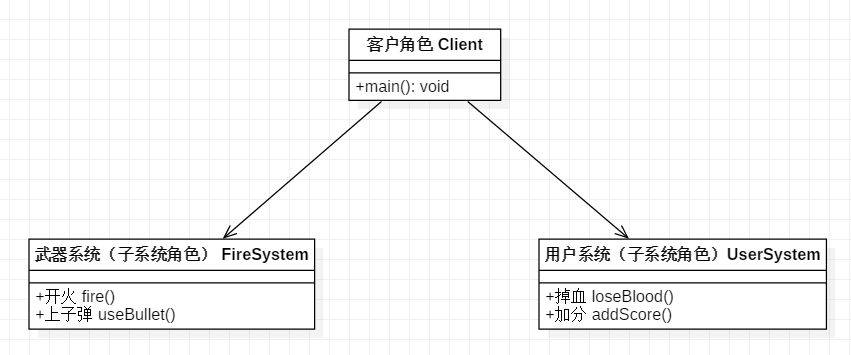
1. 定义

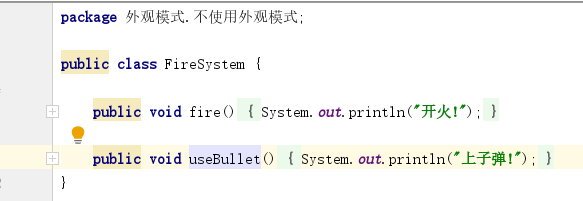
为各个子系统的一组接口提供一致的调用窗口或门面，使得子系统更容易使用，使得复杂的子系统与客户端分离解耦。

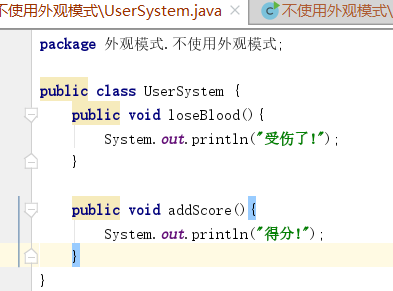
1. 使用实例

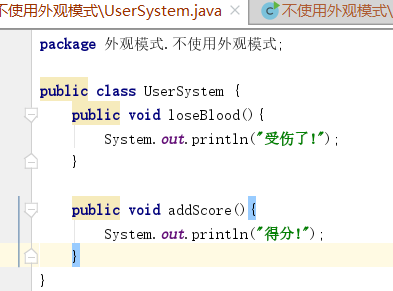
以射击游戏为例，玩家对战时，需要进行射击操作，射击操作牵扯到一连串的动作，比如:商子丹、瞄准、发射子弹、掉血、加分等一系列动作，这些动作我们可以理解为各个子系统的某个接口API，比如上子弹、发射子弹可能是武器子系统的API,掉血、加分可能是用户子系统的api，客户角度需要调用的接口其实只有一个，就是射击API，这就是具体的门面接口，门面内部的各个子系统的动作对客户是透明的，这种客户只需要调用门面接口API就实现了一连串内部动作(上子弹、瞄准、发子弹、掉血、加分等)的模式。

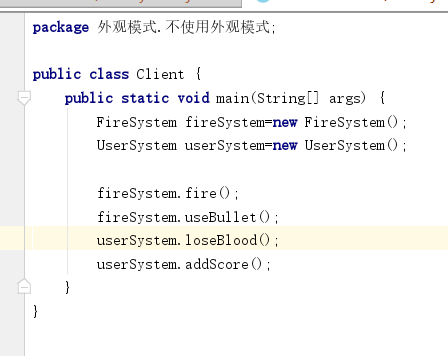
2.1 不使用外观模式





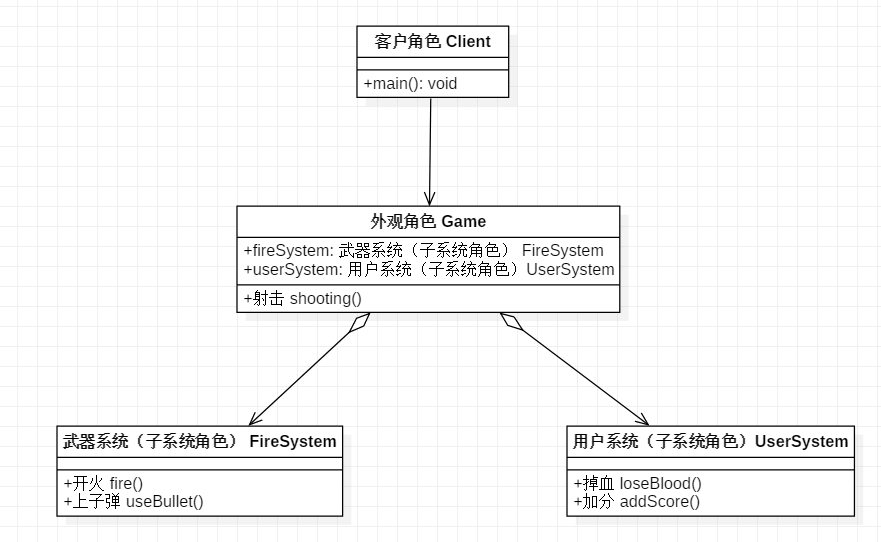




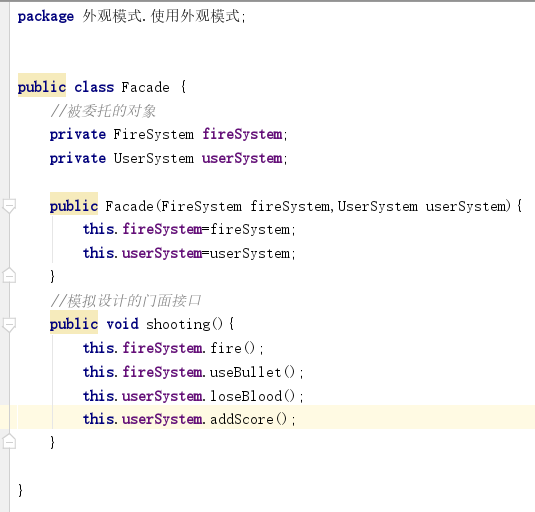


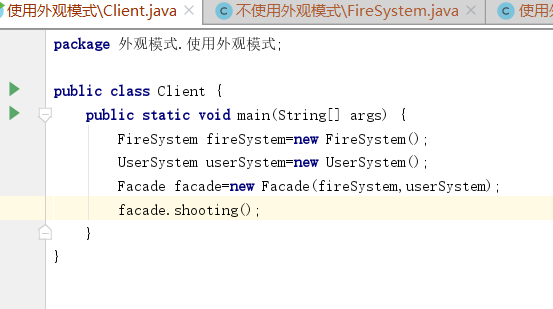
2.2 使用外观模式

上面不使用外观模式时，可以看到客户端需要自己去直接调用各个子系统API,系统模块多的时候对客户端十分不友好，我们用外观模式去改善这种情况。

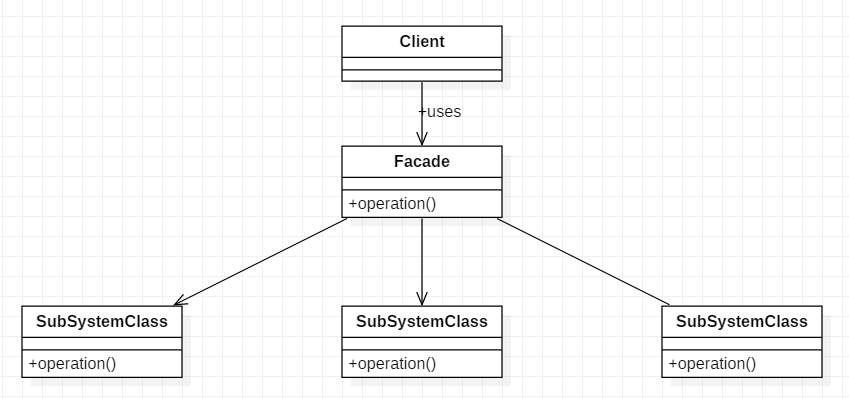


我们引入Facade角色，该角色内部包含各个子系统的委托对象，客户端的所有请求经过Facade角色中转，简化了客户端操作的复杂性。





1. 组成角色



\*门面角色(Facade): 一般情况下该角色会将客户端的请求委派给相应的子系统去调用，也就是说该角色没有什么实质性的业务逻辑，只是一个单纯的委派类，用来实现客户端和子系统的解耦；

\*子系统角色(SubSystem): 子系统并不是一个单一的类，而是众多类的一个系统集合。一般而言，子系统并不知道门面角色的存在。子系统各自实现自己的功能，包括类之间的相互调用，这些都不受门面角色的影响。

1. 优缺点

外观模式优点:

实现了子系统与客户端之间关系的解耦;

客户端屏蔽了子系统组件，使得客户端所需处理的对象数目有所减少，使得子系统使用更加方便。

缺点：

增加新的子系统可能需要修改外观类或者客户端的源代码，违背了开闭原则；

外观类并没有阻断子系统被外部使用的可能性。