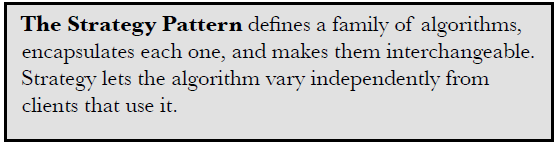
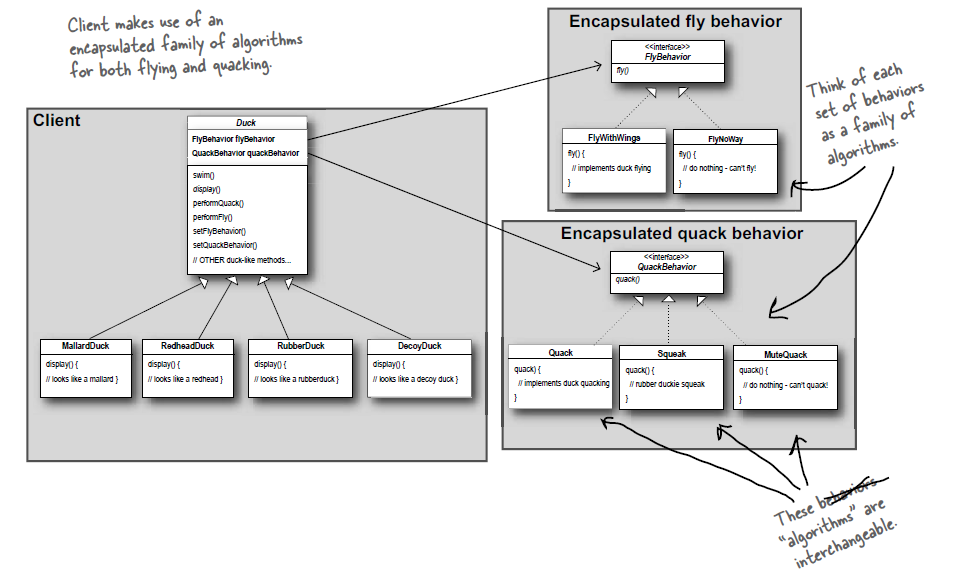
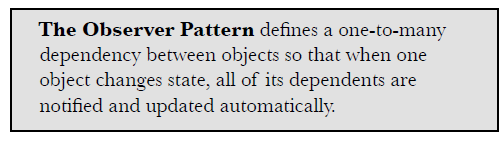
1. 策略模式



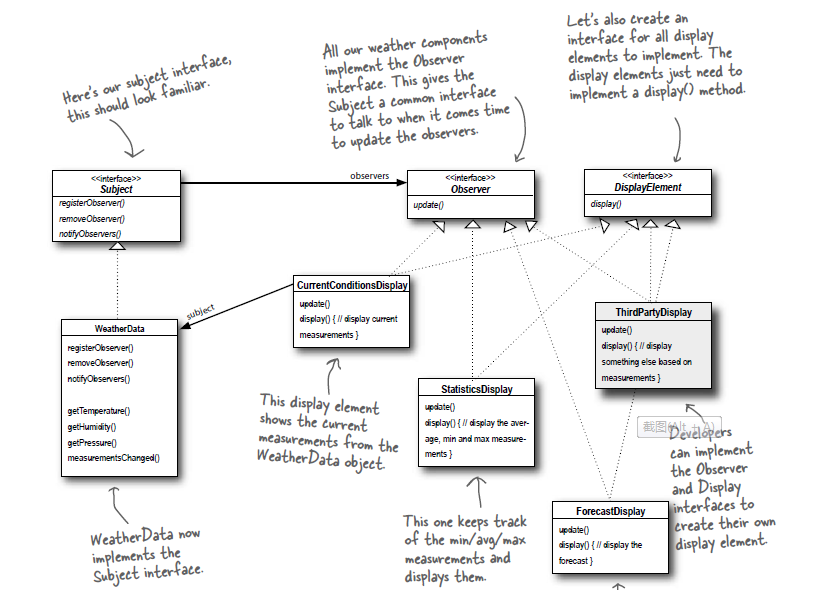


面向接口编程就是典型的策略模式

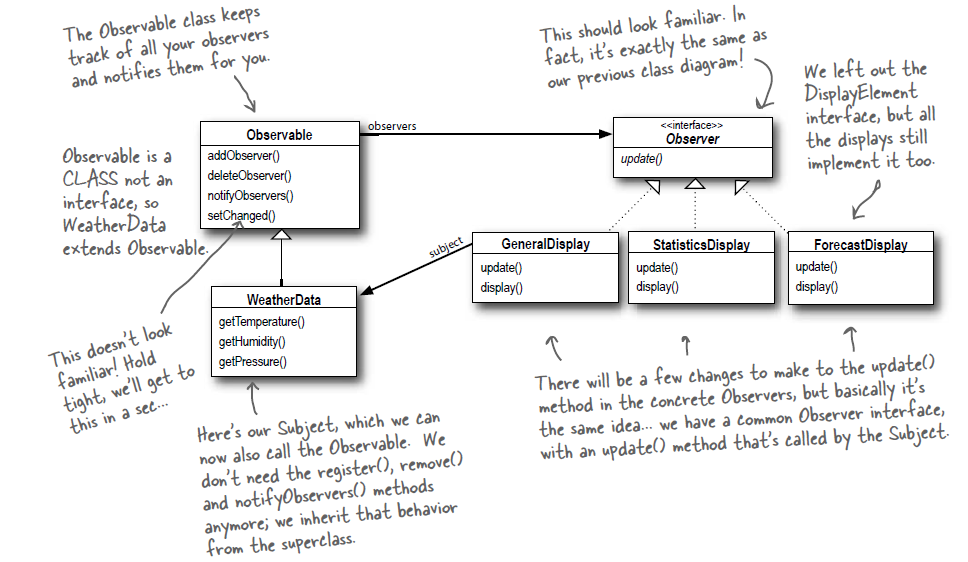
1. 观察者模式



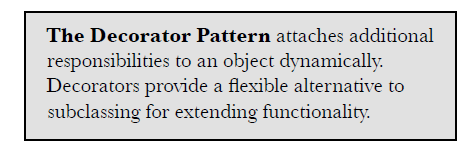
自己写



java提供

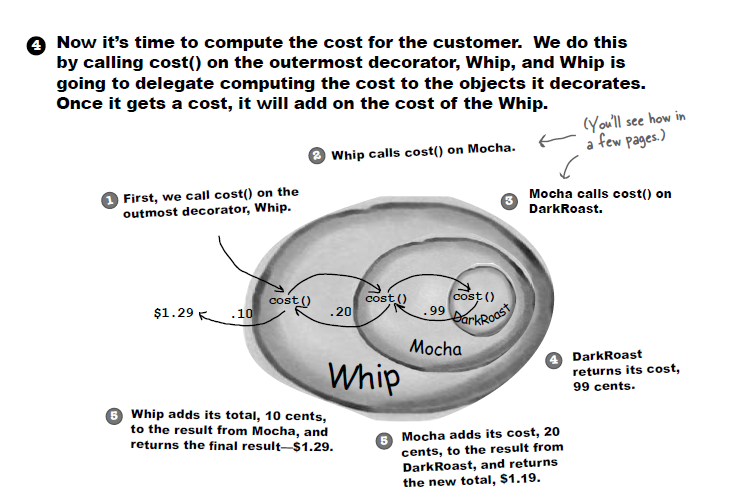


1. 装饰模式

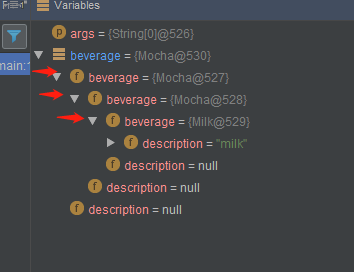


前提：装饰者和被装饰都必须是继承了同一个类。

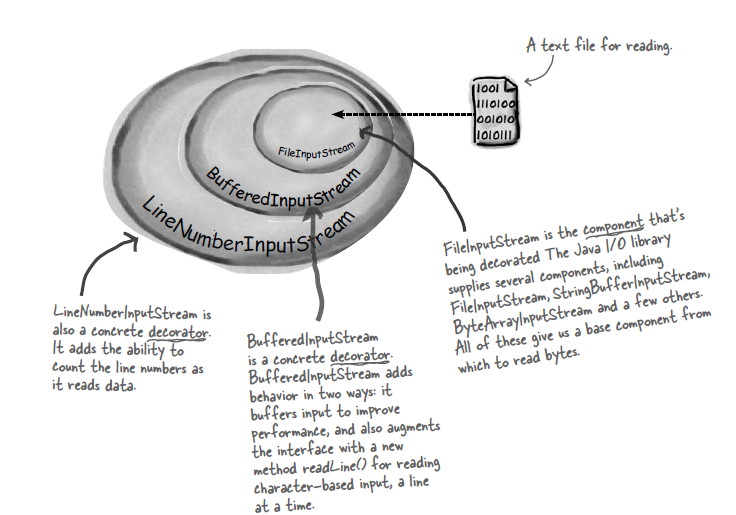
一层套一层

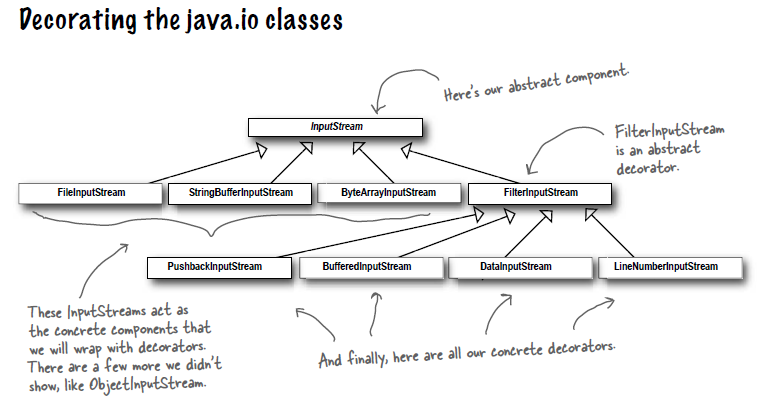


1546049180(1)

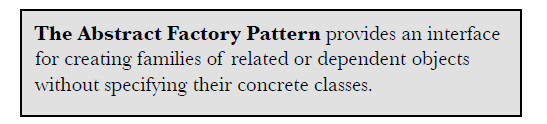
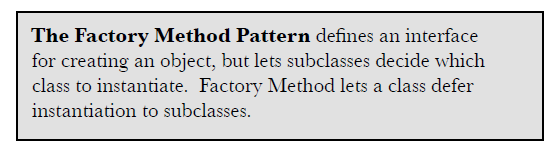


应用：javaIO



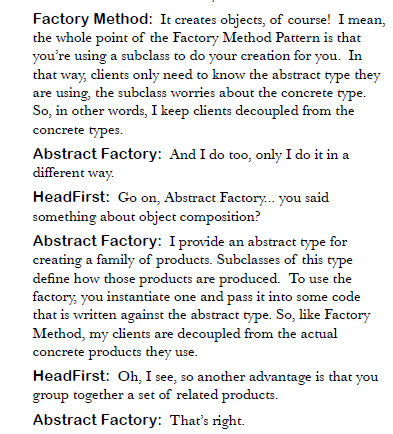


1. 工厂模式和抽象工厂

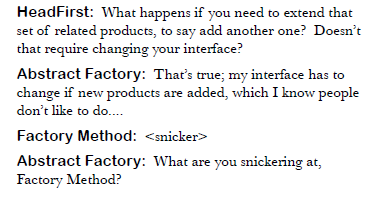
定

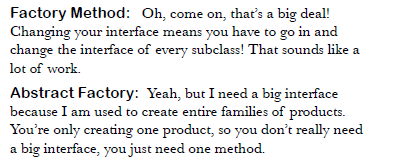
区别：

* 1. 工厂模式只负责创造实体，抽象工厂可以提供产品

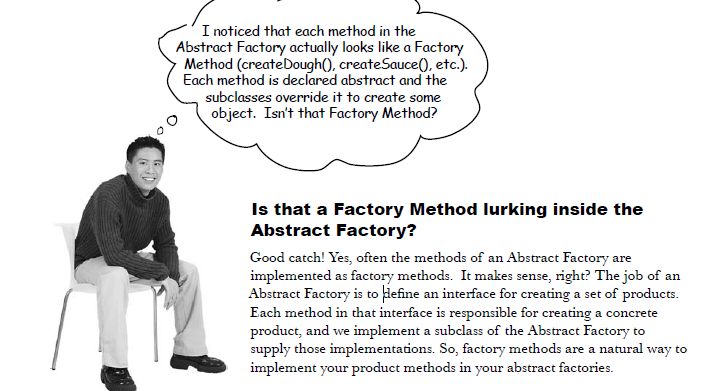


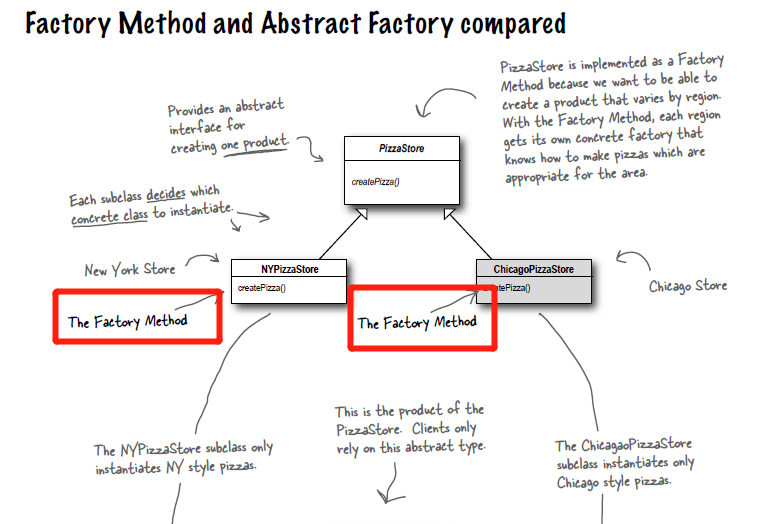
* 1. 当需要加一个类型时，工厂无需改变，只需要加对应类型即可。抽象工厂需要加上接口，因为抽象工厂提供的是一个产品

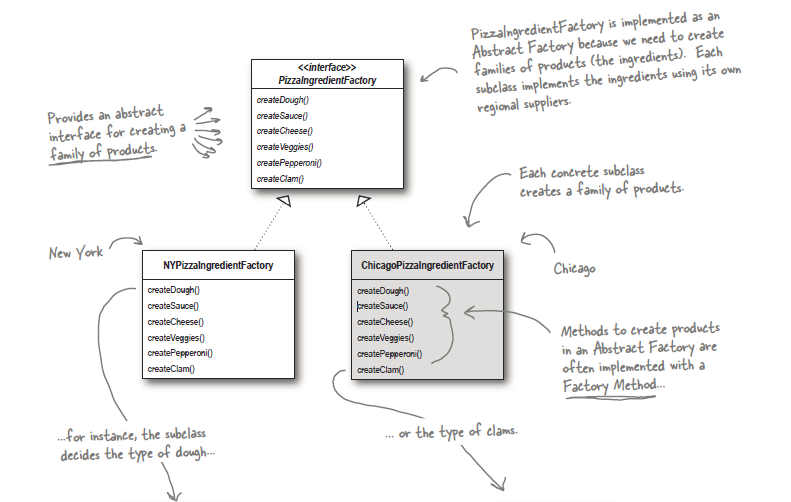


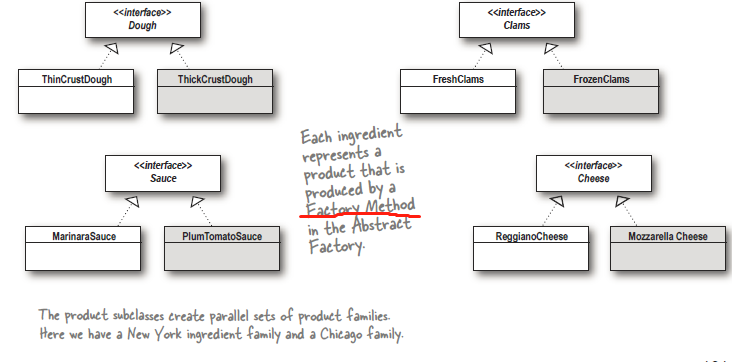


* 1. 抽象工厂提供组合，并且抽象工厂在创造组合中的元件时也会用到工厂模式



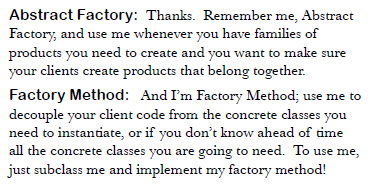




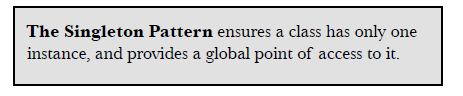


* 1. 两者应用

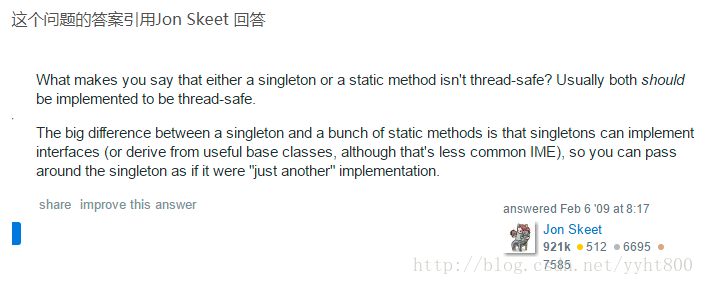
当需要一个产品时，用抽象工厂，当只需要创建实例来松耦合时用工厂



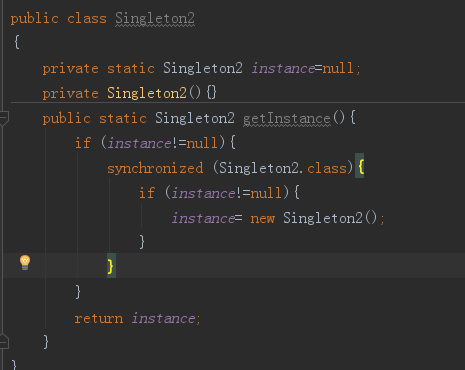
1. 单例模式



* 1. 和静态类最重要的区别

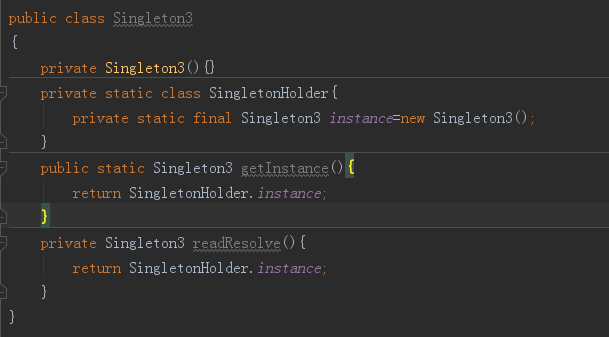


* 1. 实现方式

Double Check Lock(DCL模式)

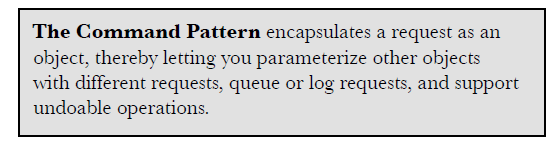
*缺点:第一次加载稍慢,由于java处理器允许乱序执行,偶尔会失败*

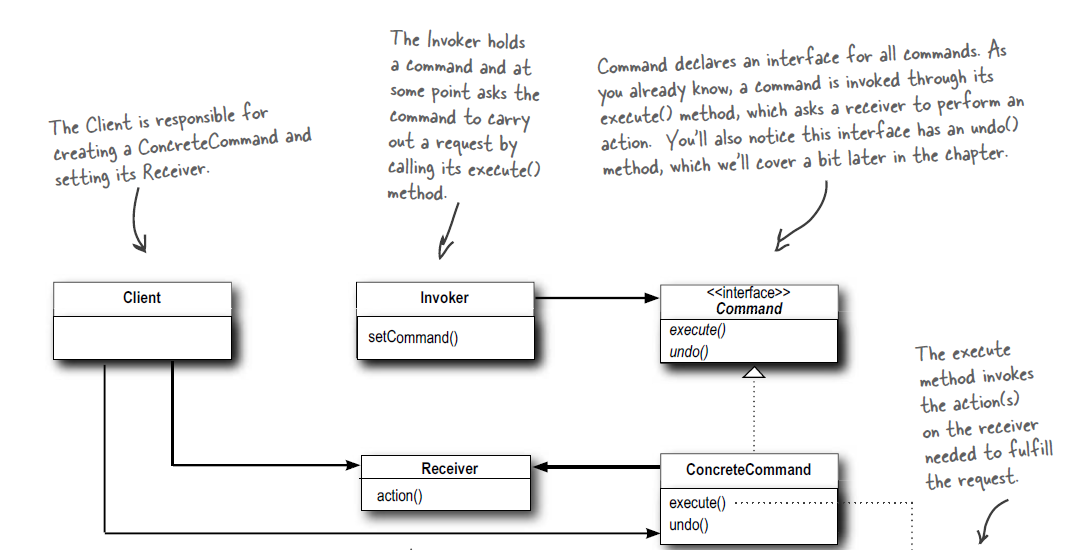
静态内部类实现单例模式

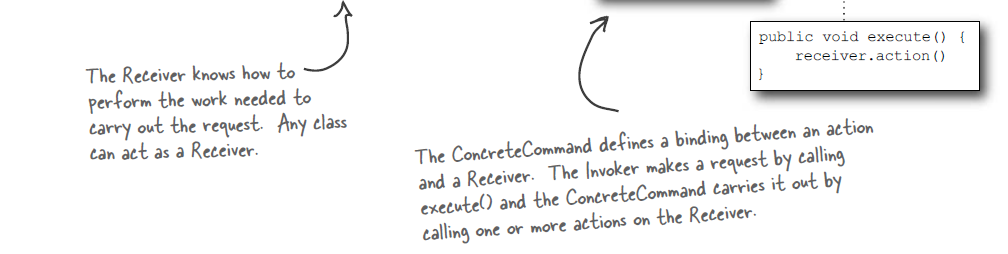


反序列化时会重新生成对象的问题:加入readResolve方法即可解决。因为反序列化时会调用这个方法。

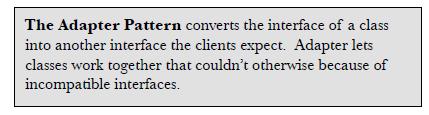
1. 命令模式



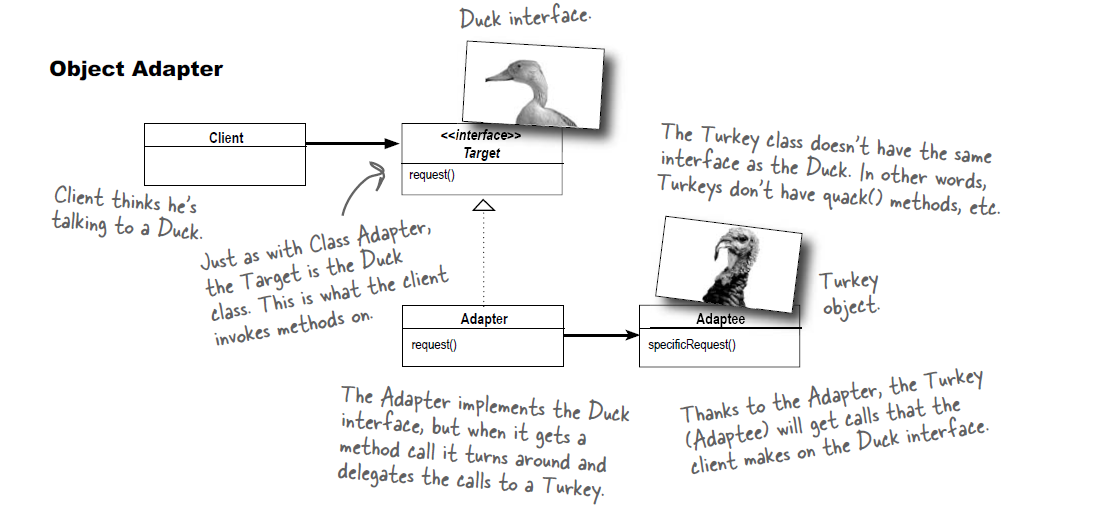




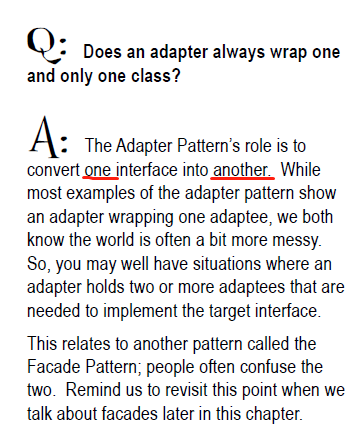
1. 适配器模式



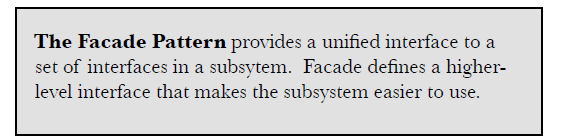
* 1. 形式



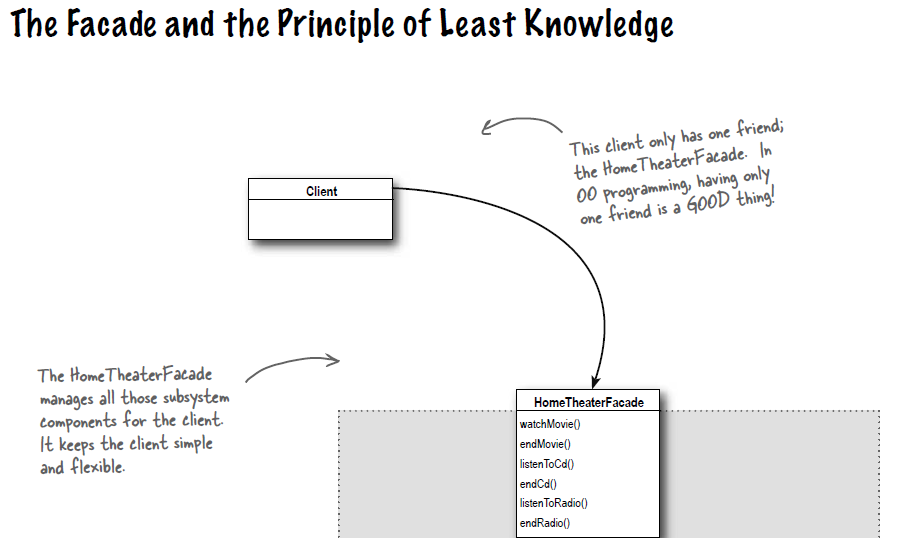
* 1. 一个只能转一个

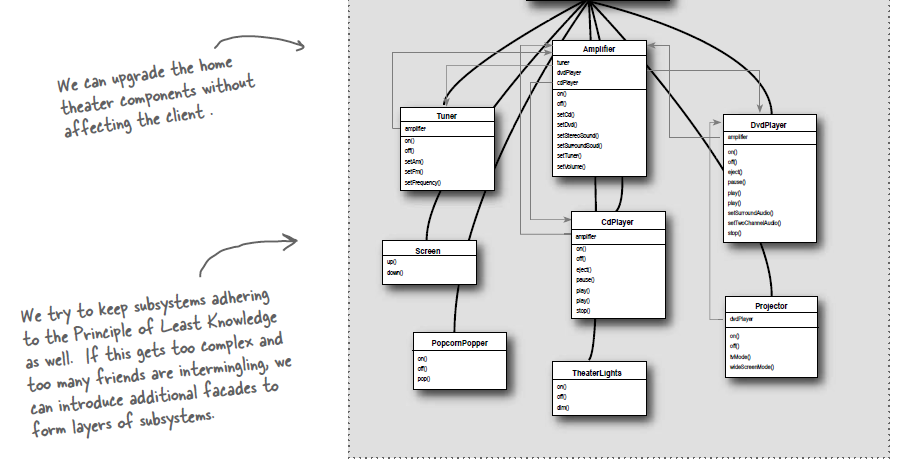


1. 门面模式

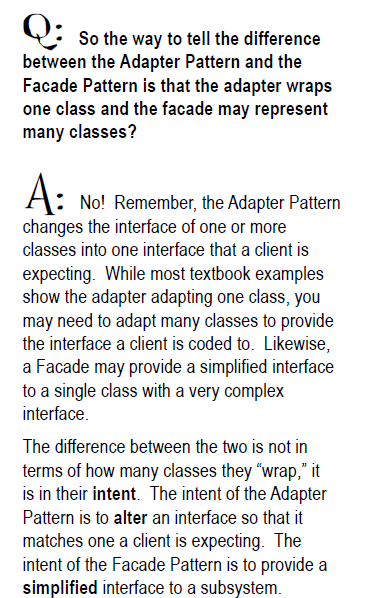


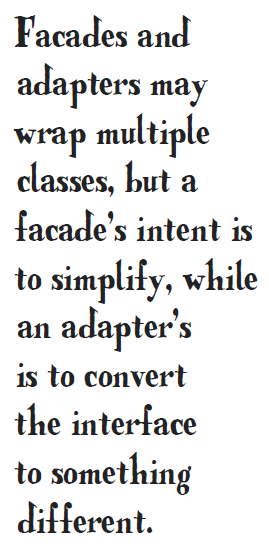
* 1. 形式





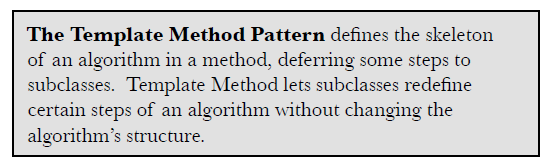
* 1. 与适配模式区别



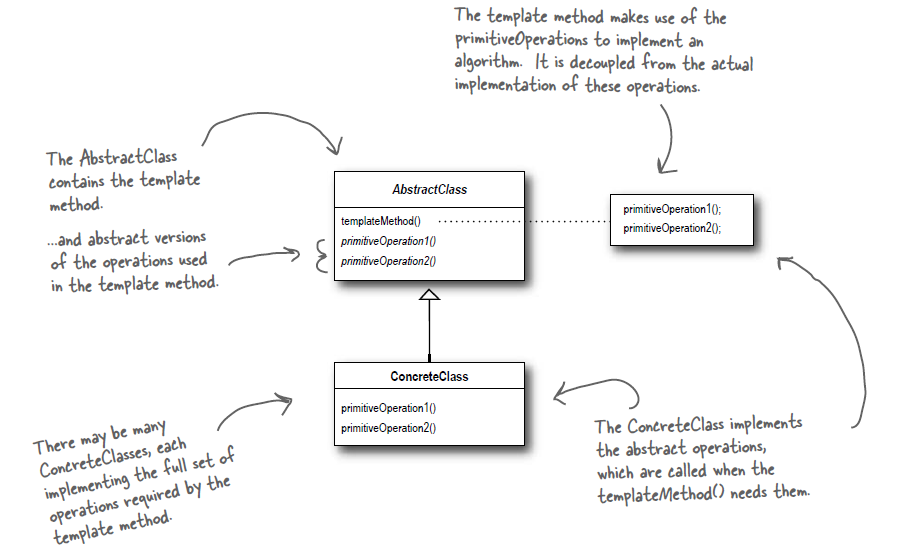


适配是把一种转化为另一种，门面是提供一个更简洁的接口给子类

1. 模板模式

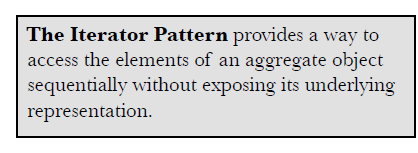


* 1. 形式

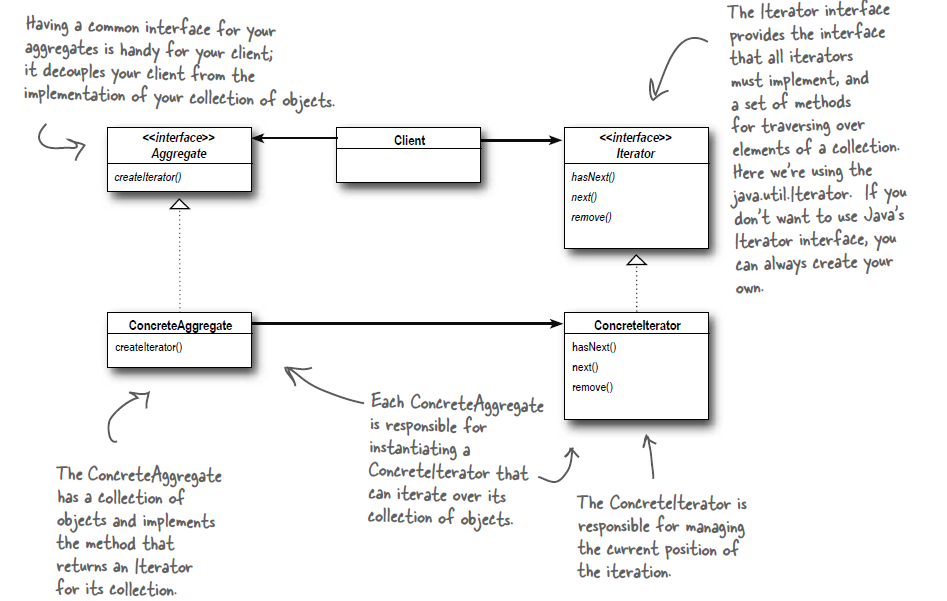


* 1. 应用：非常广泛，工厂模式就是一个特殊的模板模式，JAVA SWING也是一个模板模式，等等。

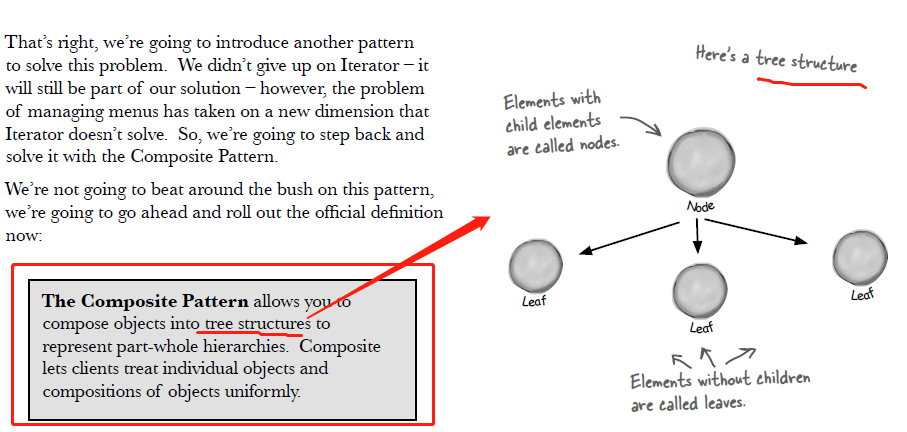
1. 迭代器模式



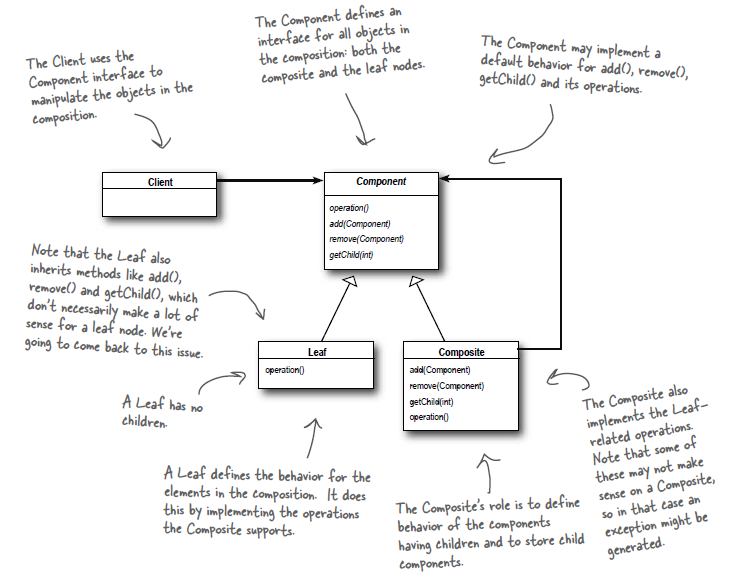
都实现Iterator接口，客户端用iterotor来调用



1. 组合模式
   1. 定义

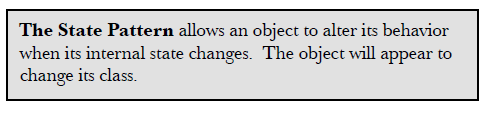


* 1. 形式

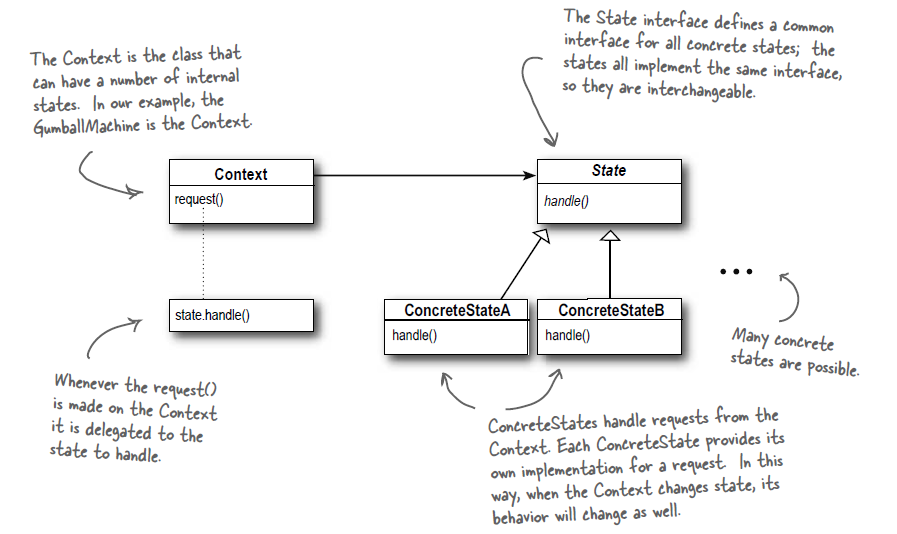


* 1. 应用：JAVA SWING就是一个树状结构

1. 状态模式
   1. 定义



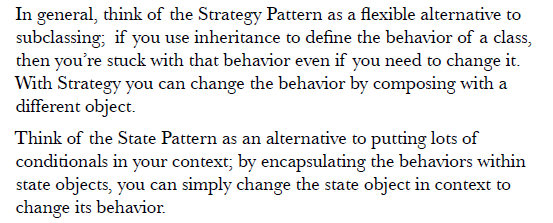
* 1. 形式

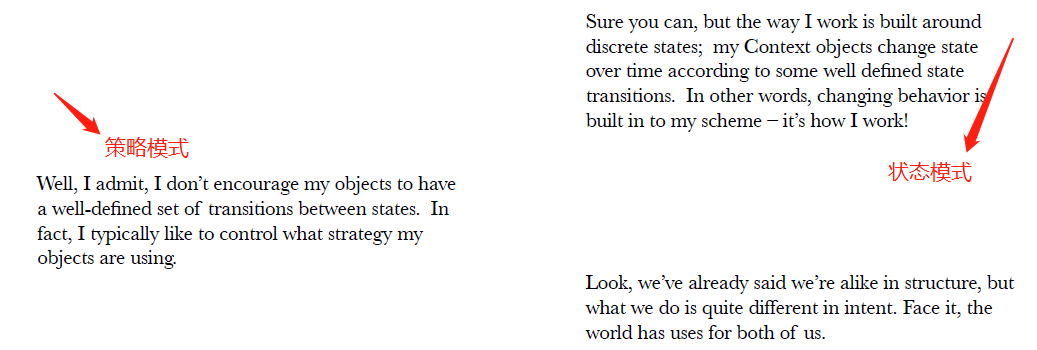


* 1. 与策略模式区别

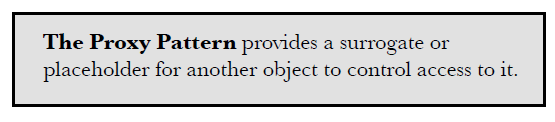
形式一样，但意图不同

* + 1. 策略模式:把许多灵活的改变归到子类。来实现不同的行为。
    2. 状态模式：把许多条件放进上下文。来更容易的改变状态对象的行为。

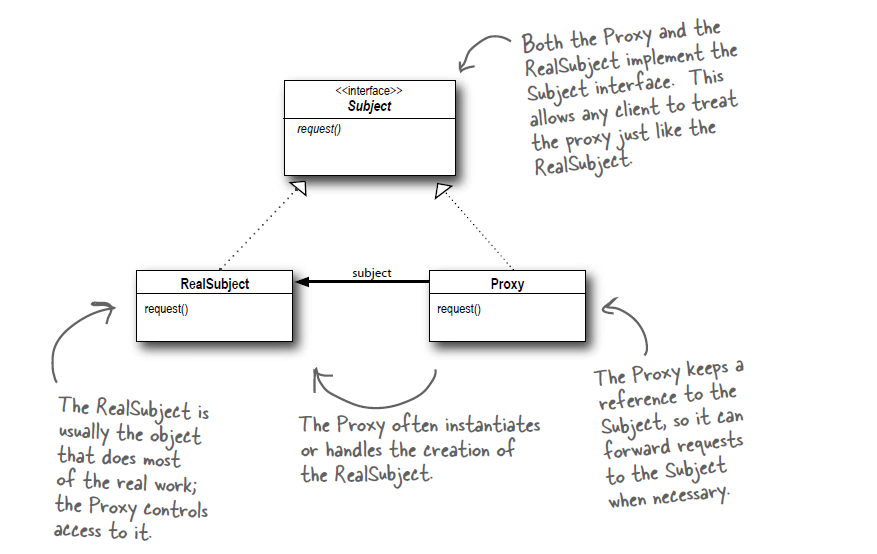


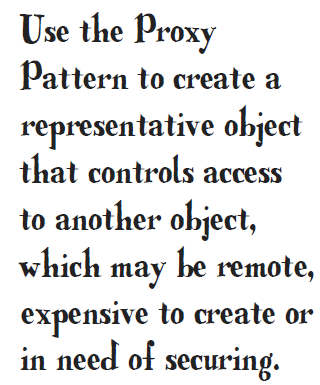


1. 代理模式



* 1. 普通代理





* 1. JAVA动态代理

