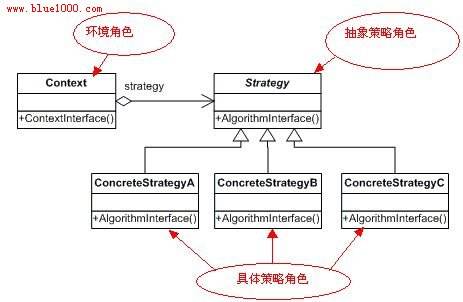
设计原则：

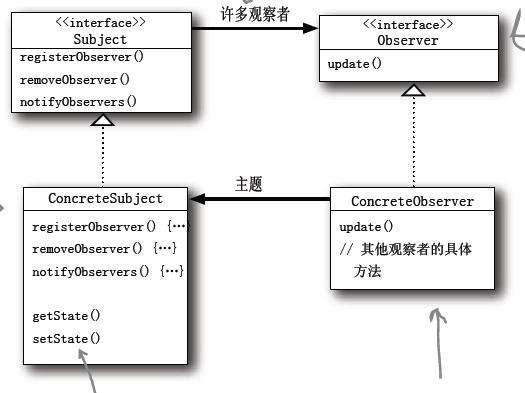
1. 找出应用中可能需要变化之处，把他们独立出来，不要和那些不需要变化的代码混在一起（开放封闭原则）把会变化的部分取出并封装起来，以便以后可以轻易地改动或扩充此部分，而不影响不需要变化的其他部分。所有的模式都提供了一套方法让“系统中的某部分改变不会影响其他部分”。
2. 针对接口编程而不是针对实现编程
3. 多用组合，少用继承
4. 为了交互对象之间的搜耦合设计而努力
5. 类应该对拓展开放，对修改关闭
6. 要依赖抽象，不要依赖具体类（依赖倒置原则）

设计模式：

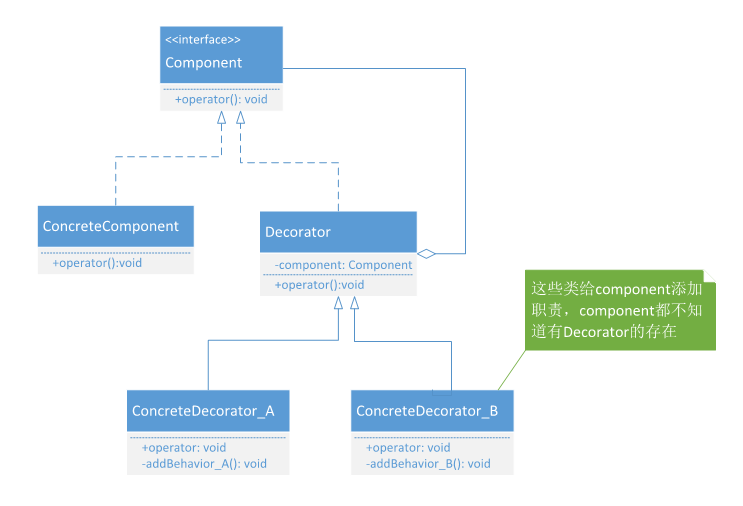
1. 策略模式定义了算法族，分别封装起来，让他们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。



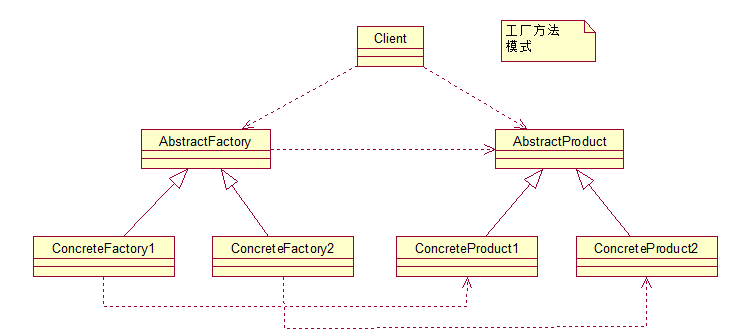
1. 观察者模式定义了对象之间的一对多依赖，这样一来，当一个对象改变状态时，它的所有依赖者都会受到通知并自动更新。



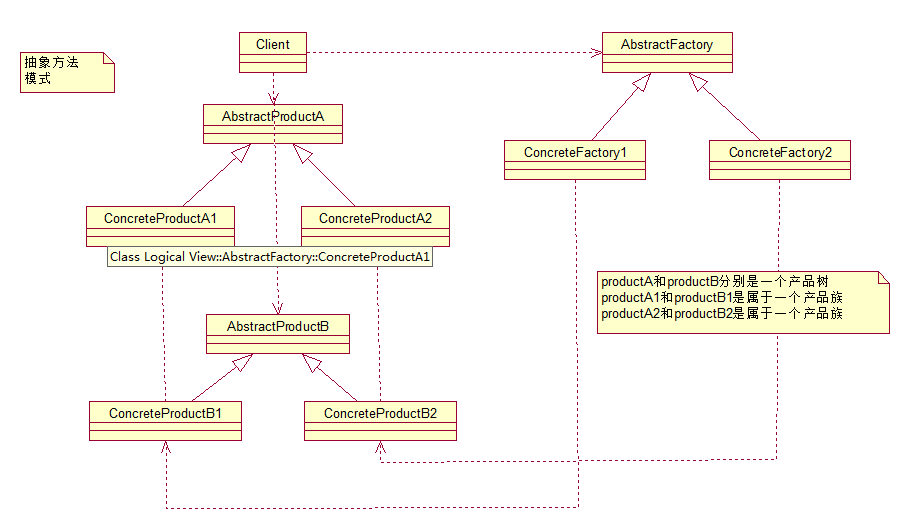
1. 装饰者模式动态地将责任附加到对象上。若要拓展功能，装饰者提供了比继承更有弹性的替代方案



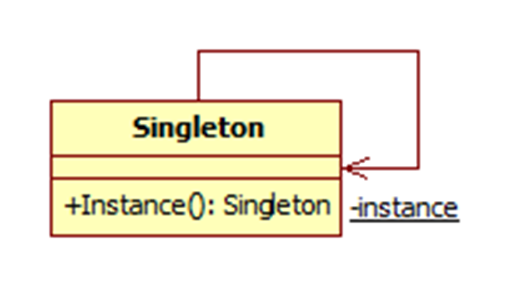
1. 工厂方法模式定义了一个创建对象的接口，但子类决定要实例化的类是哪一个。工厂方法让类把实例化推迟到子类



5、抽象工厂模式提供一个借口，用于创建相关或依赖对象的家族，而不需要明确指定具体类



1. 单例模式确保一个类只有一个实例，并提供一个全局访问点



单例模式的优点：

1. 由于只有一个实例，减少内存开支，减少系统性能的开销，避免对资源的多重占用。
2. 对唯一实例的受控访问，因为Singleton类封装它的唯一实例，所以可以严格的控制客户怎样以及何时访问它
3. 缩小名空间，Singleton模式是对全局变量的一种改进。它避免了哪些存储唯一实例的全局变量污染名空间。
4. 跨平台：使用合适的中间件，可以把singleton模式拓展为跨多个JVM和多个计算机工作
5. 适用于任何类：只需要把一个类的构造函数变成私有的，并且在其中增加相应的静态函数和变量，就可以把这个类变为SINGLETON
6. 延迟求职：如果SINGLETON从未使用过，那么就不会创建它

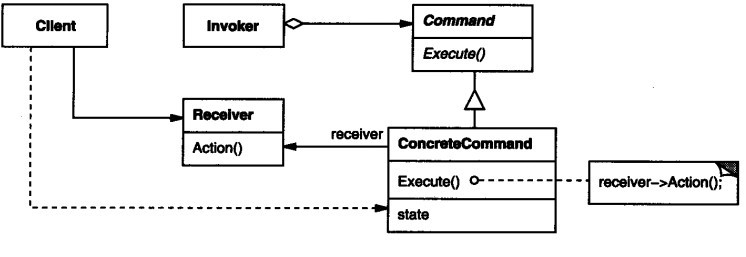
单例模式的缺点：

1. 摧毁方法未定义：没有好的方法去摧毁一个SINGLETON，或者解除其职责。
2. 不能继承：
3. 效率问题：每次调用instance方法都会执行if语句。
4. 不透明
5. 单例模式一般没有接口，拓展困难。

适用场景：

* 当类只能有一个实例而且客户可以从一个众所周知的访问点访问时
* 当这个唯一实例应该是通过子类化拓展的，并且客户应该无需要更改代码就能使用一个拓展的实例时
* 要求生成唯一序列号的环境
* 在整个项目中需要一个共享方位点或共享数据。
* 创建一个对象需要消耗的资源过多，如访问IO和数据库等资源
* 需要定义大量的静态常量和静态方法的环境

1. 命令模式将"请求"封装成对象，以便使用不同的请求、队列或者日志来参数化其他对象。命令模式也支持可撤销的操作



1. 适配器模式将一个类的接口，转换成客户期望的另一个接口。适配器让原本接口不兼容的类可以合作无间（必须是面向接口的编程才合适）
2. 外观模式提供了一个统一的接口，用来访问子系统中的一群接口。外观定义了一个高层接口，让子系统更容易使用。
3. 模板方法模式在一个方法中第一个算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以在不改变算法结构的情况下，重新定义算法中的某些步骤
4. 迭代器模式提供一种方法顺序访问一个聚合对象中的各个元素，而又不暴露其内部的表示