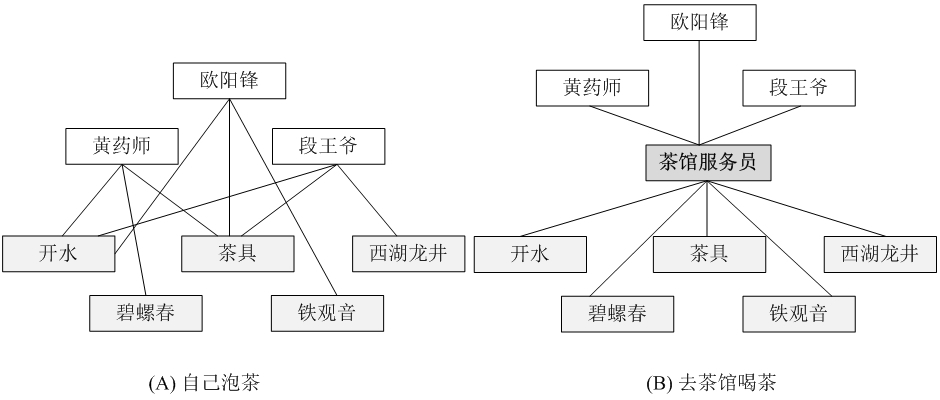
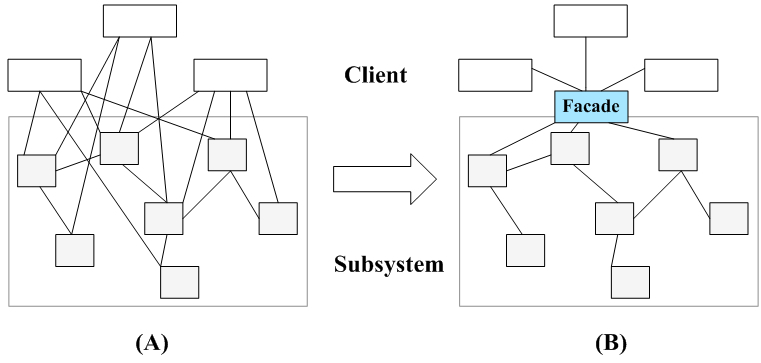
## 一、外观模式概述

      不知道大家有没有比较过自己泡茶和去茶馆喝茶的区别，如果是自己泡茶需要自行准备茶叶、茶具和开水，如图1(A)所示，而去茶馆喝茶，最简单的方式就是跟茶馆服务员说想要一杯什么样的茶，是铁观音、碧螺春还是西湖龙井？正因为茶馆有服务员，顾客无须直接和茶叶、茶具、开水等交互，整个泡茶过程由服务员来完成，顾客只需与服务员交互即可，整个过程非常简单省事，如图1(B)所示。



**图1** **两种喝茶方式示意图**

       在软件开发中，有时候为了完成一项较为复杂的功能，一个客户类需要和多个业务类交互，而这些需要交互的业务类经常会作为一个整体出现，由于涉及到的类比较多，导致使用时代码较为复杂，此时，特别需要一个类似服务员一样的角色，由它来负责和多个业务类进行交互，而客户类只需与该类交互。外观模式通过引入一个新的**外观类(Facade)**来实现该功能，外观类充当了软件系统中的“服务员”，它为多个业务类的调用提供了一个统一的入口，简化了类与类之间的交互。在外观模式中，那些需要交互的业务类被称为**子系统(Subsystem)**。如果没有外观类，那么每个客户类需要和多个子系统之间进行复杂的交互，系统的耦合度将很大，如图2(A)所示；而引入外观类之后，客户类只需要直接与外观类交互，客户类与子系统之间原有的复杂引用关系由外观类来实现，从而降低了系统的耦合度，如图2(B)所示。



**图2** **外观模式示意图**

**外观模式中，一个子系统的外部与其内部的通信通过一个统一的外观类进行，外观类将客户类与子系统的内部复杂性分隔开，使得客户类只需要与外观角色打交道，而不需要与子系统内部的很多对象打交道。**

      外观模式定义如下：

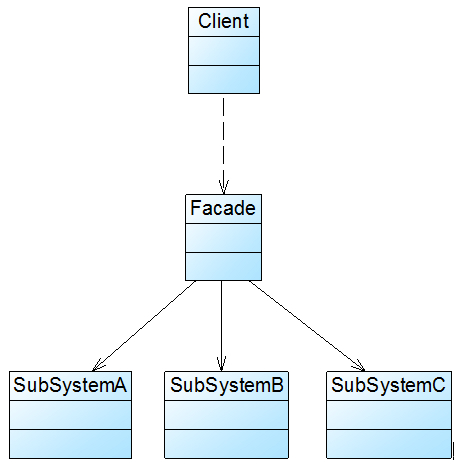
|  |
| --- |
| **外观模式：**为子系统中的一组接口提供一个统一的入口。外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。  **Facade Pattern:** Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. |

      外观模式又称为门面模式，它是一种对象结构型模式。外观模式是迪米特法则的一种具体实现，通过引入一个新的外观角色可以降低原有系统的复杂度，同时降低客户类与子系统的耦合度。

## 二、外观模式结构与实现

### 2.1 模式结构

      外观模式没有一个一般化的类图描述，通常使用如图2(B)所示示意图来表示外观模式。图3所示的类图也可以作为描述外观模式的结构图：



**图3** **外观模式结构图**

       由图3可知，外观模式包含如下两个角色：

 **(1) Facade（外观角色）：**在客户端可以调用它的方法，在外观角色中可以知道相关的（一个或者多个）子系统的功能和责任；在正常情况下，它将所有从客户端发来的请求委派到相应的子系统去，传递给相应的子系统对象处理。

 **(2) SubSystem（子系统角色）：**在软件系统中可以有一个或者多个子系统角色，每一个子系统可以不是一个单独的类，而是一个类的集合，它实现子系统的功能；每一个子系统都可以被客户端直接调用，或者被外观角色调用，它处理由外观类传过来的请求；子系统并不知道外观的存在，对于子系统而言，外观角色仅仅是另外一个客户端而已。

### 2.2 模式实现

      外观模式的主要目的在于降低系统的复杂程度，在面向对象软件系统中，类与类之间的关系越多，不能表示系统设计得越好，反而表示系统中类之间的耦合度太大，这样的系统在维护和修改时都缺乏灵活性，因为一个类的改动会导致多个类发生变化，而外观模式的引入在很大程度上降低了类与类之间的耦合关系。引入外观模式之后，增加新的子系统或者移除子系统都非常方便，客户类无须进行修改（或者极少的修改），只需要在外观类中增加或移除对子系统的引用即可。从这一点来说，外观模式在一定程度上并不符合开闭原则，增加新的子系统需要对原有系统进行一定的修改，虽然这个修改工作量不大。

      外观模式中所指的子系统是一个广义的概念，它可以是一个类、一个功能模块、系统的一个组成部分或者一个完整的系统。子系统类通常是一些业务类，实现了一些具体的、独立的业务功能

## 三、模式优缺点

**优点**

 1、引入外观模式，是客户对子系统的使用变得简单了，减少了与子系统的关联对象，实现了子系统与客户之间

的松耦合关系。

 2、只是提供了一个访问子系统的统一入口，并不影响用户直接使用子系统类

 3、降低了大型软件系统中的编译依赖性，并简化了系统在不同平台之间的移植过程

**缺点**

   1、不能很好地限制客户使用子系统类，如果对客户访问子系统类做太多的限制则减少了可变性和灵活性

   2、在不引入抽象外观类的情况下，增加新的子系统可能需要修改外观类或客户端的源代码，违背了“开闭原则”

**四、使用场景**

   1、当要为一个复杂子系统提供一个简单接口时可以使用外观模式。

   2、客户程序与多个子系统之间存在很大的依赖性。引入外观类将子系统与客户以及其他子系统解耦，可以提

高子系统的独立性和可移植性

**五、模式总结**

   1、  外观模式的主要优点就在于减少了客户与子系统之间的关联对象，使用客户对子系统的使用变得简单了，

也实现了客户与子系统之间的松耦合关系。它的缺点就在于违背了“开闭原则”。

   2、  如果需要实现一个外观模式，需要将子系统组合进外观中，然后将工作委托给子系统执行。