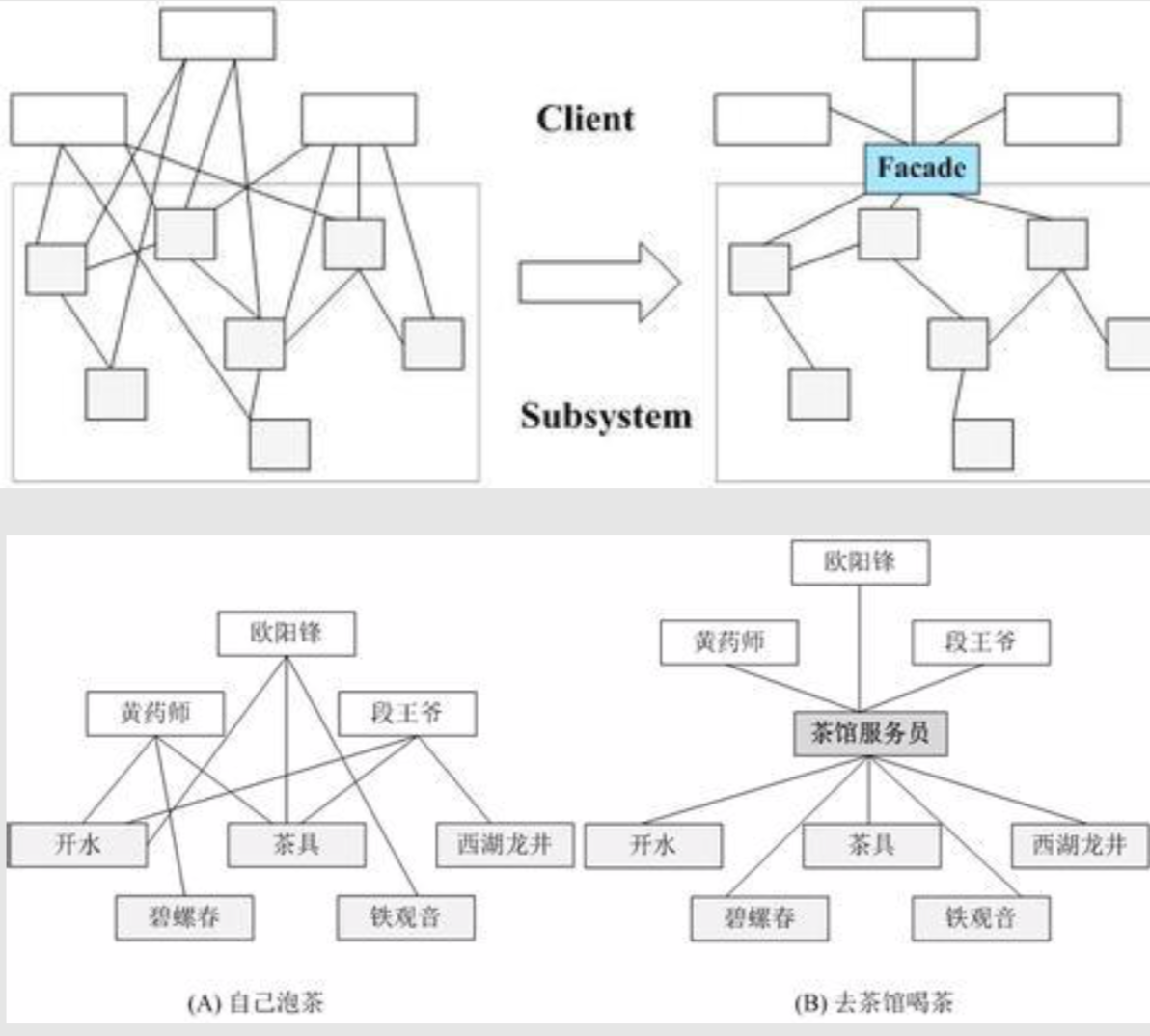
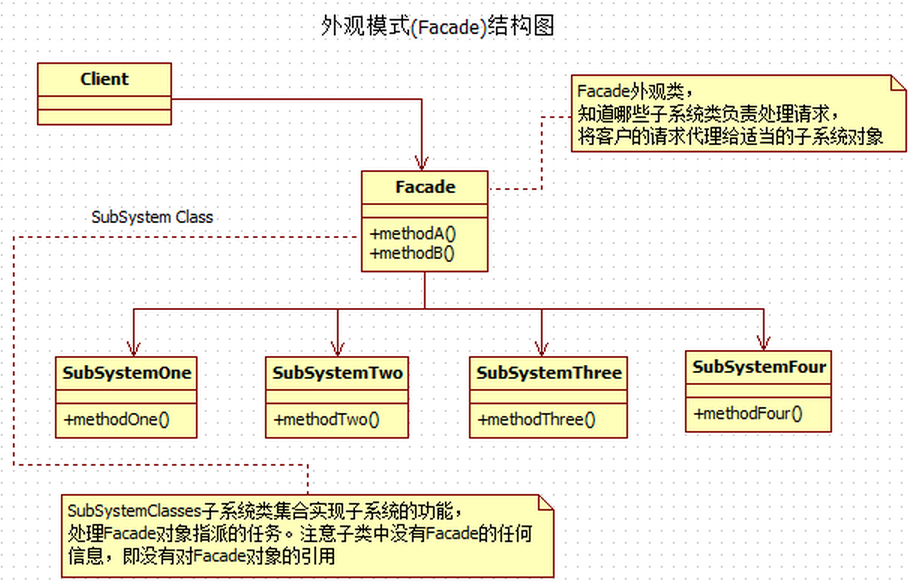
1. 外观模式(Facade-Pattern)：外观模式提供了一个统一的接口，用来访问子系统中的一群接口。外观定义了一个高层接口，让子系统更容易使用。也被称为门面模式
2. 迪米特法则(最少知识原则): 一个软件实体应当尽可能少的与其他实体发生相互作用

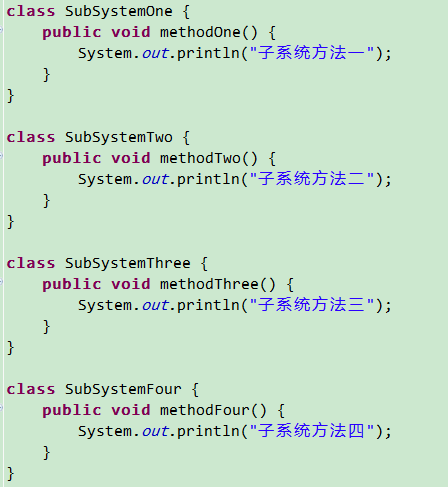


1. 外观模式的意图是提供子系统的一个简化接口，好让一个子系统更加容易使用。即为子系统提供统一的入口, 封装子系统的复杂性, 便于客户端调用. 外观模式可以理解为转换一群接口，客户只要调用一个接口，而不用调用多个接口才能达到目的
2. 外观模式就是解决多个复杂接口带来的使用困难，起到简化用户操作的作用
3. 类图

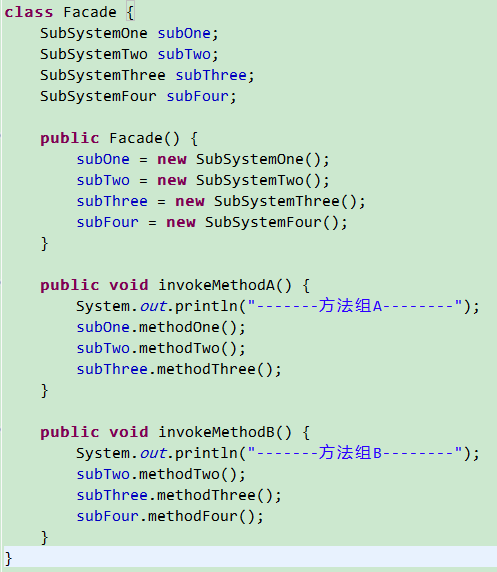


* 1. 外观类(Facade): 为调用端提供统一的调用接口, 外观类知道哪些子系统负责处理请求,从而将调用端的请求代理给适当子系统对象
  2. 调用者(Client): 外观接口的调用者
  3. 子系统的集合: 指模块或者子系统，处Facad对象指派的任务，他是功能的实际提供者

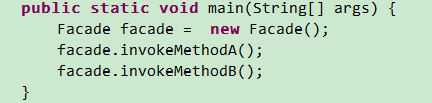
1. 实例
   1. 定义子系统



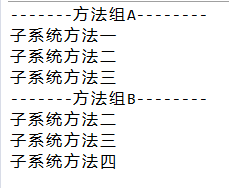
定义外观



测试



输出



1. 开发中常见的场景
   1. 自定义封装方法
   2. 框架中各种Util类等
2. 外观模式的注意事项
   1. 外观模式对外屏蔽了子系统的细节，因此外观模式降低了客户端对子系统使用的复杂性
   2. 外观模式对客户端与子系统的耦合关系 - 解耦，让子系统内部的模块更易维护和扩展
   3. 通过合理的使用外观模式，可以帮我们更好的**划分访问的层次**
   4. 当系统需要进行分层设计时，可以考虑使用Facade模式
   5. 在维护一个遗留的大型系统时，可能这个系统已经变得非常难以维护和扩展，此时可以考虑**为新系统开发一个 Facade 类**，来提供遗留系统的比较清晰简单的接口，让新系统与 Facade 类交互，提高复用性
   6. 不能过多的或者不合理的使用外观模式，使用外观模式好，还是直接调用模块好。要以让系统有层次，利于维护为目的。