2、解释门面模式的意图、适用性，并请使用门面模式设计一个在线订货网站使用的一个类叫做 OrderFacade，该类的使用使得客户无需知道下订单以后复杂的操作即可实现在线购买商品。该门面类 OrderFacade 通过调用如下两个类来实现一个订货流程：
(1).库存检查类 Inventory，该类返回一个字符串 'Inventory checked'
(2).付款处理类 Payment，该类返回一个字符串'Payment deducted successfully'
在门面类 OrderFacade 中，完成付款以后该类输出一个字符串‘Above steps completed'
请设计该门面类 OrderFacade，库存检查类 Inventory，和付款处理类 Payment。并设计一个客户端类 Client 来测试你所设计的这个门面类。并画出 UML 图解释你的程序（25 分）

解答：意图：将一个系统划分成为若干个子系统有利于降低系统的复杂性。一个常见的设计目标是使子系统间的通信和相互依赖关系达到最小。门面模式可以为子系统中的一组接口提供一个一致的访问界面，此模式定义了一个高层接口，这个接口使得子系统更加容易使用。
适用性：
1）当要为一个复杂子系统提供一个简单接口时。子系统往往因为不断演化而变得越来越复杂。大多数模式使用时都会产生更多更小的类。这使得子系统更具可重用性，也更容易对子系统进行定制，但这也给那些不需要定制子系统的用户带来一些使用上的困难。Façade 可以提供一个简单的缺省视图，这一视图对大多数用户来说已经足够，而那些需要更多的可定制性的用户可以越过 Façade 层。
2）客户程序与抽象类的实现部分之间存在着很大的依赖性。引入 Façade 将这个子系统与客户以及其他的子系统分离，可以提高子系统的独立性和可移植性。
3）当你需要构建一个层次结构的子系统时，使用 Façade 模式定义子系统中每层的入口点。如果子系统之间是相互依赖的，你可以让它们仅通过 Façade 进行通讯，从而简化了它们之间的依赖关系。

```
public class Inventory {
    public String checkInventory() {
    return 'Inventory checked';
    }
}

public class Payment {
    public String deductPayment() {
    return 'Payment deducted successfully';
    }
}

public class OrderFacade {
    private Payment pymt = new Payment();
    private Inventory inventry = new Inventory();
    public void placeOrder(String orderId) {
        String step1 = inventry.checkInventory();
        String step2 = pymt.deductPayment();
```

```
        System.out.println(orderId + 'Above steps completed');
        }
}

public class Client {
    public static void main(String args[]){
    OrderFacade orderFacade = new OrderFacade();
    orderFacade.placeOrder('OrderTest');
    System.out.println('Order processing completed');
    }
}
```