# 设计思路分析：

本次实验提交所采用的设计模式为：简单工厂和外观模式。

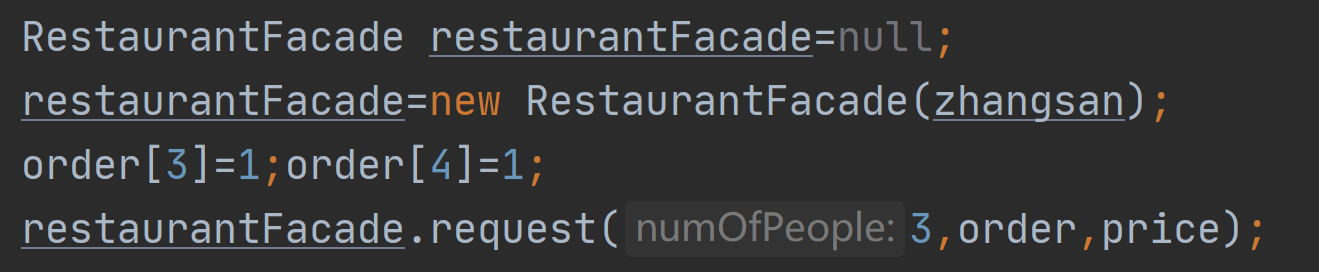
简单工厂模式是将顾客（Customer）和管理员（Administrator）分开注册，在UserFactory类中通过String类型变量控制用户注册的是Customer类还是Administrator类。输入相关参数后通过工厂创建相应的对象。



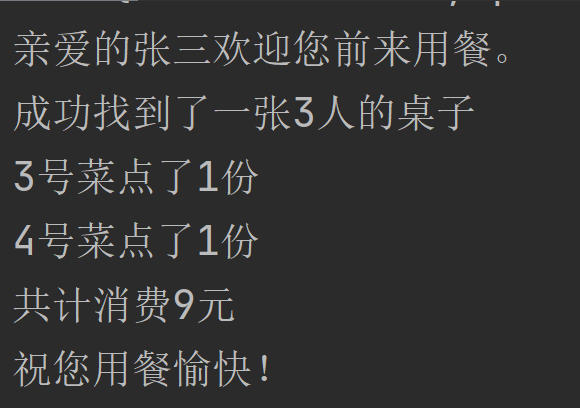
再将对象打印出来，方便观察：



外观模式是使用RestaurantFacade类作为中介将用户和后台的需求操作分离，用户直接与RestaurantFacade类交流，后台的实现类直接与RestaurantFacade类，而用户与后台实现类不直接交流。



用户直接向RestaurantFacade类发请求，得到实现的反馈信息：



# 使用简单工厂的优点:

实现了对象创建和使用的分离,客户端无须知道所创建的具体产品类的类名，只需要知道具体产品类所对应的参数即可,通过引入配置文件,可以在不修改任何客户端代码的情况下更换和增加新的具体产品类，在一定程度上提高了系统的灵活性，在本次实验中将顾客（Customer）和管理员（Administrator）分开注册，使得在将来若需要再添加新的User类子类变得更加方便快捷。

# 使用外观模式的优点：

引入一个新的外观类(RestaurantFacade)来负责和多个业务类【子系统(Subsystem)】进行交互，而客户类只需与外观类交互。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。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

它对客户端屏蔽了子系统组件，减少了客户端所需处理的对象数目，并使得子系统使用起来更加容易

它实现了子系统与客户端之间的松耦合关系，这使得子系统的变化不会影响到调用它的客户端，只需要调整外观类即可

一个子系统的修改对其他子系统没有任何影响，而且子系统的内部变化也不会影响到外观对象