1. 利斯科夫替换原则（Liskov Substitution Principle，LSP）

原则定义：子类对象必须能够替换其父类对象，而不影响程序的正确性。即，子类必须确保不改变父类的行为约定。

实践应用：在商品网上交易系统中，假设有一个基类`PaymentMethod`，表示支付方式。它有两个子类`CreditCardPayment`和`PayPalPayment`，分别表示信用卡支付和PayPal支付。在实现订单支付功能时，无论是使用`CreditCardPayment`还是`PayPalPayment`，都应确保可以无缝替换而不影响支付流程的正常执行。

2. 单一职责原则（Single Responsibility Principle，SRP）

原则定义：一个类应该只有一个引起变化的原因，即一个类只负责一个职责。

实践应用：在用户管理模块中，可以将用户信息管理、用户登录与注册、用户收藏夹管理等功能分开。创建单独的类`UserProfileManager`负责用户信息管理，`AuthenticationService`负责用户登录与注册，`FavoritesManager`负责用户收藏夹管理。这使得每个类的职责单一，便于维护和扩展。

3. 开闭原则（Open/Closed Principle，OCP）

原则定义：软件实体（类、模块、函数等）应该对扩展开放，对修改关闭。即，软件实体应该能够通过扩展来增强其功能，而无需修改其现有代码。

实践应用：在商品搜索功能中，初始版本可能只支持按商品名称搜索。随着需求增加，可能需要按价格、类别等条件搜索。通过定义一个抽象的`SearchCriteria`接口，并为每种搜索条件创建具体实现类，如`NameSearchCriteria`、`PriceSearchCriteria`等，可以在不修改现有代码的情况下，扩展新的搜索功能。

4. 德（迪）米特法则（Law of Demeter，LoD）

原则定义：一个对象应该对其他对象有尽可能少的了解，即尽量减少对象之间的依赖关系。

实践应用：在订单管理模块中，假设订单对象需要获取用户的地址信息。如果通过`order.getUser().getAddress()`这样的方式来获取，会增加对象之间的耦合。可以通过在订单对象中添加一个方法`getShippingAddress()`，由订单对象负责获取用户的地址信息，从而减少耦合。

5. 依赖倒转原则（Dependency Inversion Principle，DIP）

原则定义：高层模块不应该依赖于低层模块，两者都应该依赖于抽象。抽象不应该依赖于细节，细节应该依赖于抽象。

实践应用：在支付模块中，创建一个`PaymentProcessor`接口，高层模块通过这个接口来处理支付。具体的支付实现，如`CreditCardProcessor`和`PayPalProcessor`，实现这个接口。这样，支付处理逻辑依赖于抽象的`PaymentProcessor`接口，而不是具体的支付实现，方便以后添加新的支付方式。

6. 合成复用原则（Composite Reuse Principle，CRP）

原则定义：优先使用对象组合（composition）而不是继承来实现功能复用。

实践应用：在购物车管理模块中，如果需要为购物车添加不同的促销规则，可以创建一个`Promotion`接口，每种促销规则实现这个接口。通过将多个`Promotion`对象组合到购物车中，而不是让购物车类继承各种促销规则类，实现促销规则的复用和扩展。

总结

在商品网上交易系统中，通过应用上述面向对象设计原则，可以提高系统的灵活性和可维护性。具体地，使用LSP确保子类可以替换父类而不引入错误，使用SRP使每个类职责单一，使用OCP通过扩展实现功能增强而不修改现有代码，使用LoD减少对象之间的耦合，使用DIP使高层模块依赖于抽象而不是具体实现，使用CRP通过组合而非继承实现代码复用。这些原则共同作用，构建出一个高质量的软件系统。