### 科斯科夫替换原则：

1. 核心思想：**子类型（Subtype）对象必须能够替换其基类型（Base type）对象，而程序的行为保持不变。**
2. **具体内容：  
    LSP 对子类覆盖或实现父类方法的行为提出了具体的约束，确保子类不会破坏父类的“契约”：**

（1）前置条件不能强化：

父类方法接受输入参数有一定的前提条件（Preconditions）。子类覆盖该方法时，不能要求比父类更严格的前置条件（即不能缩小可接受的输入范围）。

原因：如果客户端代码按照父类的前置条件调用方法，并传递了一个对父类有效的参数，但当替换为子类对象时，该参数被子类认为无效（因为子类加强了前置条件），就会导致程序出错或行为不一致。

（2）后置条件不能弱化：

父类方法执行后，会保证某些结果或状态（Postconditions）。子类覆盖该方法时，不能提供比父类更弱的保证（即不能缩小输出的范围或减弱状态的保证）。

原因： 客户端代码依赖于父类方法的后置条件。如果子类提供了更弱的保证（比如父类保证返回非负数，子类却可能返回负数），那么替换后，客户端代码基于父类后置条件所做的假设就可能被打破。

（3）不变量必须保持：

父类定义了一些在对象生命周期内始终为真的属性或状态（Invariants）。子类必须维持这些不变量。

原因：客户端代码依赖于这些不变量来理解对象的状态和行为。如果子类破坏了这些不变量，替换后对象的状态就可能进入客户端无法预料或处理的状态。

（4）不抛出新的异常：

父类方法声明抛出某些类型的异常。子类覆盖该方法时，不能抛出父类方法未声明的新的检查型异常（Checked Exception）。可以抛出父类已声明异常的子类型异常，或者不抛出异常。运行时异常（Runtime Exception）通常没有这个严格限制，但也要谨慎。

原因：客户端代码只处理父类声明的异常。如果子类抛出一个父类未声明的新检查型异常，客户端代码没有捕获它的准备，程序就会崩溃。

1. 实例

（1）初始设计



User.deleteComment() 的后置条件之一是：如果用户没有权限，方法将抛出 UnauthorizedException。 这是 User 类行为契约的一部分。AdminUser.deleteComment() 从不抛出 UnauthorizedException，它总是成功删除评论。

当用AdminUser 替换User时，客户端代码 user.deleteComment (someComment); 预期可能会捕获 UnauthorizedException 来处理权限不足的情况。但 AdminUser 的行为改变了：它永远不会抛出这个异常（后置条件被弱化了——不再保证在无权限时抛异常，因为它总是有权限）。如果客户端依赖这个异常流（比如记录权限失败日志），用 AdminUser 替换后，这部分逻辑就失效了，程序行为发生了改变。

1. 改进设计

重新审视继承关系： AdminUser 能删除评论是额外能力，而不是对 User “不能删除（或受限删除）”行为的替代。让 AdminUser 直接继承 User 并改变“不能删除”的行为到“总能删除”，容易违反 LSP。

方案：定义一个 CommentModerator 接口，包含 deleteComment 方法。让 AdminUser 实现 User 和 CommentModerator 接口。客户端需要删除评论时，检查对象是否是 CommentModerator。

### 单一职责原则

1. 核心思想：一个类（或者模块、函数）应该只有一个引起它变化的原因。
2. 具体内容

（1）聚焦于“职责”而非“功能”：一个类应该只负责一件事，或者说只代表一个特定的功能点或业务概念。

（2）高内聚：将紧密相关的属性和方法组织在一起，不相关的功能拆分出去。

（3）变化隔离：当需求变更时，只需要修改负责该变更点的特定类（或少量相关类），不会牵一发而动全身。如果一个类承担了多个职责，那么修改其中一个职责的代码时，可能会无意中破坏其他职责的功能。

（4）避免“上帝类”：防止创建那种无所不能、包含大量不相关功能的庞大类。

1. 实例

（1）初始设计



UserService 同时承担了三项责任：

用户账户管理：注册、登录、更新资料。

权限检查：判断用户是否有权编辑特定文章。

通知：发送密码重置邮件。

权限检查 (canEditArticle) 本质上属于授权(Authorization)逻辑，与核心的用户账户管理 (register, login, updateProfile) 关系并不紧密。发送密码重置邮件属于通知逻辑。

1. 改进设计

设置PermissionService函数：专门负责权限判断。



### 三、开闭原则

1.核心思想：软件实体（类、模块、函数等）应该对扩展开放，但对修改关闭。

1. 具体内容

（1）对扩展开放（Open for Extension）

系统应该允许在不修改现有代码的情况下添加新功能。

可以通过添加新类、新模块或新函数来扩展系统行为。

（2）对修改关闭（Closed for Modification）

一旦一个模块完成并通过测试，就不应该再修改其源代码。

现有代码应该保持稳定，避免因添加新功能而引入错误。

(3)核心目标：通过最小化对现有代码的修改来增加新功能，从而：

减少回归错误的风险、提高代码稳定性、增强系统的可维护性和可扩展性。

1. 实例



初始需求是过滤评论中的敏感词，新增需求要过滤广告链接。

这就导致每次添加新过滤规则都需要修改CommentProcessor类，违反了对修改关闭原则。