“Принципы S.O.L.I.D. объектно-ориентированного проектирования архитектуры ПО.”

Владислав Павлюк

АКАДЕМИЯ ШАГ

Группа ПР211

2023

Зачем нужны принципы S.O.L.I.D. программисту?

Принципы SOLID это пять золотых правил, цель которых улучшить процессы проектирования и обслуживаимости ПО. Эти принципы особенно актуальны для Agile-разработки, иак как они помогают создавать гибкий, масштабируемый и легко модифицируемый программный код.

Работодатели чаще ищут кандидатов, которые имеют глубокое понимание принципов SOLID, так как они могут помочь снизить затраты улучшить долгосрочное устойчивое развитие программных продуктов.

Ряд преимуществ достигаются следуя принципам SOLID при проектировании и разработке ПО. Среди них:

Улучшенная обслуживаемость: Программист может создавать код, который легче поддерживается и модифицируется с прошествием времени потому, что прицины SOLID стимулируют создание модульного, гибкокого кода, который менее склонен к появлению ошибок и более устойчив к изменениям и требованиям.

Упрощенность: Принципы SOLID помогают снизить сложность ПО продвигая абстракцию и инкапсулязию, которые облегчают понимание и работу с кодом.

Расширенная гибкость: Принципы стимулируют создание гибко-изменяемого кода, который открыт к расширению, но закрыт к модификации, что стимулирует изменяемость без нарушения существующей функциональности.

Повышенная масштабируемость: Принципы SOLID могут помочь делать ПО более масштабирумым, так как они стимулируют использование абстракций и разделение зависимостей, которые помагают предотвращать излишнюю сложность и запутанность исходного кода, трудную для поддержки.

Что такое принципы S.O.L.I.D.?

SOLID это аббривиатура 5 принципов проектрирования ООП которые влекут за собой читабельность, адаптируемость и масштабируемость кода. Принципы SOLID могут быть применены к любой программе ООП.

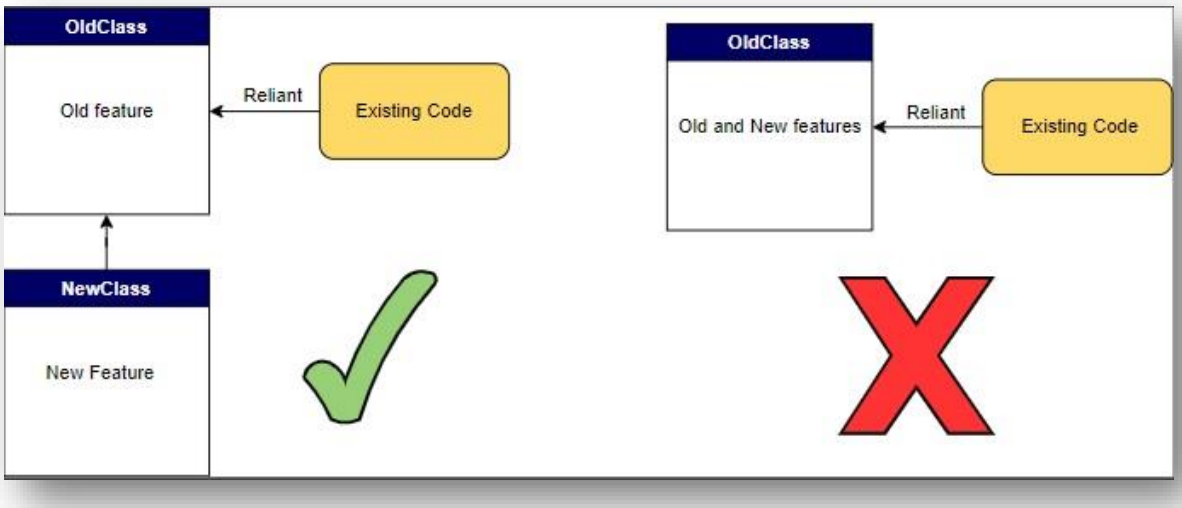
5 принципами SOLID являются:

- **S**ingle-responsibility principle (принцип единственной обязанности)
- **O**pen-closed principle (Принцип открытости / закрытостиости)
- **L**iskov substitution principle (принцип замены Лисков)
- **I**nterface segregation principle (Принцип отделения интерфейса)
- **D**ependency inversion principle (Принцип инверсии зависимостей)

- **Принцип Единственной Обязанности:** каждый объект должен иметь одну и только одну обязанность и эта обязанность должна быть полностью инкапсулирована в модуль или класс. Все методы и свойства класса (экземпляра класса) должны быть направлены исключительно на выполнение этой обязанности. Объединение нескольких сущностей, имеющих разные сферы ответственности в одном классе или модуле, считается неудачным проектным решением. каждый объект в программе должен иметь единственную обязанность.

Если объект выполняет множество различных обязанностей – его необходимо разделить. Например, объект печати отчётов ответственен за формат и за содержимое отчётов – это неправильно. За формат должен отвечать один объект, за содержимое – другой.

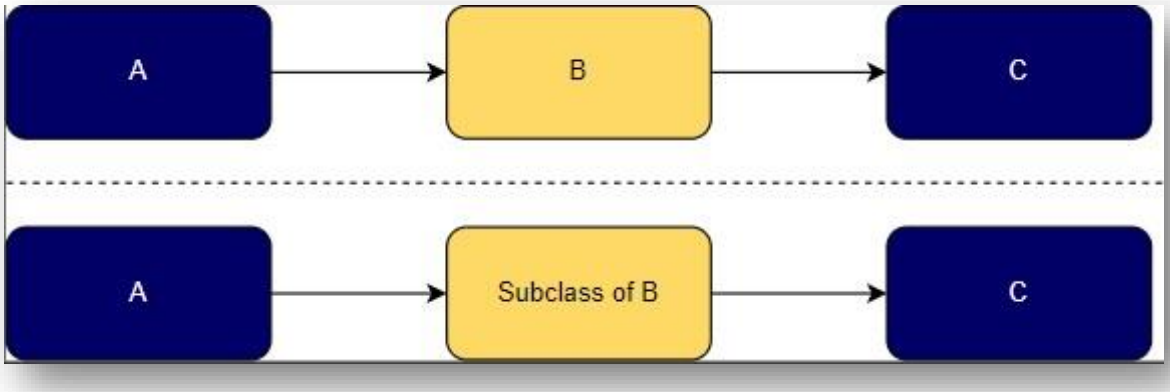
- **Принцип открытости / закрытости (OCP):** “Програмные сущности должны быть открыты для раширения, но закрыты для модификации.” (Роберт С.Мартин). OCP принцип вызывает сущности которые могут быть широко адаптированы, но остаются неизменном состоянии. Этостимулирует программиста создавать дубликаты сущностей с особым поведение посредством **Полиморфизма**. Благодаря П., мы можем расширять нашу родительскую сущность в соответсвии с требованиями наследника в то же время оставляя ее в неизменном состоянии.



Родительская сущность будет служить абстрактным базовым классом, который может быть испльзован повторно с дополнительными возможностями посредством **Наследования**. Однако, оригинальная сущность закрыта позволяя программе быть одновременно, как закрытой, так и открытой. Преимущество OCP заключается в том, оно минимализирует программный риск, когда программист добавляе новые сферы использования для сущности. Вместо того, чтобы перерабатывать базовый клас в соответствии с требованиями изменяющимися в процессе использования, программист создает производнй класс отдельно от классов текущего представления ПО. Таким образом, программист может работать с этим уникальным

производным классом не опасаясь, что это может повлять на родительский, либо любой другой производный класс.

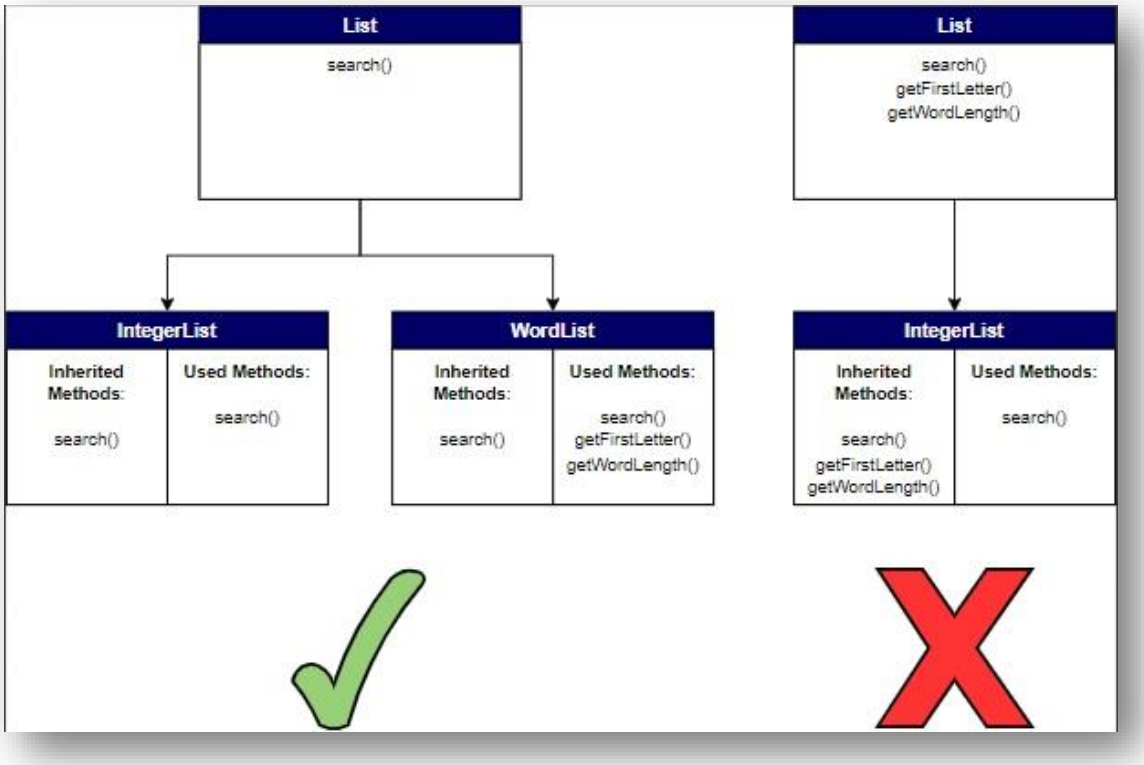
- **Принцип Подстановки Лисков LSP:**



“Объекты ПО долны быть взаимозаменяемы их субтипами без модификации самой программы”. Иными словами, каждый субкласс должен выполнять все поведение базового класса. Преимущество LSP состоит в том, что он ускоряет разработку новых субклассов, также всех субклассов того же типа соответствующие использованию.

- **Принцип Изоляции Интерфейса ISP:** один из принципов объектно-ориентированного проектирования, позволяющих разработчикам исключить недостатки проекта, сформировав наилучший проект на основе имеющегося набора свойств. Утверждает, что правильнее использовать множество специализированных интерфейсов, чем сгруппировывать абсолютно несвязанные между собой методы в один

интерфейс.



- **Принцип инверсии зависимостей DIP:** один из принципов объектно-ориентированного проектирования, позволяющих разработчикам исключить недостатки проекта, сформировав наилучший проект на основе имеющегося набора свойств. Соблюдение принципа инверсии зависимостей необходимо для обеспечения заменяемости объектов без правок по всему исходному коду и для ослабления зависимостей в коде. Когда у вас есть класс A, имеющий указатель на класс B, – классы считаются сильно связанными. Для замены класса B на любой другой

придётся исправлять код класса A, – что не совсем хорошо.

Предлагается вывести интерфейс класса B, назовем его IB, и заменить тип указателя в классе A на IB. Таким образом, зависимость A->B заменилась на A->IB<-B. Теперь можно вместо B использовать любую другую реализацию интерфейса IB.

Источник: S.O.L.I.D. Principles of Object-Oriented Programming in C# (educative.io)