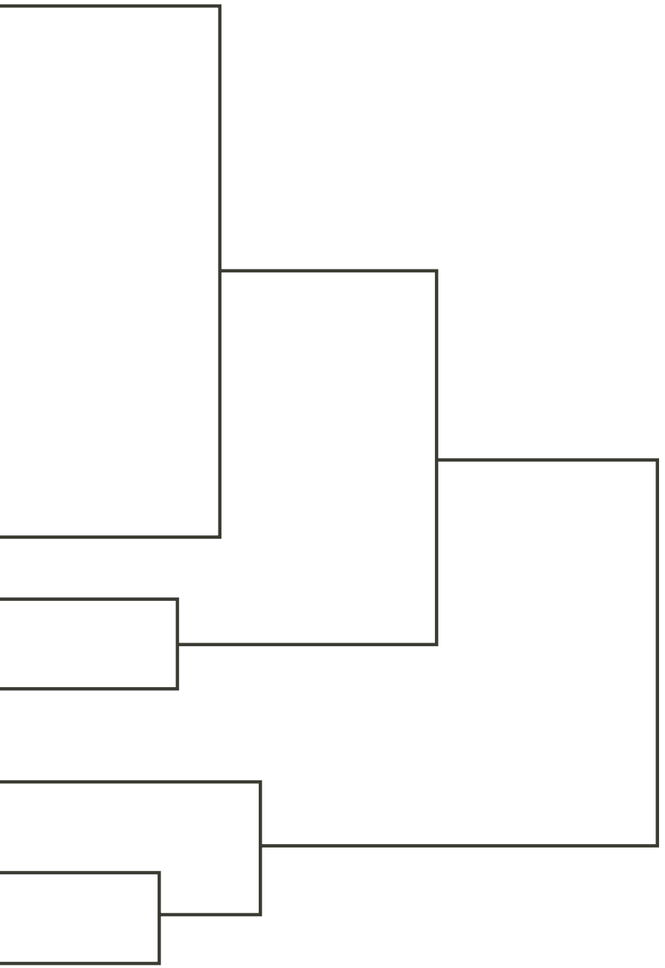
Паттерны и практики написания кода

SOLID-принципы. Часть 3

принцип подстановки Лисков *Liskov Substitution Principle (LSP)*



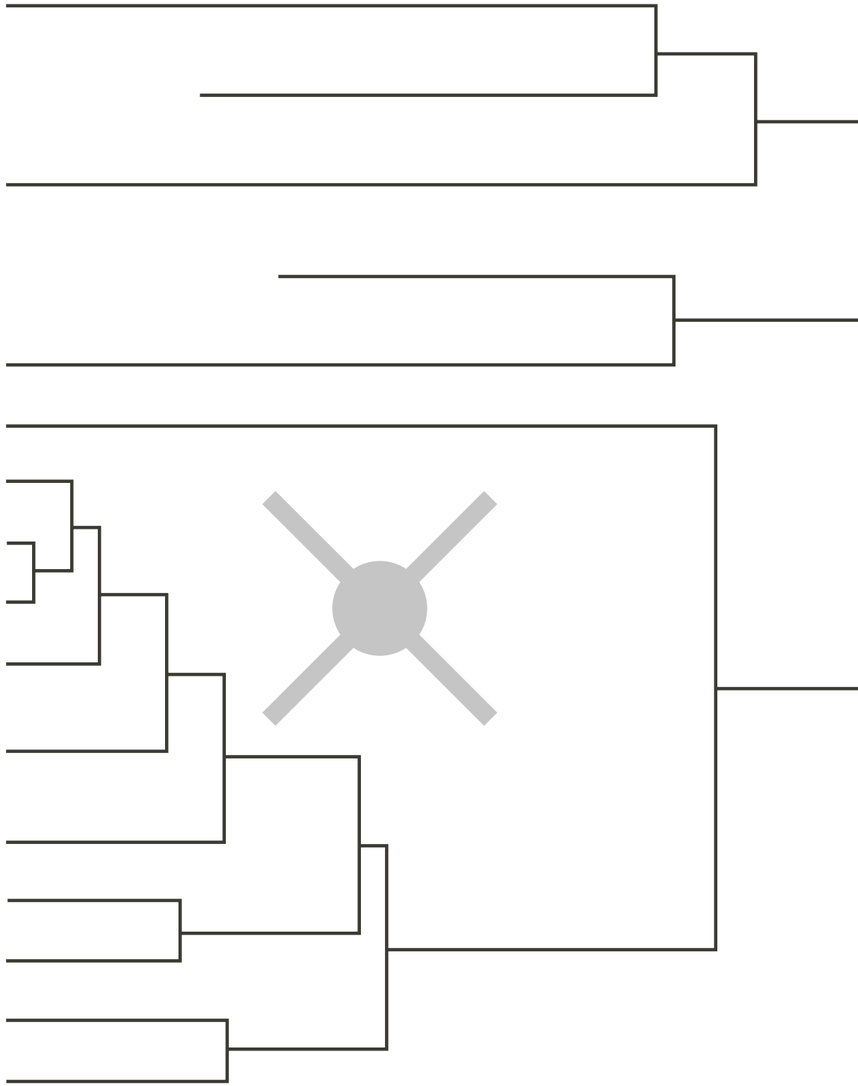
Определение сформулировала математик *Барбара Лисков* в 1988 году. *Роберт Мартин* упростил его: **«Подтипы должны быть взаимозаменяемыми для своих базовых типов».**

Для php определение можно трансформировать к такому варианту: **Наследники должны быть взаимозаменяемыми для своих родительских классов.**

ЦЕЛЬ ПРИНЦИПА
сделать правильное наследование классов

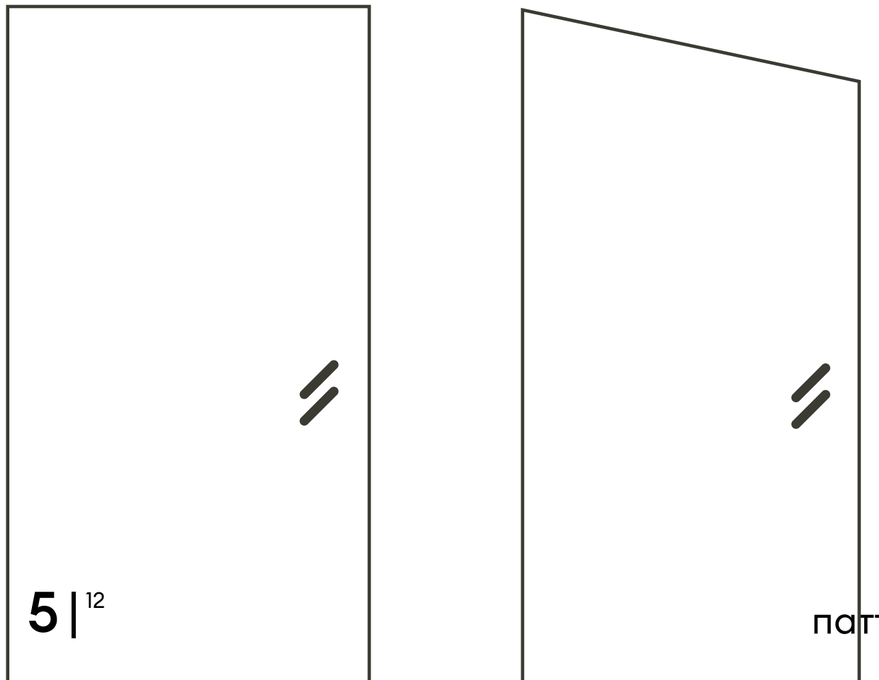
используемые методы и правила

- **по способам действия наследование не должно менять поведение исходного класса.** При нарушении получаем сложно уловимые баги
- **модели нельзя рассматривать отдельно от их задач,** иначе они могут стать непригодными
- **правила в реальном и ООП-мире могут отличаться.** Из-за разных правил в двух похожих мирах мы можем получать разные результаты, а это приводит к багам



проектируемые методы по контракту

→ **необходимые ограничения на входные условия.** Наследуемый класс должен иметь точно такие же ограничения, что и родительский, или не выполнять некоторые из них. Добавлять в наследнике ограничения, отсутствующие в родительском классе нельзя.



← **необходимые ограничения на выходные условия.** Наследуемый класс должен иметь точно такие же ограничения, что и родительский, а также может добавлять свои. Убирать в наследнике ограничения, которые присутствуют в родительском классе, нельзя.

задачи метода с точки зрения контракта

- **Входные параметры включают в себя аргументы и их последующую проверку в теле метода (валидаторы)**
- **После выполнения всех входных проверок идёт бизнес-логика.** Её реализация не влияет на входные и выходные условия.
- **За выходные параметра отвечает языковая конструкция return и возвращаемый тип данных, указанный в сигнатуре метода.** Здесь важно помимо аргументов проверить состояние свойств класса. Если при выполнении метода они изменяются, проверяем их в контракте.

ИТОГ

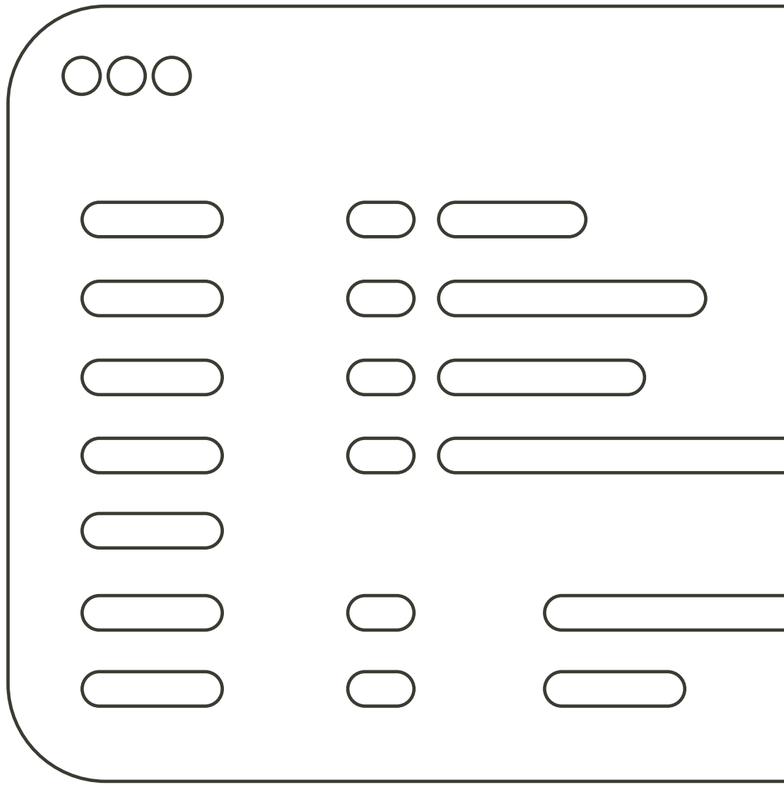
Наследование не должно менять поведение исходного класса. Добиваемся этого благодаря проектированию по контракту. Это формализует и регламентирует условия работы всех методов системы.

принцип разделения интерфейсов

Interface Segregation Principle (ISP)

ОПРЕДЕЛЕНИЕ

Клиенты не должны вынужденно зависеть от методов, которыми они не пользуются. Чтобы разорвать связь между классами, нужно поместить между ними абстракции, то есть интерфейсы. Мы создаем маленькие интерфейсы, нужные потребителям. Обычно классы содержат большое число методов, а клиентам нужна только определённая часть функциональности.



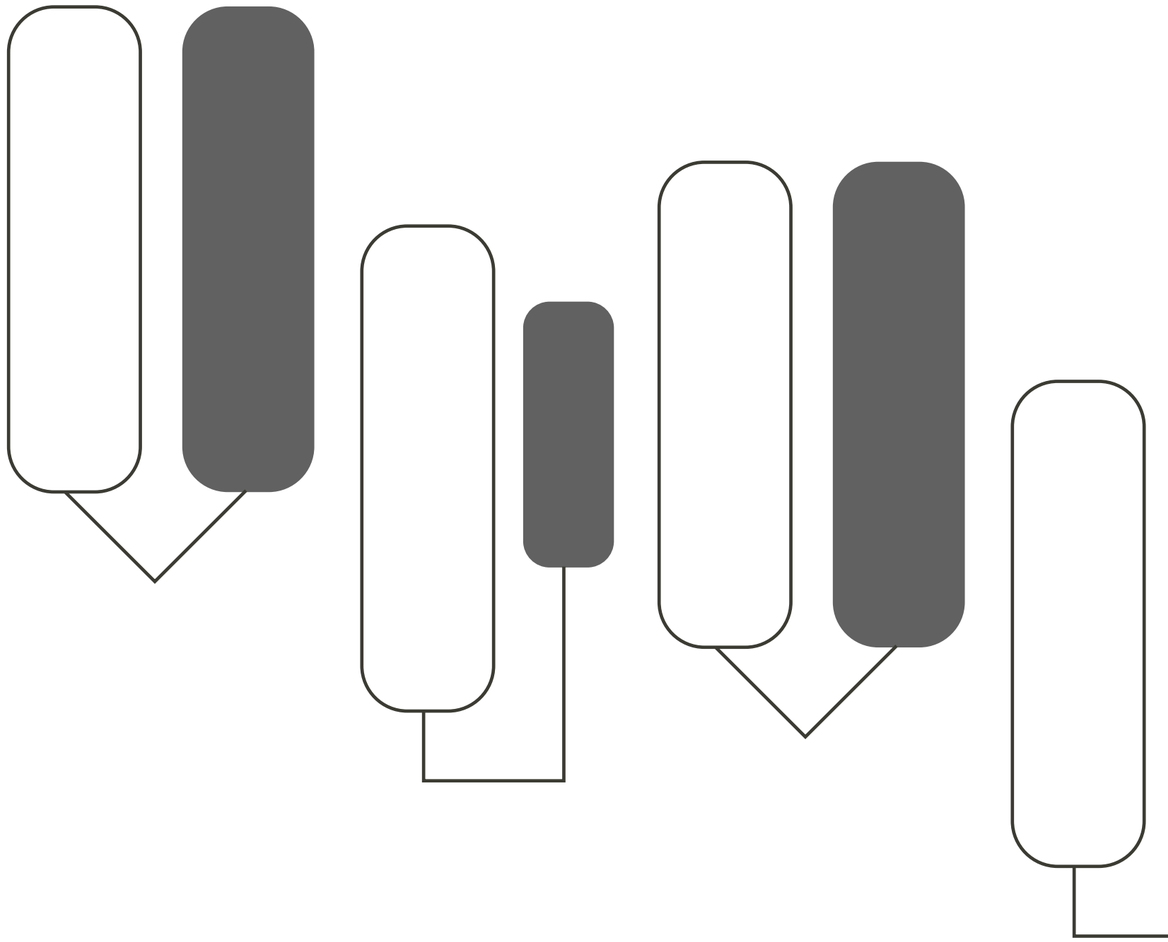
плюсы принципа

+ **Он уменьшает связанность кода и позволяет легко подменять один класс другим.** Этот подход прикладной, с ним принимаем проектные решения.

- **Можно сказать, что это принцип персональной ответственности, но для интерфейсов.** Мы концентрируем минимально необходимый набором методов для клиентского класса, ненужные ему могут использоваться в других сценариях.

клиентам не нужно знать о методах, которые написаны не для них

При смешении сценариев растет риск использовать то, что для них не предназначено.



ЕЩЕ СОВЕТ

Имена интерфейсов должны быть универсальными и не привязываться к реализации.

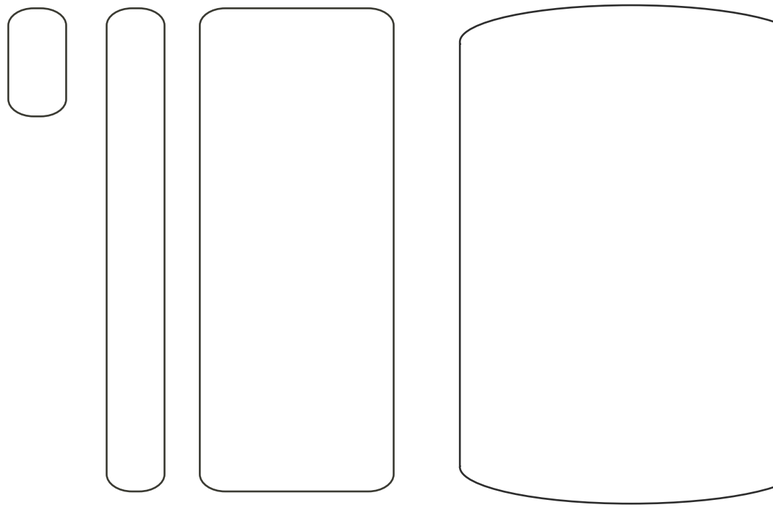
сложности имплементации интерфейса с большим количеством целевых классов

- **Добавление нового метода в интерфейс обязывает реализовать их во всех целевых классах,** а он может быть не свойственен для сценария и в целом плохо подходить для новых реализаций.

- **Неизвестно, как реализовывать в классе не свойственные для него методы.** Если целевые классы используются в разных сценариях, то часть методов им может не понадобится.

к чему ведёт правильное применение разделения интерфейса

- → **Интерфейсы прерывают зависимость классов друг от друга.** То есть полноценно удовлетворяют low coupling.
- → **Теперь нет методов, которые не свойственны или не нужны конкретной абстракции.** При этом клиентский класс имеет доступ только к тем методам, которые есть в абстракции.
- → **Наследник теперь может реализовывать свои методы, а клиент при этом не обязан о них знать.**



ИТОГ

Много специализированных интерфейсов лучше, чем один универсальный.


avito.tech