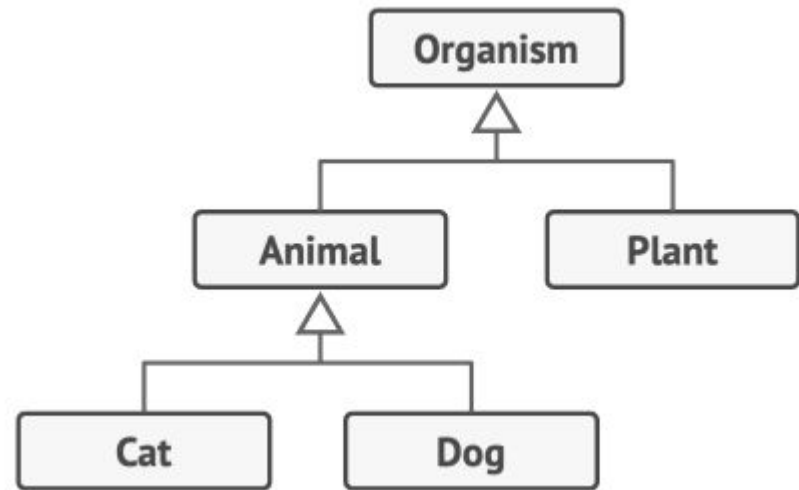
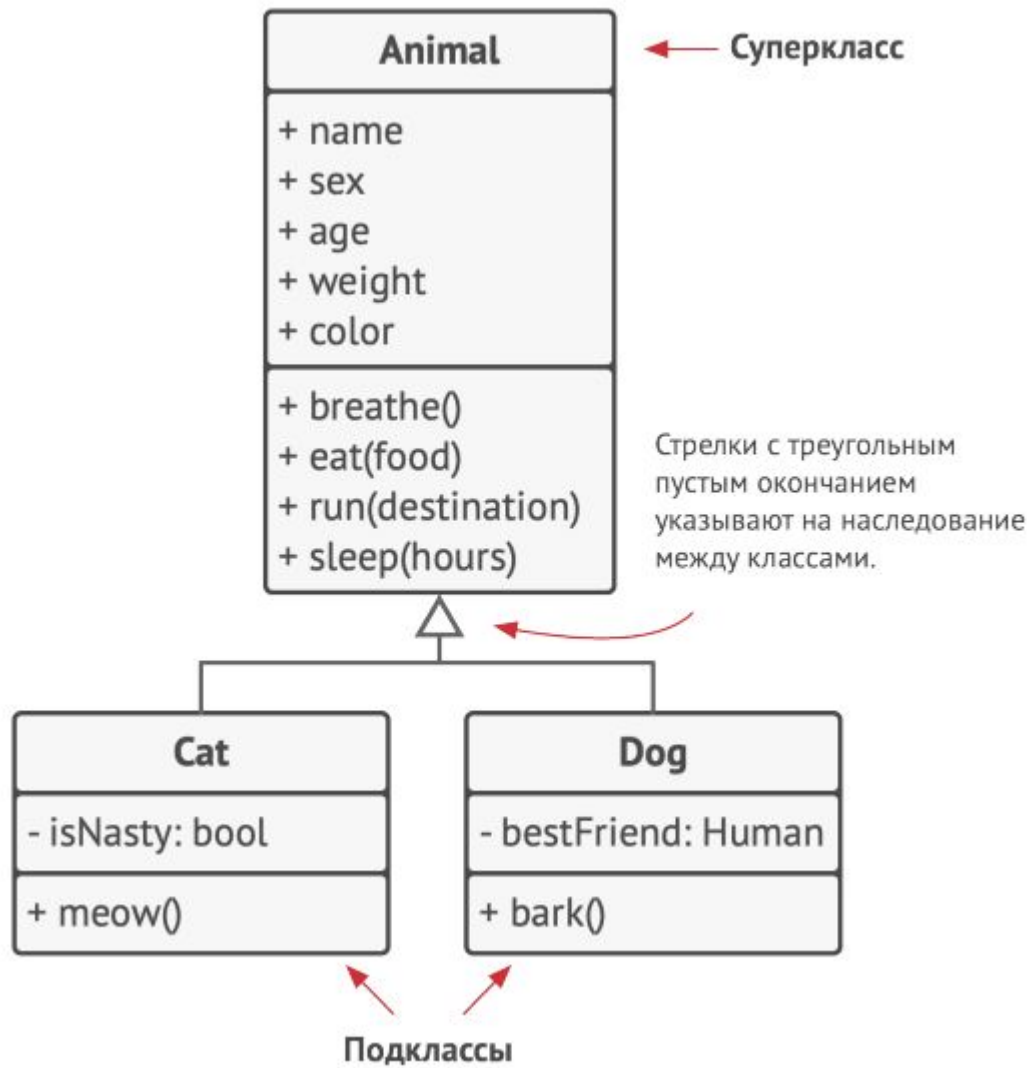


+	Публичный (Public)
-	Приватный (Private)
#	Защищённый (Protected)
/	Производный (Derived) (может быть совмещён с другими)
~	Пакет (Package)





Abstraction

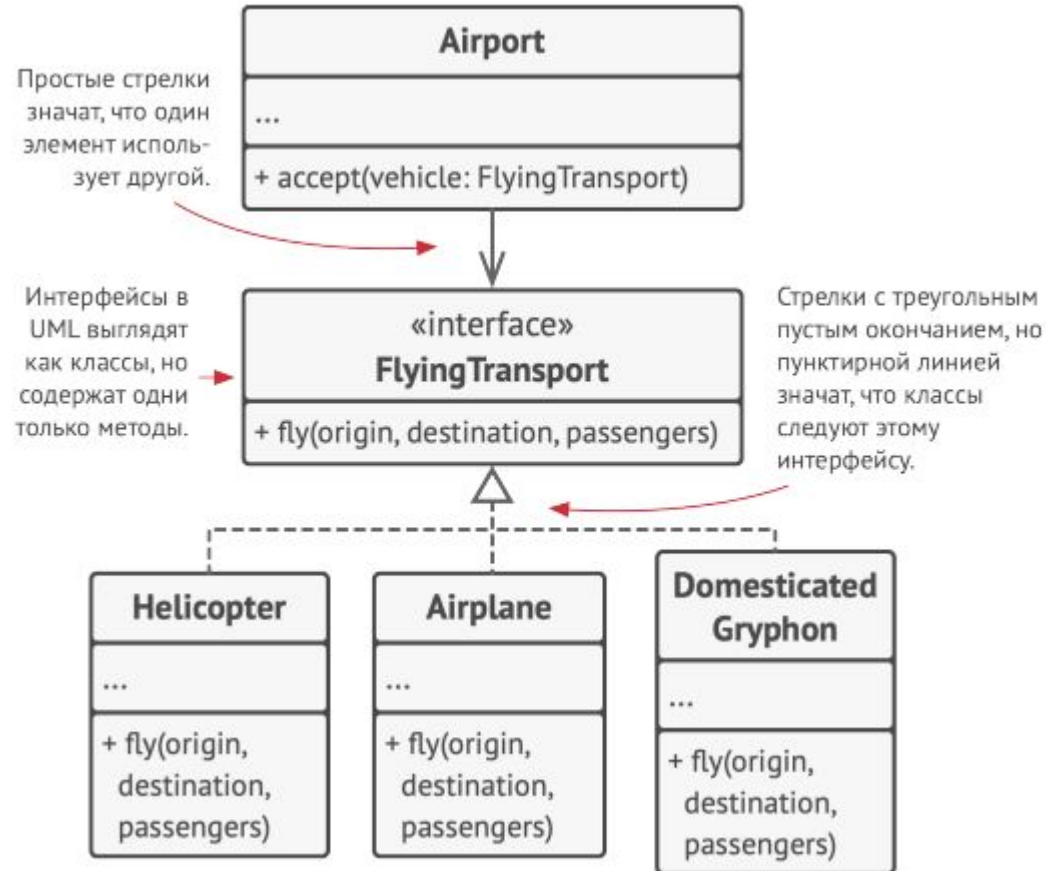


Airplane
- speed - altitude - rollAngle - pitchAngle - yawAngle
+ fly()

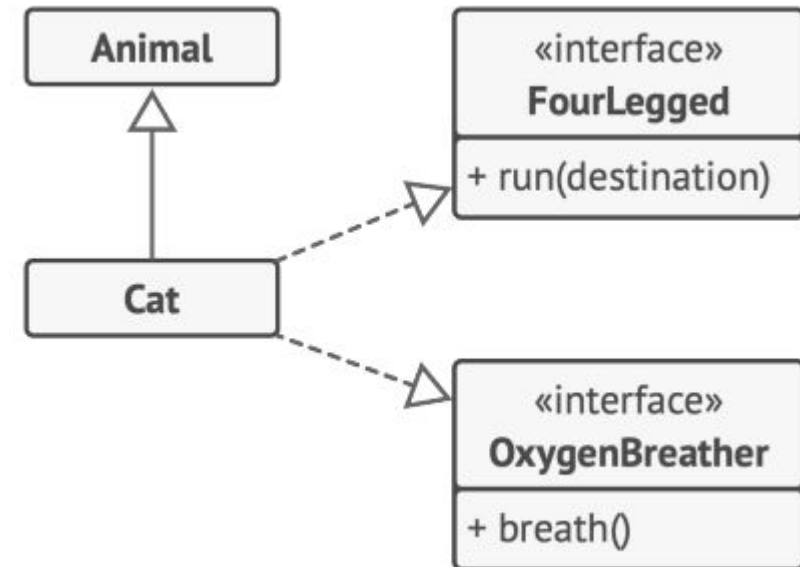


Airplane
- seats
+ reserveSeat(n)

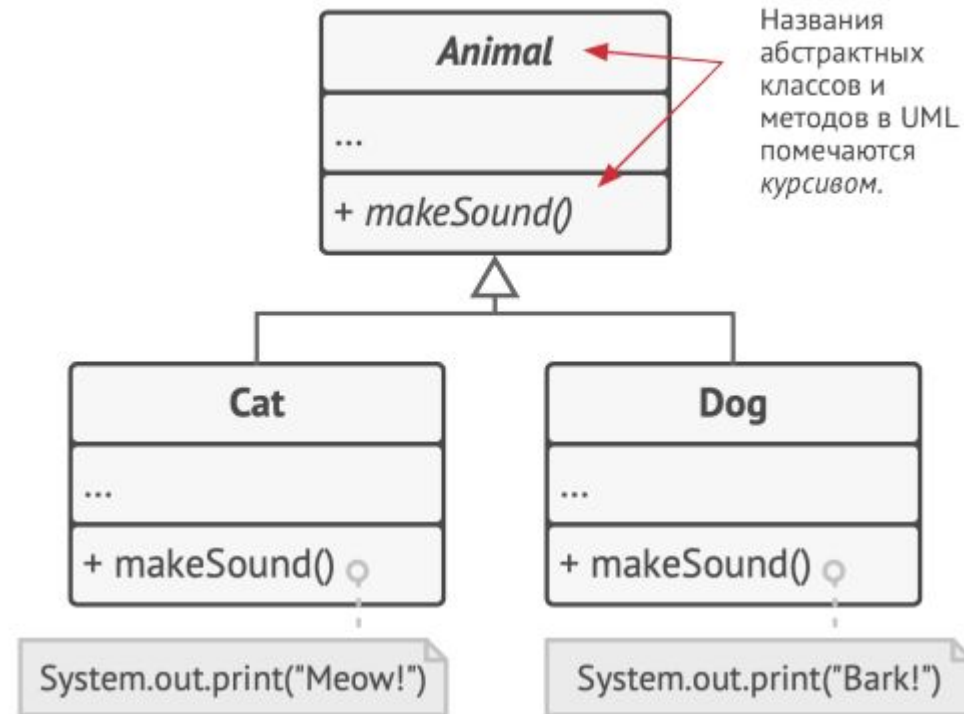
Инкапсуляция



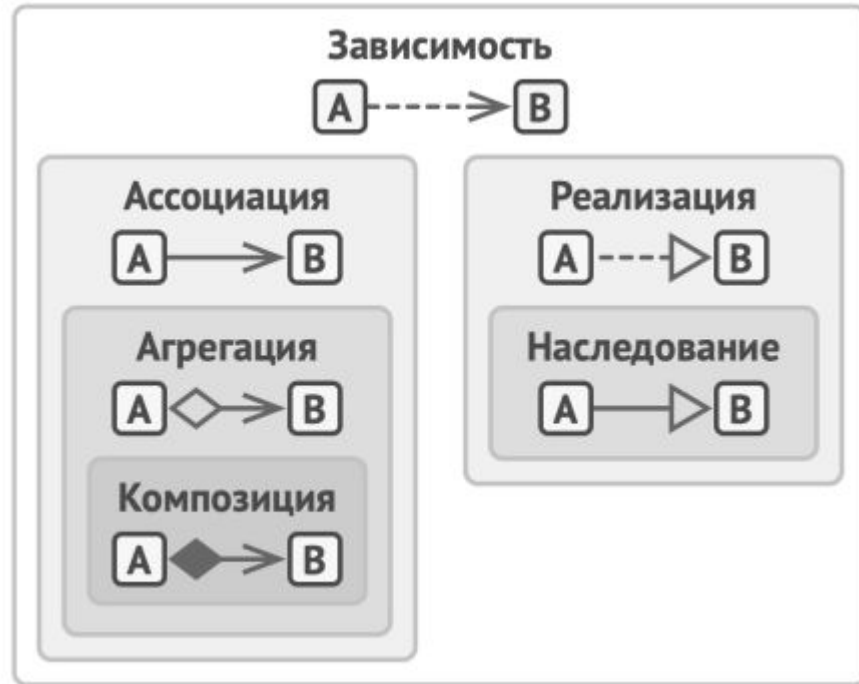
Наследование



Полиморфизм



Отношения между объектами



Ассоциация



Агрегация

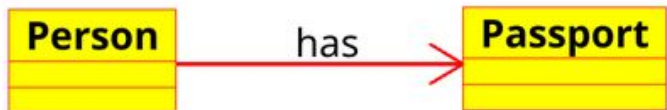
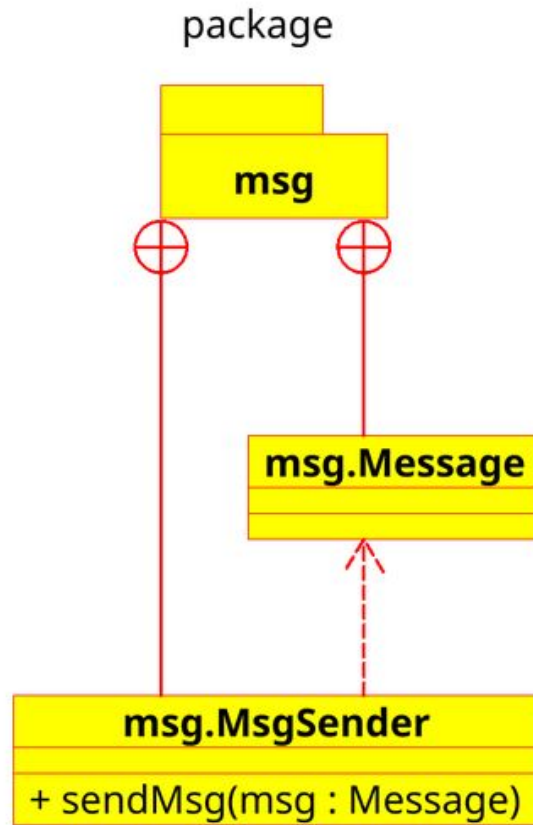
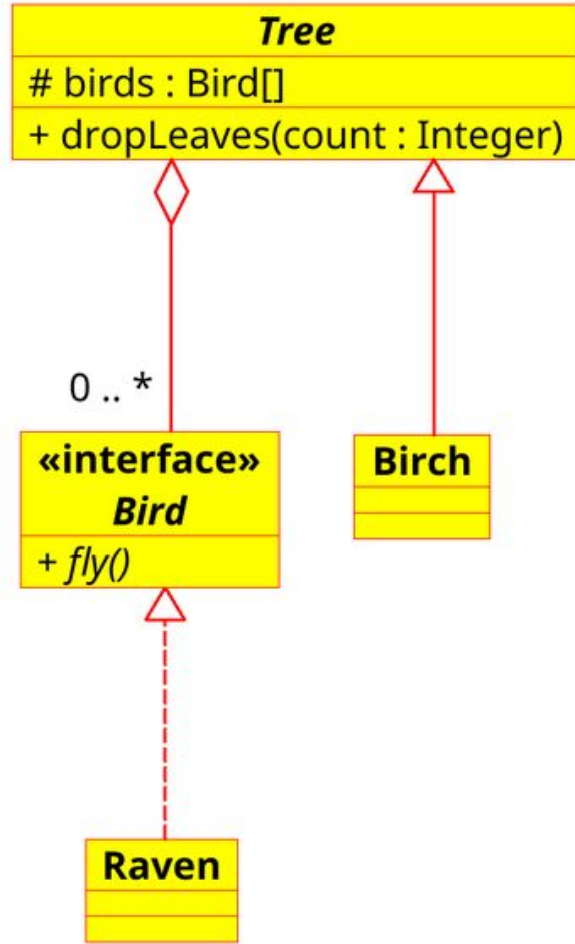
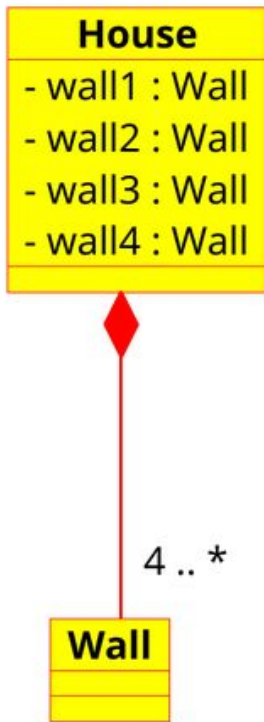


Композиция



Зависимость





Паттерны проектирования

- **Порождающие паттерны** беспокоятся о гибком создании объектов без внесения в программу лишних зависимостей.
- **Структурные паттерны** показывают различные способы построения связей между объектами.
- **Поведенческие паттерны** заботятся об эффективной коммуникации между объектами.

Базовые принципы проектирования

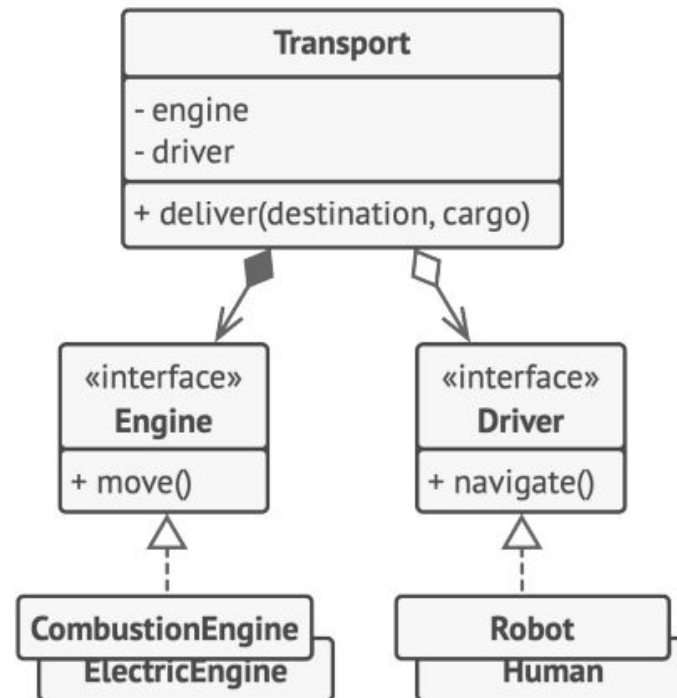
Инкапсулируйте то, что меняется

Определите аспекты программы, класса или метода, которые меняются чаще всего, и отделите их от того, что остаётся постоянным.

Программируйте на уровне интерфейса

Программируйте на уровне интерфейса, а не на уровне реализации. Код должен зависеть от абстракций, а не конкретных классов.

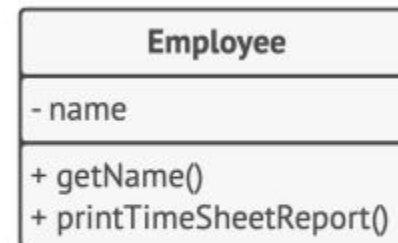
Предпочитайте композицию наследованию



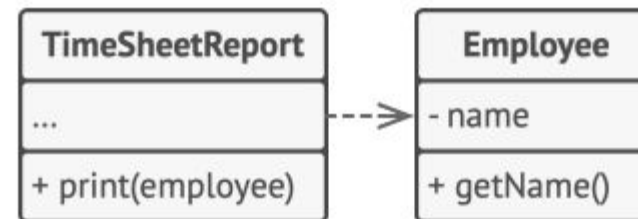
SOLID

S Принцип единственной ответственности **Single Responsibility Principle**

У класса должен быть только один мотив для изменения.



ДО: класс сотрудника содержит разнородные поведения.



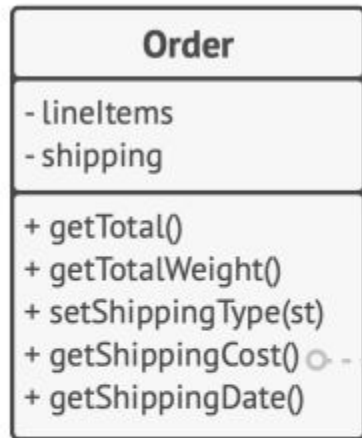
ПОСЛЕ: лишнее поведение переехало в собственный класс.

O

Принцип открытости/закрытости

pen/Closed Principle

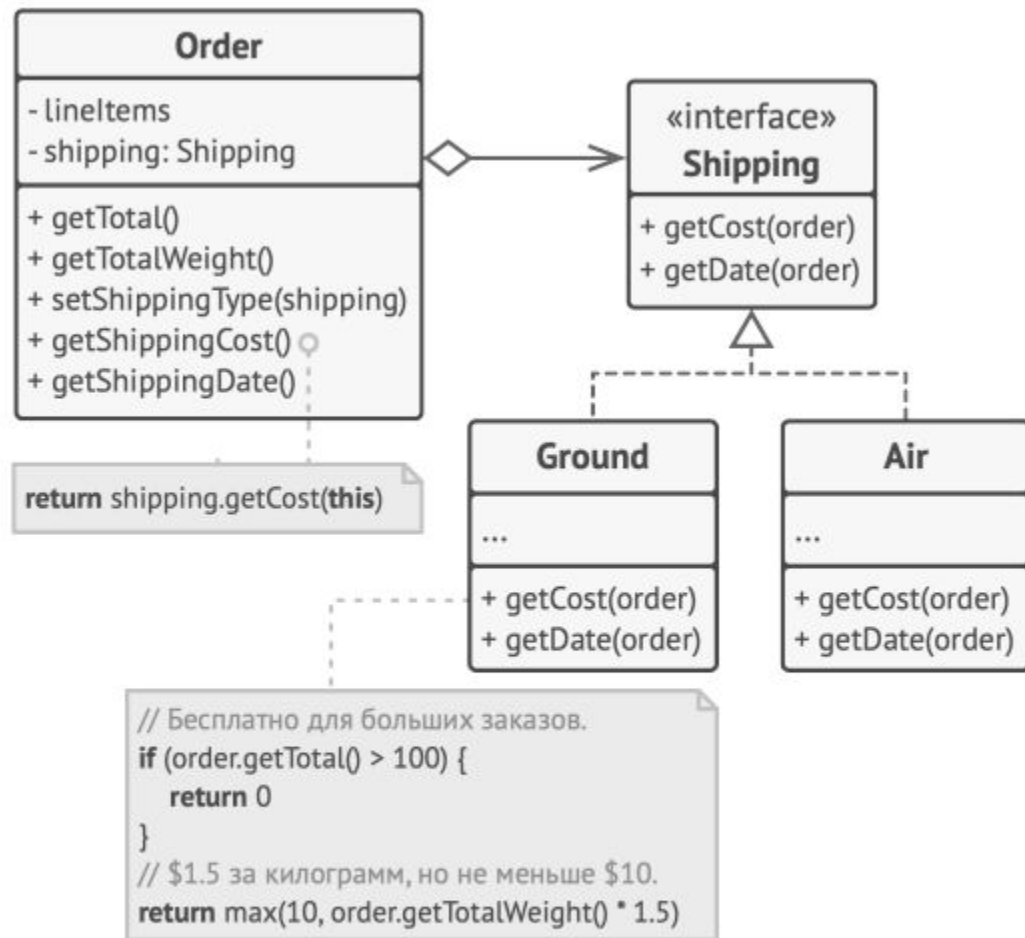
Расширяйте классы, но не изменяйте их первоначальный код.



```
if (shipping == "ground") {
    // Бесплатно для больших заказов.
    if (getTotal() > 100) {
        return 0
    }
    // $1.5 за килограмм, но не меньше $10.
    return max(10, getTotalWeight() * 1.5)
}

if (shipping == "air") {
    // $3 за килограмм, но не меньше $20.
    return max(20, getTotalWeight() * 3)
}
```

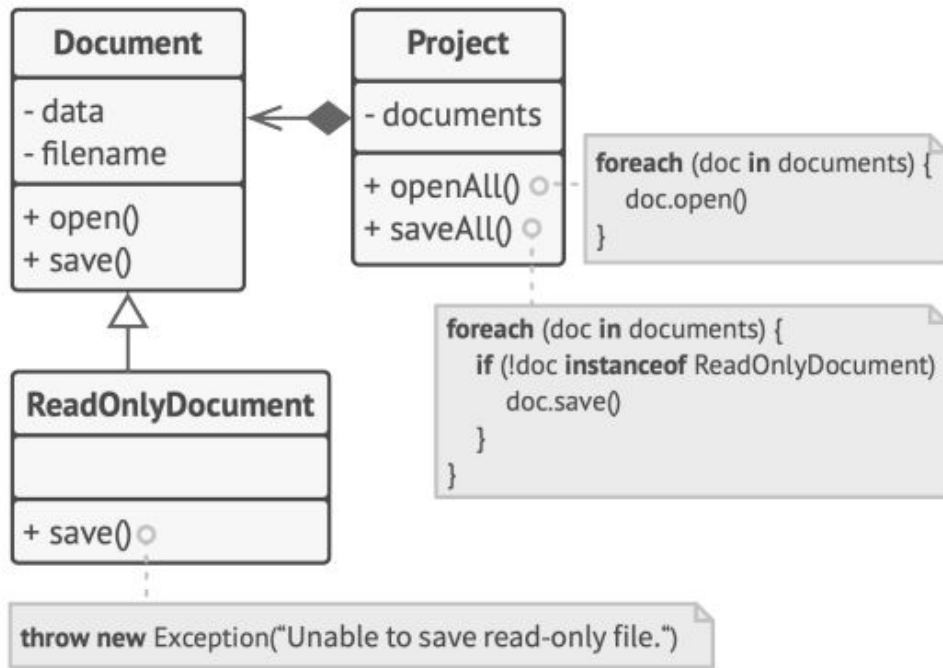
ДО: код класса заказа нужно будет изменять при добавлении нового способа доставки.



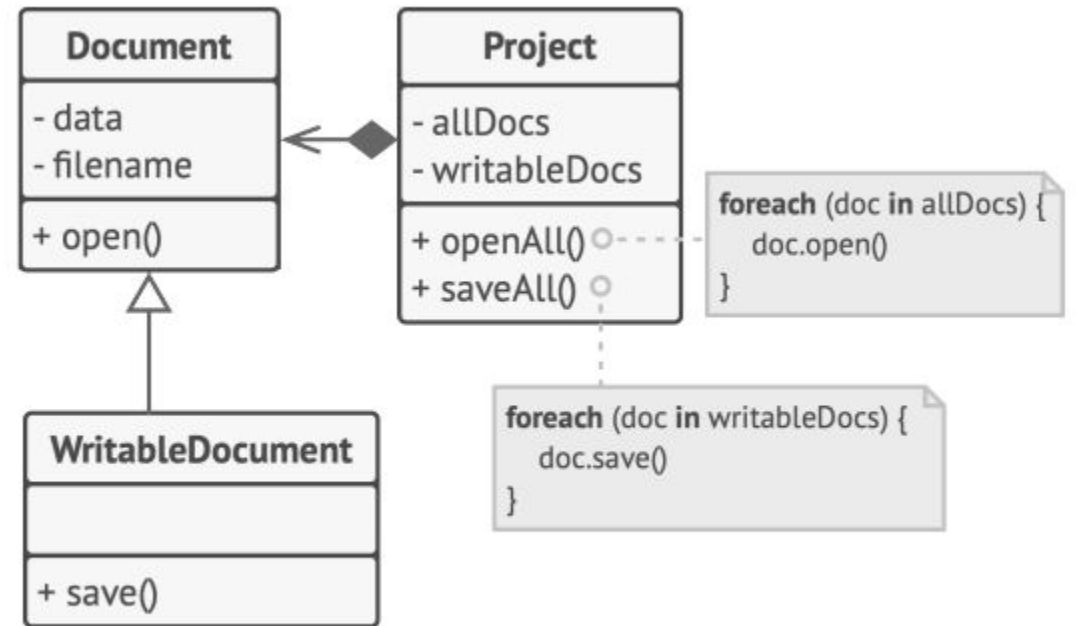
ПОСЛЕ: новые способы доставки можно добавить, не трогая класс заказов.

L Принцип подстановки Лисков¹ iskov Substitution Principle

Подклассы должны дополнять, а не замещать поведение базового класса.



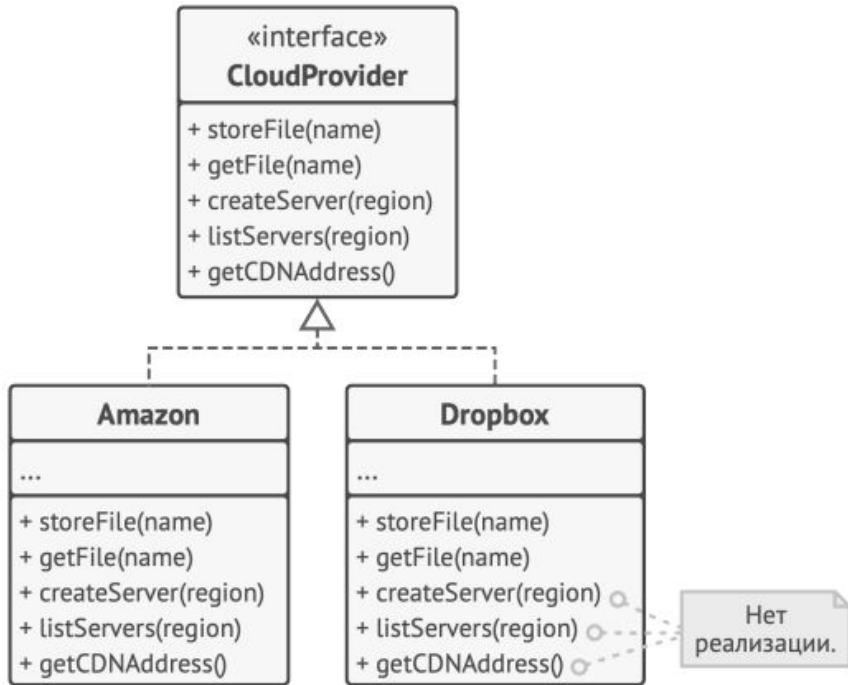
ДО: подкласс «обнуляет» работу базового метода.



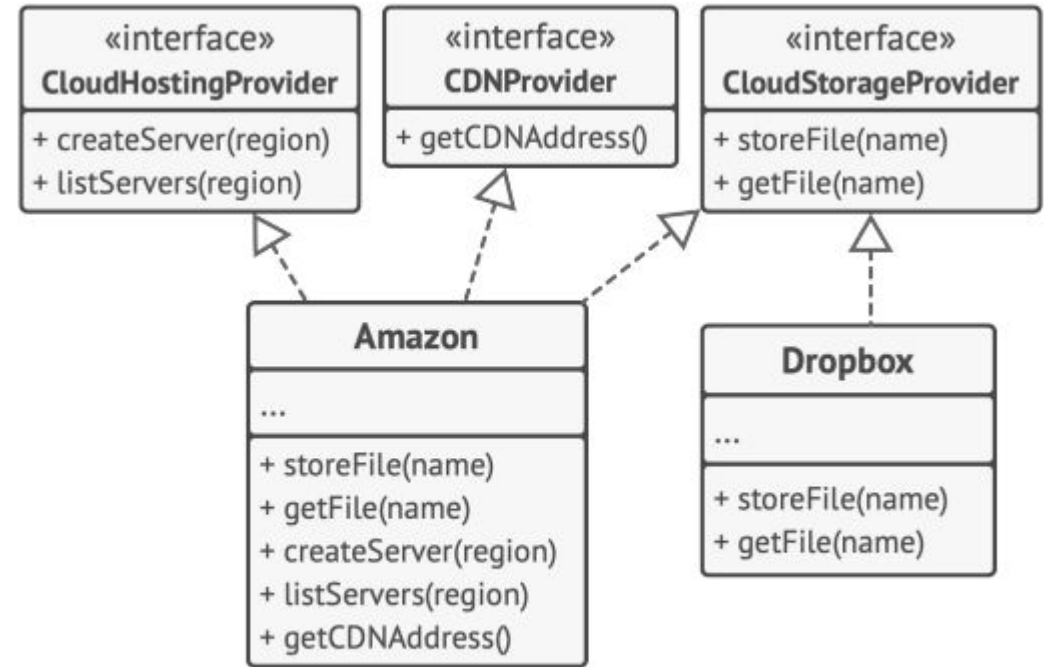
ПОСЛЕ: подкласс расширяет базовый класс новым поведением.

Принцип разделения интерфейса Interface Segregation Principle

Клиенты не должны зависеть от методов, которые они не используют.



ДО: не все клиенты могут реализовать операции интерфейса.



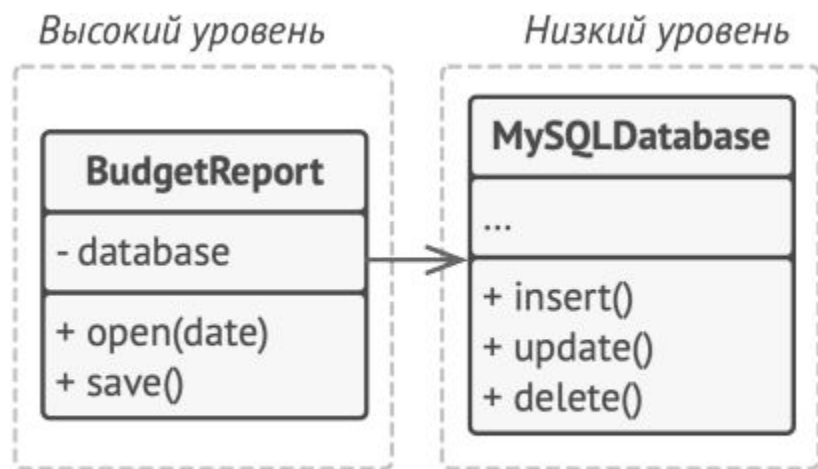
ПОСЛЕ: раздутый интерфейс разбит на части.

D

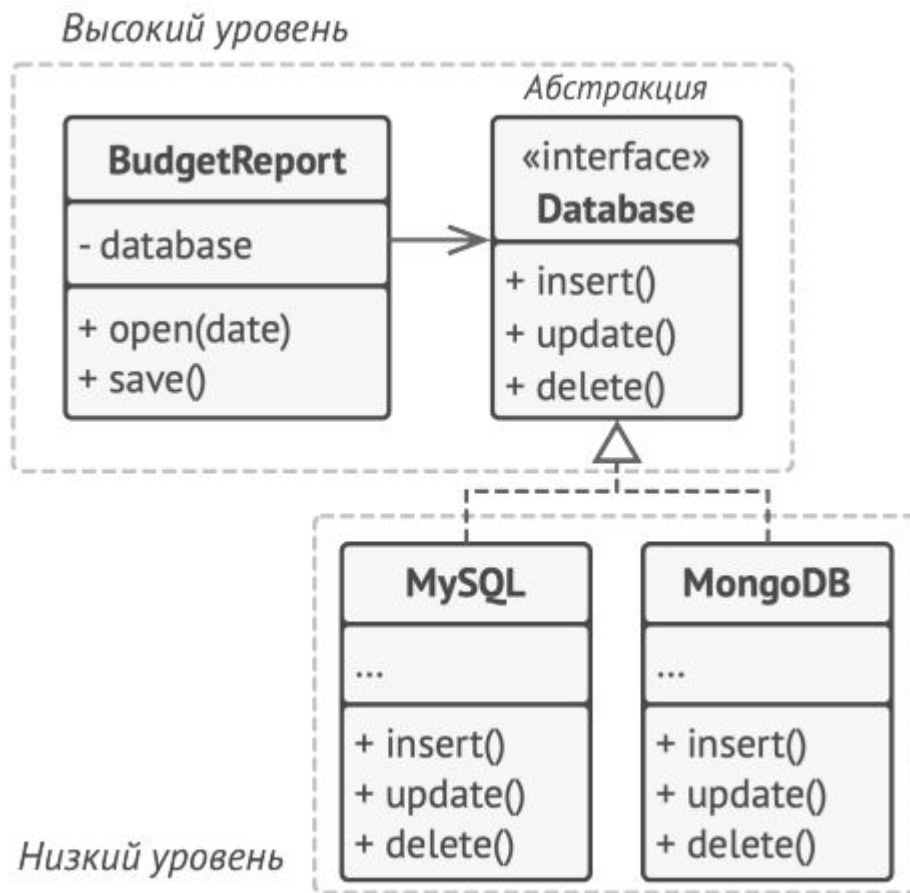
Принцип инверсии зависимостей

Dependency Inversion Principle

Классы верхних уровней не должны зависеть от классов нижних уровней. Оба должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

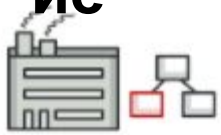


ДО: высокоуровневый класс зависит от низкоуровневого.



ПОСЛЕ: низкоуровневые классы зависят от высокоуровневой абстракции.

Порождающ ие



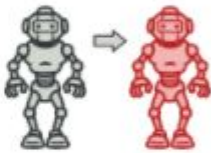
Фабричный метод
Factory Method



Абстрактная фабрика
Abstract Factory



Строитель
Builder

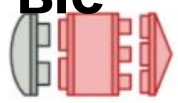


Прототип
Prototype



Одиночка
Singleton

Структурн ые



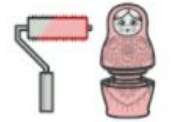
Адаптер
Adapter



Мост
Bridge



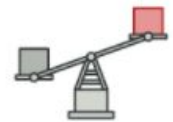
Компоновщик
Composite



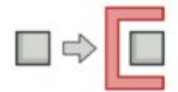
Декоратор
Decorator



Фасад
Facade



Легковес
Flyweight



Заместитель
Proxy

Поведенческ ие



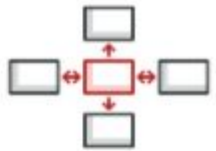
Цепочка обязанностей
Chain of Responsibility



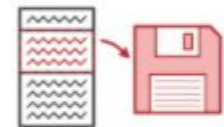
Команда
Command



Итератор
Iterator



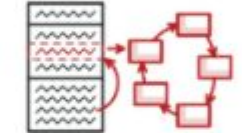
Посредник
Mediator



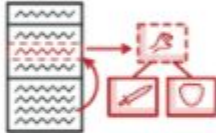
Снимок
Memento



Наблюдатель
Observer



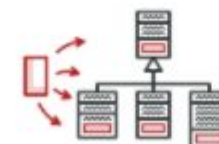
Состояние
State



Стратегия
Strategy



Шаблонный метод
Template Method

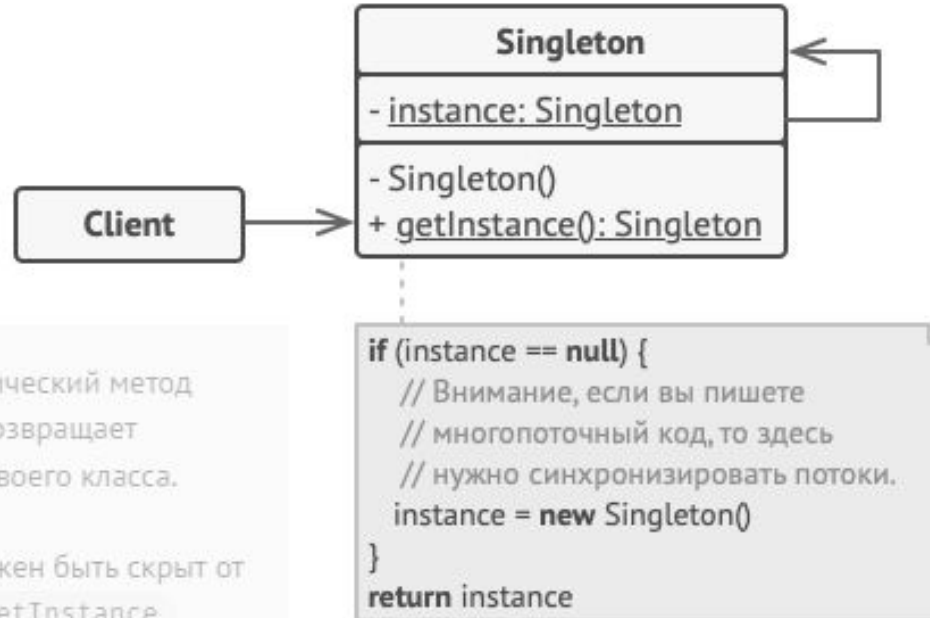


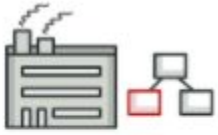
Посетитель
Visitor



Одиночка

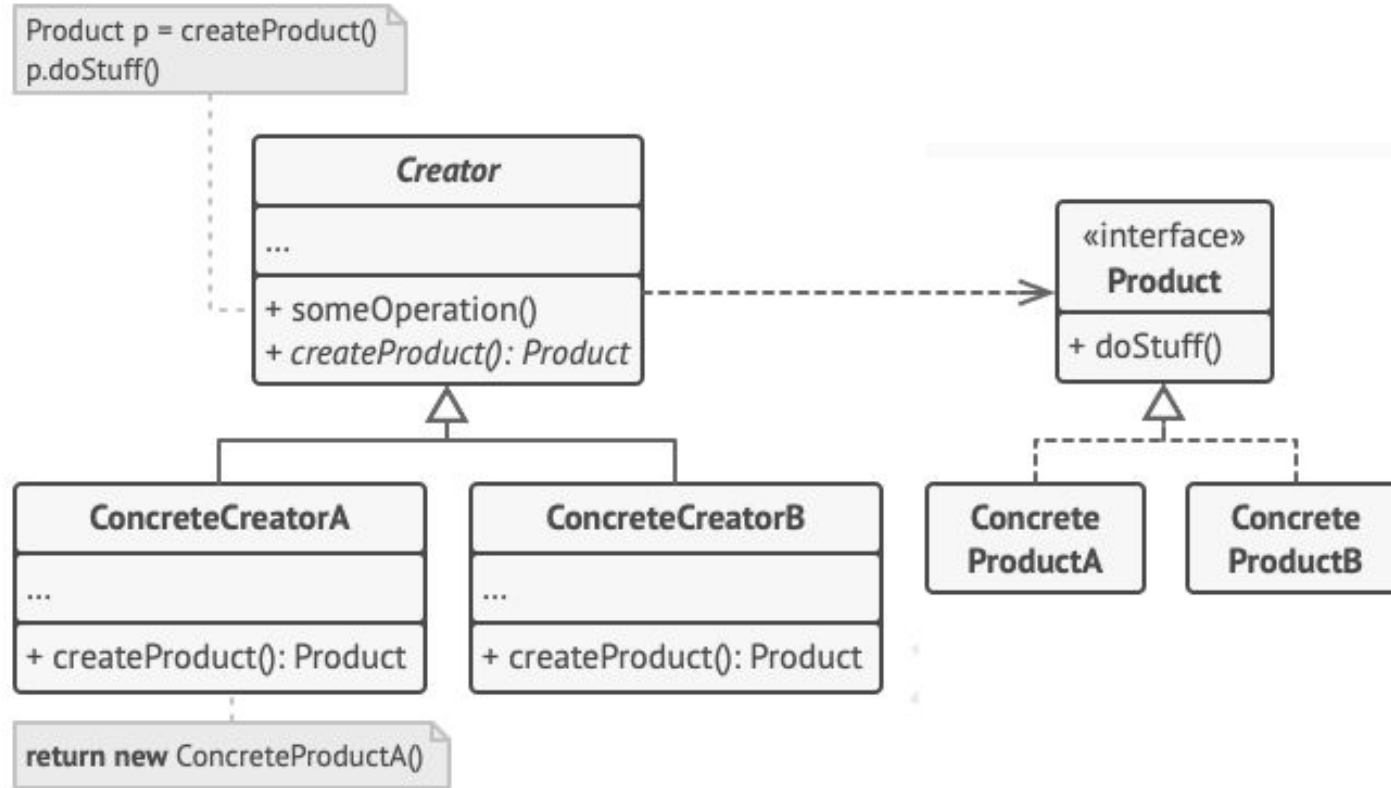
Singleton





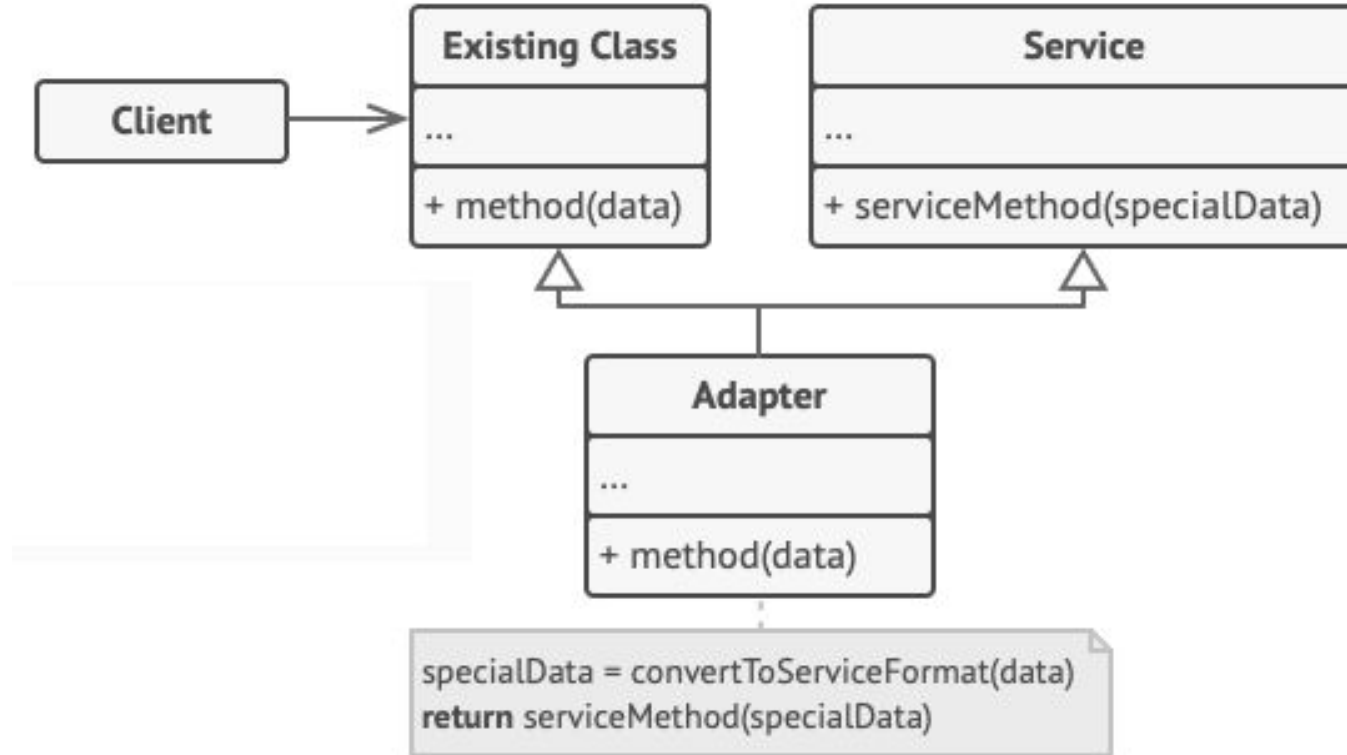
Фабричный метод

Factory Method





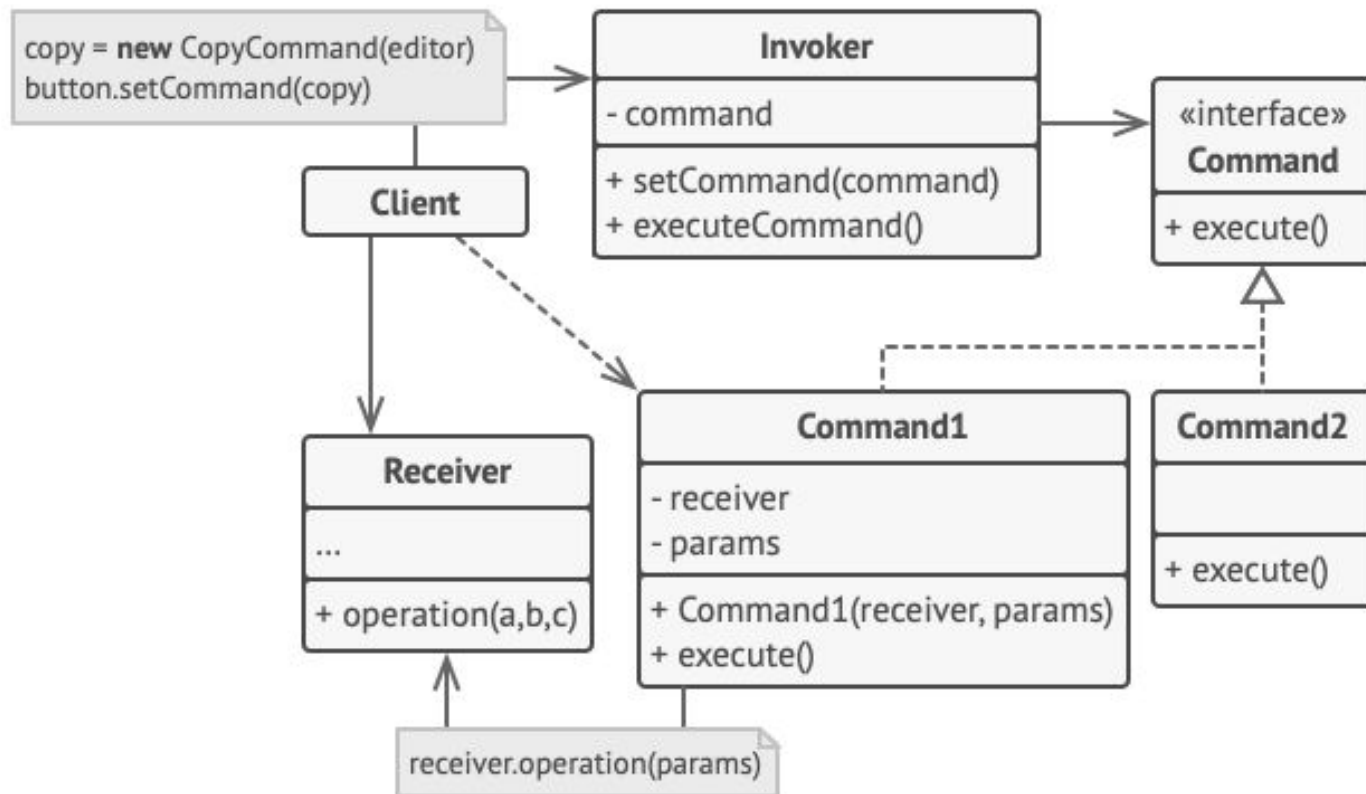
Адаптер
Adapter





Команда

Command



DRY – расшифровывается как Don't Repeat Yourself – не повторяйся, также известен как **DIE** – Duplication Is Evil – дублирование это зло. Этот принцип заключается в том, что нужно избегать повторений одного и того же кода. Лучше использовать универсальные свойства и функции.

KISS – Keep It Simple, Stupid – не усложняй! Смысл этого принципа программирования заключается в том, что стоит делать максимально простую и понятную архитектуру, применять шаблоны проектирования и не изобретать велосипед.