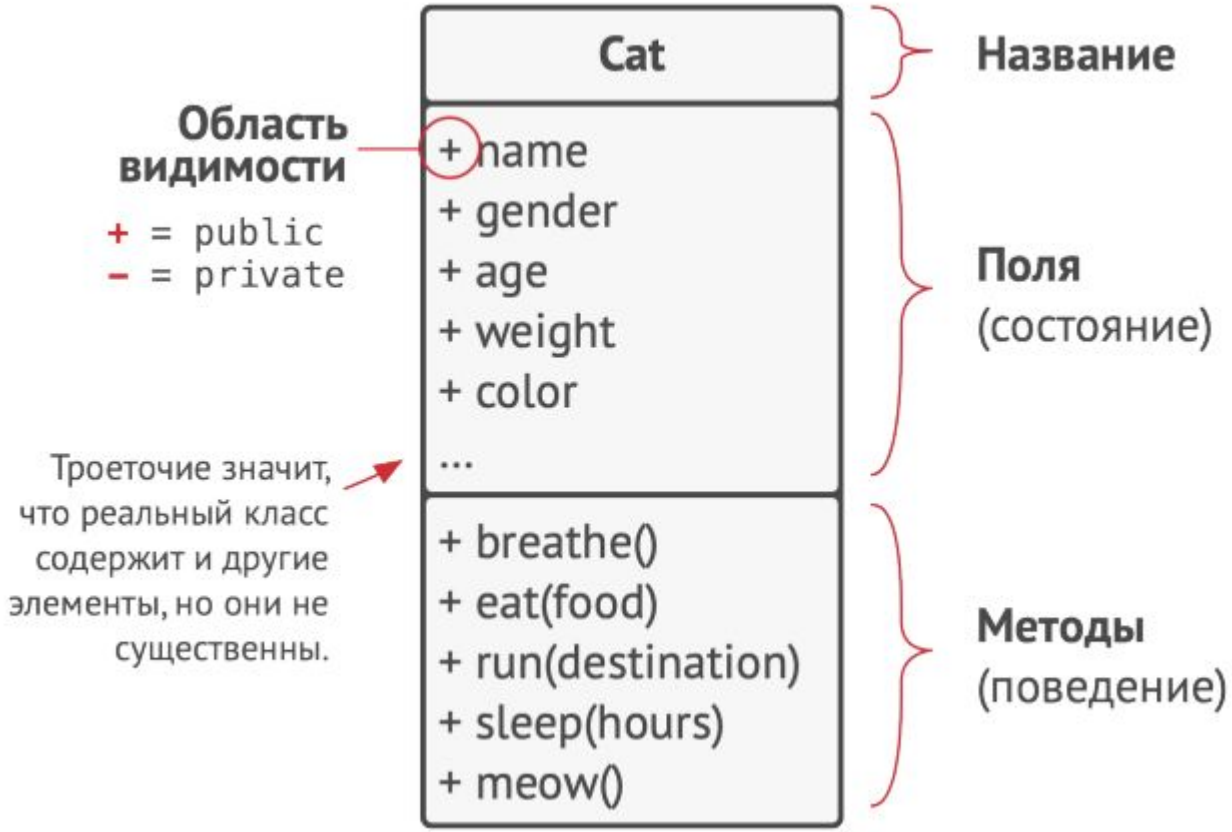
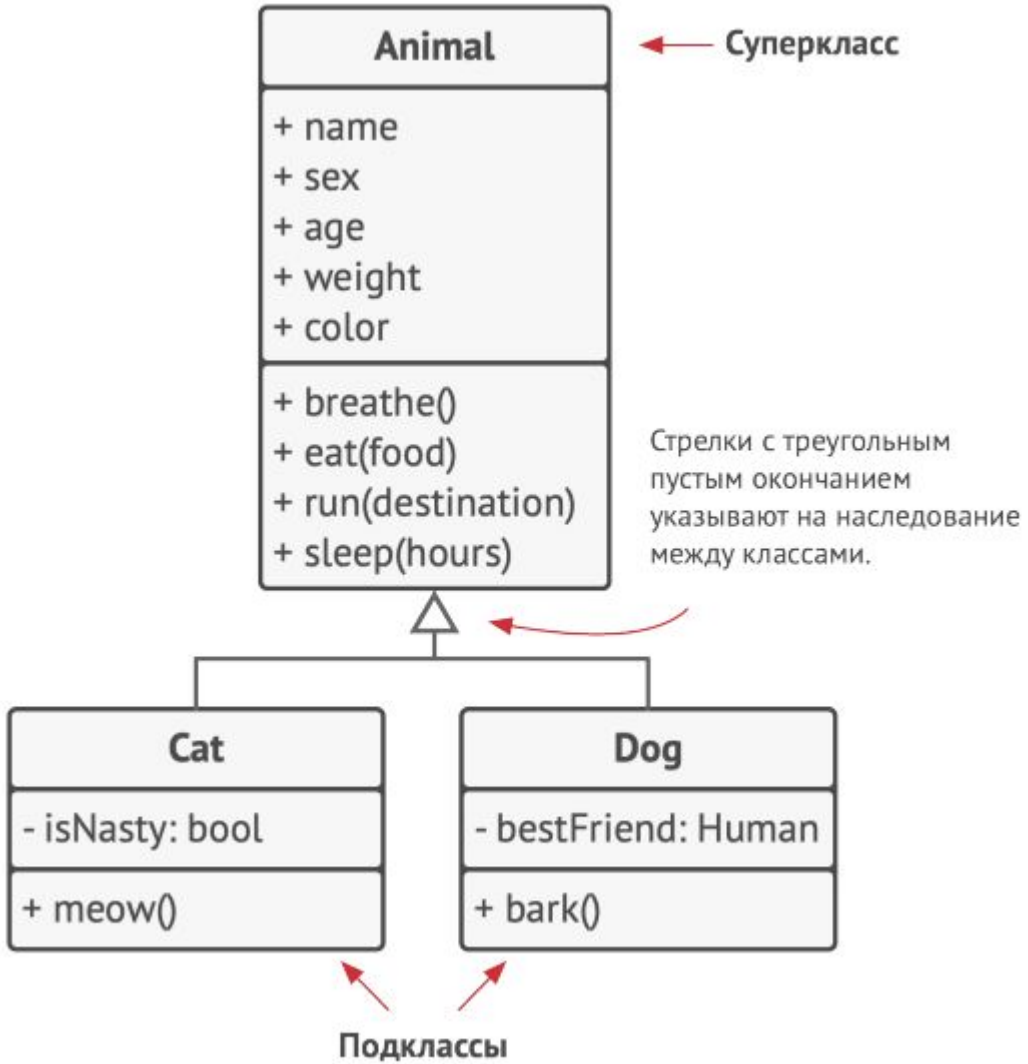
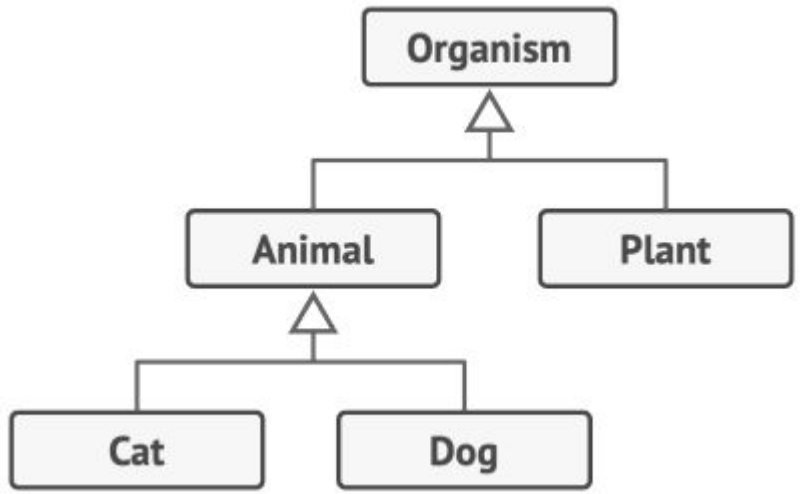


+	Публичный (Public)
-	Приватный (Private)
#	Защищённый (Protected)
/	Производный (Derived) (может быть совмещён с другими)
~	Пакет (Package)


Animal
Суперкласс
+ name
+ sex
+ age
+ weight
+ color
+ breathe()
+ eat(food)
+ run(destination)
+ sleep(hours)
Стрелки с треугольным пустым окончанием указывают на наследование между классами.
Cat
- isNasty: bool
+ meow()
Dog
- bestFriend: Human
+ bark()
Подклассы

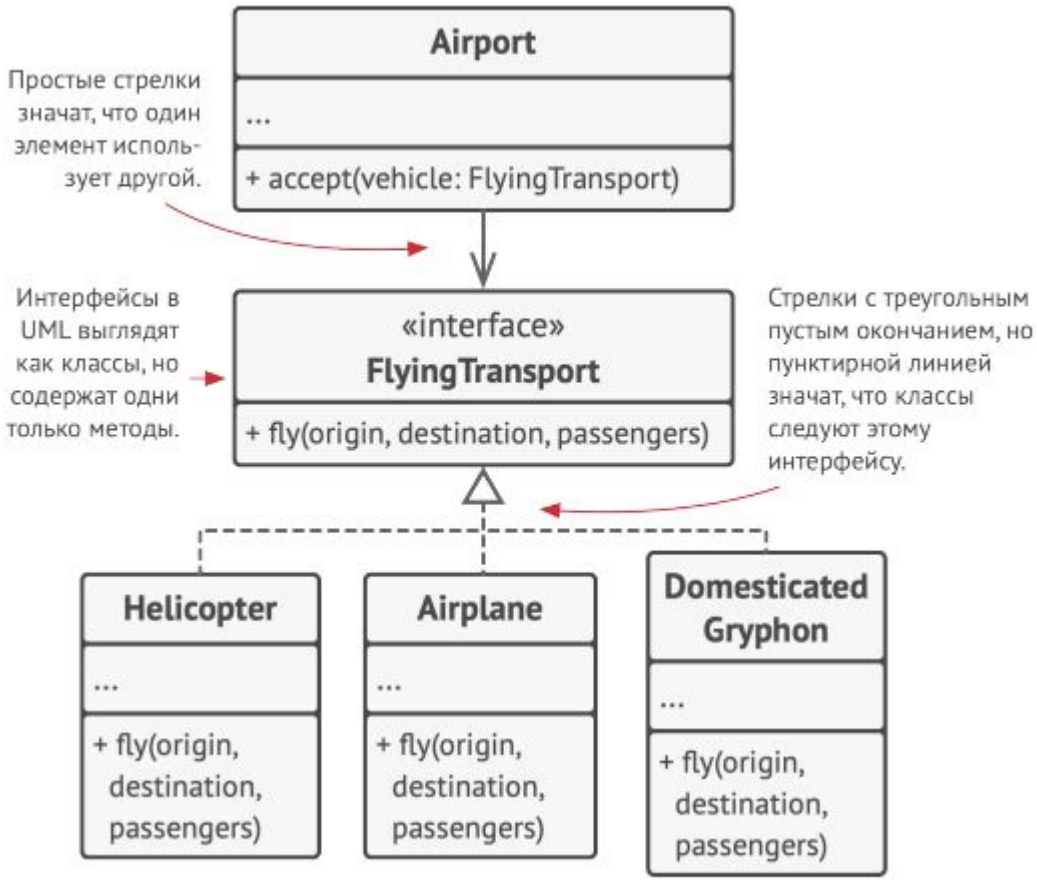

Organism
Animal
Plant
Cat
Dog

Абстракция

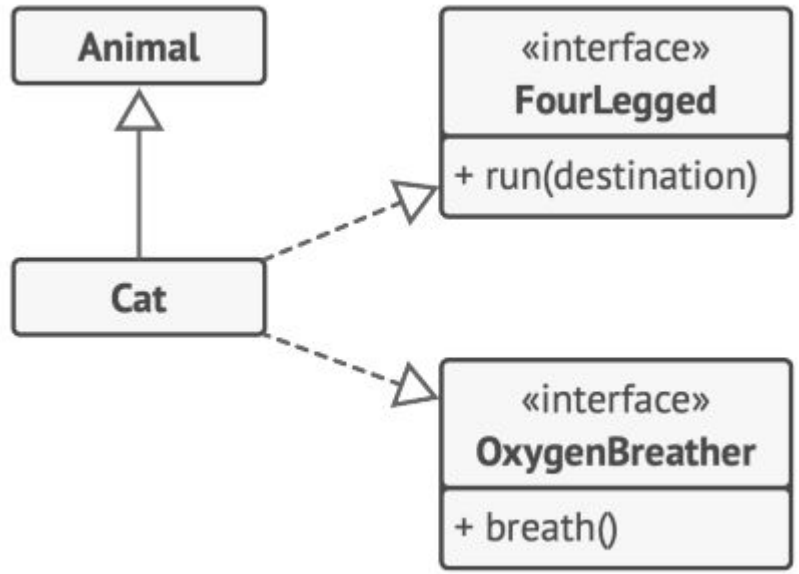
Airplane
- speed - altitude - rollAngle - pitchAngle - yawAngle
+ fly()

Airplane
- seats
+ reserveSeat(n)

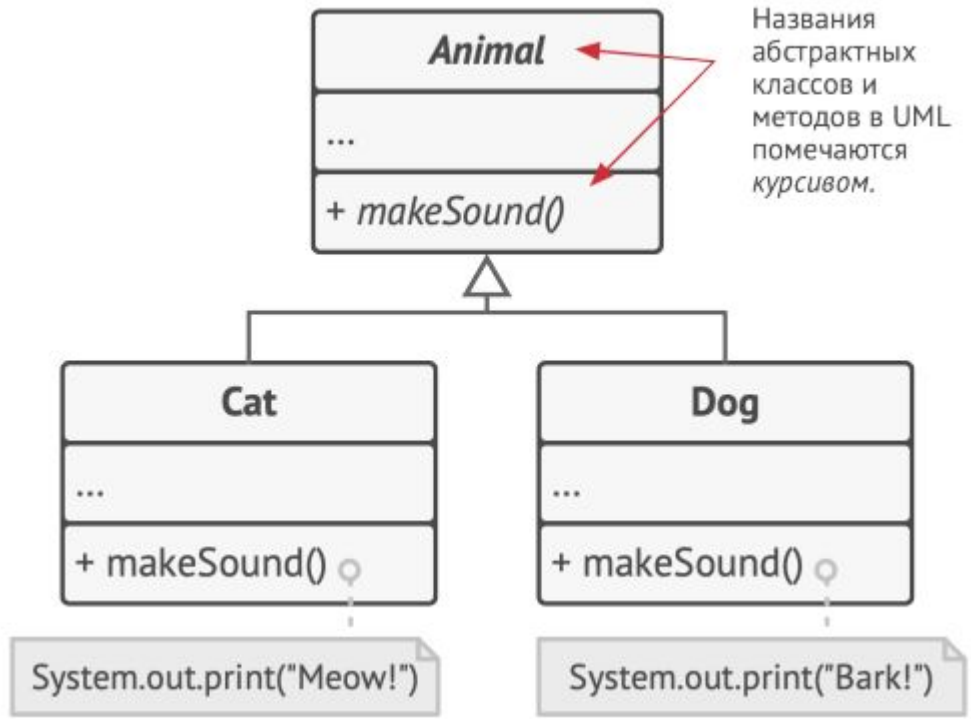
Инкапсуляция



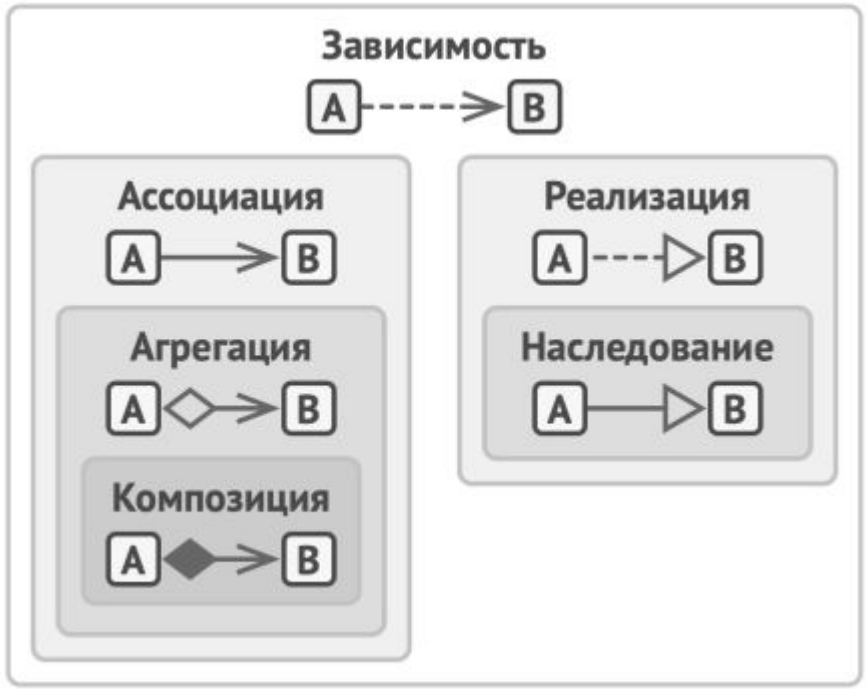
Наследование



Полиморфизм



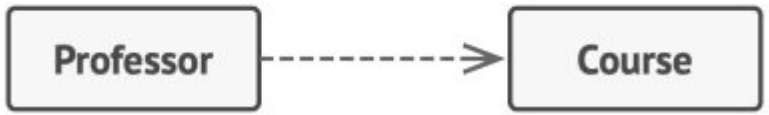
Отношения между объектами



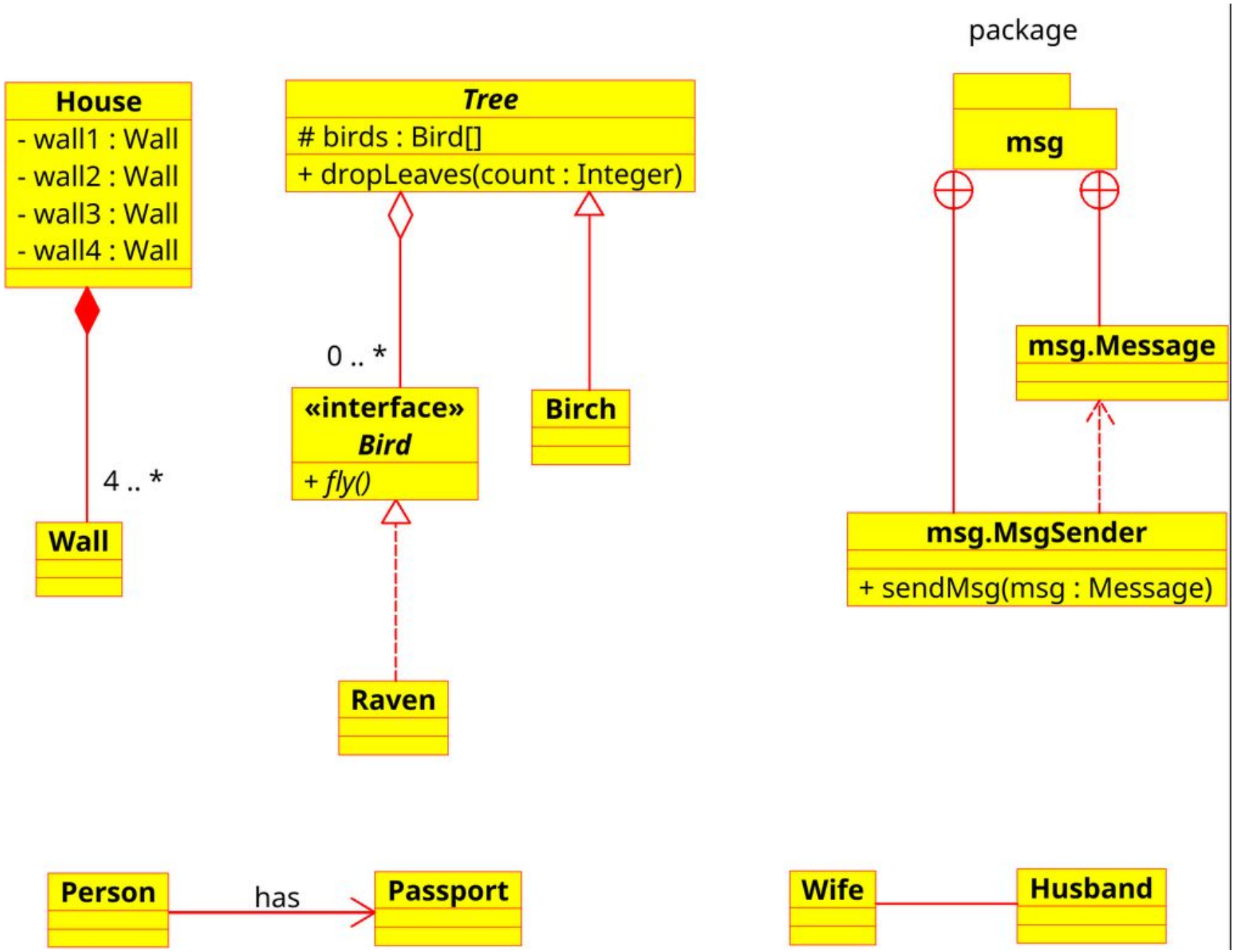
Ассоциация

Агрегация

Зависимость



Композиция


House
- wall1 : Wall
- wall2 : Wall
- wall3 : Wall
- wall4 : Wall
4 .. *
Wall
Tree
birds : Bird[]
+ dropLeaves(count : Integer)
0 .. *
«interface»
Bird
+ fly()
Birch
Raven
package
msg
msg.Message
msg.MsgSender
+ sendMsg(msg : Message)
Person
has
Passport
Wife
Husband

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

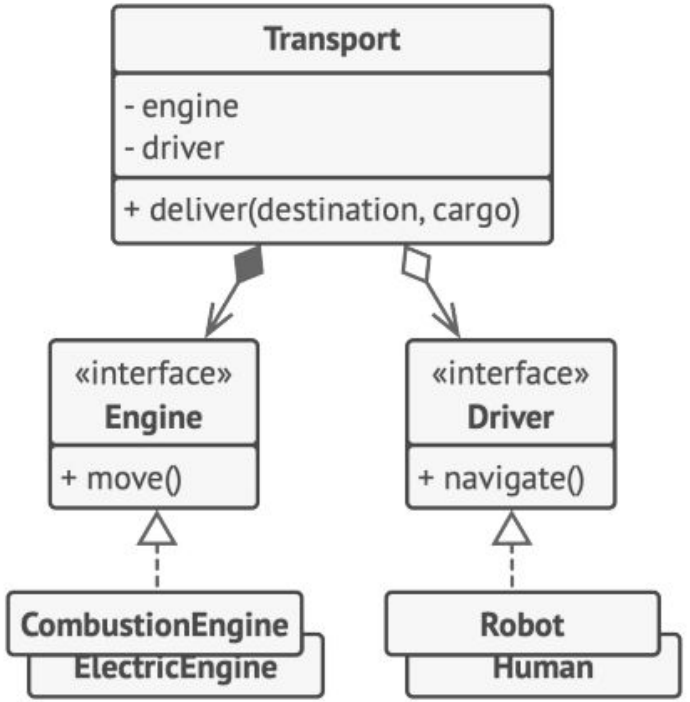
Инкапсулируйте то, что меняется

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.

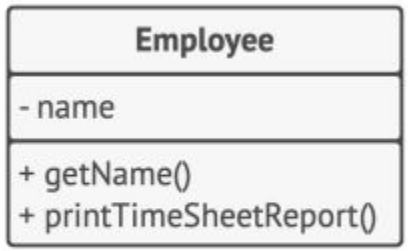
Предпочитайте композицию наследованию



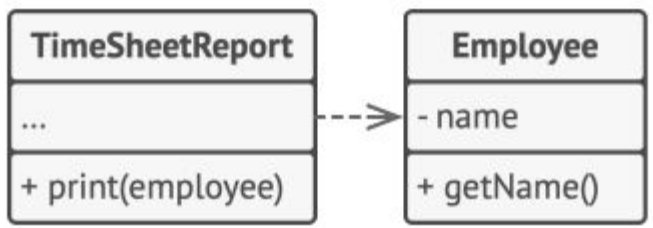
SOLID

S Принцип единственной ответственности
ingle Responsibility Principle

У класса должен быть только один мотив для изменения.



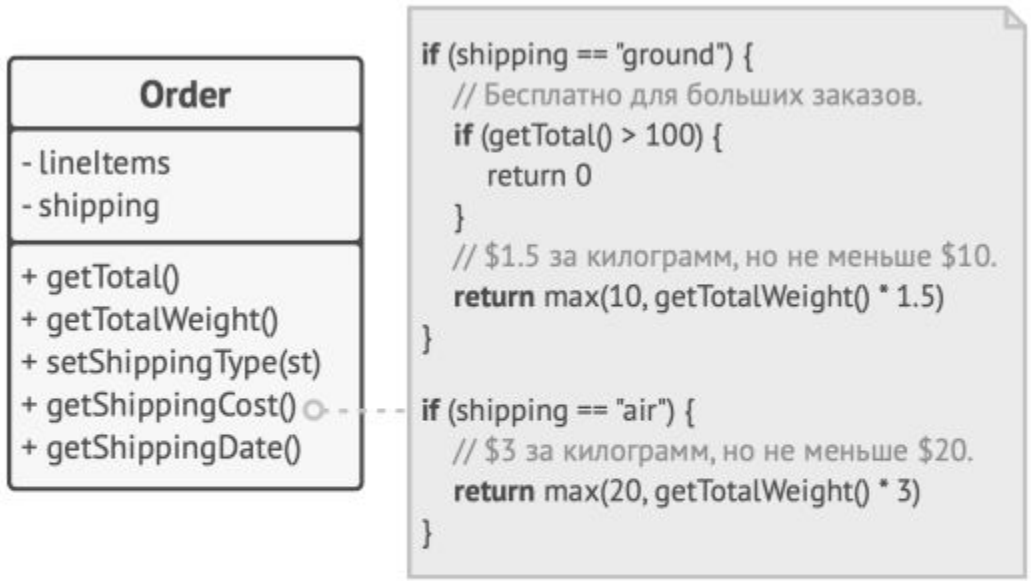
ДО: класс сотрудника содержит разнородные поведения.



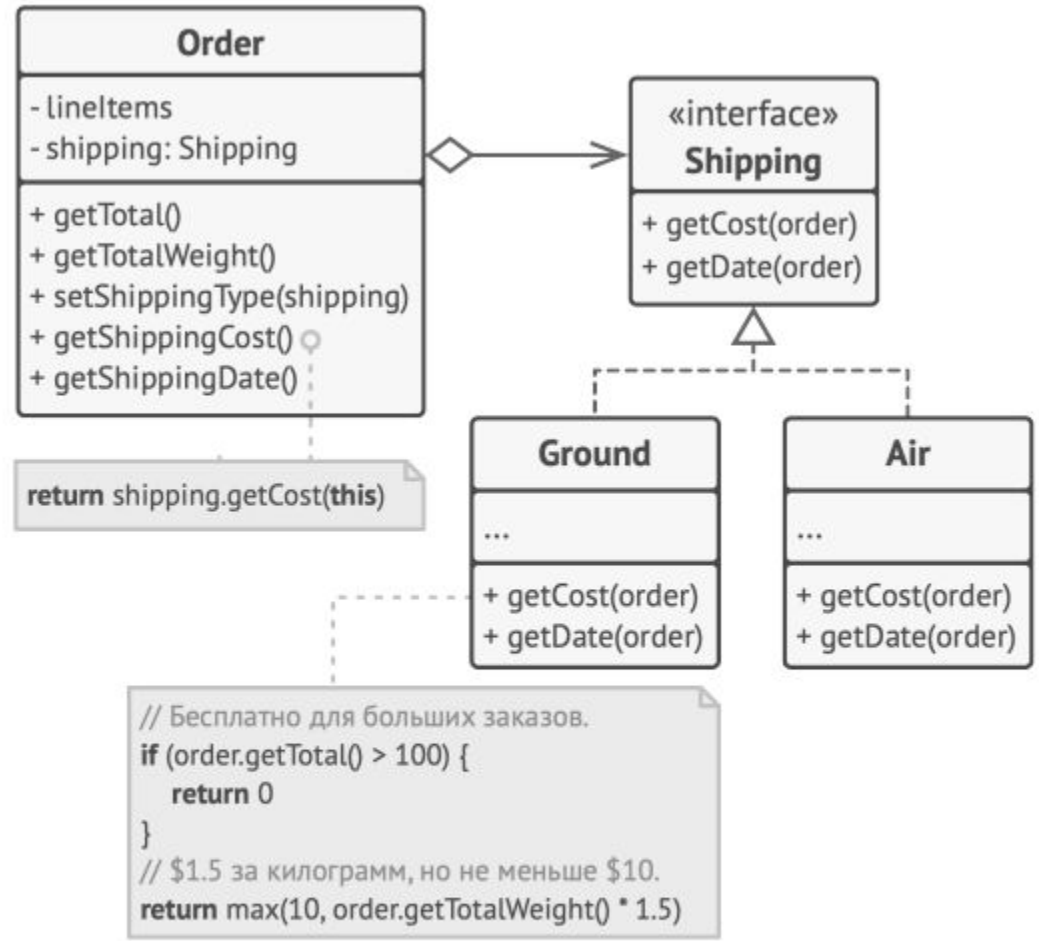
ПОСЛЕ: лишнее поведение переехало в собственный класс.

O Принцип открытости/закрытости
pen/Closed Principle

Расширяйте классы, но не изменяйте их первоначальный код.



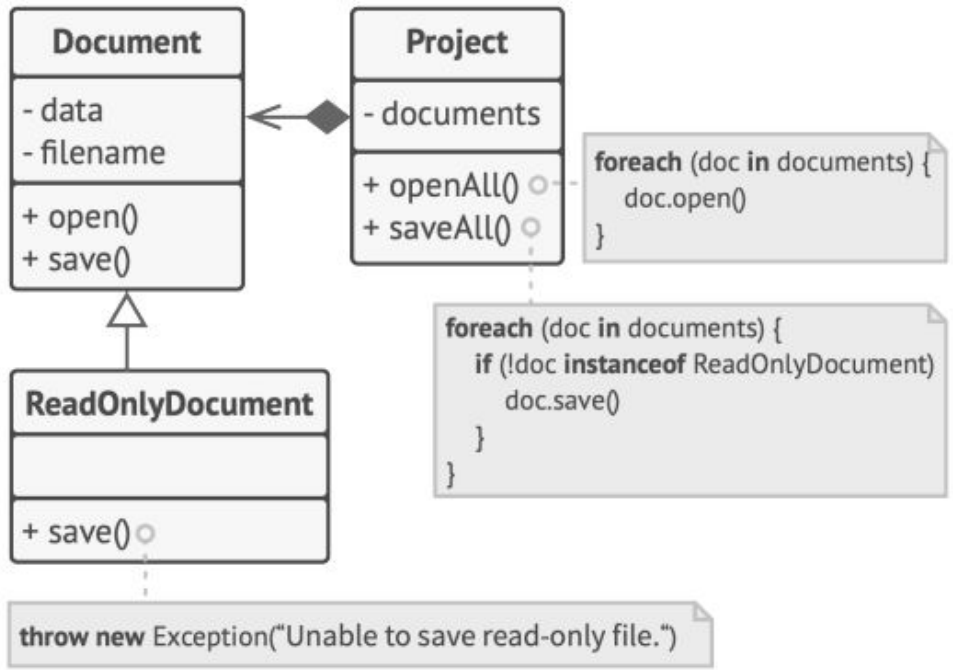
ДО: код класса заказа нужно будет изменять при добавлении нового способа доставки.



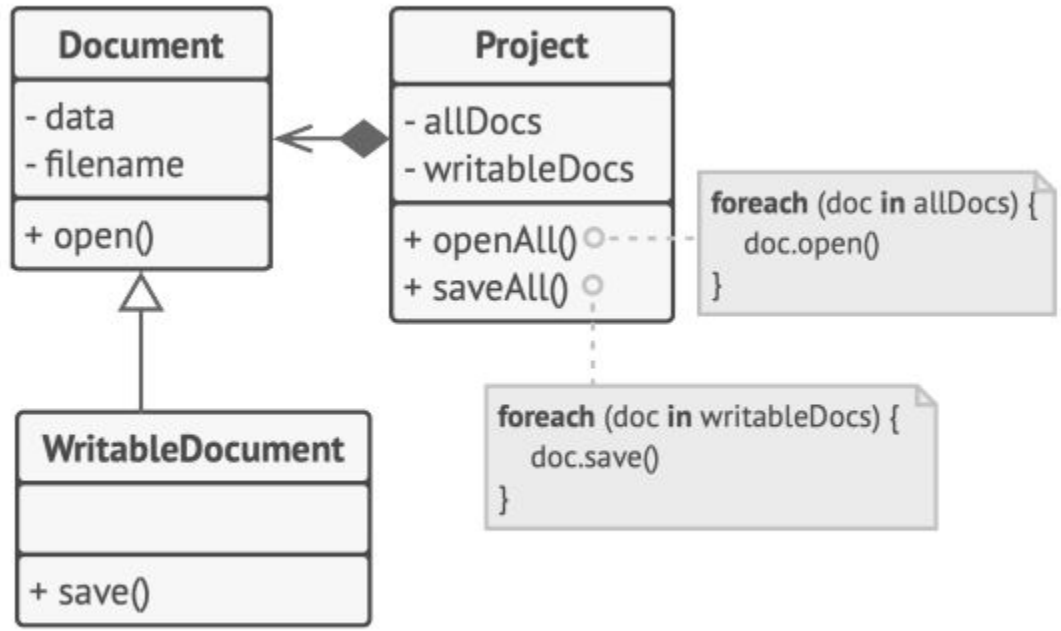
ПОСЛЕ: новые способы доставки можно добавить, не трогая класс заказов.

Принцип подстановки Лисков[1]
Liskov Substitution Principle

Подклассы должны дополнять, а не замещать поведение базового класса.



ДО: подкласс «обнуляет» работу базового метода.

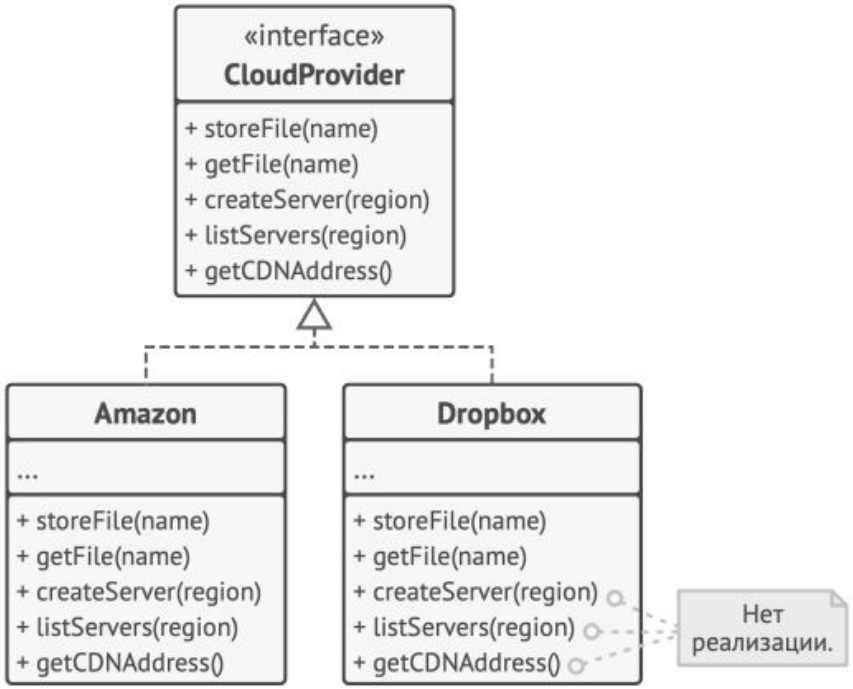


ПОСЛЕ: подкласс расширяет базовый класс новым поведением.

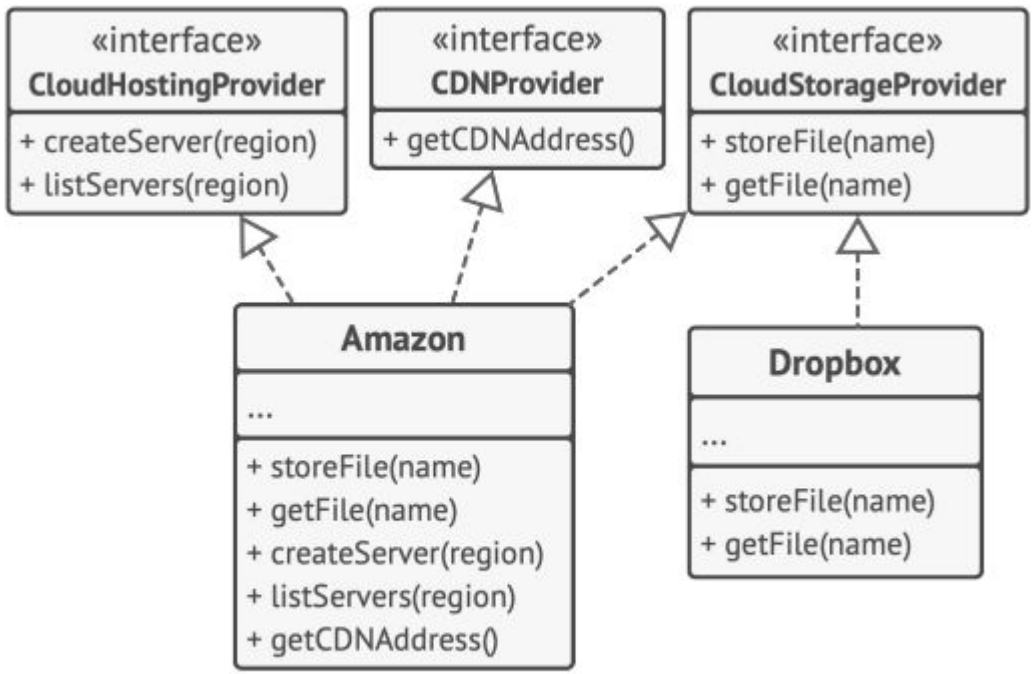
Принцип разделения интерфейса

Interface Segregation Principle

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



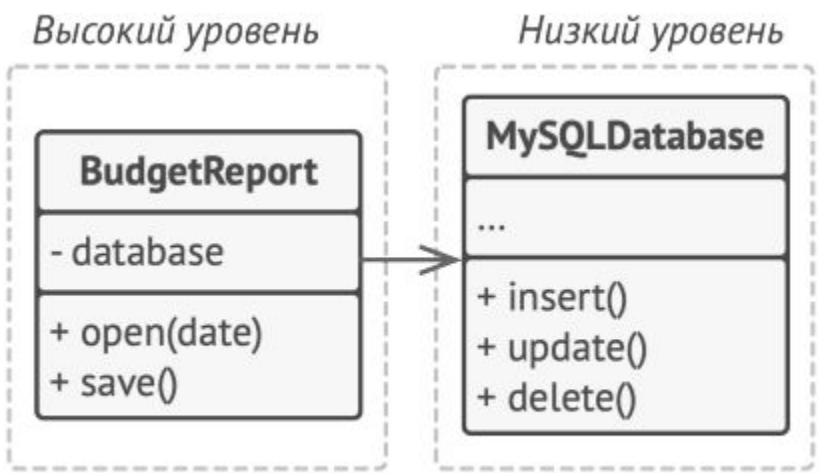
ДО: не все клиенты могут реализовать операции интерфейса.



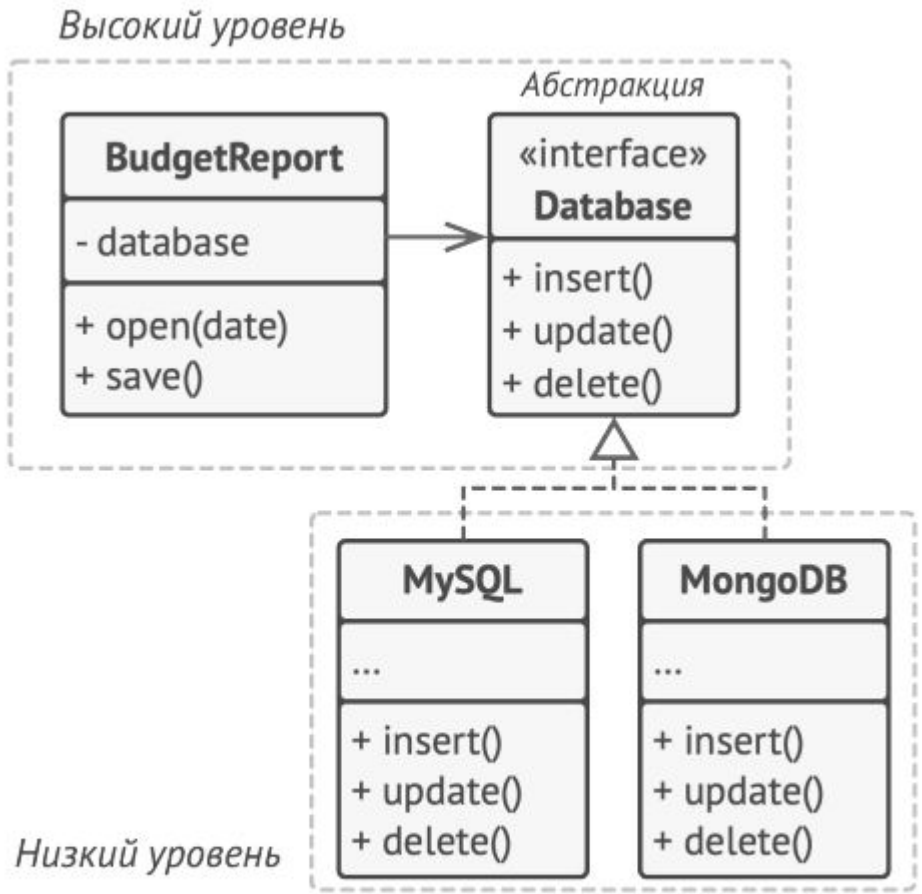
ПОСЛЕ: раздутый интерфейс разбит на части.

D Принцип инверсии зависимостей
ependency 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.

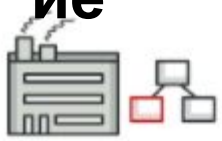


ДО: высокоуровневый класс зависит от низкоуровневого.



ПОСЛЕ: низкоуровневые классы зависят от высокоуровневой абстракции.

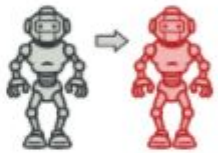
Порождающие



Фабричный метод
Factory Method

Абстрактная фабрика
Abstract Factory

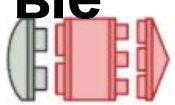
Строитель
Builder



Прототип
Prototype

Одиночка
Singleton

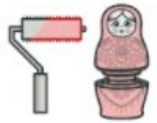
Структурные



Адаптер
Adapter

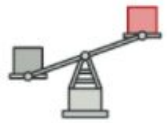
Мост
Bridge

Компоновщик
Composite

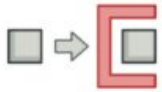


Декоратор
Decorator

Фасад
Facade



Легковес
Flyweight



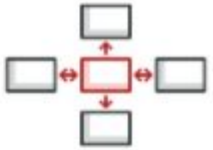
Заместитель
Proxy

Поведенческие

Цепочка обязанностей
Chain of Responsibility

Команда
Command

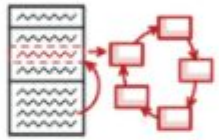
Итератор
Iterator



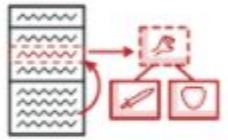
Посредник
Mediator

Снимок
Memento

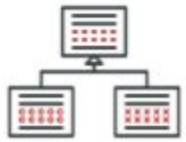
Наблюдатель
Observer



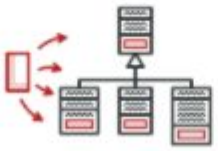
Состояние
State



Стратегия
Strategy



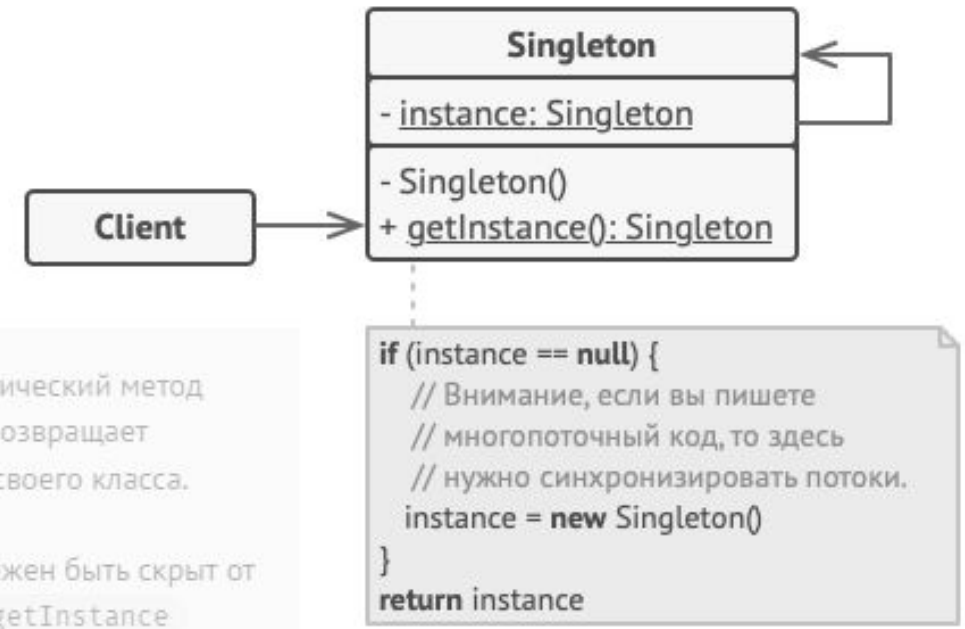
Шаблонный метод
Template Method



Посетитель
Visitor

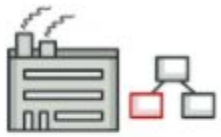
Одиночка

Singleton



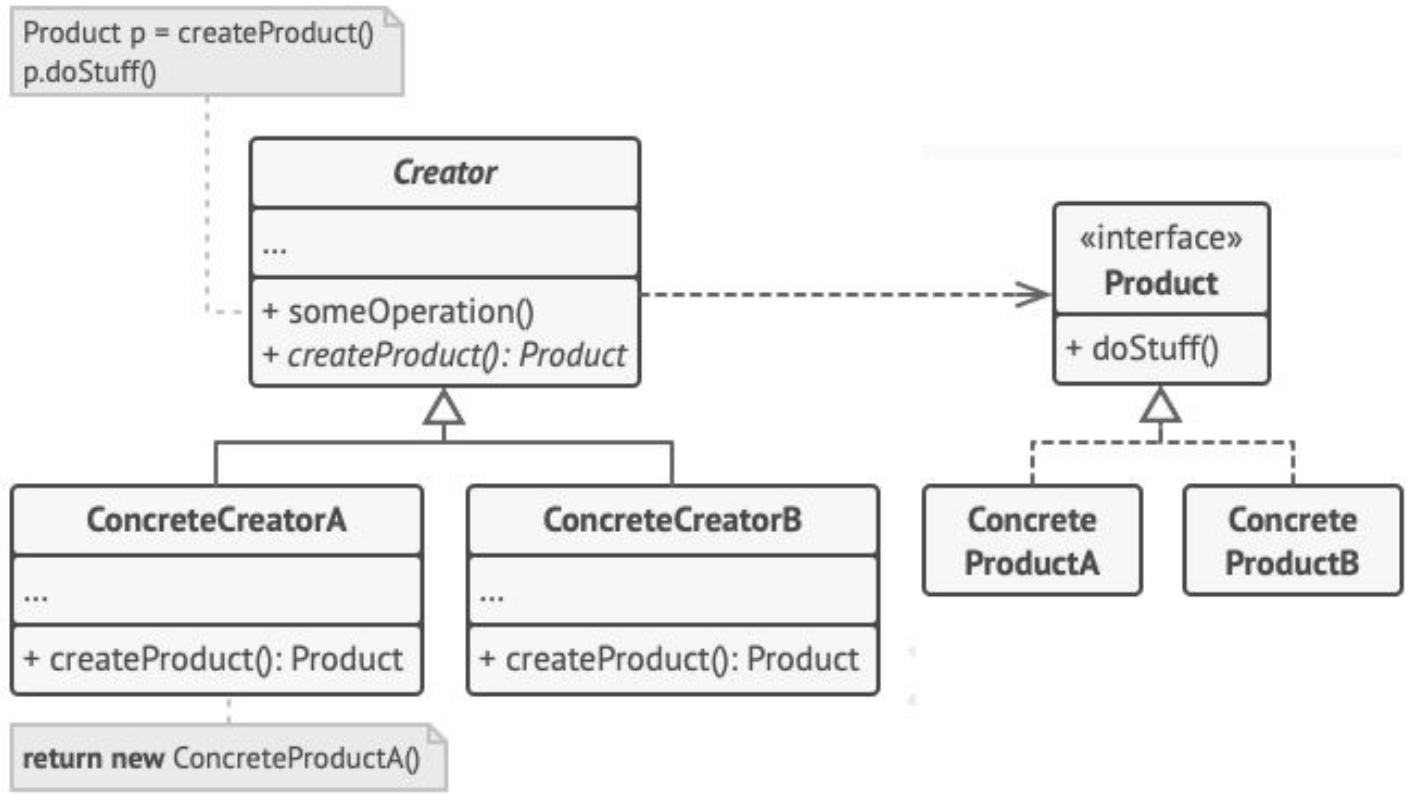
1 **Одиночка** определяет статический метод `getInstance`, который возвращает единственный экземпляр своего класса.

Конструктор одиночки должен быть скрыт от клиентов. Вызов метода `getInstance` должен стать единственным способом получить объект этого класса.



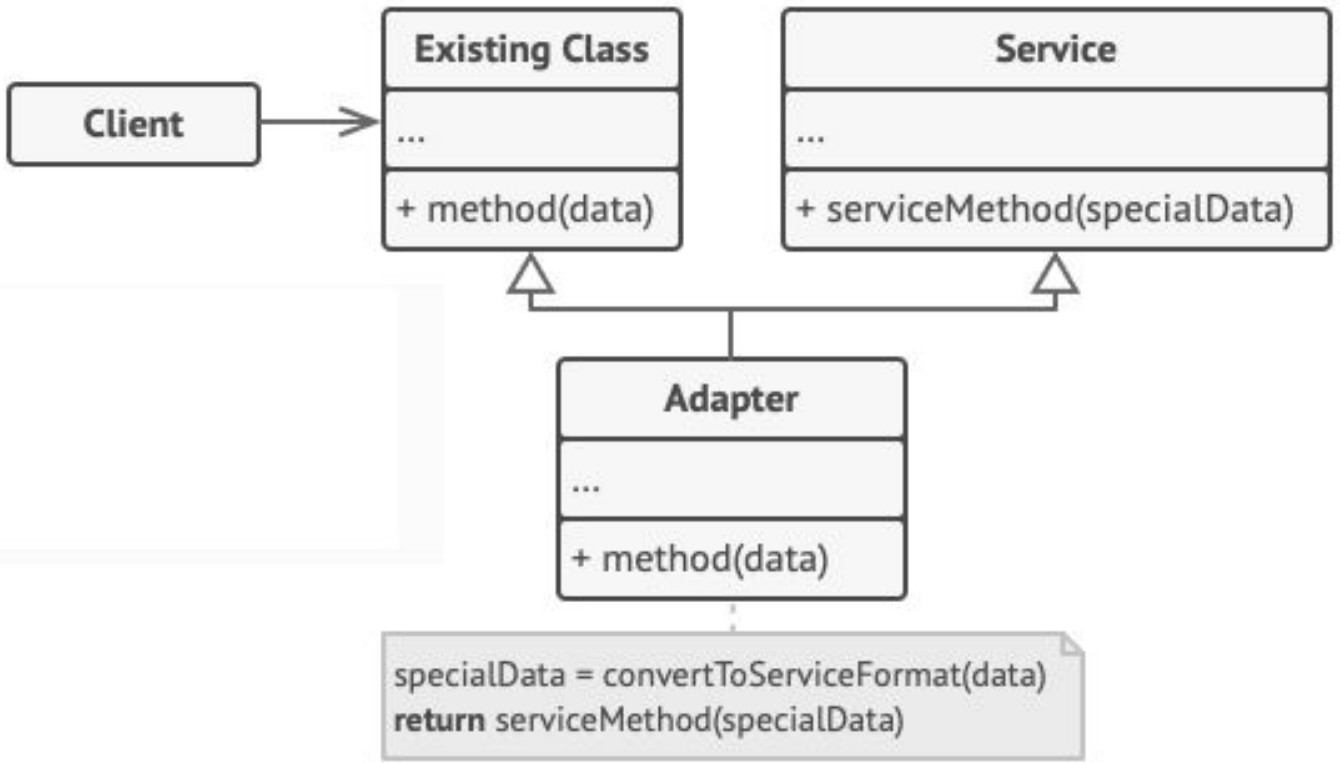
Фабричный метод

Factory Method



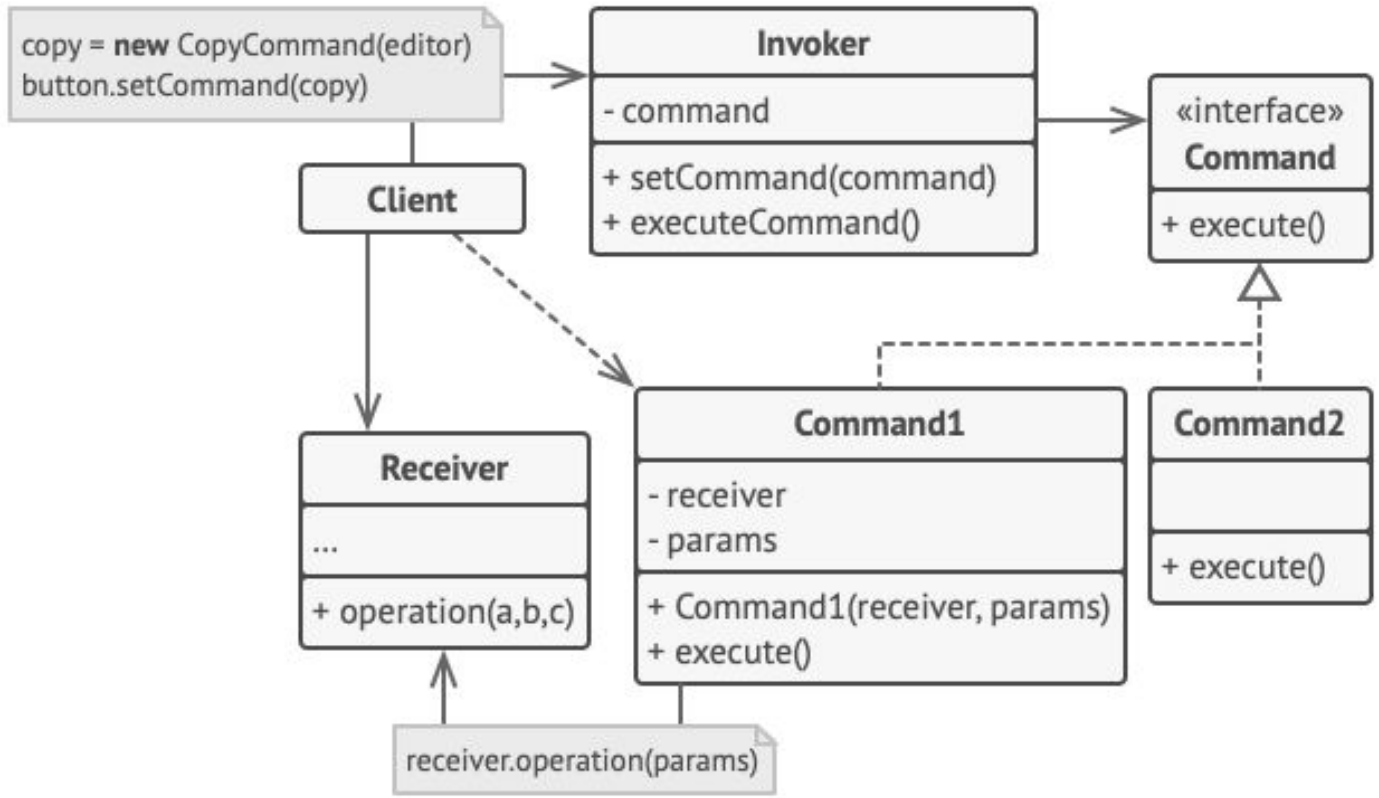
Адаптер

Adapter



Команда

Command



DRY – расшифровывается как Don't Repeat Youself – 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.