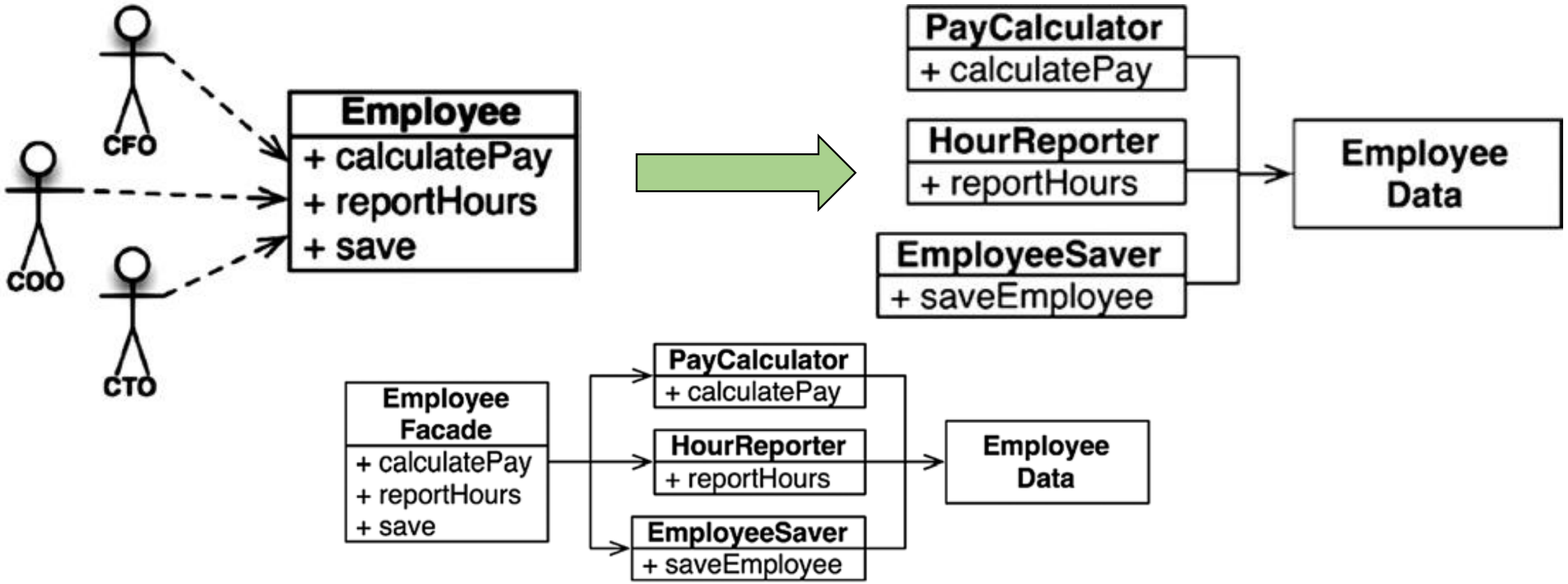
SOLID

Принципы SOLID

S
O
L
I
D

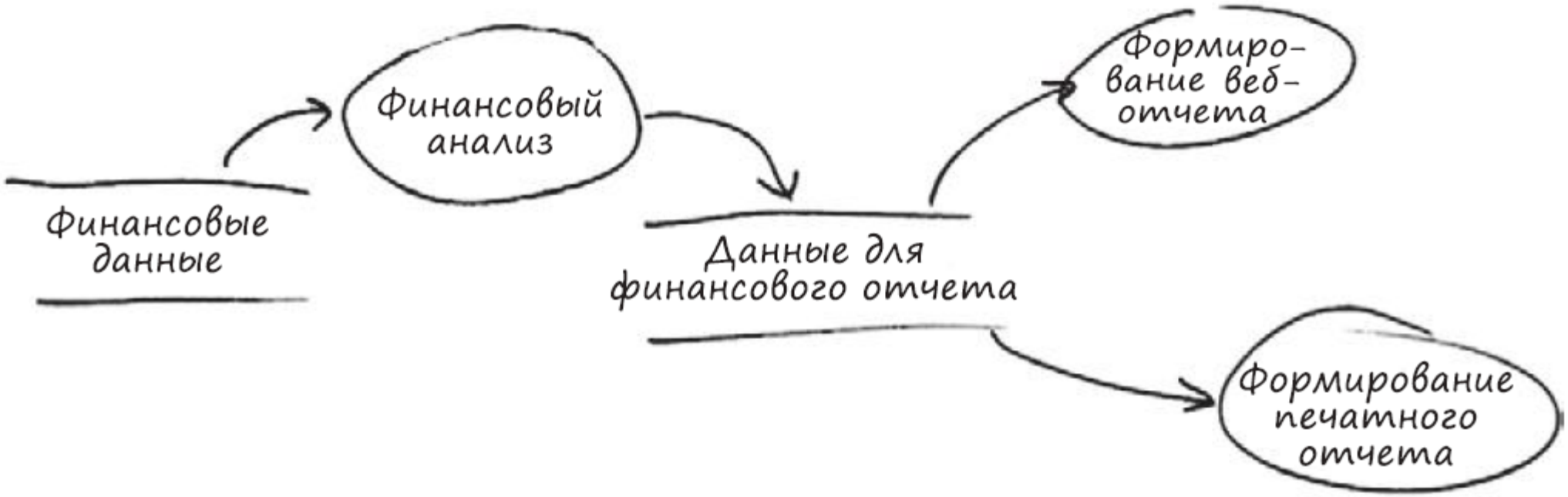
— **S**ingle Responsibility (Принцип единственной ответственности)
— **O**pen-Closed (Принцип открытости-закрытости)
— **L**iskov Substitution (Принцип подстановки Барбары Лисков)
— **I**nterface Segregation (Принцип разделения интерфейсов)
— **D**ependency Inversion (Принцип инверсии зависимостей)

Принцип единственной ответственности



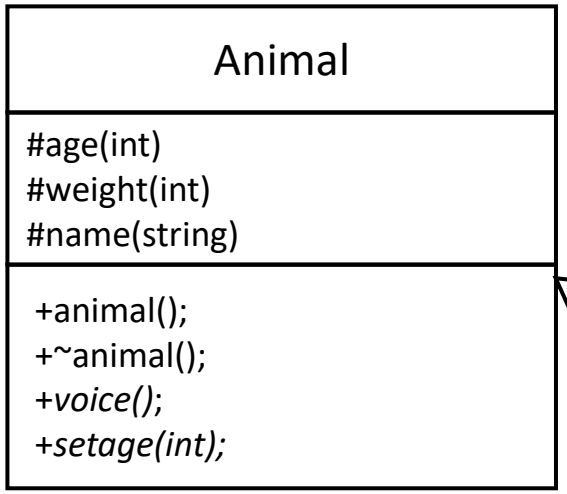
Модуль должен иметь одну и только одну причину для изменения

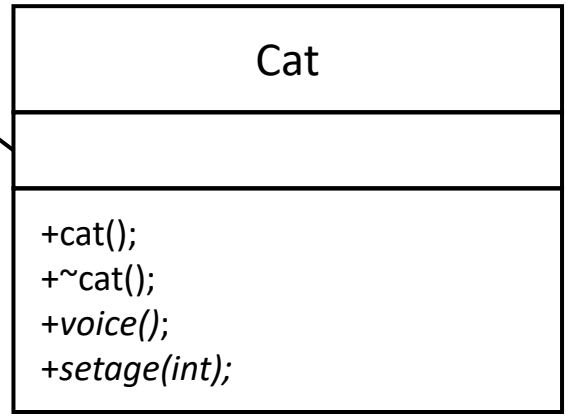
Принцип открытости/закрытости



Программные сущности должны быть открыты для расширения и закрыты для изменения

Принцип подстановки Барбары Лисков



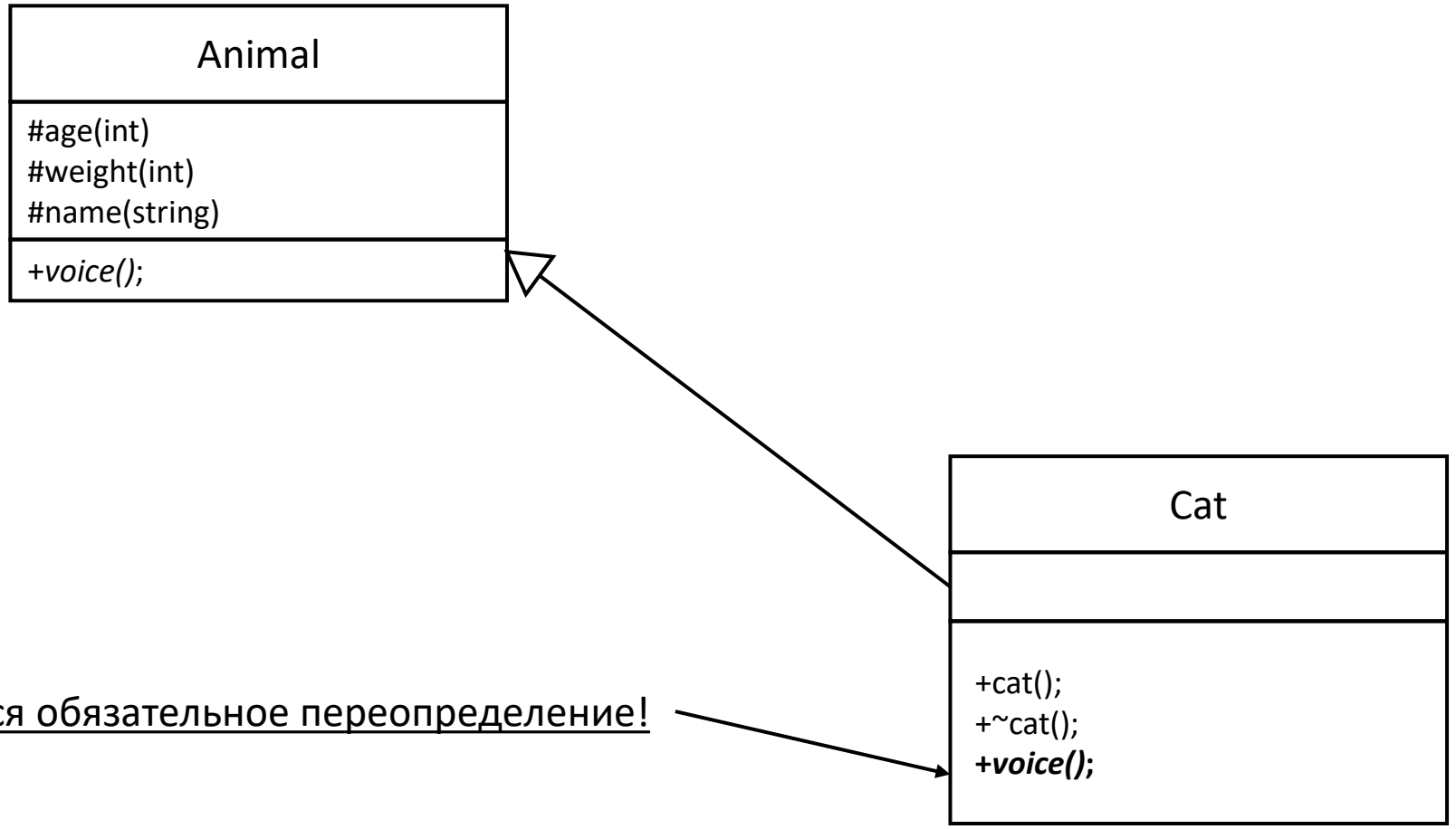



```
void animal::setage(int age) {
    if (age < 100)
        this->age = age;
    else
        std::cout << "animal age error" << std::endl;
}
```

```
void cat::setage(int age) {
    if (age < 40)
        this->age = age;
    else
        std::cout << "cat age error" << std::endl;
}
```

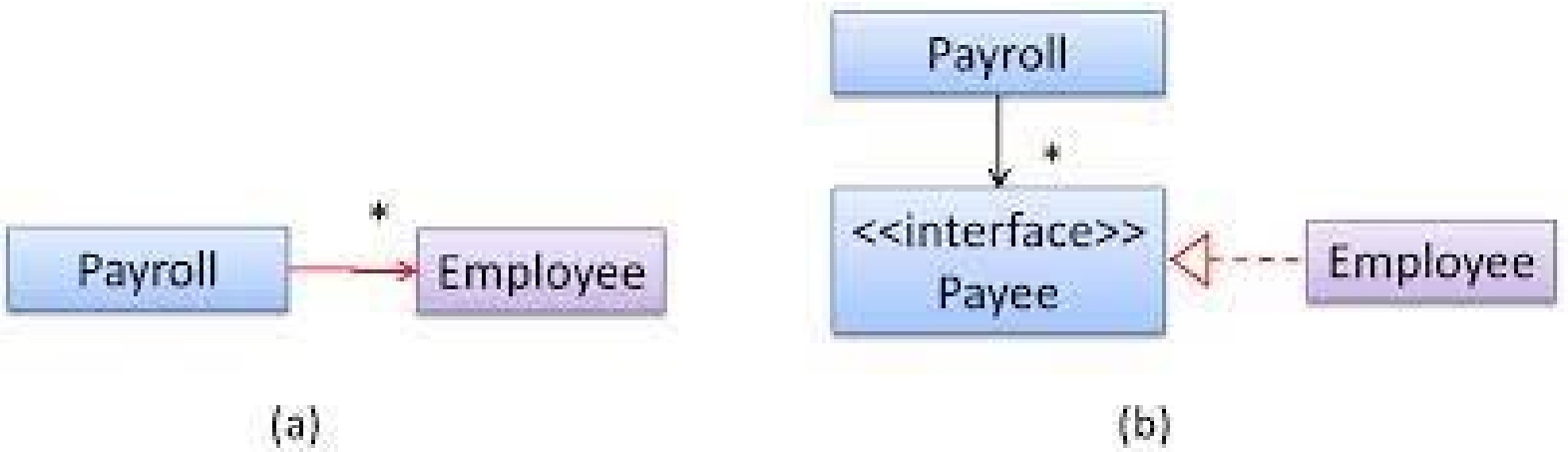
Подклассы должны дополнять, а не замещать поведение базового класса.

Принцип разделения интерфейсов



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

Принцип инверсии зависимостей



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.

KISS — *Keep It Simple, Stupid*

Держи код простым. Не усложняй там, где можно обойтись простым решением.

```
// Сложно читать
if ((x > 0 && y > 0) || !(x < 0 || y < 0)) {
    doSomething();
}

// Просто и понятно
if (x >= 0 && y >= 0) {
    doSomething();
}
```

DRY — *Don't Repeat Yourself*

Не повторяй один и тот же код в разных местах.

```
// Дублирование
void printCat() { std::cout << "Cat" << std::endl; }
void printDog() { std::cout << "Dog" << std::endl; }

// DRY
void printAnimal(const std::string &name) { std::cout << name << std::endl; }
printAnimal("Cat");
printAnimal("Dog");
```

Паттерны проектирования

Порождающие паттерны

1. Паттерн **Фабричный метод** (Factory Method)
2. Паттерн **Абстрактная фабрика** (Abstract Factory)
3. Паттерн **Строитель** (Builder)
4. Паттерн Прототип (Prototype)
5. Паттерн **Одиночка** (Singleton)

Поведенческие паттерны

1. Паттерн Цепочка обязанностей (Chain of Responsibility)
2. Паттерн Команда (Command)
3. Паттерн Итератор (Iterator)
4. Паттерн Посредник (Mediator)
5. Паттерн Снимок (Memento)
6. Паттерн **Наблюдатель** (Observer)
7. Паттерн Состояние (State)
8. Паттерн **Стратегия** (Strategy)
9. Паттерн **Шаблонный метод** (Template Method)
10. Паттерн Посетитель (Visitor)
11. Паттерн Посетитель и двойная диспетчеризация

Структурные паттерны

1. Паттерн **Адаптер** (Adapter)
2. Паттерн Мост (Bridge)
3. Паттерн **Компоновщик** (Composite)
4. Паттерн **Декоратор** (Decorator)
5. Паттерн **Фасад (Facade)**
6. Паттерн Легковес (Flyweight)
7. Паттерн **Заместитель** (Proxy)

23 паттерна

Домашнее задание

Дедлайн: 12.11.25

Подготовить выступление на 5 минут про один из паттернов проектирования

В докладе должны быть:

- Общее представление, какую проблему решаем
- UML диаграмма классов
- Пример программы на C++ (должны компилироваться)

Источники:

GOF, Design Patterns: Elements of Reusable Object-Oriented Software
https://refactoring.guru/ru

} *Ищи на Яндекс диске*