ООП. SOLID

Пишем поддерживаемый код

Валерий Алексеевич Овчинников
valery.ovchinnikov@phystech.edu

Мотивация

Мотивация: Чего мы хотим от кода?

1. Легко читать
2. Легко поддерживать
3. Легко расширять

Мотивация: Чего мы хотим от кода?

1. Легко читать
2. Легко поддерживать
3. Легко расширять
4. High cohesion and loose coupling
   Сильная связность при слабой связанности (запутанности)

Мотивация: Зачем коду быть понятным?

1. Код пишут один раз, а читают много

Мотивация: Зачем коду быть понятным?

1. Код пишут один раз, а читают много
2. Не нужно читать документацию, чтобы разобраться

Мотивация: Зачем коду быть понятным?

1. Код пишут один раз, а читают много
2. Не нужно читать документацию, чтобы разобраться
3. Проще искать ошибки

Мотивация: Зачем коду быть понятным?

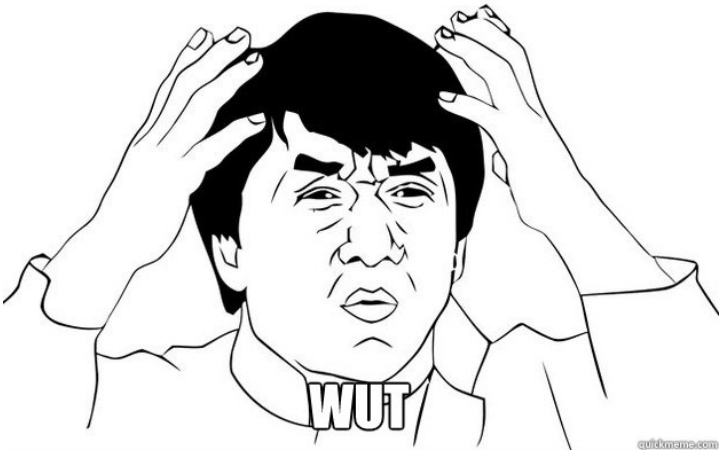
1. Код пишут один раз, а читают много
2. Не нужно читать документацию, чтобы разобраться
3. Проще искать ошибки
4. Проще менять и дополнять

Мотивация: Зачем коду быть понятным?

Что здесь происходит?

```
double f(int a, int b) {
  double y = b;
  double z = 1;
  while (a > 0) {
    if (a % 2 == 1)
      z *= y;
    y = Math.pow(y, 2);
    a /= 2;
  }
  return z;
}
```

Мотивация: Зачем коду быть понятным?



Revolut МФТИ

Мотивация: Зачем коду быть понятным?

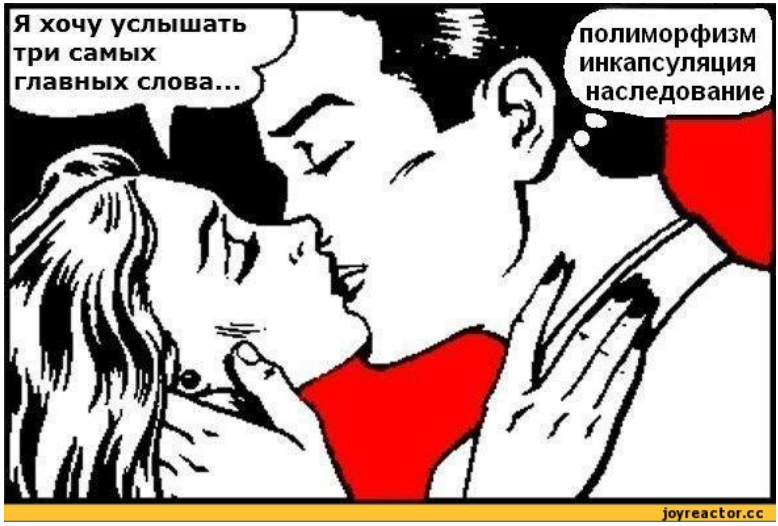
Что здесь происходит?

```
double fastPow(double base, int power) {
    double result = 1;
    while (power > 0) {
        if (isOdd(power)) {
            result *= base;
        }
        base = Math.pow(base, 2);
        power /= 2;
    }
    return result;
}
```

ООП

ООП: Записка на полях

Интерфейс – набор методов, через которые внешний мир может контактировать с объектом



ООП: Основные идеи

1. Инкапсуляция
2. Наследование (типизация)
3. Полиморфизм
4. Абстракция

ООП: Инкапсуляция

- Сокрытие внутреннего устройства объекта
- Контроль над изменением состояния объекта

ООП: Наследование

- Инструмент для создания иерархии типов, достижения полиморфизма
- Позволяет переиспользовать код

ООП: Полиморфизм

- Позволяет работать с объектами разного типа через общий интерфейс

ООП: Абстракция

Чтобы всё вышеперечисленное работало, нужно правильно выбирать абстракции
Оставлять в интерфейсе только значимые общие элементы для всех объектов, реализующих этот интерфейс

SOLID

SOLID: Расшифровка

1. Single Responsibility Principle (SRP)
   Принцип единственной обязанности (цели)
2. Open/Closed Principle (OCP)
   Принцип открытости/закрытости интерфейсов
3. Liskov Substituition Principle (LSP)
   Принцип подстановки Лисков
4. Interface Segregation Principle (ISP)
   Принцип разделения интерфейсов
5. Dependency Inversion Principle (DIP)
   Принцип инверсии зависимостей

SOLID: Принцип единственной обязанности (цели)

Каждый класс должен представлять одну сущность. Должна быть только одна причина, чтобы **менять** этот класс
Каждый метод должен делать одно какое-то действие

Позволяет легко ориентироваться в коде и быстро понимать в каком компоненте ошибка
Легко заменить реализацию конкретного функционала

SOLID: Принцип открытости/закрытости интерфейсов

Интерфейсы должны быть открыты для расширения, но закрыты для изменения

Интерфейс это контракт, если меняется интерфейс, то ломается код, использующий его
Добавление новых методов в интерфейс не ломает существующий код, но расширяет возможности

SOLID: Принцип подстановки Лисков

Если вместо объектов базового типа подставить объекты его подтипа, то код должен продолжать корректно работать

Позволяет менять реализации, сохраняя интерфейс и не меняя остальной код

SOLID: Принцип разделения интерфейсов

Лучше иметь много маленьких, специализированных интерфейсов, чем один всемогущий
Перекликается с SRP

Помогает реализовать предыдущие принципы

SOLID: Принцип инверсии зависимостей

Абстракции не должны зависеть от деталей, детали должны зависеть от абстракций
Иными словами на заданном уровне должен использоваться один и тот же уровень абстракции

Помогает лучше выбирать абстракцию, сохранять инкапсуляцию, улучшает читаемость кода

Другие принципы

Другие принципы: Tell, don't ask

Говорите объекту что нужно сделать, а не спрашивайте о его состоянии
Состояние класса не должно раскрываться даже через методы (например get/set)
Объект должен сам основываясь на своем состоянии совершать действия

Обеспечивает логическую инкапсуляцию

Другие принципы: Предпочитайте агрегацию наследованию

Лучше реализовать интерфейс расширяемого класса и завести в новом классе поле типа расширяемого класса, при необходимости вызывать нужные методы расширяемого класса

Позволяет избегать глупых и неочевидных ошибок, строитб более правильную с точки зрения ООП иерархию типов

Revolut МФТИ