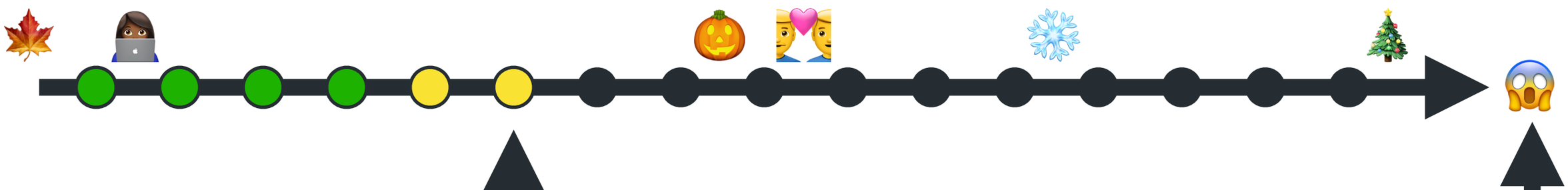
ИНФОРМАТИКА

```
~/polytech $ date
осень 2019
~/polytech $ name
константин кориков
~/polytech $ topic
бонус: ооп, функциональное программирование, tdd и чистый код
```

ГДЕ МЫ?



Мы тут

Сессия

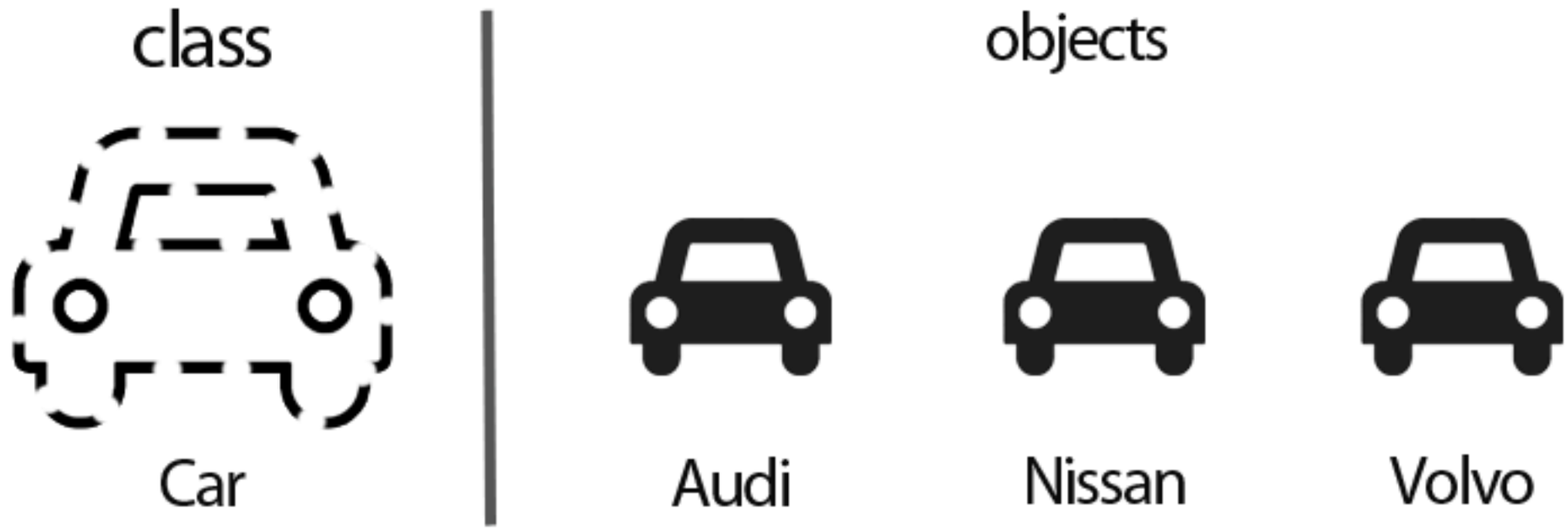
О ЧЁМ ПОГОВОРИМ?

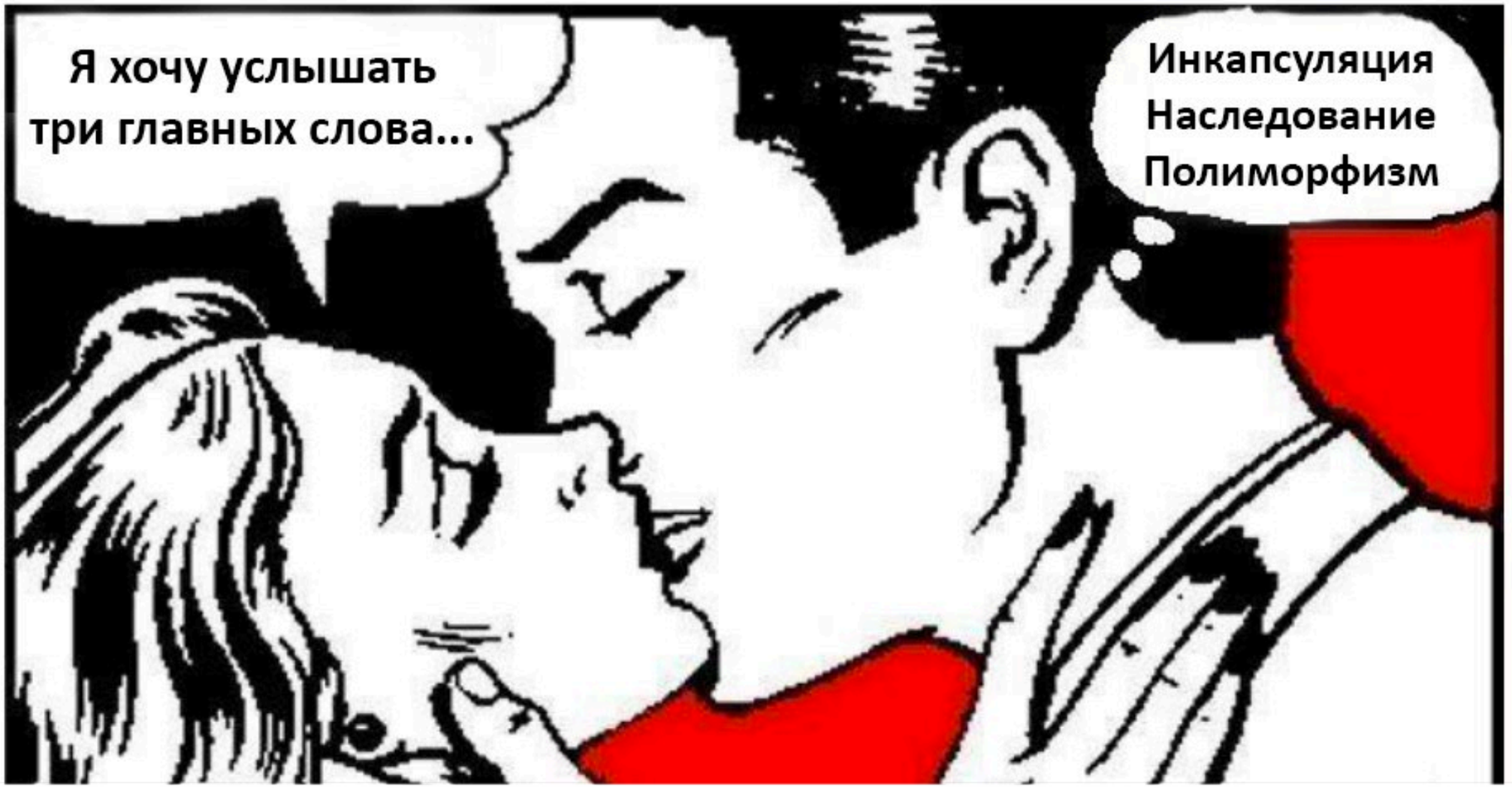
1. Объектно-ориентированное программирование
2. Функциональное программирование
3. Разработка через тестирование (TDD)
4. Чистый код

Объектно-ориентированное программирование

Идея: сущности как объекты

Классы

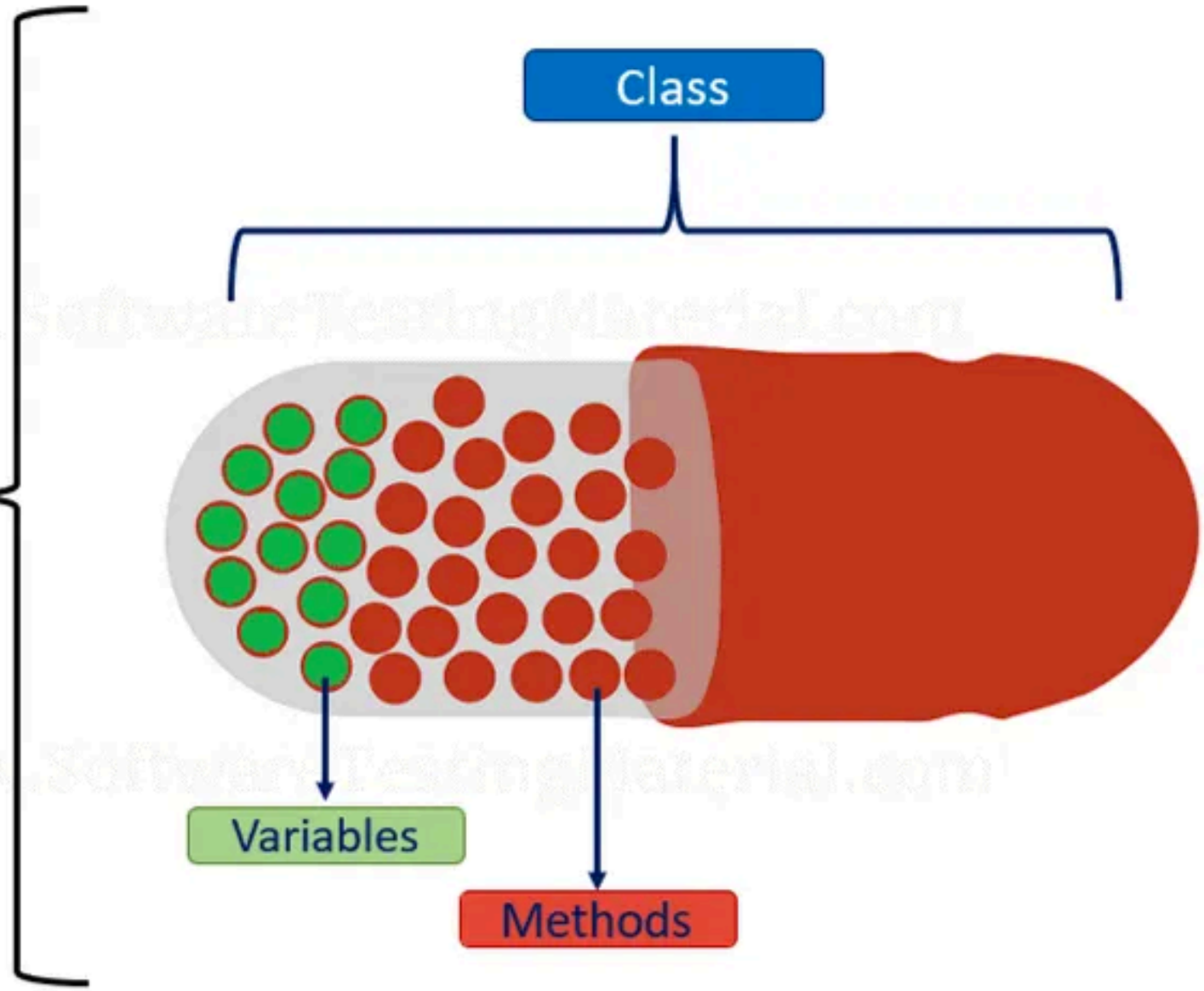



Я хочу услышать
три главных слова...
Инкапсуляция
Наследование
Полиморфизм

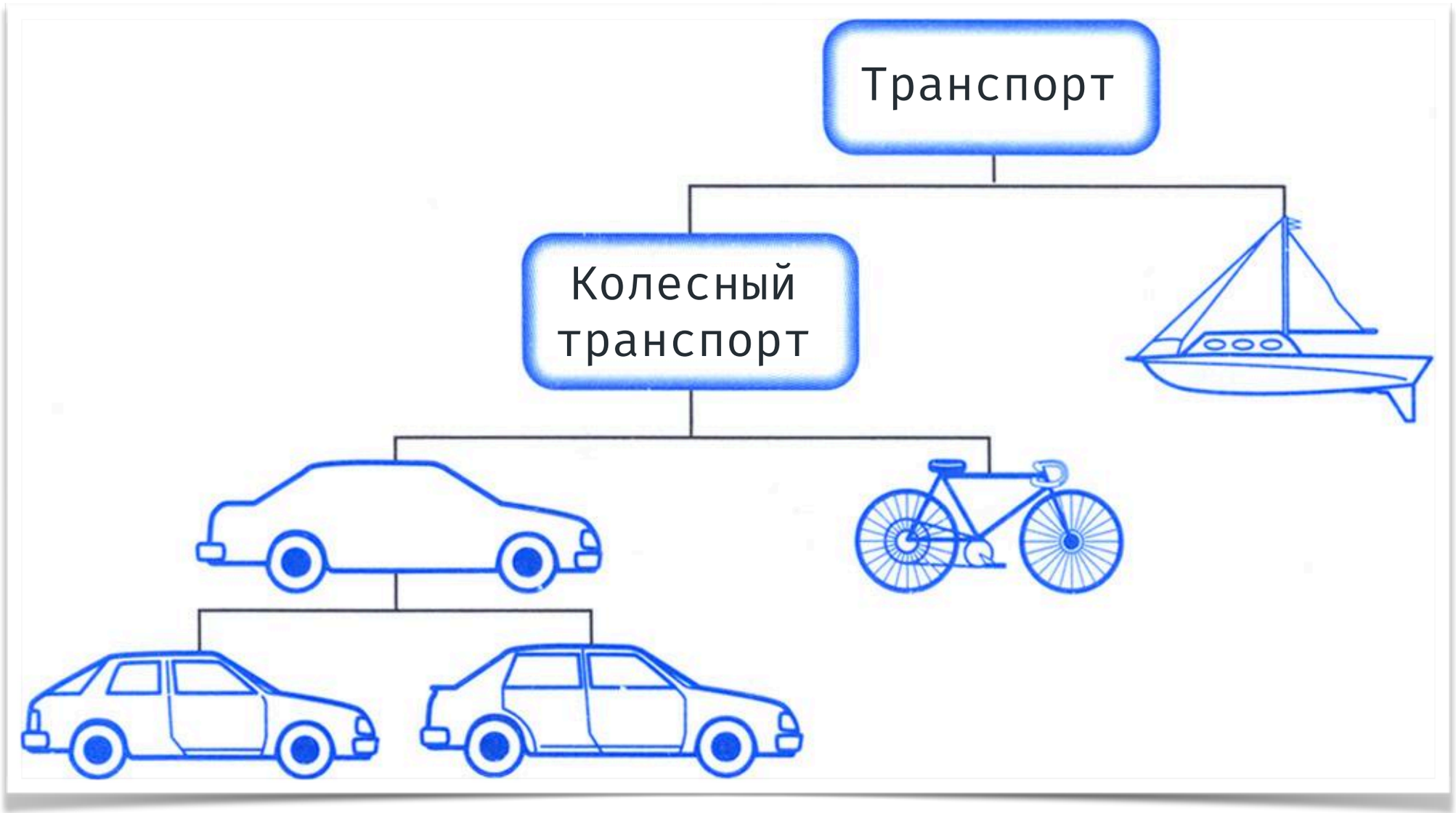
Термины

1. Абстракция
2. Инкапсуляция
3. Наследование
4. Ассоциация, композиция и агрегация
5. Полиморфизм

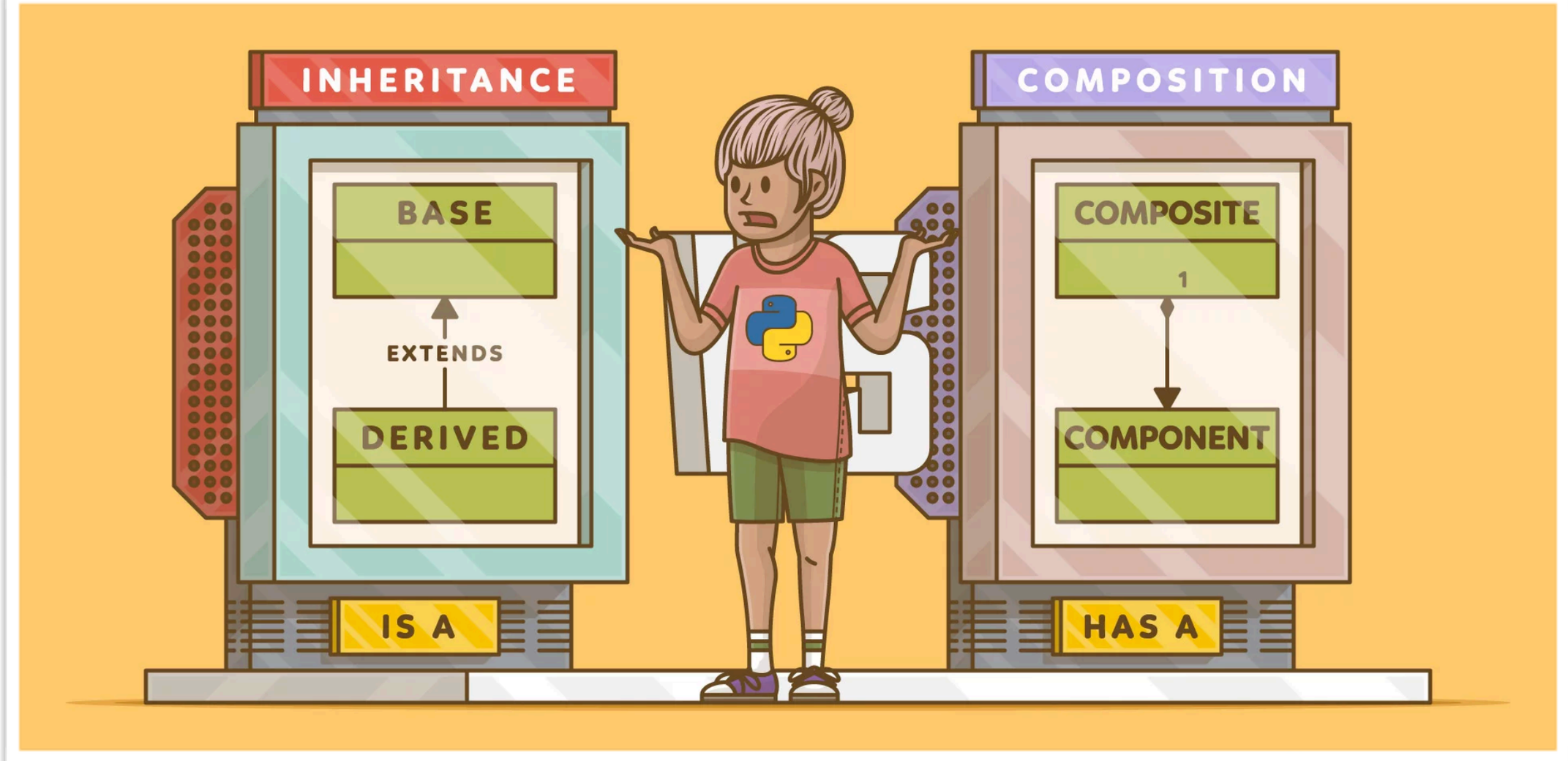
Инкапсуляция



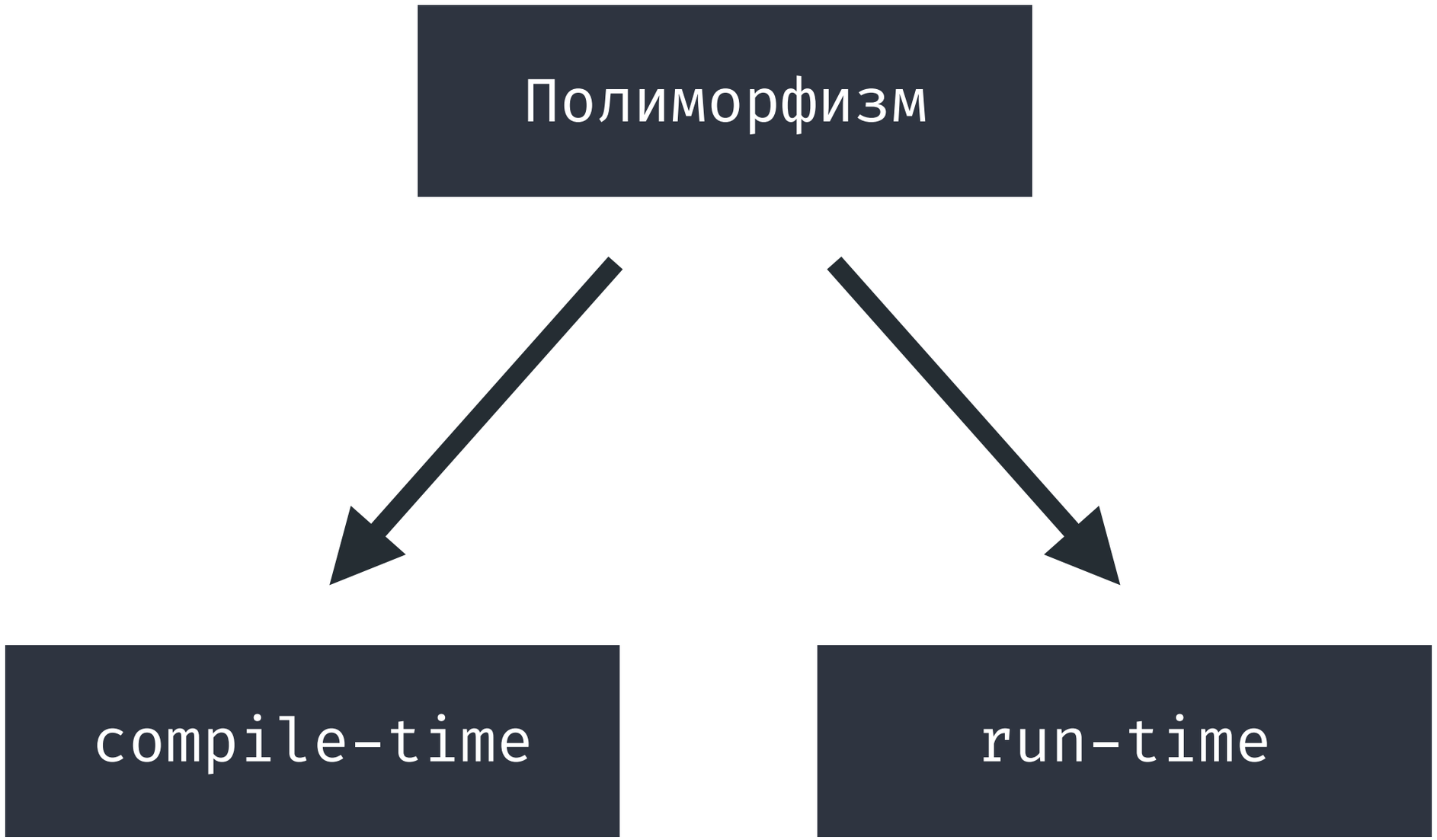
Наследование



ассоциация



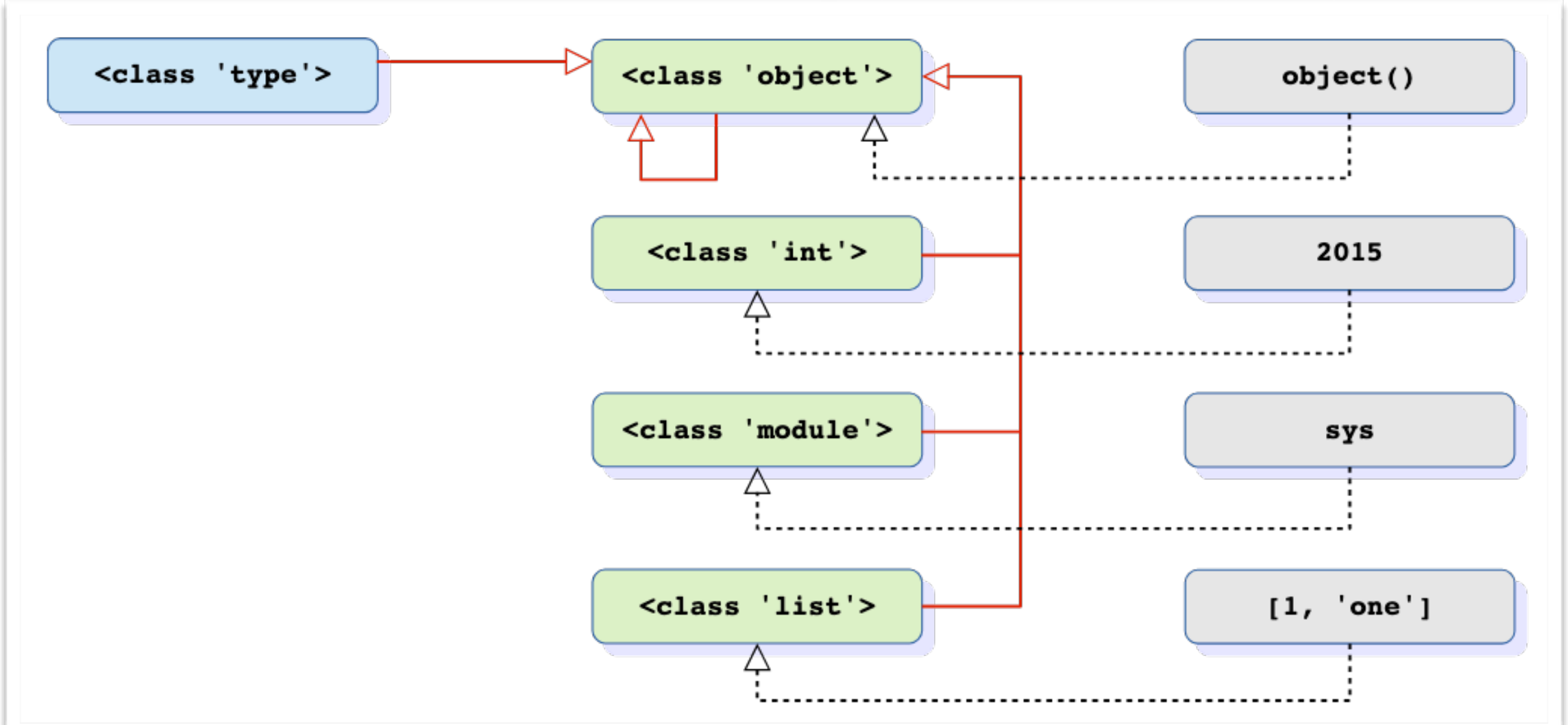
агрегация



Перегрузка

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart

x = Complex(3.0, -4.5)
print(x.r, x.i)
```

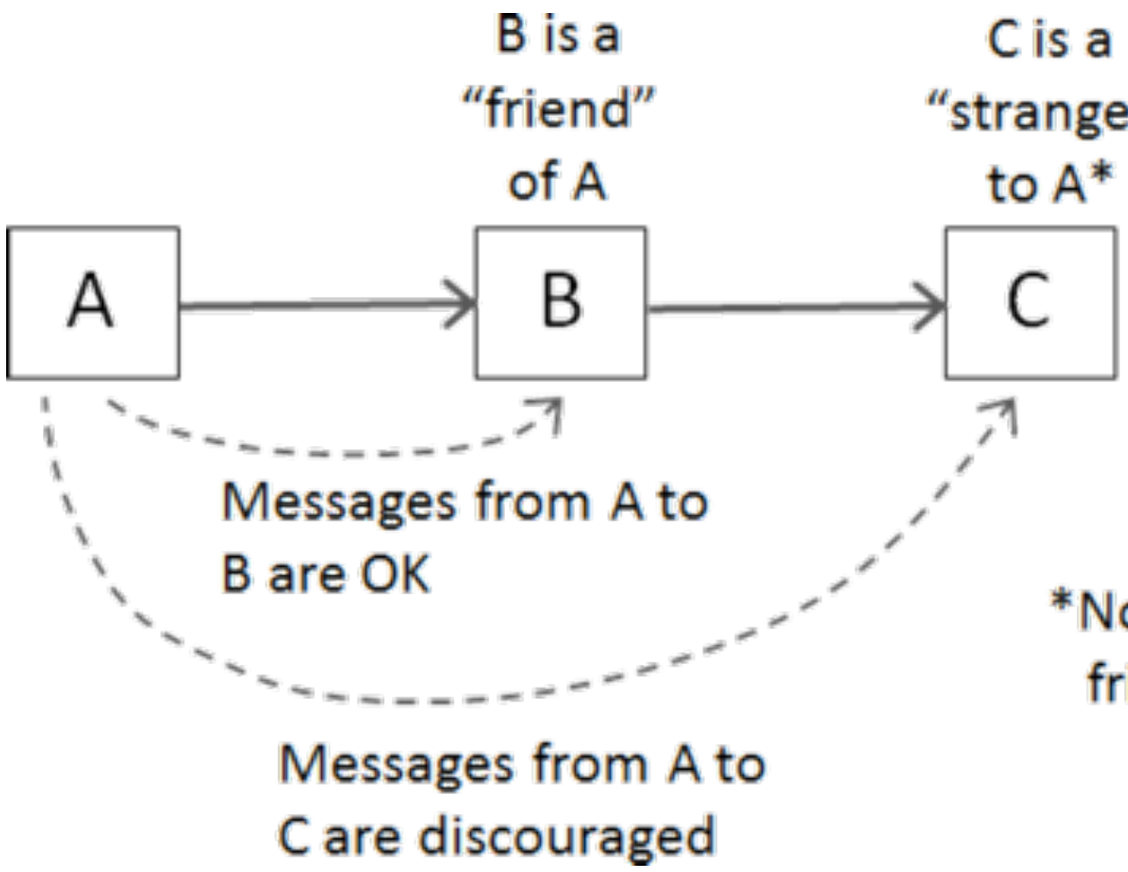
```python
SIDES = 6

class Square:
    def __init__(self, side):
        self.side = side

    def area(self):
        return self.side * self.side

class Cube(Square):
    def area(self):
        return super().area() * SIDES

    def volume(self):
        return super().area() * self.side

cube = Cube(5)
area = cube.area()

print(area)
```


<class 'type'>
<class 'object'>
object()
<class 'int'>
2015
<class 'module'>
sys
<class 'list'>
[1, 'one']

```python
class Engine:
    def on(self):
        pass

    def off(self):
        pass

class Car:
    def __init__(self):
        self._engine = Engine()

    def start(self):
        self._engine.on()

    def stop(self):
        self._engine.off()

car = Car()
car.start()
car.stop()
```

Рекомендация

~~Закон Деметры~~



*Note: a friend of a friend is a stranger.

Single Resposibility Principle

A class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)

Open / Closed Principle

A software module (it can be a class or method) should be open for extension but closed for modification.

Liskov Substitution Principle

Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.

Interface Segregation Principle

Clients should not be forced to depend upon the interfaces that they do not use.

Dependency Inversion Principle

Program to an interface, not to an implementation.

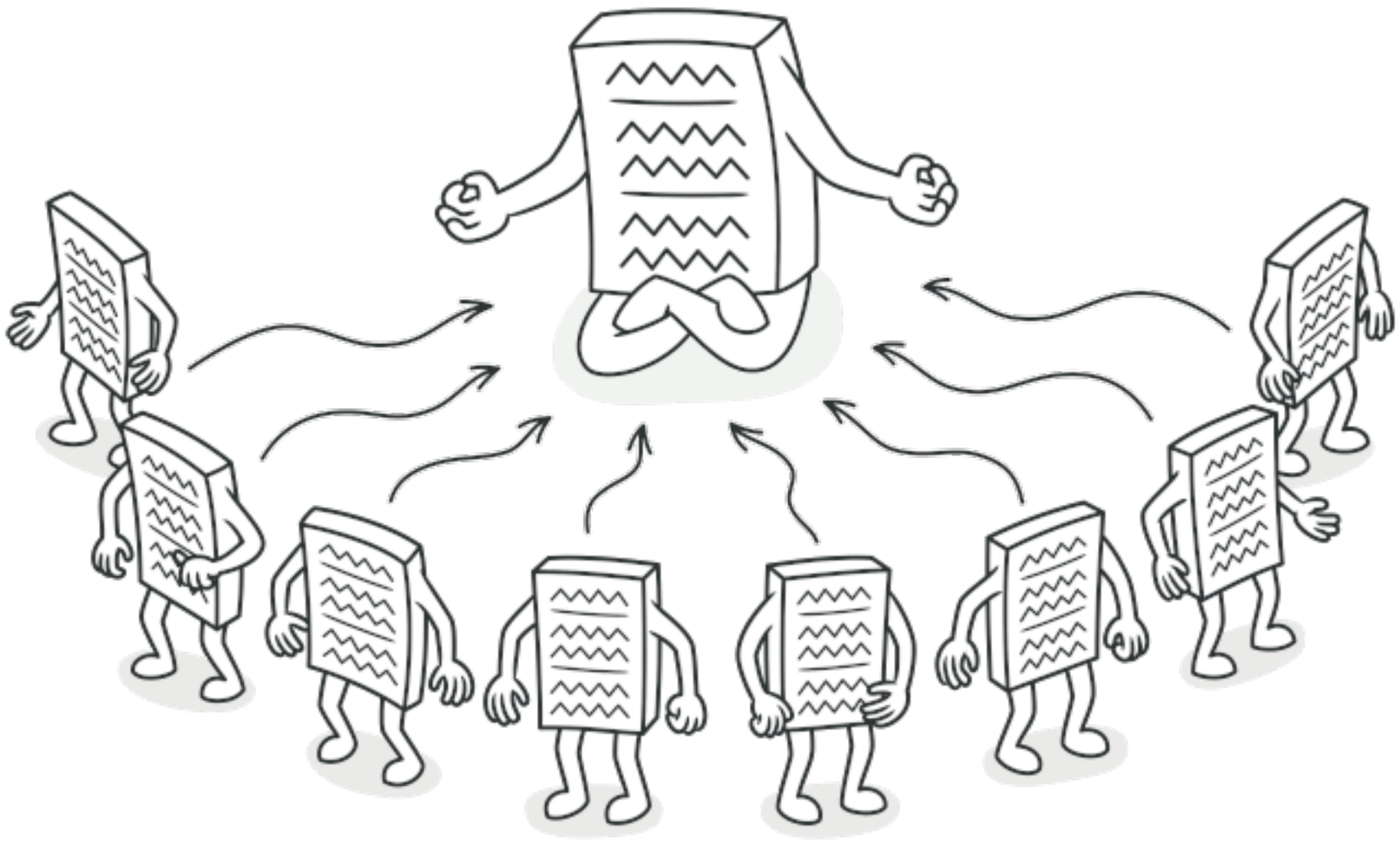
Паттерны

- **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.

«Одиночка»



Порождающий паттерн

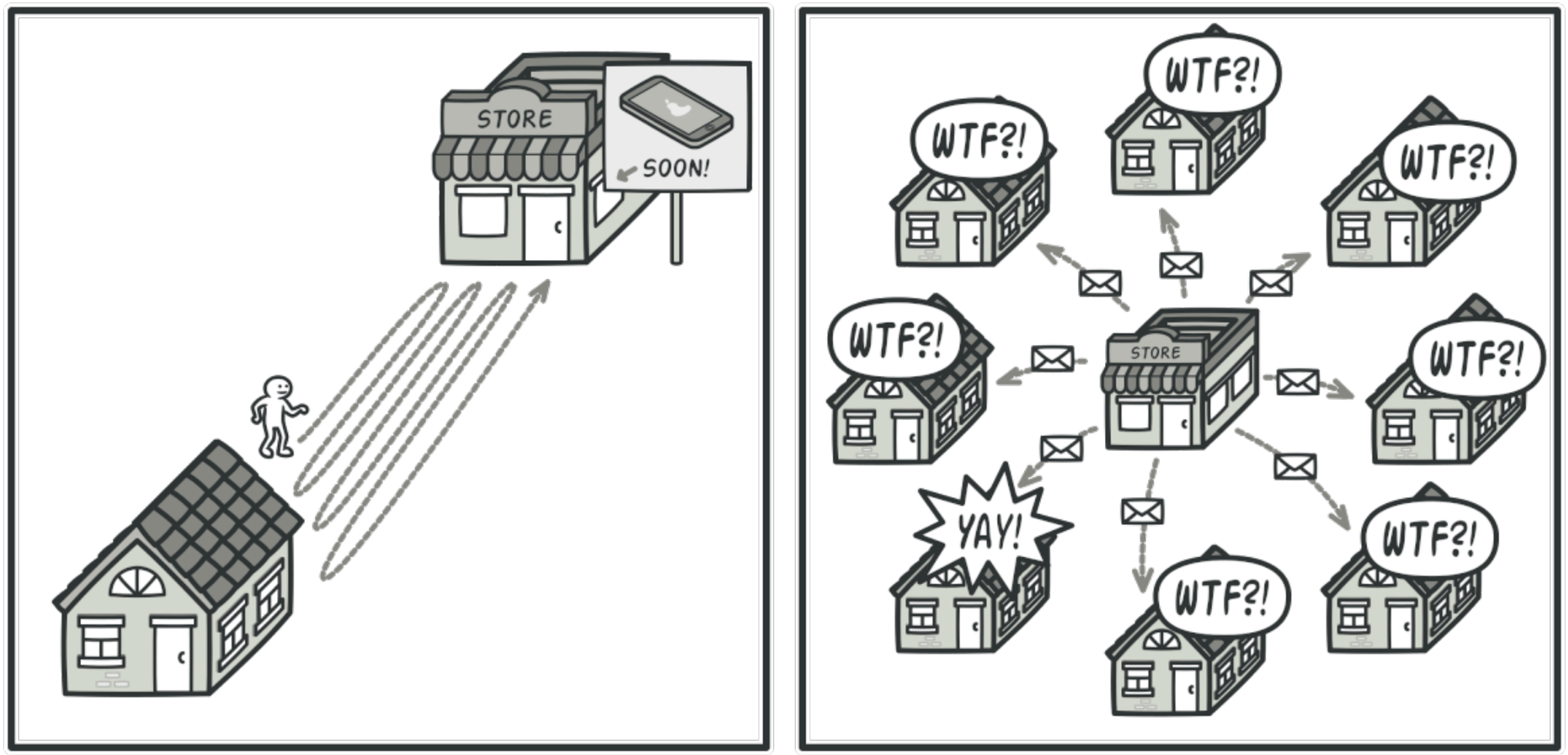
Одиночка — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

«Декоратор»

Структурный паттерн

Декоратор — это структурный паттерн проектирования, который позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки».

«Наблюдатель»



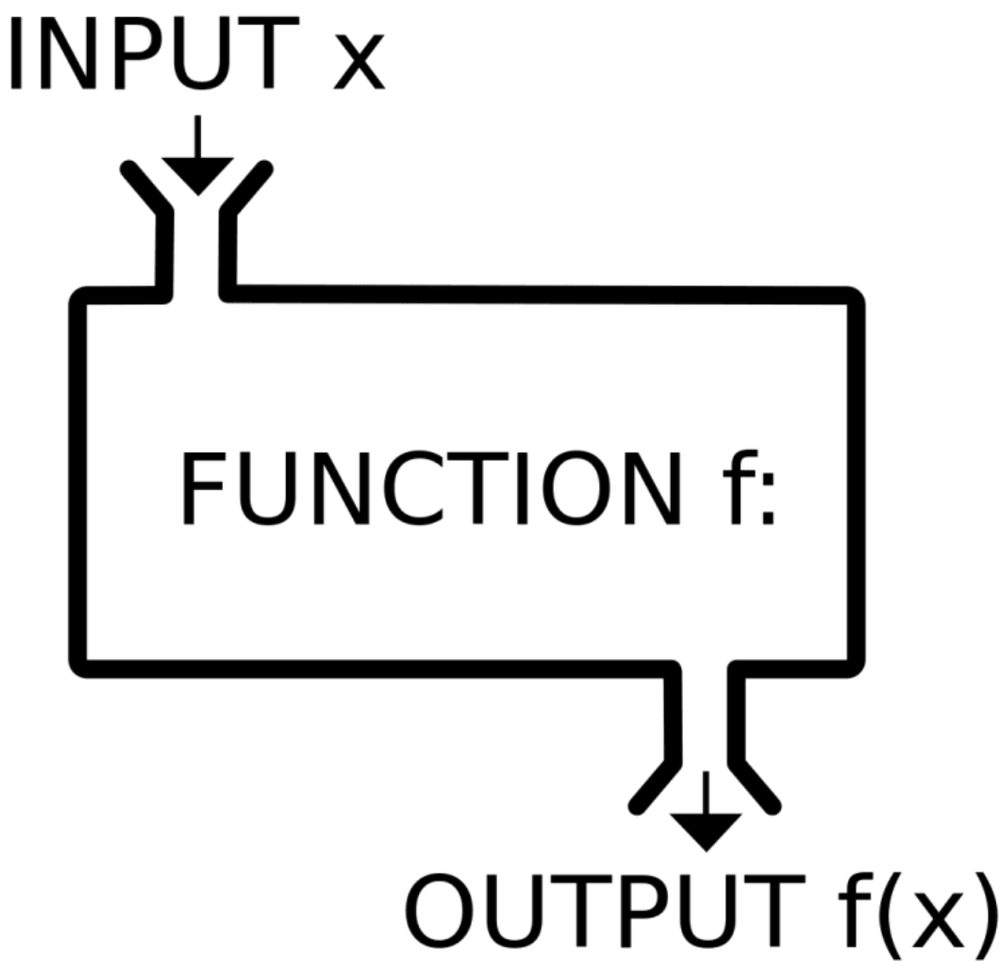
Поведенческий паттерн

Наблюдатель — это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

Функциональное программирование

Идея: функции как объекты

Чистые функции



```python
# обычная функция
def add(x, y):
    return x+y

# лямбда-функция
add = lambda x, y: x+y
```

Высшие функции

map

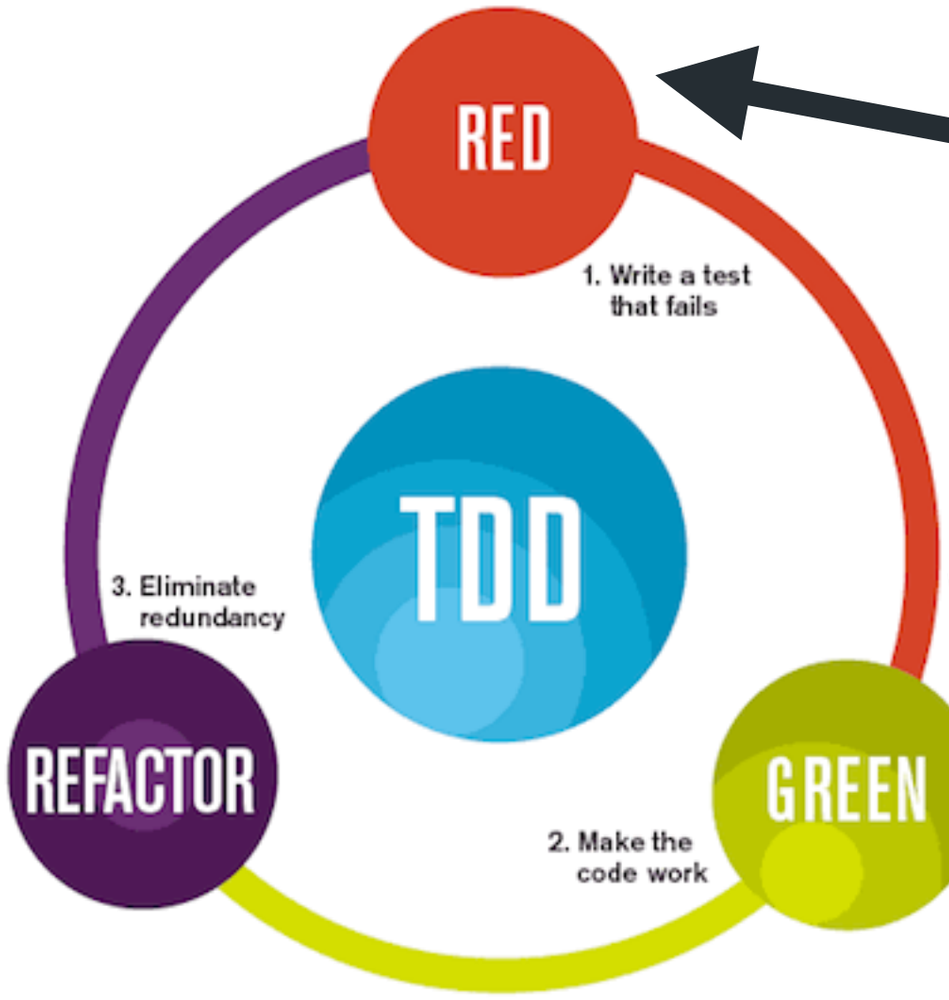
filter

reduce

```python
original = [0,1,2,3,4,5]

# квадраты
result = map(lambda x: x*x, original)
print(list(result))

# только четные
result = filter(lambda x: x % 2 == 0, original)
print(list(result))

# сумма квадратов
from functools import reduce
result = reduce(lambda x, y: x+y*y, original, 0)
print(result)
```

Разработка через тестирование

Три закона TDD



Минимальный работающий код

```
def square(x):
    '''Return the square of x.'''
    return x * x

if __name__ == '__main__':
    assert square(2) == 4
    assert square(-2) == 4
```

```python
def square(x):
    '''Return the square of x.

    >>> square(2)
    4
    >>> square(-2)
    4
    '''
    return x * x

if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

О тестировании

- Хороший тест — удобочитаемый тест
- Используйте предметно-ориентированный язык
- Один тест — одна проверка
- Принципы F.I.R.S.T:
  - fast,
  - independent,
  - repeatable,
  - self-validating,
  - timely

Чистый код

refactoring.guru

NB: Правила не должны противоречить здравому смыслу

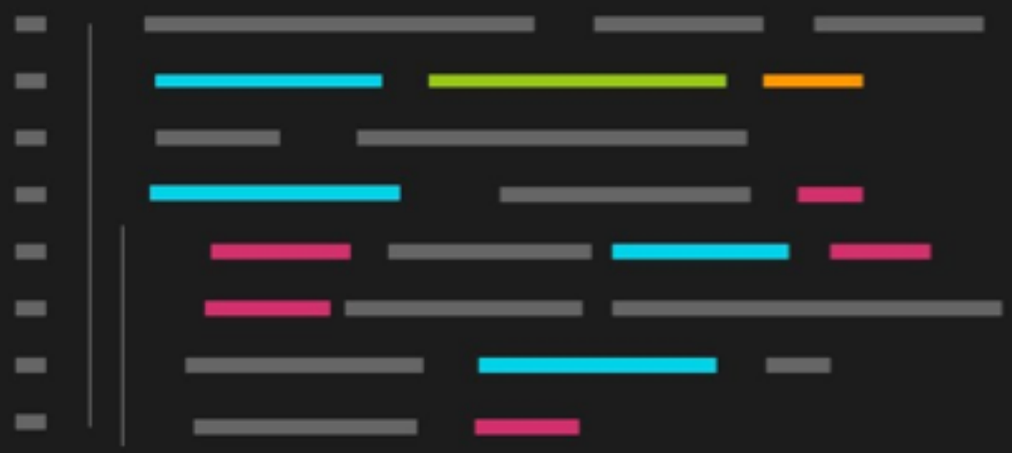
Имена

- Покажите намерения
- Уберите «магические числа»
- Не используйте ложные ассоциации
- Используйте английский
- Избегайте избыточности
- Сделай имена удобными для чтения
- Сделайте имена удобными для поиска
- Укажите префиксом контекст имени (если надо)
- Не заставляйте думать читателя: уберите шутки, каламбуры, трюки
- Будьте последовательны
- Классы = существительные, методы = глаголы
- Используйте термины задачи
- Используйте термины решения

Функции

- Функции должны быть компактными
- Избегайте глубокой вложенности
- Функция должна выполнять одну операцию
- Функция должно работать на одном уровне абстракций
- Повествование сверху вниз
- У функции должно быть понятное имя
- Чем меньше аргументов, тем лучше
- Не используйте аргументы для результата
- Разделяйте синтаксически функции: запросы, команды, события
- У функции не должно быть побочных эффектов
- Исключения лучше кодов ошибок

Три ключевых принципа хорошего кода



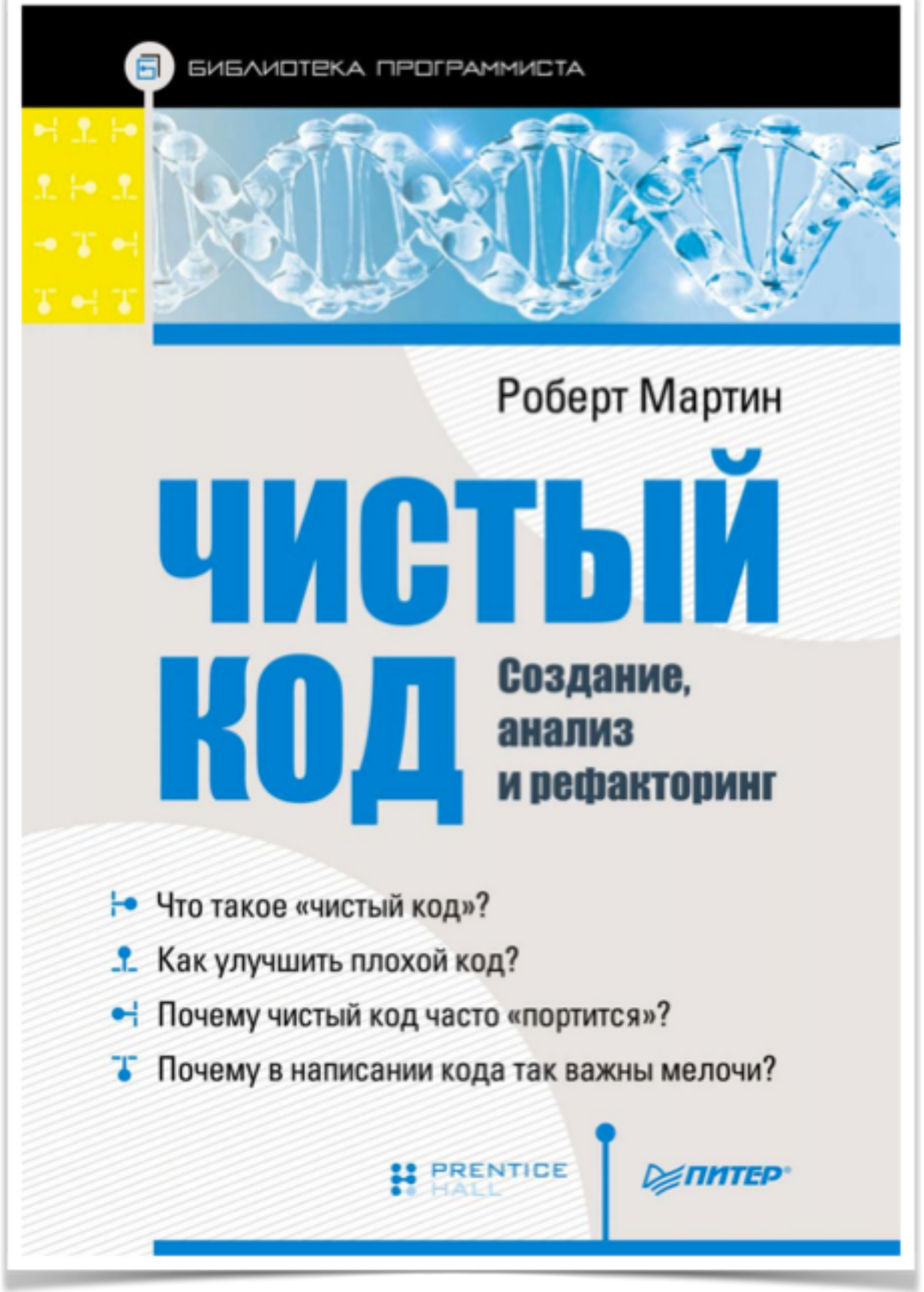
DRY, KISS, YAGNI

Комментарии

- Пишите так, чтобы комментарии не нужны были
- Напишите предисловие функции (abstract)
- Если встречается неожиданность, то объясните
- Предупредите, если это важно или опасно
- Используйте метки: TODO
- Уберите бред, ложь, шум
- Если есть контроль версий, то не над писать: историю, авторов, закомментированный код
- Комментарий должен быть компактный, локальный и очевидный

Форматирование

- Не спагетти-код:
  - код должен быть локальным
  - Код должен быть последовательным
- Ширина строки 80/120
- Горизонтальное выравнивание вносит мало порядка
- Отступы — добро!
- Смешанные отступы — зло


БИБЛИОТЕКА ПРОГРАММИСТА
Роберт Мартин
ЧИСТЫЙ
КОД
Создание, анализ и рефакторинг
Что такое «чистый код»?
Как улучшить плохой код?
Почему чистый код часто «портится»?
Почему в написании кода так важны мелочи?
PRENTICE HALL
ПИТЕР