



Распространенные ошибки при написании юнит-тестов

Катерина Павленко

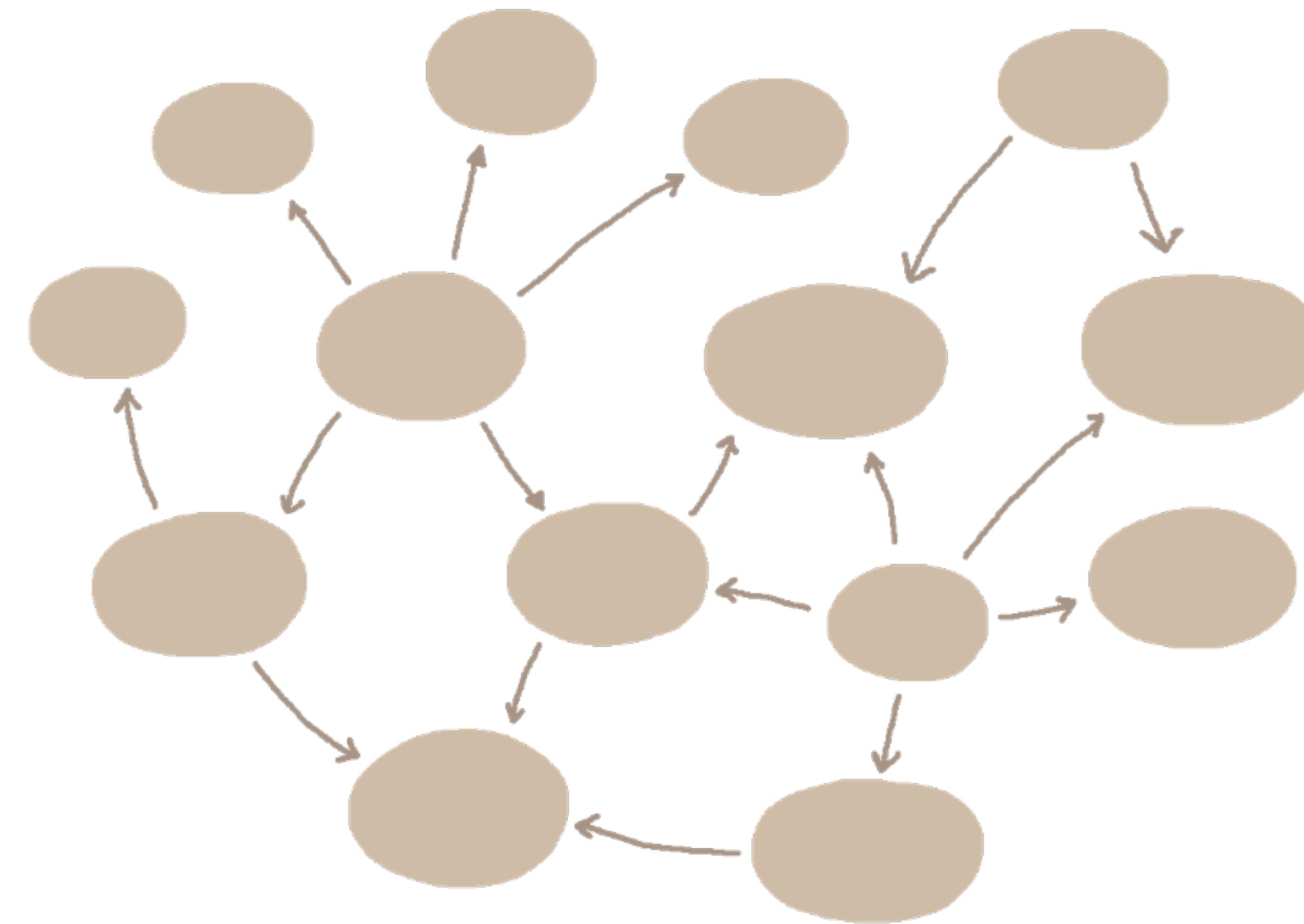
Катерина Павленко

Frontend developer
at **Tinkoff.ru**

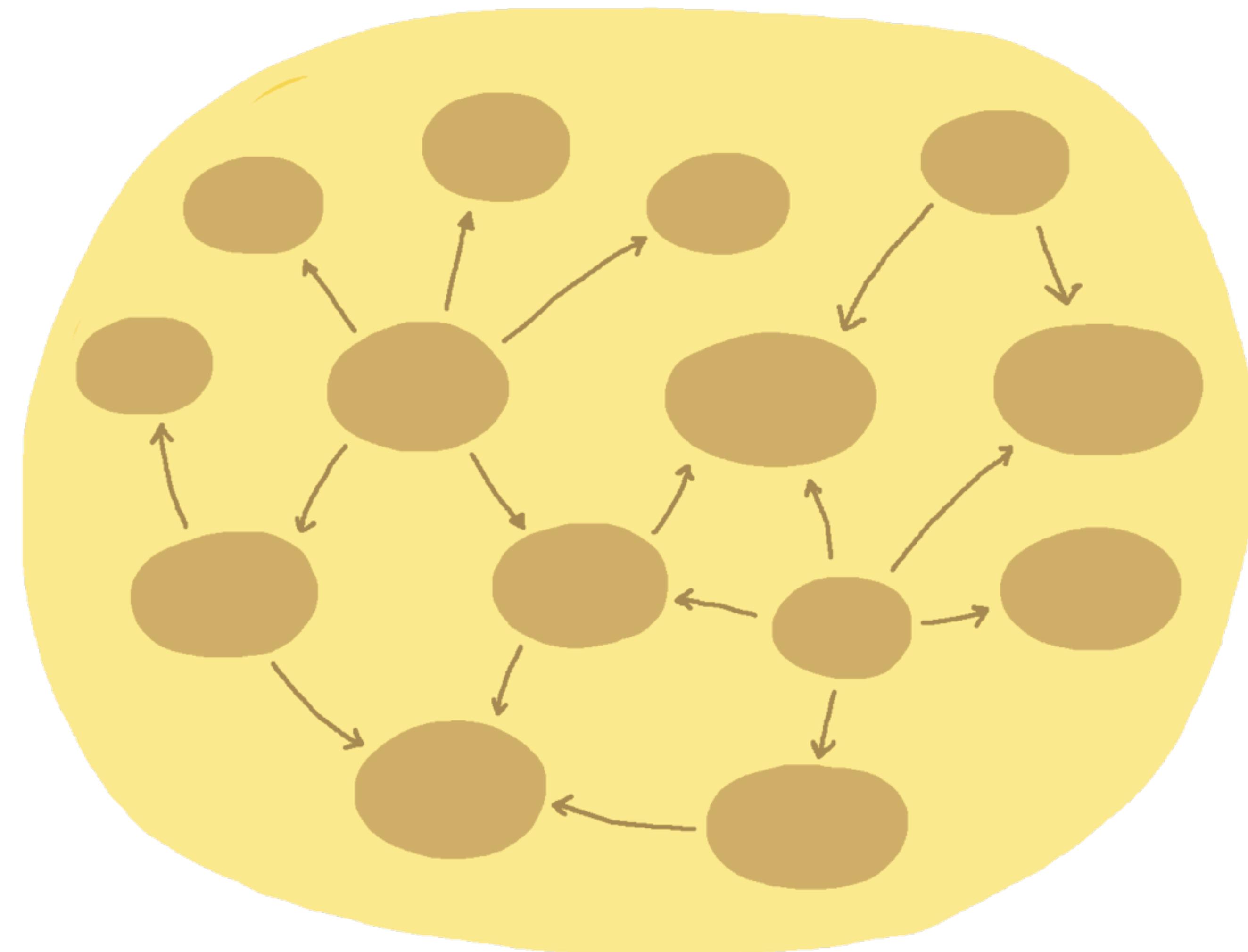
@cakeinpanic
pavlenko.katerina@gmail.com



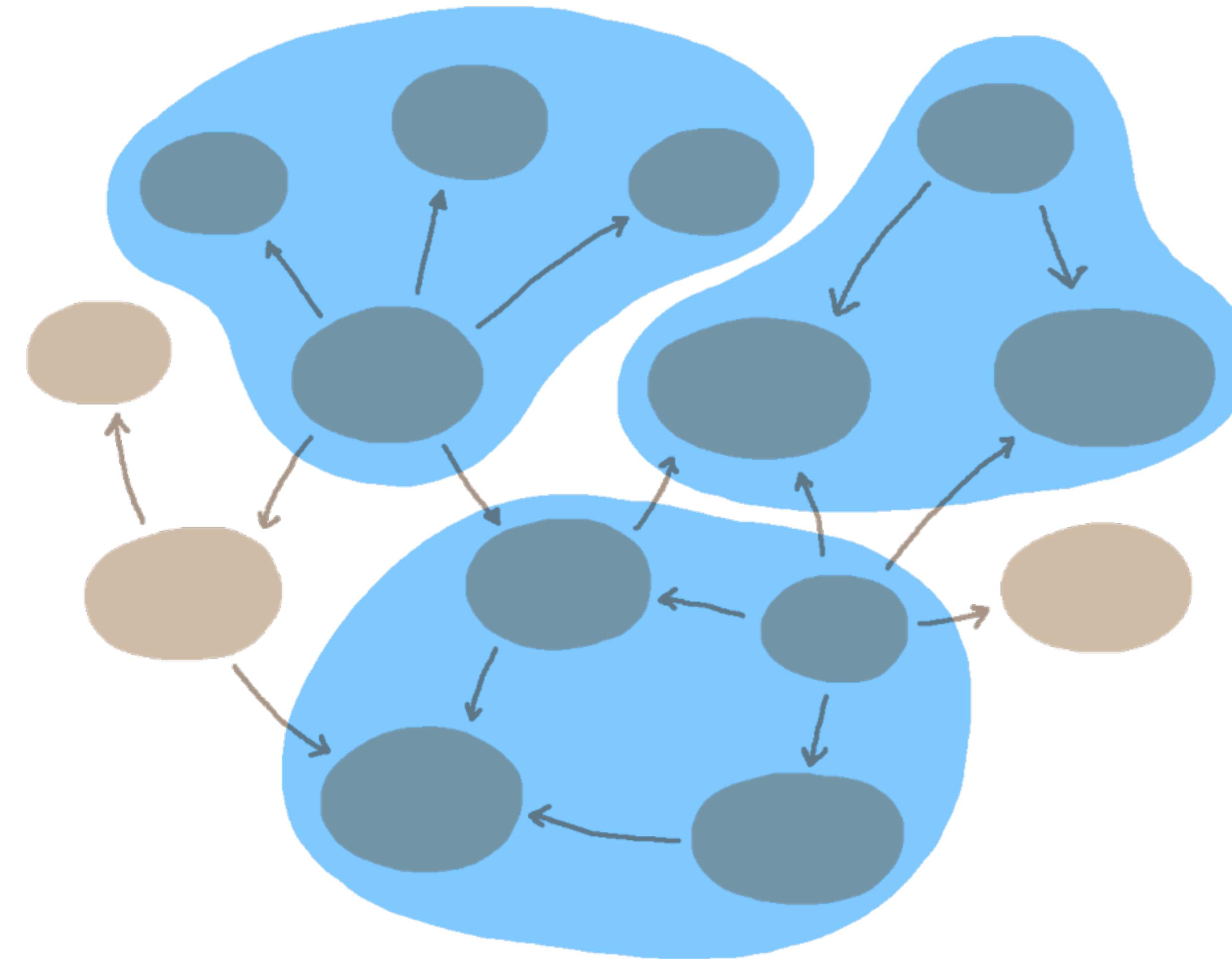
Приложение



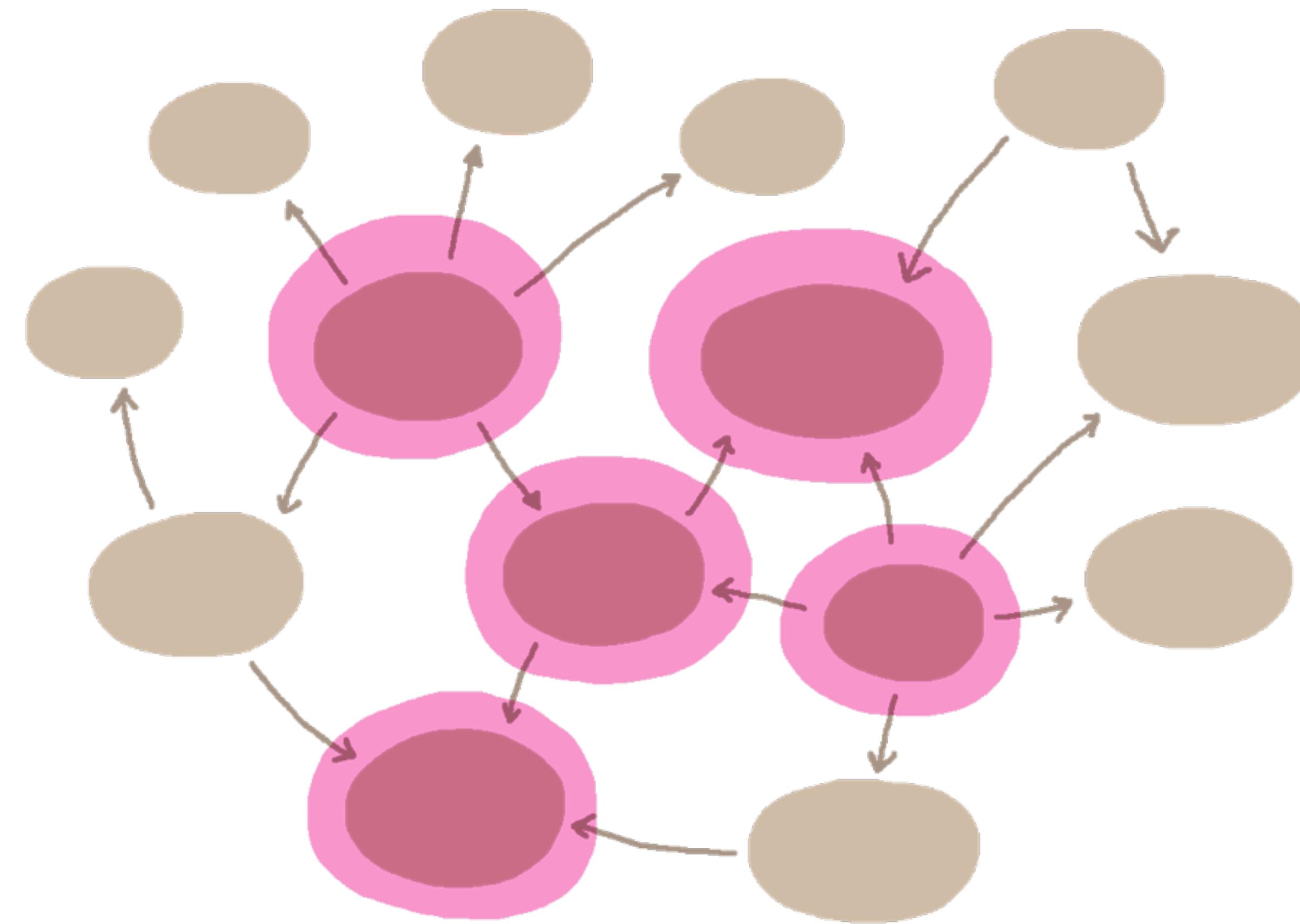
e2e-тесты



Интеграционные тесты



ЮНИТ-ТЕСТЫ





Зачем нужны юнит-тесты

- Убедиться в том, что каждый модуль корректно выполняет задачу, для которой он написан
- Безопасный рефакторинг
- Дополнительная документация

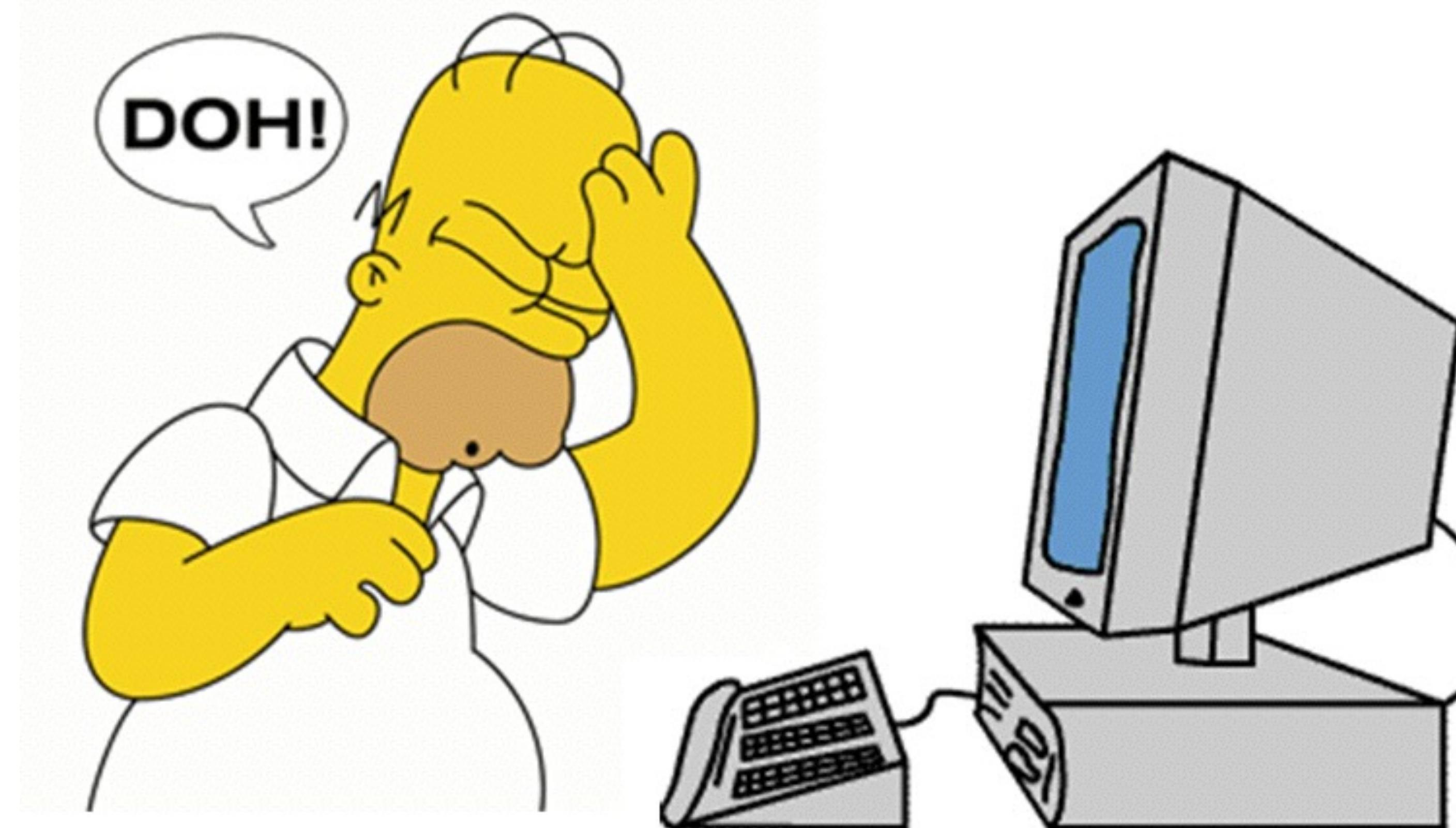
Хорошие unit-тесты

Заслуживают доверия

Легко сопровождать

Легко понять

Устал писать юнит-тесты

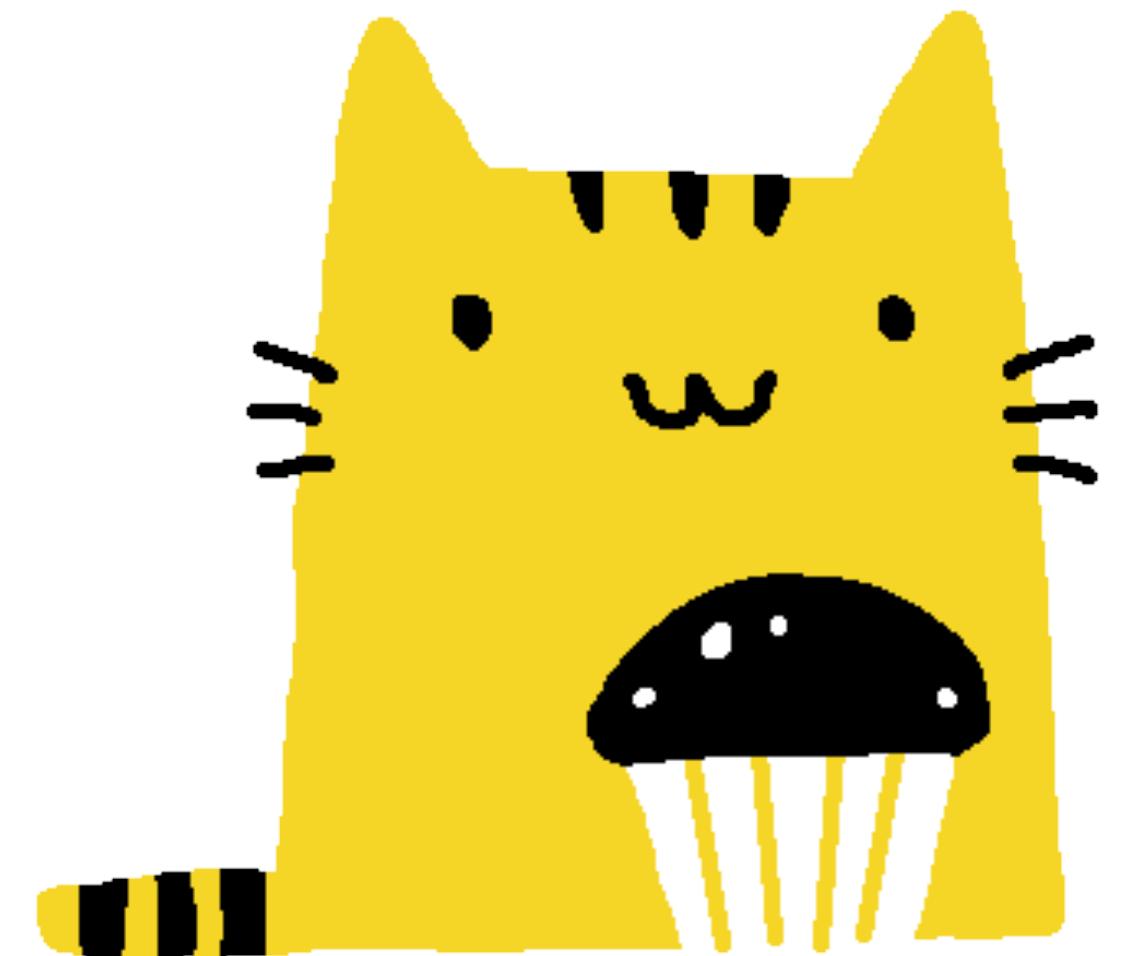


Тесты писать сложно

- Плохо тестируемый код
- Неправильный подход к написанию тестов

Поговорим о

- Тестирование приватных методов
- Тестирование зависимостей
- 100% покрытие
- Test driven development



Приватные методы

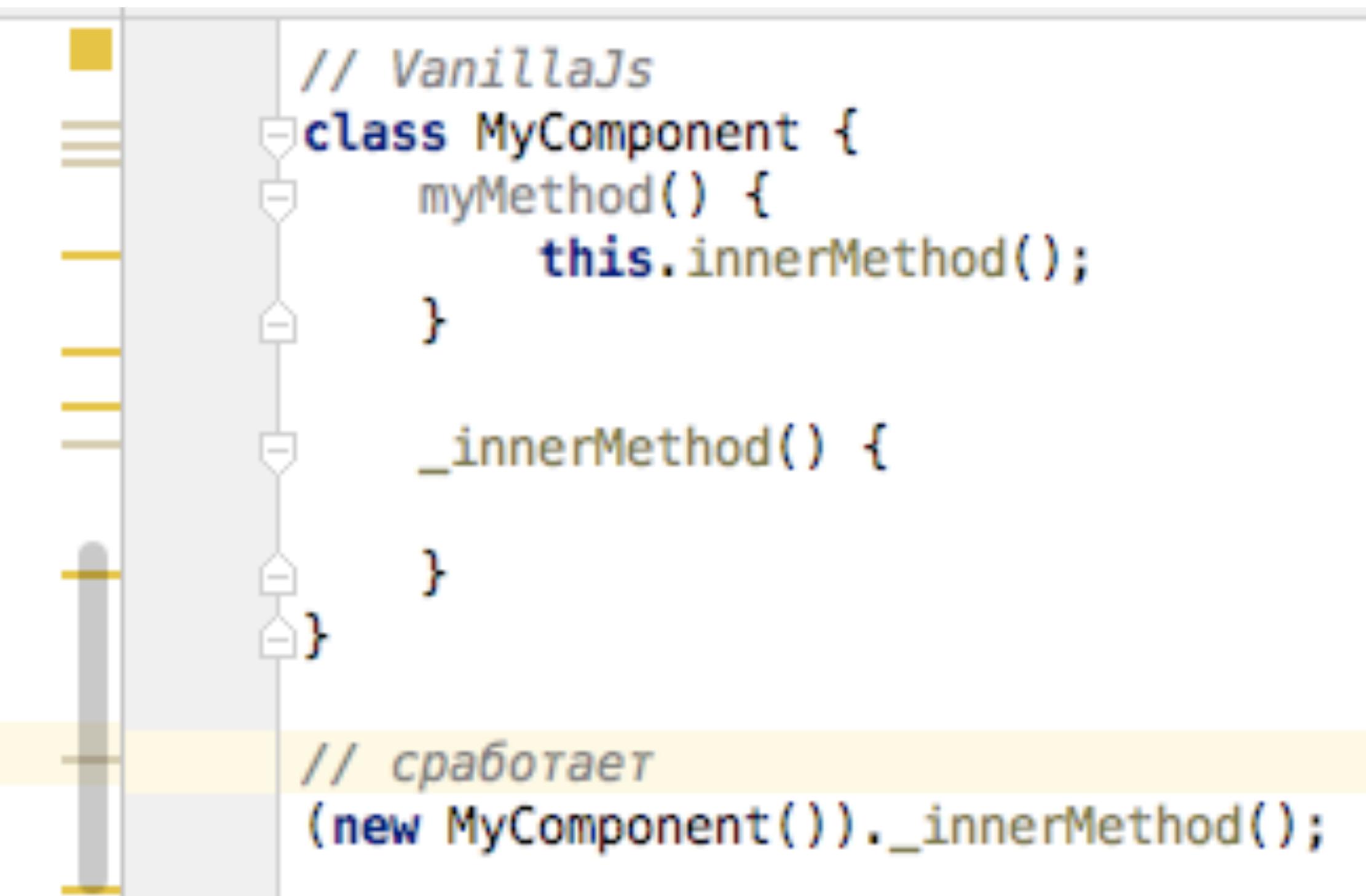
Приватные методы

```
// TypeScript
class MyComponent {
    public myMethod() {
        this.innerMethod();
    }

    private innerMethod() {

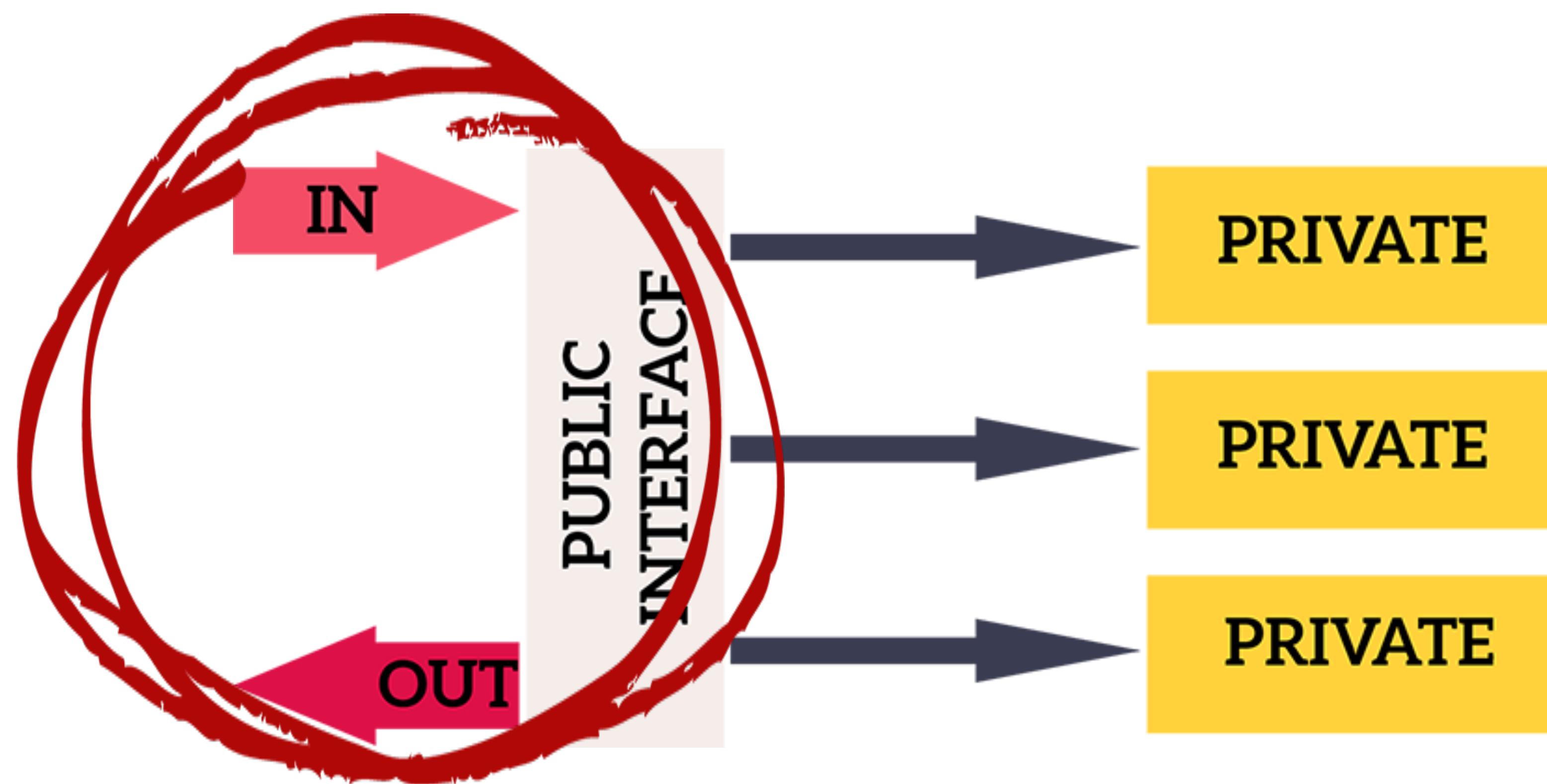
    }
}

// не сработает
(new MyComponent()).innerMethod();
```



Зачем нужны приватные методы

- Переиспользуемые участки кода
- Разделение на несколько методов для облегчения читаемости
- Никогда не пригодится снаружи





**Everytime you test a private method,
a unicorn dies.**

```
class CoffeeShop {  
    public brewedCups: number;  
    private _beans: number;  
  
    public makeCoffeeCup() {  
        this._buyBeans(1);  
        this._brewOneCup();  
    }  
  
    private _buyBeans(amount: number) {  
        this._beans += amount;  
    }  
  
    private _brewOneCup() {  
        if (this._beans < 1) {  
            throw Error('No beans((((');  
        }  
        this._beans--;  
        this.brewedCups++;  
    }  
}
```



```
it('method makeCoffeeCup calls buyBeans and brewOneCup', () => {
  testComponent.makeCoffeeCup();
  expect(testComponent._buyBeans).toHaveBeenCalledWith(1);
  expect(testComponent._brewOneCup).toHaveBeenCalled();
});
```

```
it('method makeCoffeeCup calls buyBeans and brewOneCup', () => {
  testComponent.makeCoffeeCup();
  expect(testComponent._buyBeans).toHaveBeenCalledWith(1);
  expect(testComponent._brewOneCup).toHaveBeenCalled();
});
```

```
it('method buyBeans increases beans amount', () => {
  testComponent._beans = 0;
  testComponent.buy();
  expect(testComponent._beans).toEqual(2);
});
```

```
it('method brewOneCup increases cups amount and decreases beans amount', () => {
  testComponent.brewedCups = 1;
  testComponent._beans = 1;
  testComponent.brew();
  expect(testComponent._beans).toEqual(0);
  expect(testComponent.brewedCups).toEqual(2);
});
```

```
it('method brewOneCup throws an error if there is 0 beans left', () => {
  testComponent._beans = 0;
  expect(testComponent._brewOneCup()).toThrowError('No beans!!!');
});
```

Сложно доверять

Оригинал

```
public makeCoffeeCup() {  
    this._buyBeans(1);  
    this._brewOneCup();  
}
```

Работает нестабильно,
тесты проходят

```
public makeCoffeeCup() {  
    this._brewOneCup();  
    this._buyBeans(1);  
}
```

Работает,
тесты падают

```
public makeCoffeeCup() {  
    this._buyBeans(10);  
    this._brewOneCup();  
}
```

Не работает,
тесты проходят

```
public makeCoffeeCup() {  
    this._beans = -1;  
    this._buyBeans(1);  
    this._brewOneCup();  
}
```

Сложно доверять

Сложно поддерживать

Работает, тесты падают

```
public makeCoffeeCup() {  
    this.brewedCups++;  
}
```

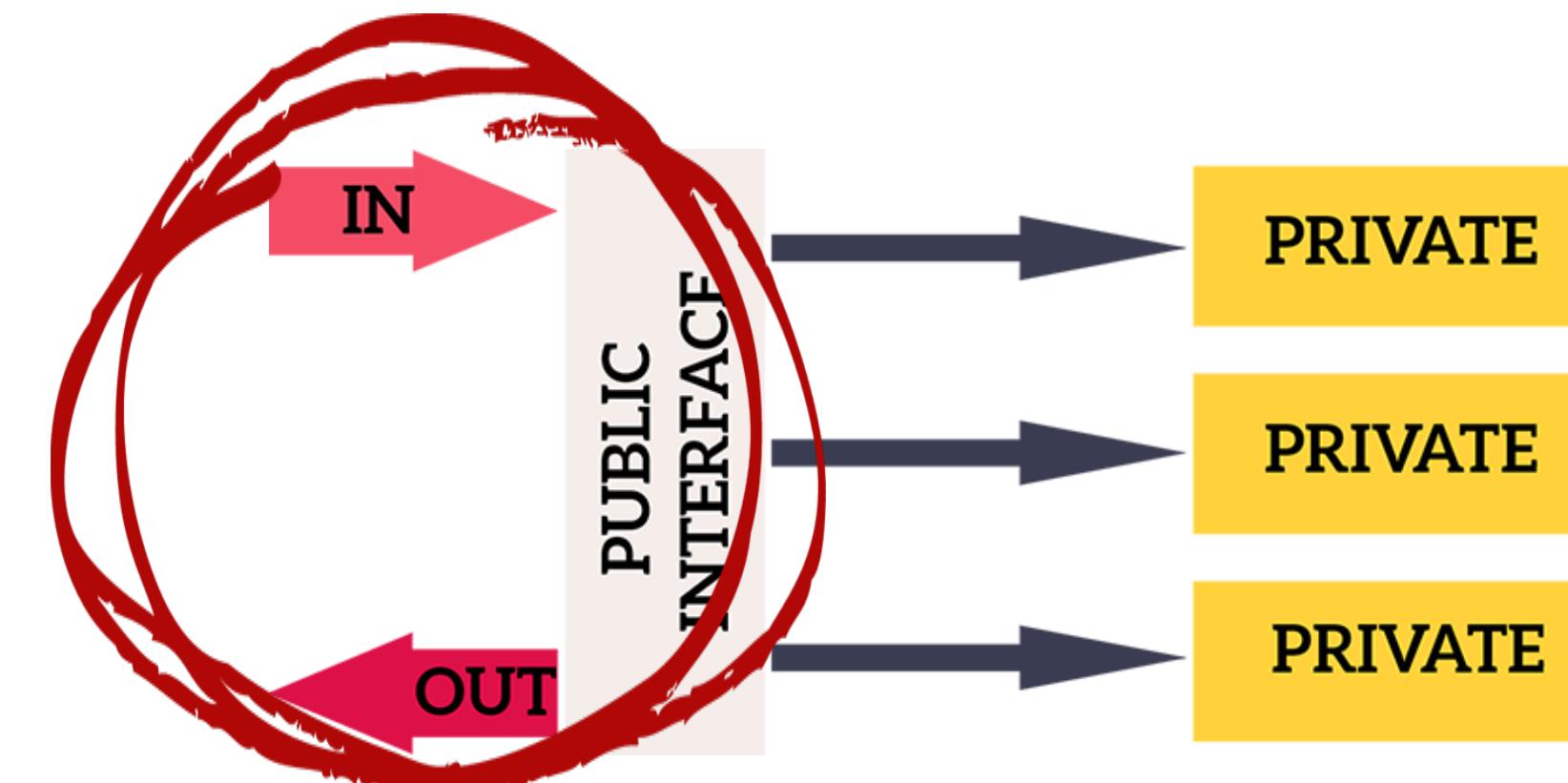


Тестируем логику, а не реализацию

- Если внешнее API не меняется – тесты падать не должны

Тестируем логику, а не реализацию

```
it('method makeCoffeeCup increases number of cups by one', () => {
  testComponent.brewedCups = 1;
  testComponent.makeCoffeeCup();
  expect(testComponent.brewedCups).toEqual(2);
});
```

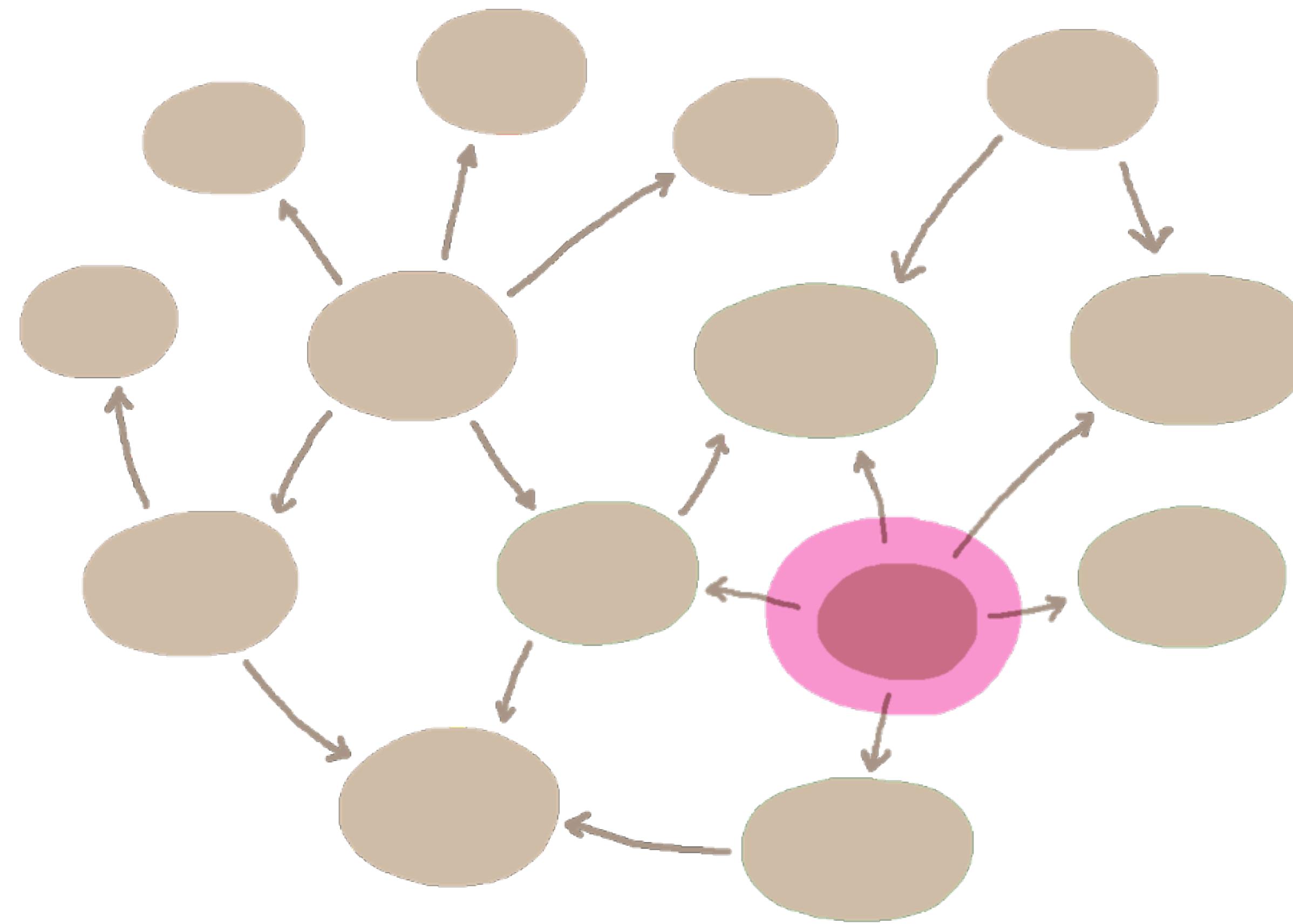


Тестирование приватных методов

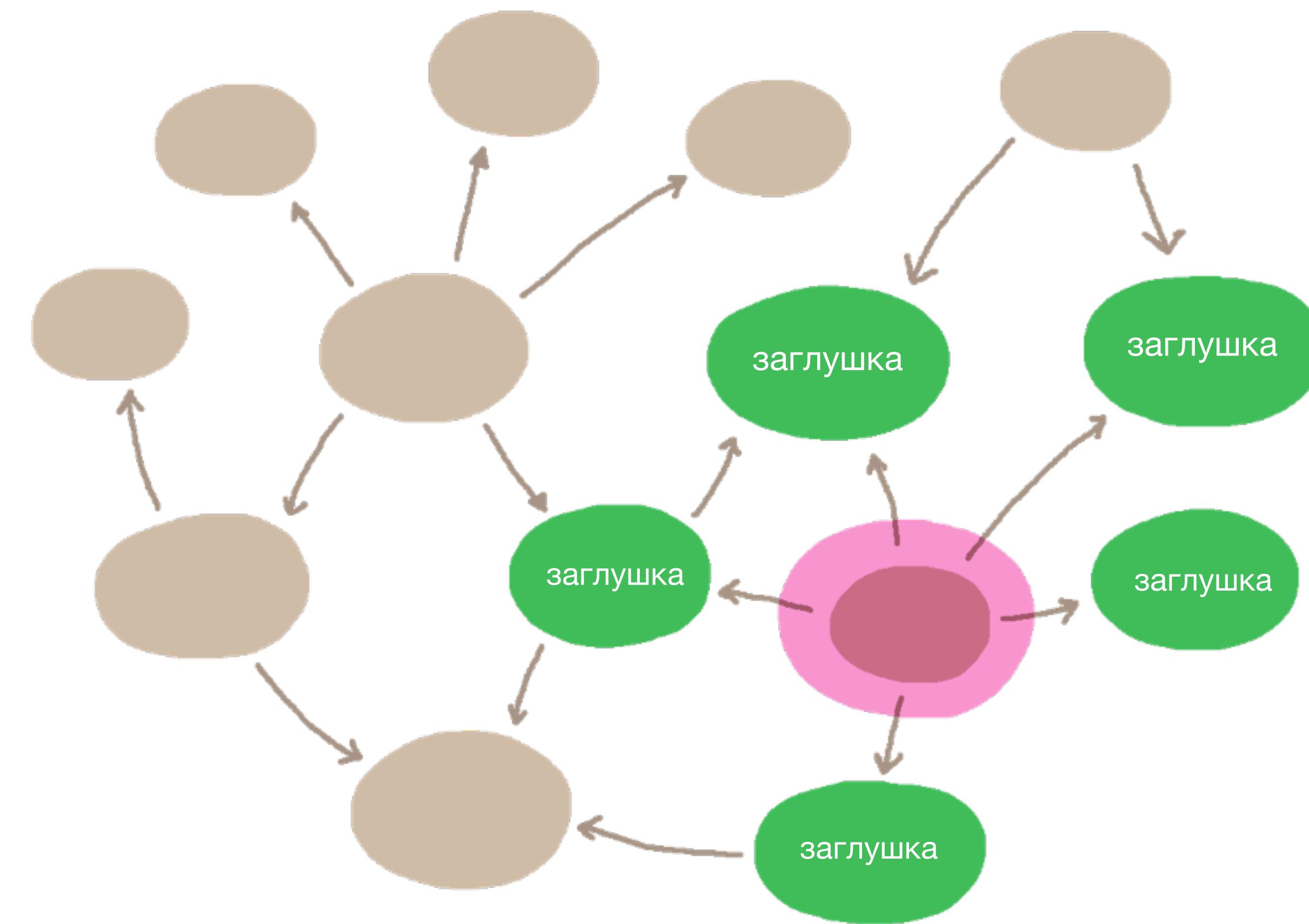
- Тестируем логику, а не реализацию
- Тестируем только публичные методы
- Не делать методы публичными ради тестов. Это убивает инкапсуляцию
- Если логику можно вынести в отдельный класс – почему бы и нет?

Тестирование зависимостей

ЮНИТ-ТЕСТЫ



Мокирование



```
class CityService {
  getCities() {
    return Promise.resolve([
      {name: 'Saint-Petersburg'},
      {name: 'Moscow'}
    ])
  }
}

class CoffeeShop {
  cityListService = new CityService();

  getCityNames() {
    return this.cityListService
      .getCities()
      .then(cities => cities.map((city) => city.name))
  }
}
```

Проверяем, что из списка объектов
корректно получается список строк

```
it('get cities names correctly', async () => {
  let cityNamesArray = await testCoffeeShop.getCityNames();
  expect(cityNamesArray).toBe(['Saint-Petersburg', 'Moscow']);
});
```

Сложно поддерживать

Теперь наш тест ходит в интернет....

```
class CityService {  
    getCities() {  
        return fetch('/cities').then(response => response.json());  
    }  
}
```

а мы все еще хотим проверять,
что из списка объектов
корректно получается список строк



```
class CityService {
    userService = new UserService();

    getCurrentCity() {
        return this.userService.getUser().cityName;
    }
}

class CoffeeShop {
    cityService = new CityService();

    getDiscount() {
        return this.cityService.getCurrentCity() === 'Moscow';
    }
}
```

```
class UserServiceMock {
    getUser() {
        return {cityName: 'Moscow'};
    }
}

it('calculates discount for Moscow', async () => {
    expect(testComponent.getDiscount()).toBe(true);
});
```

Сложно доверять

Сложно понять

```
class CoffeeShop {  
    cityService = new CityService();  
  
    getDiscount() {  
        return this.cityService.getCurrentCity() === 'Moscow';  
    }  
}
```

```
class UserServiceMock {  
    getUser() {  
        return {cityName: 'Moscow'};  
    }  
}  
  
it('calculates discount for Moscow', async () => {  
    expect(testComponent.getDiscount()).toBe(true);  
});
```



Замокирем удобно

```
class CityListServiceMock {
    getCities() {
        return [{name: 'Test-Petersburg'}];
    }

    getCurrentCity() {
        return 'Moscow';
    }
}

it('get cities names correctly', async () => {
    let result = await testComponent.getCityNames();
    expect(result).toBe(['Test-Petersburg']);
});

it('calculates discount for Moscow', async () => {
    expect(testComponent.getDiscount()).toBe(true);
});
```

```
class CoffeeShop {
    cityService = new CityService();

    getCityNames() {
        return this.cityService
            .getCities()
            .then(cities => cities.map((city) => city.name));
    }

    getDiscount() {
        return this.cityService.getCurrentCity() === 'Moscow';
    }
}
```

Мокирование

- Если зависимость сейчас маленькая и простая, то потом нет гарантий, что она не изменится
- Тесты, рассчитывающие на логику зависимостей, хрупкие и нестабильные

100% coverage

```
△ ⊖ public boolean addAll(int index, Collection c) {  
◆ if(c.isEmpty()) {  
◆ ◆ return false;  
◆ } else if(_size == index || _size == 0) {  
◆ ◆ return addAll(c);  
◆ } else {  
◆ ◆ Listable succ = getListableAt(index);  
◆ ◆ Listable pred = (null == succ) ? null : succ.prev();  
◆ ◆ Iterator it = c.iterator();  
◆ ◆ while(it.hasNext()) {  
◆ ◆ ◆ pred = insertListable(pred, succ, it.next());  
◆ ◆ }  
◆ ◆ return true;  
◆ }  
}
```

◆ 1 of 2 branches missed.

Press 'F2' for focus

```
class CoffeeShop {
    makeCoffee(coffeeType) {
        switch(coffeeType) {
            case 'macchiato': return 'MACCHIATO';
            case 'cappuchino': return 'CAPPUCHINO';
            case 'lattee': return 'LATTE';
            default: return 'AMERICANO';
        }
    }
}

it('makes americano by default', () => {
    let coffee = testCoffeeShop.makeCoffee();
    expect(coffee).toEqual('AMERICANO');
});
```

Сложно доверять

```
class CoffeeShop {  
    makeCoffee(coffeeType) {  
        switch(coffeeType) {  
            case 'macchiato': return 'MACCHIATO';  
            case 'cappuchino': return 'CAPPUCHINO';  
            case 'lattee': return 'LATTE';  
            default: return 'AMERICANO';  
        }  
    }  
}  
  
it('makes americano by default', () => {  
    testCoffeeShop.makeCoffee('macchiato');  
    testCoffeeShop.makeCoffee('cappuchino');  
    testCoffeeShop.makeCoffee('lattee');  
    let coffee = testCoffeeShop.makeCoffee();  
    expect(coffee).toEqual('AMERICANO');  
});
```



100% coverage

- Это инструмент, но не слепая метрика
- Покрывать нужно не просто строки код, а сценарии вызова методов

TDD
Test driven development

TDD: Мифы

- Надо писать все-все-все тесты, а только потом реализацию

TDD: Правда

- Надо писать итеративно:
 - маленький тест -> немного кода
 - маленький тест -> немного кода
 - маленький тест -> немного кода

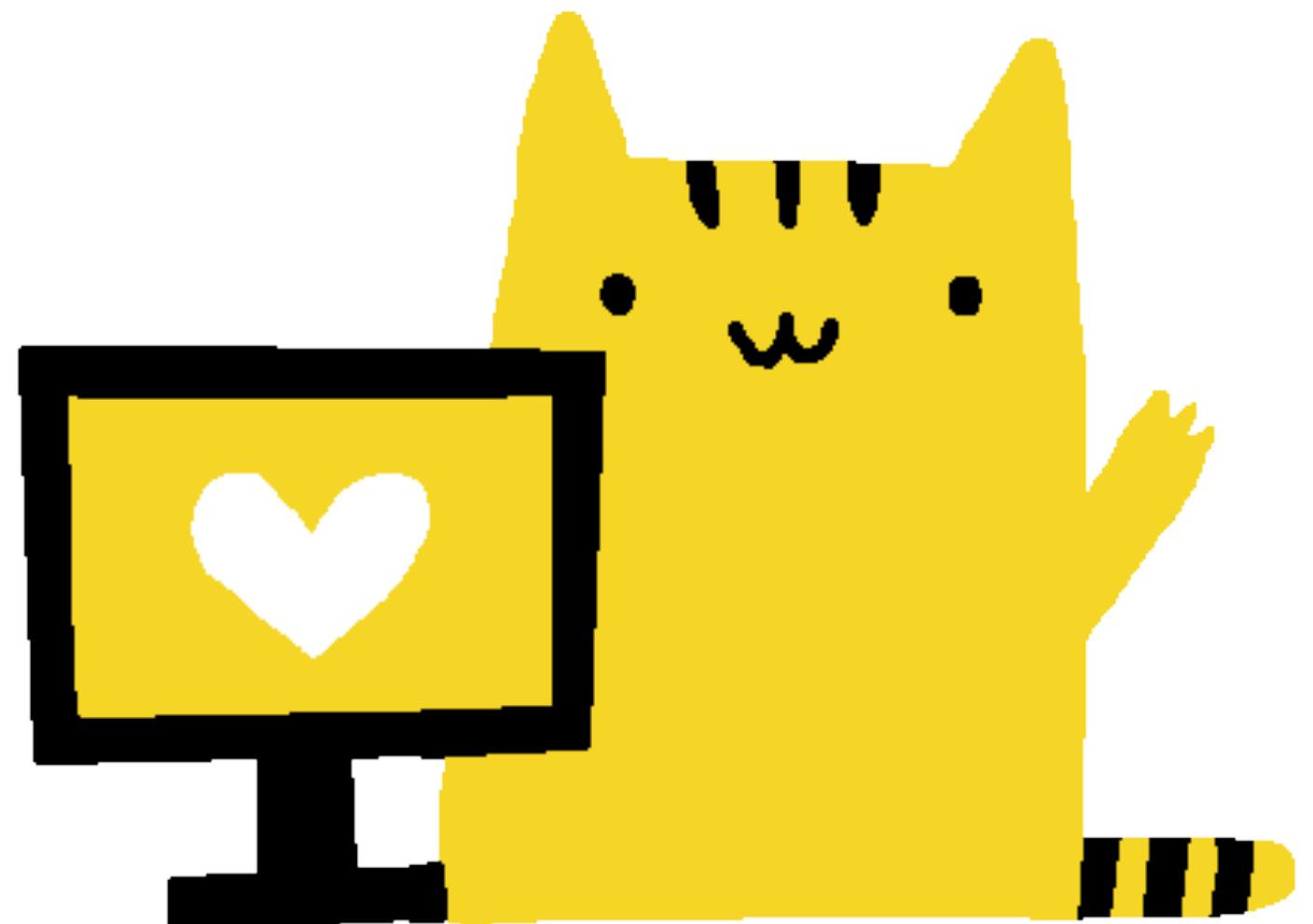
TDD

- TDD нужно применять только тогда, когда вы понимаете, что именно тут это выгодно
- Вы 100% знаете требования к коду, который вы планируете писать сейчас

Выводы

Выводы

- Тесты тоже код, их надо поддерживать
- Всегда закладываемся на расширяемость
- Тестируем логику, а не реализацию
- Руководствуемся здравым смыслом



Ссылка на все ссылки



Вопросы

