SOLID-принципы с примерами на PHP

Андрей Нестер, Минск
Senior Software Engineer @ SugarCRM, Intetics

http://www.devconf.ru

Что такое SOLID принцины?

Single responsibility
Open-closed
Liskov substitution
Interface segregation
Dependency inversion

Single Responsibility

На каждый объект должна быть возложена одна единственная обязанность

Single Responsibility

Плохой код

```
class Order
{
    public function calculateTotalSum(){/*...*/}
    public function getItems(){/*...*/}
    public function getItemCount(){/*...*/}
    public function addItem($item){/*...*/}
    public function deleteItem($item){/*...*/}

    public function printOrder(){/*...*/}
    public function showOrder(){/*...*/}

    public function load(){/*...*/}
    public function save(){/*...*/}
    public function update(){/*...*/}
    public function delete(){/*...*/}
}
```

Single Responsibility

Почему плохой код?

Такой код сложнее в сопровождении.

Каждая ответственность — причина для изменений, а изменения потенциально опасны.

Каскадные изменения тесносвязанных ответственностей.

Single Responsibility

Хороший код

```
class Order
{
    public function calculateTotalSum(){/*...*/}
    public function getItems(){/*...*/}
    public function getItemCount(){/*...*/}
    public function addItem($item){/*...*/}
    public function deleteItem($item){/*...*/}
}
```

```
class OrderRepository
{
    public function load(){/*...*/}
    public function save(){/*...*/}
    public function update(){/*...*/}
    public function delete(){/*...*/}
}
```

```
class OrderViewer
{
    public function printOrder(){/*...*/}
    public function showOrder(){/*...*/}
}
```

Single Responsibility

Реальный пример нарушения

PHPMailer — класс содержит 3925 строк кода и порядка 80 публичных методов

CodeIgniter — Большое количество классов, нарущающих SRP.

Open-closed

Программные сущности должны быть открыты для расширения, но закрыты для модификации

Open-closed

Плохой код

```
class OrderCalculator
{
    public function calculate($orders)
    {
        $sum = 0;
        foreach ($orders as $order) {
            if ($order instanceof SingleOrder) {
                $sum += $this->calculateSingle($order);
            }

            if ($order instanceof MultiOrder) {
                $sum += $this->calculateMulti($order);
            }
        }

        return $sum;
    }

    private function calculateSingle(SingleOrder $order) {}
    private function calculateMulti(MultiOrder $order) {}
}
```

Open-closed

Почему плохой код?

Плохо расширяемый код.
Его сложно повторно использовать.
Необходимо менять OrderCalculator для новых Order.

Open-closed
Хороший код

```php
class OrderCalculator
{
    /**
     * @param OrderInterface[] $orders
     */
    public function calculate($orders)
    {
        $sum = 0;
        foreach ($orders as $order) {
            $sum += $order->calculate();
        }

        return $sum;
    }
}

interface OrderInterface
{
    public function calculate();
}

class SingleOrder implements OrderInterface {}
class MultiOrder implements OrderInterface {}
```

Open-closed

Реальный пример нарушения

Symfony 2

```
public function test($test)
{
    switch ($this->operator) {
        case '>':
            return $test > $this->target;
        case '>=':
            return $test >= $this->target;
        case '<':
            return $test < $this->target;
        case '<=':
            return $test <= $this->target;
        case '!=':
            return $test != $this->target;
    }

    return $test == $this->target;
}
```

Open-closed
Реальный пример - Исправленный

```
class Comparator
{
    // ...

    public function test($test)
    {
        return $this->comparatorOperation->compare($this->target, $test);
    }
}

abstract class ComparatorOperation
{
    abstract function compare($a, $b);
}

class EqualOperation extends ComparatorOperation {}
class LessOrEqualOperation extends ComparatorOperation {}
```

Liskov substitution

Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.

Liskov substitution

Плохой код (пока ещё хороший)

```php
class Order
{
    protected $items = [];

    public function addItem(Item $item)
    {
        $this->items[] = $item;
    }

    public function getItems()
    {
        return $this->items;
    }
}

class OrderCollector
{
    public function collect(Order $order, $items)
    {
        foreach ($items as $item) {
            $order->addItem($item);
        }
    }
}
```

Liskov substitution

Плохой код

```
class FreeOrder extends Order
{
    public function addItem(Item $item)
    {
        if ($item->price() !== 0) {
            throw new Exception();
        }

        $this->items[] = $item;
    }
}
```

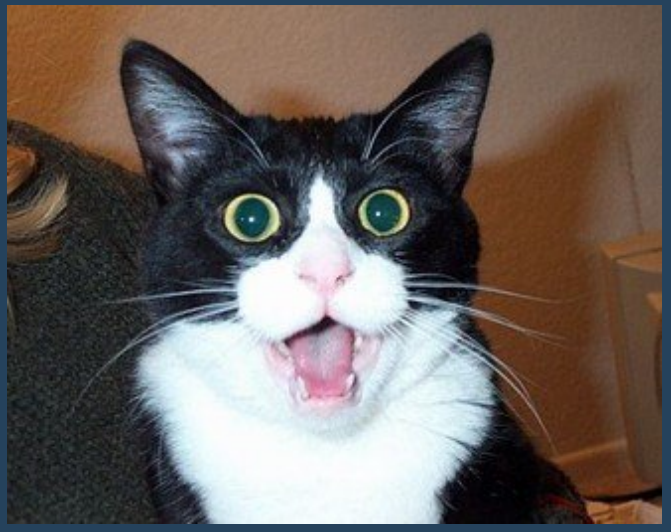
Liskov substitution

Почему плохой код?

При использовании FreeOrder “ломается” OrderCollector.

FreeOrder не является настоящим подклассом Order.

Подкласс должен определяться на основе поведения.



Liskov substitution

Хороший код

```php
abstract class OrderList
{
    protected $items = [];

    public function getItems()
    {
        return $this->items;
    }
}
```

```php
class Order extends OrderList
{
    public function addItem(Item $item)
    {
        $this->items[] = $item;
    }
}
```

```php
class FreeOrder extends OrderList
{
    public function addItem(FreeItem $item)
    {
        $this->items[] = $item;
    }
}
```

Liskov substitution

Реальный пример нарушения

Symfony 2

```
class FrozenParameterBag extends ParameterBag
{
    // ...

    public function clear()
    {
        throw new LogicException('Impossible to call clear() on a frozen ParameterBag.');
    }

    // ...
}
```

Interface segregation

Много специализированных интерфейсов лучше, чем один универсальный

Interface segregation

Плохой код

```
interface ItemInterface
{
    public function applyDiscount($discount);
    public function applyPromocode($promocode);

    public function setColor($color);
    public function setSize($size);

    public function setCondition($condition);
    public function setPrice($price);
}
```

Interface segregation

Почему плохой код?

Интерфейс слишком большой, что затрудняет создание классов, реализующих его.

Большие интерфейсы не так удобно повторно использовать.

Вероятно нарушение SRP и LSP при реализации интрефейса

Interface segregation

Хороший код

```php
interface ItemInterface
{
    public function setCondition($condition);
    public function setPrice($price);
}

interface ClothesInterface
{
    public function setColor($color);
    public function setSize($size);
    public function setMaterial($material);
}

interface DiscountableInterface
{
    public function applyDiscount($discount);
    public function applyPromocode($promocode);
}
```

Interface segregation

Реальный пример нарушения

Laravel 5

```
interface Authenticatable
{
    public function getAuthIdentifierName();

    public function getAuthIdentifier();

    public function getAuthPassword();

    public function getRememberToken();

    public function setRememberToken($value);

    public function getRememberTokenName();
}
```

Dependency Inversion

Зависимости должны строится относительно абстракций, а не деталей

Dependency Inversion

Плохой код

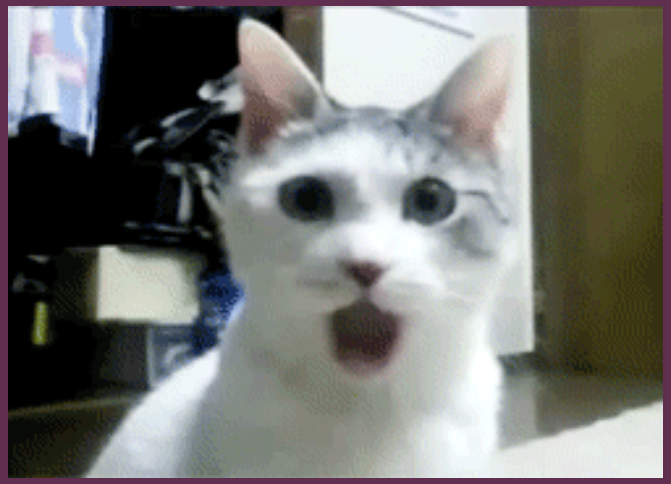
```php
class Customer
{
    private $currentOrder = null;

    public function buyItems()
    {
        if (is_null($this->currentOrder)) {
            return false;
        }

        $processor = new OrderProcessor();
        return $processor->checkout($this->currentOrder);
    }
}

class OrderProcessor
{
    public function checkout($order){/*...*/}
}
```

Dependency Inversion

Почему плохой код?

Зависимости от деталей приводит к снижению гибкости.

Такой код тяжелее тестировать.



Dependency Inversion

Хороший код

```
class Customer
{
    private $currentOrder = null;

    public function buyItems(OrderProcessorInterface $processor)
    {
        if(is_null($this->currentOrder)){
            return false;
        }

        return $processor->checkout($this->currentOrder);
    }
}

interface OrderProcessorInterface
{
    public function checkout($order);
}

class OrderProcessor implements OrderProcessorInterface
{
    public function checkout($order){/*...*/}
}
```

Dependency Inversion

Реальный пример нарушения

Laravel 5

```
class Dumper
{
    public function dump($value)
    {
        if (class_exists(CliDumper::class)) {
            $dumper = 'cli' === PHP_SAPI ? new CliDumper : new HtmlDumper;

            $dumper->dump((new VarCloner)->cloneVar($value));
        } else {
            var_dump($value);
        }
    }
}
```

Заключение

SOLID — не панацея

”Любую проблему можно решить с помощью дополнительных абстракций, кроме проблемы избыточных абстракций”

Главное — управление сложностью кода

Контакты

Андрей Нестер

andrew.nester.dev@gmail.com

GitHub: @andrewnester

Вопросы?