Object Calisthenics:

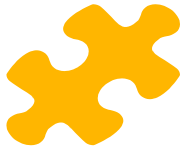
9 regras para melhorar seu código

Olá!

Me chamo Jéssica

Programadora na RD, host do PodProgramar e mentora

Meus contatinhos em: jessicazanelato.dev

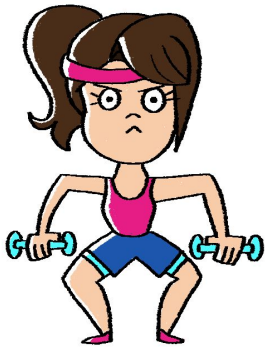


Object?

= Programação Orientada a Objetos

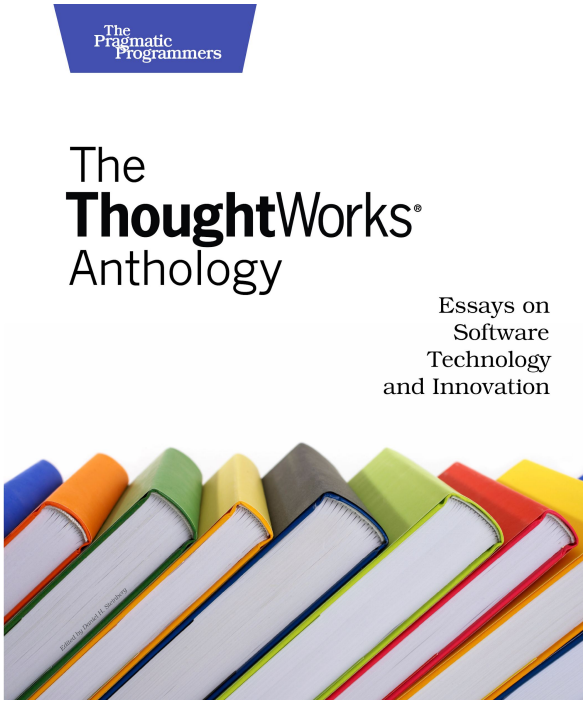
Calisthenics?

= origem na palavra grega 'exercício'



Origem

Jeff Bay em 2009



Por que é importante?

Porque continuamos a escrever código difícil de **entender**, **testar** e **manter**

Conceitos por trás do bom design de código:

- Coesão
- Baixo acoplamento
- Sem redudância
- Encapsulamento
- Testabilidade
- Legibilidade
- Foco

1

Apenas um nível de indentação por método

Problemas:

- Baixa coesão
- Menor legibilidade
- Dificuldade em dar manutenção
- Dificuldade em testar

```php
class Customer {
    public function getPromoCode(string $promoName) {
        // 1
        if ($this->promoCode) {
            // 2
            if (false === $this->promoCodeExpired()) {
                // 3
                if ($this->promoName == $promoName) {
                    return $this->promoCode;
                } else {
                    throw new Exception('Promoção não existe mais');
                }
            } else {
                throw new Exception('Promoção Expirada');
            }
        } else {
            throw new Exception('Cliente sem código de promoção');
        }
    }
}
```

```php
class Customer {
    public function getPromoCode(string $promoName){
        if ($this->promoCode) {
            return $this->getValidPromoCode($promoName);
        } else {
            throw new Exception('Cliente sem código de promoção');
        }
    }

    protected function getValidPromoCode(string $promoName) {
        if (false === $this->promoCodeExpired()) {
            return $this->getPromoExists($promoName);
        } else {
            throw new Exception('Promoção Expirada');
        }
    }

    protected function getPromoExists(string $promoName) {
        if ($this->promoName == $promoName) {
            return $this->promoCode;
        } else {
            throw new Exception('Promoção não existe mais');
        }
    }
}
```

2

Não use Else

Problemas:

- Redundância
- Dificuldade em dar manutenção

```php
protected function indexAction()
{
    if ($this->security->isGranted('ADMIN')) {
        $view = 'admin/index.html.twig';
    } else {
        $view = 'home/access_denied.html.twig';
    }
    return $this->render($view);
}
```

```php
protected function indexAction()
{
    // early return + defensive approach (pessimista)
    if ($this->security->isGranted('ADMIN')) {
        return $this->render('admin/index.html.twig');
    }
    return $this->render('home/access_denied.html.twig');
}
```

```php
protected function indexAction()
{
    // variável com valor padrão
    $view = 'admin/index.html.twig';

    // defensive approach (otimista)
    if (false == $this->security->isGranted('ADMIN')) {
        $view = 'home/access_denied.html.twig';
    }

    return $this->render($view);
}
```

```php
foreach ($orders as $order) {
    // early return + defensive approach (pessimista)
    if ($order->paid()) {
        $report[] = [$order->status => 'Paid'];
        continue;
    }
    $report[] = [$order->status => 'Not Paid'];
}
```

Outras técnicas:

- Polimorfismo
- Null Object Pattern
- Strategy Pattern
- State Pattern

3

Encapsule os tipos primitivos

Problemas:

- Menor legibilidade
- Difícil de testar

```php
class Pedido
{
    public function notificarComprador(String $email)
    {
        if (false == filter_var($email, FILTER_VALIDATE_EMAIL)) {
            throw new InvalidEmailException;
        }

        return $this->repository->sendEmail($email);
    }
}
```

```php
class Email
{
    public $email;
    public function __construct(String $email)
    {
        if (filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
            throw new InvalidEmailException;
        }
        return $this->email = $email;
    }
}

class Pedido
{
    public function notificarComprador(Email $email)
    {
        return $this->repository->sendEmail(new Email($email));
    }
}
```

4

Envolva suas coleções em classes

Problemas:

- Baixa coesão
- Não cumprir o Princípio da Responsabilidade Única

Crie uma classe dedicada para a sua coleção (collection).

5

Use apenas um ponto por linha

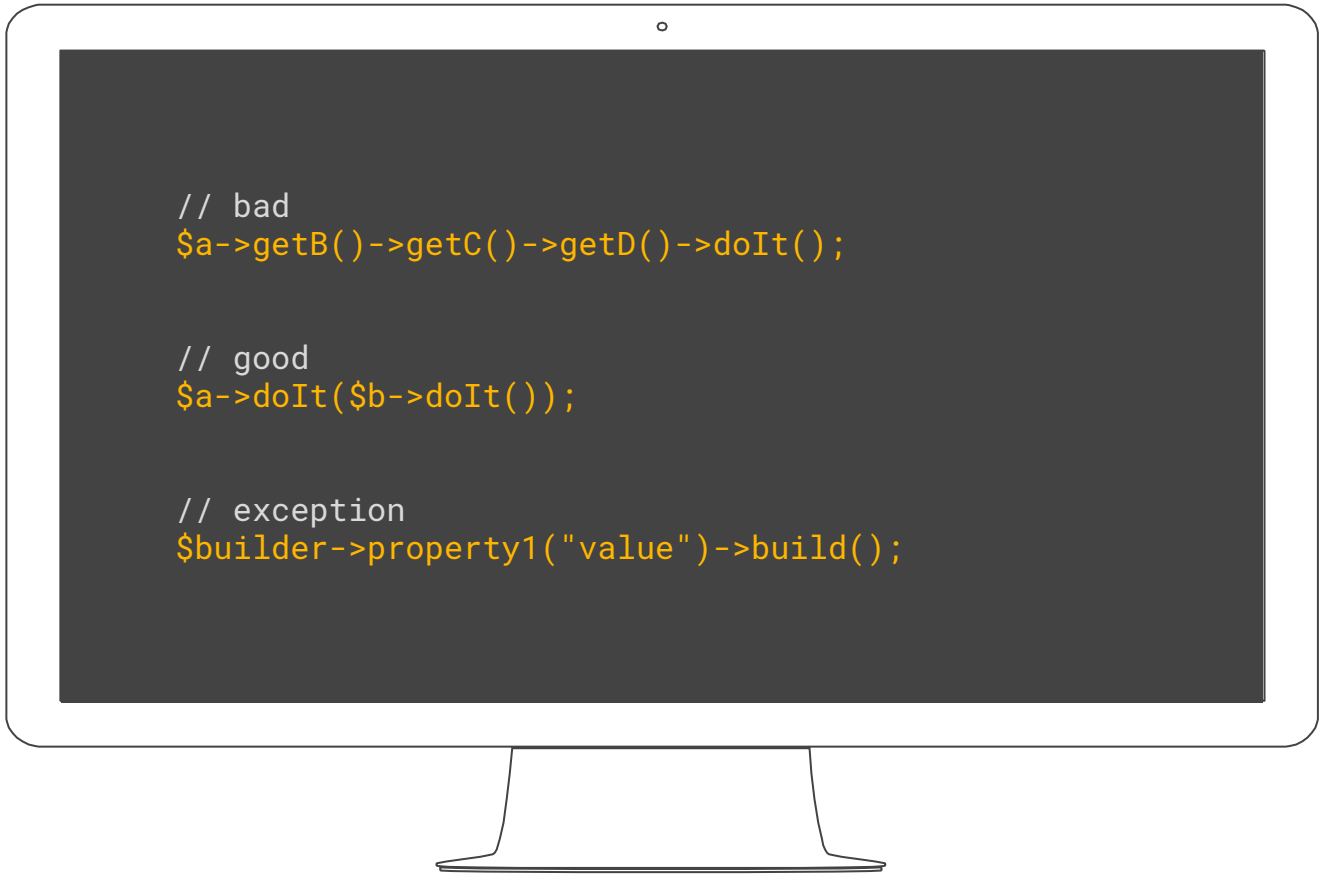
Problemas:

- Seu objeto sabe demais
- Está violando o encapsulamento

Considere mover o comportamento para um dos outros objetos.

Somente fale com seus amigos próximos, não fale com estranhos.

Lei de Deméter


// bad
$a->getB()->getC()->getD()->doIt();
// good
$a->doIt($b->doIt());
// exception
$builder->property1("value")->build();

6

Não abrevie

Dicas:

- Nome de classe, método ou variável com no máximo duas palavras
- Evite repetir palavras:

```
// bad
$perfil->criarPerfil();


// good
$perfil->criar();
```

7

Mantenha todas as entidades pequenas

50 linhas
por classe

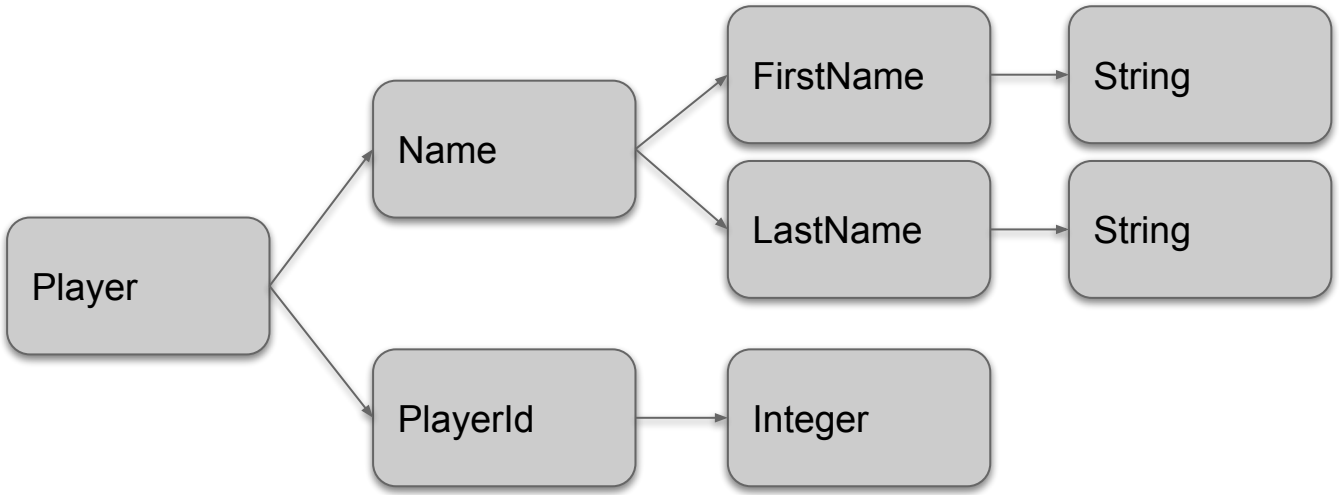
10 arquivos
por pacote

8

Não tenha mais que duas variáveis de instância por classe

Problema:

- Baixa coesão


Player
Name
PlayerId
FirstName
LastName
String
String
Integer

9

Não use *Getters* ou *Setters*

Problemas:

- Redundância
- Menor legibilidade

Enriqueça seu objeto com lógica e métodos mais valiosos e significativos.

```
class Game
{
    private $score;

    public function setScore($score) {
        $this->score = $score;
    }

    public function getScore() {
        return $score;
    }

}

// Using
$game->setScore($game->getScore() + ENEMY_DESTROYED_SCORE);
```

```
class Game
{
    public function addScore($delta) {
        $this->score += $delta;
    }
}

// Using
$game->addScore(ENEMY_DESTROYED_SCORE);
```

Quer treinar estes conceitos?

Obrigada!

Perguntas?

jessicazanelato.dev