Next-Gen PHP

Aula 05

Boas práticas para uma excelente manutenção

Boas práticas?

O que são?

As dores

1. Falta de padrão de código.
2. Portabilidade do seu projeto.
3. Adaptação dificultada.
4. Dificuldade na adição de novas regras.
5. Necessidade de reescrita

Single Responsibility

ActiveRecord é um ANTI-PATTERN!
Um usuário no sistema não se “salva”, o que é “all”? o que significa “find”

```php
class User extends Model
{
    // Crítica: uma classe deformada
    public function save(): bool
    {

    }
}

$user->save();

User::all();

User::find();
```

A responsabilidade de lidar com persistência tem que virar componentes que atuam nos Modelos

```php
// Implementação em Doctrine
$userRepository = $entityManager->getRepository(UserEntity::class);

$users = $userRepository->findAll();

$user = $userRepository->findById(355);

$user->setName('Leonardo');
$user->setEmail('leonardo@differdev.com');

$entityManager->save($user);

$entityManager->flush();
```

Open/Closed

Aberto para extensão e fechado para modificação, significa que nosso componente pode sobre mudança sem que precisamos alterar seu código interno, veja o exemplo:

```php
class Customer
{
    public function canBuy(): bool
    {
        if ($this->customer->credits >= $this->order->value) {
            return true;
        }
        return false;
    }
}
```

```php
class Customer
{
    public function canBuy(): bool
    {
        if ($this->customer->credits >= $this->order->value) {
            return true;
        }
        if ($this->product->hasStock()) {
            return true;
        }
        return false;
    }
}
```

Open/Closed na prática

```php
$customer = new Customer();
$customer->addRule(new HasCredit());
$customer->addRule(new ProductHasStock());
```

Liskov Substitution

É o princípio de substituição, onde diz que é possível trocar um objeto pelo outro sem que seu código principal deixe de funcionar, pense um componente Presenter que queremos ter a possibilidade de imprimir um objeto tanto em JSON quando em XML

Aqui não conseguimos substituir as chamadas, pois são métodos diferentes
violando a SRP(responsabilidade única)

```php
return $presenter->toJson($response);

return $presenter->toXML($response);
```

Liskov na prática

```php
// Configurado em Container
$presenter = new XmlPresenter();

// dentro do controller
return $this->presenter->getPresentation($response);
```

```php
// Configurado em Container
$presenter = new JsonPresenter();

// dentro do controller
return $this->presenter->getPresentation($response);
```

Interface Segregation

É um princípio que pensa na responsabilidade das interfaces, uma interface deve propor assinaturas somente para um tipo de ação, muitos métodos forçam quem implementa a criar métodos vazios, por isso a melhor opção é segregar.

Veja que a Interface está com muitos métodos, que podem não ser necessários

```php
interface ApiHandler
{
    public function fetchData(): Collection;
    public function fetchStringData(): string;
    public function sendData(BodyData $body): void;
    public function deleteData(): string;
}
```

ISP na prática

A CepApi só implementa ações de busca

```php
class CepApi implements ApiFetchHandler
{
    public function fetchData(): Collection { /*...*/ }
    public function fetchStringData(): string { /*...*/ }
}
```

```php
interface ApiSendDataHandler
{
    public function sendData(BodyData $body): void;
    public function deleteData(): string;
}
```

```php
interface ApiFetchHandler
{
    public function fetchData(): Collection;
    public function fetchStringData(): string;
}
```

Dependency Inversion

Inverter uma dependência é não depender diretamente de uma implementação, ou seja você recebe como dependência classes da família da interface, garantindo a presença dos métodos necessários.

Dependemos diretamente da PagarMe

```php
class Payment
{
    public function process(PagarMe $pagarme)
    {
        $pagarme->setUser($this->user);
        $pagarme->setOrder($this->order);

        $pagarme->pay();
    }
}
```

Dependemos indiretamente de um Gateway de pagamento

```php
class Payment
{
    public function process(PaymentInterface $payment)
    {
        $payment->setUser($this->user);
        $payment->setOrder($this->order);

        $payment->pay();
    }
}
```

PSRs

São recomendações de boas práticas que visam padronizar formato de escrita, portabilidade de componentes importantes e nomes de métodos.

PSR-1	Basic Coding Standard
PSR-4	Autoload
PSR-3	Logger Interface
PSR-7	HTTP Message Interface
PSR-11	Container Interface
PSR-12	Extended Coding Style Guide
PSR-14	Event Dispatcher
PSR-15	HTTP Handlers
PSR-16	Simple Cache
PSR-17	HTTP Factories
PSR-20	Clock

PSR-7

Um “standard” super recomendado, com ele conseguimos portar controllers, comunicação com APIs e inclusive testar mensagens Http, seguindo essa comunicação também podemos portabilizar nosso controllers.

```php
class MyController
{
    public function create(ServerRequestInterface $request, ResponseInterface $response): ResponseInterface
    {
        $securedUserDTO = $this->input->handleSecureInput($request->getParsedBody());
        $outputDTO = $this->userService->create($securedUserDTO);

        /**
         * ResponseInterface
         */
        return $this->output->toJson($response, $outputDTO);
    }
}
```

PSR-11

Padronização de container de injeção de dependência

```php
class ComponentFactory
{
    public function create(ContainerInterface $container)
    {
        return new Component(
            new Dependency,
            $container->get(DependencyComplex::class)
        );
    }
}

// Bindings
$dependency = [
    ComponentFactory::class => Component::class
];
```

PSR-15

HttpHandlers para transformar controllers como elementos para tratar requisições e middlewares HTTP de forma padronizada, veja abaixo:

```php
class EntryHandler implements RequestHandlerInterface
{
    public function handler(ServerRequestInterface $request) : ResponseInterface
    {
        // ...
        $requestBody = $request->getParsedBody();

        // new JsonResponse
        return $this->output->toJson();
    }
}
```

```php
class SomeMiddleware implements MiddlewareInterface
{
    public function process(ServerRequestInterface $request, RequestHandlerInterface $handler) : ResponseInterface
    {
        // Antes
        $response = $handler->handle($request);
        //Depois
        return $response;
    }
}
```

Object Calisthenics

Only One Level Of Indentation Per Method

```php
class MyClass
{
    public function myMethodName()
    {
        if ($prop->a === Enum::VALUE) {
            // level .1
            foreach ($this->elements as $element) {
                // level .2
                if ($element->getValue() > 100) {
                    // level .3
                    return $element->result;
                }
            }
        }
    }
}
```

Aplicado

```php
class MyClass
{
    protected object $elements;
    public function myMethodName()
    {
        if ($this->elements->status === Enum::VALID) {
            // level .1
            return $this->getTheElementResult();
        }
        return false;
    }

    public function getTheElementResult(): object
    {
        foreach ($this->elements as $element) {
            // level .1
            return $this->getResult($element);
        }
    }

    public function getResult(object $element)
    {
        if ($element->getValue() > 100) {
            // level .1
            return $element->result;
        }
    }
}
```

Object Calisthenics

Don’t Use The ELSE Keyword

```php
class MyClass
{
    protected object $elements;

    public function myMethodName()
    {
        if ($this->elements->hasElement()) {
            if ($this->elements->hasName()) {
                return 'element has name';
            } elseif ($this->elements->hasAddress()) {
                if ($this->elements->hasZip()) {
                    return 'element has address';
                }
            }
        } else {
            return 'no elements';
        }
    }
}
```

Vamos resolver juntos!

Object Calisthenics

Wrap All Primitives And Strings

```php
class SimpleDTO1
{
    protected string $document;
    protected string $Uuid;
    protected string $url;
}

class SimpleDTO2
{
    protected Document $document;
    protected Uuid $Uuid;
    protected Url $url;
}
```

Aplicado

First Class Collections

```php
class Customer
{
    public string $name;

    /**
     * @var []Address $addresses
     */
    protected array $addresses;

    public function addAddress(Address $address)
    {
        // ...
    }

    // Violação do SRP
    public function orderAddresses()
    {
        sort($this->addresses);
    }
}
```

Aplicado

```php
class Customer
{
    public string $name;

    /**
     * @var Address $addresses
     */
    protected Addresses $addresses;

    public function addAddress(Address $address)
    {
        $addresses->append($address);
    }
}

$addresses = $customer->getAddresses();
$addresses->asort();
```

Object Calisthenics

One Dot Per Line

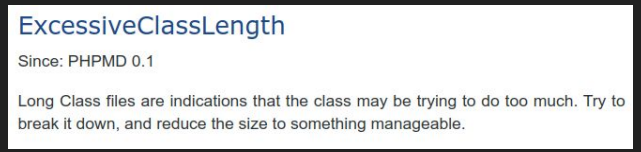
```
$comments = $customer->find(23)->posts(2)->comments(3)->get();
```

Don’t Abbreviate

```
$canBeDeleted = $customer->cbd();
```

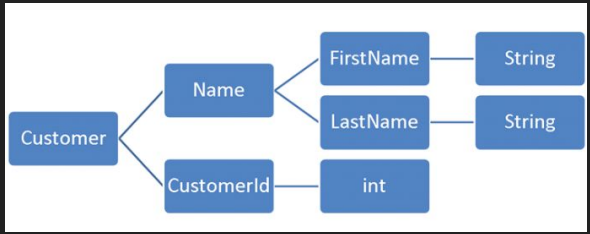
Keep All Entities Small

Classes até no máximo 150 linhas, pacotes até no máximo 10 arquivos.
Mess Detector pode ajudar nisso, o padrão pra classe é 1000 linhas



Object Calisthenics

No Classes With More Than Two Instance Variables



No Getters/Setters/Properties

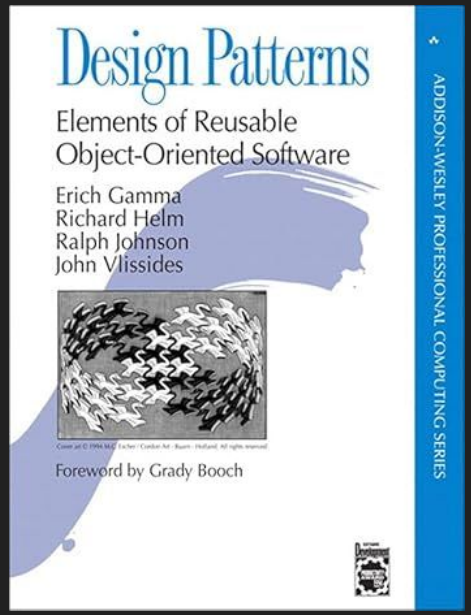
```
$user->setUnderAge($user->getAge() < Country::CONSENT);
```

Aplicado

```
$user->setUnderAge(Country::CONSENT);
$user->isUnderAge();
```

Padrões de Projetos Gang of Four

São padrões de projetos reconhecidos por quatro desenvolvedores nos anos 90 que decidiram se reunir e catalogar os padrões encontrados em inúmeros projeto que trabalharam, o livro é uma boa referência de evolução ao OOP.



Padrões de Projetos - Grupos

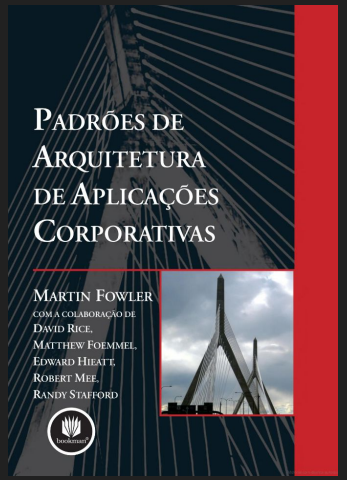
Criacionais
Abstract factory
Builder
Factory method
Prototype
Singleton

Estruturais
Adapter
Bridge
Composite
Decorator
Facade
Flyweight
Proxy

Comportamentais
Chain of responsibility
Command
Interpreter
Iterator
Mediator
Memento
Observer
State
Strategy
Template method
Visitor

Padrões de Arquitetura de Aplicações Corporativas
Patterns of Enterprise Application Architecture

Esse livro é uma continuação do livro GoF(Gang of Four), visando o mapeamento dos padrões de arquitetura de softwares encontrados por Martin Fowler e outros autores, o livra é bem antigo e muito do que se encontra por lá já não é mais praticado, ou foi superado por outras técnicas, mas ainda assim é uma boa referência!



Desafio

Presente no repositório do nosso curso:

https://github.com/DifferDev/NextGenPHP

Boa noite!

Obrigado pela presença!