Next-Gen PHP

Aula 04

Pensamento analítico com visualização e desenho

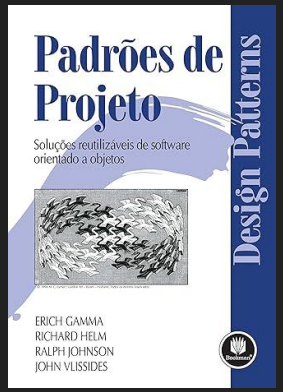
O que é design?

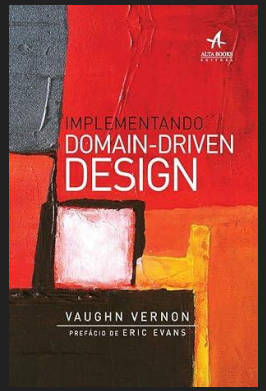
E o que é design de código?

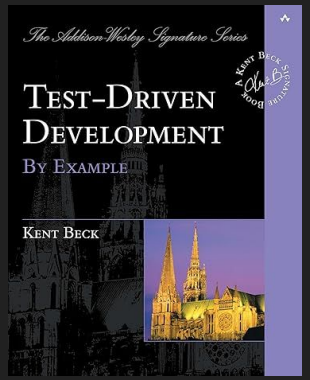
As dores

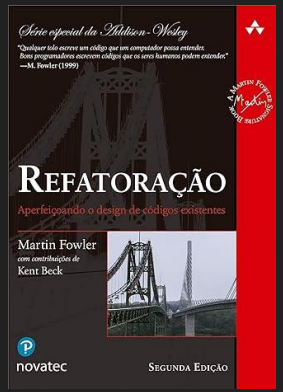
1. Difícil de dar manutenção.
2. Código mal organizado.
3. Não entender como funciona.
4. Aumento da complexidade.
5. Dificuldade em alterar!
6. Escrever mais do que o necessário.

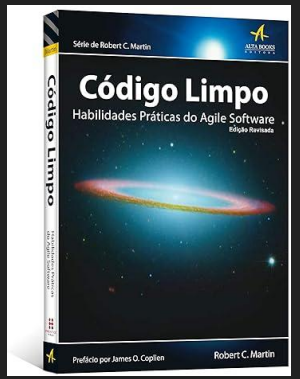
Referências

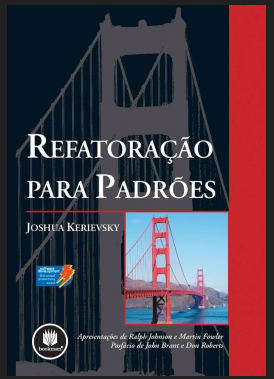




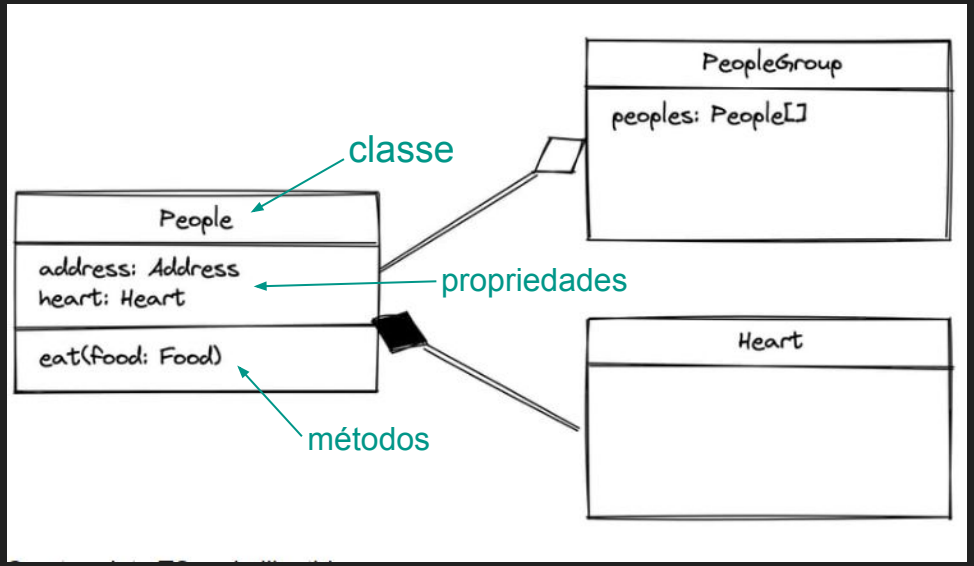








Desenhar ajuda a enxergar as soluções



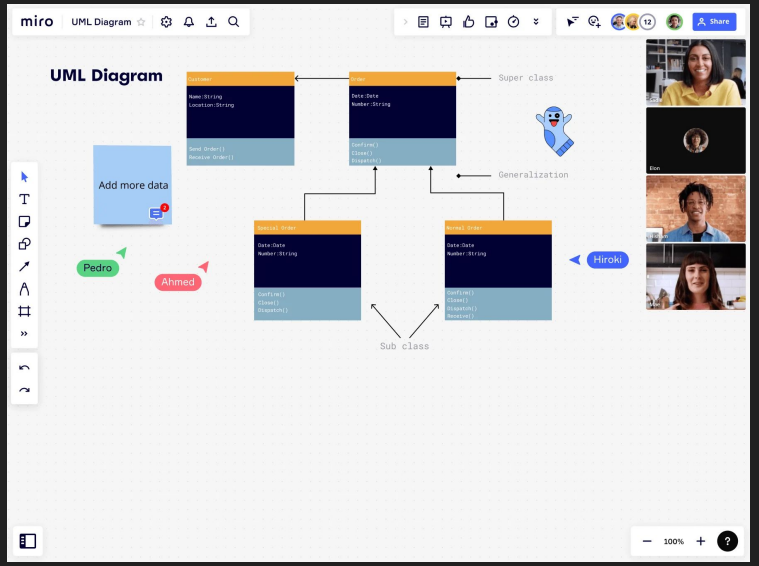
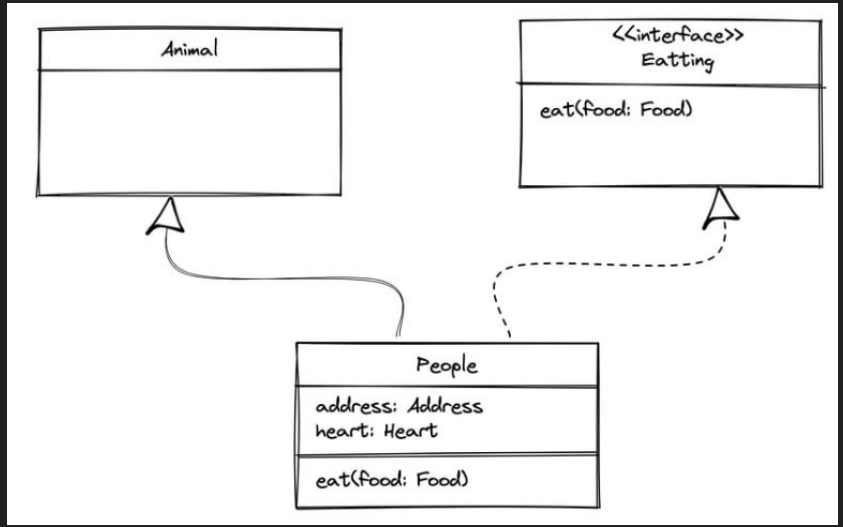


Diagrama de classes da UML

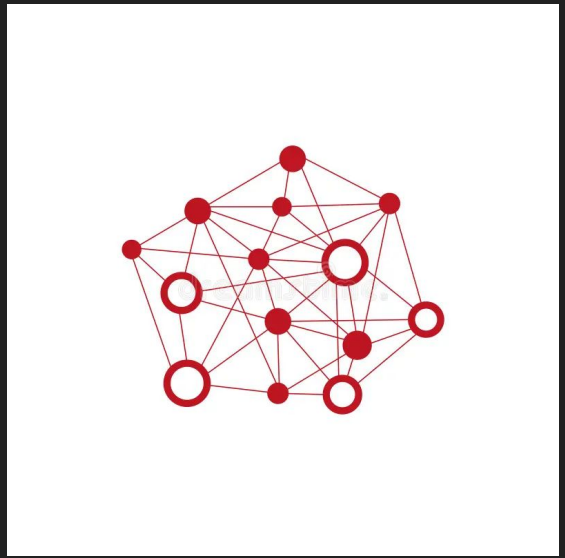
Identificar quais Domínios estarão presentes no sistema, tudo que é palpável, inclusive operações de negócio, por exemplo: Venda é uma operação super importante num marketplace, Reserva também é uma operação importante em hotelaria.



https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-diagrama-classes-relacionamentos_v01.pdf

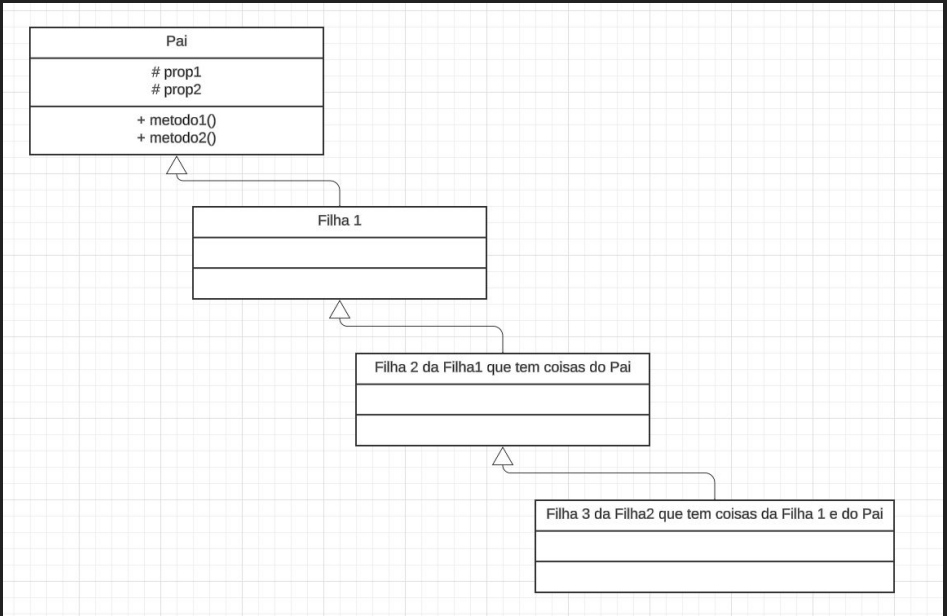
Objetos e a portabilidade

Trate os elementos do sistema como peças, considerando instâncias de classes como elementos físicos, assim quanto maior o trabalho em memória melhor a portabilidade do sistema e também mais fácil testá-los.



Simplicidade

Evite herança em favor de composição indireta(interfaces), caso necessário, não ultrapasse mais do segundo nível.



Objeto concreto vs Método estático

Qual é melhor?

```php
<?php

$obj = new HTTPRequest('POST', 'http://teste.com.br');
$obj->send();

// vs

HTTPRequest::send('POST', 'http://teste.com.br');
```

Transporte

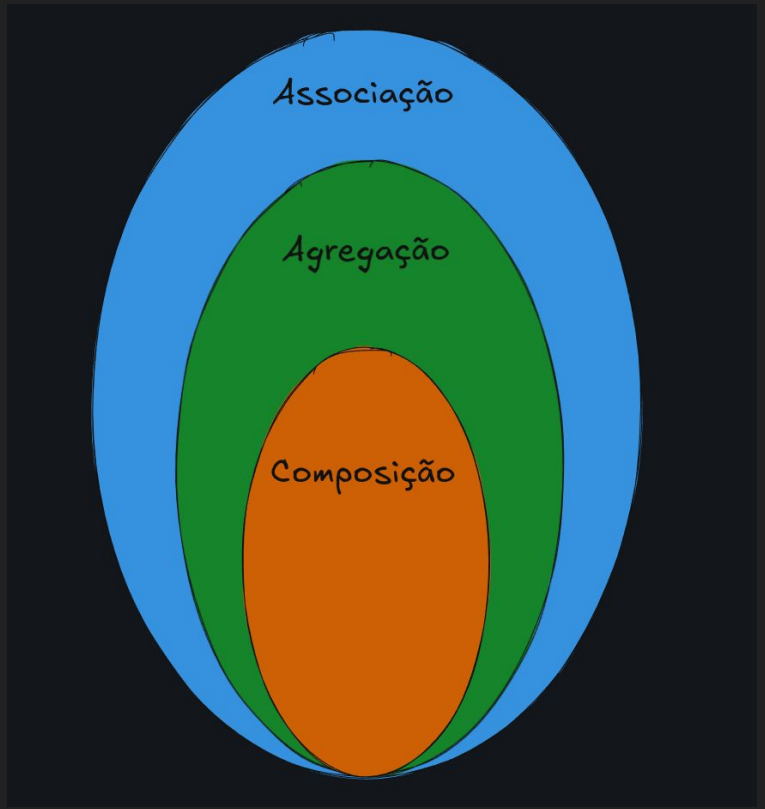
```php
class ServiceTest
{
    public function create(HttpRequest $request): ResponseInterface
    {
        $request->send();
    }
}



$obj = new HttpRequest('POST', 'http://teste.com.br');
// $obj->send();

$service = new ServiceTest();
$service->create($obj);
```

Associações



Hands-on

Agregação e Composição

Exemplo: Classe Logger

Baixo acoplamento, o problema.

```php
class UseCase
{
    // alto acoplamento
    public function makeABid(int $id, string $name, float $value)
    {
        try {
            $bidInput = new BidInput(
                id: $id,
                name: $name,
                value: $value,
                created: Date::now()
            );

            Bid::create($bidInput);

            Logger::success('Created');
        } catch (\Throwable $th) {
            Logger::error('Error');
            throw $th;
        }
    }
}
```

```php
class UseCaseTest
{
    public function testClassUseCaseShouldMakeABid()
    {
        // Como testar???
        // recursos externos (DB, Date, Logger)
        $result = $this->useCase->makeABid(1, 'A name', 2344.33);
        $this->assertEquals((object)[
            'id'      => $id,
            'name'    => $name,
            'value'   => $value,
            'created' => Date::now()
        ], $result);
    }
}
```

Baixo acoplamento, uma solução

```php
class UseCase
{
    // baixo acoplamento
    public function makeABid(BidInput $bidInput)
    {
        try {
            $this->bidRepository->save($bidInput);

            $this->logger->log(LogType::SUCCESS, 'Created');
        } catch (\Throwable $th) {
            $this->logger->log(LogType::ERROR, 'Error');
            throw $th;
        }
    }
}
```

```php
class UseCaseTest
{
    public function testClassUseCaseShouldMakeABid()
    {
        $bidInput = new BidInput(
            id: $id,
            name: $name,
            value: $value,
            created: new DateTimeImmutable('2022-02-02 22:22:22')
        );

        // Mocks de dependencias externas
        $this->useCase->setLogger($this->loggerMock);
        $this->useCase->setLogger($this->bidRepositoryMock);

        $result = $this->useCase->makeABid($bidInput);

        $this->assertEquals((object)[
            'id'      => $id,
            'name'    => $name,
            'value'   => $value,
            'created' => new DateTimeImmutable('2022-02-02 22:22:22')
        ], $result);
    }
}
```

Alta coesão

Em uma simples definição, classes coesas vão facilitar a capacidade de adaptação do código, vamos entender alguns pontos.

```php
// Pré config
$myComponent = new Component(new Dependency1(), new Dependency2(), $closure);
```

```php
// Pós config
$myComponent = new Component();

$myComponent->setDependency1(new Dependency1());
$myComponent->setDependency2(new Dependency2());
```

Interfaces

Interfaces são contratos, ou seja métodos que precisam ser implementados, pode-se fazer também o mesmo com classes abstratas que possuem métodos abstratos, mas incentivamos o uso de interfaces para que haja composição ao invés de herança.

```php
interface SomeActionsInterface
{
    public function formatData(MyCollection $myCollection): void;
    public function doYourJumps(string $howHigh): bool;
}
```

S.O.L.I.D

Single Responsibility (SRP)

Open/Closed (OCP)

Liskov Substitution (LISP)

Interface Segregation (ISP)

Dependency Inversion (DIP)

Desafio

Presente no repositório do nosso curso:

https://github.com/DifferDev/NextGenPHP

Boa noite!

Obrigado pela presença!