State (Implementação alternativa)

Definição do Open Closed Principle: *"Entidades de software (classes, módulos, funções, etc.) devem ser abertas para extensão, mas fechadas para modificação"*. Ou seja, tal entidade pode permitir que seu comportamento seja estendido sem modificar seu código-fonte.

Na aula é citado que o padrão *State* segue o **Open Closed Principle do SOLID**, de fato ele segue, porém, no exemplo da aula a abordagem seguida não permite a adição de novos estados sem que seja necessário alterar a classe **Pedido**, uma vez que em seu construtor temos as definições dos estados disponíveis. No corpo da classe **Pedido** também temos os métodos **getters** dos estados disponíveis.

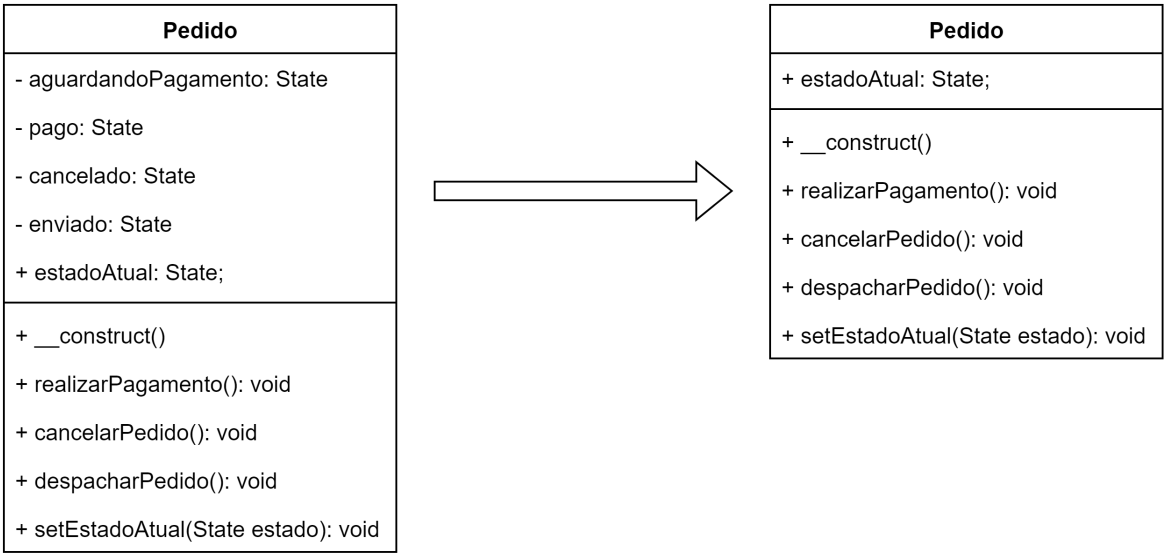
```
class Pedido
{
    private State $aguardandoPagamento;
    private State $pago;
    private State $cancelado;
    private State $enviado;

    private State $estadoAtual;

    public function __construct()
    {
        $this->aguardandoPagamento = new AguardandoPagamentoState($this);
        $this->pago = new PagoState($this);
        $this->cancelado = new CanceladoState($this);
        $this->enviado = new EnviadoState($this);
        $this->estadoAtual = $this->aguardandoPagamento;
    }

    //... continuação da classe ...

    public function getAguardandoPagamento(): State
    {
        return $this->aguardandoPagamento;
    }

    public function getPago(): State
    {
        return $this->pago;
    }

    public function getCancelado(): State
    {
        return $this->cancelado;
    }

    public function getEnviado(): State
    {
        return $this->enviado;
    }

    //... continuação da classe ...
}
```

Caso quiséssemos adicionar um novo estado, como por exemplo **Finalizado**, além de criar a nova classe de estado teríamos também que adicionar uma nova dependencia no construtor de **pedido** e o seu respectivo método **getter**.

```
class Pedido
{
    private State $aguardandoPagamento;
    private State $pago;
    private State $cancelado;
    private State $enviado;
    private State $finalizado;

    private State $estadoAtual;

    public function __construct()
    {
        $this->aguardandoPagamento = new AguardandoPagamentoState($this);
        $this->pago = new PagoState($this);
        $this->cancelado = new CanceladoState($this);
        $this->enviado = new EnviadoState($this);
        $this->finalizado = new FinalizadoState($this);
        $this->estadoAtual = $this->aguardandoPagamento;
    }

    //... continuação da classe ...

    public function getAguardandoPagamento(): State
    {
        return $this->aguardandoPagamento;
    }

    public function getPago(): State
    {
        return $this->pago;
    }

    public function getCancelado(): State
    {
        return $this->cancelado;
    }

    public function getEnviado(): State
    {
        return $this->enviado;
    }

    public function getFinalizado(): State
    {
        return $this->finalizado;
    }

    //... continuação da classe ...
}
```

Para evitar que isso aconteça e fazer com que nossa implementação realmente siga o Open Closed Principle "ao pé da letra", precisamos fazer alguns ajustes simples:

1. **Remover as instâncias dos estados do construtor da classe Pedido, mantendo apenas a referência ao estado atual.**




```php
class Pedido
{
    //Os atributos de referência aos estados disponíveis foram removidos

    private State $estadoAtual;

    public function __construct()
    {
        //Os atributos não precisam mais serem inicializados
        $this->estadoAtual = $this->aguardandoPagamento;
    }

    //... continuação da classe ...

    //Os getters não são mais necessários

    //... continuação da classe ...
}
```

2. **Nas classes de estado, trocar as referência dos estados antes contidos na classe Pedido para nova instâncias de objetos de estado:**

```php
class AguardandoPagamentoState implements State
{
    private Pedido $pedido;

    public function __construct(Pedido $pedido)
    {
        $this->pedido = $pedido;
    }

    public function sucessoAoPagar(): void
    {
        $this->pedido->setEstadoAtual($this->pedido->getPago()); //Antes
        $this->pedido->setEstadoAtual(new PagoState($this->pedido)); //Depois
    }

    public function cancelarPedido(): void
    {
        throw new \Exception('Operação não suportada, o pedido ainda não foi pago.');
    }

    public function despacharPedido(): void
    {
        throw new \Exception('Operação não suportada, o pedido ainda não foi pago.');
    }
}
```

```php
class PagoState implements State
{
    private Pedido $pedido;

    public function __construct(Pedido $pedido)
    {
        $this->pedido = $pedido;
    }

    public function sucessoAoPagar(): void
    {
        throw new \Exception('Operação não suportada, o pedido já foi pago.');
    }

    public function cancelarPedido(): void
    {
        $this->pedido->setEstadoAtual($this->pedido->getCancelado()); //Antes
        $this->pedido->setEstadoAtual(new CanceladoState($this->pedido)); //Depois
    }

    public function despacharPedido(): void
    {
        $this->pedido->setEstadoAtual($this->pedido->getEnviado()); //Antes
        $this->pedido->setEstadoAtual(new EnviadoState($this->pedido)); //Depois
    }
}
```

Com esses simples ajustes que foram feitos, agora é possível criar novos estados sem que seja necessário alterar a classe Pedido.

ANTES

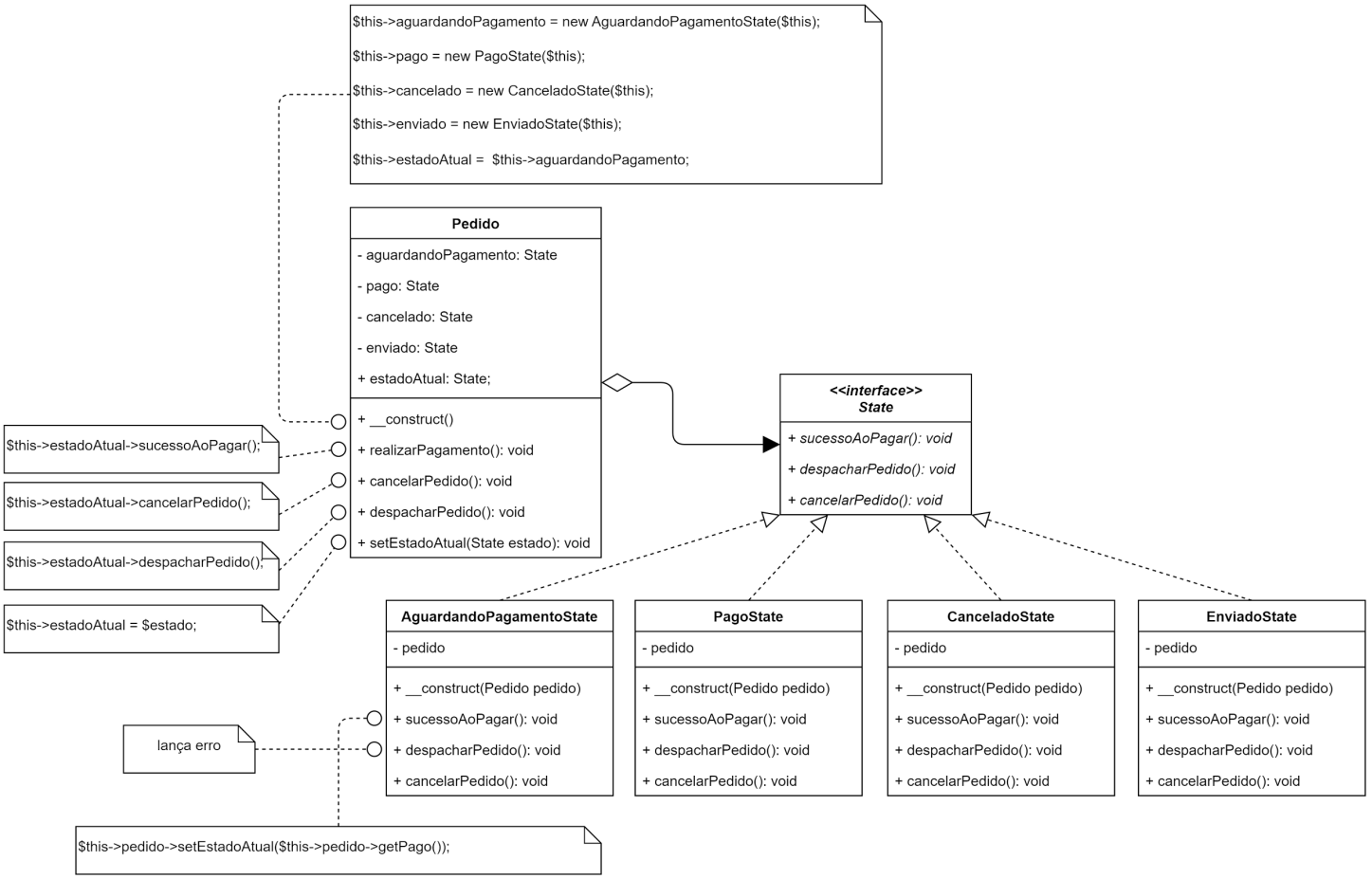


Diagrama de classes do exemplo (Getters foram ocultados)

DEPOIS

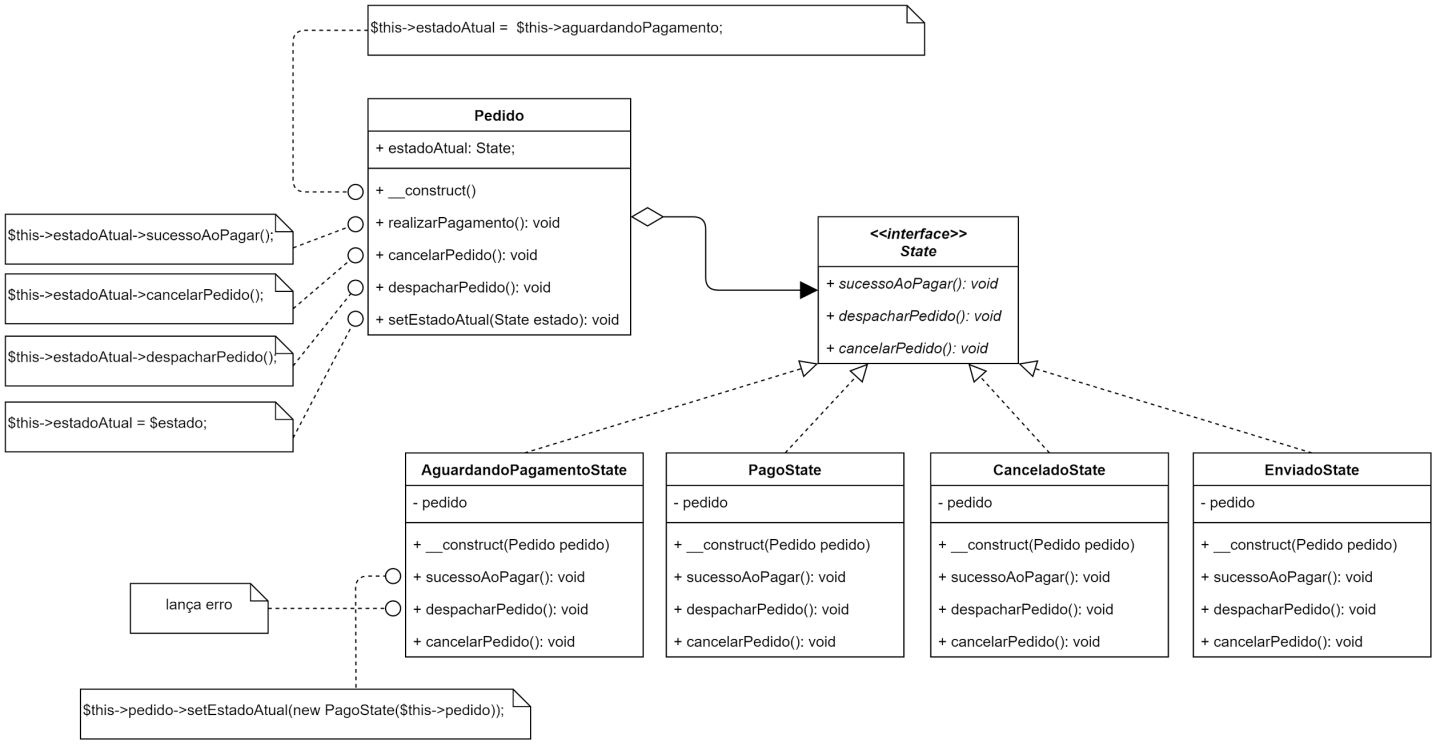


Diagrama de classes do exemplo (Getters foram ocultados)

Prós e contras de cada abordagem

Abordagem 1 (Com estados no construtor da classe Pedido):

- Ao olhar o código da classe Pedido é possível identificar de forma explícita todos os estados que ela pode assumir.
- A dependências ficam centralizada na classe `Pedido` e as classes de estado não precisam conhecer quais são os outros estados.
- Ao adicionar um novo estado no sistema, a classe `Pedido` terá que ser alterada para suportar o novo estado (Criação de novo atributo e seu respectivo método getter).
- Mesmo tendo que alterar a classe `Pedido`, a lógica dos novos estados ficarão fora dela, deste modo a classe `Pedido` não crescerá de forma desordenada em complexidade. Essa abordagem **não** respeita o Open Closed Principle em sua totalidade.

Abordagem 2 (Sem estados no construtor da classe Pedido):

- Ao olhar o código da classe Pedido **não** é possível identificar de forma explícita todos os estados que ela pode assumir.
- As dependências ficam dispersas entre as classes de estado. Cada estado precisa conhecer as outras classes de estado para quais eles devem transitar.
- Ao adicionar um novo estado no sistema, a classe `Pedido` não precisará ser alterada. Segue totalmente o Open Closed Principle.

Com isso você tem informações suficientes para tomar suas decisões ao projetar o seu código.