

Design Patterns e MVC

o que é cada um deles e como aplicá-los em
um projeto de software

O que são Design Patterns?

- **Padrões de design** são soluções típicas para problemas comuns em design de software. Eles são como projetos pré-fabricados que você pode personalizar para resolver um problema de design recorrente em seu código.
- O padrão não é um código específico, mas um conceito geral para resolver um problema em particular. Você pode seguir os detalhes do padrão e implementar uma solução que se adapte às realidades do seu próprio programa.

O que são Design Patterns?

- Uma analogia com um algoritmo é uma receita de culinária: ambos têm passos claros para alcançar um objetivo. Por outro lado, **um padrão é mais como um modelo**: você pode ver qual é o resultado e suas características, mas a ordem exata de implementação depende de você.

Em que consiste um pattern?

A maioria dos padrões é descrita muito formalmente para que as pessoas possam reproduzi-los em muitos contextos. Aqui estão as seções que geralmente estão presentes em uma descrição de padrão:

- **A intenção** do padrão descreve brevemente tanto o problema quanto a solução.
- **A motivação** explica ainda mais o problema e a solução que o padrão possibilita.
- **A estrutura** das classes mostra cada parte do padrão e como elas estão relacionadas.
- **O exemplo de código** em uma das linguagens de programação populares torna mais fácil entender a ideia por trás do padrão.

Classificações de design patterns

Os padrões de design diferem por sua complexidade, nível de detalhe e escala de aplicabilidade para todo o sistema que está sendo projetado.

- Os padrões mais básicos e de baixo nível são frequentemente chamados de *idioms* (expressões idiomáticas). Eles geralmente se aplicam apenas a uma única linguagem de programação.
- Os padrões mais universais e de alto nível são os *architectural patterns* (padrões arquitetônicos). Os desenvolvedores podem implementar esses padrões em praticamente qualquer idioma. Ao contrário de outros padrões, eles podem ser usados para projetar a arquitetura de uma aplicação inteira.

Classificações de design patterns

Os padrões de design podem ser divididos em três tipos:

- **Creational patterns** - Os padrões criacionais fornecem mecanismos de criação de objetos que aumentam a flexibilidade e a reutilização do código existente.
- **Structural patterns** - Os padrões estruturais explicam como montar objetos e classes em estruturas maiores, mantendo essas estruturas flexíveis e eficientes.
- **Behavioral patterns** - Padrões comportamentais cuidam da comunicação eficaz e da atribuição de responsabilidades entre os objetos.

Exemplo prático de um design pattern

- Neste exemplo, utilizaremos o padrão de design comportamental **Observer**, também conhecido como Listener ou Event-Subscriber.
- O **Observer** é um padrão de projeto comportamental que permite que você defina um mecanismo de assinatura para notificar múltiplos objetos sobre quaisquer eventos que aconteçam com o objeto que eles estão observando.
- O código exemplo está em **scripts/observer.php**
- Neste exemplo, é simulado um sistema simples de notificação para uma carteira de ações .

Objetivo do Observer

- Imagine que você tem dois tipos de objetos: um Cliente e uma Loja. O cliente está muito interessado em uma marca particular de um produto
- O cliente pode visitar a loja todos os dias e checar a disponibilidade do produto. Mas enquanto o produto ainda está a caminho, a maioria desses visitas serão em vão.
- O padrão Observer sugere que você adicione um mecanismo de assinatura para a classe Loja (nesse exemplo) para que objetos individuais possam assinar ou desassinar uma corrente de eventos vindo daquela Loja.

Estrutura do Observer

Observer.php

A estrutura básica do padrão é a seguinte:

- **1. Subject (Observado)**
- Mantém uma lista de observers inscritos.
- Oferece métodos para:
 - adicionarObserver()
 - removerObserver()
 - notificarObservers() (aciona a atualização em todos os observers).

```
//subject, no padrao MVC sempre fica no MODEL. já os observers, podem ficar em outras partes
class Acao
{
    private float $valor;
    private $acionistas = []; // lista que armazena todos os 'ouvintes' do subject

    public function __construct(float $valAcaoInicial)
    {
        $this->valor = $valAcaoInicial;
    }

    public function alterarValor(float $val)
    {
        $this->valor = $val;
        $this->notificar(); // notificará sempre o valor da acao mudar
    }

    public function remover(Investidor $inv)
    {
        $k = array_search($inv, $this->acionistas, true);
        if ($k !== false) {
            unset($this->acionistas[$k]);
        }
    }

    public function add(Investidor $inv)
    {
        $this->acionistas[] = $inv;
    }

    private function notificar()
    {
        foreach ($this->acionistas as $ac) {
            $ac->update($this->valor);
        }
    }
}
```

Estrutura do Observer

Observer.php

A estrutura básica do padrão é a seguinte:

- **2. Observer (Observador)**
- Define uma interface (ou contrato) com um método de atualização, como:
 - atualizar() ou onNotify()
- Cada observer concreto implementa sua própria reação às mudanças do Subject.

```
<?php
interface Investidor //padrao que todos os observers deverão seguir
{
    public function update(float $val);
}

class User implements Investidor //observer concreto
{
    private string $nome;

    public function __construct(string $nome)
    {
        $this->nome = $nome;
    }

    public function update(float $val)
    {
        echo "Olá " . $this->nome . ", O valor da sua ação é " . $val . "\n";
    }
}
```

Estrutura do Observer

- **Fluxo Básico**
- O Subject muda de estado.
- Chama notificarObservers().
- Cada observer na lista executa seu método de atualização (update()).

```
$petrobras = new Acao(10000);


$ac1 = new User("Joao");
$ac2 = new User("Maria");

$petrobras->add($ac1);
$petrobras->add($ac2);

$petrobras->alterarValor(90000);

$petrobras->remover($ac1);

//simular variacoes aleatorias
echo "\n";
for ($i = 0; $i < 5; $i++) {
    $val = rand(1000, 10000);
    $petrobras->alterarValor($val);
    sleep(1);
}
```



```
$ php observer.php
Olá Joao, O valor da sua ação é 90000
Olá Maria, O valor da sua ação é 90000

Olá Maria, O valor da sua ação é 1050
Olá Maria, O valor da sua ação é 3123
Olá Maria, O valor da sua ação é 4093
Olá Maria, O valor da sua ação é 8282
Olá Maria, O valor da sua ação é 4474
```

Referências

- <https://refactoring.guru/design-patterns/php>
- <https://refactoring.guru/design-patterns/what-is-pattern>
- <https://refactoring.guru/pt-br/design-patterns/observer>