Academia Indiana

Programação Orientada a Objetos

Matheus Barbosa Ribeiro

Orientação a Objetos

O ato de codificar envolve a criação de soluções por meio de uma ou mais linguagens de programação. A forma como realizamos esse processo, ou seja, como traduzimos uma solução em código, pode variar de acordo com o paradigma utilizado. De fato, existem diversos paradigmas de programação, cada um com suas abordagens específicas para solucionar problemas.

Entre os principais paradigmas de programação, destacam-se:

- **Paradigma Procedural/Estruturado:** Neste paradigma, os programas são estruturados em sequências de instruções, normalmente organizadas em funções ou procedimentos. O foco está na manipulação direta dos dados e nas ações a serem executadas, seguindo uma ordem predefinida.
- **Paradigma Orientado a Objetos:** Baseado no conceito de "objetos", este paradigma enfatiza a organização dos dados e suas interações por meio de classes e instâncias. As classes representam estruturas que encapsulam dados e comportamentos relacionados, permitindo a reutilização e a criação de hierarquias entre objetos.
- **Paradigma Funcional:** Nesse paradigma, a ênfase está na avaliação de funções matemáticas puras. A imutabilidade dos dados é valorizada, reduzindo efeitos colaterais e facilitando a compreensão do comportamento do programa.

Ao optar por um determinado paradigma de programação, os desenvolvedores moldam a abordagem geral de seus códigos e a forma como estruturam suas soluções.
Neste módulo, mergulharemos no paradigma Orientado a Objetos, que teve seu surgimento no final da década de 60 e alcançou grande popularidade a partir da década de 90. Apesar de ter sido concebido há algum tempo, este paradigma continua sendo amplamente utilizado nos dias de hoje, com relevância e aplicabilidade inabaláveis.

Como a POO funciona

A POO se fundamenta na criação e manipulação de classes e objetos. Nas classes, encontramos uma espécie de esqueleto ou molde que define a estrutura e o comportamento dos objetos que serão instanciados a partir delas. Podemos imaginar as classes como uma planta ou receita que determina como os objetos serão criados e quais características e comportamentos eles possuirão.

Os atributos em uma classe representam as características ou propriedades dos objetos. Eles definem o estado do objeto e representam os valores que o objeto pode armazenar. Por exemplo, em uma classe "Carro", os atributos poderiam ser:

- Cor;
- Modelo;
- Ano;
- etc.

Já os métodos em uma classe representam os comportamentos ou ações que os objetos podem executar. Na classe "Carro" citada acima, os métodos poderiam ser:

- Acelerar;
- Frear;
- Ligar faróis;
- Entre outros.

Construção de classes no PHP

Agora, exploraremos o exemplo simples citado acima para entender como escrever **classes** no PHP. Em seguida, abordaremos os pilares da Programação Orientada a Objetos, o que nos permitirá aprimorar nosso código.

Em PHP, podemos criar uma classe utilizando a palavra-chave **"class"**, seguida pelo **nome da classe** e **um bloco de código entre chaves**. Nesse bloco, definimos os **atributos** e os **métodos** da classe, conforme a estrutura e o comportamento desejados para os objetos que serão criados a partir dela.

Veja o exemplo abaixo:

```php
<?php

class Carro {
    //Atributos
    public string $cor;
    public int $portas;
    public int $velocidade;
    public bool $faroisLigados;
    public bool $transmissaoManual;

    //Métodos
    public function setCor(string $cor): void
    {
        $this->cor = $cor;
    }

    public function getCor(): string
    {
        return $this->cor;
    }

    public function setVelocidade(int $velocidade): void
    {
        $this->velocidade = $velocidade;
    }

    public function getVelocidade(): int
    {
        return $this->velocidade;
    }

    public function setFarois(bool $faroisLigados): void
    {
        $this->faroisLigados = $faroisLigados;
    }
```

```
    public function getFarois(): bool
    {
        return $this->faroisLigados;
    }

    public function setTransmissaoManual(bool $transmissaoManual): void
    {
        $this->transmissaoManual = $transmissaoManual;
    }

    public function getTransmissaoManual(): bool
    {
        return $this->transmissaoManual;
    }
}
```

Neste exemplo, definimos a classe "Carro" com os atributos públicos: **$cor**, **$portas**, **$velocidade**, **$faroisLigados** e **$transmissaoManual**, que armazenam as informações do carro. Em seguida, criamos métodos setters e getters.

Os métodos com o prefixo “set” neste exemplo permitem definir os valores dos atributos, enquanto os métodos com o prefixo “get” possibilitam obter os valores já atribuídos a eles.

Objeto

Como mencionado anteriormente, os objetos são instâncias de uma classe. Uma vez que tenhamos criado uma classe, podemos criar objetos a partir dela.
Veja o exemplo abaixo:

```
$carro = new Carro();
```

Podemos criar uma instância de uma classe utilizando a palavra reservada **"new"** seguida pelo nome da classe junto com os parênteses **"()"**. Ao fazer isso, estamos alocando espaço na memória para o objeto e associando-o à classe em questão.
No exemplo dado, ao escrever **“$carro = new Carro();”**, estamos criando um novo objeto a partir da classe "Carro" e atribuindo-o a uma variável chamada **“$carro”**. Essa variável agora será uma referência ao objeto criado, permitindo-nos acessar e manipular os recursos de forma mais conveniente.

Com o objeto criado e atribuído a uma variável, podemos interagir com seus atributos e métodos usando a notação de seta **"->"**. Por exemplo:

```php
$carro = new Carro();

//Métodos:
$carro->setCor('Vermelho');
$carro->setVelocidade(100);

//Atributos:
echo $carro->cor;
echo $carro->velocidade;
```

Dessa forma, usando objetos e suas instâncias, podemos criar e manipular os dados de forma organizada e concisa.

Essa é apenas uma introdução ao conceito de classes em PHP. Mais adiante, iremos explorar os pilares fundamentais da Programação Orientada a Objetos: Abstração, Encapsulamento, Herança e Polimorfismo. Com uma compreensão sólida desses conceitos, estaremos preparados para aprimorar nosso código e criar soluções mais robustas e flexíveis.

Pilares da POO

Os pilares da POO são os princípios nos quais esse paradigma se baseia, ou seja, sua sustentação.
Vamos avançar no entendimento de cada pilar e ao mesmo tempo aplicar seu conceito em nosso código.

Abstração

Este pilar é o resumo do que estudamos até agora. É o ato de conseguir traduzir características essenciais de um objeto do mundo real ou um conceito de um problema em uma classe focada nos aspectos relevantes para o sistema.

Encapsulamento

O encapsulamento consiste em ocultar a implementação interna dos objetos, permitindo que somente os métodos e atributos essenciais estejam acessíveis externamente. Isso é alcançado por meio da definição de visibilidade (**public**, **private e protected**) para os atributos e métodos. O encapsulamento protege os dados de modificações não autorizadas e facilita a manutenção do código, pois alterações internas podem ser feitas sem impactar o código que utiliza a classe.
Veja o exemplo abaixo:

```php
<?php

class Carro
{
    private string $cor;
    private int $portas;
    private int $velocidade;
    public bool $faroisLigados;
    public bool $transmissaoManual;
```

```php
    public function __construct(int $portas, string $cor)
    {
        $this->portas = $portas;
        $this->cor = $cor;
    }

    public function getCor(): string
    {
        return $this->cor;
    }

    public function setVelocidade(int $velocidade): void
    {
        if($velocidade < 0) {
            echo "Não é possível setar uma velocidade negativa";

            return;
        }

        $this->velocidade = $velocidade;
    }

    public function getVelocidade(): int
    {
        return $this->velocidade;
    }

    public function setFarois(bool $faroisLigados): void
    {
        $this->faroisLigados = $faroisLigados;
    }

    public function setTransmissaoManual(bool $transmissaoManual): void
    {
        $this->transmissaoManual = $transmissaoManual;
    }
}
```

Herança

A herança permite que uma classe herde características de outra classe, estabelecendo uma relação de "é um" entre as classes. Isso significa que a classe derivada (**subclasse**) herda atributos e métodos da classe base (**superclasse**). Com a herança, é possível reutilizar código, promover a hierarquia de classes e criar estruturas mais genéricas e especializadas. Isso leva a um código mais eficiente e organizado.

Polimorfismo

O polimorfismo permite que objetos de diferentes classes possam ser tratados de uma mesma forma quando possuem métodos com a mesma **assinatura**(**nome e parâmetros**), mas com implementações específicas para cada classe. Isso possibilita a substituição de objetos de diferentes tipos, tornando o código mais flexível e adaptável a diferentes situações.