UniSENAI

CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ARTIGO SOBRE FUNÇÕES E ARRAYS EM PHP

Danillo de Souza Koch

Programação orientada a objetos

JOINVILLE
2025

1. DEFINIÇÃO DO PARADIGMA DE ORIENTAÇÃO A OBJETOS

O paradigma de Orientação a Objetos (POO) é uma metodologia de programação que visa modelar sistema como um conjunto de objetos que interagem entre si.

Esse paradigma se concentra em organizar o código em terno de objetos, que são instâncias de classes e encapsulam dados (atributos) e comportamentos (métodos) relacionados.

Comparação com o Paradigma Estruturado

A POO é amplamente considerada a sucessora do Paradigma Estruturado (PE), que foca na lógica de procedimentos e funções para manipular dados globais. O POO oferece vantagens significativas em termos de organização, manutenção e reutilização de código, especialmente em sistemas complexos.
A tabela a seguir resume as principais diferenças entre os dois paradigmas:

Característica	Paradigma Estruturado (PE)	Paradigma Orientado a Objetos (POO)
Foco Principal	Procedimentos e Funções (Lógica)	Objetos (Dados e Comportamento)
Organização	Código em grandes blocos, sub-rotinas sem organização de dados [1].	Código em classes que centralizam dados e funcionalidades [1].
Dados	Dados manipulados por funções externas, podendo ser globais.	Dados (atributos) encapsulados dentro do objeto.
Reuso	Reuso de código através de funções.	Reuso de código através de herança, composição e polimorfismo.
Manutenção	Dificuldade em sistemas complexos devido à interdependência de dados e funções.	Facilidade de manutenção e expansão devido à modularidade e encapsulamento.

fonte: do autor

CONCEITOS FUNDAMENTAIS DA ORIENTAÇÃO A

OBJETOS

Classe

A Classe é o molde ou plano de construção para a criação de objetos. Ela define a estrutura (quais atributos terá) e o comportamento (quais métodos executará) que todos os objetos criados a partir dela possuirão.

Atributo

Os atributos são as características ou dados que definem o estado de um objeto em um determinado instante. São as variáveis declaradas dentro da classe. Em uma classe Carro , por exemplo, os atributos podem ser: **cor , modelo e velocidadeAtual .**

Método

Os Métodos são as funcionalidades ou comportamentos que um objeto pode executar. São as funções declaradas dentro da classe que geralmente manipulam os atributos do objeto. Na classe Carro , os métodos poderiam ser acelerar() , frear() ou ligar() .

Objeto

O Objeto é a entidade concreta criada a partir da classe. É a representação de um conceito do mundo real dentro do programa. Um objeto possui valores específicos para os atributos definidos em sua classe.

Instância de um Objeto

O termo instância de um objeto significa que o objeto foi **criado** (instanciado) a partir do molde da classe. A classe é a definição (o tipo), e a instância é a materialização dessa definição na memória do computador, com seus próprios valores de atributos. O processo de criação é chamado de **instanciação**.

- **Exemplo de classe e instância(PHP)**

Em PHP, uma classe é um molde que define os atributos e métodos de um tipo de objeto:

```php
<?php
// ---------------------------
// Definição da classe
// ---------------------------
class Pessoa {
    // Atributos (características)
    public $nome;
    public $idade;

    // Construtor (executado ao criar o objeto)
    public function __construct($nome, $idade) {
        $this->nome = $nome;
        $this->idade = $idade;
    }

    // Método (ação que o objeto pode fazer)
    public function falar() {
        echo "Olá, meu nome é {$this->nome} e tenho {$this->idade} anos.<br>";
    }
}
```

```php
php

<?php
$p1 = new Pessoa("João", 30); // Criando o 1º objeto
$p2 = new Pessoa("Maria", 25); // Criando o 2º objeto

$p1->falar(); // "Olá, meu nome é João e tenho 30 anos."
echo "<br>";
$p2->falar(); // "Olá, meu nome é Maria e tenho 25 anos."
?>
```

- Toda classe contém **atributos**, de modo global, deixamos **public $nome e $idade**, ou seja todas as pessoas da classe terão nome e idade.

- Executado ao criar o objeto.

```
public function __construct($nome, $idade) {
        $this->nome = $nome;
        $this->idade = $idade;
    }
```

__construct é um método especial chamado construtor.
Ele é executado automaticamente quando criamos um objeto da classe.

$this->nome e $this->idade dizem: “este objeto (instância) vai ter esses valores”.
neste caso a atributo **nome** está recebendo valores da variável **$nome** e o mesmo para idade.

Em sequência agregamos o **Método ou Ação** que o objetos podem fazer

```
    public function falar( ) {
        echo "Olá, meu nome é {$this->nome} e tenho {$this->idade} anos.<br>";
    }
}
```

falar() é um método, ou seja, uma ação que a pessoa pode realizar.
Exemplo:
Quando chamamos **$p1->falar();**, ele imprime a frase com o **nome e idade.**

Criação das instâncias (objetos).

```
$p1 = new Pessoa("João", 30);
$p2 = new Pessoa("Maria", 25);
```

Aqui estamos instanciando a classe Pessoa.

new Pessoa("João", 30) cria um objeto real, que é guardado em **$p1.**
new Pessoa("Maria", 25) cria outro objeto, guardado em **$p2**.

Cada objeto tem seus próprios valores de nome e idade. Mesmo sendo da mesma classe, eles são independentes.

Chamando o método 'falar' de cada objeto.

$p1->falar();
$p2->falar();
?>
$p1->falar(); → usa o método falar() do objeto $p1. Ele imprime:
"Olá, meu nome é João e tenho 30 anos."

$p2->falar(); → imprime:
*"Olá, meu nome é Maria e tenho 25 anos.**"***

OBS:

- Então quando o programa chega na linha do $p1->falar(); ele já executa a impressão.

- Ou seja, cada instância age de acordo com os seus próprios dados.

Modificadores de Acesso

Os modificadores de acesso são palavras-chave que controlam a visibilidade dos membros (atributos e métodos) de uma classe, sendo cruciais para o princípio do Encapsulamento (proteção dos dados). Abaixo exemplo:

```php
<?php
class ContaBancaria {
    public $titular;
    private $saldo;
    protected $agencia;

    public function __construct($titular, $saldoInicial, $agencia) {
        $this->titular = $titular;
        $this->saldo = $saldoInicial;
        $this->agencia = $agencia;
    }

    public function depositar($valor) {
        $this->saldo += $valor;
    }

    public function consultarSaldo() {
        return $this->saldo;
    }
}
```

```php
class ContaPremium extends ContaBancaria {
    public function bonus() {
        $this->saldo += 100;      // ERRO: saldo é private, não acessível
        $this->agencia = "9999";  // OK: agencia é protected, acessível na subclasse
    }
}

$conta = new ContaBancaria("Maria", 500, "1234");
$conta->depositar(200);
echo $conta->consultarSaldo(); // 700
?>
```

Etapa 1: definição da classe:

class Conta {

Aqui o PHP entende que estamos criando uma classe chamada Conta.
Uma classe é um modelo para criar objetos com propriedades e comportamentos.

Etapa 2: declaração dos atributos:

public $titular;
private $saldo;
protected $agencia;

Atributo		Significa	Quem pode acessar
`$titular`	`public`	visível para todos	qualquer parte do código
`$saldo`	`private`	visível só dentro da classe	apenas métodos da própria classe
`$agencia`	`protected`	visível na classe e em subclasses	classe e classes filhas

---

Etapa 3: o construtor

```
public function __construct($titular, $saldo, $agencia) {
    $this->titular = $titular;
    $this->saldo = $saldo;
    $this->agencia = $agencia;
}
```

O método __construct() é chamado **automaticamente** quando você cria um novo objeto.
Ele serve para **inicializar** os valores dos atributos.

*$this-> significa **"este objeto"**.*

Por exemplo, $this->titular se refere ao atributo da classe, não à variável que está entre parênteses.

Quando você cria o objeto new Conta("Maria", 500, "1234"), o PHP faz o seguinte internamente:

$titular = "Maria"
$saldo = 500
$agencia = "1234"

E guarda esses valores dentro do objeto recém-criado.

Etapa 4: método depositar()

```
public function depositar($valor) {
    $this->saldo += $valor;
}
```

Esse método **adiciona** um valor ao saldo.

$valor é o valor do depósito (por exemplo, 200).

$this->saldo += $valor soma esse valor ao saldo atual do objeto.

Se o saldo era 500, depois do depósito será 700.

Etapa 5: método consultarSaldo()

```
public function consultarSaldo() {
    return $this->saldo;
}
```

Esse método **retorna o saldo atual** do objeto.Como $saldo é private, ele não pode ser acessado diretamente fora da classe, então é preciso um método público para obter o valor.

Etapa 6: criação do objeto

```
$conta = new Conta("Maria", 500, "1234");
```

Aqui criamos uma instância da classe (um objeto real chamado $conta).

Esse objeto tem seus próprios valores:

titular = "Maria"

saldo = 500

agencia = "1234"

Etapa 7: chamada de métodos

$conta->depositar(200);

A seta **->** é usada para acessar métodos e atributos de um objeto.

Aqui, o PHP executa o método depositar(200) da classe Conta.

Resultado: o saldo passa de 500 → 700.

Etapa 8: exibir o saldo

echo $conta->consultarSaldo(); // 700

O método consultarSaldo() retorna o valor do saldo atual (700). O comando echo exibe esse valor na tela.

Resultado final

700

---

2. O que é encapsulamento em POO?

Em Programação Orientada a Objetos, **encapsulamento** é o princípio que diz que os **dados e o comportamento de um objeto devem ficar protegidos dentro da classe**, permitindo o acesso apenas por meio de métodos definidos.

Traduzindo:
A classe **controla o que pode ou não ser acessado**, evitando que outras partes do programa mexam diretamente em seu funcionamento interno.

Como isso acontece na classe Calculadora do exemplo?

Na calculadora citada:

✔ As operações (somar, subtrair, multiplicar, dividir) estão **todas dentro da classe Calculadora**
✔ O código de como cada operação funciona **fica escondido dentro da classe**
✔ Quem usa a calculadora **não precisa saber como ela faz o cálculo**, apenas chama o método

✔ **A calculadora não guarda estado**, ou seja, não salva valores internamente
✔ Ela apenas **recebe valores, calcula e devolve o resultado**
✔ Não possui atributos como "marca", "modelo", etc.

Isso também é considerado encapsulamento, pois **a classe controla como cada operação funciona e expõe apenas os métodos necessários**.

```
public class Calculadora {

    // Métodos encapsulados (a lógica está dentro da classe)
    public int somar(int a, int b) {
        return a + b;
    }
    public int subtrair(int a, int b) {
        return a - b;
    }
    public int multiplicar(int a, int b) {
        return a * b;
    }
    public double dividir(double a, double b) {
        return a / b;
    }
}
```

---

3. **Métodos Getter e Setter**

   São funções usadas em Programação Orientada a Objetos para **controlar o acesso aos atributos de uma classe**.

- **Getter**: método que retorna (lê) o valor de um atributo.

- **Setter**: método que altera (define) o valor de um atributo, muitas vezes verificando se o valor é válido.
- Eles existem para respeitar o **encapsulamento**, evitando que o atributo seja acessado ou modificado diretamente de fora da classe.

4. Tema

Uso da palavra reservada **$this** em Programação Orientada a Objetos no PHP.

Ideia Central

O **$this** é uma variável especial que representa **a própria instância (o próprio objeto)** da classe.
Ele é usado para acessar atributos e métodos internos do objeto dentro da própria classe.

Por que usamos $this?

Usamos o **$this** para:

- Diferenciar atributos da classe de variáveis locais com o mesmo nome;

- Permitir que um método altere ou leia dados **do objeto que o chamou**;

- Garantir que o valor modificado ou acessado pertence à instância correta da classe.

Sem o this, seria impossível, por exemplo, saber qual variável interna está sendo modificada quando há nomes iguais no método.

Exemplo

```php
<?php
class Pessoa {
    private $nome;

    public function setNome($nome) {
        $this->nome = $nome;
        // Aqui o $this indica que estamos alterando
        // o atributo 'nome' da instância do objeto
    }

    public function getNome() {
        return $this->nome;
        // Usando $this para acessar o valor armazenado no objeto
    }
}

// Uso
$p = new Pessoa();
$p->setNome("Ana");
echo $p->getNome(); // Resultado: Ana
?>
```

Explicação

- Dentro do método setNome(), a linguagem não saberia se nome se refere ao atributo da classe ou ao parâmetro recebido.

- Por isso usamos:

$this->nome

que significa: **“o atributo chamado nome desta instância do objeto”**.

---

5. Observação

Cada objeto criado de uma classe tem seu próprio espaço de memória.
O **$this garante que o método está acessando os dados do objeto específico que o chamou**, e não de outro.

---

6. Interfaces em Programação Orientada a Objetos

O que é uma interface em POO?

Uma interface é um **contrato** que define *quais métodos uma classe deve implementar*, sem fornecer código interno. Ela lista apenas as **assinaturas dos métodos**, garantindo que todas as classes que a implementem sigam o mesmo padrão. Interfaces não podem ser instanciadas diretamente.
Ideia central: *define “o que fazer”, não “como fazer”.*

Diferença entre interface e classe abstrata

Interface	Classe Abstrata
Não possui implementação de métodos (somente assinaturas).	Pode ter métodos abstratos **e** métodos já implementados.

Não possui atributos de instância (pode ter constantes).	Pode ter atributos e construtores.
Permite múltiplas implementações por uma mesma classe.	Permite herança única (uma classe abstrata por classe concreta).
Ideal para padronizar comportamentos sem herança.	Ideal quando existe código comum a ser reutilizado.

Por que interfaces melhoram manutenção e extensibilidade?

- **Desacoplamento**: o código depende da interface, não da classe concreta, facilitando substituições.
- **Flexibilidade e polimorfismo**: várias classes diferentes podem implementar o mesmo contrato.
- **Padronização**: deixa claro quais métodos devem existir, evitando inconsistências no sistema.
- **Extensibilidade**: novas classes podem ser adicionadas sem alterar código já existente.

Em resumo: **interfaces tornam o código mais organizado, fácil de manter e pronto para crescer**.

Referência

ALURA. *POO: o que é programação orientada a objetos?* Alura, 22 out. 2019. Disponível em: https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos. Acesso em: 29 out. 2025.

GILLIS, Alexander; LEWIS, Sarah. *What is object-oriented programming (OOP)?* TechTarget, 14 jun. 2024. Disponível em: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP. Acesso em: 30 out. 2025.

PHP.NET. *Visibility — Manual PHP.* The PHP Group. Disponível em: https://www.php.net/manual/en/language.oop5.visibility.php. Acesso em: 30 out. 2025.

PROGRAMÁTICO. *O que é Programação Orientada a Objetos | Programático.* Programático, 2025. Disponível em: https://programatico.com.br/cursos/programacao-python/classes-objetos/. Acesso em: 29 out. 2025.

PROGRAMÍCIO. *Programação Orientada a Objetos: Classes e Objetos em Java.* Programício, 26 jul. 2025. Disponível em: https://www.programicio.com/java/java-se/programacao-orientada-a-objetos/classes-e-objetos. Acesso em: 29 out. 2025.

DEVEDIA. Orientação a Objetos – Encapsulamento. DevMedia, 2024. Disponível em: https://www.devmedia.com.br/orientacao-a-objetos-parte-ii/7161. Acesso em: 24 nov. 2025.

TECHTARGET. Encapsulation. TechTarget, 2024. Disponível em: https://www.techtarget.com/searchnetworking/definition/encapsulation. Acesso em: 24 nov. 2025.

FREECODECAMP. *Getters e Setters em Java explicados*. FreeCodeCamp (em português), 2024. Disponível em:

https://www.freecodecamp.org/portuguese/news/getters-e-setters-em-java-explicados/. Acesso em: 24 nov. 2025.

PHP. Pseudo-variáveis. In: Manual PHP. [S. l.]: The PHP Group, [2025]. Disponível em: https://www.php.net/manual/pt_BR/language.variables.basics.php. Acesso em: 24 nov. 2025.

PHP Group. *PHP: The Basics — Classes and Objects*. PHP Manual. Disponível em: https://www.php.net/manual/en/language.oop5.basic.php. Acesso em: 26 nov. 2025.

PHP.NET. *Interfaces de Objetos*. Disponível em: https://www.php.net/manual/pt_BR/language.oop5.interfaces.php. Acesso em: 01 dez. 2025.

DEVMEDIA. *Polimorfismo, Classes Abstratas e Interfaces*. Disponível em: https://www.devmedia.com.br/polimorfismo-classes-abstratas-e-interfaces-fundamentos-da-poo-em-java/26387. Acesso em: 01 dez. 2025.

DIO.ME. *Interfaces x Classes Abstratas*. Disponível em: https://www.dio.me/articles/interfaces-x-classes-abstratas. Acesso em: 01 dez. 2025.

DELFTSTACK. *Diferença entre Classe Abstrata e Interface*. Disponível em: https://www.delftstack.com/pt/howto/java/abstract-class-vs-interface-java/. Acesso em: 01 dez. 2025.