UNIVERSIDADE DO OESTE DE SANTA CATARINA – UNOESC

Construtores, Classes e Padrões de Projeto em PHP

Autor: Leandro Otavio Cordova Vieira

Curso de Ciência da Computação
Unidade Curricular: Programação II

1

Construtores, Classes e Padrões de Projeto em PHP

1.1 Introdução

A programação orientada a objetos (POO) é um dos paradigmas mais influentes da engenharia de software moderna. No PHP, especialmente a partir da versão 5 e consolidando-se nas versões 7 e 8, o suporte a orientação a objetos se tornou robusto e alinhado às melhores práticas de desenvolvimento. Este capítulo tem como objetivo aprofundar o estudo dos **construtores e destruidores**, analisar princípios de **código limpo aplicados a classes** e discutir a aplicação de **design patterns** (padrões de projeto) no contexto do PHP.

1.2 Construtores e Destrutores no PHP

1.2.1 O Método `__construct()`

O método `__construct()` é automaticamente invocado quando um objeto é instanciado. Ele é utilizado para inicializar os atributos da classe e preparar o objeto para uso.

```php
class Usuario {
    private string $nome;

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

    public function getNome(): string {
        return $this->nome;
    }
```

```php
}

$usuario = new Usuario("Maria");
echo $usuario->getNome();
```

1.2.2 O Método `__destruct()`

Já o método `__destruct()` é chamado quando o objeto é destruído ou o script finalizado. Ele é útil para liberar recursos, fechar conexões com banco de dados ou encerrar sessões.

```php
class Conexao {
    private $link;

    public function __construct() {
        $this->link = mysqli_connect("localhost","root","","db");
    }

    public function __destruct() {
        mysqli_close($this->link);
    }
}
```

1.2.3 Boas Práticas

- Mantenha construtores curtos e claros.
- Prefira a **injeção de dependência**, em vez de criar dependências rígidas no construtor.
- Utilize destruidores apenas quando necessário, visto que o PHP já gerencia memória automaticamente.

1.3 Clean Code e Classes

1.3.1 Princípios Fundamentais

Baseando-se no livro *Clean Code*, de Robert C. Martin, destacam-se alguns princípios fundamentais:

- **Classes devem ser pequenas:** cada classe deve ter apenas uma responsabilidade.

- **Nomes significativos:** o nome da classe deve refletir sua intenção.
- **Alta coesão:** métodos e atributos devem estar intimamente relacionados.
- **Baixo acoplamento:** as classes devem depender minimamente umas das outras.

1.3.2 Más Práticas Comuns

- Classes "Deus", que concentram muitas responsabilidades.
- Métodos longos e de difícil leitura.
- Falta de encapsulamento, expondo atributos diretamente.

1.3.3 Refatoração em PHP

A refatoração de classes envolve a criação de estruturas mais claras e coesas. Por exemplo, ao invés de uma classe `Sistema` que faz tudo, criamos classes menores, como `Usuario`, `Produto` e `Pedido`.

1.4 Design Patterns em PHP

1.4.1 Singleton

O padrão Singleton garante que apenas uma instância de uma classe exista durante a execução.

```php
class Conexao {
    private static ?Conexao $instancia = null;

    private function __construct() {}

    public static function getInstance(): Conexao {
        if (self::$instancia === null) {
            self::$instancia = new Conexao();
        }
        return self::$instancia;
    }
}
```

1.4.2 Factory Method

O Factory Method centraliza a criação de objetos, permitindo que subclasses decidam qual instância criar.

1.4.3 Strategy

O padrão Strategy permite a seleção de algoritmos em tempo de execução.

1.4.4 Observer

O padrão Observer estabelece uma relação de dependência, em que mudanças em um objeto são notificadas automaticamente a outros objetos dependentes.

1.5 Integração entre os Conceitos

Construtores bem definidos fornecem clareza sobre a inicialização de objetos. Classes que seguem princípios de *Clean Code* tornam-se mais fáceis de manter. Padrões de projeto fornecem soluções reutilizáveis e organizadas para problemas recorrentes.

1.6 Conclusão

A maturidade no desenvolvimento PHP não depende apenas de conhecer a sintaxe, mas de adotar práticas arquiteturais sólidas. O domínio de construtores e destruidores, somado à aplicação de princípios de *Clean Code* e à utilização de *design patterns*, contribui para a construção de sistemas mais robustos, escaláveis e manuteníveis.

1.7 Referências

- Documentação Oficial do PHP: `https://www.php.net/manual/pt_BR/`
- MARTIN, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship.* Prentice Hall, 2008.
- GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.