Resumo de Capítulos segundo Ebook disponibilizado no portal

Acadêmico: João Victor Surdi de Melo

Capítulo 2:

O capítulo começa nos mostrando que a POO (Programação Orientada à Objetos) é um paradigma central no desenvolvimento de softwares modernos, fala também, que a linguagem PHP é uma grande aliada, já que, desde sua versão mais recentes, incorpora essa metodologia com robustez.

Logo após, o capítulo nos apresenta o conceito de Classe e Objeto, onde Classe é um molde e Objeto o que será moldado pela classe:

*“onde uma classe funciona como um molde para a criação de objetos, que são instâncias dessa classe.”*

Um exemplo de código abaixo, onde Classe é definida pela palavra-chave “class”:

Class Carro {

public $cor;

public $modelo;

public function \_construct($cor, $modelo) {

$this->cor = $cor;

$this->modelo = $modelo;

}

public function detalhesCarro(){

return “Modelo: ”. $this->modelo . “Cor: ”. $this->cor;

}

}

Objetos são criados com o operador “new”, como mostra o exemplo abaixo:

$meuCarro = new Carro(“vermelho”, “ferrari”);

echo $meuCarro->detalhesCarro();

A saída dessa operação será:

Modelo: ferrari Cor: vermelho

É necessário que entendamos a necessidade desse tipo de metodologia no mercado atual...

O uso adequado dessa metodologia faz com que o desenvolvedor se encontre em um código mais limpo e organizado, tornando mais fácil sua manutenção e expansão, sendo mais fácil e prático projetar e modelar processos reais dentro do software.

Além de tornar o código mais limpo e modular, também nos deixa preparados para trabalhar com conceitos mais avançados da POO, como herança e polimorfismo que serão vistos mais à frente neste documento.

Outro conceito fundamental também na POO é o conceito de encapsulamento, que mantém os dados de dentro da classe protegidos, através dos níveis de segurança e proteção (public, private ou protected), este método, como o próprio nome sugere, encapsula todos os dados dentro da classe para não ser possível o acesso de terceiros, apenas sendo possível acessar os métodos públicos.

A compreensão desses fundamentos que citei acima, são necessários não só para fortalecer a base dos programas que são construídos, mas também agiliza na manutenção do sistema e reutilização de código.

Definições básicas:

**- Classe:** É o “Molde” para o objeto, ele irá definir os atributos relacionados ao objeto.

**- Objeto:** Instâncias da Classe, objetos seguirão e manterão os estado e comportamentos definidos por sua classe.

**- Encapsulamento:** Protege os dados de dentro da classe, apresentando apenas o que é necessário por meio de métodos públicos

**- Construtores:** São principalmente úteis para criar novas instâncias de objetos e realizar quaisquer configurações necessárias

Atributos, ou comumente chamadas propriedades, são, como o próprio nome sugere, os atributos dentro da classe, vamos resgatar o exemplo de cima, onde “$cor” e “$modelo” são atributos da classe “Carro”.

“

Class Carro {

public $cor;

public $modelo;

public function \_construct($cor, $modelo) {

$this->cor = $cor;

$this->modelo = $modelo;

}

public function detalhesCarro(){

return “Modelo: ”. $this->modelo . “Cor: ”. $this->cor;

}

}

“

Como podem ver, atributos têm níveis de segurança e acessibilidade, esses podem ser: “public”, “private” ou “protected”.

Atributo público (“public”): Podem ser acessados de qualquer lugar.

Atributo protegido (“protected”): Pode ser acessado somente dentro da própria classe e/ou subclasse.

Atributo privado (“private”): pode ser acessado somente de sua classe de onde foi criado.

Métodos são parecidos com os atributos em muitos aspectos, assim como eles, os métodos possuem os níveis de acessibilidade (“public”, “protected” e “private”).

Métodos são, de forma simplificada, operações que ocorrem entre os objetos e/ou nos dados da classe.

Partindo para o capítulo 3 do ebook, vemos um pouco mais sobre os pilares da POO seriam eles: Encapsulamento, Polimorfismo e Herança.

Encapsulamento engloba o que já descrevi anteriormente, sobre os “Níveis de Acessibilidade”, o termo correto para esses fatores é chamado de Encapsulamento.

Ele é fundamental para um bom uso do método POO, já que, ele quem irá garantir a segurança do código e a organização dos componentes internos.

A principal vantagem do uso de encapsulamento no código, é a proteção do estado interno do objeto... Muitas vezes em um sistema, para uso externo, não é preciso mostrar todo o funcionamento interno dos objetos e classes, nesses momentos o encapsulamento se encaixa perfeitamente.

Além de, o encapsulamento ajudar também na parte estrutural do código, deixando mais fácil de compreender e realizar a manutenção.

Logo após o Encapsulamento, temos descrito o conceito de Herança, que é nada mais nada menos que reutilizar uma classe já existente em outras, causando uma economia na hora de “reutilizar” o código e deixar a manutenção do sistema mais simples.

Porém, nem tudo são flores, o mal uso da Herança pode causar um problema conhecido como “Explosão de Herança”, que é quando a hierarquia fica muito grande, atrapalhando mais o desenvolvedor do que ajudando, portanto, esse recurso deve ser entendido e usado com sabedoria para que não se crie uma hierarquia desnecessariamente extensa.

Me resumo, a Herança é uma ferramenta poderosa, se usada com sabedoria a fim de maximizar as vantagens e diminuir as complicações do código.

Após isso dito, passamos para um dos conceitos centrais e mais importantes da POO, o Polimorfismo...

Polimorfismo é em resumo você estender as funcionalidades do sistema, ele permite que objetos de diferentes classes sejam tratados como instâncias de uma classe “Pai” em comum.

O polimorfismo pode ser implementado principalmente de 2 maneiras diferentes:

A primeira forma é a Sobrecarga, esta forma permite que vários métodos compartilhem o mesmo nome, porém com listas de parâmetros ou tipos retornados diferentes.

Já a outra maneira é a substituição, ou sobrescrita, ela ocorre quando uma subclasse redefine completamente implementação de um método herdado da superclasse.

O polimorfismo não só melhora a compreensão do sistema como um todo, mas também melhora a manutenção (assim como os outros pilares) do software e suas funcionalidades, abrindo portas para designs mais dinâmicos e adaptáveis.

O capítulo 4 se inicia falando sobre relacionamentos entre as classes, mais precisamente sobre Associação...

Associação é muito útil e fundamental na orientação a objetos, principalmente no contexto do PHP 8.3, onde a modelagem e relacionamento de classes é muito importante para o desenvolvimento de softwares robustos.

Em uma associação, uma ou mais classes são ligadas através de suas instâncias, um aspecto crucial da associação é a multiplicidade, ela define quantas instâncias de uma classe podem se relacionar com outras de outra classe.

Essa associação pode ser tanto unidirecional quanto bidirecional... Em uma associação bidirecional, as classes tem ciência uma da outra, e comunicam entre si, já na unidirecional uma é referenciada pela outra

Outro tipo de relacionamento é a Agregação, ela descreve uma forma especializada de associação, ela descreve uma relação “todo-parte” onde duas classes se relacionam, mas não são dependentes uma da outra, uma existe independente da outra existir.

Na Agregação, temos um design mais “limpo”, organizado, sendo agradável a quem preza por design de software, porém, não só pra quem preza por isso, e sim para todos já que torna mais fácil a reutilização de código e manutenção, além da alteração de parâmetros caso necessário.

Por último, mas não menos importante, temos a Composição, assim como a Agregação, esse método trabalha com um “todo-parte”, porém, diferente da Agregação, esse método se faz crucial a existência de um todo...

Ou seja, a existência de uma classe, depende totalmente da superclasse relacionada, e, caso essa superclasse for excluída, suas subclasses também serão.

As vantagens desse método, além do encapsulamento e a segurança que ele apresenta, é uma organização clara e uma melhor compreensão do clico de vida dos componentes, mantendo o programa mais robusto e simplifica o gerenciamento das tarefas e classes.