SOLID com PHP: Princípios da programação orientada a objetos

- Esse curso capacita a:
  - Aprender conceitos avançados de orientação a objetos;
  - Escrever código coeso com Single Responsibility Principle;
  - Saber como lidar com acoplamento;
  - Entender a fundo os ganhos do encapsulamento;
  - Dominar os princípios de código sólido;
  - Técnicas e exemplos em PHP;

Aulas:

1. **Preparando o terreno:**
   - Coesão:
     - Uma classe coesa faz bem uma única coisa;
     - Classes coesas não devem ter várias responsabilidades;
   - Encapsulamento:
     - Getters e setters não são formas eficientes de aplicar encapsulamento;
     - É interessante fornecer acesso apenas ao que é necessário em nossas classes;
     - O encapsulamento torna o uso das nossas classes mais fácil e intuitivo;
   - Acoplamento:
     - Acoplamento é a dependência entre classes;
     - Acoplamento nem sempre é ruim, e que é impossível criar um sistema sem nenhum acoplamento;
     - Devemos controlar o nível de acoplamento na nossa aplicação;

2. **Melhorando a coesão:**
   - Que classes/métodos/funções/módulos devem ter uma única responsabilidade bem definida;
   - Que, segundo o **Princípio de Responsabilidade Única (SRP)**, uma classe deve ter um e apenas um motivo para ser alterada;
   - Como realizar uma refatoração no nosso sistema;
   - Como extrair uma classe;

3. **Trabalhando no acoplamento:**
   - Que cada classe deve conhecer e ser responsável por suas próprias regras de negócio;
   - Que o **Princípio Aberto/Fechado (OCP)** diz que um sistema deve ser aberto para a extensão, mas fechado para a modificação.
     - Isso significa que devemos poder criar novas funcionalidades e estender o sistema sem precisar modificar muitas classes já existentes;

- Uma classe tende a crescer “para sempre” é uma forte candidata a sofrer alguma espécie de refatoração;

4. **Quebra de confiança:**
    - Que, embora a assinatura de um método esteja sendo respeitada em uma herança, ainda assim podemos estar quebrando algum contrato;
    - Que o **Princípio de Substituição de Liskov (LSP)** diz que devemos poder substituir classes base por suas classes derivadas em qualquer lugar, sem problema;
    - Que não devemos alterar um comportamento de um método estendido, mesmo que a assinatura seja mantida;

5. **Encapsulando melhor:**
    - Que é mais interessante e mais seguro para o nosso código depender de interfaces (classes abstratas, assinaturas de métodos e interfaces em si) do que das implementações de uma classe;
    - Que as interfaces são menos propensas a sofrer mudanças enquanto implementações podem mudar a qualquer momento;
    - Que o **Princípio de Inversão de Dependência (DIP)** diz que implementações devem depender de abstrações e abstrações não devem depender de implementações;
    - Que as interfaces devem definir apenas os métodos que fazem sentido para seu contexto;
    - Que uma classe pode implementar diversas interfaces;
    - Que o **Princípio de Segregação de Interfaces (ISP)** diz que uma classe não de ser obrigada a implementar um método que ela não precisa;
    - Os conceitos aprendidos neste treinamento formam o acrônimo **SOLID**:
        - **S**ingle Responsibility Principle;
        - **O**pen Closed Principle;
        - **L**iskov Substituition Principle;
        - **I**nterface Segregation Principle;
        - **D**ependency Inversion Principle;