Linguagem de Programação II

Prof.ª Ma. Jessica Oliveira

FBr Faculdade Brasília

Aula 06 – 09/09/2024

Introdução à Orientação a Objetos

O que é Orientação a Objetos?

- É um **paradigma de programação** que organiza o *software* em torno de "objetos", que são instâncias de "classes".
- Cada objeto representa uma entidade do mundo real ou conceito abstrato, encapsulando tanto os dados quanto os comportamentos associados a essa entidade.
- Diferente do **paradigma procedural**, que se concentra na execução de funções ou procedimentos, a OO foca na criação de estruturas que podem ser reutilizadas, estendidas e organizadas de maneira lógica.

Procedural *versus* Orientada a Objetos

- **Programação Procedural:** baseia-se na execução sequencial de instruções, agrupadas em funções ou procedimentos. Cada função realiza uma tarefa específica, e os dados são frequentemente manipulados diretamente por essas funções, sem uma estrutura definida para representar entidades complexas.
- **Programação Orientada a Objetos:** introduz o conceito de classes e objetos, onde os dados (atributos) e os comportamentos (métodos) são encapsulados em unidades de código. Em vez de manipular dados diretamente, a OO manipula objetos que são representações desses dados, permitindo uma modelagem mais intuitiva e próxima do mundo real.

Modelagem do mundo real

- A orientação a objetos permite representar entidades do mundo real como **objetos** que possuem características (**atributos**) e comportamentos (**métodos**).
- Por exemplo, em um sistema de gerenciamento de biblioteca, um `Livro` pode ser representado como uma **classe**, com **atributos** como `título`, `autor` e `editora`, e **métodos** como `emprestar` e `devolver`.
- Isso torna a modelagem do sistema mais **intuitiva** e facilita a manutenção e extensão do código.

Por que usar Orientação a Objetos?

- A OO facilita a **reutilização de código** por meio de herança, onde novas classes podem ser criadas a partir de classes existentes, reutilizando e estendendo funcionalidades sem a necessidade de reescrever o código.
- Além disso, classes bem projetadas podem ser reutilizadas em diferentes partes de um sistema ou mesmo em diferentes projetos.

Por que usar Orientação a Objetos?

- A modularidade proporcionada pela OO, através de encapsulamento e herança, torna o **código mais fácil de manter**.
- Mudanças em uma parte do sistema podem ser feitas isoladamente, sem afetar outras partes, desde que as interfaces públicas dos objetos sejam mantidas.
- Isso reduz a chance de introduzir erros ao modificar o código e facilita a evolução do sistema ao longo do tempo.

Por que usar Orientação a Objetos?

- A OO **promove uma organização mais lógica e natural do código**, agrupando dados e comportamentos relacionados em unidades (classes) que refletem as entidades do domínio do problema.
- Isso facilita a compreensão do código por outros desenvolvedores e torna o processo de desenvolvimento mais eficiente.

Conceitos Fundamentais da Orientação a Objetos

Classe

- **Definição:** é um modelo ou *blueprint* para a criação de objetos. Ela define um conjunto de atributos e métodos que os objetos criados a partir dela terão. Em termos mais práticos, uma classe pode ser vista como **uma descrição abstrata de um conjunto de objetos com características e comportamentos comuns**.
- **Atributos e Métodos:**
  - **Atributos:** são as **características** ou **propriedades** da classe, representando o estado de um objeto. Por exemplo, em uma classe “Carro”, os atributos poderiam ser “cor”, “marca”, “modelo” e “ano”.
  - **Métodos:** são as **ações** ou **comportamentos** que um objeto pode realizar. Na classe “Carro”, um método poderia ser “acelerar()”, “frear()”, ou “ligar()”. Métodos operam sobre os atributos e podem alterar o estado do objeto.

Objeto

- **Definição:** é uma **instância** de uma classe. Quando uma classe é definida, ela serve como um molde para a criação de objetos. Cada objeto possui seus próprios valores para os atributos definidos pela classe, permitindo que diferentes objetos tenham diferentes estados, mesmo que compartilhem a mesma estrutura.
- **Exemplo prático:** se a classe “Carro” define atributos como “cor”, “marca”, “modelo” e “ano”, dois objetos da classe “Carro” poderiam ter valores diferentes para esses atributos.

- Objeto 1: ‘cor = cinza’, ‘marca = Fiat’, ‘modelo = Marea’, ‘ano = 2004’

- Objeto 2: ‘cor = preto’, ‘marca = Jeep’, ‘modelo = Compass’, ‘ano = 2025’

Encapsulamento

- **Definição:** é o conceito de **esconder a implementação interna de um objeto**, expondo apenas o que é necessário para a interação com o objeto. Isso é realizado por meio de modificadores de acesso, que controlam a visibilidade dos atributos e métodos.
- **Vantagens:**
  - **Segurança:** protege os dados de um objeto contra acessos não autorizados ou alterações indesejadas.
  - **Modularidade:** isola a implementação interna, permitindo que o código seja alterado sem afetar o restante do sistema.

Encapsulamento

- **Modificadores de acesso:**
  - ***public*:** permite que um atributo ou método seja acessado de qualquer lugar.
  - ***private*:** restringe o acesso a um atributo ou método apenas à própria classe.
  - ***protected*:** permite o acesso a um atributo ou método dentro da classe e em suas subclasses.

Herança

- **Definição:** é o mecanismo que permite que uma nova classe seja criada com base em uma classe existente. A nova classe, chamada de subclasse ou classe-filho, herda atributos e métodos da classe existente, chamada de superclasse ou classe-pai.
- **Vantagens:**
  - **Reutilização de código:** código comum é definido na superclasse e reutilizado em todas as subclasses.
  - **Extensibilidade:** subclasses podem adicionar novas funcionalidades ou modificar comportamentos existentes, facilitando a adaptação e evolução do código.

Herança

- **Hierarquias de classes:** a herança permite a criação de hierarquias de classes, onde classes mais genéricas (superclasses) podem ser especializadas em classes mais específicas (subclasses).
- Por exemplo, uma classe “Veículo” pode ser a superclasse de “Carro” e “Motocicleta”, com cada uma dessas subclasses adicionando ou modificando atributos e métodos específicos.

Polimorfismo

- **Definição:** é a capacidade de um objeto de assumir diferentes formas. Na prática, isso significa que um método pode se comportar de diferentes maneiras dependendo da classe do objeto que o invoca.

Polimorfismo

- **Sobrecarga e Sobrescrita de Métodos:**
  - **Sobrecarga:** permite que múltiplos métodos com o mesmo nome sejam definidos, desde que tenham diferentes assinaturas (ou seja, diferentes parâmetros). Isso permite que o mesmo método execute diferentes ações dependendo dos parâmetros recebidos.
  - **Sobrescrita:** permite que uma subclasse forneça uma implementação específica de um método já definido na sua superclasse. Isso é útil para modificar ou especializar o comportamento herdado de uma superclasse.

Polimorfismo

- **Exemplo prático:** considere uma classe “Animal” com um método “falar()”. Subclasses como “Cachorro” e “Gato” podem sobrescrever esse método para implementar comportamentos específicos, como “latir()” e “miar()”, respectivamente.

Declaração de Classes e Criação de Objetos em PHP

Atributos e Métodos

Atributos

- **Definição:** são variáveis definidas dentro de uma classe que armazenam o estado de um objeto. Eles representam as características do objeto, como nome, cor, tamanho, etc.
- **Tipos de dados:** podem ser de vários tipos de dados, como *int*, *string*, *float*, *array*, etc. Esses tipos determinam o tipo de valor que pode ser armazenado no atributo.

```php
<?
class Pessoa {
    public $nome;   //string
    public $idade;  //int
    public $altura; //float
}
?>
```

Métodos

- **Definição:** são **funções definidas dentro de uma classe** que realizam ações ou operações sobre os atributos de um objeto. Eles representam o **comportamento** do objeto.
- **Parâmetros e retorno de valores:** métodos podem receber parâmetros, que são valores passados para o método para influenciar seu comportamento. Eles também podem retornar valores, que são resultados da execução do método.

```
?>
class Calculadora {
    public function somar($a, $b) {
        return $a + $b;
    }
}
?>
```

Visibilidade (*public, private, protected*)

public

- Membros (atributos e métodos) declarados como "*public*" são acessíveis de qualquer lugar do código, dentro ou fora da classe.
- Isso significa que qualquer parte do código pode modificar ou acessar esses membros.

private

- Membros declarados como “*private*” são acessíveis apenas dentro da própria classe onde foram definidos.
- Eles não podem ser acessados diretamente de fora da classe, nem mesmo por subclasses.
- Isso é útil para proteger dados sensíveis ou garantir que certos métodos não sejam utilizados incorretamente.

protected

- Membros declarados como “*protected*” são acessíveis dentro da classe onde foram definidos e em qualquer subclasse derivada.
- Isso permite que subclasses herdem e utilizem esses membros, mas eles ainda estão protegidos contra acessos externos.

Vamos para a prática?

Não esqueçam, todos do trio devem realizar as entregas!

Microprojeto 2: Arquitetura Orientada a Objetos

IMPORTANTE!

- **Compactar a pasta do projeto** e subir a mesma no AVA até às 21h50min de **hoje** (09/09/2024).
- O **relatório** pode ser enviado **até amanhã** (10/09/2024) às 23h59min.
- **ATENTE-SE À ORGANIZAÇÃO FEITA NO AVA!** Cada microprojeto (e cada parte dele) tem seu espaço para ser submetido.
- Se eu receber não nada, não conseguirei avaliar seu projeto e sua nota será **ZERO**!

Passo 1: Criação das Classes

- Crie uma classe `Evento` que representará os eventos do sistema.
- Esta classe terá atributos como `nome`, `data`, `local`, e `descricao`.
- Implemente métodos na classe para gerenciar esses atributos (*getters* e *setters*).

Passo 2: Refatoração do código

- Refatore o código do `Microprojeto 1` para encapsular a lógica de negócios dentro da classe `Evento`.
- Substitua a lógica procedural por uma abordagem orientada a objetos.

Passo 3: Adição de outras classes

- Para expandir o sistema, crie outras classes como `Organizador` e `Participante`.
- Cada classe **deve** ter seus próprios atributos e métodos, seguindo a lógica da Orientação a Objetos.

Passo 4: Testes

- Teste as funcionalidades diretamente no código PHP, criando e manipulando objetos das classes `Evento`, `Organizador`, e `Participante`.
- **Objetivo:** Garantir que as classes estão funcionando corretamente e que a lógica do sistema foi bem encapsulada.

Na próxima aula...

Aula 06 (09/09/2024) - Formulários HTML e Introdução ao CSS.

Dúvidas?

jessica.oliveira@fbr.edu.br