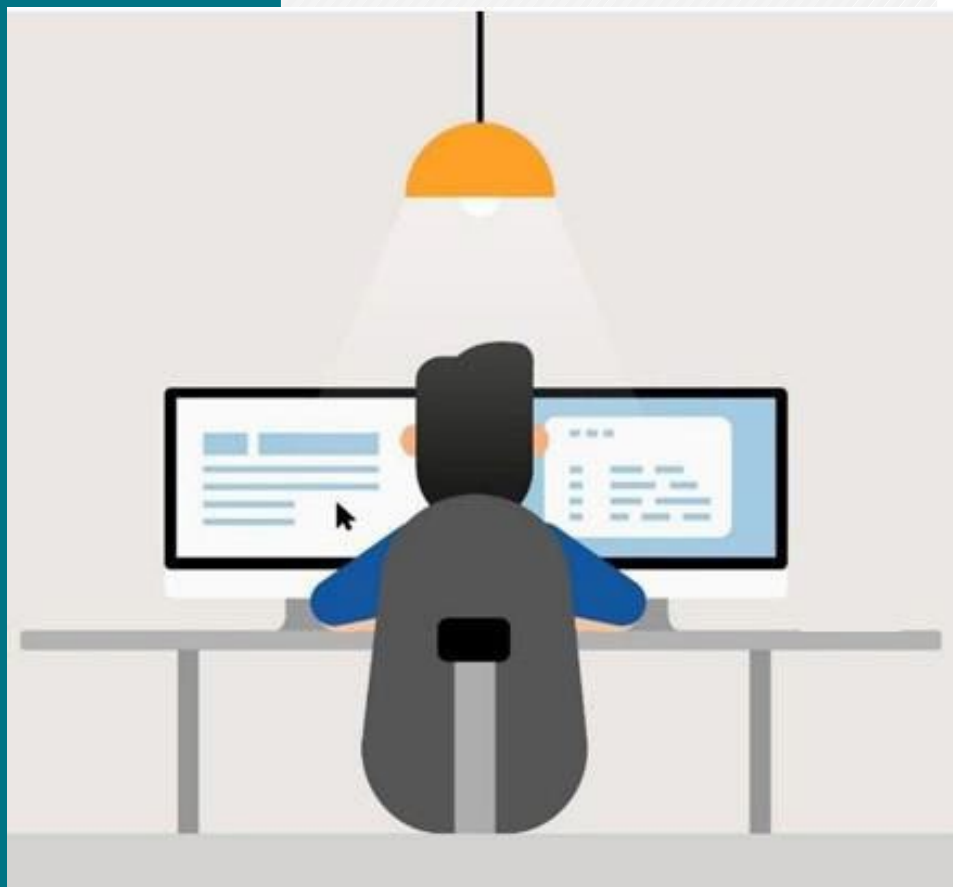

AGENDA 09

PHP - ORIENTADO A OBJETO





MERGULHANDO NO TEMA

Como iniciar a programar Orientado a Objetos?

Claro que, para começarmos a programar com orientação a objetos, é fundamental já termos compreendido os conceitos de classes e objetos. Apenas para relembrar: uma classe é uma estrutura que define um tipo de dado, podendo conter atributos (variáveis) e funções (métodos) para acessar e manipular esses atributos.

Relembre o conceito de Orientação a Objetos visto na agenda 11 do módulo 1 do curso:

[Material Mergulhando no tema Módulo 1](#)

Criando a Classe

Vamos começar desenvolvendo uma classe em PHP. Para isso, abra o editor de código Visual Studio Code (VS Code). Em seguida, crie um novo arquivo e salve-o com o nome **Pessoa.php**. Lembre-se de incluir os delimitadores do PHP no início do arquivo (`<?php ... ?>`).

Para definir uma classe em PHP, utilizamos a palavra reservada `class`, seguida pelo nome da classe. Veja um exemplo básico:

```
<?php
class Pessoa{ }
?>
```



Importante: Lembre-se de que a primeira letra do nome da classe deve ser maiúscula. Embora essa convenção não seja uma exigência da linguagem de programação, ela é uma boa prática amplamente adotada no paradigma de programação orientada a objetos. Seguir essa prática facilita a leitura e manutenção do código, além de torná-lo mais consistente.

Para a criação dos atributos, basta declará-los como uma variável, como podemos ver no código a seguir:

```
<?php
class Pessoa{
    public $nome;
    public $sobrenome;
}
?>
```

Observe que foi utilizado a palavra reservada **public**, que define a visibilidade do atributo. Isso significa que, ao instanciar o objeto, será possível acessar os atributos diretamente de fora da classe.

Após salvar as alterações, a classe **Pessoa** com dois atributos está criada!

Instanciando o objeto da classe

Originalmente, o PHP não foi desenvolvido como uma linguagem orientada a objetos. Por isso, muitos programadores PHP adotavam a programação estruturada. Basicamente, criavam arquivos PHP e organizavam funções relacionadas, de acordo com sua utilização. Um exemplo comum seria o desenvolvimento de um CRUD para Banco de Dados, onde todas as funções para inserir, atualizar, deletar e pesquisar dados seriam colocadas em um único arquivo. Depois, bastaria incluir esse arquivo nas páginas que precisassem dessas funcionalidades.

Vale lembrar que essa inclusão pode ser feita utilizando as instruções `include`, `require`, `include_once` ou `require_once`, dependendo da necessidade de verificação da existência prévia do arquivo.

- `Include` - Quando não encontra o arquivo retorna um Warning.
- `Require` - Quando não encontra o arquivo retorna um Fatal Error.
- `Include_once` e `Require_once` - A diferença está apenas ao tentar incluir o arquivo, se o mesmo já foi incluído será retornado falso e não será incluído novamente.

Por que todas essas explicações?

Simples! Para garantir que a classe criada possa ser utilizada e permita a instanciação de um objeto a partir dela, no index de seu site precisamos fazer a inclusão deste arquivo.

Agora, vamos instanciar objeto da classe recém-criada, **Pessoa**. Para isso, devemos criar um novo arquivo PHP e chamá-lo de **index.php**.



Todos estes arquivos estão no diretório root, do servidor apache.

No arquivo **index.php**, vamos fazer a inclusão da classe **Pessoa**, que acabamos de desenvolver:

```
<body>
  <?php
    include_once 'Pessoa.php';
  ?>
</body>
```

O código apresentado garante que o objeto pode ser instanciado sem problemas. Agora, basta implementar o código a seguir:

```
$p = new Pessoa();
$p->nome = "Zeca";
$p->sobrenome = "Silva";

echo $p->nome.' '. $p->sobrenome;
```

Vamos entender o código:

- ✓ Primeiro, declaramos uma variável **\$p** e, com a palavra-chave **new** juntamente com o construtor da classe, conseguimos criar uma instância.
- ✓ Nas duas linhas seguintes, atribuímos valores aos atributos. Note que, para acessar os atributos, é necessário utilizar o operador **->**.
- ✓ Em seguida, usamos o comando **echo** para exibir na página o conteúdo dos atributos, resultando no seguinte:

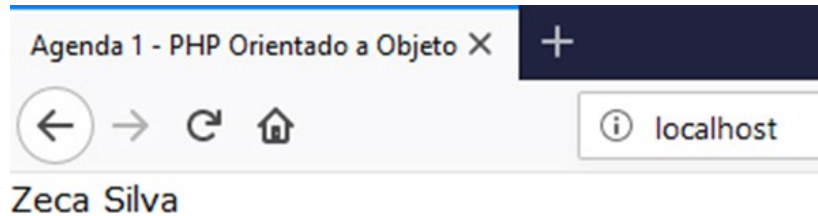


Imagem 1 – Resultado da codificação e uso das instâncias em nossa página html.

Não se esqueça!

O método construtor é utilizado para definir o comportamento inicial de um objeto, atribuindo valores iniciais aos seus atributos. O construtor é automaticamente executado quando instanciamos um objeto usando o operador **new** seguido do nome da classe. É importante destacar que o método construtor não deve retornar nenhum valor, pois, por definição, ele retorna implicitamente o próprio objeto que está sendo instanciado.

Caso não seja definido um método construtor, todos os atributos do objeto receberão o valor **NULL**. Nesse caso, não criamos o método construtor, portanto ao instanciar, os atributos ficarão **NULL**.

Encapsulamento

Encapsulamento é um dos conceitos fundamentais do paradigma de orientação a objetos, e no PHP ele também desempenha um papel crucial. Para garantir que esse conceito não seja esquecido, vamos revisá-lo.

Encapsulamento consiste em proteger o acesso aos atributos e métodos internos de uma classe ou objeto, garantindo que essas informações sejam ocultadas e controladas adequadamente. Essa proteção é importante para manter a integridade dos dados e a segurança do sistema.

Uma analogia interessante encontrada no fórum do iMasters (forum.iMasters.com.br) ajuda a ilustrar o conceito de encapsulamento:

“Fulano: Destranque a porta, abra-a e vá para o outro lado.

Quando Fulano for destrancar a porta, ele simplesmente colocará a chave correta no orifício, girará, e a porta será destrancada.

Fulano não precisa saber como o mecanismo da fechadura funciona, ele apenas executa a ação. Na verdade, ele não quer sequer saber que dentro da fechadura existe um mecanismo que só funcionará se a chave possuir o segredo correto.

Na orientação a objetos, podemos dizer que a fechadura é um objeto com uma operação chamada destrancar, que aceita apenas um objeto do tipo chave com o segredo correto. A lógica do mecanismo da tranca é



Imagem 2 – freepik.com

ocultada; nem Fulano, nem a chave precisam saber como o mecanismo interno funciona. Essa ocultação de informação é chamada de encapsulamento.”

Para implementar o encapsulamento em PHP, é necessário entender os modificadores de visibilidade, que controlam o acesso a atributos e métodos:

- ✓ **public:** sem qualquer restrição de acesso, todos os atributos e/ou métodos declarados com essa visibilidade podem ser acessados por qualquer parte do código, incluindo outras classes e objetos. Geralmente, essa visibilidade é usada quando queremos ou precisamos que esses atributos ou métodos sejam manipulados diretamente através do objeto instanciado.
- ✓ **private:** limita o acesso de atributos e métodos à própria classe. Somente a classe em que esses atributos ou métodos estão declarados pode acessá-los diretamente. Para acessar atributos private, é necessário utilizar métodos públicos, como os conhecidos getters e setters.

Para esclarecer melhor, veja a codificação a seguir:

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

Observe que no código apresentado o atributo **nome** está com o modificador **private**, ou seja, não é possível acessá-lo de fora da classe. Então, se tentarmos executar o seguinte código no index.

```
<?php
    include_once 'Pessoa.php';

    $p = new Pessoa();
    $p->nome = "Zeca"; echo $p->nome;
?>
```

Ocasionará um erro, como o apresentado na imagem a seguir:

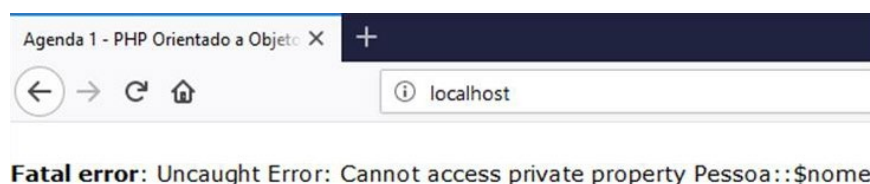


Imagem 3 – Erro ao tentar acessar o atributo nome com o modificado private.

Para solucionar o erro apresentado, é necessário criar os famosos métodos getters e setters. Esses métodos permitem acessar e modificar os atributos privados por meio de métodos públicos, garantindo que a alteração dos atributos seja feita conforme o desejado. O código da classe deve ficar da seguinte forma:

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

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

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



Utilizamos **\$this->nome** para referenciar o atributo da classe nome, pois no método **setNome** existe um parâmetro local também chamado **\$nome**. A palavra-chave **\$this** é usada para distinguir o atributo da classe do parâmetro do método, garantindo que estamos atribuindo o valor ao atributo correto.

O **index.php** deve ficar da seguinte forma:

```
<body>
<?php
include_once 'Pessoa.php';

$p = new Pessoa();
$p->setNome("Zeca");
echo $p->getNome();
?>
</body>
```

Resultado no navegador:

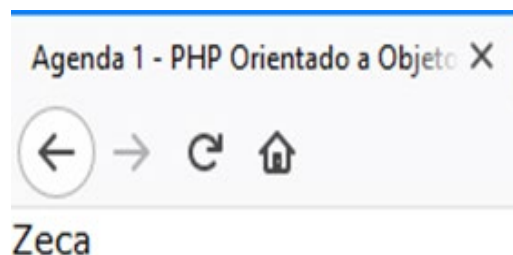


Imagem 4 – Exemplo de funcionamento do Index utilizando encapsulamento de atributos.

Herança

A herança é, sem dúvida, uma das principais características da Orientação a Objetos. Ela permite a implementação de uma estrutura hierárquica de classes, possibilitando a criação de classes de uso geral, que contêm características comuns a várias entidades relacionadas. Essas classes gerais podem ser estendidas por outras classes, resultando em subclasses mais especializadas, com implementações específicas.

No PHP, a herança funciona da mesma forma. Qualquer classe pode ser herdada, e isso é feito declarando uma nova classe que será uma extensão de outra, utilizando a palavra-chave **extends**. Isso permite que a

nova classe herde atributos e métodos da classe pai, além de adicionar ou sobrescrever funcionalidades conforme necessário.

O comando **protected**: A visibilidade protected é intermediária entre **public** e **private**. Para entender melhor, é necessário utilizar herança. Resumidamente, quando um atributo ou método é declarado como protected, ele pode ser acessado pelas classes herdeiras como se fosse public. Isso significa que as subclasses podem utilizar e modificar essas propriedades ou métodos, mas eles ainda permanecem inacessíveis para o mundo externo, fora da hierarquia de herança.

Vamos entender melhor programando, para isso utilizaremos a mesma classe pessoa do último exemplo de encapsulamento, mas com apenas uma alteração no atributo nome, que estava com o modificador private colocando protected, como podemos observar a seguir:

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

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

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

Agora, criamos duas outras classes Física e Jurídica: cada uma com um atributo e métodos específicos.

Classe Física:

```
<?php
require_once 'Pessoa.php';
class Fisica extends Pessoa
{
    private $cpf;

    public function setCpf($cpf)
    {
        $this->cpf = $cpf;
    }
    public function getCpf()
    {
        return $this->cpf;
    }
}
```

Classe Jurídica:

```
<?php
require_once 'Pessoa.php';
class Juridica extends Pessoa
{
    private $cnpj;

    public function setCnpj($cnpj)
    {
        $this->cnpj = $cnpj;
    }
    public function getCnpj()
    {
        return $this->cnpj;
    }
}
?>
```

Isso foi necessário, porque cada uma das classes está em um arquivo PHP distinto.

Agora, com as três classes prontas, podemos testá-las codificando o **index.php** da seguinte forma:

```
<?php

include_once 'Pessoa.php';
include_once 'Fisica.php';
include_once 'Juridica.php';

$pessoa = new Pessoa();
$pessoa->setNome("Zeca");
echo 'Nome: ' . $pessoa->getNome() . '<br>';

$fisica = new Fisica();
$fisica->setCpf("111111111");
echo 'CPF: ' . $fisica->getCpf() . '<br>';

$juridica = new Juridica();
$juridica->setCnpj("222222222");
echo 'CNPJ: ' . $juridica->getCnpj() . '<br>';

?>
```

No navegador, o resultado deve ser o seguinte:

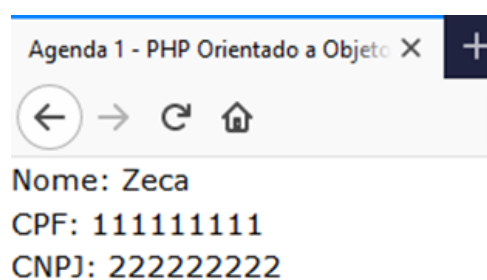


Imagem 5 – Exemplo de funcionamento do Index.

O resultado é simples, porém ainda não mostra o funcionamento da herança propriamente dita, para isso vamos alterar o index e ver o resultado no navegador.

Primeiro passo: remover o objeto “pessoa” do index.

Segundo Passo: atribuir nome nos objetos “fisica” e “juridica” respectivamente.

Terceiro passo: exibir no navegador.

O código do index.php deverá ficar da seguinte forma:

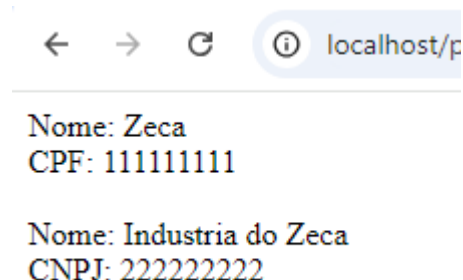
```
<?php

include_once 'Fisica.php';
include_once 'Juridica.php';

$fisica = new Fisica();
$fisica ->setNome("Zeca");
$fisica->setCpf("111111111");
echo 'Nome: ' . $fisica->getNome() . '<br>';
echo 'CPF: ' . $fisica->getCpf() . '<br>';

echo '<br>';

$juridica = new Juridica();
$juridica->setNome("Industria do Zeca");
$juridica->setCnpj("22222222");
echo 'Nome: ' . $juridica->getNome() . '<br>';
echo 'CNPJ: ' . $juridica->getCnpj() . '<br>';
?>
```



← → ↻ ⓘ localhost/p

Nome: Zeca
CPF: 111111111

Nome: Industria do Zeca
CNPJ: 22222222

Imagem 6 – Exemplo de funcionamento do Index.



Note que não foi necessário utilizar o include da classe **Pessoa**, isso ocorreu por que não instanciamos nenhum objeto de sua classe, utilizamos apenas as classes herdeiras **Fisica** e **Juridica**.

Neste código, podemos observar claramente o uso da herança. As classes filhas (**Física** e **Jurídica**) não possuem o método **setNome** nem o atributo **nome**. No entanto, como a classe pai, **Pessoa**, define tanto o método **setNome** quanto o atributo **nome**, as classes filhas os herdam automaticamente. Isso permite que as classes **Física** e **Jurídica** utilizem esses métodos e atributos sem a necessidade de reimplementá-los.

Mas vamos relembrar que programamos o atributo **nome** da classe **pessoa** com o modificador **protected**, mas até agora não vimos o que ele faz em PHP, então vamos lá!

Vamos alterar o código da classe **Fisica**, acrescentando a função **mudarNome()**:

```

<?php
require_once 'Pessoa.php';
class Fisica extends Pessoa
{
    private $cpf;

    public function setCpf($cpf)
    {
        $this->cpf = $cpf;
    }
    public function getcpf()
    {
        return $this->cpf;
    }

    public function mudarNome()
    {
        $this->nome = "Protegido";
    }
}
?>

```

Note que o valor é atribuído em nome da mesma forma que na classe pai. Isso porque foi utilizado o modificador **protected**, lembrando que eles deixam o atributo com acesso total para as classes filhas. Agora, vamos modificar o **index.php** para realizar o teste e observar a função do comando **protected** em herança.

```

<?php

include_once 'Fisica.php';
include_once 'Juridica.php';

$fisica = new Fisica();
$fisica ->setNome("Zeca");
$fisica->setCpf("111111111");
echo 'Nome: ' . $fisica->getNome() . '<br>';
echo 'CPF: ' . $fisica->getCpf() . '<br>';

$fisica->mudarNome();
echo 'Nome: ' . $fisica->getNome() . '<br>';
echo 'CPF: ' . $fisica->getCpf(); echo '<br>';

echo '<br>';

$juridica = new Juridica();
$juridica->setNome("Industria do Zeca");
$juridica->setCnpj("222222222");
echo 'Nome: ' . $juridica->getNome() . '<br>';
echo 'CNPJ: ' . $juridica->getCnpj() . '<br>';

```

```
?>
```

O resultado deve ser o seguinte:

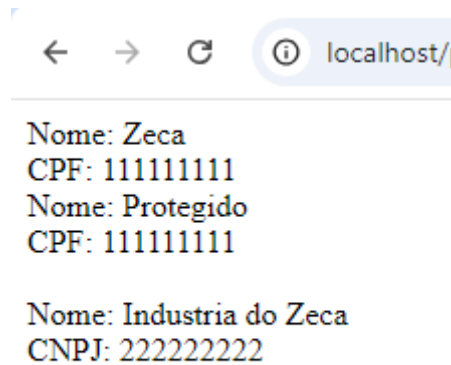
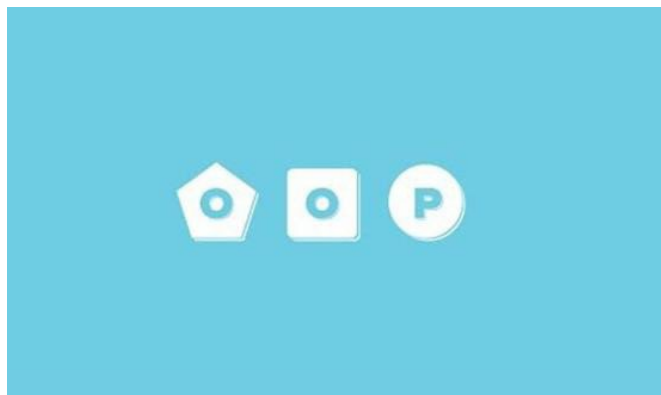


Imagem 7 – Exemplo de funcionamento do Index.

Entenda os princípios da Orientação a Objetos assistindo o seguinte vídeo:



Disponível em: https://www.youtube.com/watch?v=_mBqvoSJIBU. Acessado em 17/09/2024.



**VOCÊ
COMANDO**

Agora que já conhecemos os princípios da Orientação a Objetos em PHP, vamos colocá-los em prática com um projeto real.

Crie três classes em PHP e teste de acordo com o Diagrama de Classe a seguir:



Imagem 8 - Diagrama de classe sem tipos em atributos.

Não esqueça de criar os métodos getters e setters.

1. Para testar crie:
 - a. Uma página Index.php.
 - b. Um objeto Professor.
 - c. Um objeto Aluno.
 - d. Crie os respectivos atributos das classes Professor e Aluno.
 - e. Não esqueça de atribuir também nos atributos da superclasse.
 - f. Exiba os dados dos atributos de ambos objetos (professor e aluno) no Navegador.

Confira se você conseguiu resolver os desafios propostos!

Classe Pessoa.php:

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

    public function setNome($nome)
    {
        $this->nome = $nome;
    }
    public function getNome()
    {
        return $this->nome;
    }

    public function setCpf($cpf)
    {
        $this->cpf = $cpf;
    }
    public function getCpf()
    {
        return $this->cpf;
    }
}
?>
```

Classe Professor.php:

```
<?php
require_once 'Pessoa.php';
class Professor extends Pessoa
{
    private $formacao;

    public function setFormacao($formacao)
    {
        $this->formacao = $formacao;
    }
    public function getFormacao()
    {
        return $this->formacao;
    }
}
```

Classe Aluno.php:

```
<?php
require_once 'Pessoa.php';
class Aluno extends Pessoa
{
    private $curso;
    public function setCurso($curso)
    {
        $this->curso = $curso;
    }
    public function getCurso()
    {
        return $this->curso;
    }
}
```

Teste no arquivo Index.php:

```
<?php
include_once 'Aluno.php';
include_once 'Professor.php';

$a = new Aluno();
$a->setNome("José");
$a->setCpf("111.111.111.11");
$a->setCurso("Técnico em Desenvolvimento de Sistemas");
echo 'Nome: ' . $a->getNome() . '<br>';
echo 'CPF: ' . $a->getCpf() . '<br>';
echo 'Curso: ' . $a->getCurso() . '<br>';
echo '<br>';

$p = new Professor();
$p->setNome("Paulo");
$p->setCpf("222.222.222.22");
$p->setFormacao("Ciência da Computação");
echo 'Nome: ' . $p->getNome() . '<br>';
echo 'CPF: ' . $p->getCpf() . '<br>';
echo 'Formacao: ' . $p->getFormacao() . '<br>';
echo '<br>';
?>
```

Não esqueça de colocar a extensão .php em todos os arquivos criados nas atividades.