AGENDA 09

PHP - ORIENTADO A OBJETO

MOMENTO DE REFLEXÃO

Você já parou para pensar o quanto de código é necessário para o desenvolvimento de um projeto completo, e sempre que uma empresa for desenvolver um novo projeto ter que ser iniciado do zero, o quanto isso se torna inviável, e ainda, qual seria o custo para um novo projeto? Como podemos lidar e desenvolver um código que possa ser reutilizado e fornecer uma manutenção de forma mais fácil e adequada?

Nessa agenda, vamos estudar sobre o PHP orientado a objetos um paradigma de programação oferecendo resposta para as perguntas acima, e nos ofecerendo como resultado um código altamente reutilizável. Vamos lá?

POR QUE APRENDER?

O desenvolvimento de um site pode ser muito amplo, no ponto de vista programação. O mercado oferece diversas linguagens de programação, que seguem diferentes paradigmas. Um desses paradigmas é a Orientação a Objetos, que possívelmente hoje é o mais utilizado, principalmente pela vantagem do reaproveitamento de código e facilidade de implementação de futuras manutenções, o que é muito importante para qualquer tipo de desenvolvimento inclusive para o desenvolvimento WEB.

PARA COMEÇAR O ASSUNTO...

Veja o caso do Zeca! Ele já desenvolveu alguns sites como freelancer e, a cada novo projeto, ele começa a programar todo o sitepraticamente do zero; porém, percebendo que ele perdia muito tempo e ganhava pouco dinheiro com seus projetos, começou a estudar PHP orientado a Objetos.

E foi assim que o Zecaconseguiudesenvolver mais projetos em menos tempo. Sabe por quê? Ele aprendeu areaproveitar melhor os códigos de projetos já desenvolvidos anteriormente! Você sabe como ele faz isso? Mergulhe no tema dessa aula e descubra!

Imagem 1 – freepik.com

MERGULHANDO NO TEMA...

Como começa a programar Orientado a Objetos?

Claro que para começarmos a programar orientação a objetos é fundamental já ter absorvido e entendido os conceitos de classes e objetos, mas apenas para relembrar: classe é uma estrutura que define um tipo de dados, contendo ou não atributos (variáveis) e funções (métodos) para acessar e manipular esses atributos.

Criando a Classe

Vamos começar desenvolvendo uma classe, para isso, crie um arquivo PHP e salve-o com o nome de "Pessoa". Não se esqueça de criar os delimitadores do php.

Para criar uma classe em PHP, utilizamos a palavra reservada class, seguida do nome da classe a ser criada:

```
<?php
classPessoa{

    }
?>
```

Obs: Lembre-se, a primeira letra do nome da classe deve ser maiúscula, esta é uma boaprática do paradigma Orientado a Objeto e não da linguagemde programação, logo, sempre que for programar Orientado a Objeto essa boa prática deve ser seguida!

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

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

Obs: Repare que utilizamos uma palavra chave public que se refere à visibilidade, ou seja, quando instanciado o objeto, será possível o acesso a este atributo.

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

Instânciando o objeto da classe

O PHP originalmente não é uma linguagem que foi desenvolvida para ser orientada a objetos e geralmente programadores PHP utilizavama programação estruturada. Basicamente, criam arquivos PHP e organizam as funções correlacionadas e mais utilizadas, como exemplo, podemos utilizar o CRUD de Banco de Dados e,dentro do arquivo, colocar todas as funções para inserir, atualizar, deletar e pesquisar dados. Depois, basta apenas fazer a inclusão do mesmo nas páginas que serão necessárias à sua utilização.

Obs1: Essa inclusão poderia ser feita utilizando include, require, include_once ou require_once.

- 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.

Mas porque todas essas explicações? É simples! É uma boa prática, por exemplo, para ser possível a utilização da classe, permitindo assim a instanciação de um objeto a partir dela, no index de seu site precisamos fazer a inclusão deste arquivo.

Agora vamos instanciar um objeto da recém-criada classe “Pessoa”, para isso devemos criar um arquivo PHP e vamos denominá-lo “index”.

Obs: Não se esqueça: todos estes arquivos estão no diretório root, do servidor apache.

Agora vamos fazer a inclusão da classe que acabamos de desenvolver!

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

Esse código vai garantir que possamos instanciar o objeto sem problemas, agora basta programar o código a seguir:

```
        $p = newPessoa();
        $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 junto ao construtor da classe conseguimos criar a sua instância.

- Nas duas linhas seguintes atribuímos valores aos atributos. Note que para acessar os atributos é necessário usar “**->**”.

- Depois usamos o comando “echo” para exibir na página os conteúdos dos atributos, ficando o resultado como o apresentado a seguir:

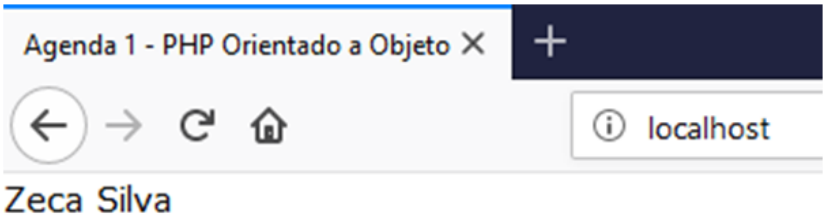

Imagem 2 – Resultado da codificação e uso da instâncias em nossa página html.

Obs: Apenas para não esquecer, o método construtor é utilizado para definir o comportamento, ou basicamente atribuir valores iniciais a um objeto. O método construtor é executado quando instanciamos um objeto por meio do operador “new” seguido do nome da classe a ter seu objeto instanciado. Assim, não devemos retornar nenhum valor por meio do método construtor porque, por definição, o método retorna 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

Imagem 3 – freepik.com

Encapsulamento é um dos recursos importantes que o paradigma orientado a objetos nos traz e no PHP ela também é muito importante, mas para não ficar esquecido vamos relembrar.

Basicamente é fornecido um meio de proteção de acesso aos atributos e métodos insternos da classe/objeto. Então, devemos entender que a ocultação/proteção dessas informações é muito importante e no fórum do iMasters tem uma analogia que demonstra de forma muito interessante a idéia de encapsulamento (forum.iMasters.com.br)[1].

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

Quando Fulano for destrancar a porta ele simplesmente enfiará a chave correta no orifício específico, girará e terá a porta destrancada.

Fulano não precisa saber como a porta será destrancada, na verdade, ele não quer sequer saber que dentro da fechadura possui todo um mecanismo que somente destrancará se a chave possuir o segredo correto. Portanto orientação a objetos, podemos chamar a fechadura de objeto que possui uma operação destrancar que somente aceita um objeto do tipo chave com o segredo correto, a lógica do mecanismo de tranca é oculto e nem o participante Fulano nem a participante chave precisam saber como o mecanismo funciona. Essa ocultação de informação chama-se encapsulamento"

Para conseguir encapsular atributos e métodos, é necessário lembrar das visibilidades:

- **public:** sem ocultação nenhuma, todos os atributos e/ou métodos declarados com essa visibilidade poderão ser acessados por todos que quiserem, geralmente é usado em operações onde queremos ou precisamos que sejam manipulados através do objeto instânciado.
- **private:** seu uso faz com que qualquer método ou atributo seja visível apenas pela classe e somente ela terá acesso direto. Os atributos do tipo private somente poderão ser acessados por meio de métodos públicos.

Exemplo: para esclarecer melhor, veja a codificação a seguir:

```
<?php
classPessoa{
private$nome;
    }
?>
```

Para entender este código, note que 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 = newPessoa();
        $p->nome = "Zeca";
        echo$p->nome;
?>
```

[1] *Forum Imasters – disponível em https://forum.imasters.com.br/topic/392880-13-visibilidade-e-encapsulamento/. Acesso em 31/09/2018.*

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

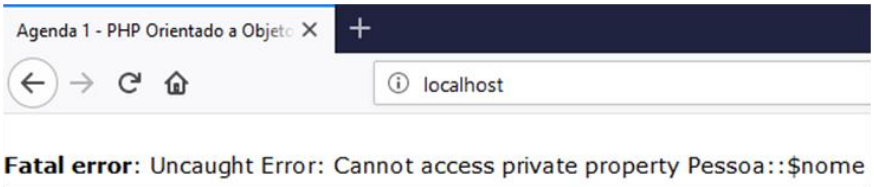

Imagem 4 – Erro ao tentar acessar o atributo nome com o modificado private.

Para solucionar esse erro, é preciso criar os famosos métodos getters e setters para acessar os atributos por meio de métodos públicos e para que a alteração do atributo seja realizada como desejamos. Para isto, o código da classe deve ficar desta forma:

```php
<?php
classPessoa{
private$nome;

publicfunctionsetNome($nome)
        {
$this->nome = $nome;
        }
publicfunctiongetNome()
{
return$this->nome;
        }
    }
?>
```

Obs.: Repare que utilizamos $this->nome para referênciar o atributo da classe $nome, já que em nosso método set existe um parâmetro local $nome.

E no index nosso código deverá ficar desta forma:

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

            $p = newPessoa();
            $p->setNome("Zeca");
            echo$p->getNome();
    ?>
```

Resultando no navegador:

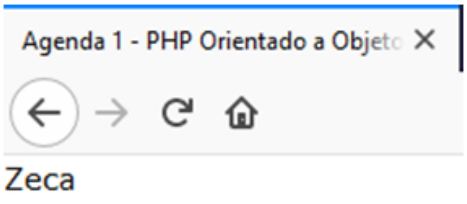

Imagem 5 – Exemplo de funcionamento do Index utilizando encapsulamento de atributos.

- **protected:** É uma intermediária entre as duas anteriores. Porém, precisamos de herança para exemplificar. Para resumir: a visibilidade protected faz com que todos os herdeiros vejam as propriedades ou métodos protegidos como se fossem públicos.

Herança

A herança representa, sem dúvida, uma das principais características da Orientação a Objetos, até porque é possível implementar dados de forma hierárquica, então, conseguimos implementar classes de uso geral, que possuam características comuns a várias entidades relacionadas. Essas classes poderão ser estendidas por outras, produzindo classes mais especializadas e que tenham implementações específicas.

Em PHP não é diferente, toda classe poderá ser herdada, seu uso se resume na declaração de uma nova classe que será uma extensão da outra, utilizando a palavra chave “extends”.

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
classPessoa{
protected$nome;

publicfunctionsetNome($nome)
        {
$this->nome = $nome;
        }
publicfunctiongetNome()
{
return$this->nome;
        }
    }
?>
```

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

Classe Física.

```
<?php
require_once'Pessoa.php';
classFisicaextendsPessoa {
private$cpf;

publicfunctionsetCpf($cpf)
        {
$this->cpf = $cpf;
        }
publicfunctiongetcpf()
{
return$this->cpf;
        }
    }
?>
```

Classe Jurídica

```
<?php
require_once'Pessoa.php';
classJuridicaextendsPessoa {
private$cnpj;

publicfunctionsetCnpj($cnpj)
        {
$this->cnpj = $cnpj;
        }
publicfunctiongetCnpj()
{
return$this->cnpj;
        }
    }
?>
```

Obs.: Note que foi necessário criar um require_once com a classe Pessoa que seria extendia pelas classes. Isso foi necessário, porque cada uma das classes está em um arquivo PHP distinto.

Agora, com as três classes prontas, podemos testá-las no index codificando da seguinte forma:

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

        $p = newPessoa();
        $p->setNome("Zeca");
        echo'Nome: '.$p->getNome();

        echo'<br>';

        $f = newFisica();
        $f->setCpf("111111111");
        echo'CPF: '.$f->getCpf();

        echo'<br>';

        $j = newJuridica();
        $j->setCnpj("222222222");
        echo'CNPJ: '.$j->getCnpj();
?>
```

Então obteremos o seguinte resultado no navegador.

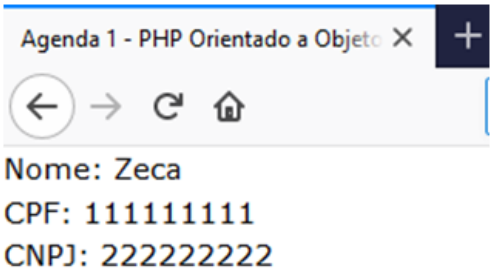

Imagem 6 – Exemplo de funcionamento do Index.

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 “p” do index.
Segundo Passo: atribuir nome nos objetos “f” e “j”. respectivamente.
Terceiro passo: exibir no navegador.

Obtendo, então, o seguinte código:

```php
<?php

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


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

        echo'<br>';

        $j = newJuridica();
        $j->setNome("Industria Zeca");
        $j->setCnpj("222222222");
        echo'Nome: '.$j->getNome().'<br>';
        echo'CNPJ: '.$j->getCnpj();

 ?>
```

Obs: Perceba que não foi mais necessário utilizar o include da classe pessoa, já que não instanciamos nenhum objeto de sua classe.

Nesse código já percebemos, de forma clara, o uso da herança, visto que as classes filhas (Física e jurídica) não possuem o método setNome e nem o atributo nome; porém, como a classe pai "Pessoa" possui este método (setNome) e atributo (nome), as classes filhas (Física e Jurídica) os recebem como herança, e podem fazer uso deles sem nenhum problema.

Veja o resultado a seguir.

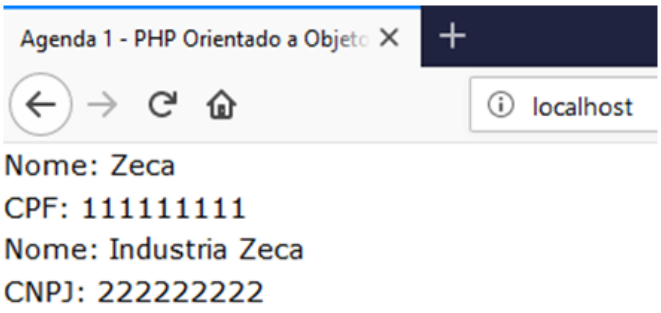

Imagem 7 – Exemplo de funcionamento do Index.

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!

Primeiro vamos alterar o código da nossa classe Pessoa, com o seguinte código:

```php
<?php
require_once'Pessoa.php';
classFisicaextendsPessoa {
private$cpf;

publicfunctionsetCpf($cpf)
        {
$this->cpf = $cpf;
        }
publicfunctiongetcpf()
        {
return$this->cpf;
        }

publicfunctionmudarNome()
{
$this->nome = "Protegido";
        }
    }
?>
```

Perceba que o valor é atribuído em nome da mesma forma que na classe pai. Isso porque foi utilizado o modificador protected, relembrando que eles deixam o atributo com acesso total para as classes filhas. Agora, vamos mudar o index para conseguir realizar o teste e perceber a função “protected” em herança.

```php
<?php
    include_once'Juridica.php';
        $f = newFisica();
        $f->setNome("Zeca");
        $f->setCpf("111111111");
        $f->mudarNome();
        echo'Nome: '.$f->getNome().'<br>';
        echo'CPF: '.$f->getCpf();
        echo'<br>';
?>
```

O resultado será o demonstrador a seguir.

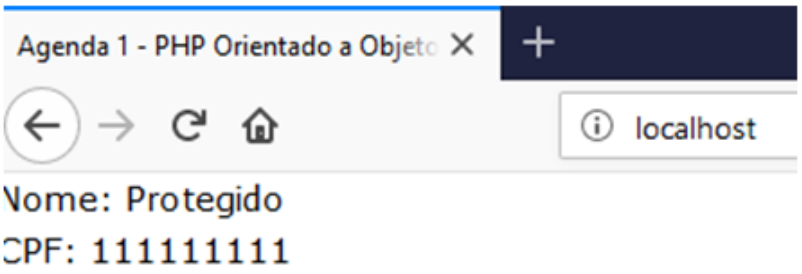

Imagem 8 – Exemplo de funcionamento do Index.

Com o resultado, percebemos que não foi alterado o nome para “protegido”, isso acontece, pelo fato do atributo agora ser privado e seu acesso deve ser realizado apenas pelos métodos públicos da classe Pessoa.

Obs.: Em PHP não é gerado erro pelas particularidades da sua concepção nos tipos de variáveis.

Alguma dúvida ainda?

Assista o Vídeo “entendo em 30 minutos um pouco melhor PHP OO”.

Imagem 09 - Aprenda PHP Orientado a Objetos em 30 minutos. (RBTECH, 2018).

Esse vídeo é para quem precisa aprender o essencial sobre php orientado a objetos. Seja para iniciar no mundo da programação ou simplesmente para complementar seus estudos sobreo assunto, esse vídeo é uma síntese sobre orientação a objetos com PHP. Disponível em: https://www.youtube.com/watch?v=_mBqvoSJIBU. Acessado em 10/09/2018.

VOCÊ NO COMANDO

Utilizando o que foi visto até agora....

1. Crie três classes em PHP e teste de acordo com o Diagrama de Classe a seguir. Obs: Não se esqueça de criar os métodos getters e setters.

2. Para testar crie:

a. Uma página Index.

b. Um objeto Professor.

c. Um objeto Aluno.

d. Crie os respectivos atributo das classes Professor e Aluno.

• Não esqueça de atribuir também nos atributos da superclasse.

e. Exiba os dados dos atributos de ambos objetos(professor e aluno) no Navegador.

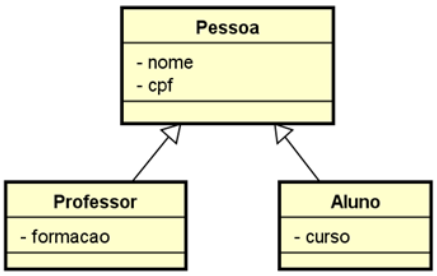

Imagem 10 - Diagrama de classa sem tipos em atributos.

Confira abaixo se você conseguiu resolver os desafios propostos!

1. Desenvolvendo as Classes

a. Pessoa.php

```php
<?php
classPessoa{
private$nome;
private$cpf;

publicfunctionsetNome($nome)
        {
$this->nome = $nome;
        }
publicfunctiongetNome()
        {
return$this->nome;
        }

publicfunctionsetCpf($cpf)
        {
$this->cpf = $cpf;
        }
publicfunctiongetCpf()
        {
return$this->cpf;
        }
    }
?>
```

b. Professor.php

```php
<?php
require_once'Pessoa.php';
classProfessorextendsPessoa{
private$formacao;

publicfunctionsetFormacao($formacao)
        {
$this->formacao = $formacao;
        }
publicfunctiongetFormacao()
        {
return$this->formacao;
        }

    }
?>
```

c. Aluno.php

```php
<?php
require_once'Pessoa.php';
classAlunoextendsPessoa{
private$curso;

publicfunctionsetCurso($curso)
        {
$this->curso = $curso;
        }
publicfunctiongetCurso()
        {
return$this->curso;
        }

    }
?>
```

2. Testando no Arquivo Index.

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

        $a = newAluno();
        $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 = newProfessor();
        $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>';
?>
```

ATIVIDADE ONLINE

Envio de Atividade

Agora o Zeca consegue desenvolver em PHP utilizando o paradigma de Programação Orientado a Objetos, com isso, foi aberto um leque de possibilidades e ele passou a receber diversar propostas de trabalho. Uma delas foi um contrato como freelancer para programação de um projeto hospitalar que deverá ser desenvolvido em PHP. Para conseguir entregar o projeto no prazo solicitado, Zeca precisará de sua ajuda!

Imagine que você ficou responsável por criar a classe Paciente, de acordo com o Diagrama de Classe a seguir, e por criar os métodos Getters e Setters.

Após a criação da classe e dos seus métodos, envie o arquivo para o seu professor-tutor por meio do AVA.

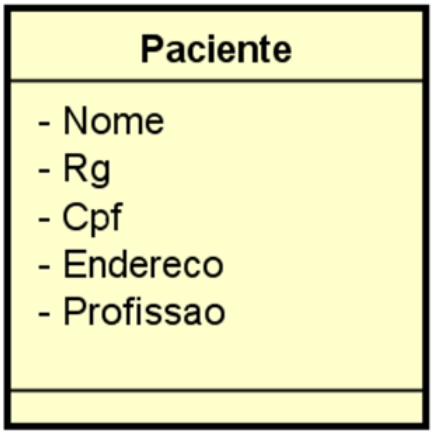

Imagem 11 - Classe Paciente.

AMPLIANDO HORIZONTES

Não deixe também de assistir aos vídeos:
Curso POO PHP #03b - Configurando Visibilidade de Atributos e Métodos (CURSO EM VIDEO, 2018).

Nessa aula de POO, vamos aprender na prática como utilizar os modificadores de visibilidade public, private e protected e qual é o efeito de cada um deles. Disponível em:
https://www.youtube.com/watch?v=48NaNTtcguA&list=PLHz_AreHm4dmGuLII3tsvryMMD7VgcT7x&index=7.
Acessado em 10/09/2018.

Links

DevMedia – Artigo Introdução à Orientação a Objetos em PHP. Acessado em 15/09/2018.
https://www.devmedia.com.br/introducao-a-orientacao-a-objetos-em-php/26762

Livros

• OGLIO, Pablo D. Php Programando Com Orientacao a Objetos. 4ª Edição. São Paulo. Novatec. 2018. ISBN: 978-8575226919