Aula 4º

Orientação a Objetos

Olá,

Iniciaremos a Aula 4 da matéria Desenvolvimento Voltado à Web I. Na última aula, o funcionamento da interação com o usuário via GET e POST foi explicado, permitindo que o lado do cliente interaja com o lado do servidor. Também foi demonstrada a utilização de funções e o contexto em que a variável é instanciada, ou seja, seu escopo. Por fim, foi apresentado o recurso de armazenamento no lado do usuário e no servidor, Cookies e Sessions respectivamente.

Na aula de hoje será exemplificada a utilização de Orientação a Objetos no PHP, demonstrando como criar classes e objetos e realizar operações sobre eles.

Leia atentamente esta aula e, se tiver alguma dúvida, use os recursos que estão na sua área do aluno.

Bons estudos!

Objetivos de aprendizagem

Ao término desta aula, vocês serão capazes de:

- criar classes e objetos no PHP.

Seções de estudo

1 – Classes e Objetos
2 – Associação, Agregação e Composição
3 – Herança

1- Classes e Objetos

A orientação a objetos é um paradigma de programação muito utilizado no desenvolvimento de sistemas, além de permitir um maior reaproveitamento de código. Quando bem utilizada, também é possível enxugar a arquitetura (SILVA, 2014).

W3Schools (2019) relata que a programação orientada a objetos tem várias vantagens quando comparada com a programação procedural:

1. Mais fácil e rápida de executar;
2. Proporciona uma estrutura mais limpa para os programas;
3. A manutenção, modificação e debug ficam mais fáceis;
4. Aplicações reutilizáveis e menor tempo de desenvolvimento.

As classes e objetos são os aspectos principais da orientação a objetos. Enquanto as classes reúnem características que são comuns a um determinado elemento, os objetos podem ser instanciados a partir delas (SILVA, 2014).

Figura 1 – Classes e Objetos. Fonte: W3SCHOOLS (2019).

Na Figura 1, podemos observar essa distinção. A classe fruta pode conter informações que são comuns a todas as frutas e que são instanciadas independentemente nos objetos: maça, banana e manga.

Como já foi mencionado, a reutilização é uma grande característica da orientação a objetos, então nos exemplos que seguem iremos criar as classes em arquivos separados, que serão incluídos ou requeridos no código principal. Isso facilita também para que grupos de desenvolvedores disponibilizem esses códigos (bibliotecas) para que possam ser utilizados por mais pessoas (SILVA, 2014).

Seguindo o exemplo das aulas anteriores, vamos criar uma classe para armazenar uma pessoa, seu IMC, sua classificação e seu grau de obesidade.

Vamos criar, então, um novo arquivo, que chamaremos de "imc.class.php". Dentro dele faremos a implementação da classe IMC que possui cinco atributos nome, altura, peso, classificacao e grau. Todos esses são privados, sendo acessados somente de dentro da própria classe. São criados também os métodos: setNome, setAltura, setPeso, getNome, getAltura, getPeso, getClassificacao, getGrau e setIMC, todos métodos públicos que podem ser acessados por qualquer outro arquivo.

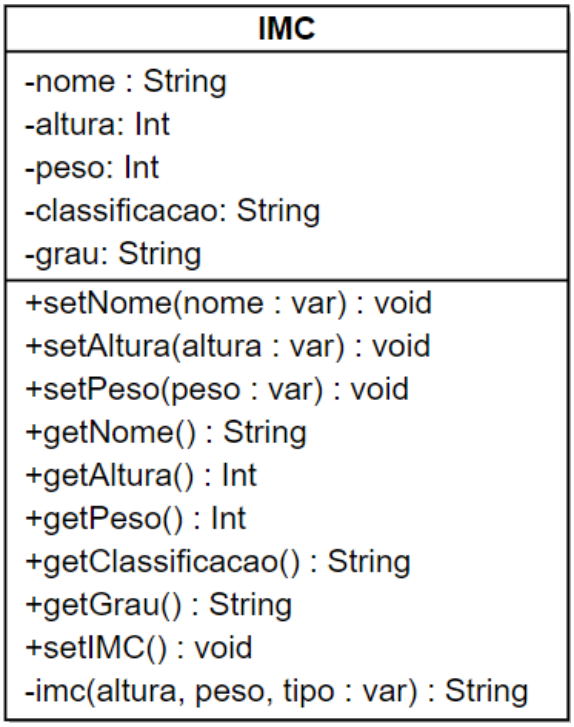

Figura 2 – Classe IMC. Fonte: Acervo pessoal.

Para criar uma classe, utilizamos a palavra reservada class, seguida do nome da classe e de colchetes. Dentro destes é que todos os atributos e métodos serão criados. As palavras reservadas private e public são utilizadas para métodos privados e públicos, respectivamente. Sempre que for preciso fazer referência a um atributo da classe, deve-se utilizar a palavra reservada $this, caso contrário, o PHP entende que é uma variável local e não o atributo.

```
<?php
   //arquivo: imc.class.php
   class Imc{
         private $nome;
         private $altura;
         private $peso;
         private $classificacao;
         private $grau;
         public                function
setNome($nome){
               $this->nome = $nome;
         }
   public function setAltura($altura){
               $this->altura        =
$altura;
         }
   public function setPeso($peso){
               $this->peso = $peso;
```

```
        }
    public function getNome(){
            return $this->nome;
        }
    public function getAltura(){
            return $this->altura;
        }
    public function getPeso(){
            return $this->peso;
        }
    public function getClassificacao(){
            return          $this-
>classificacao;
        }
    public function getGrau(){
            return $this->grau;
        }
    }
    ?>
```

Até agora foram criados os gets e sets para acessar indiretamente que é uma característica da orientação a objetos, o encapsulamento que mantém os atributos 'escondidos', só permitindo que os atributos sejam acessados pelos métodos da classe.

Além das visibilidades para métodos e atributos já explicadas public e private, existe o protected (protegido), que somente a própria classe e suas herdeiras podem acessar as informações.

Existem ainda dois métodos padrão a todas as classes o __construct (construtor) e o __destruct (destrutor). O primeiro é chamado assim que o objeto é instanciado, utilizando a palavra reservada new, enquanto o segundo é chamado para finalização da classe, quando a execução da página é encerrada.

Para o nosso exemplo, a implementação da classe setIMC depende da instanciação da classe pelo construtor, já que no construtor ele zera os valores de peso e altura, que são comparados dentro de setIMC, para verificar se o usuário já entrou com os dados. Além disso, podemos aproveitar a função imc(), criada na aula 2, para colocá-la dentro da nossa classe.

```
    <?php
    public function __construct(){
            $this->peso=0;
            $this->altura=0;
       }
    public function setIMC(){
       if($this->$pesso!=0&&$this-
>$altura!=0){
            $this->imc    =    imc($this-
>$altura, $this->$peso,0);
    $this->classificacao    =    imc($this-
>$altura, $this->$peso,1);
    $this->grau    =    imc($this->$altura,
$this->$peso,2);
    }
       }
       private                    function
imc($altura,$peso,$tipo=0){
            $imc=$peso/$altura*$altura;
    if($imc<18.5){
                $class = "MAGREZA";
    $grau = "0";
    }elseif($imc 24.9){
    $class = "NORMAL";
    $grau = "0";
    }elseif($imc <29.9){
    $class = "SOBREPESO";
    $grau = "I";
    }elseif($imc <39.9){
    $class = "OBESIDADE";
    $grau = "II";
    }else{
    $class = "OBESIDADE GRAVE";
    $grau = "III";
    }

    if($tipo===0){
                return $imc;
          }elseif($tipo===1){
                return $classificacao;
    }elseif($tipo===2){
       return $grau;
    }
    }

    ?>
```

Por fim, instanciamos e utilizamos a nossa classe em outro arquivo, o "index.php". Depois de instanciado o objeto com o new IMC(), são setados, nome, peso e altura. Por fim, a função setIMC é chamada para preencher os atributos que faltam, podendo, então, após isso, utilizar os gets para retornar a classificação e o grau de obesidade do objeto.

```
    <?php
       //arquivo: index.php
       include_once('imc.class.php');
       $obj = new IMC();

       $obj->setNome("Felipe");
    $obj->setPeso(85);
    $obj->setAltura(1.75);

    $obj->setIMC();

    echo $obj->getNome()." é classificado
como:   ".$obj->getClassificacao."   tem
obesidade grau ".$obj->getGrau;
    ?>
```

2- Associação, Agregação e Composição

Associação, agregação e composição são relacionamentos que podem ser feitos entre classes. Essa capacidade de relacionamento é um dos pontos fortes da orientação a objetos. Os três citados podem ter descrição similar no PHP, porém com significados bem distintos.

Subseção A – Associação

A associação é um relacionamento entre classes, porém, é uma forma mais simples de relacionamento. Isso se dá quando o relacionamento entre as classes não é forte, ou seja, uma classe não perde o sentido se a outra deixar de existir.

Silva (2014) exemplifica citando o relacionamento entre o carro e o motorista: um carro é dirigido por um motorista e um motorista dirige um carro, porém, se o carro deixar de existir, o motorista não perde seu sentido, ele pode passar a dirigir outro tipo de veículo.

Na Figura 3, podemos ver uma associação entre as classes carro e a motorista representada por uma linha contínua que pode ser acompanhada por uma seta que indica o sentido do relacionamento. Nesse caso, o carro possui um motorista.

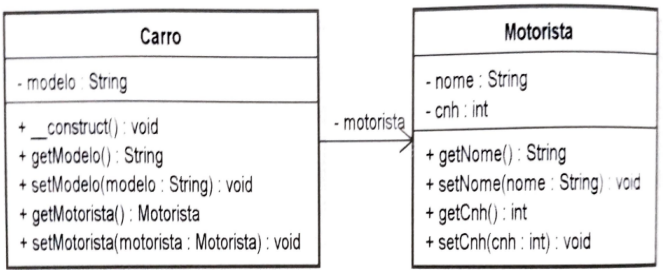

Figura 3 – Classes Carro e Motorista. Fonte: SILVA (2014).

Subseção B – Agregação

Na agregação, temos um relacionamento no qual uma classe faz parte da outra classe de alguma maneira. Continuando no exemplo do carro, as rodas fazem parte do carro, não podendo então ser uma associação, pois se as rodas deixarem de existir, o carro não fará mais sentido para seu funcionamento.

Na Figura 4, podemos ver uma agregação entre as classes carro e roda com um fator de multiplicidade (4 em uso + 1 estepe), representada por uma linha contínua acompanhada de um losango vazado junto à classe que compõe o todo. A classe roda é parte de um todo, classe carro (SILVA, 2014).

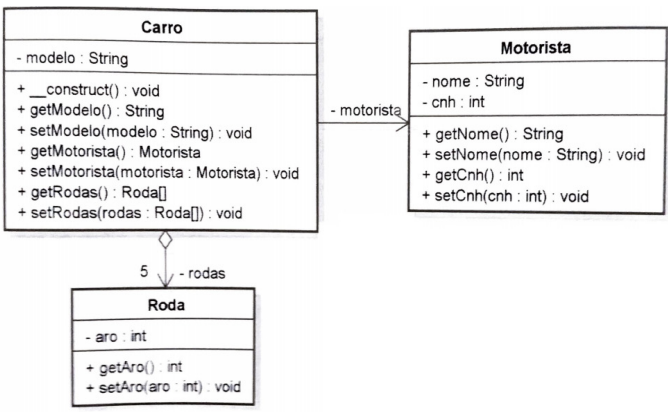

Figura 4 – Classes Carro, Motorista e Roda. Fonte: SILVA (2014).

Subseção C – Composição

Por fim, na composição temos o relacionamento mais forte que a agregação. Enquanto em um carro as rodas podem ser substituídas de alguma maneira, o chassi do carro é único e não é passível de substituição (dentro da legislação vigente). A representação da composição é similar à agregação, com a diferença de que o losango é preenchido (SILVA, 2014).

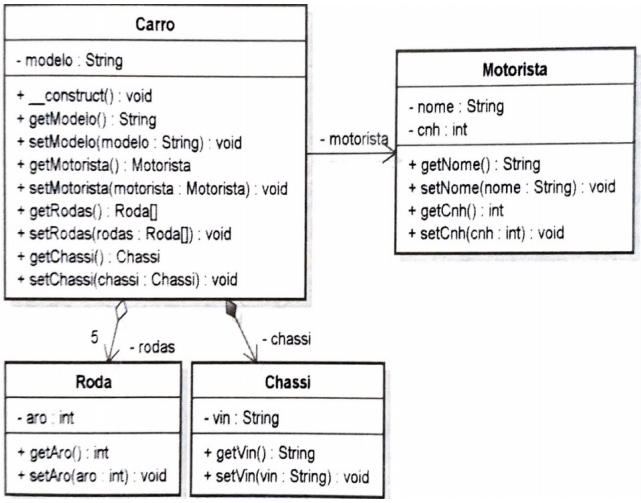

Figura 5 – Classes Carro, Motorista, Roda e Chassi. Fonte: SILVA (2014).

O código abaixo foi retirado de Silva (2014). Nele foram implementadas as quatro classes (Carro, Motorista, Roda e Chassi). Quando um relacionamento é exigido pode-se gerar um atributo que representará esse relacionamento em uma das classes, ou em ambas.

```
<?php
   //motorista.class.php
class Motorista{
   private $nome;
   private $cnh;
   public function getNome(){
         return $this->nome;
   }
   public function setNome($nome){
         $this->nome = $nome;
}
public function getCnh(){
   return $this->cnh;
}
public function setCnh($cnh){
   $this->cnh = $cnh;
}
}
?>

<?php
//roda.class.php
class Roda{
   private $aro;
   public function getAro(){
         return $this->aro;
   }
   public function setAro($aro){
         $this->aro = $aro;
                        }
                  }
                  ?>
<?php
```

```
    //chassi.class.php
    class Chassi{
       private $vin;
       public function getVin(){
             return $this->vin;
       }
       public function setVin($vin){
             $this->vin = $vin;
    }
    }
    ?>

    <?php
    //carro.class.php
    include_once('motorista.class.
php');
    include_once('roda.class.php');
    include_once('chassi.class.php');
    class Carro{
       private $modelo;
       private $motorista;
       private $rodas; //array
       private $chassi;
    public function __construct(){
       $this->motorista = new Motorista();
       $this->roda = array();
       $this->chassi = new Chassi();
    }
       public function getModelo(){
             return $this->modelo;
       }
       public function setModelo($modelo)
{
             $this->modelo = $modelo;
    }
    public function getMotorista(){
             return $this->motorista;
       }
       public                          function
setMotorista($motorista){
             $this->motorista                =
$motorista;
    }
    public function getRodas(){
             return $this->rodas;
       }
       public function setRodas($rodas){
             $this->rodas = $rodas;
    }
    public function getChassi(){
             return $this->chassi;
       }
       public function setChassi($chassi)
{
             $this->chassi = $chassi;
    }

    }
    ?>
```

3- Herança

Assim como na genética, em que os filhos herdam as características dos pais, na programação orientada a objetos não é diferente. Utilizamos herança quando queremos criar uma classe que deve herdar os métodos e atributos de uma outra classe. Chamaremos, então, as classes mãe de Superclasses e as classes filha de Subclasses (TEO, 2019)

Podemos utilizar, como exemplo, a superclasse veículo, Figura 6, que pode ser criada de maneira genérica, e ser herdada por duas subclasses diferentes, caminhão e carro. Ambas as subclasses possuem características em comum: herdando de veículo é possível evitar a reescrita de várias linhas de código.

Silva (2014) define ainda que a classe pai descreve um objeto de maneira generalizada (com os métodos e atributos que são semelhantes às subclasses), enquanto as classes filhas são mais específicas (com métodos e atributos únicos às subclasses).

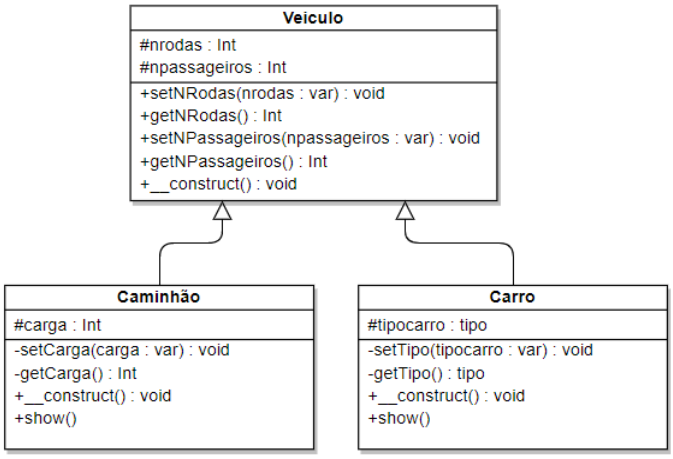

Figura 6 – Classes Veículo, Caminhão e Carro. Fonte: Acervo pessoal.

A representação da herança na Figura 6 é uma linha contínua com um triângulo vazado apontado para a superclasse que se quer herdar os atributos e métodos. Além disso, a superclasse veículo possui atributos que são protegidos, representados pelo símbolo #, e poderá ser acessado pelas classes caminhão e carro. Para implementar as classes que herdarão, utilizaremos a palavra reservada extends, que, traduzindo, diz que a classe caminhão estende a classe veículo.

```
<?php
//veiculo.class.php
class Veiculo{
   protected $nrodas;
   protected $npassageiros;
public function __construct(){

}
   public function getNRodas(){
         return $this->nrodas;
   }
   public function setNRodas($nrodas)
```

```
{
            $this->nrodas = $nrodas;
    }
    public function getNPassageiros(){
            return $this->npassageiros;
       }
       public                    function
setNPassageiros($npassageiros){
            $this->npassageiros        =
$npassageiros;
    }

    ?>

    <?php
    //carro.class.php
    include_once('veiculo.class.php');

    class Carro extends Veiculo{
       protected $tipocarro;
    public function __construct(){

    }
       private                    function
setTipo($tipocarro){
            $this->tipocarro           =
$tipocarro;
       }
    private function getTipo(){
            return $this->tipocarro;
       }
       public function show(){
    }
    }
                ?>
    <?php
    //caminhao.class.php
    include_once('veiculo.class.php');

    class Caminhao extends Veiculo{
       protected $carga;
    public function __construct(){

    }
       private function setCarga($carga)
{
            $this->carga = $carga;
       }
    private function getCarga(){
            return $this->carga;
       }
       public function show(){
    }
    }
                ?>
```

Classe Abstrata

Uma classe pode se chamar abstrata quando ela não puder ser instanciada. Isso significa que não se pode criar um objeto dessa classe. Quando um método é declarado como abstrato, o funcionamento dele não é implementado, então a classe a que esse método possui também deve ser abstrata.

Quando alguma subclasse herda de uma classe abstrata, ela precisa implementar os métodos abstratos que foram declarados na superclasse.

Silva (2014) apresenta um exemplo de uma herança da classe bebida que é abstrata e estende para duas classes refrigerante e suco. A classe bebida define um método calculaCaloria() abstrato e fica por responsabilidade das classes que o herdam de implementar a funcionalidade desse método.

```
    <?php
    //bebida.class.php
    abstract class Bebida {
       protected $medida;
    public function __construct($medida)
{

$this->medida=$medida;
    }
       abstract      public      function
calculaCaloria();
    }
                ?>

    <?php
    //refrigerante.class.php
    include_once('bebida.class.php');
    class Refrigerante extends Bebida {
       public function calculaCaloria(){
            if($this->medida==300){
                return 200;
            }elseif($this->medida==500)
{
                return 350;
            }
       }
    }
                ?>

    <?php
    //suco.class.php
    include_once('bebida.class.php');
    class Suco extends Bebida {
       public function calculaCaloria(){
            if($this->medida==300){
                return 150;
            }elseif($this->medida==500)
{
                return 250;
            }
       }
    }
                ?>
```

Sobrescrita

Característica das subclasses, a sobrescrita é a possibilidade das classes filhas substituírem o funcionamento de um método herdado da superclasse. Para fazer isso, basta a subclasse criar um método com o mesmo nome da superclasse e o funcionamento do método será reescrito.

Retomando a aula

Chegamos ao final da quarta aula. Vamos recordar?

1 – Classes e Objetos

Na primeira seção, foi possível entender a diferença entre classes e objetos e iniciar os estudos na programação orientada a objetos no PHP.

2 – Associação, Agregação e Composição

Na seção seguinte foram apresentadas algumas maneiras de se tratar o relacionamento entre classes, através da associação que uma classe está vinculada a outra, mas sem relação forte, da agregação que uma classe é parte da outra e da composição, que além de uma classe ser parte da outra não pode ser substituída.

3 – Herança

Por fim, os conceitos de herança foram exemplificados, detalhando como uma classe herda os atributos e métodos de outra para simplificar a escrita de código na hora do desenvolvimento.

Vale a pena

Vale a pena **ler**

SOARES, W. *Programação Web com PHP 5,* Érica, 2014.

BEIGHLEY, L; MORRISON, M. *Use a Cabeça!* PHP & MySQL. Alta Books, 2010.

SILVA, J. M. C. *PHP na prática 200 Exercícios resolvidos.* Elsevier, 2014.

BENTO, E. J. *Desenvolvimento web com PHP e MySQL.* Editora Casa do Código, 2014.

Vale a pena **acessar**

PHP. In: PHP Documentation, 2019. Disponível em: https://www.php.net/docs.php. Acesso em: 20 out. 2019.

W3SCHOOLS. In: PHP - What is OOP?, 2019. Disponível em: https://www.w3schools.com/php/php_oop_what_is.asp. Acesso em: 23 out. 2019.

TEO. In: HERANÇA E POLIMORFISMO EM PHP, 2019. Disponível em: https://www.todoespacoonline.com/w/2014/08/heranca-e-polimorfismo-em-php-orientado-objetos/. Acesso em: 23 out. 2019.

Minhas anotações