



BACK-END

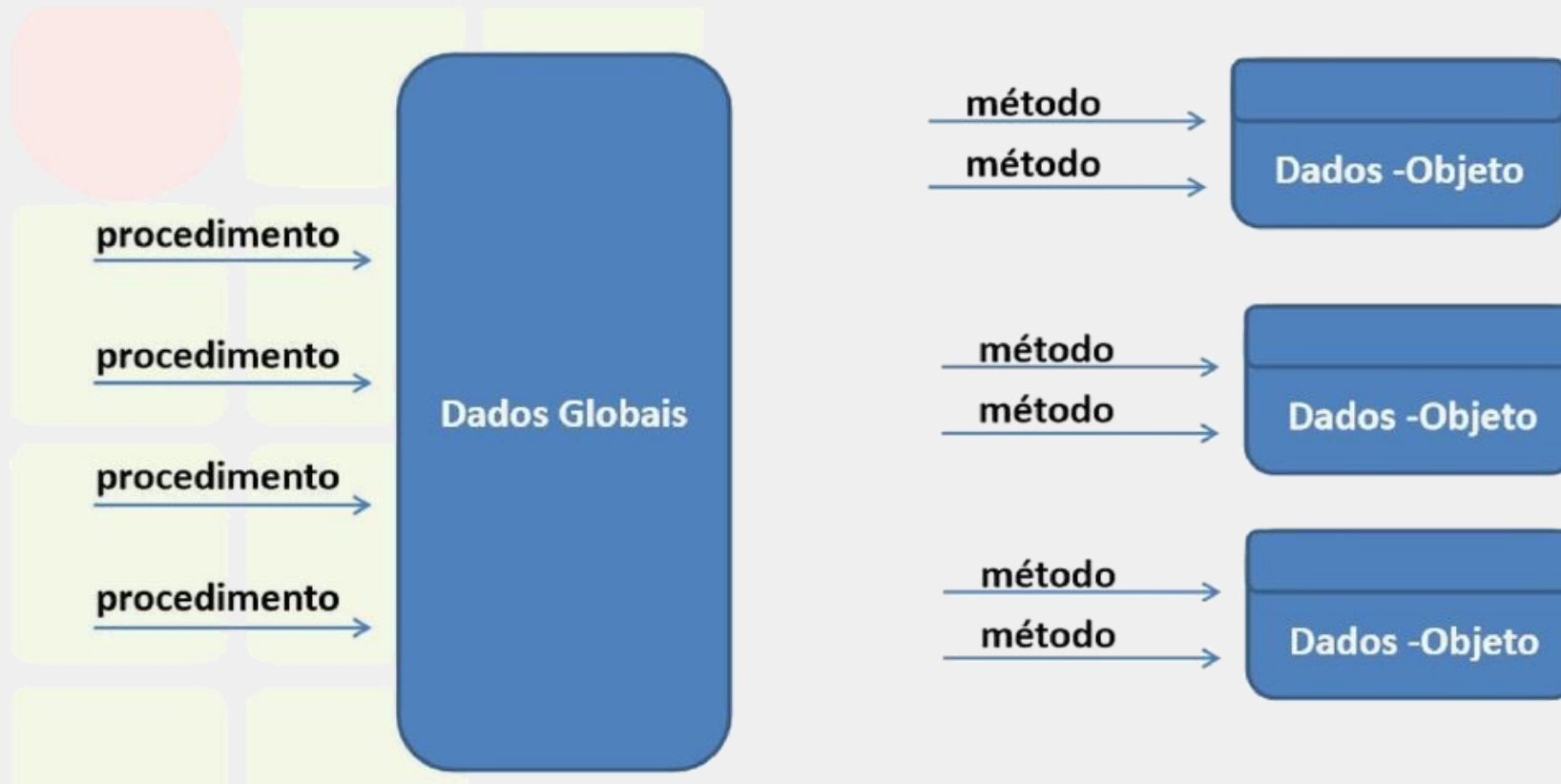
Desenvolvimento de Software para Internet

ORIENTAÇÃO A OBJETOS COM PHP



Paradigmas de programação

Estrutural x Orientação a Objeto





Principais diferenças

- **Programação estruturada ou procedural:**
 - Programa é tipicamente escrito em uma única rotina (ou função) podendo ser quebrado em subrotinas.
 - Variáveis não possuem muitas restrições de acesso

Exemplo de função: parâmetros por valor



```
<?php
function calcular_pontuacao ($resultado,$pontuacao)
{
    if ($resultado == 'venceu')
        $pontuacao += 3;
    elseif ($resultado == 'empatou')
        $pontuacao +=1;

    return $pontuacao;
}
$atual_pontuacao = 51;
echo "Qual foi o resultado do jogo? ";
$resultado = readline('venceu | empatou | perdeu ');
$nova_pontuacao = calcular_pontuacao($resultado,$atual_pontuacao);
echo "A nova pontuação é: ".$nova_pontuacao;
?>
```

resultado

venceu

pontuacao

51

atual_pontuacao

51

resultado

venceu

Exemplo de função: parâmetros por referência



```
<?php
function calcular_pontuacao ($resultado,&$pontuacao)
{
    if ($resultado == 'venceu')
        $pontuacao += 3;
    elseif ($resultado == 'empatou')
        $pontuacao +=1;
}
$pontos = 51;
echo "Qual foi o resultado do jogo? ";
$resultado = readline('venceu | empatou | perdeu ');
calcular_pontuacao($resultado,$pontos);
echo "A nova pontuação é: ".$pontos;
?>
```

resultado

empatou

pontuacao



pontos

51

resultado

empatou

6



Principais diferenças [2]

- **Programação estruturada ou procedural:**
 - Programa é tipicamente escrito em uma única rotina (ou função) podendo ser quebrado em subrotinas.
 - Variáveis não possuem muitas restrições de acesso
- **Programação orientada a objetos:**
 - Aproxima o manuseio das estruturas de um programa do mundo real
 - Conceitos chave: classes e objetos | +Proteção dos dados





Conceitos básicos de orientação a objetos

- **Classes**
- **Objetos**
- Encapsulamento (controles ou níveis de acesso)
- Herança
- Polimorfismo



Canal Dobra de Bits no YouTube

Programação Orientada a Objetos | Explicação Simples

Link: https://www.youtube.com/watch?v=pbb0jzXt_xA





Nelson



Jeremias





Características dos cachorrinhos



Crédito: [Dobra de Bits](#)



nome

=

"Nelson"

comida

=

3

sono

=

Falso



nome =

"Jeremias"

comida =

1

sono =

Verdadeiro



DORMIR()



sono = False





COMER()



comida -= 1





PHP sem OO: Funções

```
function comer($comida)
{
    $comida = $comida - 1;
    return $comida;
}
```

```
function dormir()
{
    $sono = False;
    return $sono;
}
```



PHP sem OO: Criação dos Cachorros

```
<?php
$nome_cachorro1 = "Nelson";
$comida_cachorro1 = 3;
$sono_cachorro1 = False;

$nome_cachorro2 = "Jeremias";
$comida_cachorro2 = 1;
$sono_cachorro2 = True;
?>
```



PHP sem OO: Colocando o Jeremias para dormir



```
<?php
```

```
$sono_cachorro2 = dormir();
```

```
?>
```



PHP sem OO: Colocando o Nelson para comer



```
<?php
```

```
$comida_cachorro1 = comer($comida_cachorro1);
```

```
?>
```





PHP sem OO: Exemplo completo

```
function comer($comida)
{
    $comida = $comida - 1;
    return $comida;
}
function dormir()
{
    $sono = False;
    return $sono;
}
```

```
$nome_cachorro1 = "Nelson";
$comida_cachorro1 = 3;
$sono_cachorro1 = False;
$nome_cachorro2 = "Jeremias";
$comida_cachorro2 = 1;
$sono_cachorro2 = True;

//Colocando o Nelson para comer
$comida_cachorro1 = comer($comida_cachorro1);

//Colocando o Jeremias para dormir
$sono_cachorro2 = dormir();
```



CLASSE

CACHORRO





CACHORRO

nome
comida
sono

comer()
dormir()



PHP com OO: Classe cachorro



```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    public function __construct($nome,$comida,$sono)
    {
        $this->nome = $nome;
        $this->comida = $comida;
        $this->sono = $sono;
    }
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}
```

```
$cachorro1 = new Cachorro("Nelson",3,False);
$cachorro2 = new Cachorro("Jeremias",1,True);
$cachorro1->comer();
$cachorro2->dormir();
```

```
var_dump($cachorro1);
var_dump($cachorro2);
```

```
object(Cachorro)#1 (3) {
    ["nome":"Cachorro":private]=>
    string(6) "Nelson"
    ["comida":"Cachorro":private]=>
    int(2)
    ["sono":"Cachorro":private]=>
    bool(false)
}
object(Cachorro)#2 (3) {
    ["nome":"Cachorro":private]=>
    string(8) "Jeremias"
    ["comida":"Cachorro":private]=>
    int(1)
    ["sono":"Cachorro":private]=>
    bool(false)
}
```





Variáveis que referenciam objetos e suas propriedades

```
$cachorro1 = new Cachorro("Nelson",3,False);  
$cachorro2 = new Cachorro("Jeremias",1,True);
```

ao invés de ...

```
$nome_cachorro1 = "Nelson";  
$comida_cachorro1 = 3;  
$sono_cachorro1 = False;  
$nome_cachorro2 = "Jeremias";  
$comida_cachorro2 = 1;  
$sono_cachorro2 = True;
```





Chamadas às funções como parte da variável

```
$cachorro1 = new Cachorro("Nelson",3,False);  
$cachorro2 = new Cachorro("Jeremias",1,True);  
$cachorro1->comer();  
$cachorro2->dormir();
```





Diferença entre os códigos (sem classes e funções)

```
$nome_cachorro1 = "Nelson";  
$comida_cachorro1 = 3;  
$sono_cachorro1 = False;  
$nome_cachorro2 = "Jeremias";  
$comida_cachorro2 = 1;  
$sono_cachorro2 = True;  
  
//Colocando o Nelson para comer  
$comida_cachorro1 = comer($comida_cachorro1);  
  
//Colocando o Jeremias para dormir  
$sono_cachorro2 = dormir();
```

Sem orientação a objetos

Com orientação a objetos

```
$cachorro1 = new Cachorro("Nelson",3,False);  
$cachorro2 = new Cachorro("Jeremias",1,True);  
$cachorro1->comer();  
$cachorro2->dormir();
```





Vantagens da OO: Reutilização



```
$cachorro1 = new Cachorro("Nelson", 3, False);  
$cachorro2 = new Cachorro("Jeremias", 1, True);  
$cachorro3 = new Cachorro("Cléo", 247, True);  
$cachorro4 = new Cachorro("Francisco", 5, True);  
$cachorro5 = new Cachorro("Conan", 19, False);  
$cachorro6 = new Cachorro("Leleco", 3, False);  
$cachorro7 = new Cachorro("Babi", 13, True);
```





Vantagens da OO: Organização e legibilidade



```
$cachorro1 = new Cachorro("Nelson",3,False);  
$cachorro2 = new Cachorro("Jeremias",1,True);  
$cachorro1->comer();  
$cachorro2->dormir();
```



Vantagens da OO: Facilidade de manutenção



```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    public function __construct($nome,$comida,$sono)
    {
        $this->nome = $nome;
        $this->comida = $comida;
        $this->sono = $sono;
    }
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}
```

Passear?



```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    private $passeio;
    public function __construct($nome,$comida,$sono)
    {
        $this->nome = $nome;
        $this->comida = $comida;
        $this->sono = $sono;
    }
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
    public function passear()
    {
        $this->passeio = True;
    }
}
```



Ainda não acho que é tão
interessante assim ...





Frameworks usam OO

Frameworks é como uma “biblioteca” que armazena diversas funções básicas. O objetivo é fornecer uma fundação para que você desenvolva seus projetos mais eficientemente.



Sobre frameworks PHP: <https://www.hostinger.com.br/tutoriais/framework-php>

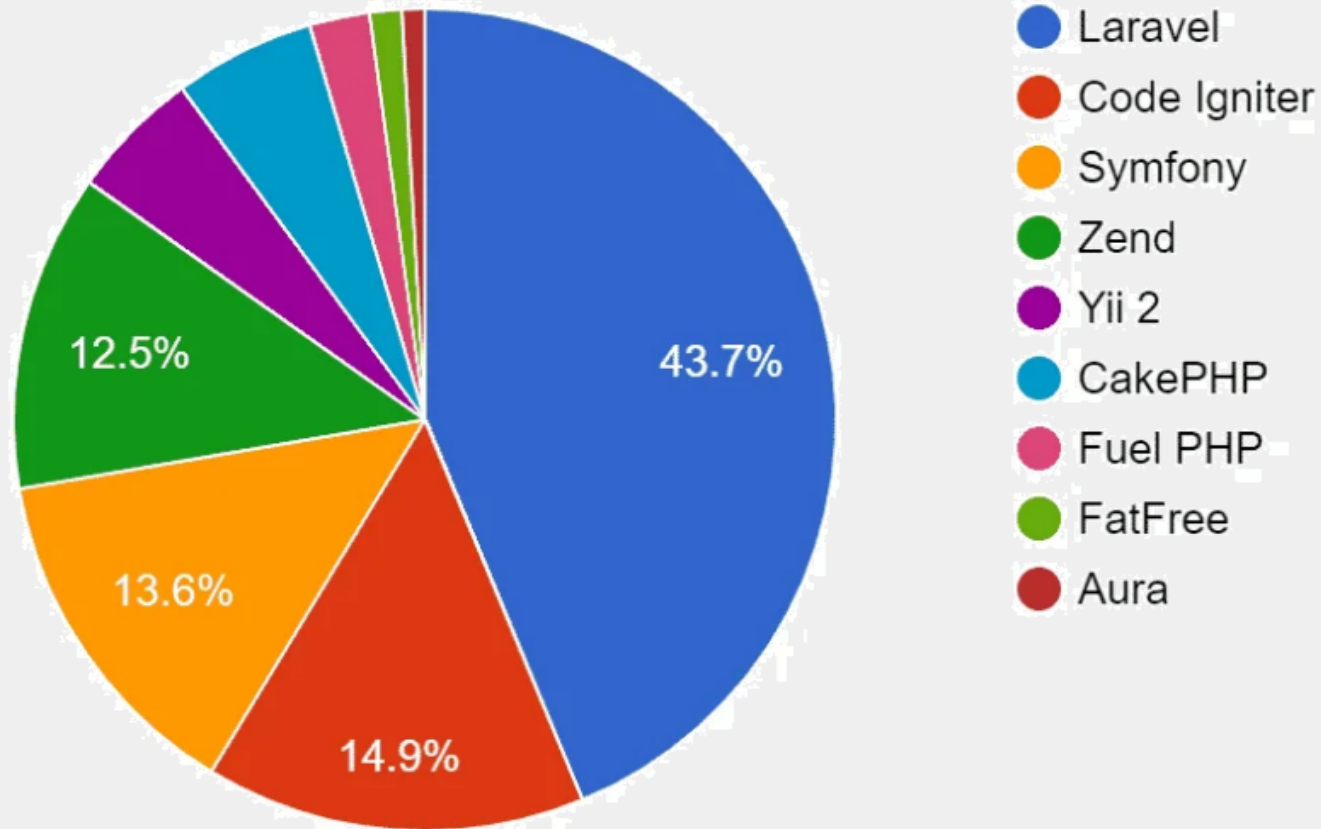




Top PHP Framework Statistics For 2025:

⋮ Pie-Chart of PHP Frameworks usage in Web Projects:

PHP Framework Used for Project Use



Frameworks
mais usados
(2025)

Fonte: <https://www.excellentwebworld.com/best-php-frameworks/>





Conceitos básicos de OO [1]

- **Classes**

Representação estática do objeto.

Exemplo: a **forma ou o molde de um bolo**.

Isso denota uma representação estática, pois essa mesma forma irá servir para a produção ou execução de outros bolos.



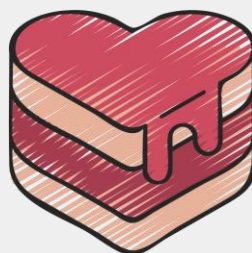


Conceitos básicos de OO [2]

- Classes
- **Objetos**

É uma representação dinâmica da classe.

Exemplo: **Diferentes bolos, criados a partir da forma**. Partindo do exemplo anterior, toda a execução ou o processo dinâmico da fabricação do bolo vai se dar com base na forma ou molde desse bolo.



Conceitos básicos de OO [3]

- Classes
- **Objetos**

métodos da classe

Variáveis que
referenciam objetos

```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;

    public function __construct($nome,$comida,$sono)
    {
        $this->nome = $nome;
        $this->comida = $comida;
        $this->sono = $sono;
    }

    public function comer()
    {
        $this->comida -= 1;
    }

    public function dormir()
    {
        $this->sono = False;
    }
}

$cachorro1 = new Cachorro("Nelson",3,False);
$cachorro2 = new Cachorro("Jeremias",1,True);
$cachorro1->comer();
$cachorro2->dormir();
```

atributos da classe

construtor



Métodos especiais: construtores e destrutores

`__construct()`: Método acionado automaticamente a cada objeto recém criado (com **`new`**). É apropriado para qualquer inicialização que o objeto necessite antes de ser utilizado.

`__destruct()`: Método acionado assim que todas as referências a um objeto forem removidas (**ex.: `$cachorro1 = null`**) ou quando o objeto for explicitamente destruído

Mais detalhes: https://www.php.net/manual/pt_BR/language.oop5.decon.php



Conceitos básicos de OO [3]

- Classes
- **Objetos**

CONTRUTORES NÃO SÃO OBRIGATÓRIOS.

Mas, neste caso, a criação do objeto precisa fornecer três parâmetros!

```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    public function __construct($nome,$comida,$sono)
    {
        $this->nome = $nome;
        $this->comida = $comida;
        $this->sono = $sono;
    }
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}

$cachorro1 = new Cachorro("Nelson",3,False);
$cachorro2 = new Cachorro("Jeremias",1,True);
$cachorro1->comer();
$cachorro2->dormir();
```




Conceitos básicos de OO [2]

- Classes
- Objetos
- **Encapsulamento**

Modificador	Classe	Subclasses	Globalmente
public	sim	sim	sim
protected	sim	sim	não
private	sim	não	não

É uma forma de implementar **controles ou níveis de acesso** aos atributos e métodos da classe.

★ **private**: mais restritivo (só permite ser acessado pela própria classe)

★ **public**: menos restritivo (pode ser acessado de qualquer ponto da aplicação)

protected: utilizando em herança.

Restringe o acesso às subclasses





Modificadores

```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}

$cachorro1 = new Cachorro();
$cachorro1->nome="Nelson";
$cachorro1->comida = 3;
$cachorro1->sono = False;
```



E se eu quisesse informar os valores dos atributos sem utilizar um construtor?

Uncaught Error: Cannot access private property Cachorro::\$nome




Modificadores




```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}

$cachorro1 = new Cachorro();
$cachorro1->nome="Nelson";
$cachorro1->comida = 3;
$cachorro1->sono = False;
```



```
class Cachorro
{
    public $nome;
    public $comida;
    public $sono;
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}

$cachorro1 = new Cachorro();
$cachorro1->nome="Nelson";
$cachorro1->comida = 3;
$cachorro1->sono = False;
```



No entanto, este nível de acesso não é recomendado e muito menos amplamente utilizado!



Modificadores: getters e setters



```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}
```



```
$cachorro1 = new Cachorro();
$cachorro1->nome="Nelson";
$cachorro1->comida = 3;
$cachorro1->sono = False;
```



```
class Cachorro
{
    public $nome;
    public $comida;
    public $sono;
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}
```



```
$cachorro1 = new Cachorro();
$cachorro1->nome="Nelson";
$cachorro1->comida = 3;
$cachorro1->sono = False;
```



```
class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    public function setName($nome)
    {
        $this->nome = $nome;
    }
    public function getName()
    {
        return $this->nome;
    }
}
```



```
$cachorro1 = new Cachorro();
$cachorro1->setName("Nelson");
echo $cachorro1->getName();
```





Conceitos básicos de OO

- Classes
- Objetos
- Encapsulamento
- **Herança**

Recurso que permite que **classes** **compartilhem atributos e métodos**. São usadas na intenção de reaproveitar código ou comportamento generalizado



O que cachorros
e gatos têm em
comum?



ANIMAIS

possuem os mesmos atributos: nome, comida e sono e métodos: comer() e dormir()!

```

class Cachorro
{
    private $nome;
    private $comida;
    private $sono;
    public function __construct($nome,$comida,$sono)
    {
        $this->nome = $nome;
        $this->comida = $comida;
        $this->sono = $sono;
    }
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}

```



```

class Gato
{
    private $nome;
    private $comida;
    private $sono;
    public function __construct($nome,$comida,$sono)
    {
        $this->nome = $nome;
        $this->comida = $comida;
        $this->sono = $sono;
    }
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}

```



ANIMAL

nome
comida
sono

comer()
dormir()

Gatos e cachorros
herdam atributos e
métodos de **ANIMAL**



```
class Animal
{
    private $nome;
    private $comida;
    private $sono;
    public function __construct($nome,$comida,$sono)
    {
        $this->nome = $nome;
        $this->comida = $comida;
        $this->sono = $sono;
    }
    public function comer()
    {
        $this->comida -= 1;
    }
    public function dormir()
    {
        $this->sono = False;
    }
}
```

```
class Cachorro extends Animal
{
    private $morder
    public function morder()
    {
        return $this->morder = True;
    }
}
```

```
class Gato extends Animal
{
    private $arranhar
    public function arranhar()
    {
        return $this->morder = True;
    }
}
```

```
$cachorro1 = new Cachorro ("Nelson",3,False);
$cachorro2 = new Cachorro ("Jeremias",1,True);
$gato1 = new Gato ("Lola",50,True);
$cachorro1->morder();
$gato1->arranhar();
```



```
var_dump($cachorro1);  
var_dump($cachorro2);  
var_dump($gato1);
```



```
object(Cachorro)#1 (4) {  
  ["morder":"Cachorro":private]=>  
  bool(true)  
  ["nome":"Animal":private]=>  
  string(6) "Nelson"  
  ["comida":"Animal":private]=>  
  int(3)  
  ["sono":"Animal":private]=>  
  bool(false)  
}  
object(Cachorro)#2 (4) {  
  ["morder":"Cachorro":private]=>  
  NULL  
  ["nome":"Animal":private]=>  
  string(8) "Jeremias"  
  ["comida":"Animal":private]=>  
  int(1)  
  ["sono":"Animal":private]=>  
  bool(true)  
}  
object(Gato)#3 (4) {  
  ["arranhar":"Gato":private]=>  
  bool(true)  
  ["nome":"Animal":private]=>  
  string(4) "Lola"  
  ["comida":"Animal":private]=>  
  int(50)  
  ["sono":"Animal":private]=>  
  bool(true)  
}
```





E o protected?



Modificadores: public



```
class Animal
{
    public $nome;
}
class Cachorro extends Animal
{
    private $morder;
    public function morder()
    {
        return $this->morder = True;
    }
}
$animal = new Animal();
$animal->nome = "Belinha";
```

Não há restrição de acesso para o uso do atributo ou método!

```
var_dump($animal);
```

```
object(Animal)#1 (1) {
    ["nome"]=>
    string(7) "Belinha"
}
```



Modificadores: protected



```
class Animal
{
    protected $nome;
}
class Cachorro extends Animal
{
    private $morder;
    public function morder()
    {
        return $this->morder = True;
    }
}
$animal = new Animal();
$animal->nome = "Belinha";
```

**Somente pode ser
acessado pela própria
classe ou pela classe
herdeira**

**Fatal error: Uncaught Error: Cannot access
protected property Animal::\$nome**



Modificadores: protected



```
class Animal
{
    protected $nome;
}

class Cachorro extends Animal
{
    private $morder;
    public function setNome($nome)
    {
        $this->nome=$nome;
    }

    public function morder()
    {
        return $this->morder = True;
    }
}

$animal = new Cachorro();
$animal->setNome("Belinha");
```

**Somente pode ser
acessado pela própria
classe ou pela classe
herdeira**

```
var_dump($animal);
```



```
object(Cachorro)#1 (2) {
    ["morder":"Cachorro":private]=>
    NULL
    ["nome":protected]=>
    string(7) "Belinha"
}
```

Modificadores: private



```
class Animal
{
    private $nome;
}

class Cachorro extends Animal
{
    private $morder;
    public function setNome($nome)
    {
        $this->nome=$nome;
    }
    public function morder()
    {
        return $this->morder = True;
    }
}

$animal = new Cachorro();
$animal->setNome("Belinha");
```

**Somente pode ser acessado
pela própria classe!
Não dá erro, mas...**

```
var_dump($animal);
```



```
object(Cachorro)#1 (3) {
    ["morder":"Cachorro":private]=>
    NULL
    ["nome":"Animal":private]=>
    NULL
    ["nome"]=>
    string(7) "Belinha"
}
```



Modificadores: private (o correto)



```
class Animal
{
    private $nome;
    public function setNome($nome)
    {
        $this->nome=$nome;
    }
}

class Cachorro extends Animal
{
    private $morder;
    public function morder()
    {
        return $this->morder = True;
    }
}

$animal = new Cachorro();
$animal->setNome("Belinha");
```

Somente pode ser acessado pela própria classe!
Deve-se criar um método na classe para manipular os atributos "private"

```
var_dump($animal);
```



```
object(Cachorro)#1 (2) {
    ["morder":"Cachorro":private]=>
    NULL
    ["nome":"Animal":private]=>
    string(7) "Belinha"
}
```



Conceitos básicos de OO

- Classes
- Objetos
- Encapsulamento
- Herança
- **Polimorfismo**

Reescrita de um método


herdado, para realizar implementações diferentes!



Conceitos básicos de OO

- Classes
- Objetos
- Encapsulamento
- Herança
- **Polimorfismo**

```
class Animal
{
    private $nome;
    private $comida;
    private $sono;
    public function emitirSom()
    {
        echo "Grunhido qualquer";
    }
}
class Cachorro extends Animal
{
    public function emitirSom()
    {
        echo "Au-au";
    }
}
class Gato extends Animal
{
    public function emitirSom()
    {
        echo "Miau";
    }
}
```



```
$animal = new Animal();
$animal->emitirSom();
echo "\n";
$cachorro = new Cachorro();
$cachorro->emitirSom();
echo "\n";
$gato = new Gato();
$gato->emitirSom();
```



```
Grunhido qualquer
Au-au
Miau
```





Cursos sobre PHP Orientado a Objetos



Node Studio Treinamentos:

https://www.youtube.com/playlist?list=PLwXQLZ3FdTVEau55kNj_zLgpXL4JZUg8l

- Curso em Vídeo (Gustavo Guanabara):

https://www.youtube.com/watch?v=KIIL63MeyMY&list=PLHz_AreHm4dmGuLl3tsvryMMD7VgcT7x





DÚVIDAS?



Créditos:

Profª: Carolina Sacramento
Carolina.sacramento@fiocruz.br

