PHP com Programação Orientada a Objetos
Node Studio Treinamentos (Rodrigo Santos)

https://www.youtube.com/watch?v=hzy_P_H-1CQ&list=PLwXQLZ3FdTVEau55kNj_zLgpXL4JZUg8I&index=1

Resumo do curso feito por Roberto Pinheiro

Aula 02 – Classes, Atributos e Métodos

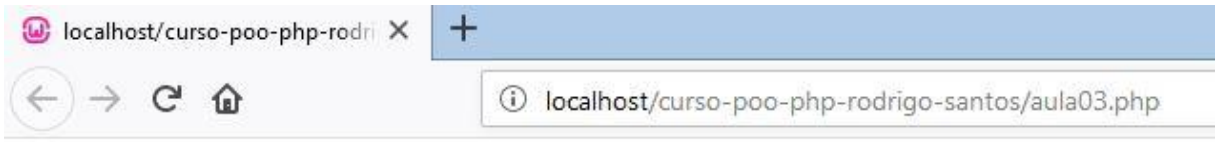
aula02.php

```
<?php

class Pessoa {
  public $nome;
  public $idade;

  public function falar(){
    echo $this->nome . " de " . $this->idade . " anos acabou de falar";
  }
}

$rodrigo = new Pessoa();
$rodrigo -> nome = "Rodrigo Santos de Oliveira";
$rodrigo -> idade = 25;
$rodrigo -> falar();
```

Aula 03 – Getters e Setters

aula03.php

```
<?php

 class Login {

   private $email;
   private $senha;

   public function getEmail() {
     return $this->email;
   }

   public function setEmail($e) {
     $email = filter_var($e, FILTER_SANITIZE_EMAIL); //filtro que limpa caracteres inválidos da variável $email
     $this->email = $email;
   }

   public function getSenha() {
     return $this->senha;
   }

   public function setSenha($s) {
     $this->senha = $s;
   }

   public function Logar() {
     if ($this->email == "teste@teste.com") {
       echo "Logado com sucesso!";
     } else {
       echo "Dados inválidos!";
     }
   }

 }

 $logar = new Login();
 $logar->setEmail("teste()/@teste.com");
 $logar->setSenha("123456");
 $logar->Logar();
 echo "<br />";
 echo $logar->getEmail();
```

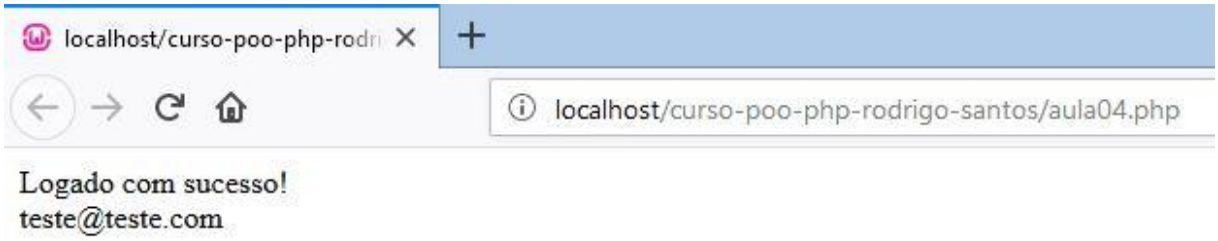


Logado com sucesso!
teste@teste.com

Aula 04 – Construtor

aula04.php

```
<?php

 class Login {

   private $email;
   private $senha;
   private $nome;

   // Construtor

   public function __construct($email, $senha, $nome) {
     $this->nome = $nome;
     $this->setEmail($email);
     $this->setSenha($senha);
   }

   public function getEmail() {
     return $this->email;
   }

   public function setEmail($e) {
     $email = filter_var($e, FILTER_SANITIZE_EMAIL); //filtro que limpa caracteres inválidos
da variável $email
     $this->email = $email;
   }

   public function getSenha() {
     return $this->senha;
   }

   public function setSenha($s) {
     $this->senha = $s;
   }

   public function Logar() {
     if ($this->email == "teste@teste.com") {
       echo "Logado com sucesso!";
     } else {
       echo "Dados inválidos!";
     }
   }

 }

 $logar = new Login("teste()/@teste.com", "123456", "Rodrigo Oliveira");
 $logar->Logar();
 echo "<br />";
 echo $logar->getEmail();
```



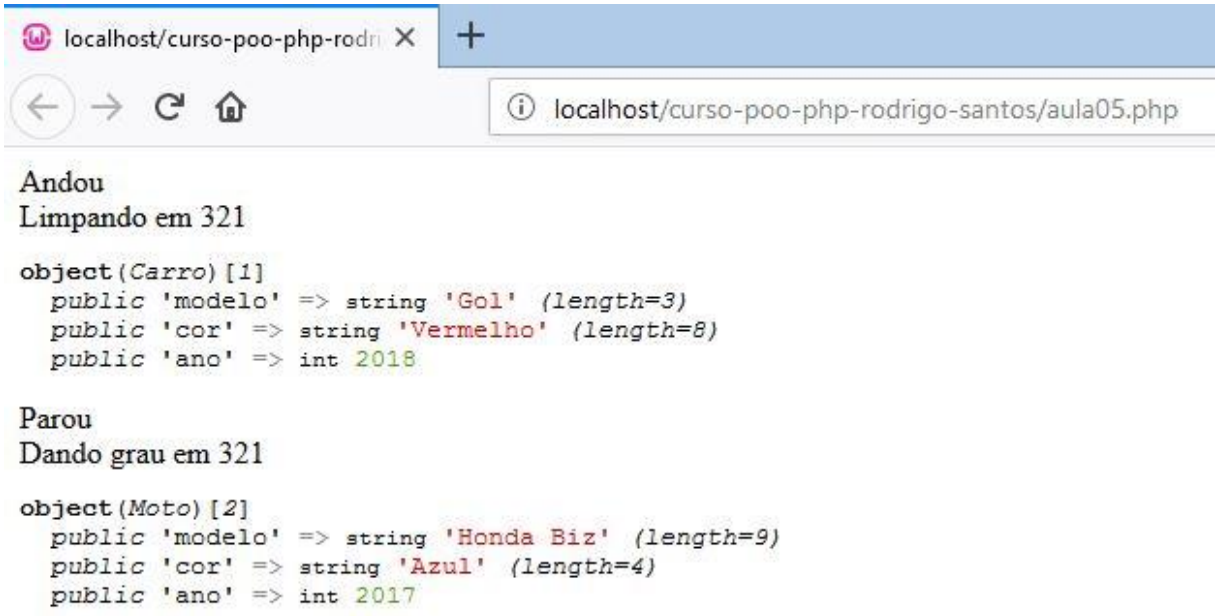
Logado com sucesso!
teste@teste.com

Aula 05 – Herança

aula05.php

```
<?php

 /* A herança é um recurso que permite que classes compartilhem atributos e métodos a fim de reaproveitar códigos ou comportamentos generalizados. */

 class Veiculo {

   public $modelo;
   public $cor;
   public $ano;

   public function Andar() {
     echo "Andou";
   }

   public function Parar() {
     echo "Parou";
   }

 }

 class Carro extends Veiculo {

   public function ligarLimpador() {
     echo "Limpando em 321";
   }

 }

 class Moto extends Veiculo {
   public function darGrau() {
     echo "Dando grau em 321";
   }

 }

 $carro = New Carro();
 $carro->modelo = "Gol";
 $carro->cor = "Vermelho";
 $carro->ano = 2018;
 $carro->Andar();
 echo "<br />";
 $carro->ligarLimpador();
 var_dump($carro);
```

```
$moto = New Moto();
$moto->modelo = "Honda Biz";
$moto->cor = "Azul";
$moto->ano = 2017;
$moto->Parar();
echo "<br />";
$moto->darGrau();
var_dump($moto);
```

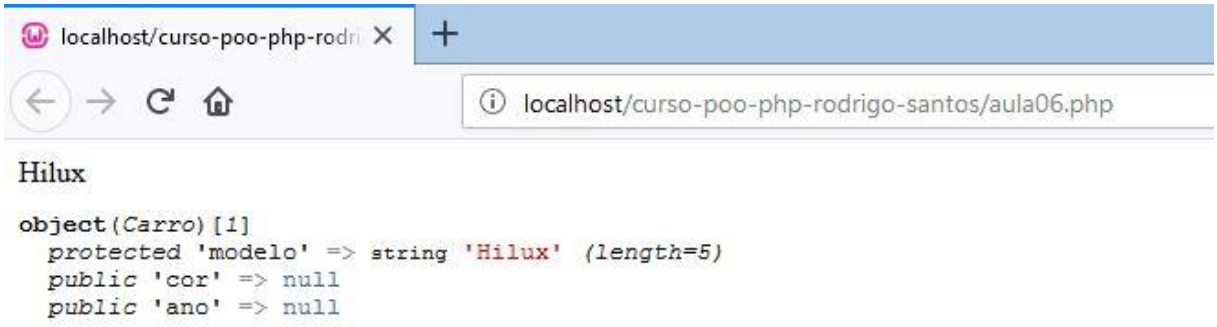


Aula 06 – Modificadores de acesso – Parte 1

aula06.php

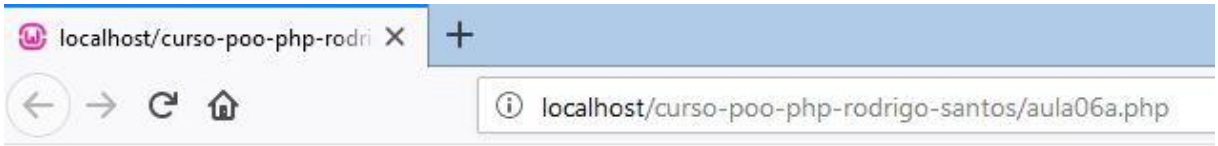
```
<?php

 /* public - fará com que não haja ocultação nenhuma, toda propriedade ou método declarado com public são acessíveis por todos que querem acessá-los */
 /* protected - visibilidade protected faz com que todos os herdeiros vejam as propriedades ou métodos protegidos como se fossem públicos */
 /* private - Ao contrário do public este modificador fará com que qualquer método ou propriedade seja acessível somente pela classe que o declarou */

 class Veiculo {

   protected $modelo;
   public $cor;
   public $ano;

   public function Andar() {
     echo "Andou";
   }

   public function Parar() {
     echo "Parou";
   }

 }

 class Carro extends Veiculo {

   public function setModelo($m){
     $this->modelo = $m;
   }

   public function getModelo(){
     return $this->modelo;
   }

   public function ligarLimpador() {
     echo "Limpando em 321";
   }

 }

 class Moto extends Veiculo {

   public function darGrau() {
     echo "Dando grau em 321";
   }

 }
```

```
$carro = New Carro();
$carro->setModelo("Hilux");
echo $carro->getModelo();
var_dump($carro);
```



aula06a.php

```
<?php

 /* public - fará com que não haja ocultação nenhuma, toda propriedade ou método declarado com public são acessíveis por todos que querem acessá-los */
 /* protected - visibilidade protected faz com que todos os herdeiros vejam as propriedades ou métodos protegidos como se fossem públicos */
 /* private - Ao contrário do public este modificador fará com que qualquer método ou propriedade seja acessível somente pela classe que o declarou */

 class Veiculo {

   private $modelo;
   public $cor;
   public $ano;

   public function setModelo($m){
     $this->modelo = $m;
   }
   public function getModelo(){
     return $this->modelo;
   }
   public function Andar() {
     echo "Andou";
   }
   public function Parar() {
     echo "Parou";
   }

 }

 class Carro extends Veiculo {

   public function ligarLimpador() {
     echo "Limpando em 321";
   }

 }

 class Moto extends Veiculo {

   public function darGrau() {
     echo "Dando grau em 321";
   }

 }

 $carro = New Veiculo();
 $carro->setModelo("Hilux");
 echo $carro->getModelo();
 var_dump($carro);
```



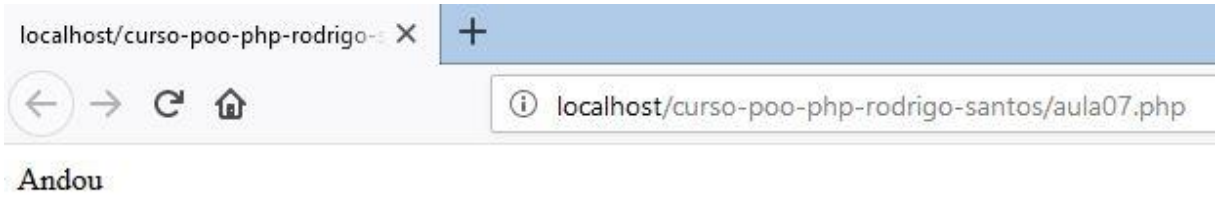
Hilux

```
object(Veiculo)[1]
  private 'modelo' => string 'Hilux' (length=5)
  public 'cor' => null
  public 'ano' => null
```

Aula 07 – Modificadores de acesso – Parte 2

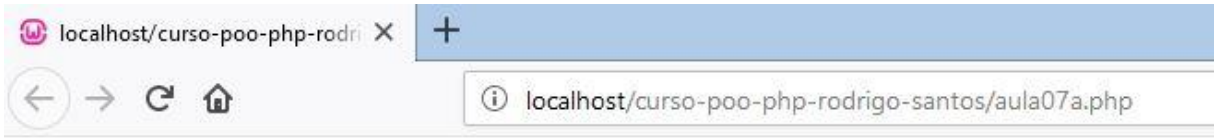
aula07.php

```
<?php

 /* A herança é um recurso que permite que classes compartilhem atributos e métodos a fim de reaproveitar códigos ou comportamentos generalizados. */

 class Veiculo {

   public $modelo;
   public $cor;
   public $ano;

   private function Andar() {
     echo "Andou";
   }

   public function Parar() {
     echo "Parou";
   }

   public function mostrarAcao(){
     $this->Andar();
   }

 }

 class Carro extends Veiculo {

   public function ligarLimpador() {
     echo "Limpando em 321";
   }

   public function mostrarAcao(){
     $this->Andar();
   }

 }

 class Moto extends Veiculo {

   public function darGrau() {
     echo "Dando grau em 321";
   }

 }

 $carro = New Veiculo();
 $carro->mostrarAcao();
```



aulao07a.php

```
<?php

 /* A herança é um recurso que permite que classes compartilhem atributos e métodos a fim de reaproveitar códigos ou comportamentos generalizados. */

 class Veiculo {

   public $modelo;
   public $cor;
   public $ano;

   protected function Andar() {
     echo "Andou";
   }

   public function Parar() {
     echo "Parou";
   }
 }

 class Carro extends Veiculo {

   public function ligarLimpador() {
     echo "Limpando em 321";
   }

   public function mostrarAcao(){
     $this->Andar();
   }

 }

 class Moto extends Veiculo {

   public function darGrau() {
     echo "Dando grau em 321";
   }

 }

 $carro = New Carro();
 $carro->mostrarAcao();
```

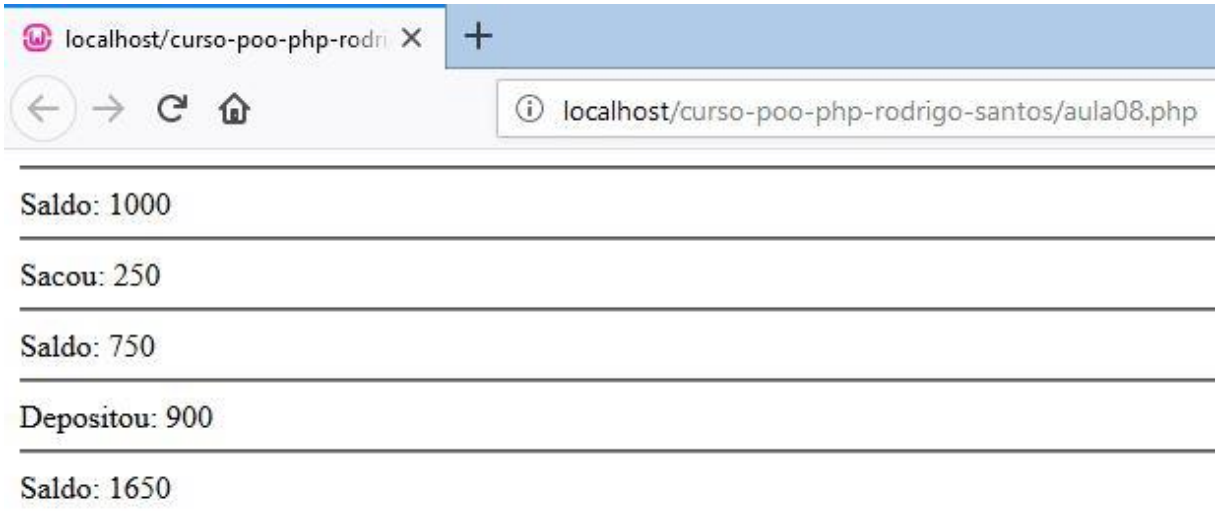

localhost/curso-poo-php-rodri
localhost/curso-poo-php-rodrigo-santos/aula07a.php

Andou

Aula 08 – Abstração

aula08.php

```
<?php

 abstract class Banco {

   protected $saldo;
   protected $limiteSaque;
   protected $juros;

   public function setSaldo($s){
     $this->saldo = $s;
   }

   public function getSaldo(){
     return $this->saldo;
   }

   abstract function Sacar($s);
   abstract function Depositar($d);

 }

 class Itau extends Banco {
  public function Sacar($s){
    $this->saldo -= $s;
    echo "<hr />Sacou: " . $s;
  }
  public function Depositar($d){
    $this->saldo += $d;
    echo "<hr />Depositou: " . $d;
  }
 }

 $itau = new Itau();
 $itau->setSaldo(1000);
 echo "<hr />Saldo: " . $itau->getSaldo();
 $itau->Sacar(250);
 echo "<hr />Saldo: " . $itau->getSaldo();
  $itau->Depositar(900);
 echo "<hr />Saldo: " . $itau->getSaldo();
```


localhost/curso-poo-php-rodr
localhost/curso-poo-php-rodrigo-santos/aula08.php
Saldo: 1000
Sacou: 250
Saldo: 750
Depositou: 900
Saldo: 1650

Aula 09 – Constantes, self e parent

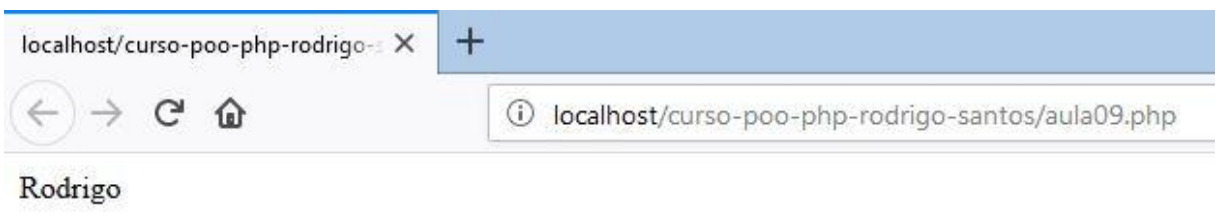
Para criar uma constante:
const nome = "Rodrigo";

Para referenciar a constante:
echo self::nome;

Para referencia o método pai:
echo parent::nome;

aula09.php

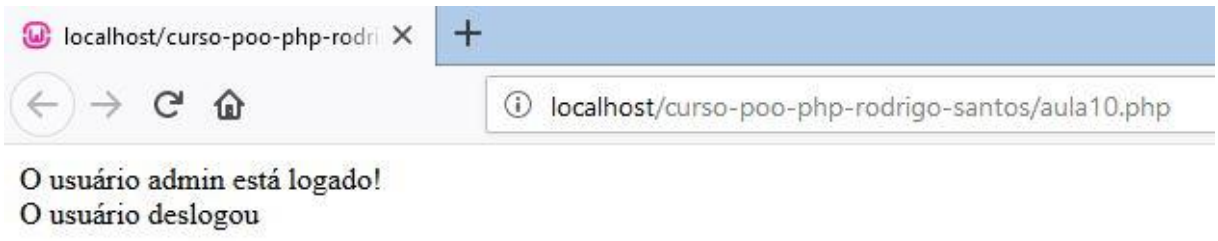
```
<?php

 class Pessoa {

   const nome = "Rodrigo";

   public function exibirNome() {
     echo self::nome;
   }

 }

 class Rodrigo extends Pessoa {

   const nome = "Oliveira";

   public function exibirNome() {
     echo parent::nome;
   }

 }

// $pessoa = new Pessoa();
// $pessoa->exibirNome();

 $rodrigo = new Rodrigo();
 $rodrigo->exibirNome();
```



Aula 10 – Métodos e atributos estáticos

aula10.php

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title></title>
   </head>
   <body>
         <?php

          class Login {

            public static $user;

            public static function verificaLogin() {
              echo "O usuário " . self::$user . " está logado!<br />";
            }

            public function sairSistema(){
              echo "O usuário deslogou";
            }

          }

          Login::$user = "admin";
          Login::verificaLogin();

          $login = new Login();
          $login->sairSistema();

         ?>

   </body>
</html>
```



Aula 11 – Polimorfismo

aula11.php

```
<?php

 class Animal {

   public function Andar() {
     echo "O animal andou";
   }

   public function Correr(){
     echo "O animal correu";
   }
}

 class Cavalo extends Animal{
   public function Andar() {
     // echo "O cavalo andou";
     $this->Correr();
   }
}

 $animal = new Cavalo();
 $animal->Andar();
```

Aula 12 – Interfaces

Uma interface serve para definir o modelo a ser usado por outras classes.

Os métodos das interfaces devem ser públicos.

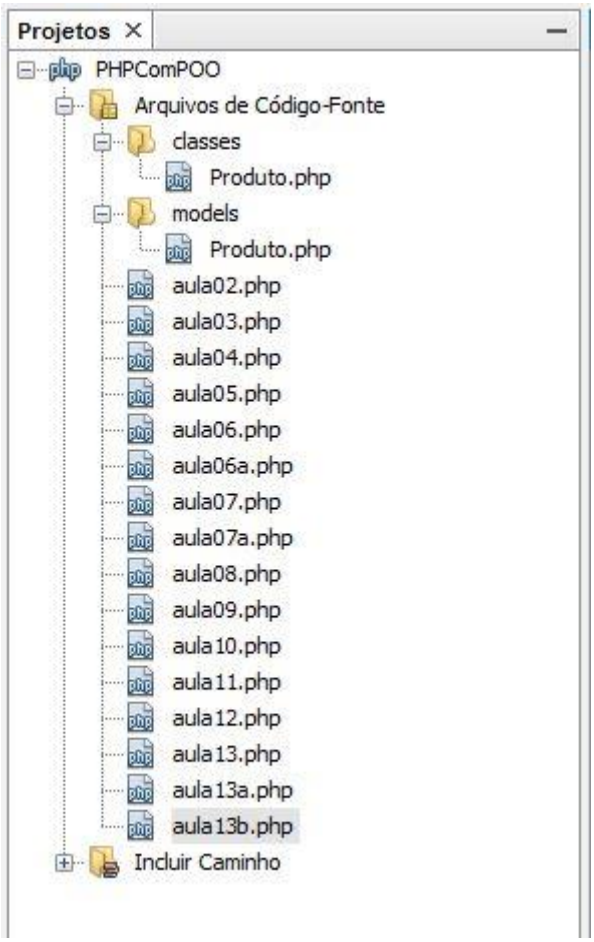
Numa interface só é necessário declarar os métodos e quando necessário, passar os parâmetros

aula12.php

```
<?php

 interface Crud {

   public function create();

   public function read();

   public function update();

   public function delete();
 }

 class Noticias implements Crud {

   public function create() {
     // lógica para criar uma notícia
   }

   public function read() {
     // lógica para ler uma notícia
   }

   public function update() {
     // lógica para atualizar uma notícia
   }

   public function delete() {
     // lógica para apagar uma notícia
   }

 }
```

Aula 13 – Namespaces

É usado para evitar conflitos entre duas classes de mesmo nome em pastas diferentes.



Produto.php

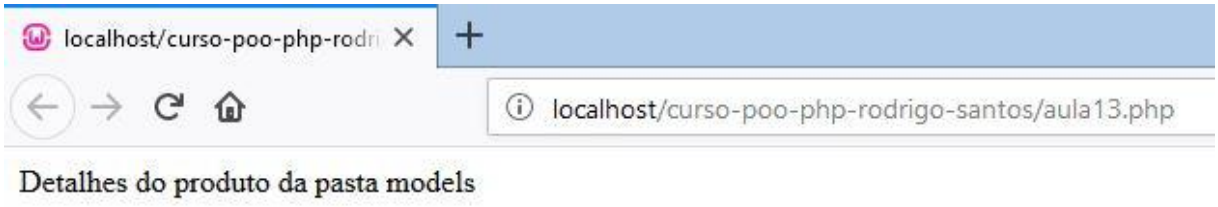
```
<?php

namespace classes;

class Produto{
  public function mostrarDetalhes(){
    echo "Detalhes do produto da pasta classes";
  }
}
```

Produto.php

```
<?php

namespace models;

class Produto{
  public function mostrarDetalhes(){
    echo "Detalhes do produto da pasta models";
  }
}
```

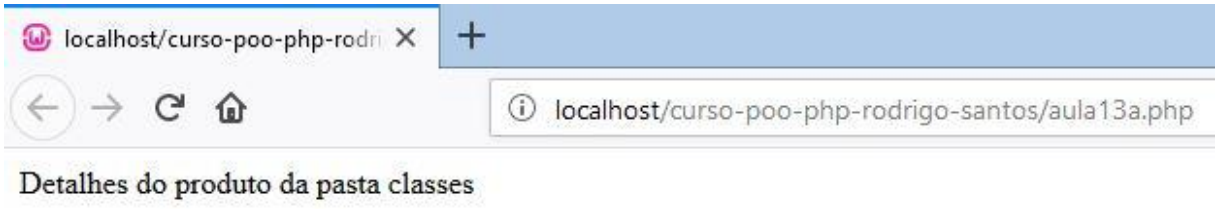
aula13.php

```
<?php

require_once 'classes/produto.php';
require_once 'models/produto.php';

// $produto = new \classes\Produto();
// $produto->mostrarDetalhes();

$produto = new \models\Produto();
$produto->mostrarDetalhes();
```



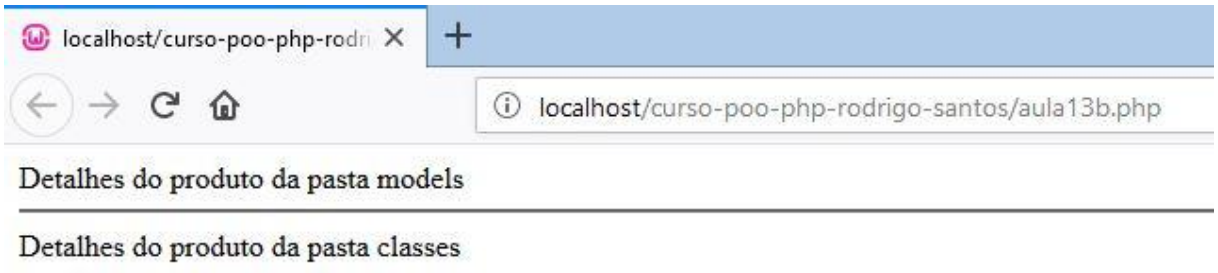
aula13a.php

```
<?php

require_once 'classes/produto.php';
require_once 'models/produto.php';

use classes\Produto;

$produto = new Produto();
$produto->mostrarDetalhes();
```



aula13b.php

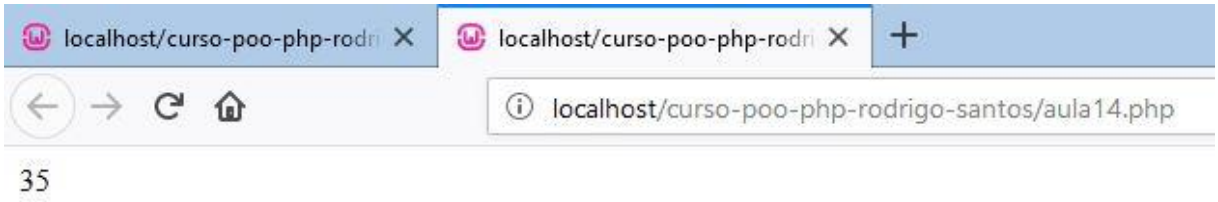
```
<?php

require_once 'classes/produto.php';
require_once 'models/produto.php';

use models\Produto as productModels;
use classes\Produto as productClass;

$produto = new productModels();
$produto->mostrarDetalhes();

echo "<hr />";

$produto2 = new productClass();
$produto2->mostrarDetalhes();
```



Aula 14 – Referência e clonagem de objetos

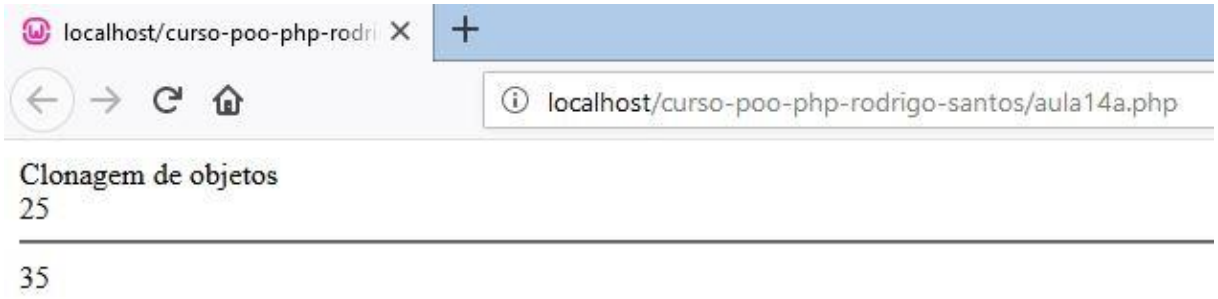
aula14.php

```
<?php

class Pessoa{
  public $idade;
}

$pessoa = new Pessoa();
$pessoa->idade = 25;

$pessoa2 = $pessoa;
$pessoa2->idade = 35;

echo $pessoa->idade;
```



aula14a.php

```
<?php

class Pessoa{
  public $idade;

  public function __clone() {
    echo "Clonagem de objetos<br />";
  }
}

$pessoa = new Pessoa();
$pessoa->idade = 25;

$pessoa2 = clone $pessoa;
$pessoa2->idade = 35;

echo $pessoa->idade;
echo "<hr />";
echo $pessoa2->idade;
```



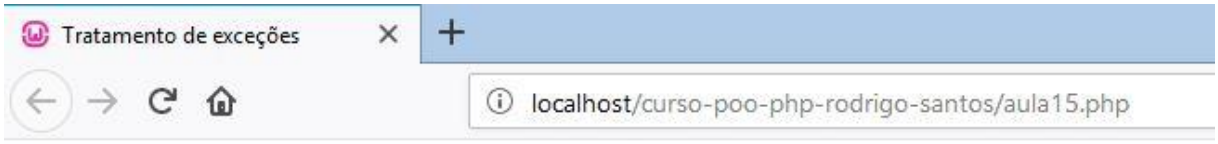
Clonagem de objetos
25

35

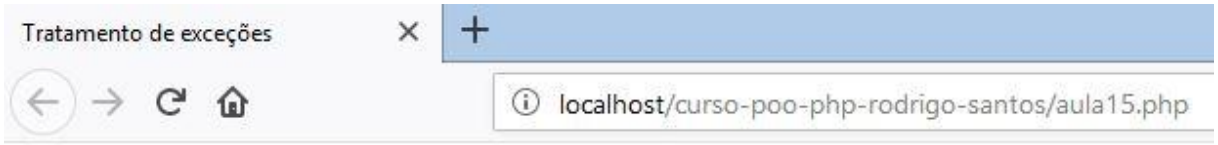
Aula 15 – Tratamento de exceções

aula15.php

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Tratamento de exceções</title>
   </head>
   <body>
        <?php

// Ocorrência normal que afeta o funcionamento da aplicação
// Exception é á classe base para todas as Exceptions
// message, code, file, line

          class Newsletter {

            public function cadastrarEmail($email) {
              if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                throw new Exception("Este email é inválido!", 1);
              } else {
                echo "Email cadastrado com sucesso!";
              }
            }

          }

          $newsletter = new Newsletter();
          try {
            $newsletter->cadastrarEmail("contato@");
          } catch (Exception $e) {
            echo "Mensagem: " . $e->getMessage() . "<br />";
            echo "Código: " . $e->getCode(). "<br />";
            echo "Linha: " . $e->getLine(). "<br />";
            echo "Arquivo: " . $e->getFile(). "<br />";
          }
         ?>

   </body>
</html>
```



Email cadastrado com sucesso!



Mensagem: Este email é inválido!
Código: 1
Linha: 18
Arquivo: C:\wamp\www\curso-poo-php-rodrigo-santos\aula15.php