

# PHP OO



Universidade Federal de Uberlândia  
Faculdade de Computação  
PET Sistemas de Informação

## Apostila do Curso de PHP Orientado a Objetos



# PHP O.O

## Sumário

O que é o PHP?.....	4
Como funciona o PHP?.....	4
O que será necessário para utilizar o PHP neste curso?.....	4
O arquivo PHP.ini.....	5
Primeiro programa em PHP Orientado a Objetos: Construindo um Login.....	7
Classes e Métodos Mágicos.....	8
Variáveis e métodos.....	10
Propriedades (atributos ou campos).....	11
Variáveis estáticas e métodos estáticos.....	13
Passagem de parâmetros em métodos.....	14
Definindo parâmetros default em métodos.....	15
Herança.....	17
Interfaces, Classes e métodos abstratos .....	18
Traits.....	24
Sobreposição (Overriding).....	25
A palavra-chave “final” .....	26
Clonagem de objetos.....	27
Comparação de objetos.....	29
Créditos:.....	31

## Objetivos desta apostila:

Este material foi desenvolvido com o intuito de auxiliar os alunos e professores ao longo do curso de PHP Orientado a Objetos. Presume-se que os alunos deverão possuir conhecimentos de Programação Orientada a Objetos, pois o foco do curso será na aplicação deste estilo de programação na linguagem PHP. Alguns assuntos serão descritos brevemente por se tratar de uma apostila mais de prática de conceitos, porém, existirão várias referências externas para aprofundamento teórico e prático. Esta apostila contém muitas ilustrações para facilitar o aprendizado.



## PHP OO

### O que é o PHP?

PHP (PHP: Hypertext Preprocessor) é uma linguagem de programação caracterizada por ser bastante utilizada no desenvolvimento de aplicações Web e pela sua facilidade de ser embutida em códigos HTML. PHP permite a criação de páginas dinâmicas, ou seja, que apresentam conteúdo que mudam de acordo com os dados de uma base de dados, por exemplo. É possível coletar dados de formulários e ainda enviar ou receber cookies. PHP é uma linguagem simples para iniciantes, porém, cheia de recursos. Também é possível desenvolver aplicações Desktop utilizando o [PHP-GTK](#).

A versão estável mais atual é a 5.4 (em 03/06/2013). É possível baixar o manual do PHP no link: <http://www.php.net/download-docs.php>. Para acompanhar lançamentos de novas versões e novidades navegue no site: <http://www.php.net/>.

Veja a história do PHP em [http://www.php.net/manual/pt\\_BR/history.php](http://www.php.net/manual/pt_BR/history.php).

### Como funciona o PHP?

O PHP é interpretado no lado do computador servidor (server-side), ou seja, quando o computador cliente envia uma requisição ao servidor, este último fica responsável por tratá-la de acordo com a necessidade da aplicação: consultar e armazenar dados no banco, processar dados, gerar o HTML que será interpretado e exibido pelo navegador do cliente, dentre outros.

PHP é uma linguagem interpretada, o que quer dizer que não é gerado um código binário executável, portanto, será necessário fornecer o código fonte ao servidor.

### O que será necessário para utilizar o PHP neste curso?

Existem várias maneiras de utilizar o PHP, como pode ser visto em [http://www.php.net/manual/pt\\_BR/install.general.php](http://www.php.net/manual/pt_BR/install.general.php). Para este curso, será utilizado:

- Interpretador PHP;
- Servidor WEB (Apache);
- Browser (Ex.: Google Chrome, Mozilla Firefox, Internet Explorer, Opera, Safari, etc);

- Banco de dados MySQL<sup>1</sup>;

É possível hospedar aplicações PHP em hosts gratuitos na WEB, por exemplo, através do [Hostinger](#). Desta maneira, não é necessário a preocupação com instalação e configuração do Apache, PHP e MySQL.

Existem várias combinações de softwares livres que facilitam o processo de instalação e configuração para utilizar o PHP. Exemplos:

- [XAMPP](#) (X qualquer sistema operacional, Apache, MySQL, PHP, Perl)
- [WAMP](#) (Windows, Apache, MySQL, PHP)

Qualquer uma das opções acima contém o pacote de software necessário para desenvolver aplicações em PHP. Portanto, instale um deles para prosseguir as atividades do curso.

Outra ferramenta que facilitará o processo de escrever os scripts PHP é a **IDE** (Integrated Development Environment), um ambiente integrado para desenvolvimento de software. Neste curso será utilizado o software livre [NetBeans](#).

Existem várias fontes de conhecimentos disponíveis em grupos de usuários que respondem dúvidas. O site <http://www.phpusergroups.org> contém uma lista de grupos de usuários em diversos países, inclusive o Brasil.

## **O arquivo PHP.ini**

O PHP tem um arquivo de configuração chamado *php.ini*. Nele estão definidos várias diretivas que controlam o comportamento do PHP em tempo de execução. Este arquivo é carregado no momento em que o servidor é iniciado, portanto, caso altere alguma das diretivas definidas nele, será necessário que o servidor seja reiniciado.

Vejam na tabela a seguir algumas das diretivas:

---

<sup>1</sup> Não é obrigatório utilizar o MySQL ou qualquer outro banco de dados no desenvolvimento de uma aplicação em PHP, porém, neste curso ele será utilizado em exercícios.

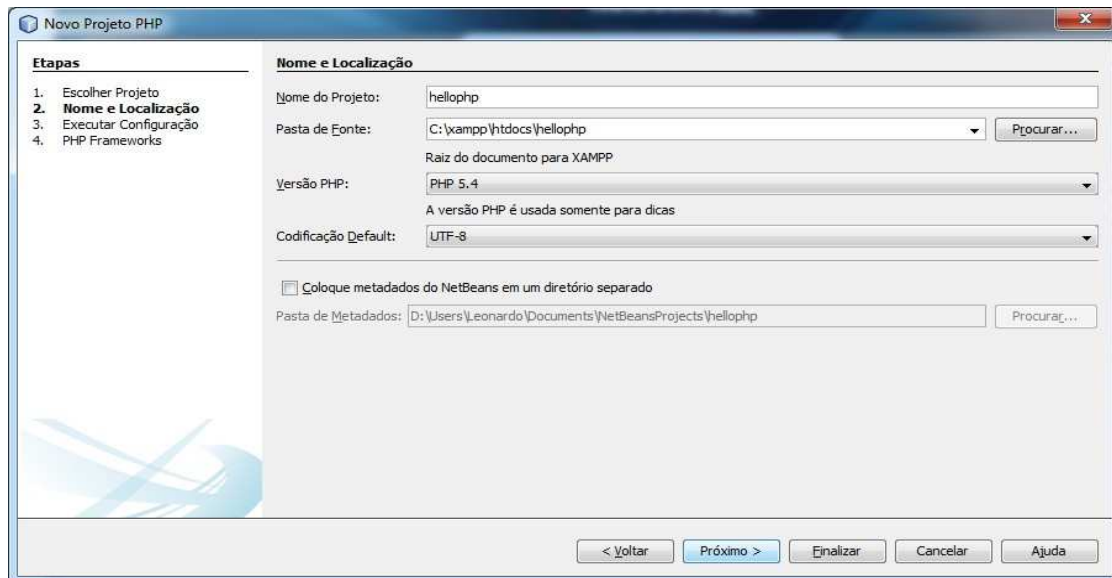
Diretiva	Valor Default	Significado
<b>expose_php</b>	On	Com o valor Off a quantidade de informações disponíveis ao atacante. Ex.: expose_php=Off
<b>date.timezone</b>		Com o valor America/Sao_Paulo está sendo configurado a data e horário corresponde ao de São Paulo para as funções que usam data e horas. Ex.: date.timezone America/Sao_Paulo
<b>disable_functions</b>		É possível desabilitar funções para que o PHP não execute-as. Ex.: disable_functions=phpinfo,exec,proc_open
<b>display_errors</b>	1	Determine se os erros devem ser impresso na tela ou escondido do usuário. 1 imprime, 2 esconde.
<b>file_uploads</b>	1	1 Permite uploads de arquivo via HTTP.
<b>upload_max_filesize</b>	2M	Tamanho máximo de upload arquivo.
<b>max_file_uploads</b>	20	Número máximo de uploads de arquivos simultaneamente
<b>max_execution_time</b>	30	Tempo máximo em segundos que um script PHP pode ser executado.
<b>memory_limit</b>	128M	Define a quantidade de memória que um script pode alocar. Se o valor for -1 não existe limite de memória.
<b>mssql.timeout</b>	60	Tempo máximo da execução de um script no mysql.
<b>post_max_size</b>	8M	Tamanho máximo uma post de dados incluindo upload de arquivos.

Veja uma lista completa de diretivas em <http://www.php.net/manual/en/ini.list.php>.

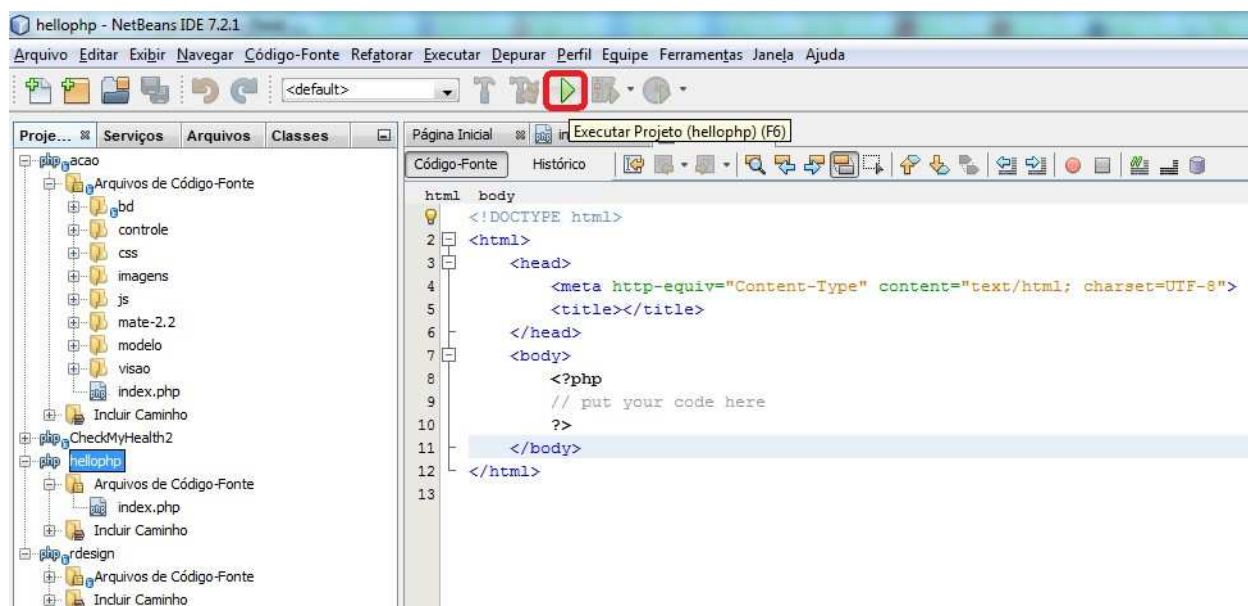
## Primeiro programa em PHP Orientado a Objetos: Construindo um Login

Abra o NetBeans e crie um novo projeto, acessando **Arquivo** → **Novo Projeto** → **PHP** → **Aplicação PHP**.

Escolha um nome para o projeto, por exemplo: *hellophp*. Certifique que a pasta fonte do projeto esteja no diretório htdocs do XAMPP, por exemplo, *C:\xampp\htdocs\hellophp*.

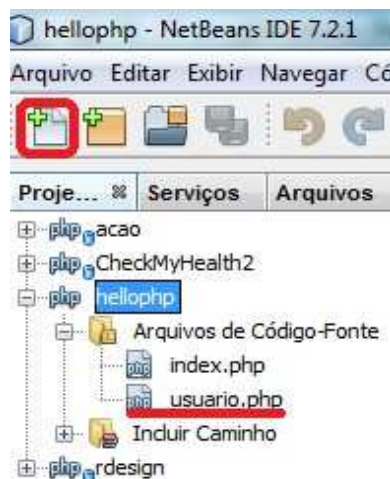


Clique em finalizar. Após criar o projeto selecione a raiz do projeto e vá em executar ou pressione F6.



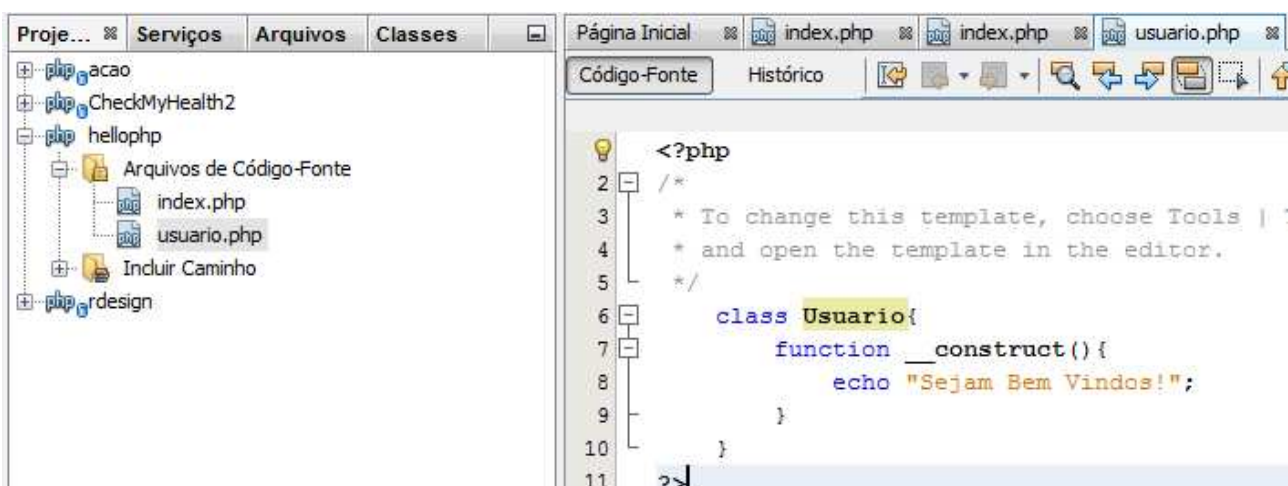
Todo código PHP deve estar dentro de um `<?php ?>`. O resultado esperado é que seja aberto uma página em branco no browser padrão do sistema operacional. Caso o mesmo não abra, digite no browser o endereço URI **http://localhost/PASTA RAIZ DO PROJETO/index.php** (Substitua PASTA RAIZ DO PROJETO pela pasta correspondente ao projeto no diretório *htdocs*). Uma página em branco deverá aparecer.

Crie um novo arquivo chamado *usuario.php*.



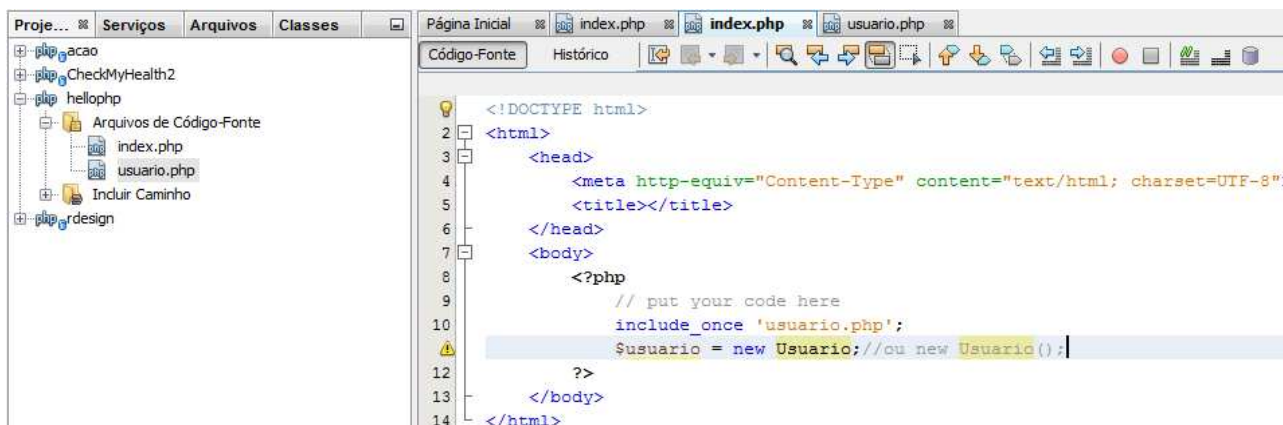
## Classes e Métodos Mágicos

Selecione o arquivo criado e crie uma classe concreta chamada `Usuario`. Nesta classe será criado o método construtor (`__construct`).



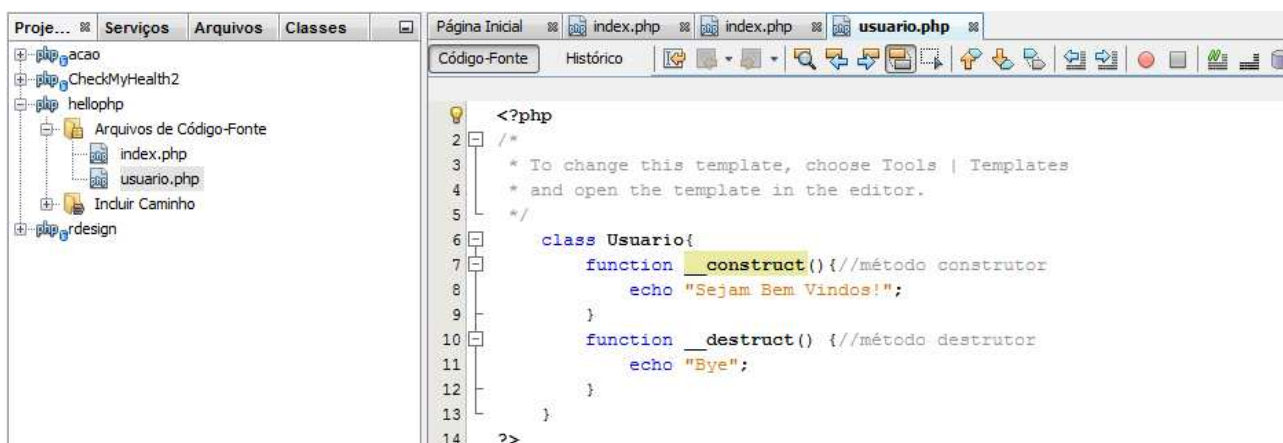


No arquivo index.php instancie a classe que você acabou de criar.



```
<!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5     <title></title>
6   </head>
7   <body>
8     <?php
9       // put your code here
10      include_once 'usuario.php';
11      $usuario = new Usuario; // ou new Usuario();
12    ?>
13  </body>
14 </html>
```

Desta vez, crie o método destrutor ( \_\_destruct ) da classe Usuario.



```
<?php
2 /*
3  * To change this template, choose Tools | Templates
4  * and open the template in the editor.
5  */
6
7  class Usuario{
8      function construct() { //método construtor
9          echo "Sejam Bem Vindos!";
10      }
11      function __destruct() { //método destrutor
12          echo "Bye";
13      }
14  }
15  ?>
```

Esses dois métodos implementados são chamados pelo PHP de “métodos mágicos” e começam com \_\_ (underscore). Além destes métodos mágicos, existem outros. Veja mais em: [http://www.php.net/manual/pt\\_BR/language.oop5.magic.php](http://www.php.net/manual/pt_BR/language.oop5.magic.php)

### CUIDADO

PHP reserva todas as funções com nomes começando com \_\_ como mágicas. É recomendado que você não use funções com nomes com \_\_ no PHP a não ser que você queira alguma funcionalidade mágica documentada.

### NOTA

Não é possível criar dois ou mais métodos construtores no PHP.

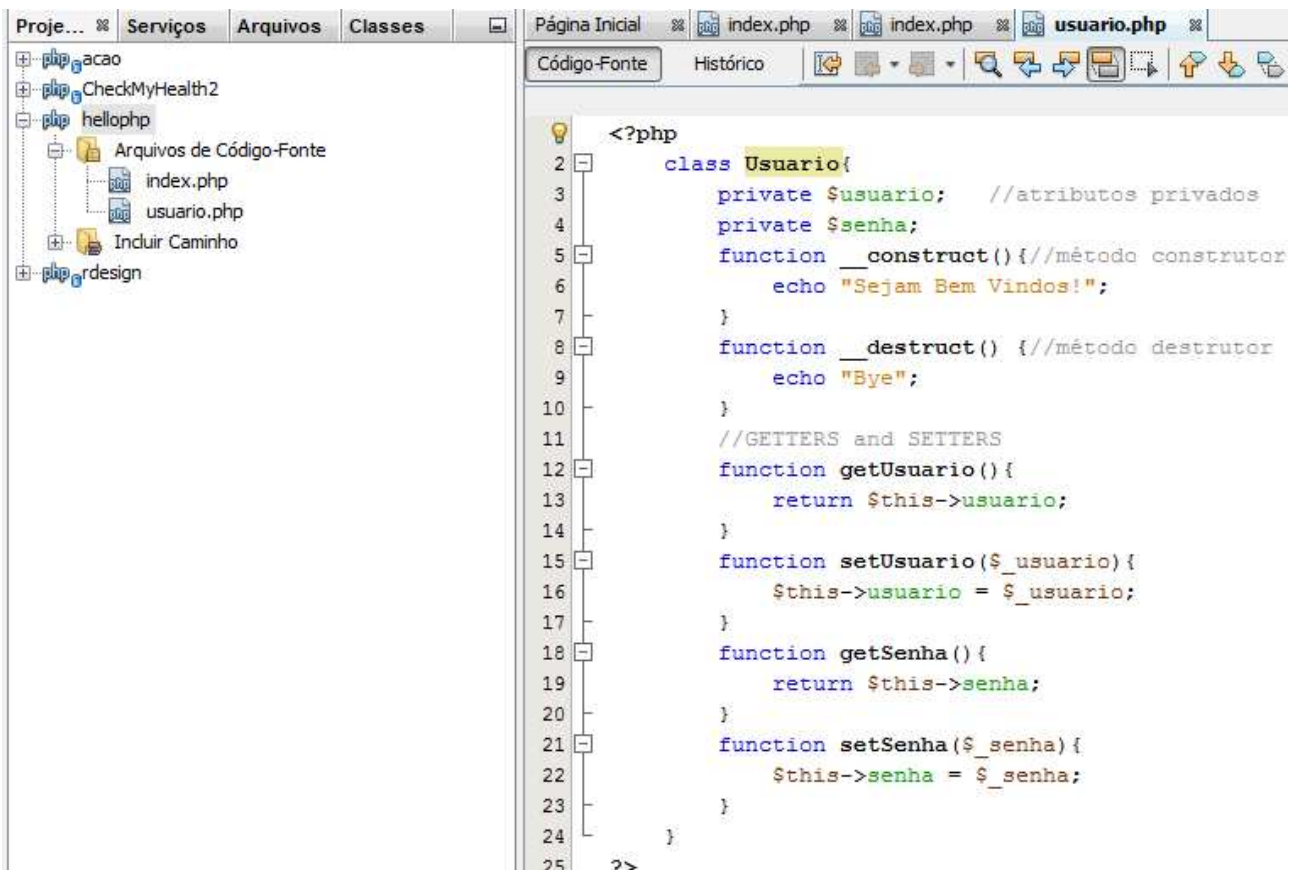
## Variáveis e métodos

Variáveis em PHP são iniciadas com o símbolo \$. Além disso, PHP é uma linguagem **não-tipada**. Isto quer dizer que não é necessário declarar o tipo das variáveis, pois o PHP possui a capacidade de inferir sobre tais tipos.

Crie duas variáveis uma chamada `usuario` e a outra `senha`. Em seguida crie os respectivos métodos GETTERS e SETTERS.

### DICA

Caso não consiga gerar os métodos GETTERS e SETTERS automaticamente pelo Netbeans, utilize o site <http://www.icurtain.co.uk/getset.php>, para agilizar na construção dos métodos.



```
<?php
2 class Usuario{
3     private $usuario; //atributos privados
4     private $senha;
5     function __construct(){//método construtor
6         echo "Sejam Bem Vindos!";
7     }
8     function __destruct(){//método destrutor
9         echo "Bye";
10    }
11    //GETTERS and SETTERS
12    function getUsuario(){
13        return $this->usuario;
14    }
15    function setUsuario($_usuario){
16        $this->usuario = $_usuario;
17    }
18    function getSenha(){
19        return $this->senha;
20    }
21    function setSenha($_senha){
22        $this->senha = $_senha;
23    }
24 }
25 ?>
```

### NOTA

Variáveis ou atributos no PHP iniciam com o caractere '\$'. Ex.: `$usuario`, `$senha`.

## NOTA

`$this->` é uma pseudo variável utilizada para acessar uma propriedade NÃO estática que está disponível na classe. Ela é uma referência para o objeto que chama inicialmente.

## NOTA

É possível definir constantes nas classes PHP. Constantes são valores fixos, ou seja, que não variam ao longo da execução do programa. Veja um exemplo de constante:

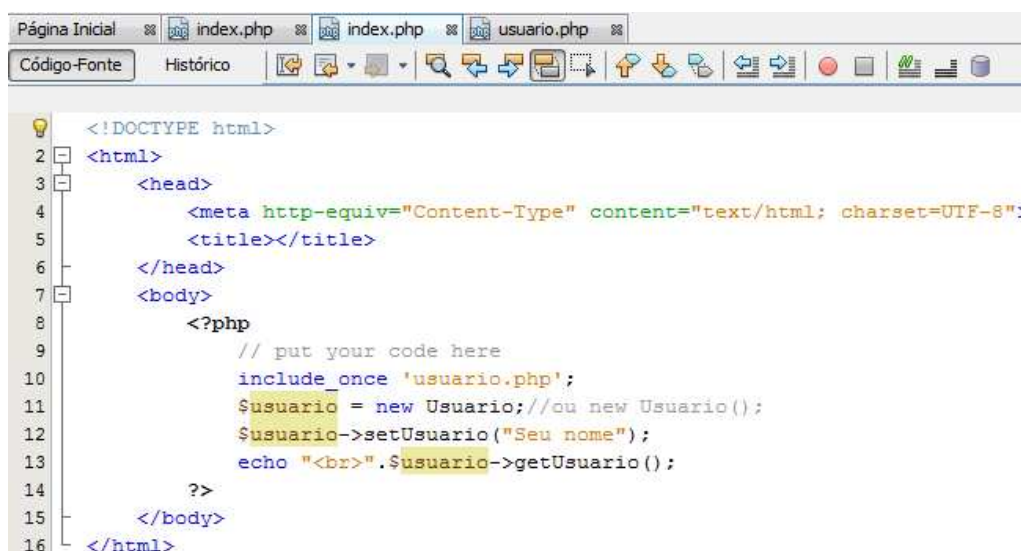
```
const pi= 3.14;
```

Veja que diferentemente de um atributo da classe, uma constante não possui o símbolo \$.

Para acessar uma constante use:

`Classe::pi`; onde `Classe` é o nome da classe.

Agora que você já criou as variáveis `$usuario` e `$senha` com os respectivos métodos GETTERS e SETTERS, teste-os, através do arquivo `index.php` como ilustrado na figura a seguir:



```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title></title>
</head>
<body>
<?php
    // put your code here
    include_once 'usuario.php';
    $usuario = new Usuario;//ou new Usuario();
    $usuario->setUsuario("Seu nome");
    echo "<br>".$usuario->getUsuario();
?>
</body>
</html>
```

## NOTA

Perceba, que um código HTML `<br>` foi posto junto ao PHP. Esta é uma das facilidades que o PHP fornece.

## Propriedades (atributos ou campos)

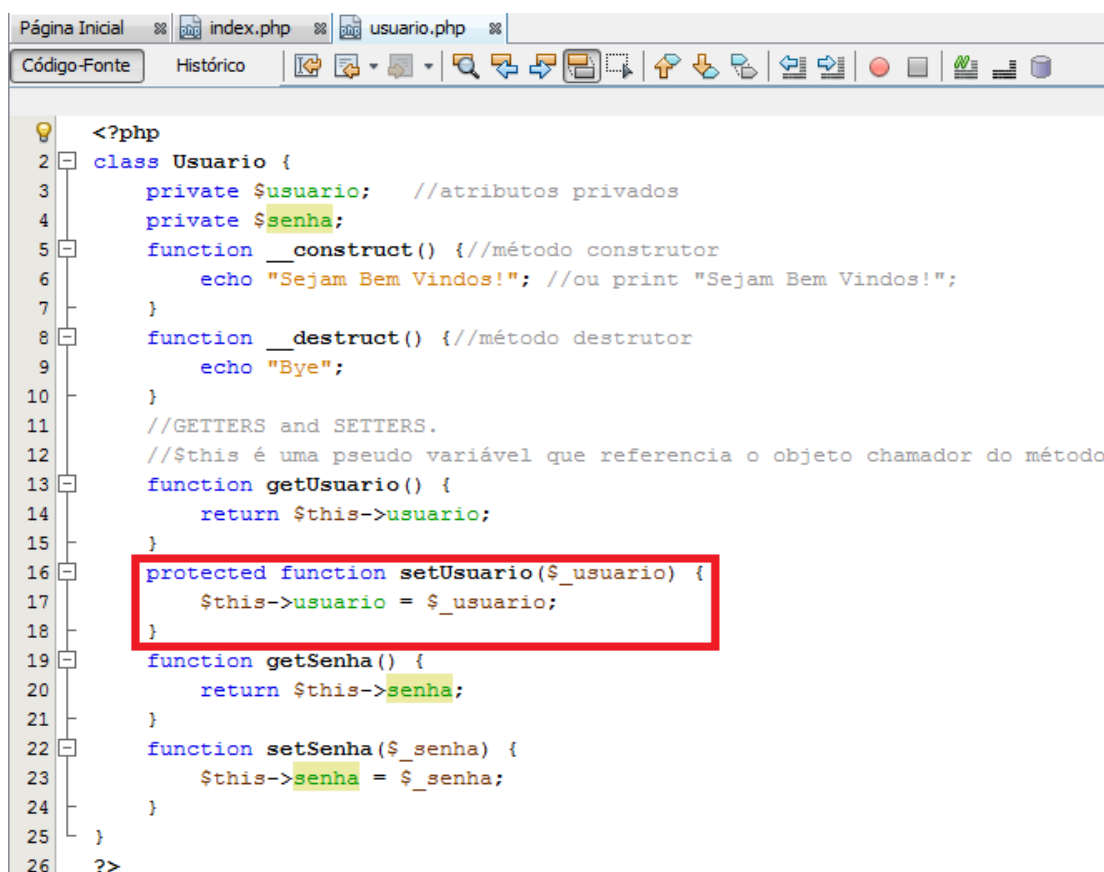
Os métodos e as propriedades (atributos ou campos) possuem visibilidade (contexto) **public**, **protected** e **private** (público, protegido e privado).

Ex.:

Métodos	Atributos	Visibilidade (contexto)
<code>public function getNome();</code> //ou <code>function getNome();</code>	<code>public \$nome;</code>	Default para os métodos – Em qualquer contexto é possível acessá-lo. <i>Visibilidade fraca.</i>
<code>private function getNome();</code>	<code>private \$nome;</code>	Apenas no contexto da classe é possível acessá-lo. <i>Visibilidade forte.</i>
<code>protected function getNome();</code>	<code>protected \$nome;</code>	Apenas no contexto da classe e subclasses é possível acessá-lo. <i>Visibilidade média.</i>

Caso queira uma explicação mais detalhada e com exemplos, consulte [http://www.php.net/manual/pt\\_BR/language.oop5.visibility.php](http://www.php.net/manual/pt_BR/language.oop5.visibility.php)

Na classe usuário, altere a visibilidade do método `setUsuario()` para `protected` e execute o arquivo *index.php* no Browser.



```
<?php
2 class Usuario {
3     private $usuario; //atributos privados
4     private $senha;
5     function __construct() { //método construtor
6         echo "Sejam Bem Vindos!"; //ou print "Sejam Bem Vindos!";
7     }
8     function __destruct() { //método destrutor
9         echo "Bye";
10    }
11    //GETTERS and SETTERS.
12    //$this é uma pseudo variável que referencia o objeto chamador do método
13    function getUsuario() {
14        return $this->usuario;
15    }
16    protected function setUsuario($_usuario) {
17        $this->usuario = $_usuario;
18    }
19    function getSenha() {
20        return $this->senha;
21    }
22    function setSenha($_senha) {
23        $this->senha = $_senha;
24    }
25 }
26 ?>
```

Ocorrerá o seguinte erro fatal devido não ser permitido chamar um método protegido fora do contexto da classe e das subclasses:



Sejam Bem Vindos!

**Fatal error:** Call to protected method Usuario::setUsuario() from context " in C:\xampp\htdocs\hellophp\index.php on line 12

Altere novamente a visibilidade do método para public.

## Variáveis estáticas e métodos estáticos

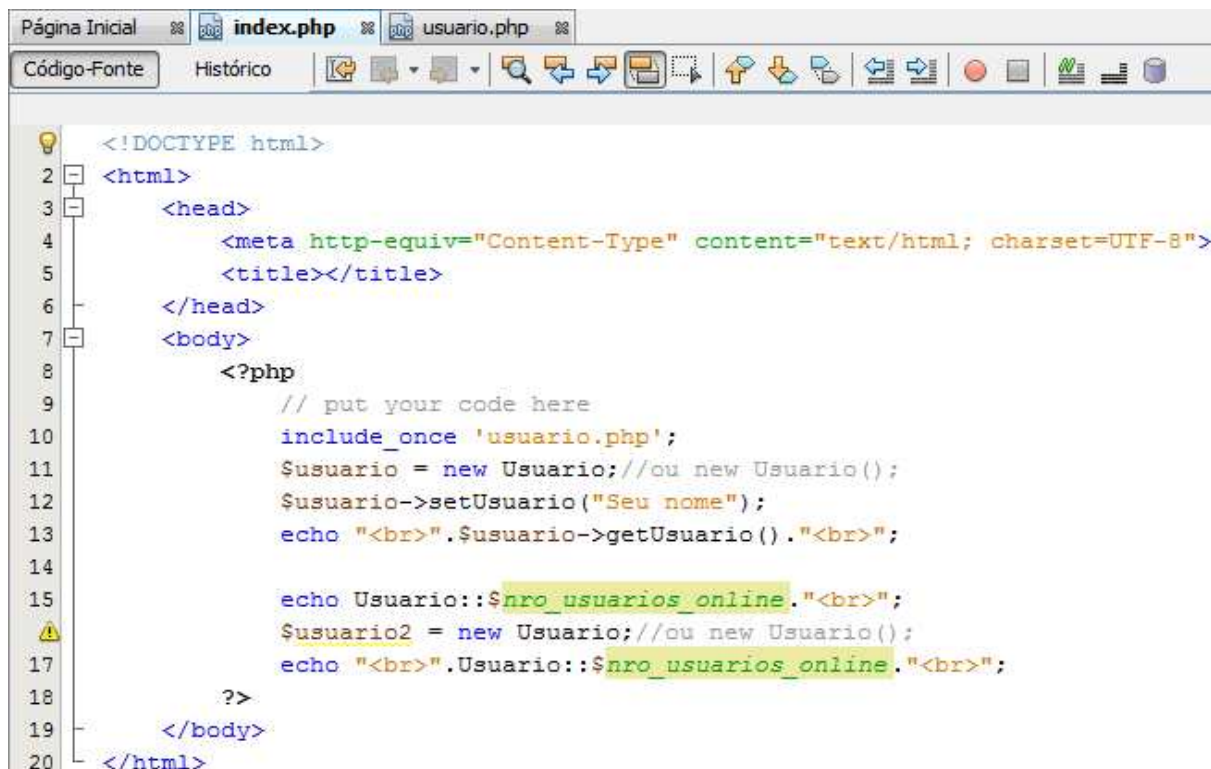
Crie uma variável estática chamada `$nro_usuarios_online` e atribua o valor 0 para ela. Em seguida, crie um método chamado `alteraNroUsuarios` para que cada vez que o método construtor for chamado, seja chamado o método `alteraNroUsuarios` que por sua vez, incrementará em 1 o número de usuários online.

```
<?php
2 class Usuario {
3     private $usuario;    //atributos privados
4     private $senha;
5     public static $nro_usuarios_online = 0;
6
7     function __construct() { //método construtor
8         echo "Sejam Bem Vindos!"; //ou print "Sejam Bem Vindos!";
9         Usuario::alteraNroUsuariosOnline();
10    }
11
12    function __destruct() { //método destrutor
13        echo "Bye";
14    }
15
16    static function alteraNroUsuariosOnline() {
17        Usuario::$nro_usuarios_online += 1;
18    }
19 }
```

### DICA

`self::alteraNroUsuariosOnline()` tem o mesmo efeito de `Usuario::alteraNroUsuariosOnline()`. A palavra reservada `self::` é usada para referenciar elementos estáticos da classe. Enquanto que `$this->` é usado para referenciar elementos não estáticos.

Teste o método criado alterando o arquivo *index.php*.



```
<?DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // put your code here
      include_once 'usuario.php';
      $usuario = new Usuario;//ou new Usuario();
      $usuario->setUsuario("Seu nome");
      echo "<br>".$usuario->getUsuario()."<br>";

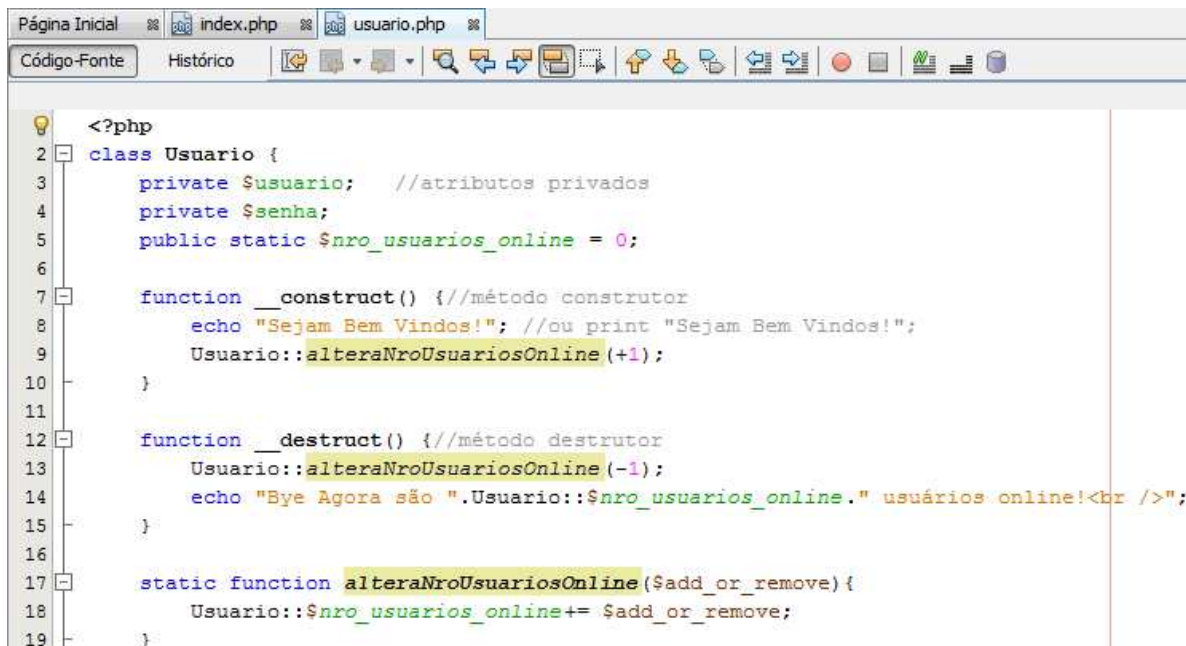
      echo Usuario::$nro_usuarios_online."<br>";
      $usuario2 = new Usuario;//ou new Usuario();
      echo "<br>".Usuario::$nro_usuarios_online."<br>";
    ?>
  </body>
</html>
```

Para mais exemplos de membros estáticos, consulte o link:  
[http://www.php.net/manual/pt\\_BR/language.oop5.static.php](http://www.php.net/manual/pt_BR/language.oop5.static.php).

## Passagem de parâmetros em métodos

O método `alteraNroUsuariosOnline()` foi feito, porém deseja-se que quando o método `__destruct` seja chamado, seja decrementada em 1 a variável `$nro_usuarios_online`. Uma das soluções, é passar um parâmetro para a função `alteraNroUsuariosOnline()` que permitirá incrementar ou decrementar a variável.





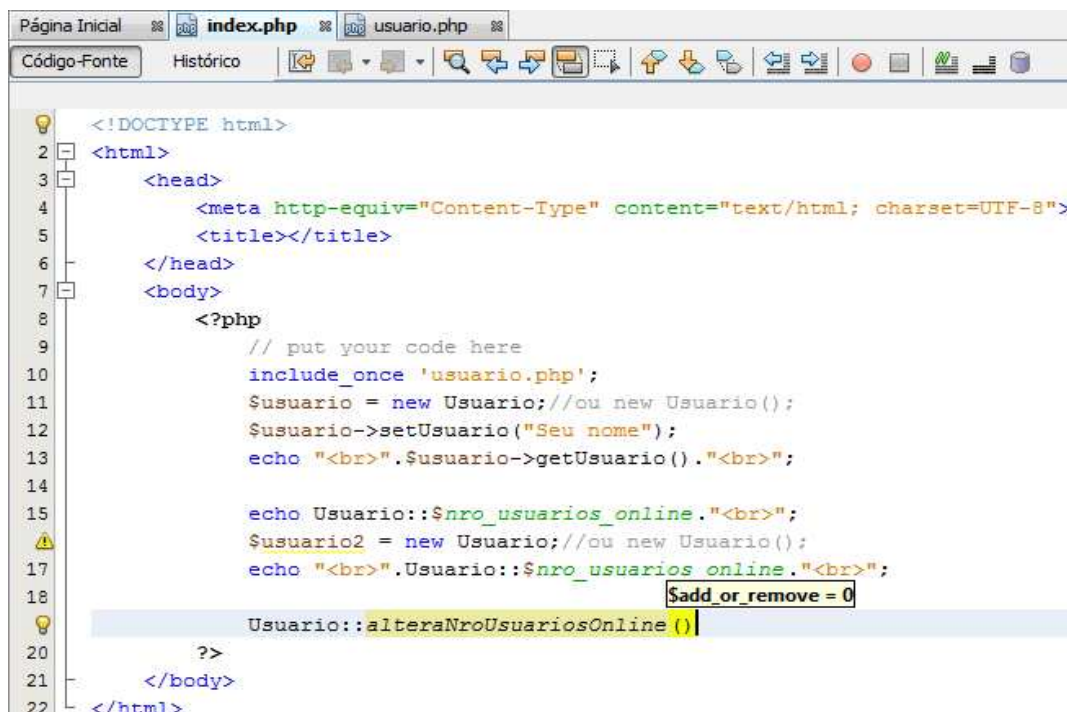
```
<?php
2 class Usuario {
3     private $usuario;    //atributos privados
4     private $senha;
5     public static $nro_usuarios_online = 0;
6
7     function __construct() { //método construtor
8         echo "Sejam Bem Vindos!"; //ou print "Sejam Bem Vindos!";
9         Usuario::alteraNroUsuariosOnline(+1);
10    }
11
12    function __destruct() { //método destrutor
13        Usuario::alteraNroUsuariosOnline(-1);
14        echo "Bye Agora são ".Usuario::$nro_usuarios_online." usuários online!<br />";
15    }
16
17    static function alteraNroUsuariosOnline($add_or_remove){
18        Usuario::$nro_usuarios_online+= $add_or_remove;
19    }
}
```

## NOTA

Embora o método `alteraNroUsuariosOnline($add_or_remove)` tenha recebido um número inteiro como parâmetro, é possível passar qualquer tipo de dados, não somente tipos primitivos (inteiro, float, string, etc.) como também tipos complexos (objetos).

## Definindo parâmetros default em métodos

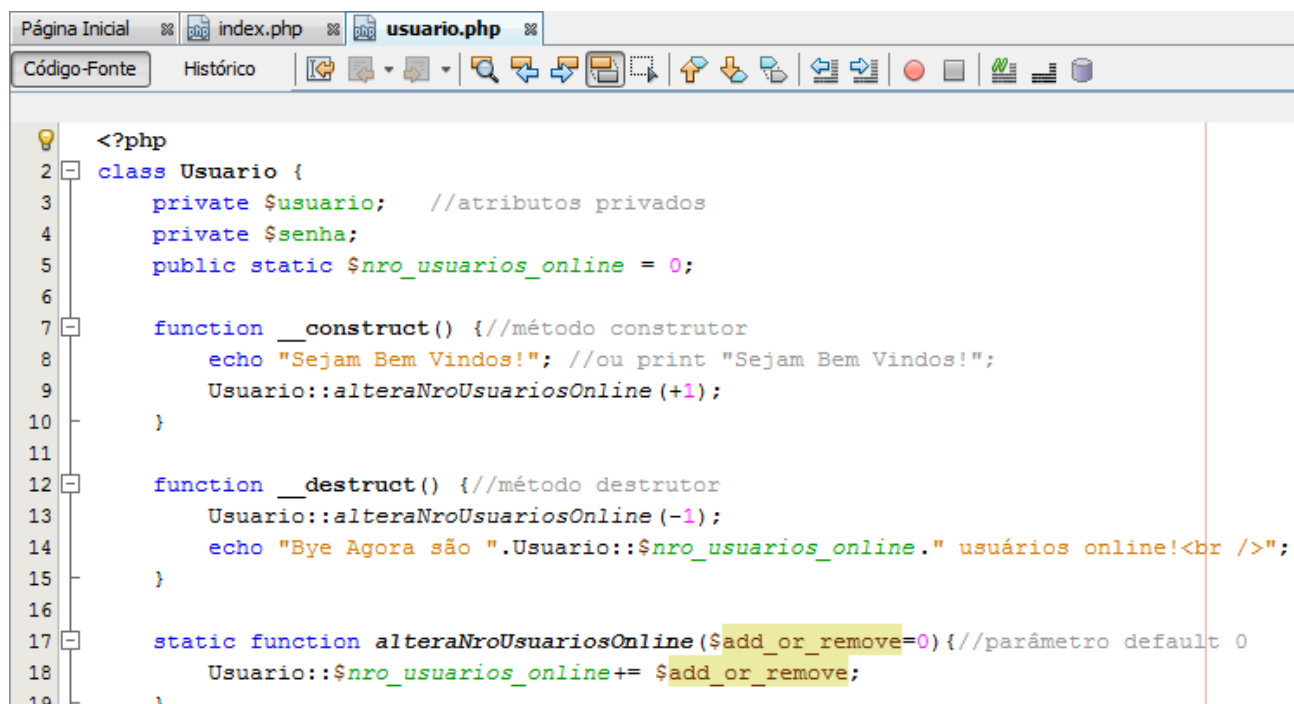
Um recurso interessante do PHP é definir valores default para parâmetros de métodos. No exemplo a seguir, se o programador não passar nenhum parâmetro para o método `alteraNroUsuariosOnline($add_or_remove)`, o valor que esta variável assumirá é 0.



```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title></title>
</head>
<body>
<?php
// put your code here
include_once 'usuario.php';
$usuario = new Usuario;//ou new Usuario();
$usuario->setUsuario("Seu nome");
echo "<br>".$usuario->getUsuario()."<br>";

echo Usuario::$nro_usuarios_online."<br>";
$usuario2 = new Usuario;//ou new Usuario();
echo "<br>".Usuario::$nro_usuarios_online."<br>";
$add_or_remove = 0;
Usuario::alteraNroUsuariosOnline()
?>
</body>
</html>
```

Perceba na figura anterior, que é possível utilizar o método `alteraNroUsuariosOnline()` sem passar nenhum parâmetro para a função, pois, o valor default, que no caso é 0, foi definido.



```
<?php
class Usuario {
    private $usuario; //atributos privados
    private $senha;
    public static $nro_usuarios_online = 0;

    function __construct() { //método construtor
        echo "Sejam Bem Vindos!"; //ou print "Sejam Bem Vindos!";
        Usuario::alteraNroUsuariosOnline(+1);
    }

    function __destruct() { //método destrutor
        Usuario::alteraNroUsuariosOnline(-1);
        echo "Bye Agora são ".Usuario::$nro_usuarios_online." usuários online!<br />";
    }

    static function alteraNroUsuariosOnline($add_or_remove=0) { //parâmetro default 0
        Usuario::$nro_usuarios_online+= $add_or_remove;
    }
}
```

Para mais detalhes sobre como passar argumentos para funções e métodos acesse: [http://www.php.net/manual/pt\\_BR/functions.arguments.php](http://www.php.net/manual/pt_BR/functions.arguments.php)

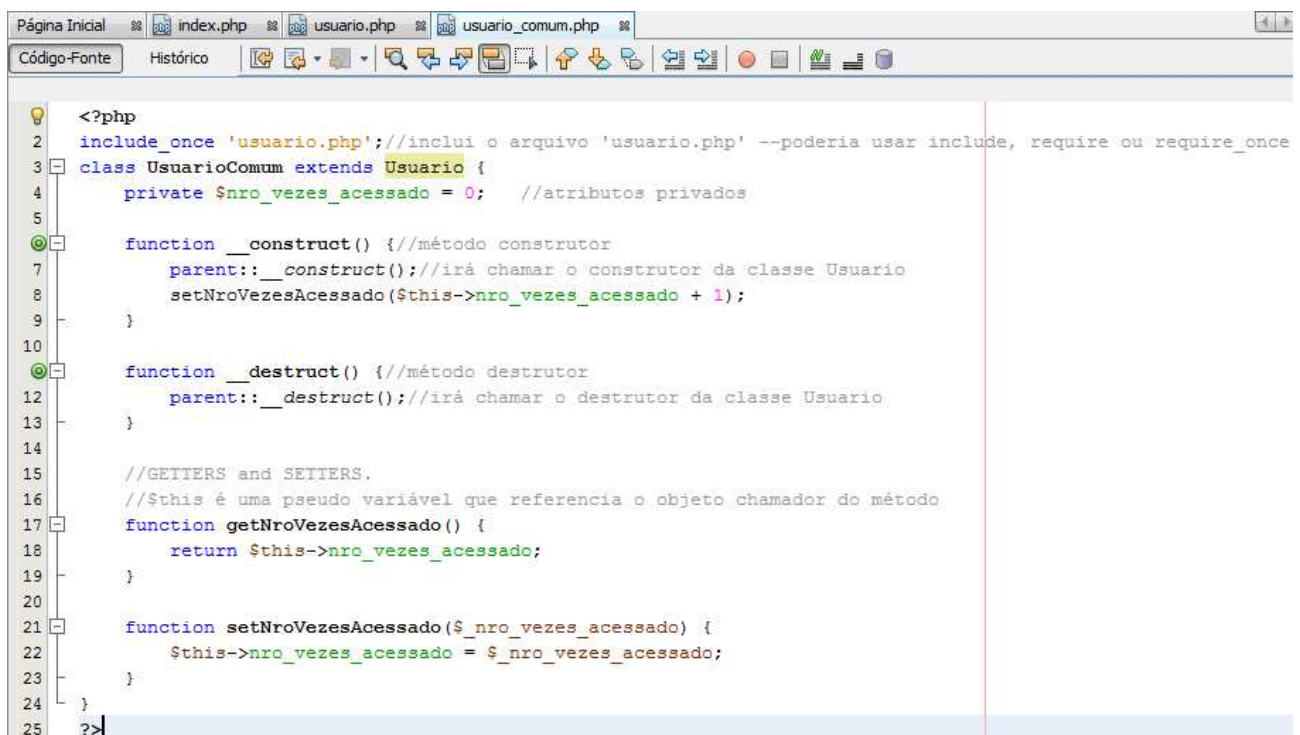


## Herança

No paradigma Orientado a Objetos, vários são os benefícios da herança. Como visto em <http://erpbasic.blogspot.com.br/2012/01/inheritance-advantages-and.html>, a herança permite reusabilidade de código, extensibilidade, ocultamento de dados, dentre outros. Em PHP, utilizar herança é muito semelhante ao modo de utilizar na linguagem Java. Para isso, é utilizada a palavra reservada `extends`.

Exemplo: no mesmo projeto, adicione uma classe chamada `UsuarioComum` que estenderá a classe `Usuario`.

Na classe `UsuarioComum` existe um método construtor e destrutor que invoca o construtor e destrutor da super classe (`Usuario`). Além disso, um atributo chamado `nro_vezes_acessado` estará na classe para fins estáticos saber quantas vezes um usuário comum acessou o sistema. Certifique de que este atributo será incrementado em um a cada vez que o usuário comum acessar o sistema.

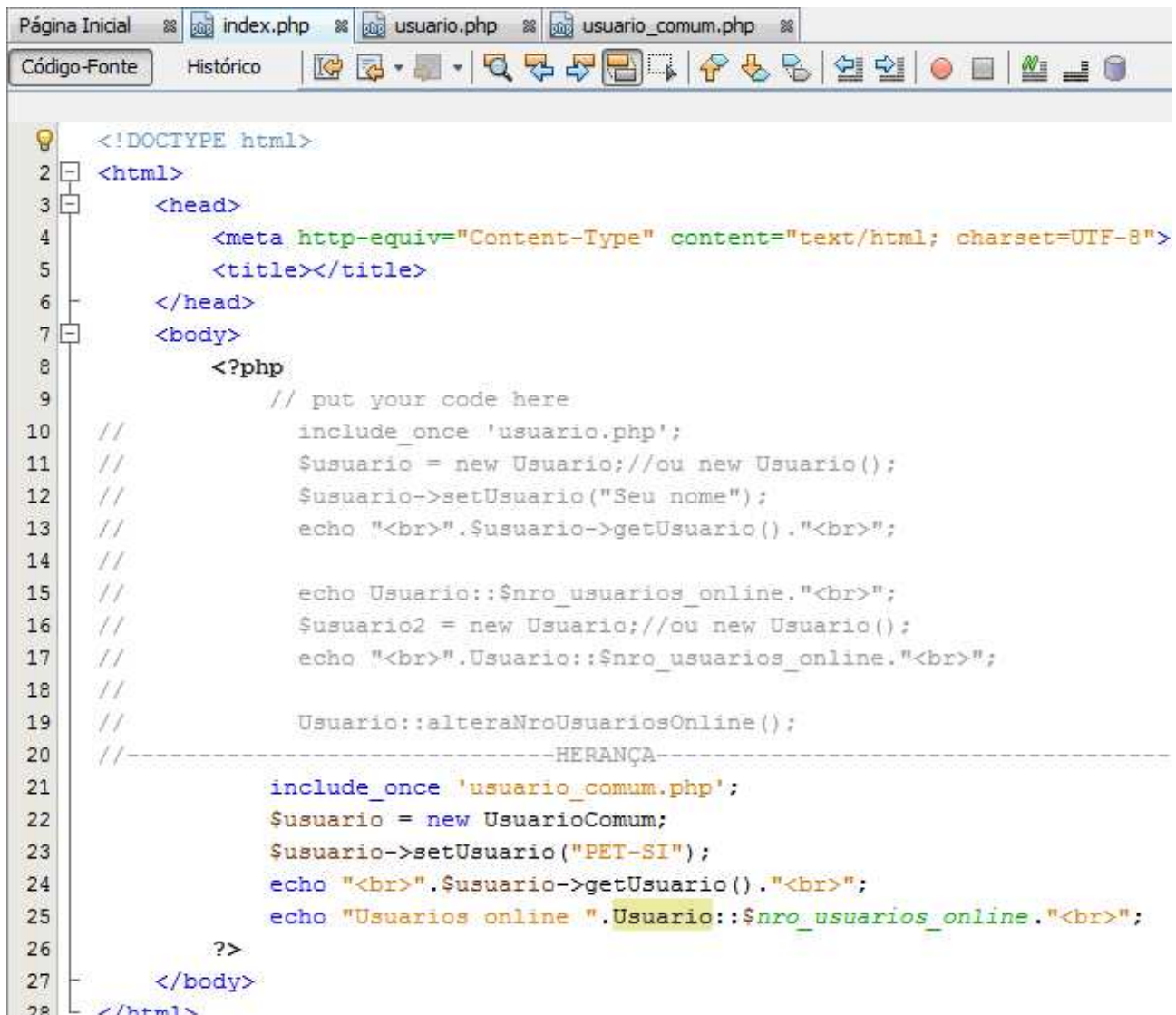


```
<?php
2 include_once 'usuario.php'; //inclui o arquivo 'usuario.php' --poderia usar include, require ou require_once
3 class UsuarioComum extends Usuario {
4     private $nro_vezes_acessado = 0; //atributos privados
5
6     function __construct() { //método construtor
7         parent::__construct(); //irá chamar o construtor da classe Usuario
8         setNroVezesAcessado($this->nro_vezes_acessado + 1);
9     }
10
11     function __destruct() { //método destrutor
12         parent::__destruct(); //irá chamar o destrutor da classe Usuario
13     }
14
15     //GETTERS and SETTERS.
16     //$this é uma pseudo variável que referencia o objeto chamador do método
17     function getNroVezesAcessado() {
18         return $this->nro_vezes_acessado;
19     }
20
21     function setNroVezesAcessado($_nro_vezes_acessado) {
22         $this->nro_vezes_acessado = $_nro_vezes_acessado;
23     }
24 }
25 ?>
```

**Questão:** É uma boa opção que o atributo `nro_vezes_acessado` seja estático?

**Resposta:** Não, pois sendo estático, todos os objetos da classe `UsuarioComum` terão acessado o mesmo número de vezes, gerando assim um erro semântico.

Para testar a classe criada, insira no `index.php` o seguinte código:



```
<?DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // put your code here
      // include_once 'usuario.php';
      // $usuario = new Usuario;//ou new Usuario();
      // $usuario->setUsuario("Seu nome");
      // echo "<br>".$usuario->getUsuario()."<br>";
      //
      // echo Usuario::$nro_usuarios_online."<br>";
      // $usuario2 = new Usuario;//ou new Usuario();
      // echo "<br>".Usuario::$nro_usuarios_online."<br>";
      //
      // Usuario::alteraNroUsuariosOnline();
      // -----HERANÇA-----
      include_once 'usuario_comum.php';
      $usuario = new UsuarioComum;
      $usuario->setUsuario("PET-SI");
      echo "<br>".$usuario->getUsuario()."<br>";
      echo "Usuarios online ".Usuario::$nro_usuarios_online."<br>";
    ?>
  </body>
</html>
```

Houve algum erro? Pense em como corrigir isso. Uma dica: `this` is the problem.

**Exercício:** Implemente uma classe `Endereco` que estenda a classe `Empregado`. O `Empregado` deve ter um ID, nome, sexo e um salário. A classe `Endereco` deve conter, além dos atributos da classe mãe, os atributos país, cidade, rua e número. As classes devem conter implementados em si os conceitos de encapsulamento (atributos privados e métodos de manipulação públicos).

## Interfaces, Classes e métodos abstratos

Classes abstratas são classes que não podem ser instanciadas, isto é, não é possível criar objeto da classe abstrata.

Caso um método de uma classe for abstrato, obrigatoriamente a classe será abstrata.

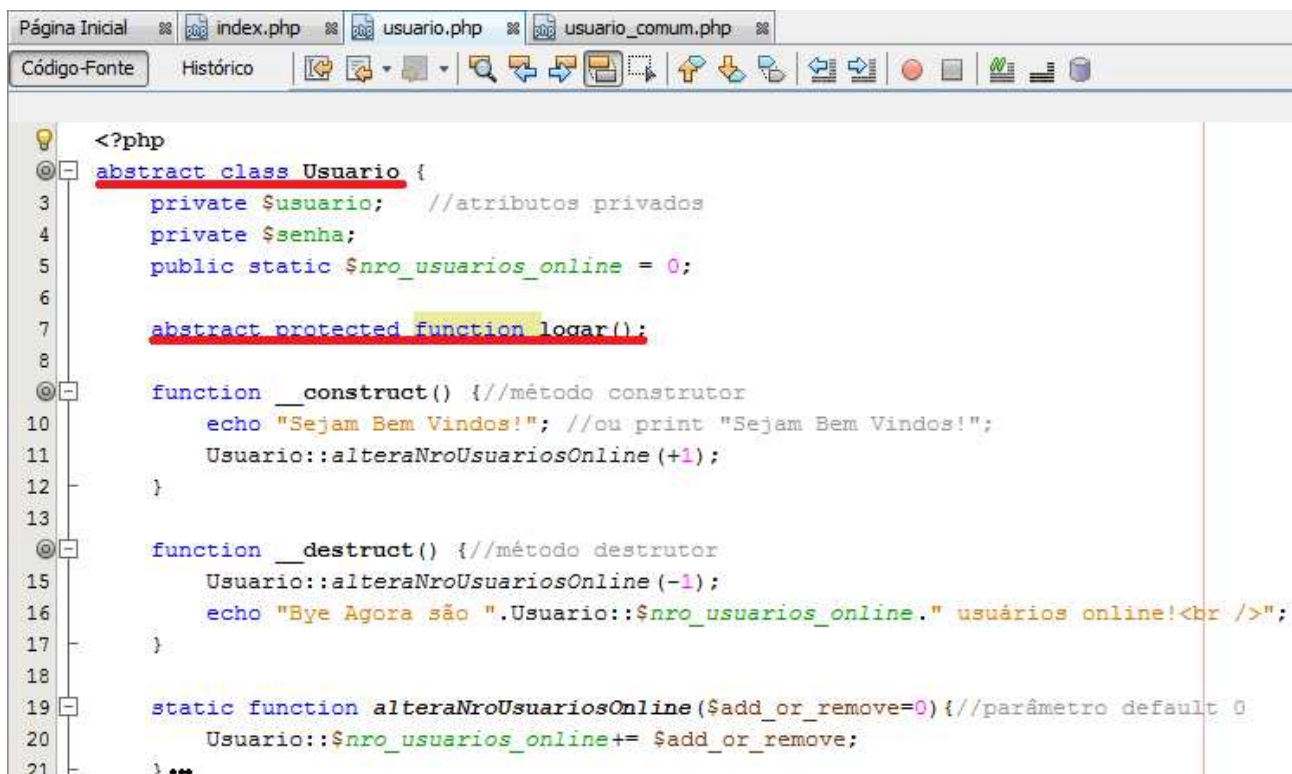
Um método abstrato possui apenas a sua assinatura (especificação) na classe abstrata, isto “obriga” que as subclasses (classe filha) o implemente. Ele ainda deve possuir visibilidade pública ou protegida.

Uma classe abstrata pode conter métodos abstratos (sem implementação) e métodos não abstratos.

Uma interface contém apenas métodos abstratos.

A classe filha define o método com mesma ou mais fraca visibilidade.

Para praticar, a classe `Usuario` será uma *abstract class*. Além disso, será adicionado o método abstrato `logar()`.



```
<?php
abstract class Usuario {
    private $usuario; //atributos privados
    private $senha;
    public static $nro_usuarios_online = 0;

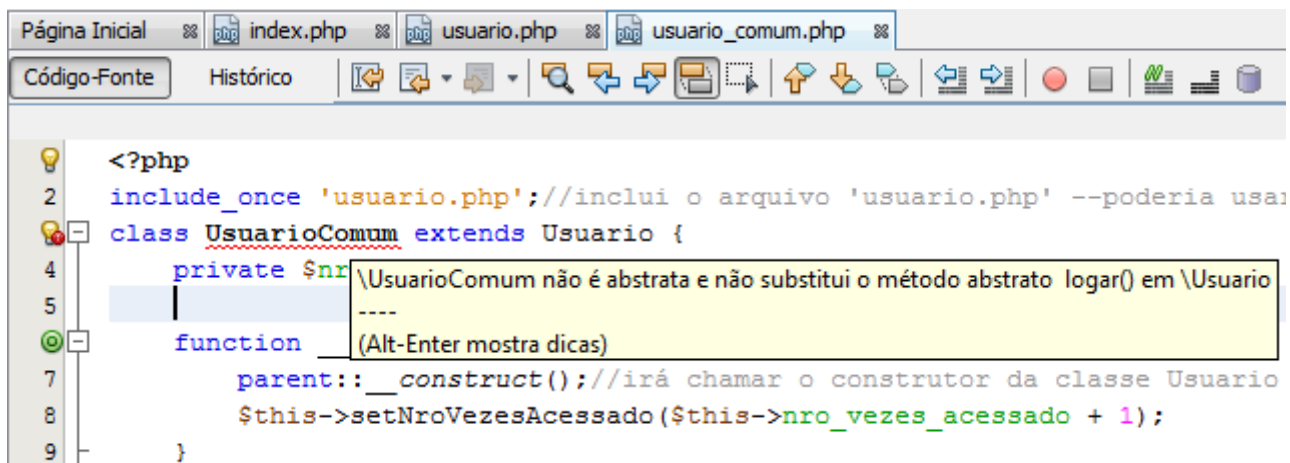
    abstract protected function logar();

    function __construct() { //método construtor
        echo "Sejam Bem Vindos!"; //ou print "Sejam Bem Vindos!";
        Usuario::alteraNroUsuariosOnline(+1);
    }

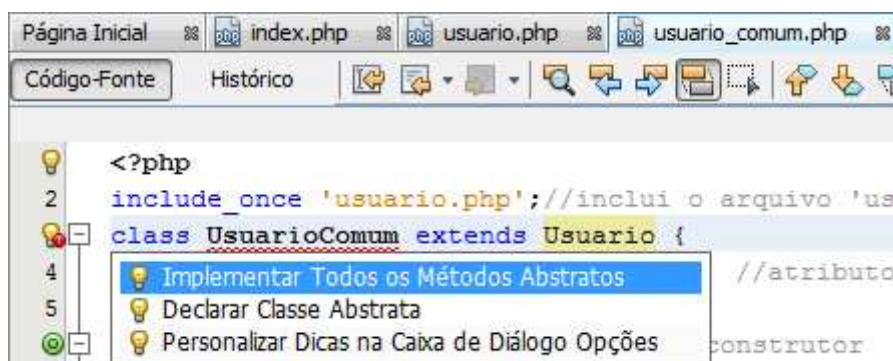
    function __destruct() { //método destrutor
        Usuario::alteraNroUsuariosOnline(-1);
        echo "Bye Agora são ".Usuario::$nro_usuarios_online." usuários online!<br />";
    }

    static function alteraNroUsuariosOnline($add_or_remove=0) { //parâmetro default 0
        Usuario::$nro_usuarios_online+= $add_or_remove;
    }
}
```

Devido à classe `UsuarioComum` ser uma subclasse da classe abstrata `Usuario`, será necessário que a classe `UsuarioComum` implemente o método `abstract logar()`.



Portanto, para implementar automaticamente o método logar no NetBeans, coloque o cursor do mouse na linha que está ocorrendo o erro (no caso linha 3) e tecle ALT+ENTER:



Selecione a alternativa Implementar Todos os Métodos Abstratos.

Teste a aplicação rodando o arquivo index.php. Em seguida, experimente descomentar as linhas 10 e 11 correspondentes na figura a seguir:

#### NOTA

Uma classe pode *estender* apenas uma classe.

```

<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // put your code here
      include_once 'usuario.php';
      $usuario = new Usuario; //ou new Usuario();
      $usuario->setUsuario("Seu nome");
      echo "<br>".$usuario->getUsuario()."<br>";
      //
      echo Usuario::$nro_usuarios_online."<br>";
      //
      $usuario2 = new Usuario; //ou new Usuario();
      //
      echo "<br>".Usuario::$nro_usuarios_online."<br>";
      //
      Usuario::alteraNroUsuariosOnline();
      //-----HERANÇA-----
      include_once 'usuario_comum.php';
      $usuario = new UsuarioComum;
      $usuario->setUsuario("PET-SI"); //usando métodos da Super classe
      echo "<br>".$usuario->getUsuario()."<br>";
      echo "Usuarios online ".Usuario::$nro_usuarios_online."<br>";
    ?>
  </body>
</html>

```

Note a importância do conhecimento dos conceitos teóricos na compreensão dos erros que acontecem na prática.

**Exercício:** Crie uma classe chamada UsuarioAdmin, que herda da classe Usuario.

Para exemplificar o uso de interfaces, crie uma interface chamada IUsuario, que contém o método logar (Usuario u). u é um objeto da classe Usuario.

```

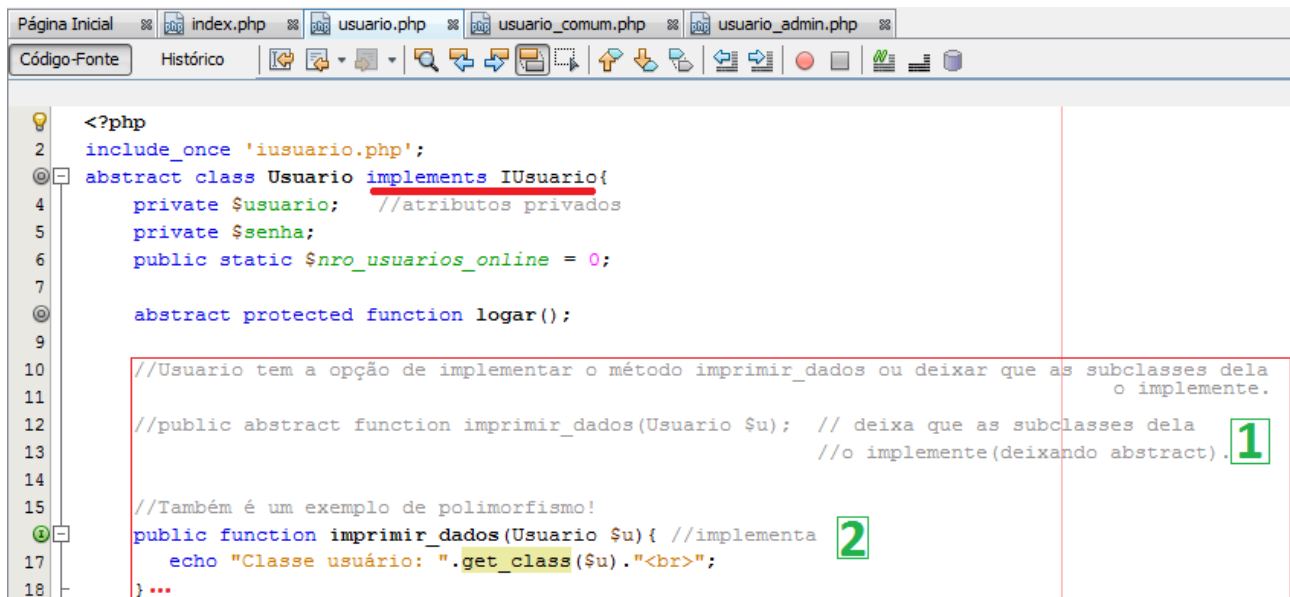
<?php
interface IUsuario{
  //Uma interface obriga a implementação dos métodos dela nas classes que implementam a interface IUsuario.
  public function imprimir_dados(Usuario $u); //ou simplesmente $u.
}

```



Quando é definido que o método `imprimir_dados (Usuario u)` recebe um objeto `u` da classe `Usuario`, significa que o método está sendo forçado a receber um objeto deste tipo. Isto é denominado *Indução de tipo*.

Feito isso, a imagem a seguir ilustra a classe abstrata `Usuario` implementando a interface `IUsuario`.



```
<?php
2 include_once 'iusuario.php';
3 abstract class Usuario implements IUsuario{
4     private $usuario; //atributos privados
5     private $senha;
6     public static $nro_usuarios_online = 0;
7
8     abstract protected function login();
9
10    //Usuario tem a opção de implementar o método imprimir_dados ou deixar que as subclasses dela
11    //o implemente.
12    //public abstract function imprimir_dados(Usuario $u); // deixa que as subclasses dela
13    //o implemente(deixando abstract). 1
14
15    //Também é um exemplo de polimorfismo!
16    public function imprimir_dados(Usuario $u){ //implementa 2
17        echo "Classe usuário: ".get_class($u)."<br>";
18    } ...
```

Veja na imagem, que existem duas maneiras de tratar o método `imprimir_dados(Usuario $u)` que foi definido em `IUsuario`:

- 1) O fato de a classe `Usuario` ser abstrata possibilita que ela deixe que esse método seja implementado pelas subclasses dela.
- 2) Implementa o método eliminando a necessidade de que as subclasses implementem os.

No arquivo `index.php` faça com que um objeto do tipo `Usuario` chame o método `imprimir_dados(Usuario $u)`.

```

5      <title></title>
6      </head>
7      <body>
8          <?php
9              //          include_once 'usuario.php';
10             //          $usuario = new Usuario;//ou new Usuario();
11             //          $usuario->setUsuario("Seu nome");
12             //          echo "<br>".$usuario->getUsuario()."<br>";
13             //
14             //          echo Usuario::$nro_usuarios_online."<br>";
15             //          $usuario2 = new Usuario;//ou new Usuario();
16             //          echo "<br>".Usuario::$nro_usuarios_online."<br>";
17             //
18             //          Usuario::alteraNroUsuariosOnline();
19             //-----HERANÇA-INTERFACE-----
20             include_once 'usuario_comum.php';
21             $usuario = new UsuarioComum;
22             $usuario->setUsuario("PET-SI");//usando métodos da Super classe
23             echo "<br>".$usuario->getUsuario()."<br>";
24             echo "Usuarios online ".Usuario::$nro_usuarios_online."<br>";
25             $usuario->imprimir_dados($usuario);//chamando método definido na interface
26         ?>
27     </body>
28 </html>

```

**NOTA** Classes abstratas e concretas podem implementar várias interfaces.

Interfaces podem definir constantes. Constantes são uma espécie de variável e cujo o seu valor não é alterado durante a execução do programa. Ex.:

**NOTA**

```

<?php
    interface IUsuario{
        //Uma interface obriga a implementação dos
        public function imprimir_dados(Usuario $u)
        const curso = "CURSO PHP - PET-SI";
    }
?>

```

Interfaces podem “extender” outras classes:

```

<?php interface IFoo
{
    public function Foo();
}

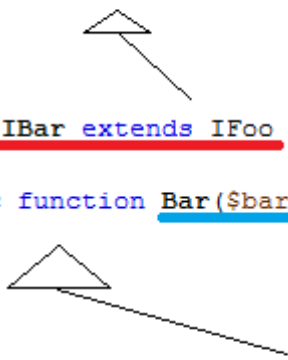
interface IBar extends IFoo
{
    public function Bar($bar);
}

class FooBar implements IBar
{
    public function Foo()
    {
        echo "Foo";
    }

    public function Bar($bar)
    {
        echo $bar;
    }
}

$IFoo = new FooBar;
$IFoo->Foo();
$IFoo->Bar("teste");
?>

```



**Exercício:** Crie uma interface no projeto que declara pelo menos um método. Em seguida, mostre um exemplo de como utilizá-la.

## Traits

Trait é mais um mecanismo de reuso de código. O PHP não implementa herança múltipla, por isso, uma maneira de reduzir esta limitação é por meio do uso de traits. Desta maneira, o desenvolvedor é capaz de reusar um conjunto de métodos situados em classes independentes que estão em outra hierarquia.

Um trait é semelhante a uma classe, porém, com um grupo de funcionalidades. Não é possível instanciá-lo como uma classe concreta.



```

<?php

trait Hello {
    public function sayHello() {
        echo "Olá";
    }
}

trait World {
    public function sayWorld() {
        echo "Mundo";
    }
}

class View {
    use Hello, World;
}

$view = new View();
$view->sayHello();
$view->sayWorld();
// exibe: Olá Mundo

?>

```

Neste exemplo, utilizando traits é possível que a classe `View` use métodos dos traits `Hello` e `World`, o que seria impossível utilizando herança simples.

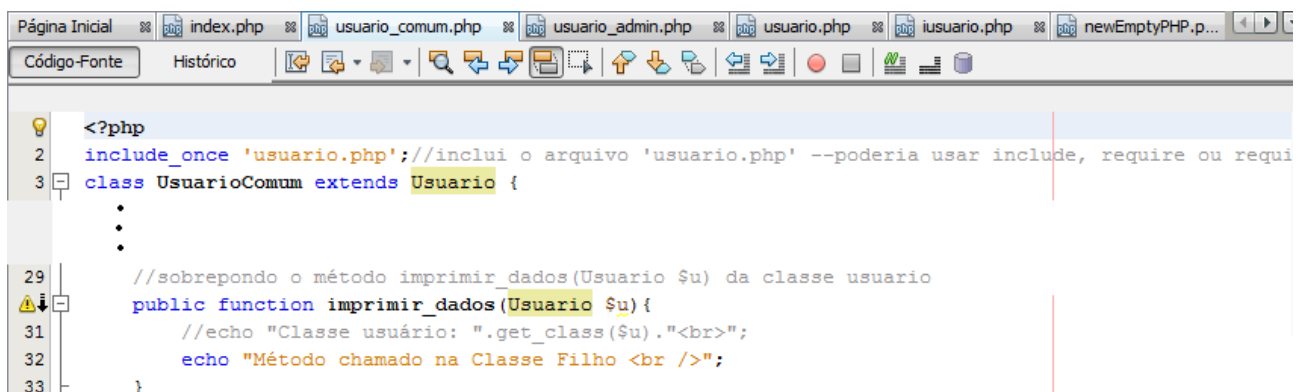
É possível realizar outras coisas com traits, como por exemplo, mudar a visibilidade de um método. Para saber mais sobre traits consulte: [http://www.php.net/manual/pt\\_BR/language.oop5.traits.php](http://www.php.net/manual/pt_BR/language.oop5.traits.php)

#### DICA

Utilize vírgulas para separar os traits a serem utilizados, como mostrado no exemplo da imagem (`use Hello, World;`).

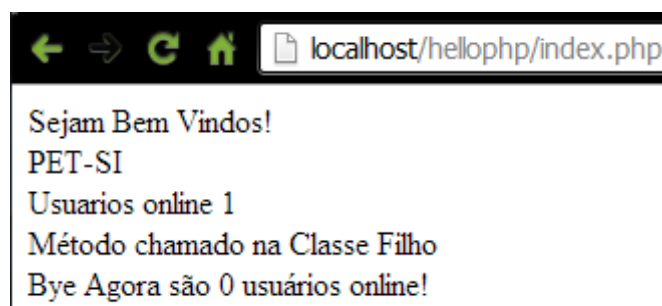
## Sobreposição (Overriding)

É possível sobrepor métodos no PHP de maneira que o método em uma subclasse seja invocado ao invés do método de sua superclasse. Veja na imagem a seguir um exemplo de sobreposição do método `imprimir_dados(Usuario $u)` da classe `usuarioComum`. Este método também é considerado polimórfico, já que ele é herdado e sobreposto da superclasse e ainda, cada subclasse que implemente este método poderá dar um comportamento diferente para o mesmo método.



```
<?php
2 include_once 'usuario.php'; //inclui o arquivo 'usuario.php' --poderia usar include, require ou requi
3 class UsuarioComum extends Usuario {
    .
    .
    .
29 //sobrepondo o método imprimir_dados(Usuario $u) da classe usuario
30 public function imprimir_dados(Usuario $u){
31     //echo "Classe usuário: ".get_class($u)."<br>";
32     echo "Método chamado na Classe Filho <br />";
33 }
```

Execute o *index.php*.



```
localhost/hellophp/index.php

Sejam Bem Vindos!
PET-SI
Usuarios online 1
Método chamado na Classe Filho
Bye Agora são 0 usuários online!
```

## A palavra-chave “final”

A palavra-chave `final` previne que classes filhas sobreponham um método mais acima na hierarquia. Deste modo, torna-se impossível criar métodos de mesmo nome derivados da classe que utiliza a palavra-chave `final`.

```
<?php
class ClasseBase {
    public function teste() {
        echo "ClasseBase::teste() chamado\n";
    }

    final public function maisTeste() {
        echo "ClasseBase::maisTeste() chamado\n";
    }
}

class ClasseFilha extends ClasseBase {
    public function maisTeste() {
        echo "ClasseFilha::maisTeste() chamado\n";
    }
}
?>
```

O código anterior resulta em um erro fatal pois ClasseFilha tenta realizar a sobreposição do método maisTeste de ClasseBase, que por sua vez está definido utilizando a palavra-chave final, que não permite que métodos de sobreposição sejam criados para o método inicial.

Assim como métodos, as classes que utilizam a palavra-chave final também não permitem que outras classes herdem seus atributos e métodos, o que resultaria em erro, como o código do exemplo abaixo:

```
<?php
final class ClasseBase {
    public function teste() {
        echo "ClasseBase::teste() chamado\n";
    }

    final public function maisTeste() {
        echo "ClasseBase::maisTeste() chamado\n";
    }
}

class ClasseFilha extends ClasseBase {

}
?>
```

### Pratique:



1. Na classe UsuarioComum faça a sobreposição do método imprimir\_dados(Usuario \$u).
2. Na classe Usuario coloque a palavra reservada final no método imprimir\_dados(Usuario \$u). Certifique de que entendeu o conceito e em seguida deixe o método imprimir\_dados(Usuario \$u) sem a palavra final.

## Clonagem de objetos

Clonagem de objetos é utilizada quando se deseja criar uma cópia de um objeto, ao invés, de ter a referência dele propriamente dita. A clonagem de um objeto é feita através da palavra-chave clone. Ex.: \$copia\_do\_objeto = clone \$objeto;.

Veja o exemplo a seguir que ilustra o arquivo *index.php* mostra diferença entre usar clonagem de objetos e usar a cópia da referência do objeto:

```

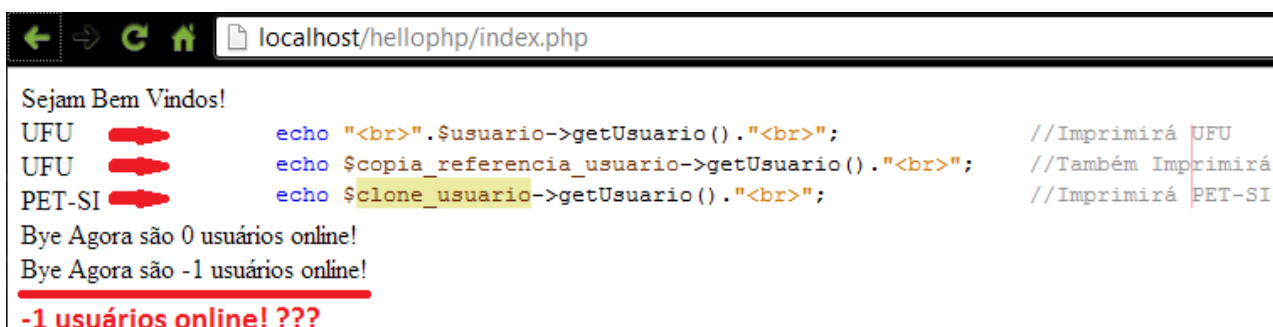
include_once 'usuario_comum.php';
$usuario = new UsuarioComum;
$usuario->setUsuario("PET-SI");//usando métodos da Super classe
//echo "<br>".$usuario->getUsuario()."<br>";
//echo "Usuarios online ".Usuario::$nro_usuarios_online."<br>";
//$usuario->imprimir_dados($usuario);//chamando método definido na interface

//clonagem de objetos
$copia_referencia_usuario = $usuario;
$clone_usuario = clone $usuario;

$usuario->setUsuario("UFU");

echo "<br>".$usuario->getUsuario()."<br>";           //Imprimirá UFU
echo $copia_referencia_usuario->getUsuario()."<br>"; //Também Imprimirá UFU!!!
echo $clone_usuario->getUsuario()."<br>";           //Imprimirá PET-SI

```



```

localhost/hellophp/index.php

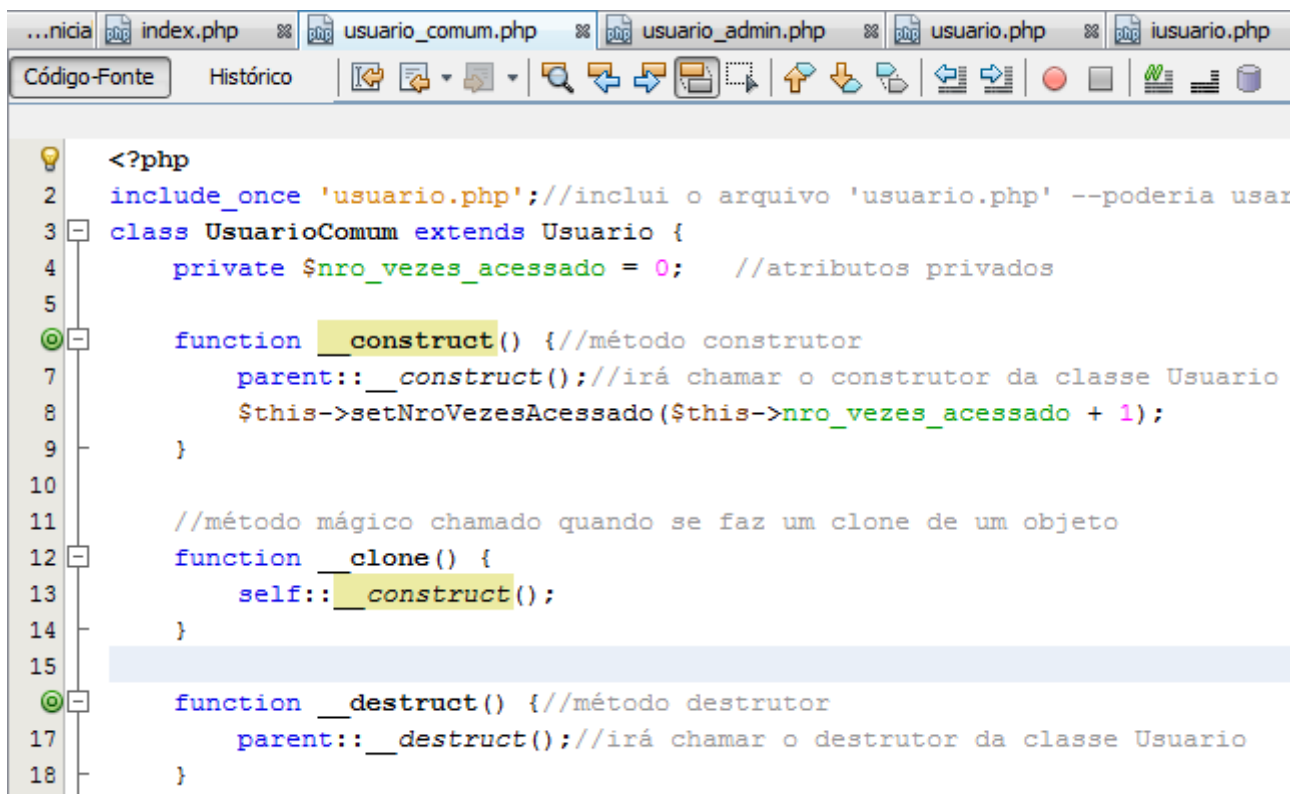
Sejam Bem Vindos!
UFU
UFU
PET-SI
Bye Agora são 0 usuários online!
Bye Agora são -1 usuários online!
-1 usuários online! ???

```

## CUIDADO

Veja que na figura anterior imprimiu “-1 usuários online!”. Isto ocorre porque o clone não invoca o método `__construct` do objeto `UsuarioComum`, que adicionaria em 1 o número de usuários, porém invoca o método `__destruct` que decrementa em 1 o número de usuários. Para contornar este problema, uma solução será mostrada a seguir.

Na classe UsuarioComum foi implementado o método mágico `__clone` de maneira que ele invoca o método construtor.



```
<?php
2  include_once 'usuario.php'; //inclui o arquivo 'usuario.php' --poderia usar
3  class UsuarioComum extends Usuario {
4      private $nro_vezes_acessado = 0; //atributos privados
5
6      function __construct() { //método construtor
7          parent::__construct(); //irá chamar o construtor da classe Usuario
8          $this->setNroVezesAcessado($this->nro_vezes_acessado + 1);
9      }
10
11     //método mágico chamado quando se faz um clone de um objeto
12     function __clone() {
13         self::__construct();
14     }
15
16     function __destruct() { //método destrutor
17         parent::__destruct(); //irá chamar o destrutor da classe Usuario
18     }
```

## Comparação de objetos

Existem duas maneiras de comparar objetos:

- (`$objeto1 == $objeto2`) e (`!=`): verifica se dois objetos são da mesma classe e possuem os mesmos atributos e valores.
- (`$objeto1 === $objeto2`) e (`!==`): verifica se dois objetos são da mesma classe, possuem os mesmos atributos e valores e representam a **mesma instância da classe**.

Alterando o arquivo *index.php* para realizar a comparação entre o objeto `$usuario`, `$clone_usuario` e `$copia_referencia_usuario`.

```
Página Inicial  index.php  usuario_admin.php  usuario.php  iusuario.php  newEmptyPHP.php  JavaApplication7.java
Código-Fonte  Histórico

html  body
22      $usuario->setUsuario("PET-SI");//usando métodos da Super classe
23      //echo "<br>".$usuario->getUsuario()."<br>";
24      //echo "Usuarios online ".$usuario::$nro_usuarios_online."<br>";
25      //$usuario->imprimir_dados($usuario);//chamando método definido na interface
26
27      //clonagem de objetos
28      $copia_referencia_usuario = $usuario;
29      $clone_usuario = clone $usuario;
30
31      //      $usuario->setUsuario("UFU");
32      //      echo "<br>".$usuario->getUsuario()."<br>";           //Imprimirá UFU
33      //      echo $copia_referencia_usuario->getUsuario()."<br>"; //Também Imprimirá UFU!!!
34      //      echo $clone_usuario->getUsuario()."<br>";           //Imprimirá PET-SI
35
36      //comparação de objetos
37      1  if ($usuario == $copia_referencia_usuario)
38          echo "<br>Usuário tem as mesmas propriedades da copia de sua referência."<br>";
39      else
40          echo "Usuário não tem as mesmas propriedades da copia de sua referência."<br>";
41      2  if ($usuario == $clone_usuario)
42          echo "Usuário tem as mesmas propriedades de seu clone."<br>";
43      else
44          echo "Usuário não tem as mesmas propriedades da copia de seu clone."<br>";
45
46      3  if ($usuario === $copia_referencia_usuario)
47          echo "Usuário é a mesma instância da copia de sua referência."<br><br>";
48      else
49          echo "Usuário não é a mesma instância da copia de sua referência."<br><br>";
50      4  if ($usuario === $clone_usuario)
51          echo "Usuário é a mesma instância do seu clone."<br>";
52      else
53          echo "Usuário não é a mesma instância do seu clone."<br>";
```

O resultado será para cada sequência de if e else:

- 1- Usuário tem as mesmas propriedades da cópia de sua referência
- 2- Usuário não tem as mesmas propriedades da cópia de seu clone
- 3- Usuário é a mesma instância da cópia de sua referência
- 4- Usuário não é a mesma instância do seu clone

Isso mostra mais uma vez a diferença entre clonar um objeto e copiar a referência de um objeto.

## **Créditos:**

**Higor Ernandes Ramos Silva**  
**Leonardo Pimentel Ferreira**