# PHP 5

# Orientado a Objetos

# Introdução ao PHP 5

Com as primeiras 2 versões de PHP, PHP 3 e PHP 4, conseguiram uma plataforma potente e estável para a programação de páginas do lado do servidor. Estas versões serviram muito de ajuda para a comunidade de desenvolvedores, tornando possível que PHP seja a linguagem mais utilizada na web para a realização de páginas avançadas.

Entretanto, ainda existiam pontos negros no desenvolvimento de PHP que trataram de solucionar com a versão 5, aspectos que fizeram falta na versão 4, quase desde o dia de seu lançamento. Referimo-nos principalmente à programação orientada a objetos (POO) que, apesar de estar suportada a partir de PHP3, só implementava uma parte muito pequena das características deste tipo de programação.

*Nota: A orientação a objetos é uma maneira de programar que trata de modelar os processos de programação de uma maneira próxima à realidade: tratando a cada componente de um programa como um objeto com suas características e funcionalidades. Podemos ver uma pequena introdução no artigo O que é a programação orientada a objetos.*

O principal objetivo de PHP5 foi melhorar os mecanismos de POO para solucionar as carências das versões anteriores. Um passo necessário para conseguir que PHP seja uma linguagem apta para todo tipo de aplicações e meios, inclusive os mais exigentes.

# Instalação de PHP5 com WAMP5

**Existe um pacote de instalação chamado WAMP5 que pode instalar em conjunto Apache, PHP 5, MySQL e PHPMyAdmin. Neste capítulo, ainda nos referimos à forma de instalação de WAMP5 e outras opções para aumentar as possibilidades do pacote.**

Existe uma maneira de começar a utilizar PHP5 em Windows sem ter que sofrer as complicações típicas da instalação dos servidores necessários para programar em PHP. Trata-se de instalar um pacote chamado WAMP, que permite instalar e configurar em um só processo o servidor Apache, a base de dados MySQL e o módulo de programação em PHP versão 5.

WAMP é um sistema indicado para os usuários que não têm instalado no sistema nenhum dos programas necessários para programar em PHP (Apache, PHP e MySQL), já que realiza uma instalação completa e desde zero. Mas também podem utilizar este programa os usuários que dispõem de Apache, PHP e/ou MySQL em seu sistema. Em cujo caso, simplesmente se realizará outra cópia das aplicações em um diretório distinto, que à princípio, não tem porque interferir com as outras instalações alojadas em nosso computador.

**Programas que contem WAMP5**

O software que se instala com WAMP5 contem os seguintes servidores e programas:

Apache 2.2.6. O servidor de páginas web mais difundido do mercado. Embora a última versão deste servidor seja Apache 2, instala-se uma versão anterior que é mais estável. Existe um Add-on que permite substituir a versão 1.3.31 de Apache pela última versão.

PHP 5.2.5. O motor renovado da linguagem.

MySQL 5.0.45. A base de dados mais difundida para utilizar com PHP.

PHPmyadmin. Um software que permite administrar uma base de dados através de uma interface web.

SQLitemanager. Um sistema para administrar uma base de dados a partir de sentenças SQL.

Instalação de WAMP

*Nota: A versão dos software acima são referentes até a data de criação desta apostila.*

A instalação se realiza através de um executável Windows onde podem se introduzir poucas configurações, apenas o diretório onde desejarmos que se instalem os programas. Depois do processo de instalação foram criados dois serviços com o servidor web e o de base de dados:

Serviço wampapache: Relacionado com o servidor Apache.

Serviço wampmysql: Relacionado com a base de dados MySQL.

Ademais, dentro do diretório onde tivermos instalado WAMP5 terá sido criado uma pasta chamada "www", que corresponde com o diretório de publicação, ou seja, o lugar onde devem ser colocadas as

páginas web.

Durante a instalação também devemos decidir se desejamos que WAMP5 se inicie automaticamente ao ligar o computador ou se desejamos que seu funcionamento se realize manualmente.

**Funcionamento dos servidores**

Quando instalamos WAMP5 se cria um grupo de programas chamado WampServer, onde poderemos encontrar uma opção que põe "Start Wampserver", que será necessário executar se não tivermos selecionado que o servidor se inicie automaticamente.

Uma vez o WampServer em funcionamento obteremos um ícone na barra de tarefas com a forma parecida a de um marcador de velocidade. Se clicarmos sobre esse ícone, abrirá um menu com opções variadas para providenciar os serviços relacionados com o pacote.

Podemos provar se os serviços estão funcionando perfeitamente acessando à página de início do servidor, escrevendo na barra de endereços de nosso navegador algo como http://localhost/.

Então deverá aparecer uma página com vários links às distintas ferramentas instaladas com WAMP5, além de algumas páginas de proba de PHP.

**Add-ons**

Existem vários acréscimos que podem ser instalados com WAMP, para ampliar as possibilidades do pacote. Por exemplo, podemos instalar um add-on para permitir que WAMP trabalhe com PHP5 ou com PHP4, criando una nova opção no menu de WAMP5 que permite mudar de uma versão a outra de PHP.
Existem outros acréscimos disponíveis:
Instalar ActiveState Perl em nosso sistema, para permitir a execução de CGI.
Atualizar a versão de Apache 2.
Instalar Zend Optimizer, para melhorar o comportamento em tempo de execução de PHP.
Por último, o add-on que instala Webalizer, um sistema para obter estatísticas de uso do servidor web.
Pode-se obter mais informação deste sistema e opções para download na página http://www.en.wampserver.com/

# Modelo de orientação a objetos em PHP 5

**Como PHP 5 trabalha com a orientação a objetos. Lista das novidades em relação aos objetos em versões anteriores.**

Um dos problemas mais básicos das versões anteriores de PHP era a clonagem de objetos, que se realizava ao atribuir um objeto a outra variável ou ao passar um objeto por parâmetro em uma função. Para resolver este problema PHP5 usa os manipuladores de objetos (Object handles), que são uma espécie de ponteiros que apontam os espaços de memória onde residem os objetos. Quando se atribui um manipulador de objetos ou se passa como parâmetro em uma função, se duplica o próprio object handle e não o objeto em si.

*Nota: Também pode-se realizar uma clonagem de um objeto, para obter uma cópia exata, mas que não é o próprio objeto. Para isso, utilizamos uma nova instrução chamada " clone", que veremos mais adiante.*

# Algumas características do trabalho com POO em PHP 5

**Vejamos a seguir uma pequena lista das novas características da programação orientada a objetos (POO) em PHP5. Não vamos descrever exaustivamente cada característica. Faremos isso mais adiante neste mesmo manual.**

**1.- Nomes fixos para os construtores e destrutores**
Em PHP 5 temos que utilizar nomes pré-definidos para os métodos construtores e destrutores (Os que se encarregam de resumir as tarefas de iniciação e de destruição dos objetos. Agora se chamam

__construct e __destruct.

**2.- Acceso public, private e protected a propriedades e métodos**

A partir de agora podemos utilizar os modificadores de acesso habituais da POO. Estes modificadores servem para definir que métodos e propriedades das classes são acessíveis desde cada meio.

**3.- Possibilidade de uso de interfaces**

As interfaces se utilizam na POO para definir um conjunto de métodos que implementa uma classe. Uma classe pode implementar várias interfaces ou conjuntos de métodos. Na prática, o uso de interfaces é utilizado muitas vezes para suprir a falta de herança múltipla de linguagens como PHP ou Java. Explicaremos isto com detalhe mais adiante.

**4.- Métodos e classes final**

Em PHP 5 pode-se indicar que um método é "final". Com isso, não se permite sobrescrever esse método, em uma nova classe que o herde. Se a classe é "final", o que se indica é que esta classe não permite ser herdada por outra classe.

**5.- Operador instanceof**

Utiliza-se para saber se um objeto é uma instância de uma classe determinada.

**6.- Atributos e métodos static**

Em PHP5 podemos fazer uso de atributos e métodos "static". São as propriedades e funcionalidades as quais se pode acessar a partir do nome de classe, sem a necessidade de haver instanciado um objeto de tal classe.

**7.- Classes e métodos abstratos**

Também é possível criar classes e métodos abstratos. As classes abstratas não se podem instanciar, costumam ser utilizadas para herda-las de outras classes que não têm porque serem abstratas. Os métodos abstratos não podem ser chamados, utilizam-se mais para serem herdados por outras classes, onde não têm porque serem declarados abstratos.

**8.- Constantes de classe**

Pode-se definir constantes dentro da classe. Logo, pode-se acessar tais constantes através da própria classe.

**9.- Funções que especificam a classe que recebem por parâmetro**

Agora podem se definir funções e declarar que devem receber um tipo específico de objeto. No caso de que o objeto não seja da classe correta, se produz um erro.

**10.- Função __autoload**

É habitual que os desenvolvedores escrevam um arquivo por cada classe que realizam, como técnica para organizar o código das aplicações. Por essa razão, às vezes é fatigante realizar os incluis de cada um dos códigos das classes que se utilizam em um script. A função __autoload serve para tentar incluir o código de uma classe que se necessite, e que não tenha sido declarada ainda no código que está sendo executada.

**11.- Clonagem de objetos**

Se desejarmos, podemos realizar um objeto a partir da cópia exata de outro objeto. Para isso, utiliza-se a instrução"clone". Também pode se definir o método __clone para realizar tarefas associadas coma clonagem de um objeto.

# Classes em PHP 5

**Vemos o que é uma classe, como podemos defini-la e instanciá-la.**

**Classes em PHP 5**

As classes em Programação orientada a objetos (POO) são definições dos elementos que formam um sistema, neste caso, definições dos objetos que vão intervir em nossos programas.

Um objeto se define indicando que propriedades e funcionalidades têm. Justamente essas declarações são o que é uma classe. Quando se faz uma classe simplesmente se especifica que propriedades e funcionalidades têm. Por exemplo, um homem poderia ter como propriedades o nome ou a idade e como funcionalidades, comer, mover-se ou estudar.

Na classe homem, declararíamos dois atributos: a idade ou o nome, que seriam como duas variáveis. Também deveríamos criar três métodos, com os procedimentos a seguir para que o homem possa comer, mover-se ou estudar. Estes métodos se definem declarando funções dentro da classe.

O código para definir uma classe pode ser visto a seguir:

```
class home{
    var $nome;
    var $idade;

    function comer($comida){
       //aqui o código do método
    }

    function moverse($destino){
       //aqui o código do método
    }

    function estudar($diciplina){
       //aqui o código do método
    }
}
```

Poderá se comprovar que este código não difere em nada das versões anteriores de PHP, que já suportavam certas características da POO. Esta situação mudará explorando um pouco mais as características mais avançadas de PHP 5, que implicarão melhoras que não estavam presentes nas versões anteriores

Instanciar objetos a partir de classes

Vimos que uma classe é somente uma definição. Se quisermos trabalhar com as classes devemos instanciar objetos, processo que consiste em gerar um exemplar de uma classe.

Por exemplo, temos a classe homem anterior. Com a classe em si não podemos fazer nada, mas podemos criar objetos homem a partir dessa classe. Cada objeto homem terá umas características próprias, como a idade ou o nome. Ademais poderá desempenhar umas funções como comer ou mover-se, agora também, cada um comerá ou se moverá por sua própria conta quando lhe for solicitado, sem interferir à princípio, com o que possa estar fazendo outro homem.

Aproveitando, vamos ver como se gerariam dois homens, ou seja, como se instanciariam dois objetos da classe homem. Para isso, utilizamos o operador new.

```
$carol = new nome;
$jorge = new nome;
```

**Conclusão**

É importante se dar conta da diferença entre um objeto e uma classe. A classe é uma definição de umas características e funcionalidades, algo abstrato que se concretiza com a instanciação de um objeto de tal classe.

Um objeto já tem propriedades, com seus valores concretos, e podem ser passadas mensagens (chamar aos métodos) para que façam coisas.

# Construtores em PHP 5

**Vamos ver o que é um construtor e para que serve, além de um exemplo simples de uma classe que define um construtor.**

Os construtores se encarregam de resumir as ações de iniciação dos objetos. Quando instanciamos um objeto, temos que realizar vários passos em sua iniciação, por exemplo, dar valores a seus atributos e isso é o que se encarrega o construtor. Os construtores podem receber dados para iniciar os objetos como se deseje em cada caso.

A sintaxe para a criação de construtor varia em relação a do PHP 3 e 4, pois deve se chamar com um nome fixo: `__construct`. (São dois hífens baixos(underline), antes da palavra "construct")

A longo dos exemplos deste manual vamos ir criando um código para gestão de um vídeo club. Vamos começar definindo uma classe cliente, que utilizaremos logo em nosso programa.

```
class cliente{
    var $nome;
    var $numero;
    var $filmes_alugados;

    function __construct($nome,$numero){
       $this->nome=$nome;
       $this->numero=$numero;
       $this->filmes_alugados=array;
    }

    function dame_numero{
        return $this->numero;
    }
}
```

O construtor nesta classe recebe o nome e número que atribuir ao cliente, que introduz logo em suas correspondentes propriedades. Ademais inicia o atributo filmes_alugados como um array, neste caso vazio porque ainda não tem nenhum filme em seu poder.

Logo, criamos um método muito simples para poder utilizar o objeto. Vamos ver umas ações simples para ilustrar o processo de instanciação e utilização dos objetos.

```
//instanciamos dois objetos cliente
$cliente1 = new cliente("Pedro", 1);
$cliente2 = new cliente("Roberto", 564);

//mostramos o numero de cada cliente criado
"O identificador do cliente 1 é: " . $cliente1->dame_numero;
"O identificador do cliente 2 é: " . $cliente2->dame_numero;
```

Este exemplo obteria esta saída como resultado de sua execução:

O identificador do cliente 1 é: 1
O identificador do cliente 2 é: 564

# Destrutores em PHP 5

**Explicação dos destrutores em PHP5 e exemplos de funcionamento.**

Os destrutores são funções que se encarregam de realizar as tarefas que se necessita executar quando um objeto deixa de existir. Quando um objeto já não está referenciado por nenhuma variável, deixa de ter sentido que esteja armazenado na memória, portanto, o objeto deve ser destruído para liberar seu espaço. No momento de sua destruição, a função se chama destrutor, que pode realizar as tarefas que o programador estime oportuno realizar.

A criação do destrutor é opcional. Somente devemos criá-lo, se desejarmos fazer alguma coisa quando um objeto se elimine da memória.

O destrutor é como qualquer outro método da classe, embora deve se declarar com um nome fixo: `__destruct`.

No código seguinte veremos um destrutor em funcionamento. Embora a ação que se realiza ao destruir o objeto não é muito útil, pode nos servir bem para ver como trabalha.

```
class cliente{
    var $nome;
    var $numero;
    var $filmes_alugados;

    function __construct($nome,$numero){
       $this->nome=$nome;
       $this->numero=$numero;
       $this->filmes_alugados=array;
    }

    function __destruct{
       "<br>destruido: " . $this->nome;
    }

    function dame_numero{
       return $this->numero;
    }
}

//instanciamos dois objetos cliente
$cliente1 = new cliente("Pedro", 1);
$cliente2 = new cliente("Roberto", 564);

//mostramos o numero de cada cliente criado
"O identificador do cliente 1 é: " . $cliente1->dame_numero;
"<br>O identificador do cliente 2 é: " . $cliente2->dame_numero;
```

Este código é igual que o anterior. Somente acrescentamos o destrutor, que imprime uma mensagem na tela com o nome do cliente que foi destruído. Depois de sua execução obteríamos a seguinte saída.

O identificador do cliente 1 é: 1
O identificador do cliente 2 é: 564
destruído: Pedro
destruído: Roberto

Como vimos, antes de acabar o script, libera-se o espaço na memória dos objetos, com o qual se executa o destrutor e aparece a correspondente mensagem na página.

Um objeto pode ficar sem referências e, portanto, ser destruído, por muitas razões. Por exemplo, o objeto pode ser uma variável local de uma função e ao finalizar a execução dessa função local deixaria de ter validade, sendo então destruído.

O código seguinte ilustra como uma variável local a qualquer âmbito (por exemplo, local a uma função), se destrói quando esse âmbito foi finalizado.

```
function cria_cliente_local{
    $cliente_local = new cliente("sou local", 5);
}
cria_cliente_local
```

A função simplesmente cria uma variável local que contem a instanciação de um cliente. Quando a função se acaba, a variável local deixa de existir, e então, chama-se ao destrutor definido para esse objeto.

*Nota: Também podemos nos desfazer de um objeto sem a necessidade de acabar com o âmbito onde foi criado. Para isso, temos a função unset que recebe uma variável e a elimina da memória. Quando se perde uma variável que contem um objeto e esse objeto deixa de ter referências, elimina-se ao objeto e chama-se ao destrutor.*

# Modificadores de acesso a métodos e propriedades em PHP

**São os Public, Protected e Private, que se podem conhecer porque já se utilizam e outras linguagens orientados a objetos.**

Veremos neste capítulo os novos modificadores de acesso aos métodos e atributos dos objetos que foram incorporados em PHP 5. Estes modificadores de acesso são os conhecidos public, protected e private, que já dispõem de outras linguagens como Java.

Um dos princípios da programação orientada a objetos é o encapsulamento, que é um processo no qual se ocultam as características internas de um objeto àqueles elementos que não têm porque conhece-las. Os modificadores de acesso servem para indicar as permissões que terão outros objetos para acessar a seus métodos e propriedades.

**Modificador public**

É o nível de acesso mais permissivo. Serve para indicar que o método ou atributo da classe é público. Neste caso pode-se acessar a este atributo, para visualizá-lo ou edita-lo, por qualquer outro elemento de nosso programa. É o modificador que se aplica se não se indica outra coisa.

Vejamos um exemplo de classe onde declaramos como public seus elementos, um método e uma propriedade. Trata-se da classe "dado", que tem um atributo com sua pontuação e um método para tirar o dado o obter uma nova pontuação aleatória.

```
class dado{
    public $pontos;

    function __construct{
       srand((double)microtime*1000000);
    }

    public function tirar{
       $this->pontos=$randval = rand(1,6);
    }
}

$meu_dado = new dado;

for ($i=0;$i<30;$i++){
    $meu_dado->tirar;
    "<br>Saiu " . $meu_dado->pontos . " pontos";
}
```

Vemos a declaração da classe dado, e logo umas linhas de código para ilustrar seu funcionamento. No exemplo se realiza um loop 30 vezes, nas quais se tira o dado e se mostra a pontuação que se obteve.

Como o atributo $pontos e o método tirar são públicos, podemos acessá-los de fora do objeto, ou que é o mesmo, de fora do código da classe.

**Modificador private**

É o nível de acesso mais restritivo. Serve para indicar que essa variável somente vai poder ser acessada pelo próprio objeto, nunca de fora. Se tentarmos acessar um método ou atributo declarado private de fora do próprio objeto, obteremos uma mensagem de erro indicando que não é possível a este elemento.

Se no exemplo anterior tivéssemos declarado private o método e a propriedade da classe dado, teríamos recebido uma mensagem de erro.

Aqui temos outra possível implementação da classe dado, declarando como private o atributo pontos e o método tirar.

```
class dado{
    private $pontos;

    function __construct{
        srand((double)microtime*1000000);
    }

    private function tirar{
        $this->pontos=$randval = rand(1,6);
    }

    public function dar_nova_pontuacao{
        $this->tirar;
        return $this->pontos;
    }
}

$meu_dado = new dado;

for ($i=0;$i<30;$i++){
    "<br>Han salido " . $meu_dado->dar_nova_pontuacao . " pontos";
}
```

Tivemos que criar um novo método público para operar com o dado, porque se é tudo privado não há forma de fazer uso dele. O mencionado método é dar_nova_pontuacao, que realiza a ação de tirar o dado e devolver o valor que saiu.

**Modificador protected**

Este indica nível de acesso médio e um pouco mais especial que os anteriores. Serve para que o método ou o atributo seja público dentro do código da própria classe e de qualquer classe que herde daquela onde está o método ou propriedade protected. É privado e não acessível de qualquer outra parte. Ou seja, um elemento protected é público dentro da própria classe e em suas heranças.

Mais adiante explicaremos a herança e poderemos oferecer exemplos com o modificador protected.

**Conclusão**

Muitas vezes o próprio desenvolvedor é o que fixa seu critério na hora de aplicar os distintos modificadores de acesso a atributos e métodos. Pouca proteção implica que os objetos percam seu encapsulamento e com isso, uma das vantagens da POO. Uma proteção maior pode tornar mais trabalhoso gerar o código do programa, mas em geral, é aconselhável.

# A herança em PHP5

**Explicamos a herança em PHP 5, um processo pelo qual os objetos podem herdar as características de outros, de modo que se podem fazer objetos especializados, baseados em outros mais gerais.**

A herança é um dos mecanismos fundamentais da programação orientada a objetos. Por meio da herança, podem se definir classes a partir da declaração de outras classes. As classes que herdam incluem os métodos como as propriedades da classe a partir da qual estão definidos.

Por exemplo, pensemos na classe "veículo". Esta classe geral pode incluir as características gerais de todos os veículos (atributos da classe), como a matrícula, ano de fabricação e potência. Ademais, incluirá algumas funcionalidades (métodos da classe) como poderiam ser, ligar ou mover.

Agora também, na prática existem vários tipos de veículos, como os carros, os ônibus e os caminhões. Todos eles têm umas características comuns, que foram definidas na classe veículo. Ademais,

terão uma série de características próprias do tipo de veículo, que, à princípio, não têm outros tipos de veículos. Por exemplo, os caminhões podem ter uma carga máxima permitida ou os ônibus um número de lugares disponíveis. Da mesma forma, as classes mais específicas podem ter umas funcionalidades próprias, como os caminhões carregar e descarregar, ou os ônibus aceitar_passageiros ou vender_passagem.

O normal em sistemas de herança é que as classes que herdam de outras incluam novas características e funcionalidades, à parte dos atributos e métodos herdados. Porém, isto não é imprescindível, de modo que podem se criar objetos que herdem de outros e não incluam nada novo.

**Sintaxe de herança em PHP 5**

A programação orientada a objetos nos oferece uma série de mecanismos para definir este tipo de estruturas, de modo que possam se criar hierarquias de objetos que herdam uns de outros. Veremos agora como definir estas estruturas de herança em PHP 5. Para isso, continuando com nosso exemplo da locadora de vídeo, vamos criar os distintos tipos de elementos que são oferecidos no aluguel.

Como todo mundo conhece, as locadoras de vídeos oferecem distintos tipos de elementos para alugar, como podem ser os filmes (fitas de vídeo ou DVD) e os jogos. Cada elemento tem umas características próprias e algumas comuns. Chamamos "suporte" à classe geral, que inclui as características comuns para todos os tipos de elementos em aluguel. Logo, criamos três tipos de suportes diferentes, que herdam da classe suporte, mas que incluem algumas características e funcionalidades novas. Estes tipos de suporte serão "fita_video", "dvd" e "jogo".

O esquema de herança que vamos realizar neste exemplo pode ser visto na seguinte imagem.


Começamos pela classe suporte. Seu código será o seguinte:

```
class suporte{
   public $titulo;
   protected $numero;
   private $preco;

   function __construct($tit,$num,$preco){
      $this->titulo = $tit;
      $this->numero = $num;
      $this->preco = $precio;
   }

   public function dar_preco_sem_imposto{
      return $this->preco;
   }

   public function dar_preco_com_imposto{
      return $this->preco * 1.16;
   }
   public function dar_numero_identificacao{
      return $this->numero;
   }
```

```
    public function imprime_caracteristicas{
        echo $this->titulo;
        echo "<br>" . $this->preco . " (imposto nao incluido)";
    }
}
```

Os atributos que definimos são, título, número (um identificador do suporte) e preço. Aplicamos a cada um, um modificador de acesso distinto, para poder praticar os diferentes tipos de acesso.

Definimos um constructor, que recebe os valores para a iniciação do objeto. Um método dar_preco_sim_imposto, que devolve o preço do suporte, sem aplicar o imposto. Outro método dar_preco_com_imposto, que devolve o preço uma vez aplicado por exemplo, 16% de imposto. O método dar_numero_identificacao, que devolve o número de identificador e imprime_caracteristicas, que mostra na página as características deste suporte.

*Nota: Como se foi definido como private o atributo preço, este atributo só poderá ser acessado dentro do código da classe, ou seja, na própria definição do objeto. Se quisermos acessar ao preço de fora da classe (algo muito normal se temos em conta que vamos necessitar o preço de um suporte de outras partes da aplicação) será necessário criar um método que nos devolva o valor do preço. Este método devería se definir como public, para que possa ser acessado de qualquer lugar que se necessite.*

Em todos os métodos usa-se a variável $this. Esta variável não é mais que uma referência ao objeto sobre o qual se está executando o método. Em programação orientada a objetos, para executar qualquer destes métodos, primeiro temos que ter criado um objeto a partir de uma classe. Logo, poderemos chamar os métodos de um objeto. Isto se faz com $meu_objeto->metodo_a_chamar. Dentro de método, quando se utiliza a variável $this, se está fazendo referência ao objeto sobre o qual se chamou ao método, neste caso, o objeto $meu_objeto. Com $this->titulo estamos fazendo referência ao atributo "titulo" que tem o objeto $meu_objeto.

Se quisermos provar a classe suporte, para confirmar que se executa corretamente e que oferece resultados coerentes, podemos utilizar um código como o seguinte.

```
$suporte1 = new suporte("Os Intocáveis",22,3);
echo "<b>" . $suporte1->titulo . "</b>";
echo "<br>Preco: " . $suporte1->dar_preco_sem_imposto . "euros";
echo "<br>Preco imposto incluído: " . $suporte1->dar_preco_com_imposto . "
euros";
```

Neste caso criamos uma instância da classe suporte, em um objeto que chamamos $suporte1. Logo, imprimimos seu atributo titulo (como o título foi definido como public, podemos acessa-lo desde fora do código da classe.
Logo, chamam-se aos métodos dar_preco_sem_imposto e dar_preco_com_imposto para o objeto criado.

Daría isto como resultado:

Os Intocáveis
Preço: 3 euros
Preço imposto incluído: 3,48 euros

Continuando com nosso exemplo da locadora de vídeo, vamos construir uma classe para os suportes de tipo fita de vídeo. As fitas de vídeo têm um atributo novo que é a duração da fita. Não têm nenhuma classe nova, embora devemos aprender a sobrescrever métodos criados para o suporte, visto que agora têm que fazer tarefas mais específicas.

**Sobrescrever métodos**

Antes de mostrar o código da classe fita_video, vamos falar sobre a sobrescritura ou substituição de métodos, que é um mecanismo pelo qual uma classe que herda pode redefinir os métodos que está herdando.

Pensemos em uma cafeteira. Sabemos que existem muitos tipos de cafeteiras e todas fazem café, mas o mecanismo para fazer o café é distinto dependendo do tipo de cafeteira. Existem cafeteiras express,

cafeteiras por gotas e até se pode fazer café com uma meia! Nossa cafeteira "pai" (da que vai herdar todas as cafeteiras) pode ter definido um método fazer_cafe(), mas não necessariamente todas as cafeteiras que possam herdar desta fazem o café seguindo o mesmo processo.

Então podemos definir um método para fazer café padrão, que teria a classe cafeteira. Mas ao definir as classes cafeteira_express e cafeteira_gotas, deveríamos sobrescrever o método fazer_cafe() para que se ajuste ao procedimento próprio destas.

A sobrescritura de métodos é algo bastante comum em mecanismos de herança, visto que os métodos que foram criados para uma classe "pai" não têm porquê ser os mesmos que os definidos nas classes que herdam.

Veremos como sobrescrever ou substituir métodos em um exemplo de herança, seguindo nosso exemplo da locadora de vídeo.

**Sintaxe para herdar em PHP 5**

Havíamos comentado que a locadora de vídeo dispõe de diferentes elementos para alugar, como fitas de vídeo, DVD ou jogos. Havíamos criado uma classe suporte, que vamos herdar em cada um dos elementos disponíveis para alugar. Vamos começar pela classe fita_video, cujo código será o seguinte:

```
class fita_video extends suporte{
   private $duracao;

   function __construct($tit,$num,$preco,$duracao){
      parent::__construct($tit,$num,$preco);
      $this->duracao = $duracao;
   }

   public function imprime_caracteristicas(){
      echo "Filme em VHS:<br>";
      parent::imprime_caracteristicas();
      echo "<br>Duracao: " . $this->duracao;
   }
}
```

Com a primeira linha class fita_video extends suporte estamos indicando que está sendo definida a classe fita_video e que vai herdar da classe suporte.

*Nota: Como está sendo herdado de uma classe, PHP tem que conhecer o código da classe "pai", neste caso, a classe suporte. De modo que o código da classe suporte deve estar incluído dentro do arquivo da classe fita_video. Podemos colocar os dois códigos no mesmo diretório, ou se estão em diretórios independentes, devemos incluir o código da classe suporte com a instrução include ou require de PHP.*

Na classe fita_video definimos um novo atributo chamado $duracao, que armazena o tempo que dura o filme.

Embora a classe sobre a qual herdamos (a classe suporte) tinha definido um construtor, a fita de vídeo deve iniciar a nova propriedade $duracao, que é específica das fitas de vídeo. Por isso, vamos sobrescrever ou substituir o método construtor, o que se faz simplesmente voltando a escrever o método. A graça aqui consiste em que o sistema pode basear a nova declaração do construtor na declaração que existia para a classe da que herda.

Ou seja, já havia sido definido um construtor para a classe suporte, que iniciava os atributos desta classe. Agora, para a classe fita_video, há que iniciar os atributos definidos na classe suporte, mais o atributo $duracao, que é o próprio de fita_video.

O código do construtor é o seguinte:

```
function __construct($tit,$num,$preco,$duracao){
   parent::__construct($tit,$num,$preco);
   $this->duracao = $duracao;
}
```

Na primeira linha do construtor é chamado ao construtor criado para a classe "suporte". Para isso, utilizamos parent:: e logo o nome do método da classe pai ao que se quer chamar, neste caso __construtor(). Ao construtor da classe pai lhe enviamos as variáveis que serão iniciadas com a classe pai.

Na segunda linha do construtor inicia-se o atributo duração, com o que tivermos recebido por parâmetro.

Acontece o mesmo com o método imprime_caracteristicas(), que agora deve mostrar também o novo atributo, próprio da classe fita_video. Como se pode observar no código da função, faz-se uso também de parent::imprime_caracteristicas() para utilizar o método definido na classe pai.

Se quisermos provar a classe fita_video, poderíamos utilizar um código como este:

```
$minhafita = new fita_video("Os Outros", 22, 4.5, "115 minutos");
echo "<b>" . $minhafita->titulo . "</b>";
echo "<br>Preco: " . $minhafita->dar_preco_sem_imposto() . " euros";
echo "<br>Preco IVA incluido: " . $minhafita->dar_preco_com_iva() . "
euros";
```

O que nos devolveria o seguinte:

Os Outros
Preço: 4.5 euros
Preço IVA incluído: 5.22 euros
Filme em VHS:
Os Outros
4.5 (IVA não incluído)
Duração: 115 minutos